View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by ScholarlyCommons@Penn

University of Pennsylvania
ScholarlyCommons

Technical Reports (CIS)

Department of Computer & Information Science

August 1991

Data Abstraction and General Recursion

Ramesh Subrahmanyam
University of Pennsylvania

Follow this and additional works at: https://repository.upenn.edu/cis_reports

Recommended Citation

Ramesh Subrahmanyam, "Data Abstraction and General Recursion", . August 1991.

University of Pennsylvania Department of Computer and Information Sciences Technical Report No. MS-CIS-91-58.

This paper is posted at ScholarlyCommons. https://repository.upenn.edu/cis_reports/322
For more information, please contact repository@pobox.upenn.edu.

Data Abstraction and General Recursion

Abstract

Existing approaches to semantics of algebraically specified data types such as Initial Algebra Semantics and Final Algebra Semantics do not take into account the possibility of general recursion and hence non-termination *in the ambient programming language*. Any technical development of this problem needs to be in the setting of domain theory. In this paper we present extensions of initial and final algebra semantics to algebras with an underlying domain structure. Four possibilities for specification methodologies arise: two each in the Initial and Final algebra paradigms. We demonstrate that the initial/final objects (as appropriate) exist in all four situations. The final part of the paper attempts to explicate the notion of abstractness of ADT's by defining a notion of operational semantics for ADT's, and then studying the relationship between the various algebraic-semantics proposed and the operational semantics.

Comments

University of Pennsylvania Department of Computer and Information Sciences Technical Report No. MS-CIS-91-58.

This technical report is available at ScholarlyCommons: https://repository.upenn.edu/cis_reports/322

Data Abstraction and General Recursion

MS-CIS-91-58
LOGIC & COMPUTATION 35

Ramesh Subrahmanyam

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

August 1991

Data Abstraction and General Recursion

Ramesh Subrahmanyam[1]

Department of Computer and Information Science
University of Pennsylvania
200 South 33rd St., Philadelphia, PA 19104, USA
E-mail: ramesh@saul.cis.upenn.edu

Abstract Existing approaches to semantics of algebraically specified data types such as Initial Algebra Semantics[Goguen *et al.,* 1978] and Final Algebra Semantics[Wand 1979, Kamin 1983] do not take into account the possibility of general recursion and hence non-termination *in the ambient programming language.* Any technical development of this problem needs to be in the setting of domain theory. In this paper we present extensions of initial and final algebra semantics to algebras with an underlying domain structure. Four possibilities for specification methodologies arise: two each in the Initial and Final algebra paradigms. We demonstrate that the initial/final objects (as appropriate) exist in all four situations. The final part of the paper attempts to explicate the notion of abstractness of ADT's by defining a notion of operational semantics for ADT's, and then studying the relationship between the various algebraic-semantics proposed and the operational semantics.

1 Introduction

1.1 Background

Data Abstraction is an important issue in program design: important enough that a variety of programming languages beginning with MODULA-2, and languages such as ADA and ML have supported data encapsulation features. The abstract data type (ADT) facility in a programming language allows the user to create a collection of representing types and functions defined on these types, hiding the representations of the types, and the implementations of the functions, and allowing only the use of the functions defined. A programmer, in creating a data abstraction, uses some form of specification. Oftentimes these are equations [Ehrig & Mahr 1985], Horn Clauses, first-order formulas (John Guttag's LARCH specification Language), or modal equational formulas [Moss & Thatte 1991]. Various specification methodologies exist : *inter alia* Initial Algebra Semantics [Goguen *et al.,* 1978], Final Algebra Semantics [Wand 1979, Kamin 1983], Loose Semantics [Ehrig & Mahr 1985] and Optimal Semantics [Moss & Thatte 1991]. Adopting any one of these specification methodologies, given a specification, a meaning for the specification is obtained. In reasoning about the ADT, no matter what the implementation is, one uses this meaning as the basis for reasoning.

1.2 The Problem

In the theory of programming languages, the theory of domains is an "important tool for making meanings". No doubt other algebraic structures have been proposed and studied: arguably, domain

[1]This research was done while visiting the Software Principles Research Group at AT&T Bell Laboratories, Murray Hill, NJ 07974, during May-July 1991.

theory has proved the most successful. The basic need for this complicated development is driven by two considerations: interpreting general recursion, type constructions such as type sum and type product, and interpreting higher types. Both [Stoy 1977] and [Gunter 1991] are excellent references.

When we use algebraic specifications to specify abstract data types in such a setting, it is not clear *a priori* that any of the semantics alluded to earlier is adequate. In all of the semantic methodologies mentioned , the meanings are Σ-algebras. If one wants to support the view that an abstract data type defined is yet another data type in the language, if one wants to interpret recursive programs involving these data types, one must look for semantics of algebraic specifications in a universe where one can interpret general recursion, the various type constructions (sums, products, recursive types,..), in particular, higher types: since domain theory (and in particular, the category of complete partial orders) provides one such setting, we will try to reconstruct the spirit of these semantic methodologies in domain theory. More precisely, the problem is this: Σ-Algebras do not come with fixpoint theorems. How then can we give meanings to expressions in the programming language, that (possibly)contain symbols in the signature of the abstract data type?

This paper studies the semantics of algebraically specified abstract data types in the context of a language featuring general recursion. It is quite clear, that it would be profitable to make this study in the setting of domain theory. In extending these semantic specification methodologies to domain theory we need to develop new constructions in domain theory.

1.3 Related Work

Quite clearly the Initial Algebra Semantics work of the ADJ group [Goguen *et al.,* 1978], and the Final Algebra work in algebraic semantics (particularly [Wand 1979, Kamin 1983]) are an inspiration for the research presented here. Denotational Semantics developed by Dana Scott and others (see [Gunter 1991, Stoy 1977] for an introduction) is the other area of research on which this work rests. However, in developing algebraic semantics in a domain theoretic setting, the only other work we are aware of is by the ADJ group[Goguen *et al.*, 1977]. This work studies continuous initial algebras in a setting without equations. To the best of our knowledge all the results in this paper are original with us. John Mitchell's work on representation independence and data abstraction ([Mitchell 1985]) is related to ours; the precise relationship of our results to his representation independence theorem is not clear; we plan to study it in the future.

1.4 Preview

Once we have agreed that the objects of specifications have to be domains equipped with appropriate functions for interpreting the symbols in the ADT signature, we need to examine methods of such specifications. Several different styles of specifications may be considered. In the first two styles, all the operations are assumed to be strict; the equations in the specification are expected of only the terminating expressions, or semantically, of only the non-$\perp$ elements. We call these, *specifications with $\perp$-exceptions.*

(I) Initial Algebra Specifications with $\perp$-exceptions specify the structure of the part of the domain algebra excluding the bottom element. All operations specified here will be taken to be strict in all the arguments (of course, nullary operators have no arguments, so they side-step this issue).

Example 1.1 *The specification* $(\{\mathbf{Nat}\}, \{\mathbf{0}, \mathbf{s}, -\}, \mathbf{E_1})$. *where*

$$\mathbf{E_1} = \{\mathbf{x - x = 0},\ \ \mathbf{s(x) - y = s(x - y)},\ \ \mathbf{x - 0 = x},\ \ \mathbf{0 - x = x}\} \tag{1}$$

can be given as a specification of the flat naturals with eager operations(the symbol - denotes proper subtraction), in the semantics described in section 2.2.

