1-14-1980

Using category theory to design implicit conversions and generic operators

John C. Reynolds
Carnegie Mellon University, jr4g@andrew.cmu.edu

Published In
.

This is a preprint of a paper which will appear in the Proceedings of the Aarhus Workshop on Semantics-Directed Compiler Generation (January 14-18, 1980) in the Springer-Verlag Lecture Note Series in Computer Science.

USING CATEGORY THEORY TO DESIGN IMPLICIT CONVERSIONS AND GENERIC OPERATORS[†]

John C. Reynolds
Syracuse University
Syracuse, New York

ABSTRACT A generalization of many-sorted algebras, called category-sorted algebras, is defined and applied to the language-design problem of avoiding anomalies in the interaction of implicit conversions and generic operators. The definition of a simple imperative language (without any binding mechanisms) is used as an example.

Introduction

A significant problem in the design of programming languages is the treatment of implicit conversions, sometimes called coercions, between types. A failure to provide implicit conversions can degrade the conciseness and readability of a language. On the other hand, unless great care is taken in the design of such conversions, and their interaction with operators which can be applied to operands of several types, the resulting language will exhibit anomalies that will be a rich source of programming errors. (In the author's opinion, PL/I and Algol 68 exemplify this danger.)

As a simple illustration, consider assigning the sum of two integer variables to a real variable. In the absence of an implicit conversion from integer to real, one would have to write either

x := integer-to-real(m) + integer-to-real(n)

or

x := integer-to-real(m + n) .

Clearly, one would prefer to write x := m + n. If the language permits this, however, one can ask whether the implicit conversion precedes or follows the addition, i.e., which of the above statements is equivalent to x := m + n.

[†] Work supported by National Science Foundation Grant MCS 75-22002.

It is generally believed that a precise language definition must answer this question unambiguously. However, if one were to ask the question of a mathematician (at least one who didn't know too much about programming), he would probably reply that it doesn't matter, since both of the above statements have the same meaning, and that indeed the whole point of permitting the same operator + to be applied to arguments of different type which are connected by an implicit conversion is that the resulting ambiguity should not affect the meaning.

In a sense, of course, the mathematician is wrong: some computers provide a floating-point representation with such limited precision that the ambiguity in question does affect meaning. But in a deeper sense the mathematician is right. One intuitively expects that the above statements should have nearly the same meaning, and in analogous cases where numerical approximation or overflow is not involved, one expects exactly the same meaning.

To see this, replace real by character string in the above example, and suppose that integers are implicitly converted into character strings giving their decimal representation, and that + denotes both addition of integers and concatenation of strings. Then the two possible meanings of x := m + n are radically different. This case is clearly a mistake in language design which would be likely to cause programming errors.

In this paper we will describe a method for avoiding such errors. The underlying mathematical tool will be a generalization of many-sorted algebras called category-sorted algebras, which are closely related to the order-sorted algebras invented by Goguen.(1)

Beyond the specific goal of treating implicit conversions, our presentation is intended to illustrate the potential of category theory in the area of language definition and to suggest that the "standard" denotational semantics developed by Scott and Strachey may not be the final solution to the language-definition problem. There is nothing incorrect about the Scott-Strachey methodology, and it has provided fundamental insights into many aspects of programming languages such as recursion. But it has not been so helpful in other areas of language design such as type structure. We suspect that clearer insights into these areas will require quite different applications of mathematics.

Conventional Many-Sorted Algebras

Our use of algebras is based on the ideas of Goguen, Thatcher, Wagner, and Wright,(2) which have roots as far back as Burstall and Landin.(3) In (2) a language is viewed as an initial algebra and its semantic function as the unique homomorphism from this initial algebra into some target algebra, so that defining the target algebra is tantamount to defining semantics. Here we will adopt the slightly more elaborate view that (roughly speaking) a language is the free algebra generated by some set of identifiers, that an environment is a mapping of these identifiers into the carrier of the target algebra, and that the semantic function is the function which maps each environment into its unique extension as a homomorphism from the free algebra to the target algebra.

We propose to treat implicit conversions in this framework by generalizing the concept of an algebra appropriately. To motivate this proposal we will proceed through a sequence of increasingly general definitions of "algebra".

The standard concept of a many-sorted algebra used in algebraic semantics is due to Birkhoff and Lipson,(4) who called it an "heterogeneous" algebra. According to Birkhoff and Lipson, but with changes of notation and terminology to reveal the similarity to later definitions:

(1) A signature consists of:

(1a) A set Ω of sorts. (Informally, the sorts correspond to types in a programming language.)

(1b) A family, indexed by nonnegative integers, of disjoint sets Δ_n of operators of rank n.

(1c) For each $n \geq 0$ and $\delta \in \Delta_n$, a specification $\Gamma_\delta \in \Omega^n \times \Omega$. (Informally, if $\Gamma_\delta = \langle\langle\omega_1, \ldots, \omega_n\rangle, \omega\rangle$ then the operator δ accepts operands of sorts $\omega_1, \ldots, \omega_n$ and yields a result of sort ω.)

(2) An <u>$\Omega\Delta\Gamma$-algebra</u> consists of:

(2a) A <u>carrier</u> B, which is an Ω-indexed family of sets. (Informally $B(\omega)$ is the set of meanings appropriate for phrases of type ω.)

(2b) For each $n \geq 0$ and $\delta \in \Delta_n$, an <u>interpretation</u> $\gamma_\delta \in B(\omega_1) \times \ldots \times B(\omega_n) \rightarrow B(\omega)$, where $\langle\langle\omega_1, \ldots, \omega_n\rangle, \omega\rangle = \Gamma_\delta$.

(3) If B, γ and B', γ' are $\Omega\Delta\Gamma$-algebras, then a <u>homomorphism</u> from B, γ to B', γ' is an Ω-indexed family of functions $\theta(\omega) \in B(\omega) \rightarrow B'(\omega)$ such that, for all $n \geq 0$ and $\delta \in \Delta_n$, the diagram

$$\begin{tikzcd}
B(\omega_1) \times \ldots \times B(\omega_n) \arrow[r, "\gamma_\delta"] \arrow[d, "\theta(\omega_1) \times \ldots \times \theta(\omega_n)"] & B(\omega) \arrow[d, "\theta(\omega)"] \\
B'(\omega_1) \times \ldots \times B'(\omega_n) \arrow[r, "\gamma'_\delta"] & B'(\omega)
\end{tikzcd}$$

commutes. Here $\langle\langle\omega_1, \ldots, \omega_n\rangle, \omega\rangle = \Gamma_\delta$ and $f_1 \times \ldots \times f_n$ denotes the function such that $(f_1 \times \ldots \times f_n)(x_1, \ldots, x_n) = \langle f_1(x_1), \ldots f_n(x_n)\rangle$.

Unfortunately, it is difficult to pose the implicit-conversion problem within this concept of algebra since there is no mechanism for grouping operators which are represented by the same symbol. For example, integer addition and real addition would be distinct members of Δ_2 (with specifications <<<u>integer</u>, <u>integer</u>>, <u>integer</u>> and <<<u>real</u>, <u>real</u>>, <u>real</u>>), and there is no mechanism for relating their interpretations more closely than, say, integer addition and multiplication.

Many-Sorted Algebras with Generic Operators

To solve this problem, we will employ an alternative concept of many-sorted algebras due to Higgins.(5) In this approach, the operators are (in programming jargon) generic. The specification of an operator of rank n is a partial function from Ω^n to Ω, which is defined for the combinations of sorts of operands to which the operator is applicable, and which maps each such combination into the sort of the result yielded by the operand. (Notice that this captures the idea of bottom-up type determination.) Then the interpretation of the operator is a family of n-ary functions indexed by the domain of its specification.

In our own development we will insist that the specification be a total function from Ω^n to Ω. At first sight, this simplification might appear to be untenable since it implies that every operator can be applied to operands of arbitrary sorts. Formally, however, the situation can be saved by introducing a "nonsense" sort ns, which is the sort of "type-incorrect" phrases. (If a phrase is type-incorrect whenever any of its subphrases are type-incorrect, then every specification will yield ns whenever any of the sorts to which it is applied is ns. However, one can conceive of contexts, such as the application of a constant function, where this assumption might be relaxed.)

With this simplification, and a few changes of notation and terminology, Higgins' concept of a many-sorted algebra is:

(1) A signature consists of:

(1a) (as before) A set Ω of sorts.

(1b) (as before) A family, indexed by nonnegative integers, of disjoint sets Δ_n of operators of rank n.

(1c) For each $n \geq 0$ and $\delta \in \Delta_n$, a specification $\Gamma_\delta \in \Omega^n \rightarrow \Omega$. (Informally, $\Gamma_\delta(\omega_1, \ldots, \omega_n)$ is the sort of result yielded by the generic operator δ when applied to operands of sorts $\omega_1, \ldots, \omega_n$.)

(2) An *$\Omega\Delta\Gamma$-algebra* consists of:

(2a) (as before) A *carrier* B, which is an Ω-indexed family of sets.

(2b) For each $n \geq 0$ and $\delta \in \Delta_n$, an *interpretation* γ_δ, which is an Ω^n-indexed family of functions $\gamma_\delta(\omega_1, \ldots, \omega_n) \in B(\omega_1) \times \ldots \times B(\omega_n) \to B(\Gamma_\delta(\omega_1, \ldots, \omega_n))$. (Informally, $\gamma_\delta(\omega_1, \ldots, \omega_n)$ is the interpretation of the version of the generic operator δ which is applicable to sorts $\omega_1, \ldots, \omega_n$.)

(3) If B,γ and B',γ' are $\Omega\Delta\Gamma$-algebras, then an *homomorphism* from B,γ to B',γ' is an Ω-indexed family of functions $\theta(\omega) \in B(\omega) \to B'(\omega)$ such that, for all $n \geq 0$, $\delta \in \Delta_n$, and $\omega_1, \ldots, \omega_n \in \Omega$, the diagram

$$\begin{tikzcd}
B(\omega_1) \times \ldots \times B(\omega_n) \arrow[r, "\gamma_\delta(\omega_1, \ldots, \omega_n)"] \arrow[d, "\theta(\omega_1) \times \ldots \times \theta(\omega_n)"] & B(\Gamma_\delta(\omega_1, \ldots, \omega_n)) \arrow[d, "\theta(\Gamma_\delta(\omega_1, \ldots, \omega_n))"] \\
B'(\omega_1) \times \ldots \times B'(\omega_n) \arrow[r, "\gamma'_\delta(\omega_1, \ldots, \omega_n)"] & B'(\Gamma_\delta(\omega_1, \ldots, \omega_n))
\end{tikzcd} \qquad (I)$$

commutes.

Algebras with Ordered Sorts

We can now introduce the notion of implicit conversion. When there is an implicit conversion from sort ω to sort ω', we write $\omega \leq \omega'$ and say that ω is a *subsort* (or *subtype*) of ω'. Syntactically, this means that a phrase of sort ω can occur in any context which permits a phrase of sort ω'.

It is reasonable to expect that $\omega \leq \omega$ and that $\omega \leq \omega'$ and $\omega' \leq \omega''$ implies $\omega \leq \omega''$. Thus the relation $\leq$ is a preordering (sometimes called a quasiordering) of the set Ω. Actually, in all of the examples in this paper $\leq$ will be a partial ordering, i.e., $\omega \leq \omega'$ and $\omega' \leq \omega$ will only hold when $\omega = \omega'$. However, our general theory will not impose this additional requirement upon $\leq$.

