A MODEL THEORETIC APPROACH TO THE THEORY OF ABSTRACT DATA TYPES AND DATA STRUCTURES

by

R.L. de Carvalho, T.S.E. Maibaum
T.H.C. Pequeno, A.A. Pereda, P.A.S. Veloso

Research Report CS-80-22

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

April, 1980

A MODEL THEORETIC APPROACH TO THE THEORY OF ABSTRACT DATA TYPES AND DATA STRUCTURES

R.L. de Carvalho*, T.S.E. Maibaum†, T.H.C. Pequeno*,
A.A. Pereda*, P.A.S. Veloso*

† Department of Computer Science
University of Waterloo
Waterloo, Ontario CANADA
N2L 3G1

* Departamento de Informatica
Pontificia Universidade Catolica
Rua Marques de Sao Vicente, 225
Gavea - CEP22453
Rio de Janeiro, RJ, Brazil

Abstract

We propose in this report a theory of abstract data types based on the first order predicate calculus. That is, we are in some sense reverting to first principles as espoused by Hoare. However, we do not want to abandon the important contributions of the algebraic theory. Probably the most important among these is the concept of initiality. The main thrust of this work is to present a theory of initiality for logic suitable for use in a theory of abstract data types.

The initiality result for logic is based on the concepts of minimum model (predicates have the "least" possible meaning over a given domain) and reachable model (all objects have a name). We motivate these concepts and show how they can be used to model not only data types, but also the internal structure of individual data values (objects). We illustrate the theory with a number of examples of data structures and types and indicate how a theory of implementations could be developed to complete this approach.

1. Introduction

For the past few years, the main effort in the specification of programs seems to have turned away from the study of the algorithmic nature of computer programs and towards a systematic treatment of the specification problem for the data manipulated by such programs. The lesson that was learned in the development of structured programming (essentially a tool for the specification of the algorithmic or control structure of a program) was that it was impossible to obtain structuredness in a meaningful way without also structuring the data being manipulated by the program. That is, it was hard to develop complex programs when these programs were thought of as manipulating bits and bytes (or even integers, reals, arrays, etc.).

An important concept in the discussion of data is that of "type". A type in most programming languages is seen as a collection of values (represented in a particular way). Even PASCAL views a type as a collection of values (although the representation is no longer explicit). This view began to be questioned when some interest arose in so-called user defined types, i.e. data types which are defined by a programmer for use in his program. These types were specified not in terms of the primitive data types available in some language, but in a higher level language. The programmer defined operations and tests were to be implemented later in terms of a representation for the values of the type (in terms of primitive types) and primitive operations available in the language. Thus the view grew that a data type included not only a collection of values, but also a set of operations and tests defined for the type. An early example of this is the "class" construct in SIMULA. A class is a collection of operations and tests (defined in terms of procedures)

defined for a particular representation of the data values. The fact that the specification of the class is done purely in terms of a representation is a severe drawback in trying to characterize a type since many of the properties are representation dependent. The first place where a real attempt is made to characterize data types independently of their representation is in Hoare 72-1, 72-2, 75. The concept of representation independence is called abstractness in Liskov 74, Liskov 77, Goguen 75, Goguen 78, to emphasize the fact that only the common important characteristics among different possible representations are considered. The method by which this abstractness is achieved in the specification of the type is by referring only to the names of the operations, tests and constants of the type when defining the properties of the type. Thus, no reference is made to the internal structure (i.e. representation or implementation details) of operations, tests and constants. In Hoare 72-1, 72-2, the properties of the type were specified through the language of an applied first order predicate calculus. Similar attempts were made in Baucilhon 76. The shortcomings of this approach were that the axiomatisation of the type did not define the type uniquely. Which model of the axioms was to be considered a valid representation of the type? (Clearly some models might have properties that are not those intended in the specification.) No guidance is offered in Hoare 72-1, 72-2 or Liskov 74, 77 to set some criterion. (In the algebraic approach to be discussed next, this problem again arises in the work of Guttag, et.al. (Guttag 77, 78), since they abandon the criterion accepted in this approach).

It was realized soon after the appearance of Hoare 72-1, 72-2 that algebras (a special case of models, or structures, as we will call them from now on), were also adequate for describing data types. (This is

because the relations (of structures) could be thought of as boolean valued operations). This realization was made in Liskov 74, and came into full force in the work of Guttag 77 and the work of the group referred to as ADJ (Goguen 75, 78). In the work of the latter group, it was realized that algebra provided a tool for defining exactly which class of algebras could be thought of as representations for the type. The tool used is that of initiality. An algebra is initial in a class of algebras if it belongs to the class and if for each algebra in the class, there is a unique homomorphism from the initial algebra to the given algebra. Moreover, if algebras A and B are initial in the same class, they are isomorphic. If we can specify the class of algebras in which we are interested, then the data type we are trying to define is the initial algebra in the class. Any representation must be isomorphic as an algebra to this initial algebra (and is thus itself initial in the class).

The axioms defining classes (varieties) of algebras are sets of equations (defining equalities between classes of expressions). The initial algebra in an equationally defined class is the minimal algebra satisfying the axioms (in the sense that it has exactly those properties specified by the axioms). In fact the initial algebra satisfies only the axioms and so we have an exact characterization of the data type: the isomorphism class of the initial algebra. It cannot be emphasized enough that it is this concept of initiality which gives rise to the power of the method. Once the requirement of initiality is relaxed, many of the results and proof methods associated with the concept disappear. Hence, efforts such as that reported in Lockemann 79 to define data bases are doomed to failure, since these powerful results are inapplicable and no comparative tools are supplied.

The algebraic approach does have, however, its shortcomings. A simple one is the unnaturalness (in many cases) of considering predicates to be boolean functions. Secondly, equations are a very restrictive form of axiomatisation and many properties are not stated easily or are not at all statable in terms of equations. Thirdly, a serious technical flaw is apparent in the definition of implementation (representation) as given in Goguen 78. An implementation of a given data type consists of representations for the values (congruence classes of expressions) of the type in terms of some previously specified types and derived operations over these types to implement the operations. A derived operation is defined by an expression consisting of variables (representing arguments) and the operations of the types being used for the representation. Now consider the data type "sequence of elements" with test iselinseq = sequence x element → boolean which tests whether a given element occurs in a sequence. A natural implementation in terms of a type "linked list" is:

```
procedure     iselinseq (seq: sequence, el: element): boolean;
begin              iselinseq: = false;
                   current: = headof (seq);
          while ¬  endofseq (seq) and ¬ iselinseq do
                   if  current = el
                  then iselinseq: = true
                  else current: = next (seq)
end.
```

This natural implementation does not correspond, however, to any derived operation of the type "linked list". Thus implementations of operations cannot be done by recourse to iteration or recursion (or any implicit method of definition).

In this report we intend to start a program of research to try to develop a theory of data types which overcomes the above difficulties, but retains the advantages. Specifically in this report we develop a theory of initiality for first order logic. At least, we develop such a theory for the classes of structures (models) which are appropriate for describing data structures and data types. Thus we are going back to first principles, so to speak, and propose to add to Hoare's proposals a simple criterion for defining what exactly a given axiomatization is defining.

Finally, we note that algebra does not provide a good vehicle for studying data structures (i.e., values in a data type). In many cases, a value in a type can be thought of as having some internal structure. Being able to study this structure formally would be of great advantage. (Algebra ignores this internal structure of values because it concentrates on the properties of the operations).

Starting from the observation of some general properties of data, a model-theoretic approach via Herbrand universes is used for an epistemological analysis of data types and data structures. The basic idea is to rule out certain classically possible models, which turn out to be undesirable (from a Computer Scientist's point of view). Essential features of a data structure are a set of objects (nodes, storage positions, etc.), an accessibility relation connecting them, and a set of distinguished objects (entry points), besides possibly functions and other relations. An abstract data type consists basically of a set of objects (its elements or instances), functions transforming these objects and a set of distinguished objects (which corresponds to initialization), with possibly other functions and relations.