Roughly speaking, the structures defined are all those domain algebras whose non-$\perp$ elements satisfy these equations, and all the functions interpreting the constant symbols in the signature are strict.

(II) Final Algebra Specifications with $\perp$-exceptions describe the *observable behavior* of all the non-bottom elements of the ADT. Once again all operations are implicitly strict. The specification makes use of an existing observable type, and gives equations that characterize the equality between terms of observable type.

Example 1.2 *To specify the flat domain of finite sets of naturals, we could use the flat domain of naturals and booleans (and any functions already defined, such as $eqNat$, the equality on naturals or por, "the parallel or"), and write the specification*

$$(\{\mathbf{Nat, Bool, Sets}\},\ \{\in, \mathbf{ins}, \emptyset, \mathbf{por}, \mathbf{true}, \mathbf{false}, \mathbf{eqNat}\}, \mathbf{E_2}),\ \textit{where}$$

$$\mathbf{E_2} = \{\mathbf{x} \in \emptyset = \mathbf{false},\ \ \mathbf{x} \in \mathbf{ins(y, S)} = \mathbf{eqNat(x, y)}\ \ \mathbf{por}\ \ \mathbf{x}\ \ \in \mathbf{S}\} \tag{2}$$

This semantics is described in section 2.3. Once again the structures are domain algebras, the interpretations of function symbols are all strict, the equations hold of all the non-$\perp$ elements, and the domains corresponding to the observable sorts, Nat and Bool, are the standard standard flat naturals and flat booleans, respectively.

(III) Initial Algebra Specifications of domain algebras, that is described in section 3 does not make implicit the strictness of the operators with respect to their argument positions. The specification describes the algebraic structure of the entire domain algebra: the equations hold of all elements of the algebra.

Example 1.3 *Consider the domain of lazy sets : one could view it as the domain obtained by starting with the domain of streams, and collapsing by the equivalence relation that relates two streams that differ only in the order of elements and number of occurances of their members*[2]*. We can describe it via the specification*[3]*, $(\{\mathbf{Sets}\}, \{\emptyset, \mathbf{ins}, \mathbf{conv}\}, \mathbf{E})$, where E is the following set of (conditional) equations*[4]*:*

`ins(⊥,S) = ⊥`

[2]Another way of viewing it is that each element of this domain, viewing the domain as a domain of information, is a positive description of a set.

[3]Notice the use of $\perp$ to specify strictness information. $\perp$ is treated as a pervasive that can be used in any specification.

[4]Indeed we are stepping out of equational logic, but this is largely for purposes of exposition. It is possible to rewrite tis as a pure equation, by introducing a constant *if then else* and writing the corresponding equational axioms.

```
ins(x,ins(x,S)) = ins(x,S)

conv(x) = true, conv(y) = true ==> ins(x,ins(y,S)) = ins(y,ins(x,S))

conv(0) = true            conv(s(x)) = conv(x)
```

In a sense made precise in section 3, the domain of lazy sets with constants empty set and set insertion, is indeed the initial algebra of the specification.

(IV) Final Algebra Specifications of CPO Σ-algebras describe the structure in question by describing the category of domain algebras in which it is final (section 4 defines this more accurately).

Example 1.4 *Consider the specification of lazy sets as before but with the operations of membership and insertion. The observable types we use are again the naturals and the booleans. The specification is:*

$(\{\mathbf{Nat}, \mathbf{Bool}, \mathbf{Sets}\}, \{\in, \mathbf{ins}, \emptyset, \mathbf{por}, \mathbf{true}, \mathbf{false}, \mathbf{eqNat}\}, \mathbf{E_4})$. *where*

$\mathbf{E_4} = \{\mathbf{x} \in \emptyset = \mathbf{false},\ \ \mathbf{x} \in \mathbf{ins}(\mathbf{y}, \mathbf{S}) = \mathbf{eqNat}(\mathbf{x}.\mathbf{y})\ \ \mathbf{por}\ \ \mathbf{x}\ \ \in \mathbf{S}.$

$\mathbf{ins}(\perp, \mathbf{S}) = \perp, \perp \in \mathbf{S} = \perp, \mathbf{x} \in \perp = \perp\}$

In the final algebra semantics that will be explained in Section 4 this specification has as its meaning the domain of lazy sets with the standard domain operations of null set, set insertion (strict in its first argument, and non-strict in its second), and membership testing (strict in both arguments).

In section 5 we formalize a notion of operational semantics of a programming language with abstract data types, and a notion of valid implementations given an algebraic specification. This allows us to study the relationship between the evaluation relation in a valid implementation, and the meaning of program expressions over the ADT signature. The three theorems stated there can be seen as explaining the sense in which ADT's are abstract.

Most work in Algebraic Specification stays aloof of programming languages. This is probably reasonable. Languages do not support specifications of ADT's. There are merely features for creating ADT declarations. But, in the context of supporting formal program development in a language with ADT's (or modules), a programmer should be able to specify the ADT in some logical language (see, for example, Don Sannella's work on Extended ML [Sannella 1990]). A semantic specification methodology then informs us of the meaning of the specification. This becomes the foundation for reasoning about the correctness of programs developed.

1.5 Results

1. Existence of initial and final objects in the category of Σ-strict domain algebras and strict continuous Σ-homomorphisms (Theorems 2.1 and 2.3).

2. Existence of initial object in the category of CPO Σ-Algebras and strict continuous Σ-homomorphisms (Theorem 3.1).

3. A Wand-like Final Algebra theorem for the category mentioned in (2)(Theorem 4.1).

4. Relationships between the final algebra semantics mentioned in (1) and (3), and their respective operational semantics(Theorems 5.1 , 5.2 , and 5.3).

2 Specifications with $\perp$-Exceptions

2.1 Equational Specifications: Definitions

Definition 2.1 *Given sorts S and signature Σ, an S-sorted Strict Domain Σ-algebra (Σ-SDA) is an S-indexed collection of CPO's $(D_s)_{s\in S}$, and an interpretation function $\mathcal{I}$, such that for any constant in the signature $f : s_1, .., s_n - s_{n+1}$.*

$$\mathcal{I}(f) \in D_{s_1} \otimes .. \otimes D_{s_n} -_s D_{s_{n+1}}$$

where $\otimes$ is the "smash-product" operator on domains, and $-_s$ is the strict function space constructor.

When we deal with Final Algebra Semantics certain types will be given *a priori*, and their denotations will be given. In that case, in any Σ-SDA, the universes corresponding to these sorts will have to be to be given.

The definition of a Σ-SDA homomorphism is straightforward.

Definition 2.2 *Given two Σ-SDA's $DA^{(1)}$ and $DA^{(2)}$, a Σ-SDA homomorphism from $DA^{(1)}$ to $DA^{(2)}$ is a S-indexed family of maps $(h_s : DA^{(1)}_s - DA^{(2)}_s)$ such that:*

1. *Each h_s is strict and continuous.*
2. *Ignoring the order structure of $DA^{(1)}$ and $DA^{(2)}$, $(h_s)_{s\in S}$ is a Σ-algebra homomorphism.*

We consider only strict (continuous) functions between algebras, because the notion of homomorphism embodies the notion of a range element "mimicking" the behavior of its preimage, and if the observation of behavior includes observing termination (as we intend to do) we have to build in the notion that the image of the bottom element cannot be non-bottom.