Now suppose δ is an operator of rank n, and $\omega_1, \dots, \omega_n$ and $\omega'_1, \dots, \omega'_n$ are sorts such that $\omega_i \le \omega'_i$ for each i from one to n. Then a context which permits a phrase of sort $\Gamma_\delta(\omega'_1, \dots \omega'_n)$ will permit an application of δ to operands of sorts $\omega'_1, \dots, \omega'_n$. But the context of the ith operand will also permit an operand of sort ω_i, so that the overall context must also permit an application of δ to operands of sort $\omega_1, \dots, \omega_n$, which has sort $\Gamma_\delta(\omega_1, \dots, \omega_n)$. Thus we expect that $\Gamma_\delta(\omega_1, \dots, \omega_n) \le \Gamma_\delta(\omega'_1, \dots, \omega'_n)$ or, more abstractly, that the specification Γ_δ will be a monotone function.

If $\omega \le \omega'$ then an algebra must specify a conversion function from the set $B(\omega)$ of meanings appropriate to ω to the set $B(\omega')$of meanings appropriate to ω'. At first sight, one might expect that this can only occur when $B(\omega)$ is a subset of $B(\omega')$, and that the conversion function must be the corresponding identity injection. For example, <u>integer</u> can be taken as a subsort of <u>real</u> because the integers are a subset of the reals.

However there are other situations in which this is too limited a view of implicit conversion. For example, we would like to say that <u>integer variable</u> is a subsort of <u>integer expression</u>, so that integer variables can occur in any context which permits an integer expression. But it is difficult to regard the meanings of integer variables as a subset of the meanings of integer expressions. In fact, we will regard the meaning of an integer variable as a pair of functions: an acceptor function, which maps integers into state transformations, and an evaluator function, which maps states into integers. Then the meaning of an expression will just be an evaluator function, and the implicit conversion function from variables to expressions will be a function on pairs which forgets their first components.

In general, we will permit implicit conversion functions which forget information and are therefore not injective. To paraphrase Jim Morris,(6) subtypes are not subsets. This is the main difference between our approach and that of Goguen.(1) (There are some more technical differences, particularly in the definition of signatures, whose implications are not completely clear to this author.)

However, there are still some restrictions that should be imposed upon implicit conversion functions. The conversion function from any type to itself should be an identity function. Moreover, if $\omega \leq \omega'$ and $\omega' \leq \omega''$ then the conversion function from $B(\omega)$ to $B(\omega'')$ should be the composition of the functions from $B(\omega)$ to $B(\omega')$ and from $B(\omega')$ to $B(\omega'')$. This will insure that a conversion from one sort to another will not depend upon the choice of a particular path in the preordering of sorts.

These restrictions can be stated more succinctly by invoking category theory. A preordered set such as Ω can be viewed as a category with the members of Ω as objects, in which there is a single morphism from ω to ω' if $\omega \leq \omega'$ and no such morphism otherwise. Suppose we write $\omega \leq \omega'$ to stand for the unique morphism from ω to ω' (as well as for the condition that this morphism exists), and require the carrier B to map each $\omega \leq \omega'$ into the conversion function from $B(\omega)$ to $B(\omega')$. Then we have

(i) $B(\omega \leq \omega') \in B(\omega) \rightarrow B(\omega')$.

(ii) $B(\omega \leq \omega) = I_{B(\omega)}$.

(iii) If $\omega \leq \omega'$ and $\omega' \leq \omega''$ then

$$B(\omega \leq \omega'') = B(\omega \leq \omega');B(\omega' \leq \omega'') .$$

(Throughout this paper we will use semicolons to indicate composition in diagrammatic order, i.e., $(f;g)(x) = g(f(x))$.) These requirements are equivalent to saying that B must be a functor from Ω to the category SET, in which the objects are sets and the morphisms from S to S' are the functions from S to S'.

This leads to the following definition:

(1) A <u>signature</u> consists of:

(1a) A preordered set Ω of <u>sorts</u>.

(1b) (as befoŕe) A family, indexed by nonnegative integers, of disjoint sets Δ_n of <u>operators</u> of rank n.

(1c) For each $n \geq 0$ and $\delta \in \Delta_n$, a <u>specification</u> Γ_δ, which is a monotone function from Ω^n to Ω.

(2) An *ΩΔΓ-algebra* consists of:

(2a) A *carrier* B, which is a functor from Ω to SET.

(2b) For each $n \geq 0$ and $\delta \in \Delta_n$, an *interpretation* γ_δ, which is an Ω^n-indexed family of functions $\gamma_\delta(\omega_1, \dots , \omega_n) \in B(\omega_1) \times \dots \times B(\omega_n) \to B(\Gamma_\delta(\omega_1, \dots , \omega_n))$ such that, whenever $\omega_1 \leq \omega_1', \dots , \omega_n \leq \omega_n'$, the diagram

$$\begin{tikzcd}
B(\omega_1) \times \dots \times B(\omega_n) \arrow[r, "\gamma_\delta(\omega_1, \dots ,\omega_n)"] \arrow[d, "B(\omega_1 \leq \omega_1') \times \dots \times B(\omega_n \leq \omega_n')"] & B(\Gamma_\delta(\omega_1, \dots ,\omega_n)) \arrow[d, "{B(\Gamma_\delta(\omega_1, \dots ,\omega_n) \leq \Gamma_\delta(\omega_1', \dots \omega_n'))}"] \\
B(\omega_1') \times \dots \times B(\omega_n') \arrow[r, "\gamma_\delta(\omega_1', \dots ,\omega_n')"] & B(\Gamma_\delta(\omega_1', \dots ,\omega_n'))
\end{tikzcd} \qquad \text{(II)}$$

commutes.

The above diagram asserts the relationship between generic operators and implicit conversions which originally motivated our development. To recapture our original example, suppose *integer*, *real* $\in \Omega$, *integer* $\leq$ *real*, $+ \in \Delta_2$, Γ_+(*integer*, *integer*) = *integer*, and Γ_+(*real*, *real*) = *real*. Then a particular instance of the above diagram is

$$\begin{tikzcd}
B(\text{integer}) \times B(\text{integer}) \arrow[r, "\gamma_+(\text{integer},\text{integer})"] \arrow[d, "B(\text{integer} \leq \text{real}) \times B(\text{integer} \leq \text{real})"] & B(\text{integer}) \arrow[d, "B(\text{integer} \leq \text{real})"] \\
B(\text{real}) \times B(\text{real}) \arrow[r, "\gamma_+(\text{real}, \text{real})"] & B(\text{real})
\end{tikzcd} .$$

In other words, the result of adding two integers and converting their sum to a real number must be the same as the result of converting the integers and adding the converted operands.

In essence, the key to insuring that implicit conversions and generic operators mesh nicely is to require a commutative relationship between these entities. An analogous relationship must also be required between implicit conversions and homomorphisms:

(3) If B,γ and B',γ' are $\Omega\Delta\Gamma$-algebras, then an <u>homomorphism</u> from B,γ to B',γ' is an Ω-indexed family of functions $\theta(\omega) \in B(\omega) \to B'(\omega)$ such that, whenever $\omega \leq \omega'$, the diagram

$$\begin{tikzcd}
B(\omega) \arrow[r, "\theta(\omega)"] \arrow[d, "B(\omega\leq\omega')"'] & B'(\omega) \arrow[d, "B'(\omega\leq\omega')"] \\
B(\omega') \arrow[r, "\theta(\omega')"] & B'(\omega')
\end{tikzcd} \qquad \text{(III)}$$

commutes, and (as before) for all $n \geq 0$, $\delta \in \Delta_n$, and $\omega_1, \dots, \omega_n \in \Omega$, the diagram (I) commutes.

Category-Sorted Algebras

By viewing the preordered set of sorts as a category, we have been able to use the category-theoretic concept of a functor to express appropriate restrictions on implicit conversion functions. In a similar vein, we can use the concept of a natural transformation to express the relationship between implicit conversion functions and interpretations given by diagram (II) and the relationship between implicit conversion functions and homomorphisms given by diagram (III).

In fact, diagram (III) is simply an assertion that the homomorphism θ is a natural transformation from the functor B to the functor B'. Diagram (II), however, is more complex. To express this diagram as a natural transformation, we must first define some notation for the exponentiation of categories and functors, and for the Cartesian product functor on SET:

(1) For any category K, we write:

(a) $|K|$ for the set (or collection) of objects of K.

(b) $X \underset{K}{\to} X'$ for the set of morphisms from X to X' in K.

(c) I^K_X for the identity morphism of X in K.

(d) $;_K$ for composition in K.

(2) For any category K, we write K^n to denote the category such that:

(a) $|K^n| = |K|^n$, i.e. the n-fold Cartesian product of $|K|$.

(b) $\langle X_1, \dots , X_n\rangle \underset{K^n}{\to} \langle X'_1, \dots , X'_n\rangle$
$= (X_1 \underset{K}{\to} X'_1) \times \dots \times (X_n \underset{K}{\to} X'_n)$.

(c) $I^{K^n}_{\langle X_1, \dots , X_n\rangle} = \langle I^K_{X_1}, \dots , I^K_{X_n}\rangle$.

(d) $\langle \rho_1, \dots , \rho_n\rangle ;_{K^n} \langle \rho'_1, \dots , \rho'_n\rangle = \langle \rho_1 ;_K \rho'_1, \dots , \rho_n ;_K \rho'_n\rangle$.

(Notice that when K is a preorder (e.g. Ω) this definition is consistent with the usual notion (e.g. Ω^n) of exponentiation of a preorder.)

(3) For any functor F from K to K', we write F^n to denote the functor from K^n to K'^n such that:

(a) $F^n(X_1, \dots , X_n) = \langle F(X_1), \dots , F(X_n)\rangle$.

(b) $F^n(\rho_1, \dots , \rho_n) = \langle F(\rho_1), \dots , F(\rho_n)\rangle$.

(4) We write $\times^{(n)}$ to denote the functor from SET^n to SET such that:

(a) $\times^{(n)}(S_1, \dots , S_n) = S_1 \times \dots \times S_n$.

(b) $\times^{(n)}(f_1, \dots , f_n) = f_1 \times \dots \times f_n$.

Next, we note that when Ω^n and Ω are viewed as categories, the monotone function Γ_δ can be viewed as a functor from Ω^n to Ω by defining its action on morphisms to be $\Gamma_\delta(\omega_1 \le \omega'_1, \dots , \omega_n \le \omega'_n) = \Gamma_\delta(\omega_1, \dots , \omega_n) \le \Gamma_\delta(\omega'_1, \dots , \omega'_n)$.

Then

$$\Omega^n \xrightarrow{B^n} \mathrm{SET}^n \xrightarrow{\times^{(n)}} \mathrm{SET}$$

and

$$\Omega^n \xrightarrow{\Gamma_\delta} \Omega \xrightarrow{B} \mathrm{SET}$$

are compositions of functors which can be used to rewrite diagram (II) as:

$$\begin{tikzcd}
(B^n;\times^{(n)})(\omega_1, \dots ,\omega_n) \arrow[r, "\gamma_\delta(\omega_1, \dots ,\omega_n)"] \arrow[d, "(B^n;\times^{(n)})(\omega_1\leq\omega_1', \dots ,\omega_n\leq\omega_n')"'] & (\Gamma_\delta;B)(\omega_1, \dots ,\omega_n) \arrow[d, "(\Gamma_\delta;B)(\omega_1\leq\omega_1', \dots ,\omega_n\leq\omega_n')"] \\
(B^n;\times^{(n)})(\omega_1', \dots ,\omega_n') \arrow[r, "\gamma_\delta(\omega_1', \dots ,\omega_n')"] & (\Gamma_\delta;B)(\omega_1', \dots ,\omega_n') \ .
\end{tikzcd}$$

In this form, the diagram is clearly an assertion that γ_δ is a natural transformation from the functor $B^n;\times^{(n)}$ to the functor $\Gamma_\delta;B$.

At this stage we have come to regard Ω entirely as a category. Indeed, we can justify the term "category-sorted algebra" by extending our definition to the case where Ω is an arbitrary category:

(1) A signature consists of:

(1a) A category Ω of sorts.

(1b) A family, indexed by nonnegative integers, of disjoint sets Δ_n of operators of rank n.