Our approach will treat uniformly both data types and data structures, as well as their implementations, the relevant distinctions being regarded as due to different levels of abstraction, rather than conceptual differences.

This property requires the intersection of a set of models of some axioms to be a model of the axioms. This property guarantees the existence of a unique minimal model. The initiality result then states that the minimal model (the intersection of all reachable models in the Herbrand universe) is initial in the class of all reachable models.

In section 5 we develop some examples illustrating our method and intimating the further development of the theory. This development will state an appropriate theory of implementation that allows the use of implicit definition. We then close with some remarks on the theory and discussion of future research.

We begin in the next section by outlining the definitions and results from logic on which the rest of the paper depends. In section 3, we develop ideas which allow us to define exactly what structures are allowed to be models of data type axioms. We develop two criteria: reachability and minimality. The former requires values to be constructible using the operations of the type. The latter defines the meaning of the predicates to be the minimal relations (in the set-theoretic sense) allowed by the axioms.

In section 4, we develop our theory of initiality for reachable structures. The initiality results are valid only for certain kinds of axioms. The axioms are required to have the so called "intersection property".

2. Mathematical Preliminaries

We use the notation and definitions of Enderton 72 and just recall some notation and basic definitions here. A(non-logical) language L is a triple $\langle C, F, P\rangle$ such that C is a set of constant symbols, F is a set of function symbols with associated arities, and P is a set of relation symbols with associated arities. A structure $\mathcal{D}$ (over L) is quadruple $\langle D, C, F, P\rangle$ where D is a non-empty set, C is a set of distinguished elements of D such that to $c \in C$ there corresponds $c^{\mathcal{D}} \in C$, F is a set of functions of D such that to $f \in F$ of arity n there corresponds a function $f^{\mathcal{D}}$ of arity n over D, and P is a set of relations over D such that to $r \in P$ of arity n there corresponds a relation $r^{\mathcal{D}}$ of arity n over D.

Note that if equality is in the language, it is a non-logical symbol and it is not constrained to be interpreted as identity.

A structure $\mathcal{D}$ is a model of a set Σ of sentences if $\mathcal{D} \models \sigma$ for all $\sigma \in \Sigma$ (i.e., each sentence in Σ is valid in $\mathcal{D}$). A formula α is a logical consequence of a set of formulas Γ in a language L, denoted $\Gamma \models \alpha$, iff for all structures $\mathcal{D}$ and for all valuations s (assigning values in D to variables), if $\mathcal{D} \models_s \Gamma$ then $\mathcal{D} \models_s \alpha$.

Let $\mathcal{D}_1 = \langle D_1, C_1, F_1, P_1\rangle$ and $\mathcal{D}_2 = \langle D_2, C_2, F_2, P_2\rangle$ be two structures for L. A map $h: D_1 \to D_2$ is a weak homomorphism, denoted $h: \mathcal{D}_1 \to \mathcal{D}_2$, iff:

(i) $h(c^{\mathcal{D}_1}) = c^{\mathcal{D}_2}$ for all $c \in C$;

(ii) $h(f^{\mathcal{D}_1}(a_1, \ldots, a_n)) = f^{\mathcal{D}_2}(h(a_1), \ldots, h(a_n))$ for each $f \in F$;

(iii) $\langle a_1, \ldots, a_n \rangle \in r^{\mathcal{D}_1}$ implies $\langle h(a_1), \ldots, h(a_n) \rangle \in r^{\mathcal{D}_2}$

for each $r \in P$.

Moreover, if the map h is such that

$$\langle a_1, \ldots, a_n \rangle \in r^{\mathcal{D}_1} \text{ iff } \langle h(a_1), \ldots, h(a_n) \rangle \in r^{\mathcal{D}_2}$$

for all $r \in P$, then H is called a strong homomorphism.

The Herbrand Universe of L is the set of variable free terms (expressions) built from C and F.

3. Reachable Structures and Minimum Models

Logic has proved to be an adequate tool for reasoning formally about classes of mathematical structures with some common properties. Its use is particularly convenient when the class under consideration is defined by a set of properties expressable within the system of logic we are using. Such is the case, for instance, for groups and first order logic. A group can be defined as a mathematical structure consisting of a distinguished element (0) and a binary function (+) such that 0 is a neutral element for +, every element has an inverse with respect to +, and + is associative. These properties can be expressed in the first order language <0, +> by the following sentences:

i) $\forall x \ (x + 0 = x)$;

ii) $\forall x \ \exists y \ (x + y = 0)$;

iii) $\forall x \ \forall y \ \forall z \ (x + (y+z)) = ((x+y) + z)$

A problem can appear, however, if we try to define a sub-class by adding some restriction and this restriction is not expressable in the

language. This would happen in the example above if we try to define the class of finite groups, since finiteness is not a first order property.

Sometimes we deal with the opposite situation. We are given a class of structures and we are asked to write down some properties to characterize it. Using first order predicate logic this is frequently impossible. What happens in such cases is that the first order properties of the class are not sufficient to define it. This is the case, for instance, with first order axiomatizations of natural numbers with zero and successor, $N = \langle N, 0, S \rangle$. A possible axiomatization (borrowed from Enderton 72) is:

i) $\forall x\ (S(x) \neq 0)$;

ii) $\forall x\ \forall y\ (S(x) = S(y) \rightarrow x=y)$;

iii) $\forall y\ (y \neq 0 \rightarrow \exists x\ (y = S(x)))$;

iv.1) $\forall x\ (S(x) \neq 0)$;

iv.2) $\forall x\ (S(S(x)) \neq x)$;

...

iv.n) $\forall x\ (S^n(x) \neq x)$.

where the superscript n indicates that the symbol S occurs at n consecutive places. N is a model for these axioms, but unfortunately any structure consisting of a copy of N plus an arbitrary number of copies of the integers is also a model for the axioms. One cannot argue that the inclusion of additional axioms could "fix" the given axiomatization by excluding these non-desirable models, because the given set of axioms can be proved to be complete (again see Enderton 72). In fact, this deficiency in the power of characterization is a limitation of the system of first order predicate logic as demonstrated by the following well known theorems: (see Chang 73, Enderton 72 for demonstrations.)

Löwenheim-Skolem Theorem:

a) Let Γ be a satisfiable set of formulas in a countable language. Then Γ is satisfiable in some countable structure.

b) Let Γ be a satisfiable set of formulas in a language of cardinality x. Then Γ is satisfiable in some structure of cardinality $\leq x$.

Löwenheim-Skolem -Tarski Theorem:

Let Γ be a set of formulas in a language of cardinality x, and assume that Γ is satisfiable in some infinite structure. Then for every cardinal $\lambda \geq x$, there is a structure of cardinality λ in which Γ is satisfiable. (Recall that the cardinality of L is $|C| + |F| + |R| + \omega$.)

In our approach, we are going to treat uniformly both data types and data structures as mathematical structures (in the sense defined in section 2). The situation here is the second one we mentioned above; i.e., we have a class of structures described in some way (or we just have in mind some intuitive description) and we would like to obtain some axioms to define it. So we will find here the same difficulties mentioned before. But now, since we are using first order predicate logic in a context slightly different from the one for which it was designed, we have a chance to succeed by using some general properties of data types and data structures to define the class of models in which we are interested.

First of all, note that our models here are at most denumerable. (In fact they are always finite if we take into account machine limitations.) So we can restrict our semantics to consider just denumerable models.