Given a Σ-SDA, the meaning of a term with respect to an assignment ρ is a straightforward adaptation of that in the case of Σ-algebras. That is, we ignore the order structure of the universe and pretend we are dealing with a Σ-algebra.

Definition 2.3 *A Σ-SDA satisfies an equation $t_1 = t_2$ if and only if for every assignment ρ into the Σ-SDA, such that no variable is assigned $\perp$ by ρ, $[\![t_1]\!]\rho = [\![t_2]\!]\rho$. This extends straight-forwardly to a set of equations.*

2.2 Initial Algebra Semantics

Suppose we want to specify the structure of a flat domain , with every operator being strict. Very often in computing practice it is important to specify such structures. For example, a programmer might specify the flat natural numbers with the operations of strict successor, strict addition, and the constant 0. One can view the structure as being the standard algebra of naturals with these operations given, with a constant $\perp$ adjoined to it. Since strictness is implicit, specifying the structure of the universe excluding $\perp$ is sufficient to specify the entire structure.

Initial Algebra Specifications with $\perp$-exceptions specify the structure of the part of the domain algebra excluding the bottom element. All operations specified here will be taken to be strict in all the arguments (and of course, nullary operators have no arguments, so they side-step this issue). Let us re-examine Example (1.1). We want to specify the algebra whose universe is the flat domain of natural numbers, and the operations of strict and strict proper subtraction, and the constant 0. Let us call this algebra $N_\perp$. The specification is the following: $(\{\mathbf{Nat}\}, \mathbf{\Sigma_1}, \mathbf{E_1})$, where

$$\Sigma_1 = \{0, s, -\}$$

$$\mathbf{E_1} = \{\mathbf{x - x = 0},\ \ \mathbf{s(x) - y = s(x - y)},\ \ \mathbf{x - 0 = x},\ \ \mathbf{0 - x = x}\}$$

Roughly speaking, the structures defined are all those domain algebras whose non-$\perp$ elements satisfy these equations, and all the functions interpreting the constant symbols in the signature are strict.

The basic idea in initial algebra semantics is this: given an equational specification (Σ, E), if one considers the category of all Σ-algebras that satisfy E, there is an initial object (i.e.) an algebra, unique up to isomorphism from which there is a unique Σ-homomorphism to any other algebra in the category. This algebra is then taken to be the meaning of the specification.

The idea of initiality extends to the case in hand. Our specification describes a class of strict domain algebras, namely those that satisfy the equations in it. The initial object in this class of algebras is then taken as the meaning of the specification. But does an initial object always exist? First let us establish that one exists in this particular case.

Lemma 2.1 *In Example (1.1) the initial algebra in the category of Σ_1-SDA's satisfying E_1 is $N_\perp$.*

Proof : Consider any other Strict Domain Algebra D with signature $\{0, s, +\}$ (call it Σ_1) satisfying the equations E_1[5]. Every non-$\perp$ element in $N_\perp$ is of the form $s^n(0)$, for some $n \geq 0$. By the definition of Σ-SDA homomorphism in definition 2.2, any homomorphism has to be strict, and hence the image of $\perp$ is determined. Also the element 0 in $N_\perp$ has to map to the denotation of 0 in the range SDA. Since for any homomorphism h,

$$h(s^{N_\perp}(x)) = s^D(h(x)) \tag{3}$$

by induction on the structure of $s^n(0)$ it is clear that the image of all elements in $N_\perp$ is determined. Thus the homomorphism from $N_\perp$ (if one exists) is unique.

Existence is verified by noting that the strict map, that maps each number n in $N_\perp$ to the denotation of $s^n(0)$ in D is indeed a SDA-homomorphism. This involves checking several cases. For example, to check that

$$h(s^n(0) - s^m(0)) = s^n_D(h(0)) - s^m_D(h(0)) \tag{4}$$

we do a routine induction on n and m (here s_D is the interpretation of s in the algebra D; the unsubscripted s refers to the interpretation of s in $N_\perp$).

Thus $N_\perp$ is initial in the category of Σ_1-strict domain algebras satisfying E_1. ∎

The construction in the example above is easily generalized.

[5]Of course *satisfaction* in the sense of strict domain algebras, is as stated in definition 2.3.

Construction 2.1 *Given a set of equations, define a Σ-SDA $\mathcal{I}_E$ as follows:*

Let $\mathbf{I}(E) = \{[s]_{=_E} \mid s \in T_\Sigma(\emptyset)\}$.

The universe $\mathbf{I}_\perp(E)$ *is the flat domain whose non-*$\perp$ *elements are exactly the elements of* $\mathbf{I}(E)$. *Every n-ary function symbol f is interpreted by a corresponding function* $f^{\mathbf{I}_\perp(E)}$:

$$f^{\mathbf{I}_\perp(E)}(e_1,..,e_n) = \begin{cases} \perp & \text{if for some } i\ e_i = \perp \\ ([f(t_1,..,t_n)]_{=_E}) & \text{if } \forall i.e_i = [t_i] \end{cases}$$

It is easily verified that this is a Σ-SDA. It is an easy exercise to verify that it satisfies the equations E. We can go on further, generalizing the proof of Lemma 2.1, to establish the following theorem:

Theorem 2.1 *The category of Σ-SDA's satisfying equations E,and Σ-SDA homomorphisms has $\mathcal{I}$ as its initial object.*

2.3 Final Algebra Semantics

The idea in Final Algebra Specification is to describe observable behavior of terms. Certain sorts in the specification are stated as being observable, and a certain part of the signature is given as the observable signature (from which the observable terms are constructed). The sets and associated operations corresponding to the observable sorts are given before hand (they may be presented via a set of equations, whose initial algebra is then taken as the interpretation of those sorts). We look at the category of those Σ-algebras (a) that satisfy the equations in the specification, (b) are reachable and (c) whose universes corresponding to the observable sorts are exactly the sets that are given *a priori* as the meanings of the observable sorts. The meaning of the specification is then taken to be the final object in this category. When can this object be guaranteed to exist? We describe some properties that are imposed to this end.

We have not completely specified the behavior of the objects of our observable data type if there is some observable context and some term of the type being defined, such that upon inserting the term in the context, the resultant term is not provable equal to any term over the observable signature. So we require the following property.

Definition 2.4 *An equational specification is* **sufficient complete** *if and only if for any closed term t of observable type there is some closed term s over the observable signature such that* $E \vdash t = s$. *Given t, the function* **eval** *returns the denotation of s in the observable type.*

The observable types are known, it is not the specification writer's intent to specify them. Thus the following property is reasonable to require.

Definition 2.5 *An equational specification E is said to* **preserve observables** *if and only if for any two closed terms* t_1 *and* t_2 *over the observable signature,* $E \vdash t_1 = t_2$ *if and only if* $\mathbf{OBS} \models t_1 = t_2$.

Adding these properties to our list, the following theorem due to Mitch Wand follows [Wand 1979]:

Theorem 2.2 (Final Algebra Theorem) *For every sufficient complete theory that preserves its observables, the category of reachable models of E has a final object.*

In attempting to extend Final Semantics to our case similar assumptions have to be made. The given observable sorts are required to be flat domains [6]; the equations should preserve observables and be sufficient complete.