(1c) For each $n \geq 0$ and $\delta \in \Delta_n$, a specification Γ_δ, which is a functor from Ω^n to Ω.

(2) An $\Omega\Delta\Gamma$-algebra consists of:

(2a) A carrier B, which is a functor from Ω to SET.

(2b) For each $n \geq 0$ and $\delta \in \Delta_n$, an interpretation γ_δ, which is a natural transformation from $B^n;\times^{(n)}$ to $\Gamma_\delta;B$.

(3) If B,γ and B',γ' are $\Omega\Delta\Gamma$-algebras, then an homomorphism from B,γ to B',γ' is a natural transformation from B to B' such that, for all $n \geq 0$, $\delta \in \Delta_n$, and $\omega_1, \dots , \omega_n \in \Omega$, the diagram (I) commutes.

This is a clear illustration of what we mean by applying category theory to language definition. Our intention is not to use any deep theorems of category theory, but merely to employ the basic concepts of this field as organizing principles. This might appear as a desire to be concise at the expense of being esoteric. But in designing a programming language, the central problem is to organize a variety of concepts in a way which exhibits uniformity and generality. Substantial leverage can be gained in attacking this problem if these concepts can be defined concisely within a framework which has already proven its ability to impose uniformity and generality upon a wide variety of mathematics.

It is easy to verify that $\Omega\Delta\Gamma$-algebras and their homomorphisms form a category, which we will call $ALG_{\Omega\Delta\Gamma}$. It is also evident that these category-sorted algebras reduce to the Higgins algebras (with total specifications) discussed earlier when Ω is a discrete category (i.e., a partially ordered set in which $\omega \leq \omega'$ only holds when $\omega = \omega'$.)

Algebraic Semantics

We can now explicate our claim that defining semantics is tantamount to defining a target algebra. Suppose the target algebra is a category-sorted $\Omega\Delta\Gamma$-algebra B,γ. Then $B(\omega)$ is the set of meanings of type ω. Thus we can define the set M of all meanings to be the disjoint union of $B(\omega)$ over $\omega \in |\Omega|$, i.e.,

$$M = \{\omega, x \mid \omega \in |\Omega| \text{ and } x \in B(\omega)\} \ .$$

We can also define the function $\tau_M \in M \to |\Omega|$ such that

$$\tau_M(\omega, x) = \omega \ ,$$

which gives the type of each meaning in M.

Now let I be a set of identifiers and $\tau_I \in I \to |\Omega|$ be an assignment of types to each identifier in I. Then an environment e for I,τ_I is a function from I to M which maps each identifier into a meaning of the appropriate type, i.e., which makes the diagram

$$\begin{tikzcd} I \arrow[rr, "e"] \arrow[dr, "\tau_I"'] & & M \arrow[dl, "\tau_M"] \\ & {|\Omega|} & \end{tikzcd}$$

of functions commute.

To describe this situation in category-theoretic terms,we define the category $SET\downarrow|\Omega|$ of sets with type assignments. This is the category such that

(a) The objects of $SET\downarrow|\Omega|$ are pairs S,τ, where S is a set and $\tau \varepsilon S \rightarrow |\Omega|$,

(b) $S,\tau \; {}_{SET\overset{\rightarrow}{\downarrow}|\Omega|} \; S',\tau'$ is the set of functions f from S to S' such that the diagram

$$\begin{tikzcd} S \arrow[rr, "f"] \arrow[dr, "\tau"'] & & S' \arrow[dl, "\tau'"] \\ & {|\Omega|} & \end{tikzcd}$$

commutes,

(c) Composition and identities in $SET\downarrow|\Omega|$ are the same as in SET.

Then an environment for I,τ_I is a morphism in $I,\tau_I \; {}_{SET\overset{\rightarrow}{\downarrow}|\Omega|} \; M,\tau_M$. We call this set $Env(I,\tau_I)$.

Next we define U to be the functor from $ALG_{\Omega\Delta\Gamma}$ to $SET\downarrow|\Omega|$ whose action on an $\Omega\Delta\Gamma$-algebra B,γ is given by

$U(B,\gamma) = S,\tau$ where

$S = \{\omega,x \mid \omega \;\varepsilon\; |\Omega| \text{ and } x \;\varepsilon\; B(\omega)\}$,

$\tau \;\varepsilon\; S \rightarrow |\Omega|$ is the function such that $\tau(\omega,x) = \omega$,

and whose action on an homomorphism θ from B,γ to B',γ' is given by

$U(\theta) \;\varepsilon\; U(B,\gamma) \; {}_{SET\overset{\rightarrow}{\downarrow}|\Omega|} \; U(B',\gamma')$ is the function such that

$U(\theta)(\omega,x) = \omega,\theta(\omega)(x)$.

Then M,τ_M is the result of applying U to the target algebra B,γ, so that $\mathrm{Env}(I,\tau_I) = I,\tau_I \xrightarrow[\mathrm{SET}\downarrow|\Omega|]{} U(B,\gamma)$. More generally, U is the "forgetful" functor which forgets both interpretations and implicit conversions, and maps a category-sorted algebra into the disjoint union of its carrier, along with an appropriate assignment of types to this disjoint union.

In the appendix, we will show that for any object I,τ_I of $\mathrm{SET}\downarrow|\Omega|$ there is an algebra $F(I,\tau_I)$, called the <u>free $\Omega\Delta\Gamma$-algebra generated by</u> I,τ_I, and a morphism $\eta(I,\tau_I) \in I,\tau_I \xrightarrow[\mathrm{SET}\downarrow|\Omega|]{} U(F(I,\tau_I))$, called the <u>embedding of</u> I,τ_I <u>into its free algebra</u>, such that:

> For any $B,\gamma \in |\mathrm{ALG}_{\Omega\Delta\Gamma}|$ and $e \in I,\tau_I \xrightarrow[\mathrm{SET}\downarrow|\Omega|]{} U(B,\gamma)$, there is exactly one homomorphism $\hat{e} \in F(I,\tau_I) \xrightarrow[\mathrm{ALG}_{\Omega\Delta\Gamma}]{} B,\gamma$ such that the diagram
>
> $$\begin{tikzcd} I,\tau_I \arrow[r, "\eta(I,\tau_I)"] \arrow[dr, "e"'] & U(F(I,\tau_I)) \arrow[d, "U(\hat{e})"] \\ & U(B,\gamma) \end{tikzcd}$$
>
> in $\mathrm{SET}\downarrow|\Omega|$ commutes.

Suppose $F(I,\tau_I) = B_0,\gamma_0$. Then each $B_0(\omega)$ is the set of phrases of type ω which can be constructed from identifiers in I whose types are given by τ_I. Each $\hat{e}(\omega)$ maps the phrases of type ω into their meanings in $B(\omega)$. Moreover, suppose $R,\tau_R = U(B_0,\gamma_0) = U(F(I,\tau_I))$. Then R is the set of phrases of all types, τ_R maps these phrases into their types, and $U(\hat{e})$ maps these phrases into their meanings in a way which preserves types.

The embedding $\eta(I,\tau_I)$ maps each identifier into the phrase which consists of that identifier. Thus the above diagram shows that the meaning $U(\hat{e})(\,\eta\,(I,\tau_I)(i))$ of the phrase consisting of i is the meaning e(i) given to i by the environment e.

For a given I,τ_I, one can define the $|\Omega|$-indexed family of semantic functions

$$\mu(\omega) \in B_0(\omega) \rightarrow (\mathrm{Env}(I,\tau_I) \rightarrow B(\omega))$$

such that

$$\mu(\omega)(r)(e) = \hat{e}(\omega)(r)\ .$$

Then each $\mu(\omega)$ maps phrases of type ω into functions from environments to meanings of type ω. Alternatively, one can define the single semantic function

$$\mu \in R \to (\mathrm{Env}(I,\tau_I) \to M)$$

such that

$$\mu(r)(e) = U(\hat{e})(r) \ .$$

This function maps phrases of all types into functions from environments to meanings.

It is evident that the linguistic application of category-sorted algebras depends crucially upon the existence of free algebras or, more abstractly, upon the existence of a left adjoint to the forgetful functor U. In general, if U is any functor from a category K' to a category K, F is a functor from K to K', and η is a natural transformation from I_K to F;U such that:

> For all $X \in |K|$, $X' \in |K'|$, and $\rho \in X \underset{K}{\to} U(X')$, there is exactly one morphism $\hat{\rho} \in F(X) \underset{K'}{\to} X'$ such that
>
> $$\begin{tikzcd} X \arrow[r, "\eta(X)"] \arrow[dr, "\rho"'] & U(F(X)) \arrow[d, "U(\hat{\rho})"] \\ & U(X') \end{tikzcd}$$
>
> commutes in K,

then F is said to be a left adjoint of U, with associated natural transformation η. The triple F, U, η is called an adjunction from K to K'.

In the appendix, we show the existence of free category-sorted algebras by constructing a left adjoint and associated natural transformation for the forgetful functor U from $\mathrm{ALG}_{\Omega\Delta\Gamma}$ to $\mathrm{SET}{\downarrow}|\Omega|$.

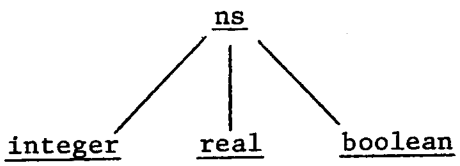
Data Algebras

To illustrate the application of category-sorted algebras, we will consider several variations of Algol 60. However, since we do not yet know how to treat binding mechanisms elegantly in an algebraic framework, we will limit ourselves to the subset of Algol which excludes the binding of identifiers, i.e., to the simple imperative language which underlies Algol. Although this is a substantial limitation, we will still be able to show the potential of our methodology for disciplining the design of implicit conversions and generic operators.

As discussed in (7) and (8), we believe that a fundamental characteristic of Algol-like languages is the presence of two kinds of type: data types, which describe variables (or expressions) and their ranges of values, and phrase types (called program types in (7)) which describe identifiers (or phrases which can be bound to identifiers) and their sets of meanings.

Algebraically, Ω should be a set of data types in order to define the values of expressions. In this case, the carrier of the free algebra is a data-type-indexed family of sets of expressions, and the carrier of the target algebra, which we will call a data algebra, is a data-type-indexed family of sets of values.

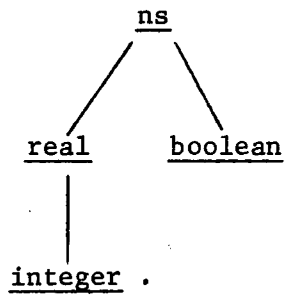
In Algol 60 itself there are three data types: integer, real, and boolean, to which we must add the nonsense type ns. To avoid implicit conversions, we would take Ω to be



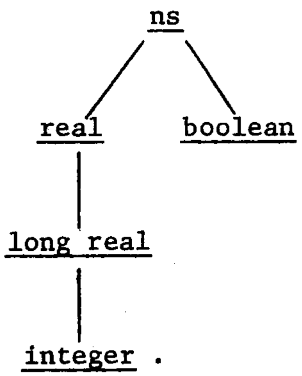
.

Notice that ns is the greatest element in this partial ordering, reflecting the notion that any sensible expression can occur in a context which permits nonsense.

On the other hand, to introduce an implicit conversion from integer to real, we would take integer to be a subtype of real:



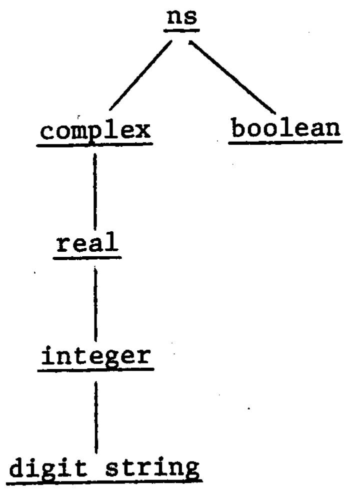
A more interesting situation arises when **long real** is introduced. One might expect **real** to be a subtype of **long real**, but an implicit conversion from **real** to **long real** would be dangerous from the viewpoint of numerical analysis, since a real value does not provide enough information to completely determine a long real value. In fact, it is the opposite implicit conversion which is numerically safe, so that **long real** should be a subtype of **real**:



In a language definition which was sufficiently concrete to make sense of the distinction between **real** and **long real**, one might take B(**real**) and B(**long real**) to be sets of real numbers with single and double precision representations, respectively, and B(**long real** ≤ **real**) to be the truncation or roundoff function from B(**long real**) to B(**real**). Notice that this function is not an injection, reflecting the fact that a conversion from **long real** to **real** loses information.