Another important issue about structures modelling data types and data structures is that they are "reachable". By reachable we mean, in

the case of data structures, that starting from the entry point of the structure we are able to reach any component of the structure by using the access functions. This connectivity is a well known and desirable property of data structures. In the case of data types, reachability means that a value can appear in the domain only if it can be obtained by successive applications of the operations to the initial (constant) values. This is quite reasonable too, because the only available way we have to obtain new values is by applying the given operations to the initial values. Another way to state the reachability property is by saying that a structure is reachable if every element of its domain is a value of a term without variables. We can view a term that denotes an element as a name for this element which is the value of the term. So a reachable structure is one in which every element of the domain is nameable by a term without variables. Formally this property can be defined as follows:

Let $\mathcal{D} = \langle D, C, F, R \rangle$ be a structure for a language L. Then we define the (functionally) accessible part of D to be the smallest subset $F(\mathcal{D})$ of D such that $C \subseteq F(\mathcal{D})$ and $F(\mathcal{D})$ is closed under the operations in F. That is, $F(\mathcal{D})$ is the smallest set such that:

i) $C \subseteq F(\mathcal{D})$;

ii) if $d_1, \ldots, d_n \in F(\mathcal{D})$, and $f \in F$ is of rank n, then $f(d_1, \ldots, d_n) \in F(\mathcal{D})$.

Notice that $F(\mathcal{D})$ is the domain of a substructure of $\mathcal{D}$ for it consists of those elements of D that are denoted by variable-free terms of the language L. Note also that this substructure is included in any other substructure of $\mathcal{D}$; i.e. it is the minimum substructure of $\mathcal{D}$. A structure $\mathcal{D}$ is reachable if $D = F(\mathcal{D})$.

Herbrand interpretations are examples of reachable structures. Reachable structures have some nice properties.

Lemma 3.1

Let $\mathcal{D}$ be a reachable structure for a language $L = \langle C, F, P \rangle$ and S be a structure for the same language L. There exists at most one homomorphism of $\mathcal{D}$ into S, which will be onto iff S is reachable. If this is the case, then $\mathcal{D}$ and S are elementary equivalent (i.e. $\mathcal{D} \models \alpha$ iff $S \models \sigma$ for any first order sentence σ of L).

Proof

If φ is a homomorphism from $\mathcal{D}$ to S, we have that for $c \in C$

$$\varphi(c^{\mathcal{D}}) = c^{S}$$

For all $f \in F$ of arity n

$$\varphi(f^{\mathcal{D}}(d_1, \ldots, d_n)) = f^{S}(\varphi(d_1), \ldots, \varphi(d_n)).$$

For all $p \in P$ of arity n, if

$$\langle d_1, \ldots, d_n \rangle \in p^{\mathcal{D}} \text{ then } \langle \varphi(d_1), \ldots, \varphi(d_n) \rangle \in p^{S}.$$

Let us suppose that there exists another homomorphism φ' from $\mathcal{D}$ to S; then for $c \in C$

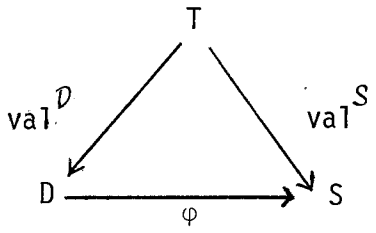
$$\varphi'(c^{\mathcal{D}}) = c = \varphi(c^{\mathcal{D}})$$

Let us suppose that $\varphi'(d_i) = \varphi(d_i)$ for $d_i \in D$, $1 \leq i \leq n$. Then for all $f \in F$ of arity n, we have that $f^{\mathcal{D}}(d_1, \ldots, d_n) \in D$, because $\mathcal{D}$ is reachable. Then

$$\begin{aligned}\varphi'(f^{\mathcal{D}}(d_1, \ldots, d_n)) &= f^{S}(\varphi'(d_1), \ldots, \varphi'(d_n)) \\ &= f^{S}(\varphi(d_1), \ldots, \varphi(d_n)) \\ &= \varphi(f^{\mathcal{D}}(d_1, \ldots, d_n))\end{aligned}$$

So, if φ is a homomorphism from $\mathcal{D}$ to S then φ is unique.

φ is onto iff S is reachable. This follows from the following commutative diagram:



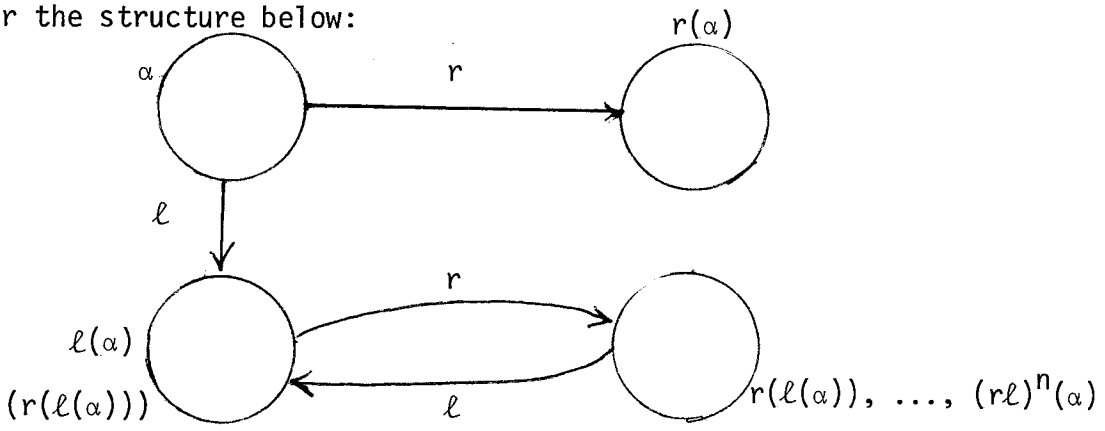
where T is the set of variable-free terms of L and val$^{\mathcal{D}}$, val^{S} are the valuation functions from terms to $\mathcal{D}$ and S, respectively.

Corollary

If $\mathcal{D}$ is a reachable structure and $\equiv$ is a congruence on $\mathcal{D}$ then $\mathcal{D}/\equiv$ is also reachable.

These properties of reachable structures will play an important role in the next section. Note now, however, that if we restrict our semantics so that we allow just reachable models, the natural numbers can be characterized exactly, for the reachable models that the axioms have are all isomorphic to N.

But the restriction to reachable structures is not yet enough. Consider the structure below:



in which α is the entry point and ℓ and r are the access functions for this data structure. We want to define the accessibility relation

Ac for this structure as the reflexive transitive closure of the relation S determined by the union of the graphs of r and ℓ. The sentence below is a quite natural way to do this.

$$\forall x \forall y \ (\underline{Ac}\ (x,y) \Leftrightarrow (x=\alpha \wedge y=\alpha) \vee S(x,y) \vee \exists z\ (\underline{Ac}\ (x,z) \wedge S(z,y)))$$

The above intends to give a recursive definition for Ac. Now we have again that the structure with

$$\underline{Ac}_0 = \{<\alpha,\alpha>, <\alpha, r(\alpha)>, <\alpha, \ell(\alpha)>, <\ell(\alpha), r\ell(\alpha)> <r\ell(\alpha), \ell(\alpha)>, \ldots, <\ell(\alpha), \ell(\alpha)>, \ldots <r\ell(\alpha), r\ell(\alpha)> \ldots\} .$$

is an interpretation for Ac - that is, the intended interpretation is a model for the given axiom. But so is a structure with, say

$$\underline{Ac}_1 = \underline{Ac}_0 \cup \{<r(\alpha), r\ell(\alpha)>, <r(\alpha), \ell(\alpha)>\}.$$

In fact, we can prove that $\underline{Ac}_0$ is the minimum relation that satisfies the axiom. In other words, a structure with $\underline{Ac}_0$ is a minimum model for the axioms.

More precisely the minimum model (if it exists) of a set of sentences Γ in a domain D is the model of Γ in D whose relations are included one by one in the corresponding relations of any model of Γ in D.

In the next section we will show how to combine the basic results of the previous section with the observations of the present section to build a model theory that will allow us to use first order predicate logic to characterize data types and data structures.