An analogous description of reachability must be made. In a Σ-SDA. elements which are the denotations of closed $T_{\Sigma\cup\{\perp\}}$ terms will be called *reachable elements.*

Reachability Assumption: The category of Σ-SDA's considered has as objects those Σ-SDA's in which the set D of reachable elements below every element d is directed. Also, $d = \sqcup D$.

The reachability assumption on Σ-SDA's forces the SDA's to have a flat domain structure wherein all the non-$\perp$ elements are denotations of closed T_Σ terms. The reason for stating the assumption in this seemingly complicated fashion is that for domain algebras in which not all operations are strict (which we will have occasion to consider in the section 4). this more general statement will be applicable.

Let us revisit Example 1.2. We want to specify the "eager set of natural numbers", denoted $NatSet_\perp$. The universe of this algebra is the flat domain whose non-$\perp$ elements are the finite sets of natural numbers. The operations are $\emptyset$, $\in$ and ins. with $\emptyset$ denoting the empty set, $\in$ denoting the membership operation which is strict in both arguments. and ins. the set insertion operation that is strict in both arguments. We are given as the observables the sort of natural numbers (i.e. the domain algebra $N_\perp$ of the previous section). and the flat domain of boolean values, with the following observable signature: $\{por, true, false, eqNat\}$ [7].

The specification is $(\{\mathbf{Nat}, \mathbf{Bool}, \mathbf{Sets}\}, \Sigma_2, E_2)$. with

$$\Sigma_2 = \{\in, \mathbf{ins}, \emptyset, \mathbf{por}. \mathbf{true}, \mathbf{false}. \mathbf{eqNat}\}, \mathbf{E}). \text{ and}$$

$$\mathbf{E_2} = \{\mathbf{x} \in \emptyset = \mathbf{false}.\ \ \mathbf{x} \in \mathbf{ins}(\mathbf{y}.\mathbf{S}) = \mathbf{eqNat}(\mathbf{x}, \mathbf{y})\ \ \mathbf{por}\ \ \mathbf{x}\ \ \in \mathbf{S}\}$$

We claim the following:

Lemma 2.2 *The equations in E_2 preserve observables . and are sufficiently complete.*

Proof : That the equations are sufficiently complete is trivially seen. As for the preservation of observables, one can make the following argument.

Observe that orienting the equations left to right gives a ground confluent and strongly normalizing rewrite system (in the presence of the equations between the observables), where the signature being considered is Σ_2 extended by a constant $\perp$, and for every function symbol and every argument place there is an equation asserting that the function is strict in that argument. For instance $f(x, \perp) = \perp$ asserts that f is strict in its second argument position. This naturally means that any two provably

[6] We require the domains given as observables to be flat. Equality on non-flat domains is non-monotone, and hence is not computable; so it does not make sense to allow observable types to be non-flat.

[7] The presentation of the two-sorted algebra of flat naturals and flat booleans is by a set of equations holding over the observable signature. The initial algebra of these equations (whose existence was demonstrated in the previous section) will be this two-sorted algebra. We have chosen not to go into the details of these equations.

equal terms reduce to the same normal form when reduced using this rewrite system. A simple analysis of the normal forms suffices to establish that every term of the form $\perp_N$, $\perp_{Bool}$, $s^n(0)$, *true* or *false* is in normal form. Also the equations over observables prove every ground term over the observable signature equal to a term of one of these forms: thus there are terms s_1 and s_2 in the form mentioned, such that t_i is provable equivalent using the equations about the observables to s_i. Thus s_1 and s_2 are provable equal using E_2. This means that they reduce to the same normal form term. But s_1 and s_2 are already in normal form, and hence they are identical. ∎

Lemma 2.3 *Consider the category of reachable Σ_2-SDA 's(whose observable sorts are the standard flat naturals and flat booleans) satisfying the equations E_2. and Σ_2-SDA homomorphisms. The algebra $NatSet_\perp$ is final in this category.*

Proof : Let D be any reachable Σ_2-SDA satisfying E_2. Every element in the sort **Sets** of D has to be the denotation of a term of the form $\perp$, $\emptyset$, or $ins(a_1,...,ins(a_n.\emptyset)..)$. If $h : D \rightarrow NatSet_\perp$ is any homomorphism then necessarily

$$h(\perp) = \perp$$

$$h(\emptyset^D) = \emptyset$$

$$h(ins([\![a_1]\!]^D,..ins([\![a_n]\!]^D,\emptyset))) = \{[\![a_1]\!]^{\mathbf{Sets}},...[\![a_n]\!]^{\mathbf{Sets}}\}$$

This means that h, if it exists is unique. To prove is existence, note that the denotation of a term t in D is mapped to the set of all those naturals n such that in D the meaning of the term $s^n(0) \in t$ is true: call this set elem(t). Also observe that for any two closed terms t_1 and t_2 whose meanings are equal in D, the fact that D has to preserve its observables guarantees that $elem(t_1) = elem(t_2)$. Thus h is well-defined. ∎

We now give a general construction for the final Σ-SDA for an arbitrary equational specification.

Construction 2.2 *Given a specification (S,Σ,E), we construct a Σ-SDA $\mathcal{F}_\perp$ as follows:*

Consider the final algebra $\mathcal{F}$ of E. Construct a flat domain by introducing a bottom element. Extend the interpretation of function symbols (given the interpretation in $\mathcal{F}$) to the flat domain, so as to make all the functions strict in all their arguments.

We now establish that this construction indeed yields the final object in the category of reachable Σ-SDA's satisfying equations E.

Theorem 2.3 *Let E be a set of equations that is* observable preserving *and sufficient complete. The category of reachable Σ-SDA 's satisfying E. with Σ-SDA homomorphisms has a final object.*

Proof : We will work with one representation of the final algebra of E, essentially the one due to [Wand 1979]. Consider the set of closed terms over Σ. If we consider a closed term t of observable type, then sufficient completeness guarantees that there is a term over the observable signature (and hence a value in the observable domain) to which it is provable equal; we will define the function *eval*

to be the map that takes t to this value. Define a binary relation of indistinguishability(notation : $\sim$) between closed terms as follows:

$$t_1 \sim t_2 \iff \forall C[].eval(C[t_1]) = eval(C[t_2]) \tag{5}$$

where $C[]$ ranges over contexts of observable type. It is easy to see that $\sim$ is a congruence, and therefore the closed term algebra over the signature Σ when quotiented by $\sim$ results in a Σ-algebra[8]. This Σ-algebra satisfies the equations E, and is final in the category of Σ-algebras and Σ-homomorphisms[Wand 1979].

Construction 2.2 yields the algebra in which every sort is a flat domain whose non-$\perp$ elements come from the above final algebra, and in which all operators are strict in all their arguments. This is $\mathcal{F}_\perp$.

Let D be any other reachable Σ-SDA satisfying equations E. If we ignore the $\perp$ elements in each of its universes, as well as the order structure we have a Σ-algebra satisfying E (call it D^-), from which there is a unique homomorphism to $\mathcal{F}$. Any Σ-SDA homomorphism from D to $\mathcal{F}$, when restricted to elements of D^- is a Σ-homomorphism; this means that a homomorphism is unique in its definition on the non-$\perp$ part of D. Of course strictness determines its value on the $\perp$ element in the various sorts. Therefore, there is a unique homomorphism into $\mathcal{F}_\perp$.