However, although this is suggestive, our methodology is not really adequate for dealing with the problems of roundoff or overflow. For this reason, we will omit the type **long real** and define our language at the level of abstraction where roundoff and overflow are ignored.

In the rest of this paper we will take Ω to be:



It should be emphasized that this choice of Ω - particularly the use of digit string - is purely for illustrative purposes, and is not put forth as desirable for a real programming language.

In the carrier of our target algebra we will have:

B(digit string) = the set of strings of digits,
B(integer) = the set of integers,
B(real) = the set of real numbers,
B(complex) = the set of complex numbers,
B(boolean) = {true, false},

with the conversion functions

B(digit string ≤ integer) = the function which maps each digit string into the integer of which it is a decimal representation.

B(integer ≤ real) = the identity injection from integers to real numbers.

B(real ≤ complex) = the identity injection from real numbers to complex numbers.

Notice that, because of the possible presence of leading zeros, the function B(digit string ≤ integer) is not an injection.

We must also specify B(ns) and the conversion functions into this set. For these conversion functions to exist, B(ns) must be nonempty, i.e., we must give some kind of meaning to nonsense expressions. The closest we can come to saying that they do not make sense is to give them all the same meaning by taking B(ns) to be a singleton set. This insures (since a singleton set is a terminal element in the category SET), that there will be exactly one possible conversion function from any data type to ns:

$B(\text{ns}) = \{<>\}$,

$B(\omega \leq \text{ns})$ = the unique function from $B(\omega)$ to $\{<>\}$.

As an example of an operator, let + be a member of Δ_2, with the specification

```
$\Gamma_+(\omega_1,\omega_2)$ = if $\omega_1 \leq$ integer and $\omega_2 \leq$ integer then integer
                 else if $\omega_1 \leq$ real and $\omega_2 \leq$ real then real
                 else if $\omega_1 \leq$ complex and $\omega_2 \leq$ complex then complex
                 else ns
```

and the interpretation

```
$\gamma_+(\omega_1,\omega_2)$ = if $\omega_1 \leq$ integer and $\omega_2 \leq$ integer then
                 $\lambda$(x,y). let x' = $B(\omega_1 \leq$integer)(x) and y' = $B(\omega_2 \leq$ integer)(y)
                       in integer-addition(x',y')
            else if $\omega_1 \leq$ real and $\omega_2 \leq$ real then
                 $\lambda$(x,y). let x' = $B(\omega_1 \leq$real)(x) and y' = $B(\omega_2 \leq$real)(y)
                       in real-addition(x',y')
            else if $\omega_1 \leq$ complex and $\omega_2 \leq$ complex then
                 $\lambda$(x,y). let x' = $B(\omega_1 \leq$complex)(x) and y' = $B(\omega_2 \leq$complex)(y)
                       in complex-addition(x',y')
            else $\lambda$(x,y). <> .
```

Although the above definition makes + a purely numerical operator, it can be extended to encompass nonnumerical"addition":

$$\begin{aligned}
\Gamma_+(\omega_1,\omega_2) = {} & \underline{\text{if}}\ \omega_1 \leq \underline{\text{boolean}}\ \underline{\text{and}}\ \omega_2 \leq \underline{\text{boolean}}\ \underline{\text{then}}\ \underline{\text{boolean}} \\
& \quad \underline{\text{else}}\ \underline{\text{if}}\ \omega_1 \leq \underline{\text{digit string}}\ \underline{\text{and}}\ \omega_2 \leq \underline{\text{digit string}}\ \underline{\text{then}}\ \underline{\text{digit string}} \\
& \quad \underline{\text{else}}\ \ldots\ \text{(as before)}
\end{aligned}$$

$$\begin{aligned}
\gamma_+(\omega_1,\omega_2) = {} & \underline{\text{if}}\ \omega_1 \leq \underline{\text{boolean}}\ \underline{\text{and}}\ \omega_2 \leq \underline{\text{boolean}}\ \underline{\text{then}} \\
& \qquad \lambda(x,y).\ \underline{\text{let}}\ x' = B(\omega_1 \leq \underline{\text{boolean}})(x)\ \underline{\text{and}}\ y' = B(\omega_2 \leq \underline{\text{boolean}})(y) \\
& \qquad\qquad \underline{\text{in}}\ \text{boolean-addition}(x',y') \\
& \quad \underline{\text{else}}\ \underline{\text{if}}\ \omega_1 \leq \underline{\text{digit string}}\ \underline{\text{and}}\ \omega_2 \leq \underline{\text{digit string}}\ \underline{\text{then}} \\
& \qquad \lambda(x,y).\ \underline{\text{let}}\ x' = B(\omega_1 \leq \underline{\text{digit string}})(x) \\
& \qquad\qquad\qquad \underline{\text{and}}\ y' = B(\omega_2 \leq \underline{\text{digit string}})(y) \\
& \qquad\qquad \underline{\text{in}}\ \text{digit-string-addition}(x',y') \\
& \quad \underline{\text{else}}\ \ldots\ \text{(as before)}\ .
\end{aligned}$$

Since there are no implicit conversions between boolean and any other type than ns, we are free to choose "boolean addition" to be any function from pairs of truth values to truth values. On the other hand, "digit-string addition" is tightly constrained by the implicit conversion from digit string to integer, which gives rise to the requirement that

$$\begin{tikzcd}
B(\underline{\text{digit string}}) \times B(\underline{\text{digit string}}) \arrow[r, "\text{digit-string addition}"] \arrow[d, "B(\underline{\text{digit string}} \leq \underline{\text{integer}}) \times B(\underline{\text{digit string}} \leq \underline{\text{integer}})"'] & B(\underline{\text{digit string}}) \arrow[d, "B(\underline{\text{digit string}} \leq \underline{\text{integer}})"'] \\
B(\underline{\text{integer}}) \times B(\underline{\text{integer}}) \arrow[r, "\text{integer addition}"] & B(\underline{\text{integer}})
\end{tikzcd}$$

commute. In other words, the sum of two digit strings must be a decimal representation of the sum of the integers which are represented by those two strings. The only freedom we have in defining digit-string addition is in the treatment of leading zeros in the result.

The definition of + suggests that a typical operator will have a significant specification and interpretation for certain "key" sorts of operands, and that its specification and interpretation for other sorts of operands can be obtained by implicitly converting the operands to key sorts. To formalize this idea, let

(1) Λ_δ be a category of <u>keys</u>.

(2) Φ_δ be a functor from Λ to Ω^n.

(3) $\overline{\Gamma}_\delta$ be a functor from Λ to Ω.

(4) $\overline{\gamma}_\delta$ be a natural transformation from $\Phi_\delta;B^n;\times^{(n)}$ to $\overline{\Gamma}_\delta;B$.

Intuitively, for each key $\lambda \varepsilon |\Lambda_\delta|$, $\Phi_\delta(\lambda)$ is the n-tuple of sorts to which the "λ-version" of δ is applicable, $\overline{\Gamma}_\delta(\lambda)$ is the sort of the result of the λ-version of δ, and $\overline{\gamma}_\delta(\lambda) \; \varepsilon \times^{(n)}(B^n(\Phi_\delta(\lambda))) \rightarrow B(\overline{\Gamma}_\delta(\lambda))$ is the interpretation of the λ-version of δ.

These entities can be extended to all sorts of operands if the functor Φ_δ possesses a left adjoint Ψ_δ, which will be a functor from Ω^n to Λ, and an associated natural transformation η_δ, which will be a natural transformation from I_{Ω^n} to $\Psi_\delta;\Phi_\delta$. Then we can define the specification

$$\Gamma_\delta = \Psi_\delta;\overline{\Gamma}_\delta \;\varepsilon\; \Omega^n \rightarrow \Omega \;,$$

and the interpretation

$$\gamma_\delta(\omega_1, \ldots , \omega_n) = \times^{(n)}(B^n(\eta_\delta(\omega_1, \ldots , \omega_n));\overline{\gamma}_\delta(\Psi_\delta(\omega_1, \ldots , \omega_n)) \;,$$

which can easily be shown to be a natural transformation from $B^n;\times^{(n)}$ to $\Gamma_\delta;B$. Intuitively, $\Psi_\delta(\omega_1, \ldots , \omega_n)$ can be thought of as the key determining the version of δ to be used for operands of sorts $\omega_1, \ldots , \omega_n$, and $\eta_\delta(\omega_1, \ldots , \omega_n)$ as the implicit conversion to be applied to these operands.

In the special case where Λ_δ and Ω are partially ordered sets, it can be shown (9, p. 93) that Ψ_δ will be a left adjoint of Φ_δ if and only if $\overline{\omega} \leq \Phi_\delta(\Psi_\delta(\overline{\omega}))$ for all $\overline{\omega} \;\varepsilon\; \Omega^n$ and $\Psi_\delta(\Phi_\delta(\lambda)) \leq \lambda$ for all $\lambda \;\varepsilon\; \Lambda_\delta$. In this case $\eta_\delta(\overline{\omega})$ will be the unique morphism $\overline{\omega} \leq \Phi_\delta(\Psi_\delta(\overline{\omega}))$, and γ_δ will be

$$\gamma_\delta(\overline{\omega}) = \times^{(n)}(B^n(\overline{\omega} \leq \Phi_\delta(\Psi_\delta(\overline{\omega}))));\overline{\gamma}_\delta(\Psi_\delta(\overline{\omega})) \;.$$

Moreover, as shown by the following proposition, Ψ_δ will be uniquely determined by Φ_δ:

Proposition Suppose Φ is a monotone function from Λ to Ω^n, where Λ and Ω^n are partially ordered sets, such that

(1) For all $\overline{\omega} \in \Omega^n$, the set $\{\lambda \mid \lambda \in \Lambda \text{ and } \overline{\omega} \leq \Phi(\lambda)\}$ has a greatest lower bound in Λ.

(2) For all $\overline{\omega} \in \Omega^n$, $\Phi(\sqcap_\Lambda \{\lambda \mid \lambda \in \Lambda \text{ and } \omega \leq \Phi(\lambda)\})$ is the greatest lower bound in Ω^n of $\{\Phi(\lambda) \mid \lambda \in \Lambda \text{ and } \overline{\omega} \leq \Phi(\lambda)\}$.

Then $\Psi(\overline{\omega}) = \sqcap_\Lambda \{\lambda \mid \lambda \in \Lambda \text{ and } \overline{\omega} \leq \Phi(\lambda)\}$ is the unique monotone function from Ω^n to Λ such that Ψ is a left adjoint of Φ.

Proof: Ψ is obviously monotone. For any $\lambda \in \Lambda$, $\Psi(\Phi(\lambda))$ is the greatest lower bound of $\{\lambda' \mid \lambda' \in \Lambda \text{ and } \Phi(\lambda) \leq \Phi(\lambda')\}$ and, since λ belongs to this set, $\Psi(\Phi(\lambda)) \leq \lambda$. For any $\overline{\omega} \in \Omega^n$, $\Phi(\Psi(\overline{\omega})) = \Phi(\sqcap_\Lambda \{\lambda \mid \lambda \in \Lambda \text{ and } \overline{\omega} \leq \Phi(\lambda)\})$ is the greatest lower bound of $\{\Phi(\lambda) \mid \lambda \in \Lambda \text{ and } \overline{\omega} \leq \Phi(\lambda)\}$ and, since $\overline{\omega}$ is a lower bound of this set, $\overline{\omega} \leq \Phi(\Psi(\overline{\omega}))$.