4. Limited Model Semantics

Assume we wrote down some axioms intended to define a certain abstract structure. This means that we have characterized just those properties we considered essential, dismissing those due to the particular characteristics of representations. Now we reduce our characterization problem to the following: how do we associate (uniquely) a special structure to these axioms such that any structure that we recognize as a realization of the abstract structure is isomorphic to that special one and vice-versa. In other words we would like to use the axioms to construct a structure such that the class of intended models for the axioms will be the isomorphism class of that structure. By the considerations of the last section, we agree that such a structure must be a minimum model, but in what domain? and with what definitions of the functions? Good candidates for answering these questions are Herbrand interpretations, for they include all the reachable structures for the language up to a quotient as indicated by the following proposition:

Proposition 4.1

Let $\mathcal{D}$ be a reachable structure for $L = \langle C, F, P \rangle$. There exists a unique Herbrand interpretation $H(\mathcal{D})$ such that $\mathcal{D} \cong H(\mathcal{D})/K(\varphi)$, where $K(\varphi)$ is the Kernel of the unique homomorphism φ of $H(\mathcal{D})$ onto $\mathcal{D}$.

Proof:

Let φ be the following function:

i) For all $c \in C$

$$\varphi(c) = c^{\mathcal{D}};$$

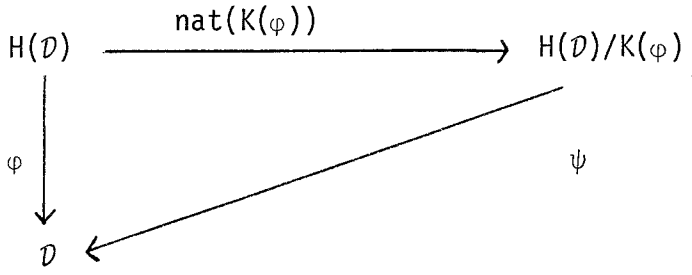
ii) For all $f \in F$

$$\varphi(f(h_1, \ldots, h_n)) = f^{\mathcal{D}}(\varphi(h_1), \ldots, \varphi(h_n)), \text{ for}$$

$$h_1, \ldots, h_n \in H(L).$$

Since the Herbrand interpretation and $\mathcal{D}$ are reachable, φ is unique and onto. We use φ to define the relations of $H(\mathcal{D})$. For all $r \in P$ and for all $h_1, \ldots, h_n \in H(L)$, $\langle h_1, \ldots, h_n \rangle \in r^{H(\mathcal{D})}$ iff $\langle \varphi(h_1), \ldots, \varphi(h_n) \rangle \in r^{\mathcal{D}}$. Notice that this makes φ a strong homomorphism.

It now remains to prove that the following diagram commutes and that ψ is a homomorphism which is one to one and onto (i.e. an isomorphism):



i) the diagram commutes:

ψ is defined by

$$\psi([h]) = \varphi(h)$$

In order to show that ψ is well defined we must have that:

$$\text{if } [h_1] = [h_2] \text{ then } \varphi(h_1) = \varphi(h_2).$$

This is assured by the fact that $K(\varphi)$ is the kernel of φ .

ii) ψ is a homomorphism:

For all $c \in C$,

$$\psi(c^{H(\mathcal{D})/K(\varphi)}) = \psi([c^{H(\mathcal{D})}])$$

$$= \psi(c^{H(\mathcal{D})})$$

$$= c^{\mathcal{D}} ;$$

For all $f \in F$,

$$\psi\,(f^{H(\mathcal{D})/K(\varphi)}([h_1], \ldots, [h_n])) =$$

$$= \psi([f^{H(\mathcal{D})}(h_1, \ldots, h_n)])$$

$$= \varphi(f^{H(\mathcal{D})}(h_1, \ldots, h_n))$$

$$= f^{\mathcal{D}}(\varphi(h_1), \ldots, \varphi(h_n))$$

$$= f^{\mathcal{D}}(\psi([h_1]), \ldots, \psi([h_n])).$$

For all $r \in P$,

$$<[h_1], \ldots, [h_n]> \in r^{H(\mathcal{D})/K(\varphi)} \quad \text{iff} \quad <h_1, \ldots, h_n> \in r^{H(\mathcal{D})}$$

$$\text{iff} \quad <\varphi(h_1), \ldots, \varphi(h_n)> \in r^{\mathcal{D}}$$

$$\text{iff} \quad <\psi([h_1]), \ldots, \psi([h_n])> \in r^{\mathcal{D}}$$

iii) φ is one to one:

Let h_1, h_2 be such that

$$\psi([h_1]) = \psi([h_2]) .$$

Then $\varphi(h_1) = \varphi(h_2)$

and $h_1 K(\varphi) h_2$ or $[h_1] = [h_2]$.

Finally, since $\mathcal{D}$ is reachable, ψ is onto.

The structure $H(\mathcal{D})$ is called the Herbrand interpretation induced by $\mathcal{D}$.

In what follows, we will show that the minimum Herbrand interpretation is (almost) that special one for which we are searching. We will do it by using the concept of initiality. We say that a structure S is initial in a class of structures if it belongs to the class and if for every structure $\mathcal{D}$ of the class there is a unique homomorphism from S to $\mathcal{D}$. A reason for using initiality is that it has the nice property of character-

izing things up to isomorphism, as stated below:

Proposition 4.2:

If two (reachable) structures A and B are initial in the same class then they are isomorphic.

Proof:

Let φ_1 be the unique homomorphism from A to B and φ_2 the unique homomorphism from B to A. We will show that φ_1 is actually an isomorphism from A to B.

Consider the following commutative diagrams

where 1_A and 1_B are the identity homomorphisms on A and B, respectively. By initiality of A, we have that $\varphi_2 \circ \varphi_1 = 1_A$ (since $1_A : A \to A$ must be unique). Similarly, $\varphi_1 \circ \varphi_2 = 1_B$. But then φ_1 (and φ_2) must be an isomorphism.

Let us get back to the central line of our development by giving a definition for what we call the "natural structure". Let Γ be a set of sentences in a language $L = \langle C, F, P \rangle$. The natural structure for Γ, $H(\Gamma)$, is the Herbrand interpretation for L in which the relations are defined as follows:

For all $r \in R$, for all $h_1, \ldots, h_n \in H(L)$, we have $\langle h_1, \ldots, h_n \rangle \in r^{H(\Gamma)}$ iff $\mathcal{D} \models r(h_1, \ldots, h_n)$ for all reachable models $\mathcal{D}$ of Γ.

Intuitively, this definition says that the relations in $H(\Gamma)$ are the minimum allowed by the sentences of Γ. We will show that in fact when $H(\Gamma) \models \Gamma$ then $H(\Gamma)$ is the minimum Herbrand model of Γ. In order to do that we must give a formal definition for the minimum Herbrand model. Let $\{\mathcal{D}_j | j \in J\}$ be a non-empty family of structures for L with the same domain and the same functions. We say that $\mathcal{D}_i$ is included in $\mathcal{D}_j$, $\mathcal{D}_i \subseteq \mathcal{D}_j$, iff for all $r \in P, r^{\mathcal{D}_i} \subseteq r^{\mathcal{D}_j}$. We define the intersection $\bigcap_{j \in J} \mathcal{D}_j$ as the structure $\mathcal{D}$ with the same domain and functions as all $\mathcal{D}_j$ and such that for all $r \in P$, $r^{\mathcal{D}} = \bigcap_{j \in J} r^{\mathcal{D}_j}$. We denote by $\mathrm{Mod}_H(\Gamma)$ the set of all Herbrand models of Γ. We call $\cap \mathrm{Mod}_H(\Gamma)$ the minimum Herbrand structure of Γ.