It is easily established that the strict map which on the non-$\perp$ elements of D is equal to the final map from D^- to $\mathcal{F}$ is indeed a Σ-SDA homomorphism. Thus $\mathcal{F}_\perp$ is indeed the final object in the category of reachable Σ-SDA's satisfying E. ∎

3 Another Initial Algebra Semantics

We now turn our attention to specifications that are expected to hold over the entire domain (thus the $\perp$ element is not excepted in determining the satisfaction of equations), and in which operations may be specified to be strict or non-strict with respect to any argument position. Specifications may refer to the constant $\perp$ in specifying strictness information . The specification of strictness for a function f (in its second argument, say) would be stated as $\mathbf{f}(\mathbf{x}.\perp) = \perp$. This means that strictness information specification can be done as part of the equational specification, and special specification mechanisms need not be instituted for it. Objects of specification will be CPO Σ-algebras defined below.

Definition 3.1 *(i) A partial-order Σ-algebra is one whose underlying universes are partially ordered sets, and the functions associated with symbols in the signature are maps monotone with respect to the partial ordering. Additionally, if we ignore the ordering on the universes, the structure should be a Σ-algebra.*

(ii) A CPO Σ-algebra has as its universes directed-complete partial ordered sets [9]*. Each function symbol should be interpreted as a continuous function, and if we ignore the ordering on the universes, the structure should be a Σ-algebra.*

[8] It is easy to see that the preservation of observables guarantees that any sort corresponding to an observable type will be the initial algebra of the equations used to present the type.

[9] Directed Complete Partial Orders are defined in Definition 3.4. All domain theoretic terminology used here should be explained in detail in any introductory text on domain theory.

Definition 3.2 *A CPO Σ-homomorphism is a sort-indexed collection of strict continuous maps from one CPO Σ-algebra to another, such that if the order structure is ignored, the map is a Σ-homomorphism.*

The notion of satisfaction of equations in a CPO Σ-algebra is as expected: we ignore the order structure, and determine whether the equations hold in the resultant Σ-algebra.

Let us examine Example 1.3 again. Let Σ_3 denote the signature of that specification, namely, {$\emptyset$,*ins*,*conv*}. Let E_3 be the set of equations below:

```
ins(⊥,S) = ⊥

ins(x,ins(x,S)) = ins(x,S)

conv(x) = true, conv(y) = true ==> ins(x,ins(y,S)) = ins(y,ins(x,S))

conv(0) = true            conv(s(x)) = conv(x)
```

We would like to show that in the category of all CPO Σ_3-algebras satisfying equations E_3, where the arrows are CPO Σ_3-homomorphisms, the algebra $LSets$ of lazy sets with the operations of $\emptyset$, set insertion, with a sort of natural numbers endowed with the strict operation of convergence, *conv*, which evaluates to true on all the non-$\perp$ elements, is indeed initial.

Consider the following algebra $FinLS$: its universe is the following disjoint sum of the set S of all finite sets with itself : $\{0\} \times S \cup \{1\} \times S$, with a partial ordering given by:

$$(\mathrm{i,A}) \leq (\mathrm{j.B}) \iff (\mathrm{i} = 0) \wedge (\mathrm{A} \subseteq \mathrm{B})$$

Define:

$$\emptyset^{FinLS} = (1.\emptyset)$$

$$ins(x,(i.A)) = \begin{cases} \perp & \text{if x} = \perp \\ (i.\{x\} \cup A) & \text{otherwise} \end{cases}$$

Here is the intuition. We are constructing the algebra of sets that are finitely constructed from the signature(i.e. not employing recursion). Since we have in our signature a $\perp$ element over the sort of sets, and we have an insertion operator lazy in its second argument, we can construct "incomplete sets" such as $ins(0,\perp)$. These incomplete sets are represented in our model as $(0, A)$. The elements $(1, A)$ represent the "complete sets".

Lemma 3.1 *The algebra $FinLS$ is a partial order Σ_3-algebra that satisfies the equations E_3. Furthermore it is initial in the category of partially ordered Σ_3-algebras satisfying E_3.*

Lemma 3.2 *The universe of algebra $LSets$ is obtained by constructing the ideal completion of the universe of $FinLS$. Its operators are the continuous extensions of the operators of $FinLS$.*

Lemma 3.3 *The algebra LSets is initial in the category of CPO Σ_3-algebras satisfying the equations E_3.*

Proof : Since $FinLS$ embeds into $LSets$ via the inclusion map, any Σ-homomorphic map from $LSets$ must be a Σ-homomorphism on $FinLS$. Thus the map restricted to $FinLS$ is unique. By the previous lemma any continuous map from $LSets$ is completely determined by its values on the elements in $FinLS$. This establishes the uniqueness.

To prove the existence of the map, show that the continuous extension of the initial map from $FinLS$ is indeed Σ-homomorphic. ∎

We generalize this construction to arbitrary specifications.

Definition 3.3 *The closed term pre-order has as its universe all closed terms constructed from $\Sigma \cup \{\bot\}$. The ordering between terms is given by the following rules and axioms:*

$$x \sqsubseteq x \qquad \bot \sqsubseteq x \qquad \frac{x \sqsubseteq y \quad y \sqsubseteq z}{x \sqsubseteq z} trans \qquad \frac{\vec{t} \sqsubseteq \vec{s}}{f(\vec{t}) \sqsubseteq f(\vec{s})} congruence$$

$t_1 \sqsubseteq t_2$ $t_2 \sqsubseteq t_1$, *where* $(t_1 = t_2) \in E$.

The closed term partial order is obtained from the closed term preorder by constructing equivalence classes of the relation $(\sqsubseteq \cap \sqsupseteq)$. The ordering on equivalence classes is inherited from the pre-order. We will write CTP_Σ for this partial order.

Lemma 3.4 *The closed term partial order is a partially ordered Σ-algebra. It satisfies the equations E. Furthermore it is initial in the category of partially ordered Σ-algebras that satisfy E.*

Since we are looking for an initial object in the category of CPO Σ-Algebras, the partial order CTP_Σ needs to be completed into a CPO in some initiality preserving fashion. The technique of ideal completion turns out to be the right technical device.

Definition 3.4 *Given a partially ordered set, a subset is said to be directed if given any two elements in the subset there is an element in the subset which is an upper bound of those two elements.*

A partial ordering is said to be directed-complete (and called a CPO) if and only if every directed subset of it has a least upper bound.

An ideal in a partial ordering is a downward closed, directed subset of the partial order.

The ideal completion of a partial order D, denoted **Idl(D)**, *is the set of all ideals of the partial order, ordered by set inclusion.*

It is a well-known fact that the ideal completion of a partial order results in an algebraic cpo whose compact elements are isomorphic to the partial order.