Suppose Ψ is a left adjoint of Φ. If $\overline{\omega} \leq \Phi(\lambda)$ then $\Psi(\overline{\omega}) \leq \Psi(\Phi(\lambda)) \leq \lambda$. Thus $\Psi(\overline{\omega})$ is a lower bound of $\{\lambda \mid \lambda \in \Lambda \text{ and } \overline{\omega} \leq \Phi(\lambda)\}$. Moreover, this set contains $\Psi(\overline{\omega})$ since $\overline{\omega} \leq \Phi(\Psi(\overline{\omega}))$. Thus any lower bound of this set must be less than $\Psi(\overline{\omega})$, so that $\Psi(\overline{\omega})$ is the greatest lower bound.

The conditions in this proposition will hold if Λ contains greatest lower bounds of all of its subsets, i.e., if Λ is a complete lattice, and Φ preserves all greatest lower bounds. However, we will sometimes use Λ's which are not complete lattices.

As an example, the purely numeric definition of + given earlier can be recast more concisely by using the set of keys

$$\Lambda_+ = \{\underline{\text{integer}}, \underline{\text{real}}, \underline{\text{complex}}, \underline{\text{ns}}\}$$

with the same partial ordering as Ω. Then the specification Γ_+ is determined by the functions Φ_+ and $\overline{\Gamma}_+$ such that

λ	$\Phi_+(\lambda)$	$\bar{\Gamma}_+(\lambda)$
integer	integer,integer	integer
real	real,real	real
complex	complex,complex	complex
ns	ns,ns	ns

and the interpretation γ_+ is determined by

$$\bar{\gamma}_+(\underline{\text{integer}}) = \text{integer addition}$$

$$\bar{\gamma}_+(\underline{\text{real}}) = \text{real addition}$$

$$\bar{\gamma}_+(\underline{\text{complex}}) = \text{complex addition}$$

$$\bar{\gamma}_+(\underline{\text{ns}}) = \lambda(x,y).\ \langle\rangle\ .$$

To extend this definition to nonnumeric types, one adds boolean and digit string to Λ_+, with

λ	$\Phi_+(\lambda)$	$\bar{\Gamma}_+(\lambda)$
boolean	boolean,boolean	boolean
digit string	digit string,digit string	digit string

and

$$\bar{\gamma}_+(\underline{\text{boolean}}) = \text{boolean addition}$$

$$\bar{\gamma}_+(\underline{\text{digit string}}) = \text{digit-string addition}\ .$$

(Notice that in this case Λ_+ is not a complete lattice, but the necessary conditions for the existence of a left adjoint to Φ_+ are still met.)

In the remainder of this section we will illustrate our approach by defining a few other binary operators. In each case Λ_+ is the listed subset of Ω, with the same partial ordering as Ω.

For the division operators / and ÷ we can define

λ	$\Phi_/(\lambda)$	$\bar{\Gamma}_/(\lambda)$
real	real,real	real
complex	complex,complex	complex
ns	ns,ns	ns

$\bar{\gamma}_/(\underline{\text{real}})$ = real division

$\bar{\gamma}_/(\underline{\text{complex}})$ = complex division

$\bar{\gamma}_/(\underline{\text{ns}}) = \lambda(x,y).\ \langle\rangle$

and

λ	$\Phi_\div(\lambda)$	$\bar{\Gamma}_\div(\lambda)$
integer	integer,integer	integer
ns	ns,ns	ns

$\bar{\gamma}_\div(\underline{\text{integer}}) = \lambda(x,y).$ the unique integer q such that
$x = q \times y + r$ where
$\underline{\text{if}}\ x \geq 0\ \underline{\text{then}}\ 0 \leq r < |y|\ \underline{\text{else}}\ -|y| < r \leq 0$,

$\bar{\gamma}_\div(\underline{\text{ns}}) = \lambda(x,y).\ \langle\rangle$.

These operations cannot be combined into a single operator since, for example, 3/2 = 1.5 but 3 ÷ 2 = 1. On the other hand, since the definition of $\gamma_\div(\underline{\text{integer}})$ extends sensibly to the case where x and y are real, one could generalize ÷ by taking $\Phi_\div(\underline{\text{integer}}) = \underline{\text{real}},\underline{\text{real}}$.

Since nonnegative integers have not been introduced as a data type and, for example, 3^{-2} is not an integer, exponentiation cannot be defined to yield an integer result for any sort of operands. If exponents are limited to integers, one can define

$\lambda_\uparrow$	$\Phi_\uparrow(\lambda)$	$\overline{\Gamma}_\uparrow(\lambda)$
real	real,integer	real
complex	complex,integer	complex
ns	ns,ns	ns

$$\overline{\gamma}_\uparrow(\underline{\text{real}}) = \lambda(x,n).\ x^n$$

$$\overline{\gamma}_\uparrow(\underline{\text{complex}}) = \lambda(x,\ n).\ x^n$$

$$\overline{\gamma}_\uparrow(\underline{\text{ns}}) = \lambda(x,y).\ \langle\rangle\ .$$

This can be extended to noninteger exponents by taking $\Phi_\uparrow(\underline{\text{complex}}) = \underline{\text{complex}},\underline{\text{complex}}$, but the multi-valued nature of complex exponentiation (as well as the time required to compute the necessary logarithms and exponentials) would probably make this unwise.

Finally, we define an equality operation:

λ	$\Phi_=(\lambda)$	$\overline{\Gamma}_=(\lambda)$
boolean	boolean,boolean	boolean
integer	integer,integer	boolean
real	real,real	boolean
complex	complex,complex	boolean
ns	ns,ns	ns

$$\begin{aligned}\overline{\gamma}_=(\lambda) = {} & \underline{\text{if}}\ \lambda \neq \underline{\text{ns}}\ \underline{\text{then}}\ \text{the equality relation for } B(\lambda)\\ & \underline{\text{else}}\ \lambda(x,y).\ \langle\rangle\ .\end{aligned}$$

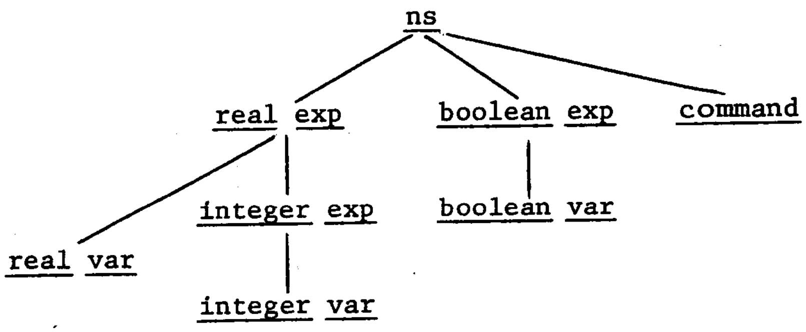
One might be tempted to add digit string to $\Lambda_=$, with $\Phi_=(\underline{\text{digit string}}) = \underline{\text{digit string}},\underline{\text{digit string}}$, $\overline{\Gamma}_=(\underline{\text{digit string}}) = \underline{\text{boolean}}$, and $\overline{\gamma}_=(\underline{\text{digit string}})$ = the equality relation for $B(\underline{\text{digit string}})$. However, the diagram

$$\begin{tikzcd}
B(\underline{\text{digit string}}) \times B(\underline{\text{digit string}}) \arrow[r, "=_{\text{digit string}}"] \arrow[d, "B(\underline{\text{digit string}} \leq \underline{\text{integer}}) \times B(\underline{\text{digit string}} \leq \underline{\text{integer}})"] & B(\underline{\text{boolean}}) \arrow[d, "I_{B(\underline{\text{boolean}})}"] \\
B(\underline{\text{integer}}) \times B(\underline{\text{integer}}) \arrow[r, "=_{\text{integer}}"] & B(\underline{\text{boolean}})
\end{tikzcd}$$

does not commute, since B(digit string $\leq$ integer) is not an injection. (For example, 6 and 06 are unequal digit strings which convert to equal integers.) Indeed, one can never use the same operator for the equality relation on different data types when the data types are connected by an implicit conversion function which is not an injection. (At the more concrete level where roundoff error is taken into account, this suggests, quite correctly, that there are special perils surrounding an equality operation for real numbers.)

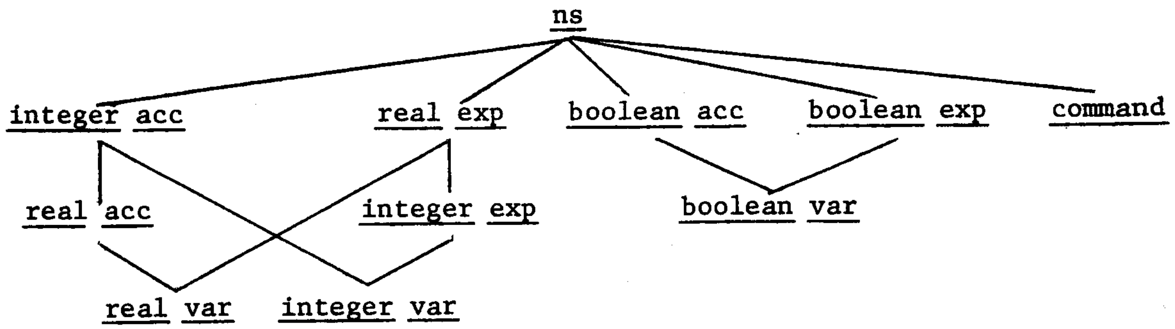
Algebras for Simple Imperative Languages

Now we move from data algebras, which describe languages of expressions, to algebras which describe simple imperative programming languages, i.e., languages with variables, expressions, and commands, but without binding operations. The sorts of our algebras will change from data types to phrase types, which can be thought of as phrase class names of the abstract syntax for the language being defined. For example, in place of the set of data types {integer, real, boolean}, Ω might be the following partially ordered set of phrase types:



It is evident that for each data type τ there will be two phrase types τ exp(ression) and τ var(iable), and that τ exp will be a subtype of τ' exp whenever the data type τ is a subtype of τ'. Moreover, τ var will be a subtype of τ exp since a variable can be used in any context which permits an expression of the same data type. On the other hand, the subtype relation will never hold between variables of distinct data types. For example, an integer variable cannot be used as a real variable since it cannot accept a noninteger value, and a real variable cannot be used as an integer variable since it might produce a noninteger value.

This kind of phrase-type structure, which describes many programming languages, is unpleasantly asymmetric. For each data type, there are variables, which can accept or produce values, and expressions, which can only produce values. Thus one might expect another kind of phrase, called an acceptor, which can only accept values. If acceptors for each data type are added to Ω, we have:



Notice that the subtype relation among acceptors is the dual of that for data types or expressions. For example, a real acceptor can be used as an integer acceptor since an integer value can be converted into a real value.

The above partial ordering has the peculiarity that there is a pair of phrase types, real var and integer var, which have no least upper bound. In general this might not be a problem, but we will find that there is one language construct, the general conditional phrase, which requires the existence of binary least upper bounds. To see the problem, suppose n is an integer variable and x is a real variable, and consider the conditional variable

if p then n else x .

In a context which calls for an expression, this phrase must be considered a real expression, since when p is false it can produce a noninteger value. But in a context which calls for an acceptor, the phrase must be considered an integer acceptor, since when p is true it cannot accept a noninteger value. The phrase type which describes this situation must be a subtype of both integer acc and real exp which in turn has real var and integer var as its subtypes. In other words, it must be the least upper bound of real var and integer var.