Proposition 4.3:

$$H(\Gamma) = \cap \mathrm{Mod}_H(\Gamma).$$

Proof:

i) $H(\Gamma) \subseteq \cap \mathrm{Mod}_H(\Gamma)$:

Let $r \in P$ and $h_1, \ldots, h_n \in H(L)$ such that $\langle h_1, \ldots, h_n \rangle \in r^{H(\Gamma)}$. By the definition of $H(\Gamma)$ and for all $\mathcal{D}$ such that $\mathcal{D} \models \Gamma$, we have that $\mathcal{D} \models r(h_1, \ldots, h_n)$;

$$\text{i.e. } \langle h_1^{\mathcal{D}}, \ldots, h_n^{\mathcal{D}} \rangle \in r^{\mathcal{D}}.$$

If $\mathcal{D}$ is an Herbrand interpretation, then $h_i^{\mathcal{D}} = h_i$, so we can write

$$\langle h_1, \ldots, h_n \rangle \in r^{\mathcal{D}}.$$

Then

$$\langle h_1, \ldots, h_n \rangle \in r^{H(\Gamma)} \Rightarrow \langle h_1, \ldots, h_n \rangle \in r^{\cap \mathrm{Mod}_H(\Gamma)}$$

For what follows, the property of $\cap \mathrm{Mod}_H(\Gamma)$ being a model of Γ is a fundamental requirement. We call this property the intersection property

for Herbrand interpretations. Unfortunately, this property is not always satisfied. Suppose Γ is $p(a) \vee q(a)$. A structure $\mathcal{D}$, with $p^{\mathcal{D}_1} = \{a\}$ and $q^{\mathcal{D}_1} = \phi$ is a model of Γ. So is $\mathcal{D}_2$ with $p^{\mathcal{D}_2} = \phi$ and $q^{\mathcal{D}_2} = \{a\}$. The structure $\mathcal{D} = \mathcal{D}_1 \cap \mathcal{D}_2$ has $p^{\mathcal{D}} = \phi$ and $q^{\mathcal{D}} = \phi$ and obviously is not a model of Γ . The basic question here is: what kinds of sets of sentences have the intersection property for Herbrand interpretations? We do not have a complete answer to this question. However, in Van Emden 76 it is shown that if Γ is equivalent to a set of Horn sentences, then it has this property. A Horn sentence is a clause containing at most one positive literal. Is this condition necessary? The answer is no, as will be shown later in the examples. So to be a set of Horn sentences is just a sufficient condition and the problem of the syntactic characterization of sentences that have the intersection property for Herbrand interpretations is still open.

From now on, we assume that we are working with sentences that have the intersection property for Herbrand interpretations.

Lemma 4.1

$H(\Gamma)$ is initial in the class of all the reachable models of Γ .

Proof:

Let $\mathcal{D}$ be a reachable model of Γ and $H(\mathcal{D})$ the Herbrand interpretation induced by $\mathcal{D}$. We know there is a unique (strong) homomorphism from $H(\mathcal{D})$ to $\mathcal{D}$. Now let i be the identity function from $H(L)$ to $H(L)$. There is no doubt that i is a homomorphism from $H(\Gamma)$ to $H(\mathcal{D})$ with respect to the constants and functions. To see that i is a (weak) homomorphism with respect to the relations, recall that $H(\Gamma) \subseteq H(\mathcal{D})$, so every relation of $H(\Gamma)$ is included in the corresponding one of $H(\mathcal{D})$. So $i \circ \varphi$ is a (weak) homomorphism from $H(\Gamma)$ to $\mathcal{D}$ and, by Lemma 3.1, it is unique and onto.

Although $H(\Gamma)$ is initial in the class of reachable models of Γ, it is not yet the structure we are looking for because in general it is not isomorphic to the structure we are trying to specify. The reason is the following: in $H(\Gamma)$ the only equality is the purely syntactic one (identity): i.e., two elements of $H(L)$ are equal in $H(\Gamma)$ iff they are the same term. In fact, we would like to consider two elements as equal if they have the same properties as far as the language we are using can determine. This means that they must be considered equal if they cannot be distinguished by the predicates we have in the language. A similar thing occurs often with implementations of data types and data structures. A given object can have different representations in the context of an implementation. For instance, if we are using arrays to implement sets, the same set can be represented by different arrays with the same elements, but in different order. Here again we would like to consider two objects as equal if they have the same properties, and that means that they appear in the same places in the same relations. More precisely, this equality can be defined as follows: Let $\mathcal{D} = \langle D, C, F, P\rangle$ be a structure. For d, $d' \in D$, $d \approx d'$ iff for all $r \in P$ we have (n being the arity of r) $\langle d_1, \ldots, d_{j-1}, d, d_{j+1}, \ldots, d_n\rangle \in r^{\mathcal{D}}$ iff $\langle d_1, \ldots, d_{j-1}, d', d_{j+1}, \ldots, d_n\rangle \in r$ for all $j = 1, \ldots, n$ and all $d_1, \ldots, d_{j-1}, d_{j+1}, \ldots d_n \in D$. Let $\equiv [\mathcal{D}]$ be the largest congruence on $\mathcal{D}$ included in $\approx$. The next lemma shows that this congruence identifies two elements exactly when they are indistinguishable.

Lemma 4.2

$d \equiv [\mathcal{D}]\ d'$ iff for every formula $\theta(x)$ of L, $\mathcal{D} \models \theta(x)[x/d]$, iff $\mathcal{D} \models \theta(x)[x/d']$. (Here $\alpha[y/e]$ denotes the substitution of e for all occurrences of y in α.)

Proof:

A straightforward proof by structural induction on formulas shows that

$$\mathcal{D} \models \theta(x_1, \ldots, x_n)[x_1/d_1, \ldots, x_n/d_n] \text{ iff } \mathcal{D} \models \theta(x_1, \ldots, x_n)[x_1/d_1', \ldots, x_n/d_n']$$

whenever $d_i \equiv [\mathcal{D}]\ d_i'$ for $i = 1, \ldots, n$,

(the basis being immediate from the definition.)

Thus, this congruence is a relativization to the language L of Leibniz's identity of indescernibles. $\equiv [\mathcal{D}]$ being a congruence gives rise to a strong natural projection onto the quotient structure $\mathcal{D}/\equiv [\mathcal{D}]$. The following lemma shows that such identifications are consistent.

Lemma 4.3

If $\mathcal{D}_2$ is a homomorphic image of the reachable structure $\mathcal{D}_1$ under φ, then there exists a unique homomorphism $\bar{\varphi}$ such that the following diagram commutes:

$$\begin{array}{ccc} \mathcal{D}_1 & \xrightarrow{\text{nat}(\equiv[\mathcal{D}_1])} & \mathcal{D}_1/\equiv[\mathcal{D}_1] \\ \varphi \downarrow & & \downarrow \bar{\varphi} \\ \mathcal{D}_2 & \xrightarrow[\text{nat}\ (\equiv[\mathcal{D}_2])]{} & \mathcal{D}_2/\equiv[\mathcal{D}_2] \end{array}$$

Proof:

We proceed by showing that $\equiv [\mathcal{D}_1]$ is included in the kernel of nat $(\equiv[\mathcal{D}_2]) \circ \varphi$ and the result then follows from the homomorphism theorem (see [Gratzer]) since all structures are reachable.

Let $\langle d, d' \rangle \in \equiv [\mathcal{D}_1]$. Then for all $r \in P$ and all $d_1, \ldots, d_{j-1}, d_{j+1}, \ldots, d_n \in D_1$ and all $j = 1, \ldots, n$, we have $\langle d_1, \ldots, d_{j-1}, d, d_{j+1}, \ldots, d_n \rangle \in r^{\mathcal{D}_1}$ iff $\langle d_1, \ldots, d_{j-1}, d_1', d_{j+1}, \ldots, d_n \rangle \in r^{\mathcal{D}_1}$. But then

$$\langle\varphi(d_1), \ldots, \varphi(d_{j-1}), \varphi(d), \varphi(d_{j+1}), \ldots, \varphi(d_n)\rangle \in r^{\mathcal{D}_2} \quad \text{iff}$$

$$\langle\varphi(d_1), \ldots, \varphi(d_{j-1}), \varphi(d'), \varphi(d_{j+1}), \ldots, \varphi(d_n)\rangle \in r^{\mathcal{D}_2}$$

since φ is a homomorphism. Thus we can conclude that $\langle\varphi(d), \varphi(d')\rangle \in \equiv [\mathcal{D}_2]$ since the above is equivalent to:

$$\langle d'_1, \ldots, d'_{j-1}, \varphi(d), d'_{j+1}, \ldots, d'_n\rangle \in r^{\mathcal{D}_2} \text{ iff } \langle d'_1, \ldots, d'_{j-1}, \varphi(d'), d'_{j+1}, \ldots, d'_n\rangle \in r^{\mathcal{D}_2}$$

for all $r \in P$, $d'_1, \ldots, d'_{j-1}, d'_{j+1}, \ldots, d'_n \in D_2$ and $j = 1, \ldots, n$. (Recall that φ is onto.) But then $\langle\varphi(d), \varphi(d')\rangle \in \equiv [\mathcal{D}_2]$. Thus if $\langle d , d'\rangle \in \equiv [\mathcal{D}_1]$, then $\langle d, d'\rangle \in \text{Ker}\,(\text{nat}\,(\equiv [\mathcal{D}_2]) \circ \varphi)$ and we are done. Note that $\bar{\varphi}$ will be strong iff φ is. We call $\equiv [\mathcal{D}]$ the natural equality on $\mathcal{D}$.