Construction 3.1 *Define the CPO Σ-algebra $\mathcal{I}$ as follows. Its universes are the CPO's obtained by taking the ideal completion of the universes (D_s) of the partial order CTP_Σ. For any function symbol $f{:}\vec{s} \to s$, let F be the interpretation of f in CTP_Σ. Let $inc_D : D \to$ **Idl**(D) be the inclusion mapping, and $\prod inc_{s_j} \triangleq inc_{D_1} \times .. \times inc_{D_n}$. Let $\prod inc_{s_j} : D_1 \times .. \times D_n \to$ **Idl**$(D_1) \times .. \times$ **Idl**(D_n).*

Then the interpretation of f in $\mathcal{I}$ is the unique continuous map $\hat{f}$ from $\prod \mathbf{Idl}(D_{s_n}) \rightarrow \mathbf{Idl}(D_s)$ that makes the following diagram commute.

$$\begin{array}{ccc} \prod D_{s_j} & \xrightarrow{\prod inc_{s_j}} & \prod \mathbf{Idl}(D_{s_j}) \\ f \downarrow & & \downarrow \hat{f} \\ D_s & \xrightarrow[inc_s]{} & \mathbf{Idl}(D_s) \end{array}$$

We can now prove that if a partial ordered Σ-algebra satisfies an equational theory then the CPO Σ-algebra obtained by taking its ideal completion also satisfies the theory. This guarantees us that $\mathcal{I}$ satisfies the equations in E. We can further go on and establish the following initiality result:

Theorem 3.1 *$\mathcal{I}$ defined in Definition 3.1 is initial in the category of CPO Σ-algebras satisfying E, and CPO Σ-homomorphisms.*

4 Final Object in CPO Σ-Algebras

As we did in the previous three cases we will examine the corresponding example (Example 1.4 in Section 1.4). Here is the specification:

$(\{\mathbf{Nat}, \mathbf{Bool}, \mathbf{Sets}\}, \{\in, \mathbf{ins}, \emptyset, \mathbf{por}, \mathbf{true}, \mathbf{false}, \mathbf{eqNat}\}, \mathbf{E_4})$, where

$\mathbf{E_4} = \{\mathbf{x} \in \emptyset = \mathbf{false},\ \ \mathbf{x} \in \mathbf{ins(y, S)} = \mathbf{eqNat(x, y)}\ \ \mathbf{por}\ \ \mathbf{x}\ \ \in \mathbf{S}.$

$\mathbf{ins}(\perp, \mathbf{S}) = \perp, \perp \in \mathbf{S} = \perp, \mathbf{x} \in \perp = \perp\}$

Consider the category **E4CAT** whose objects are CPO Σ_4-algebras with the following additional properties : the sorts **Nat** and **Bool** are interpreted by the flat natural domain and the flat boolean domain, respectively, the constants *true* and *false* are interpreted in the standard manner, the operator *eqNat* is interpreted as the strict equality function on the domain of naturals, and *por* is interpreted as the "parallel or" function on the domain of booleans. The arrows are CPO Σ_4-homomorphisms.

The algebra *LazySets* has the algebra *LSets* of the previous section as a reduct; additionally, the symbol $\in$ is interpreted by the membership function that is strict in the first argument. Further, if the second argument is an "incomplete set" (0.A), it returns true if the first argument is contained in A, and diverges otherwise.

We aim to show the following:

Lemma 4.1 *The algebra LazySets is final in* **E4CAT**.

Proof : Let us show the existence of a map h: Given any other algebra $\mathcal{A}$ in **E4CAT**, h behaves as follows:

$$h(S) = \begin{cases} \perp & \text{if } S = \perp \\ (0, \{x \mid \mathcal{A} \models x \in S = true\}) & \exists a. \mathcal{A} \models a \in S = \perp \\ (1, \{x \mid \mathcal{A} \models x \in S = true\}) & \forall a. \mathcal{A} \models a \in S \neq \perp \end{cases}$$

We leave it to the reader to verify that h is indeed a Σ-homomorphism.

Uniqueness is shown from the simple observation that any homomorphic map g has to preserve the property that for any natural number n,

$$n \in g(S) = n \in S \tag{6}$$

since g is an identity on the observable sorts. In the algebra $LazySets$, given any two distinct elements S_1 and S_2, there exists a natural n such that:

$$n \in S_1 \neq n \in S_2 \tag{7}$$

This guarantees that there can be at most one map g satisfying the property asked for in Equation (6), namely, the map h defined above. ∎

Once again we generalize the construction. The construction is described in the rest of this section; the proofs thereof are involved and are explained in adequate detail in a companion paper [Subrahmanyam 1991].

We now define a collection of sets of terms that we call *Quasi-Directed Sets*.

Definition 4.1 *A set of closed terms Q is said to be Quasi Directed if and only if the following two conditions hold:*

(i) For every t_1,$t_2 \in Q$, for every finite set of contexts $\{C_1[], ... C_n[]\}$. there is a $t \in Q$ satisfying the following properties (for $1 \leq i \leq n$):

$$eval(C_i[t]) \geq eval(C_i[t_1])), \text{ and } eval(C_i[t]) \geq eval(C_i[t_2])$$

(ii) For every t_1,$t_2 \in Q$, and for every two hole context $C[\_,\_]$, there is a t such that,

$$eval(C[t,t]) \geq evalC[t_1,t_2])$$

Let **CON** be the set of all single-holed contexts of observable type. Let us assume, without loss of generality, that there is exactly one observable type. Our observable type will be a flat domain **D**. With the discrete ordering on **CON**, the function space [**CON** — **D**] is easily seen to be a bounded-complete partial ordering, with pointwise ordering on the functions. We can associate each closed term with a function in this space via the following map:

$$\Gamma(t)(C[]) = eval(C[t])$$

Observe some simple properties. Let us denote the range of Γ by F. Abusing terminology, we will speak of quasi-directed subsets of F. A subset of F is quasi-directed exactly when its pre-image with respect to Γ is quasi-directed. Every directed subset of F,(the ordering being pointwise (notation: $\leq_f$), is also quasi-directed. Observe now. that any quasi-directed set of functions in [**CON** $\rightarrow$ **D**] is also a bounded subset of it. Thus, it makes sense to speak of the least upper bound of a quasi-directed set with respect to the partial order ([**CON** — **D**]. $\leq_f$).

Define the set F^* as follows:

$F^* = \{f : \mathbf{CON} \rightarrow D \mid \exists Q.$ Q is Quasi-directed, and $\bigsqcup Q = f\}$

Construction 4.1 *Define the CPO-algebra* **FIN** *as follows:*

The universe is the CPO F^*. With each n-ary constant f in the signature associate the following function $f^{\mathbf{FIN}}$:

$$f^{\mathbf{FIN}}(e_1,..,e_n) = \begin{cases} \Gamma(h(\Gamma^{-1}(e_1),..,\Gamma^{-1}(e_n))), & \text{if } (e_1,..,e_n) \in F \\ \bigsqcup_{\vec{x} \in \lfloor e \rfloor} h(\vec{x}), & \text{if } (e_1,..,e_n) \notin F \end{cases}$$

The proof of the well-definedness of the algebra above is non-trivial. The proof along with that of the next theorem are described in detail in our technical report [Subrahmanyam 1991].

As we mentioned in subsection 2.3, a reachability assumption must be imposed on the algebras being considered. The subcategory of CPO Σ-Algebras considered has as objects those in which the set D of reachable elements below every element d is directed. Also, $d = \sqcup D$.