The way out of this difficulty is to characterize variables by both the data type which they accept and the data type which they produce. For example, a $\underline{\text{real}}$ $\underline{\text{var}}$ is actually a "real-accepting, real-producing" variable, an $\underline{\text{integer}}$ $\underline{\text{var}}$ is actually an "integer-accepting, integer-producing" variable, and the above conditional variable is an "integer-accepting, real-producing" variable. If we write $\tau_1\ \tau_2\ \underline{\text{var}}$ to abbreviate "τ_1-accepting, τ_2-producing" variable, then we have the ordering

$$\begin{tikzcd}
& \underline{\text{integer}}\ \underline{\text{acc}} \arrow[dl, no head] \arrow[dr, no head] & & \underline{\text{real}}\ \underline{\text{exp}} \arrow[dl, no head] \arrow[dr, no head] & \\
\underline{\text{real}}\ \underline{\text{acc}} \arrow[dr, no head] & & \underline{\text{integer}}\ \underline{\text{real}}\ \underline{\text{var}} \arrow[dl, no head] \arrow[dr, no head] & & \underline{\text{integer}}\ \underline{\text{exp}} \arrow[dl, no head] \\
& \underline{\text{real}}\ \underline{\text{real}}\ \underline{\text{var}} & & \underline{\text{integer}}\ \underline{\text{integer}}\ \underline{\text{var}}\quad . &
\end{tikzcd}$$

Implicit in this discussion is the idea that phrase types are constructed from data types. More generally, since the meaning of expressions can be described by a data algebra, and expressions are a major constituent of an imperative programming language, it should be possible to define the algebra describing the programming language in terms of the data algebra describing its expressions. To emphasize this possibility we will construct a programming-language algebra for an arbitrary data algebra, with signature Ω^D, Δ^D, Γ^D, carrier B^D, and interpretation γ^D. The main restrictions we will place upon this data algebra are that $\underline{\text{ns}}$ must be the greatest sort in Ω^D, and that $\Gamma^D(\omega_1, \ldots , \omega_n) = \underline{\text{ns}}$ must hold when any ω_i is $\underline{\text{ns}}$.

The set of phrase types is

$$\Omega = \{\tau\ \underline{\text{exp}} \mid \tau \in \Omega^D - \{\underline{\text{ns}}\}\} \cup \{\tau\ \underline{\text{acc}} \mid \tau \in \Omega^D - \{\underline{\text{ns}}\}\}$$
$$\cup \{\tau_1\tau_2\ \underline{\text{var}} \mid \tau_1, \tau_2 \in \Omega^D - \{\underline{\text{ns}}\}\} \cup \{\underline{\text{comm}}, \underline{\text{ns}}\}\ ,$$

with the least partial ordering such that

$$\text{if } \tau \leq^D \tau' \text{ then } \tau\ \underline{\text{exp}} \leq \tau'\ \underline{\text{exp}}$$
$$\text{if } \tau' \leq^D \tau \text{ then } \tau\ \underline{\text{acc}} \leq \tau'\ \underline{\text{acc}}$$
$$\text{if } \tau_1' \leq^D \tau_1 \text{ and } \tau_2 \leq^D \tau_2' \text{ then } \tau_1\tau_2\ \underline{\text{var}} \leq \tau_1'\tau_2'\ \underline{\text{var}}$$
$$\tau_1\tau_2\ \underline{\text{var}} \leq \tau_1\ \underline{\text{acc}}$$
$$\tau_1\tau_2\ \underline{\text{var}} \leq \tau_2\ \underline{\text{exp}}$$
$$\omega \leq \underline{\text{ns}}\quad .$$

Our target algebra describes direct semantics. (Continuation semantics can be treated in much the same way, but it leads to more complex definitions without providing any additional insights into the concerns of this paper.) The carrier of this target algebra will map each sort into a domain (a partially ordered set containing a least element $\perp$ and least upper bounds of its directed subsets), with implicit conversion functions which are strict and continuous (i.e., which preserve $\perp$ and least upper bounds of directed sets). Specifically, the following carrier is appropriate for direct semantics:

$$B(\tau\ \underline{exp}) = S \rightarrow [B^D(\tau)]_\perp$$

$$B(\underline{comm}) = S \rightarrow [S]_\perp$$

$$B(\tau\ \underline{acc}) = B^D(\tau) \rightarrow B(\underline{comm})$$

$$B(\tau_1\tau_2\ \underline{var}) = B(\tau_1\ \underline{acc}) \times B(\tau_2\ \underline{exp})$$

$$B(\underline{ns}) = \{\perp\}$$

$$B(\tau\ \underline{exp} \leq \tau'\ \underline{exp}) = \lambda v.\ v;[B^D(\tau \leq \tau')]_\perp$$

$$B(\tau\ \underline{acc} \leq \tau'\ \underline{acc}) = \lambda a.\ B^D(\tau' \leq \tau);\ a$$

$$B(\tau_1\tau_2\ \underline{var} \leq \tau_1\ \underline{acc}) = \lambda(a,\ v).\ a$$

$$B(\tau_1\tau_2\ \underline{var} \leq \tau_2\ \underline{exp}) = \lambda(a,\ v).\ v$$

$$B(\tau_1\tau_2\ \mathrm{var} \leq \tau_1'\tau_2'\ \underline{var}) = B(\tau_1\ \underline{acc} \leq \tau_1'\ \underline{acc}) \times B(\tau_2\ \underline{exp} \leq \tau_2'\ \underline{exp})$$

$$B(\omega \leq \underline{ns}) = \lambda x.\ \perp_{B(\underline{ns})}\ .$$

Here S is an unspecified set of store states. For any set X, $[X]_\perp$ denotes the flat domain obtained by adding $\perp$ to X. For any function $f \in X \rightarrow X'$, $[f]_\perp$ denotes the strict extension of f to $[X]_\perp \rightarrow [X']_\perp$.

Basically, the meaning of a command is a state transition function (with result $\perp$ for nontermination), the meaning of an acceptor is a function from data values to state transition functions, and the meaning of a variable is a pair giving both the meaning of an acceptor and of an expression. Notice that this way of defining variables avoids the mention of any entities such as Strachey's L-values. (As a consequence, our definition permits strangely behaved variables akin to the implicit references in GEDANKEN.(10))

Next we consider operators. Each operator of the data algebra becomes an expression-producing operator of the imperative-language algebra. If $\delta \in \Delta^D_n$, then $\delta \in \Delta_n$, with the specification given by:

$$\Lambda_\delta = (\Omega^D)^n$$

$$\Phi_\delta = \phi^n \text{ , where } \phi \in \Omega^D \to \Omega \text{ is the function such that}$$

$$\phi(\tau) = \underline{if}\ \tau = \underline{ns}\ \underline{then}\ \underline{ns}\ \underline{else}\ \tau\ \underline{exp} \text{ ,}$$

$$\overline{\Gamma}_\delta = \Gamma^D_\delta;\phi \text{ .}$$

To define the interpretation of δ we must give a natural transformation $\overline{\gamma}_\delta$ from $\Phi_\delta;B^n;\times^{(n)} = \phi^n;B^n;\times^{(n)}$ to $\overline{\Gamma}_\delta;B = \Gamma^D_\delta;\phi;B$. Thus $\overline{\gamma}_\delta(\tau_1, \dots , \tau_n)$ must be a function from $B(\phi(\tau_1)) \times \dots \times B(\phi(\tau_n))$ to $B(\phi(\Gamma^D_\delta(\tau_1, \dots , \tau_n)))$. If $\Gamma^D_\delta(\tau_1, \dots , \tau_n)$ is $\underline{ns}$, then $\overline{\gamma}_\delta(\tau_1, \dots , \tau_n)$ will be the unique function from $B(\phi(\tau_1)) \times \dots \times B(\phi(\tau_n))$ to $B(\underline{ns})$. Otherwise, none of the τ_i will be $\underline{ns}$, and $\overline{\gamma}_\delta(\tau_1, \dots , \tau_n)$ will be the function from $B(\tau_1\ \underline{exp}) \times \dots \times B(\tau_n\ \underline{exp})$ $= (S \to [B^D(\tau_1)]_\perp) \times \dots \times (S \to [B^D(\tau_n)]_\perp)$ to $B(\Gamma^D_\delta(\tau_1, \dots , \tau_n)\ \underline{exp})$ $= S \to [B^D(\Gamma^D_\delta(\tau_1, \dots , \tau_n))]_\perp$ such that

$$\overline{\gamma}_\delta(\tau_1, \dots , \tau_n)(v_1, \dots , v_n)$$
$$= \lambda\sigma \in S.\ [\gamma^D_\delta(\tau_1, \dots ,\tau_n)]_{\perp\perp}(v_1(\sigma), \dots , v_n(\sigma)) \text{ ,}$$

where $[\gamma^D_\delta(\tau_1, \dots ,\tau_n)]_{\perp\perp}$ denotes the extension of $\gamma^D_\delta(\tau_1, \dots , \tau_n)$ such that $[\gamma^D_\delta(\tau_1, \dots , \tau_n)]_{\perp\perp}(x_1, \dots , x_n) = \perp$ if any $x_i = \perp$.

Assignment is an operator $:= \in \Delta_2$. This is the one case which we cannot define by using an adjunction from a set of keys. The specification is

$$\Gamma_{:=}(\omega_1,\omega_2) = \underline{if}\ (\exists\tau \in \Omega^D - \{\underline{ns}\})\ \omega_1 \le \tau\ \underline{acc}\ \underline{and}\ \omega_2 \le \tau\ \underline{exp}$$
$$\underline{then}\ \underline{comm}\ \underline{else}\ \underline{ns}$$

If a data type τ meeting the above condition exists, then the interpretation is

$$\gamma_{:=}(\omega_1,\omega_2) = \lambda(a,v).\ \underline{let}\ a' = B(\omega_1 \le \tau\ \underline{acc})(a)\ \underline{and}\ v' = B(\omega_2 \le \tau\ \underline{exp})(v)$$
$$\underline{in}\ D_{comm}(v';[a']_\Phi) \text{ ;}$$

otherwise

$$\gamma_{:=}(\omega_1,\omega_2) = \lambda(a,v).\ \perp_{B(\underline{\text{ns}})} \quad .$$

Here $[a']_{\perp}$ is the $\perp$-preserving extension of a' from $B^D(\tau) \to B(\underline{\text{comm}})$ to $[B^D(\tau)]_{\perp} \to B(\underline{\text{comm}})$, and $D_{\text{comm}} \in (S \to B(\underline{\text{comm}})) \to B(\underline{\text{comm}})$ is the diagonalizing function such that

$$D_{\text{comm}}(h)(\sigma) = h(\sigma)(\sigma).$$

A subtlety in this definition is that the data type τ may not be unique. For example, if ω_1 is $\underline{\text{real}}\ \underline{\text{acc}}$ and ω_2 is $\underline{\text{integer}}\ \underline{\text{exp}}$ then τ can be either $\underline{\text{integer}}$ or $\underline{\text{real}}$. However, the definition still gives a unique meaning to $\gamma_{:=}$. Basically, this is because the structure of Ω insures that, if

$$\begin{tikzcd}
\tau\ \underline{\text{acc}} \arrow[dr, phantom, sloped, "\geq"] & & \tau'\ \underline{\text{acc}} \arrow[dl, phantom, sloped, "\leq"] & \text{and} & \tau\ \underline{\text{exp}} \arrow[dr, phantom, sloped, "\geq"] & & \tau'\ \underline{\text{exp}} \arrow[dl, phantom, sloped, "\leq"] \\
& \omega_1 & & & & \omega_2 &
\end{tikzcd} \quad ,$$

then there are data types τ_1 and τ_2 such that

$$\begin{tikzcd}
\tau\ \underline{\text{acc}} \arrow[dr, phantom, sloped, "\geq"] & & \tau'\ \underline{\text{acc}} \arrow[dl, phantom, sloped, "\leq"] & & \tau\ \underline{\text{exp}} \arrow[dr, phantom, sloped, "\geq"] & & \tau'\ \underline{\text{exp}} \arrow[dl, phantom, sloped, "\leq"] \\
& \tau_1\ \underline{\text{acc}} \arrow[d, phantom, sloped, "\geq"] & & \text{and} & & \tau_2\ \underline{\text{exp}} \arrow[d, phantom, sloped, "\geq"] & \\
& \omega_1 & & & & \omega_2 &
\end{tikzcd} \quad .$$

Then the definition of B for the implicit conversion of acceptors and expressions implies that the diagram

$$\begin{tikzcd}
B(\omega_1) \times B(\omega_2) \arrow[d, "B(\omega_1 \leq \tau_1\underline{\text{acc}}) \times B(\omega_2 \leq \tau_2\underline{\text{exp}})"] & \\
B(\tau_1\underline{\text{acc}}) \times B(\tau_2\underline{\text{exp}}) \arrow[r, "B(\tau_1\underline{\text{acc}} \leq \tau\ \underline{\text{acc}}) \times B(\tau_2\underline{\text{exp}} \leq \tau\ \underline{\text{exp}})"] \arrow[d, "B(\tau_1\underline{\text{acc}} \leq \tau'\underline{\text{acc}}) \times B(\tau_2\underline{\text{exp}} \leq \tau'\underline{\text{exp}})"] & B(\tau\ \underline{\text{acc}}) \times B(\tau\ \underline{\text{exp}}) \arrow[d, "\lambda(a,v).\ D_{\text{comm}}(v;[a]_{\perp})"] \\
B(\tau'\underline{\text{acc}}) \times B(\tau'\underline{\text{exp}}) \arrow[r, "\lambda(a',v').\ D_{\text{comm}}(v';[a']_{\perp})"] & B(\underline{\text{comm}})
\end{tikzcd}$$

of functions commutes. A slight extension of this argument shows that $\gamma_{:=}$ is a natural transformation.