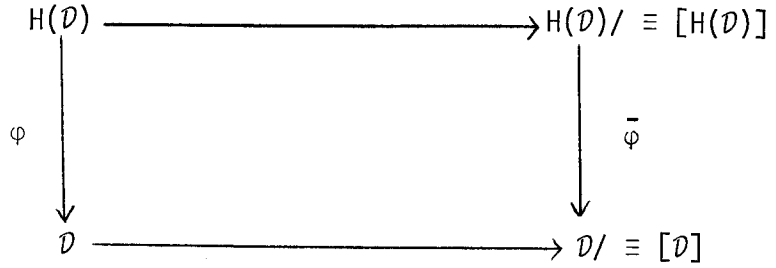
We are able now to state our main theorem:

Theorem 4.1

$H(\Gamma)/\equiv [H(\Gamma)]$ is initial in the class of quotients of reachable models of Γ by the corresponding natural equality.

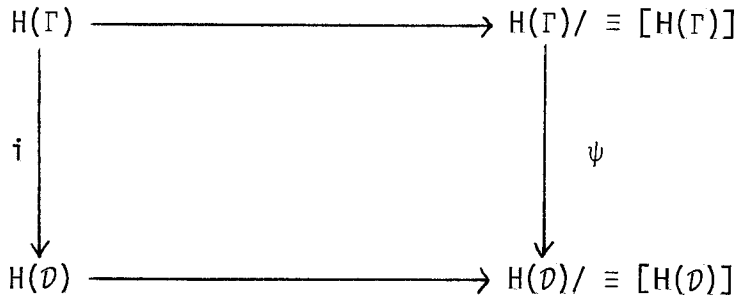
Proof:

Let $\mathcal{D}$ be a reachable model of Γ. Applying lemma 4.3 we have the following diagram:



where $\bar{\varphi}$ is a strong homomorphism since φ is (by lemma 3.1).

Applying lemma 4.3 again, we have the diagram



i is a weak homomorphism so ψ is. Combining the two diagrams we have that $\psi \circ \bar{\varphi}$ is a (weak) homomorphism from $H(\Gamma)/\equiv [H(\Gamma)]$ onto $\mathcal{D}/\equiv [\mathcal{D}]$. Since $H(\Gamma)/\equiv [H(\Gamma)]$ is reachable the homomorphism is unique by lemma 3.1

The above results give us the necessary justification to state our initial model semantics. Given a set of sentences Γ with the intersection property for Herbrand interpretations, we say that a structure $\mathcal{D}$ is an initial model of Γ or that $\mathcal{D}$ satisfies Γ initially iff $\mathcal{D}$ is isomorphic to $H(\Gamma)/\equiv [H(\Gamma)]$.

Note also that if $=$ is in L then by Lemma 4.2 $=^{\mathcal{D}} \subseteq \equiv [\mathcal{D}]$.

5. Examples

We proceed by illustrating the theory with some simple examples. Firstly, in section 5.1 we define a simple list data structure, find the natural structure induced by it and then find the quotient of this natural structure isomorphic to the original data structure. Secondly, in section 5.2 we specify two data types: binary trees and vectors (of unbounded size). We then implement the former by means of the latter in section 5.3 to illustrate the theory of implementations presently being

developed. We also make some comments on proofs of correctness for such implementations.

5.1 Data Structures

Consider the following "data structure" defined over

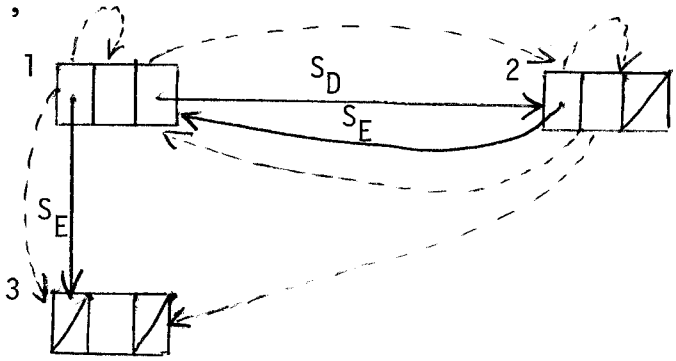
$L = \langle \alpha, \lambda, Ac, S_D, S_E \rangle$:

$I = \langle D^I, \alpha^I, \lambda^I, Ac^I, S_D{}^I, S_E{}^I \rangle$ with

$D^I = \{\lambda, 1, 2, 3\}$,

$\alpha^I = 1$,

$\lambda^I = \lambda$.



$Ac^I = \{\langle 1, 1\rangle, \langle 1,2\rangle, \langle 1,3\rangle, \langle 2,1\rangle, \langle 2,3\rangle, \langle 2,2\rangle, \langle 1,\lambda\rangle, \langle 2,\lambda\rangle, \langle 3,\lambda\rangle, \langle \lambda,\lambda\rangle\}$

$S_D{}^I(1) = 2$ $\qquad$ $S_E{}^I(1) = 3$

$S_D{}^I(2) = \lambda$ $\qquad$ $S_E{}^I(2) = 1$

$S_D{}^I(3) = \lambda$ $\qquad$ $S_E{}^I(3) = \lambda$

$S_D{}^I(\lambda) = \lambda$ $\qquad$ $S_E{}^I(\lambda) = \lambda$

Let the theory Γ be defined by:

1. $\forall x \forall y (Ac(x,y) \Leftrightarrow (x = \alpha \wedge y = \alpha) \vee$

$\exists z(Ac(x,z) \wedge S_D(z) = y) \vee$

$\exists z(Ac(x,z) \wedge S_E(z) = y))$

2. $S_D(\lambda) = \lambda \wedge S_E(\lambda) = \lambda$.

The Herbrand Universe for the theory Γ is:

$$H(\Gamma) = \{\alpha,\lambda,\ S_D(\alpha),\ S_E(\alpha),\ S_D(\lambda),\ \ldots,\ S_E(S_E(\alpha)),\ \ldots$$

$$S_D(S_D(\alpha))\ \ldots\}$$

The interpretation $H(I)$ is:

$$H(I) = \langle H(\Gamma),\ \alpha,\ \lambda,\ Ac^{H(I)},\ S_D,\ S_E\rangle$$

with

$$Ac^{H(I)} = \{\langle\alpha,\ \alpha\rangle,\ \langle\alpha,\ S_D(\alpha)\rangle,\ \langle S_D(\alpha),\ S_D(\alpha)\rangle,\ \langle\alpha,\ S_E(\alpha)\rangle,$$

$$\langle S_D(\alpha),\alpha\rangle,\ \langle S_D(\alpha),\ S_E(\alpha)\rangle,\ \langle S_E(S_D(\alpha)),\ S_E(S_D(\alpha))\rangle,$$

$$\langle\alpha,\ S_E(S_D(\alpha))\rangle,\ \langle S_E(S_D(\alpha)),\alpha\rangle,\ \langle\alpha,\ S_D(S_E(S_D(\alpha)))\rangle,\ ..\}\ .$$