Theorem 4.1 *The CPO-algebra* **FIN** *is final in the category of reachable CPO Σ-algebras and CPO Σ-homomorphisms.*

5 Relating Computation and Meaning

5.1 Operational Semantics

An operational semantics for our language can now be given in one of the standard forms (say, Plotkin's SOS style [Plotkin 1981]). Operationally, a program with ADT declarations in it should be viewed as a function that takes as its argument an environment which associates with the ADT an appropriate representing type, and associates each constant in the signature of the ADT with a closed term in the ambient language of appropriate type. Issues of type-checking to ensure that the hidden representations and implementations stay hidden are studied in [Mitchell & Plotkin 1985, Mitchell 1985]. The value of a program expression t is obtained by first replacing every function symbol by the term associated with it (the result of this translation will be written $\overline{t}$), and then using the operational semantics of the ambient language to evaluate the resultant expression.

We can now state formally the notion of a *valid implementation*. Given an implementation consider the domains that are the semantics of the representing types. Associate with each symbol in the signature the function that is denoted by the term that implements that symbol. We consider first the specification styles with $\perp$-exceptions described in Section 2.

1. In any valid implementation of an ADT specification in the initialfinal specification with $\perp$-exceptions style, all such functions should be strict in all their arguments. In such a case we have a Σ-SDA. Call it **Imp**.

2. This depends upon the nature of the specification:

(a) Initial Specification: **Imp** must satisfy the equations E.

(b) Final Specification: The substructure of **Imp** consisting of definable elements can be shown to be a flat CPO. This substructure must satisfy the equations E.

If we consider the initial algebra specification style of Section 3 then given an implementation, every element in every domain involved must be definable. The functions that are the meanings of

the terms implementing the various operations impose a CPO Σ-algebraic structure. The implementation is valid exactly when this structure satisfies E, the equations given, and is initial in the category of CPO Σ-algebras satisfying E.

If we consider final algebra specifications described in Section 4, we consider the collection of definable elements in each of the implementing domains and construct the subdomain generated by that collection. This gives us a reachable CPO Σ-algebra. The implementation is correct exactly when the domains corresponding to the observable sorts are the ones given prior to the specification, and the CPO Σ-algebra satisfies the specified equations E.

We have assumed that there is a least subdomain of a domain containing a given set of elements. One might wonder whether such a beast indeed exists. This is proved in some rigor in the next lemma (using the axiom of choice: is there a proof without recourse to it?).

Lemma 5.1 *Let D be a CPO and S be some subset of it. Then there is a subdomain S^* of D containing S, such that any other subdomain of D containing S has S^* as a subdomain.*

Proof : Let β be the cardinality of D. Define the following function (from $\beta+2$ to 2^D by transfinite recursion:

$\Gamma(0) = S$

$\Gamma(\alpha + 1) = \Gamma(\alpha) \cup \{\bigsqcup \mathrm{E} \mid \mathrm{E} \text{ is a directed subset of } \Gamma(\alpha)\}$

$\Gamma(lim\alpha_i) = \bigcup \Gamma(\alpha_i)$

In the second line, the least upper bound is taken in the partial order D. It is easily seen that $\Gamma(\beta + 1) = \Gamma(\beta)$ (invoke the pigeonhole principle and use the fact that the cardinality of D is β). That $\Gamma(\beta + 1)$ is the subdomain S^* generated follows from the inductive nature of the function. More formally, suppose it is not (i.e.) there is a subdomain R containing S but not S^*. Then there is a least ordinal α such that $\Gamma(\delta)$ is not contained in R but for every ordinal $\alpha^- < \alpha$ $\Gamma(\alpha^-) \subseteq R$. If α is a successor ordinal, then if $\Gamma(\alpha - 1) \subseteq R$. implies the limit of every directed subset of $\Gamma(\alpha - 1)$ is contained in R, R being a domain. This means that $\Gamma(\alpha) \subseteq R$. leading to a contradiction. If α is the limit of the sequence $< \alpha_i >$ then since $\Gamma(\alpha_i) \subseteq R$. it follows that $\Gamma(\alpha) = \bigcup \Gamma(\alpha_i) \subseteq R$, once again leading to a contradiction. Hence $\Gamma(\beta) \subseteq R$. ∎

5.2 Final Algebras and Operational Semantics

We will take the ambient language to be PCF(see [Gunter 1991], and for all practical purposes the same as LCF in [Plotkin 1977]). Notions of observational equivalence can be defined, now that we have an operational semantics. We will discuss one particular case in detail: Suppose the evaluation scheme for the ambient language is call-by-name, and the observations are CBN(i.e. we observe values of observable types, and termination at observable types [10].

In the denotational semantics, $\rightarrow$ is interpreted as the continuous function space [11]. For the Final Σ-SDA case of section 2.3 the model is the continuous type hierarchy over $\mathcal{F}$ (see section 2.3): this model is notated $\mathcal{CF}$. In the Final CPO Σ-Algebra case we have the continuous type-hierarchy over **FIN**: we call it **CFIN**.

[10] For the definitions of these see [Bloom & Riecke 1989]). The treatment of the other two cases, as we point out, doesn't differ significantly.

[11] For lazy observations, one needs to take the lifted function space

What can we state about the relationship between the denotational semantics and the operational semantics?[12]

Theorem 5.1 (Soundness) *Given a valid implementation, if $t \Downarrow v$, where t is a term of observable type, then v is a term over the observable signature, and $\mathcal{CF}$ equates the value of t to the value of v.*

Proof Sketch: Consider Σ-SDA's. Given a valid implementation we can construct the corresponding Σ-SDA **Imp** as outlined in the previous subsection, and the continuous type hierarchy on top of it; call this model $\mathcal{I}$. Clearly $\mathcal{I} \models t = \overline{t}$. Given that there is a soundness theorem for call-by-name PCF with respect to $\mathcal{CF}$, the meanings of $\overline{t}$ and v in $\mathcal{CF}$l are equal. The final map from **Imp** into $\mathcal{F}$ can be extended to a logical relation over $\mathcal{CF}$. By the fundamental theorem of logical relations the meanings of $\overline{t}$ and v in these two models are related by the logical relation. The logical relation on observable types being the identity(because the final map on observable types is the identity), the meanings of t and v are equal in $\mathcal{CF}$. A similar proof can be carried out for CPO Σ-algebras.

∎

The argument sketched above can be uniformly extended to the lazy and CBV models. The only part of the argument that depends on the model (CBN,lazy or CBV) is the one that invokes the soundness theorem for the model with respect to the corresponding operational semantics. The soundness theorem, with the appropriate operational semantics in place, holds for both, the CBV model and the lazy model (see [Gunter 1991]).

A similar argument can be applied to prove computational adequacy. Essentially, we invoke computational adequacy of the operational semantics (without ADT's) with respect to the corresponding denotational model.

Theorem 5.2 (Computational Adequacy) *If a term of base type (abstract or concrete) diverges, in a valid implementation of the ADT, then in $\mathcal{CF}$ its denotation is $\perp$. The corresponding statement for* **CFIN** *is true as well.*

The point that these two theorems demonstrate can be summarized as follows: Consider given any valid implementation; there is a sense in which the value of an ADT expression (in the appropriate model), is an abstraction of the denotation (of the translated expression) in the Σ-SDA corresponding to the implementation. If we place a term t in an observational context and evaluate it to v,say, the denotation of v is the same as the meaning of t in the model of the ADT. This fact is an easy corollary to the soundness theorem. *Thus, we can ignore what concretely our program expressions mean, and pretend that we are computing over the abstract domain given by the semantics.* In the Σ-SDA case there is another sense in which this "abstract domain" is abstract. We can prove that the final algebra is fully abstract with respect to the operational semantics. As in the case of the soundness and computational adequacy theorems this invokes the full abstraction result for the operational semantics(minus ADT's) with respect to the corresponding denotational model.