Next we consider conditional phrases. It is trivial to define a particular type of conditional phrase such as a conditional command, but the definition of a generic conditional, applicable to arbitrary phrase types, is more challenging. Obviously, $\underline{\text{boolean}}$ must be a data type, with $B^D(\underline{\text{boolean}}) = \{\underline{\text{true}},\underline{\text{false}}\}$. Less obviously, Ω must possess all binary least upper bounds. (Note that this imposes a restriction upon Ω^D.)

Under these conditions, we can define $\underline{\text{if}} \; \varepsilon \; \Delta_3$, with the specification

$$\Lambda_{\underline{\text{if}}} = \Omega$$

$\Phi_{\underline{\text{if}}} \; \varepsilon \; \Omega \to \Omega^3$ is the function such that

$$\Phi_{\underline{\text{if}}}(\omega) = \underline{\text{if}} \; \omega = \underline{\text{ns}} \; \underline{\text{then}} \; \langle\underline{\text{ns}},\underline{\text{ns}},\underline{\text{ns}}\rangle \; \underline{\text{else}} \; \langle\underline{\text{boolean}} \; \underline{\text{exp}},\omega,\omega\rangle$$

$$\overline{\Gamma}_{\underline{\text{if}}} = I_\Omega \; .$$

Then the left adjoint of $\Phi_{\underline{\text{if}}}$ is the function $\Psi_{\underline{\text{if}}} \; \varepsilon \; \Omega^3 \to \Omega$ such that

$$\Psi_{\underline{\text{if}}}(\omega_1,\omega_2,\omega_3) = \underline{\text{if}} \; \omega_1 \le \underline{\text{boolean}} \; \underline{\text{exp}} \; \underline{\text{then}} \; \omega_2 \sqcup \omega_3 \; \underline{\text{else}} \; \underline{\text{ns}} \; .$$

(From the proposition in the previous section, it can be shown that if there are ω_2, ω_3 in Ω which do not possess a least upper bound then Φ has no left adjoint.)

To determine the interpretation of $\underline{\text{if}}$, we must give a natural transformation $\overline{\gamma}_{\underline{\text{if}}}$ from $\Phi;B^3;\times^{(3)}$ to $\overline{\Gamma};B = B$. When $\omega = \underline{\text{ns}}$, $\overline{\gamma}_{\underline{\text{if}}}(\omega)$ is the unique function from $B(\underline{\text{ns}}) \times B(\underline{\text{ns}}) \times B(\underline{\text{ns}})$ to $B(\underline{\text{ns}})$. Otherwise it is the function from $B(\underline{\text{boolean}} \; \underline{\text{exp}}) \times B(\omega) \times B(\omega)$ to $B(\omega)$ such that

$$\overline{\gamma}_{\underline{\text{if}}}(\omega)(v,f,g) = D_\omega(v;[\lambda b \; \varepsilon \; \{\underline{\text{true}},\underline{\text{false}}\}. \; \underline{\text{if}} \; b \; \underline{\text{then}} \; f \; \underline{\text{else}} \; g]_\perp) \; ,$$

where D is the Ω-indexed family of diagonalizing functions, $D_\omega \; \varepsilon \; (S \to B(\omega)) \to B(\omega)$ such that

$$D_{\tau \; \text{exp}} = \lambda h \; \varepsilon \; S \to (S \to [B^D(\tau)]_\perp). \; \lambda\sigma \; \varepsilon \; S. \; h(\sigma)(\sigma)$$

$$D_{\text{comm}} = \lambda h \; \varepsilon \; S \to (S \to [S]_\perp). \; \lambda\sigma \; \varepsilon \; S. \; h(\sigma)(\sigma)$$

$$D_{\tau \; \text{acc}} = \lambda h \; \varepsilon \; S \to (B^D(\tau) \to (S \to [S]_\perp)). \; \lambda x \; \varepsilon \; B^D(\tau). \; \lambda\sigma \; \varepsilon \; S. \; h(\sigma)(x)(\sigma)$$

$$D_{\tau_1\tau_2 \; \text{var}} = \lambda h \; \varepsilon \; S \to B(\tau_1 \; \underline{\text{acc}}) \times B(\tau_2 \; \underline{\text{exp}}). \; \langle D_{\tau_1 \; \text{acc}}(h;(\lambda(a,v).a)), \; D_{\tau_2 \; \text{exp}}(h;(\lambda(a,v).v))\rangle$$

$$D_{\text{ns}} = \lambda h \; \varepsilon \; S \to B(\underline{\text{ns}}). \; \perp_{B(\underline{\text{ns}})} \; .$$

(Notice that D_{comm} also occurred in the definition of assignment.) This family has the property that, for all $\omega,\omega' \in \Omega$ such that $\omega \leq \omega'$ and all $h \in S \to B(\omega)$,

$$B(\omega \leq \omega')(D_{\omega}(h)) = D_{\omega'}(h;B(\omega \leq \omega')) \ .$$

It is this property that insures that $\overline{\gamma}_{\underline{\text{if}}}$ is a natural transformation.

Finally, for completeness, we define operators for statement sequencing and a while statement. Since these operators are not generic, their definition is straightforward:

$; \in \Delta_2$, $\underline{\text{while}} \in \Delta_2$

$\Lambda_{;} = \Lambda_{\underline{\text{while}}} = \{\underline{\text{comm}},\ \underline{\text{ns}}\}$ with the same partial ordering as Ω.

$\Phi_{;}(\underline{\text{comm}}) = \langle\underline{\text{comm}},\underline{\text{comm}}\rangle$, $\Phi_{\underline{\text{while}}}(\underline{\text{comm}}) = \langle\underline{\text{boolean}}\ \underline{\text{exp}},\underline{\text{comm}}\rangle$

$\Phi_{;}(\underline{\text{ns}}) = \Phi_{\underline{\text{while}}}(\underline{\text{ns}}) = \langle\underline{\text{ns}},\underline{\text{ns}}\rangle$

$\overline{\Gamma}_{;}(\underline{\text{comm}}) = \overline{\Gamma}_{\underline{\text{while}}}(\underline{\text{comm}}) = \underline{\text{comm}}$

$\overline{\Gamma}_{;}(\underline{\text{ns}}) = \overline{\Gamma}_{\underline{\text{while}}}(\underline{\text{ns}}) = \underline{\text{ns}}$

$\overline{\gamma}_{;}(\underline{\text{ns}}) = \overline{\gamma}_{\underline{\text{while}}}(\underline{\text{ns}})$ is the unique function from $B(\underline{\text{ns}}) \times B(\underline{\text{ns}})$ to $B(\underline{\text{ns}})$.

$\overline{\gamma}_{;}(\underline{\text{comm}}) = \lambda(c_1 \in S \to [S]_{\perp},\ c_2 \in S \to [S]_{\perp}).\ c_1;[c_2]_{\obot}$

$\overline{\gamma}_{\underline{\text{while}}}(\underline{\text{comm}}) = \lambda(v \in S \to [\{\underline{\text{true}},\underline{\text{false}}\}]_{\perp},\ c_1 \in S \to [S]_{\perp}).$

$$Y(\lambda c_2 \in S \to [S]_{\perp}.\ D_{\text{comm}}(v;[\lambda b.\ \underline{\text{if}}\ b\ \underline{\text{then}}\ (c_1;[c_2]_{\obot})\ \underline{\text{else}}\ J]_{\obot}) \ .$$

Here J is the identity injection from S to $[S]_{\perp}$ and Y is the least-fixed-point operator for the domain $S \to [S]_{\perp}$.

Future Directions

The approach described in this paper is still far from being able to encompass a full-blown programming language. In particular, the following areas need investigation:

(1) Binding mechanisms, i.e. declarations and procedures.

(2) Products of types, i.e. records or class elements.

(3) Sums of types, i.e. disjoint unions.

(4) Type definitions, including recursive type definitions.

(5) Syntactic control of interference.(7)

In the first three of these areas, our ideas have progressed far enough to suggest the form of the partially ordered set of phrase types. One wants a set Ω satisfying

$$\Omega = \Omega_{\text{primitive}} + \Omega_{\text{procedure}} + \Omega_{\text{product}} + \Omega_{\text{sum}} .$$

Here + denotes some kind of sum of partially ordered sets. (At present, it is not clear how this sum should treat the greatest type <u>ns</u> or a possible least type.) The partially ordered set $\Omega_{\text{primitive}}$ is similar to the Ω described in the previous section, and

$$\Omega_{\text{procedure}} = \{\omega_1 \rightarrow \omega_2 \mid \omega_1, \omega_2 \in \Omega\}$$

$$\Omega_{\text{product}} = \{\underline{\text{product}}(\omega_1, \ldots , \omega_n) \mid n \geq 0 \text{ and } \omega_1, \ldots ,\omega_n \in \Omega\}$$

$$\Omega_{\text{sum}} = \{\underline{\text{sum}}(\omega_1, \ldots , \omega_n) \mid n \geq 0 \text{ and } \omega_1, \ldots ,\omega_n\}$$

The main novelty is the partial ordering of $\Omega_{\text{procedure}}$. One wants procedure types to satisfy

$$(\omega_1 \rightarrow \omega_2) \leq (\omega_1' \rightarrow \omega_2') \text{ if and only if } \omega_1' \leq \omega_1 \text{ and } \omega_2 \leq \omega_2' ,$$

so that the type operator → is antimonotone in its first argument. For example, suppose <u>integer</u> <u>exp</u> ≤ <u>real</u> <u>exp</u>. Then a procedure of type <u>real</u> <u>exp</u> → <u>boolean</u> <u>exp</u>, which can accept any real expression as argument, can also accept any integer expression as argument, and should therefore be permissible in any context which permits a procedure of type <u>integer</u> <u>exp</u> → <u>boolean</u> <u>exp</u>. Thus (<u>real</u> <u>exp</u> → <u>boolean</u> <u>exp</u>) ≤ (<u>integer</u> <u>exp</u> → <u>boolean</u> <u>exp</u>).