with φ such that:

$$\varphi(\alpha) = 1 \qquad \varphi(S_E(S_D(\alpha))) = 1$$

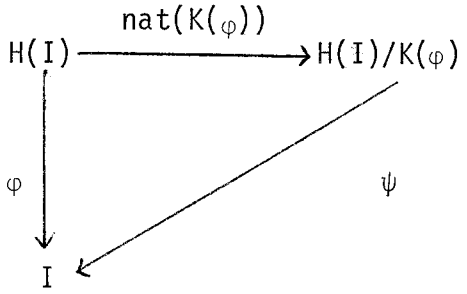
$$\varphi(S_D(\alpha)) = 2 \qquad \varphi(S_D(S_E(S_D(\alpha)))) = 2$$

$$\varphi(S_E(\alpha)) = 3 \qquad \varphi(S_E(S_E(S_D(\alpha)))) = 3$$

$$\varphi(\lambda) = \lambda$$

$$\varphi(S_E(S_E(\alpha))) = \lambda \qquad \varphi(S_D(S_D(\alpha))) = \lambda$$

we have



with

$$[\alpha] = \{\alpha, S_E(S_D(\alpha)), S_E(S_D(S_E(S_D(\alpha)))), \ldots\}$$

$$[S_D(\alpha)] = \{S_D(\alpha), S_D(S_E(S_D(\alpha))), S_D(S_E(S_D(S_E(S_D(\alpha))))), \ldots\}$$

$$[S_E(\alpha)] = \{S_E(\alpha), S_E(S_E(S_D(\alpha))), S_E(S_E(S_D(S_E(S_D(\alpha))))), \ldots\}$$

$$[\lambda] = \{\lambda, S_E(S_E(\alpha)), S_D(S_E(\alpha)), S_D(S_D(\alpha)), \ldots\}.$$

Also, $H(I)/K(\varphi)$ is such that

$$H(I)/K(\varphi) = \langle\{[\alpha], [S_D(\alpha)], [S_E(\alpha)], [\lambda]\}, [\alpha], [\lambda], Ac^{H(I)/K(\varphi)}, S_D, S_E\rangle$$

with

$$Ac^{H(I)/K(\varphi)} = \{\langle[\alpha], [\alpha]\rangle, \langle[\alpha], [S_D(\alpha)]\rangle, \langle[\alpha], [S_E(\alpha)]\rangle,$$

$$\langle[S_D(\alpha)], [\alpha]\rangle, \langle[S_D(\alpha)], [S_D(\alpha)]\rangle, \langle[\alpha], [\lambda]\rangle, \langle[S_D(\alpha)], [\lambda]\rangle,$$

$$\langle[S_E(\alpha)], [\lambda]\rangle, \langle[\lambda], [\lambda]\rangle\}$$

and

$$\psi([\alpha]) = 1$$

$$\psi([S_D(\alpha)]) = 2$$

$$\psi([S_E(\alpha)]) = 3$$

$$\psi([\lambda]) = \lambda$$

5.2 Binary Trees and Vectors

Note that in this and the following section we use the many sorted generalization of our previous work.

Consider the following language:

$$L_t = \langle 0, \lambda, error_n, succ, left, right, root, cons\rangle$$

where the sorting set is {t, n} and (using the notation of Guttag 77) the syntax is:

$error_n$: → n

0: → n

λ: → t

left : t → t

right: t → t

succ: n → n

root: t → n

cons: t x n x t → t.

The axioms are:

B1. left (cons(x, d, y)) = x

B2. right (cons(x, d, y)) = y

B3. root (cons(x, d, y)) = d

B4. left (λ) = λ

B5. right (λ) = λ

B6. root (λ) = $error_n$.

The data type vectors has language L_V = <Ω, 0, 1, succ, Λ, insert, access> where the sorting set is {vec, n, ind} and the syntax is:

Ω: → n

0: → n

1: → ind

Λ: → vec

succ: n → n

succ: ind → ind

insert: vec x ind x n → vec

access: vec x ind → n

The axioms are:

V1: $i \neq j \rightarrow$ insert (insert(v, i, d_1), j, d_2)
= insert (insert(v, j, d_2), i, d_1)

V2: insert (insert(v, i, d_1), i, d_2) = insert (v, i, d_2)

V3: insert (Λ, i, Ω) = Λ

V4: access (Λ, i) = Ω

V5: access (insert(v, i, d), i) = d

V6: $i \neq j \rightarrow$ access (insert(v, i, d), j) = access (v, j)

5.3 Implementing Binary Trees in Terms of Vectors

The axioms B1, ..., B6 and V1, ..., V6 define theories T_1 and T_2, respectively. To implement binary trees in terms of vectors means essentially that we must define a theory T_3 (the "representation theory"). T_3 is obtained basically by adding to the axioms defining T_2 definitions of the constants, operations and predicates of T_1. That T_3 has the properties of T_2 is then guaranteed by the fact that T_3 is an extension by definition of T_2. That T_3 has the properties of T_1 is a matter of making sure that the definitions are correct or adequate. One criterion for deciding whether the definitions are correct is to prove that the axioms of T_1, are theorems of T_3.

To motivate our particular representation, consider the following binary tree:

We can represent this tree by the vector:

1	2	3	Ω	5	6	7	Ω	Ω	8	Ω	Ω	9	10	11	Ω	Ω	....

This representation is obtained by using the usual heap representation for trees:

(i) The root is stored in position 1;

(ii) If a node stored in position i has a left (right) son, it is stored in position $2i$ $(2i+1)$. Otherwise, position $2i(2i+1)$ is Ω.

We now need to define a relativization predicate which tells us whether a given vector represents a tree or not:

$$D1: \quad P(v) \leftrightarrow \forall i[access(v, i) = \Omega \rightarrow access\ (v, 2i) = \Omega \wedge access(v, 2i + 1) = \Omega].$$

We now proceed to define each of the operations of the type binary tree:

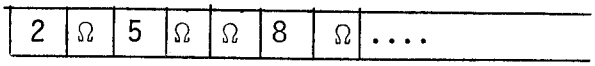
$$D2: \quad root\ (v) = d \leftrightarrow (access(v,1) = d \wedge v \neq \Lambda)\ v\ d = error_n.$$

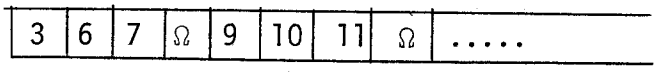
The operation of extracting left and right subtrees is somewhat complicated but can be defined as:

$$D3: \quad left(v) = u \leftrightarrow \forall i \forall j \forall k \forall \ell[(j = 2^{k+1} + \ell \wedge i \geq 2^k \wedge i < 2^{k+1} \wedge \ell = i-2^k) \rightarrow access\ (u, i) = access\ (v, j)]$$

$$D4: \quad right(v) = u \leftrightarrow \forall i \forall j \forall k \forall \ell[j = 2^{k+2} - 2^k + \ell \wedge i \geq 2^k \wedge i < 2^{k+1} \wedge \ell = i - 2^k) \rightarrow access(u, i) = access\ (v, j)].$$

For example the following are, in order, the representations of the left and right subtrees of our previous example:





Finally, to complete the definitions we have:

D5: $\text{cons}(u, d, v) = x \leftrightarrow \text{root}(x) = d \wedge \text{left}(x) = u \wedge \text{right}(x) = v$

D6: $\lambda = v \leftrightarrow \forall i[\text{access}(v, i) = \Omega]$.

The theory T_3 is then defined by the axioms V1, ..., V6 and the definitions D1, ..., D6. To prove that the representation is correct we must show two things: firstly, we must demonstrate that the domain defined by the relativization predicate P is adequate for describing all binary trees. To do this we must demonstrate the following for the set defined by P:

(a) it is non-empty;

(b) all constants of type binary tree are included;

(c) the set is closed under the operations defined on binary tree representations.