Theorem 5.3 (Full Abstraction) *Consider Final Specifications with $\perp$ exceptions. Extend the language PCF (call-by-name) with a "parallel if" constant* **pif**, *and consider any valid implementation. Then $\mathcal{CF}$ is fully abstract with respect to CBN observations.*

[12]Of course all this makes sense only in a Final Specification setting (we don't have observables in the Initial Specification setting).

Proof : Proving full abstraction involves proving that two terms t_1 and t_2 are denotationally equal if and only if they are behaviorally indistinguishable. Behavioral indistinguishability means that in any context of base type either they both diverge, or they both converge to the same value. If they are denotationally equal, computational adequacy and soundness guarantee that they are behaviorally indistinguishable. To prove the converse, assume that they have different denotations in the model $\mathcal{CF}$. Since there is a final map (which extends to a logical relation, a fact that is needed later in this argument) from **Imp** to $\mathcal{CF}$, which relates the meanings of t_i in one to the meaning of t_i in the other, and since the meanings of t_1 and t_2 in $\mathcal{CF}$ are different, the denotations of t_1 and t_2 in **Imp** are different as well. By definition the CBN continuous type hierarchy over **Imp** equates the meaning of t_1 with that of $\overline{t_1}$ (and similarly for t_2. Therefore the meanings of $\overline{t_1}$ and $\overline{t_2}$ in this model are different. Invoking the full abstraction of the operational semantics in the presence of a parallel if (minus the ADT's) with respect to the CBN continuous type hierarchy on **Imp**, we can obtain a context $C[]$ of base type that distinguishes them : if this base type is an observable type we are done; if it is the type of the abstract data type, use soundness to observe that $C[t_1]$ and $C[t_2]$ have distinct meanings in the type corresponding to the abstract data type being defined, in the model $\mathcal{CF}$. However this base type is a final algebra, and given any two terms of different denotations (in this case $C[t_1]$ and $C[t_2]$) there is a context $C_1[]$ such that

$$eval(C_1[C[t_1]]) \neq eval(C_1[C[t_2]]) \tag{8}$$

Notice that the denotations of $\overline{C_1[C[t_1]]}$ (= denotation of$(C_1[C[t_1]])$ and $\overline{C_1[C[t_2]]}$ in the continuous type hierarchy over **Imp** are different, and by the soundness theorem the two expressions evaluate differently. The context $C_1[C[]]$ is the distinguishing context, and we have the desired full abstraction theorem. ∎

Notice once again that the part of the theorem that is specific to the models (CBN,lazy or CBV) merely depends on the existence of a full abstraction theorem for the corresponding calculus(without ADT's). Thus, with the corresponding full abstraction theorems in place the proof carries over to those cases as well.

6 Conclusions and Future Work

The question motivating this paper was whether the various paradigms of traditional algebraic semantics extend as specification techniques for abstract data types, when the ambient language is a functional programming language featuring a general recursion primitive. We have formulated the questions in a precise manner, by demonstrating four different specification methodologies for this setting. Two of these methodologies adapt the initial specifications approach, and the other two extend the final specification approach. We believe the four specification methodologies described in this paper are fairly well-suited for practice. We have instantiated the necessary technical machinery to guarantee that these methodologies make mathematical sense.

We have also addressed questions typically addressed in the theory of programming languages to the setting at hand. Having formulated a notion of operational semantics for a lambda calculus featuring general recursion and abstract data types, we have studied the relationship between the denotational semantics and the operational semantics. We believe the soundness and computational adequacy theorems give us some justification for viewing the final algebra models as the meanings of abstract data types, adding weight to the argument for the final specification methodology. The

full abstraction result for the final specification methodology with $\perp$-exceptions is a technical result: arguably, it can be viewed as a justification for the adjective "abstract" in ADT's.

There are several open research questions.

1. Can we obtain a full abstraction theorem for the Final Algebra Specification methodology described in section 4? It might be possible to prove full abstraction for the semantics we have proposed, or one could look for a new construction in some category such as bounded complete partial orders, dI domains, bifinite domains or L-domains (see [Gunter 1991]).

2. Logics of Programs for each of the specification methodologies.

3. We conjecture that these results can be extended to specification formalisms beyond equations: in particular, Horn Clauses and Harrop formulas.

4. Relationship between our theorems in the last section and the representation independence results in [Mitchell 1985].

Acknowledgements: It is a pleasure to acknowledge the support and encouragement of Elsa Gunter. Elsa spent many hours discussing this work. Her criticisms and suggestions have contributed significantly to my understanding of the topic, and I am deeply indebted to her. I would also like to thank Dave MacQueen for discussion and encouragement.

References

[Bloom & Riecke 1989] B. Bloom and J. Riecke. LCF Should be Lifted. In *Proceedings of Algebraic Methods and Software Technology*, pages 72–118, 1989.

[Ehrig & Mahr 1985] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: equations and initial semantics* . Springer-Verlag, 1985.

[Goguen *et al.*, 1977] J. A. Goguen, J. W. Thatcher, and E. G. Wagner. Initial Algebra Semantics and Continuous Algebras. *Journal of the ACM*, 24(1):68–95, January 1977.

[Goguen *et al.*, 1978] 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.T. Yeh, editor, *Current Trends in Programming Methodology*. Prentice-Hall, 1978.

[Gunter 1991] C. A. Gunter. Structures and Techniques for the Semantics of Programing Languages. 1991. Lecture Notes.

[Kamin 1983] S. Kamin. Final Data Types and their Specifications. *ACM Transactions on Programming Languages and Systems*, 5(1):97–123, January 1983.

[Mitchell & Plotkin 1985] J. C. Mitchell and G. D. Plotkin. Abstract Types have Existential Type. In *Proceedings of the 12th Symposium on Principles of Programming Languages*, pages 37–51, ACM, January 1985.

[Mitchell 1985] J. C. Mitchell. Representation Independence and Data Abstraction. 1985.

[Moss & Thatte 1991] L. S. Moss and S. R. Thatte. Modal Logic and Algebraic Specifications. *Theoretical Computer Science*, 1991. To Appear.

[Plotkin 1977] G. D. Plotkin. LCF Considered as a Programming Language. *Theoretical Computer Science*, 5(3):223–256, December 1977.

[Plotkin 1981] G. Plotkin. *A Structural Approach to Operational Semantics.* Technical Report, Computer Science Department, Aarhus University, Denmark, 1981.

[Sannella 1990] D. A. Sannella. Formal Program Development in Extended ML for the working Programmer. In *Proceedings of the 3rd BCSFACS Workshop on Refinement*, January 1990.

[Stoy 1977] J. E. Stoy. *Denotational Semantics: the Scott-Strachey Approach to Programming Languages Theory.* MIT Press, 1977.

[Subrahmanyam 1991] R. Subrahmanyam. *Final Algebra Semantics in Domain Theory.* Technical Report TM 11261-910724-15. AT&T Bell Laboratories, Murray Hill, 1991.

[Wand 1979] M. Wand. Final Algebra Semantics and Data Type Extensions. *Journal of Computer and System Sciences*, 19:27–44, 1979.