Formally, this is equivalent to showing that the following <u>closure axioms</u> are theorems of T_3.

F1: $\exists x P(x)$

F2: $P(\lambda)$

F3: $P(x) \rightarrow P(\text{left}(x))$

F4: $P(x) \rightarrow P(\text{right}(x))$

F5: $P(x) \wedge P(y) \rightarrow P(\text{cons}(x,d,y))$

We leave the proofs of F1, ..., F5 to the reader.(See Pequeno 79.).

Secondly, we must demonstrate that the axioms defining the theory T_1 are theorems of T_3. Again the proofs are left to the reader.

6. In Conclusion

6.1 Summary

We propose in this report an extension of the algebraic theory of abstract data types to the language of first order predicate logic. This extension is proposed to overcome a number of drawbacks of the algebraic method the most important of which are: limitations of equational formalisms for specifying properties of data types, the impossibility of treating values (data structures) as structured objects, the lack of a proper theory of implementation allowing the use of implicit definitions. The logical theory is based on the presence of two characteristics for (logical) structures which purport to implement data types or data structures: reachability (all objects have names) and the existence of minimum model (to guarantee the presence of an initiality property). We were then able to generalize the initiality results of the algebraic theory to logic in a simple, straightforward manner. The new theory was then illustrated by a number of examples defining both data structures and data types. The theory of implementation being developed for this logical treatment was then illustrated.

6.2 Suggestions for Further Research

A number of problems related to this work still need extensive study. Firstly, the characteristics of logical structures used for implementing or defining data structures/types need more detailed study. Secondly,

the theory of implementation, based on the theory of definitions, needs a more complete formalization (Pequeno 80). Thirdly, since the structures we use in this work are minimum models, we need to develop a special logic for them. This logic would try to give both semantic and syntactic characterizations of such structures. An important and related problem is the need for a more complete syntactic characterization of those sentences which admit minimum models. Fourthly, we need to study the comparative usefulness of the algebraic and logical approaches. The power of the two systems also needs more careful analysis since the inclusion of a boolean sort in algebraic abstract data types seems to make the equational logic for these more powerful than is obvious at first sight. Finally, we will have some suggestions (Pequeno 80, Veloso 80) for using such theories in the methodology of program development.

Bibliography

Baucilhon 76: Baucilhon, F. - Data Structures: Specification and Realization, thesis, University of Michigan, C.S., 1976.

Carvalho 77: de Carvalho, R.L., Furtado, A.L., Pereda, A.A. - A Relational Model Towards the Synthesis of Data Structures, M.C.C. 17/77, D.I./PUC/RJ., 1977.

Chang 73: Chang, C.C., Keisler, H.J. - Model Theory, North-Holland, 1973.

Clark 77: Clark, K.L., Tarnlund, S. - A First Order Theory of Data and Programs, Information Processing 77, IFIP, pp. 939-944, 1977.

Cohn 65: Cohn, P.M. - Universal Algebra, Harper and Row, 1965.

Cook 75: Cook, S.A., Oppen, D.C. - An Assertion Language for Data Structures. Proc. of 2nd ACM Symposium on Principles of Programming Languages, pp. 160-166, 1975.

Enderton 72: Enderton, H.B. - A Mathematical Introduction to Logic, Academic Press, 1972.

Fleck 78: Fleck, A.C. - Recent Developments in the Theory of Data Structures, Computer Languages, Vol. 3, pp. 37-52, 1978.

Goguen 75: Goguen, J.A., Thatcher, J.W., Wagner, E.G., Wright, J.B. Abstract Data Types as Initial Algebras and the Correctness of Data Representation, Conference on Computer Graphics, Pattern Recognition and Data Structures, pp. 89-93, 1975.

Goguen 78: Goguen, J.A., Thatcher, J.W., Wagner, E.G., - An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types, Current Trends in Programming Methodology, Vol. IV, Ed. R.T. Yeh, Prentice Hall, 1978.

Gratzer 68: Gratzer, G. - Universal Algebra, D. Van Nostrand, 1968.

Guttag 77: Guttag, J.V. - Abstract Data Types and the Development of Data Structures,CACM, Vol. 20, no.6, pp. 396-404, 1977.

Guttag 78: Guttag, J.V., Horowitz, E., Musser, D.R. - Abstract Data Types and Software Validation, CACM, Vol. 21, no.12, pp. 1048-1064, 1978.

Bibliography cont'd

Hoare 72-1: Hoare, C.A.R. - Notes on Data Structuring, Structured Programming by O.-J. Dahl, E.W. Dijkstra, C.A.R. Hoare, Academic Press, 1972.

Hoare 72-2: Hoare, C.A.R. - Proof of Correctness of Data Representations, Acta Informatica, Vol. 1, no.1, pp. 271-287, 1972.

Hoare 75: Hoare, C.A.R. - Recursive Data Structures, International Journal of Computer and Information Sciences, Vol. 4, no. 2, pp. 105-132, 1975.

Horowitz 76: Horowitz, E., Sahni, S. - Fundamentals of Data Structures, Computer Science Press, 1976.

Liskov 74: Liskov, B., Zilles, S. - Programming with Abstract Data Types, SIGPLAN Notices, Vol. 9, no. 4, pp. 72-82, 1974.

Liskov 77: Liskov, B., Zilles, S. - Specification Techniques for Data Abstractions, Current Trends in Programming Methodology, Vol. I, Ed. R.T. Yeh, Prentice-Hall, 1977.

Lockemann 79: Lockemann, P.C., Mayr, H.C., Weil, W.H. Wohlleber, W.M., Data Abstraction for Database Systems, ACM Transactions on Database Systems, Vol. 4, no.1, pp. 60-75, 1979.

Maibaum 79: Maibaum, T.S.E. - Non-termination Implicit Definitions and Abstract Data Types, University of Waterloo, Technical Report CS-79- 26 , 1979.

Maibaum 80: Maibaum, T.S.E., Levy, M.R. - Continuous Data Types, to appear in SIAM Journal of Computing.

Majster 77: Majster, M.E. - Extended Directed Graphs, A Formalism for Structural Data and Data Structures, Acta Informatica, Vol. 8, no. 1, pp. 37-60, 1977.

Morris 73: Morris, J.M. - Types are not Sets, Proc. of ACM Symposium on the Prinicples of Programming Languages, pp. 120-124, 1973.

Pequeno 79: Pequeno, T.H.C. - Uma Abordagem Logica Ao Problema de Representacao de Tiposde Dados, DI/PUC/RJ, 1979.

Pequeno 80: Pequeno, T.H.C. - Uma Formalizacao Logica do Desenvolvimento de Programas Usando Tipos de Dado Abstratos, thesis, DI/PUC/RJ, 1980.

Bibliography cont'd

Pereda 79: Pereda, A.A. - Metodos de Descricao de Tipos de Dados e Estruturas de Dados, thesis, DI/PUC/RJ, 1979.

Standish 73: Standish, T.A. - Data Structures an Axiomatic Approach BBN. Computer Science Division, 1973.

Thatcher 76: Thatcher, J.W., Wagner, E.G., Wright, J.B. - Specification of Abstract Data Types using Conditional Axioms, IBM, Research Report 6214, 1976.

Van Emden 76: Van Emden, M.H., Kowalski, R.M. - The Semantics of Predicate Logic as a Programming Language, Journal of ACM, Vol. 23, no. 4, pp. 733-742, 1976.

Veloso 79: Veloso, P.A.S., Pequeno, T.H.C. - Don't Write More Axioms than You have to: A Methodology for the Complete and Correct Specification of Abstract Data Types, with examples, M.C.C. 10/79, DI/PUC/RJ, 1979.

Veloso 78: Veloso, P.A.S., Pequeno, T.H.C. - Interpretations Between Many-sorted Theories, II Encontro Brasileiro de Logica, Campinas, Brasil, July, 1978.

Veloso 80: Veloso, P.A.S. - Divide-and-Conquer via Data Types, Proc. of Panel 80, Caracas, Venezuela, 1980.