It follows that $\Omega_{\text{procedure}}$ will be isomorphic to $\Omega^{\text{op}} \times \Omega$, where Ω^{op} denotes the dual of Ω. This raises the question of how one solves the recursive equation describing Ω. The simplest answer is to impose an appropriate ordering on the least set satisfying this equation. The resulting Ω, however, will not contain certain limits which will be needed to deal with recursive type definitions. One would like to use Scott's methods to treat recursive definitions, but these methods do not encompass the operation of dualizing a partial ordering.

This difficulty does not arise for products or sums, where conventional pointwise ordering seems natural. However, a richer ordering becomes attractive when named, rather than numbered, products and sums are considered. Suppose we redefine

$$\Omega_{\text{product}} = \{\underline{\text{product}}(\overline{\omega}) \mid \overline{\omega} \in N \to \Omega \text{ for some finite set } N \text{ of names}\} ,$$

and similarly for Ω_{sum}. Then the following ordering can be used:

$\underline{\text{product}}(\overline{\omega}) \leq \underline{\text{product}}(\overline{\omega}')$ whenever

$$\text{domain}(\overline{\omega}) \supseteq \text{domain}(\overline{\omega}') \text{ and } (\forall\, n \in \text{domain}(\overline{\omega}'))\ \overline{\omega}(n) \leq \overline{\omega}'(n),$$

$\underline{\text{sum}}(\overline{\omega}) \leq \underline{\text{sum}}(\overline{\omega}')$ whenever

$$\text{domain}(\overline{\omega}) \subseteq \text{domain}(\overline{\omega}') \text{ and } (\forall\, n \in \text{domain}(\overline{\omega}))\ \overline{\omega}(n) \leq \overline{\omega}'(n).$$

The first ordering permits implicit record conversions which forget fields. The second ordering permits implicit conversions of disjoint unions which broaden the number of alternatives in a union.

In particular, the second ordering solves a long-standing problem in the type-checking of disjoint union expressions. Suppose p is a phrase of type ω, and make-n denotes the injection into a disjoint union corresponding to the alternative named n. Using bottom-up type analysis, how does one determine the type of make-n(p)? The answer is that the type is sum(n:ω), which is a subtype of any sum of the form sum(... , n:ω, ...).

APPENDIX

In this appendix we will demonstrate the existence of free category-sorted algebras by constructing an appropriate adjunction. Our basic approach will be to connect category-sorted algebras with ordinary one-sorted algebras in order to use the known existence of free ordinary algebras. We begin by stating several general properties of adjunctions which will be used in our development.

Proposition Suppose U is a functor from K' to K, F is a function from $|K|$ to $|K'|$, and η is a $|K|$-indexed family of morphisms $\eta(X) \in X \xrightarrow[K]{} U(F(X))$ such that:

For all $X \in |K|$, $X' \in |K'|$, and $\rho \in X \xrightarrow[K]{} U(X')$ there is exactly one morphism $\hat{\rho} \in F(X) \xrightarrow[K']{} X'$ such that

$$\begin{tikzcd}
X \arrow[r, "\eta(X)"] \arrow[dr, "\rho"'] & U(F(X)) \arrow[d, "U(\hat{\rho})"] \\
& U(X')
\end{tikzcd}$$

commutes in K.

Then there is exactly one way of extending F to be a functor from K to K' such that F is the left adjoint of U with η as the associated natural transformation. Namely, for each $\theta \in X \xrightarrow[K]{} X'$, $F(\theta)$ must be the unique morphism such that

$$\begin{tikzcd}
X \arrow[r, "\eta(X)"] \arrow[d, "\theta"] & U(F(X)) \arrow[d, "U(F(\theta))"] \\
X' \arrow[r, "\eta(X')"] & U(F(X'))
\end{tikzcd}$$

commutes in K.

We omit the proof (11,p. 116), the main point of which is to show that the extension of F preserves composition and identities. The utility of this proposition is that, in specifying adjunctions it is only necessary to specify the object part of the left adjoint.

Next, we consider the composition of adjunctions:

Proposition Suppose U is a functor from K' to K with left adjoint F and associated natural transformation η, and U' is a functor from K'' to K' with left adjoint F' and associated natural transformation η'. Let

$$\begin{aligned} U'' &= U';U \\ F'' &= F;F' \\ \eta''(X) &= \eta(X) ;_{K} U(\eta'(F(X))) \ . \end{aligned}$$

Then U'' is a functor from K'' to K with left adjoint F'' and associated natural transformation η''.

Again we omit the proof (9, p. 101).

Finally, we introduce the construction of categories over distinguished objects, and show that an adjunction between such categories can be built out of an adjunction between the categories from which they have been constructed.

Let K be a category and $T \in |K|$. Then $K{\downarrow}T$, called the category of objects over T, is the category such that

(a) $|K{\downarrow}T| = \{X, \tau \mid X \in |K| \text{ and } \tau \in X \underset{K}{\rightarrow} T\}$,

(b) $X,\tau \underset{K\downarrow T}{\rightarrow} X',\tau'$ is the set of morphisms $\rho \in X \underset{K}{\rightarrow} X'$ such that

$$\begin{tikzcd} X \arrow[rr, "\rho"] \arrow[dr, "\tau"'] & & X' \arrow[dl, "\tau'"] \\ & T & \end{tikzcd}$$

commutes in K.

(c) Composition and identities are the same as in K.

Then:

<u>Proposition</u> Suppose U is a functor from K' to K with left adjoint F and associated natural transformation η. Suppose $T' \varepsilon |K'|$and $T = U(T')$. Let $\overline{U}$ be the functor from $K'{\downarrow}T'$ to $K{\downarrow}T$ such that $\overline{U}(X',\tau') = U(X'),U(\tau')$ and $\overline{U}(\rho) = U(\rho)$. Then $\overline{U}$ has a left adjoint $\overline{F}$ and an associated natural transformation $\overline{\eta}$ such that

$$\overline{F}(X,\tau) = F(X),\hat{\tau}$$

$$\overline{\eta}(X,\tau) = \eta(X) \ ,$$

where $\hat{\tau} \ \varepsilon \ F(X) \underset{K'}{\rightarrow} T'$ is the unique morphism such that

$$\begin{tikzcd}
X \arrow[r, "\eta(X)"] \arrow[dr, "\tau"'] & U(F(X)) \arrow[d, "U(\hat{\tau})"] \\
& U(T') = T
\end{tikzcd}$$

commutes in K.

Proof: We leave it to the reader to verify that $\overline{U}$ is a functor from $K'{\downarrow}T'$ to $K{\downarrow}T$, that $\overline{F}$ is (the object part of) a functor from $K{\downarrow}T$ to $K'{\downarrow}T'$, and that $\overline{\eta}(X,\tau) \ \varepsilon \ X,\tau \underset{K\downarrow T}{\rightarrow} \overline{U}(\overline{F}(X,\tau))$. To show the adjunction property, suppose $X,\tau \varepsilon |K{\downarrow}T|$, $X',\tau' \varepsilon \ |K'{\downarrow}T'|$,and $\rho \ \varepsilon \ X,\tau \underset{K\downarrow T}{\rightarrow} \overline{U}(X',\tau')$. Then we must show that there is exactly one $\hat{\rho} \ \varepsilon \ \overline{F}(X,\tau) \underset{K'\downarrow T'}{\rightarrow} X',\tau'$ such that

$$\begin{tikzcd}
X,\tau \arrow[r, "\overline{\eta}(X,\tau) = \eta(X)"] \arrow[dr, "\rho"'] & \overline{U}(\overline{F}(X,\tau)) = U(F(X)),U(\hat{\tau}) \arrow[d, "\overline{U}(\hat{\rho}) = U(\hat{\rho})"] \\
& \overline{U}(X',\tau') = U(X'),U(\tau')
\end{tikzcd}$$

commutes in $K{\downarrow}T$.

Since composition is the same in $K{\downarrow}T$ as in K, $\hat{\rho}$ can only be the unique morphism in $F(X) \underset{K'}{\rightarrow} X'$ such that

$$\begin{tikzcd}
X \arrow[r, "\eta(X)"] \arrow[dr, "\rho"'] & U(F(X)) \arrow[d, "U(\hat{\rho})"] \\
& U(X')
\end{tikzcd}$$

commutes in K.

However, we must show that $\hat{\rho}$ actually belongs to the more restricted set of morphisms $\overline{F}(X,\tau) \underset{K'\downarrow T'}{\rightarrow} X',\tau'$. To establish this, we note that $\rho \in X,\tau \underset{K\downarrow T}{\rightarrow} \overline{U}(X',\tau') = X,\tau \underset{K\downarrow T}{\rightarrow} U(X'),U(\tau')$ implies that

$$\begin{tikzcd}
X \arrow[rr, "\rho"] \arrow[dr, "\tau"'] & & U(X') \arrow[dl, "U(\tau')"] \\
& T &
\end{tikzcd}$$

commutes in K, which in conjunction with the previous diagram implies that

$$\begin{tikzcd}
X \arrow[rr, "\eta(X)"] \arrow[dr, "\tau"'] & & U(F(X)) \arrow[dl, "U(\hat{\rho});\ U(\tau') = U(\hat{\rho};\tau')"] \\
& T &
\end{tikzcd}$$

commutes in K. Then the uniqueness of $\hat{\tau}$ gives $\hat{\rho};\tau' = \hat{\tau}$, so that $\hat{\rho} \in F(X),\hat{\tau} \underset{K'\downarrow T'}{\rightarrow} X',\tau' = \overline{F}(X,\tau) \underset{K'\downarrow T'}{\rightarrow} X',\tau'$.

Now we can apply these general results to the specific case of interest. Let ΩΔΓ be a fixed but arbitrary category-sorted signature, let CALG (called $\mathrm{ALG}_{\Omega\Delta\Gamma}$ in the main text) be the category of ΩΔΓ-algebras and their homomorphisms, and let ALG be the category of Δ-algebras and their homomorphisms:

(1) A Δ-algebra consists of:

(1a) A carrier R, which is a set.

(1b) For each $n \geq 0$ and $\delta \in \Delta_n$, an interpretation $\sigma_\delta \in R^n \rightarrow R$.

(2) If R,σ and R',σ' are Δ-algebras, then a homomorphism from R,σ to R',σ' is a function $h \in R \rightarrow R'$ such that, for all $n \geq 0$ and $\delta \in \Delta_n$, the diagram

$$\begin{tikzcd}
R^n \arrow[r, "\sigma_\delta"] \arrow[d, "h^n"] & R \arrow[d, "h"] \\
R'^n \arrow[r, "\sigma'_\delta"] & R'
\end{tikzcd}$$

of functions commutes.

The known existence of ordinary free algebras can be stated in the language of adjunctions by:

> Let U_A be the functor from ALG to SET which maps algebras into their carriers and homomorphisms into themselves. Then U_A possesses a left adjoint F_A with an associated natural transformation η_A.

Here $F_A(S)$ is the free Δ-algebra generated by S, and $\eta_A(S)$ is the embedding of S into the carrier of $F_A(S)$.

Of particular importance is the Δ-algebra, which we will call T, in which the carrier members are sorts and the interpretation of each operator is its category-sorted specification. More precisely, T is the Δ-algebra $|\Omega|$, Γ_{ob}, where each $\Gamma_{ob,\delta}$ is the object part of the functor Γ_δ.

We now introduce the categories $\mathrm{ALG}{\downarrow}T$ and $\mathrm{SET}{\downarrow}|\Omega|$. An object of $\mathrm{ALG}{\downarrow}T$ can be thought of as a Δ-algebra equipped with an assignment of sorts to the members of its carrier. Similarly, an object of $\mathrm{SET}{\downarrow}|\Omega|$ can be thought of as a set equipped with an assignment of sorts to its members. Since $|\Omega| = U_A(T)$, our last general proposition gives:

> Let U_T be the functor from $\mathrm{ALG}{\downarrow}T$ to $\mathrm{SET}{\downarrow}|\Omega|$ such that $U_T(\langle R,\sigma\rangle,\tau) = U_A(R,\sigma),U_A(\tau) = R,\tau$, and $U_T(h) = U_A(h) = h$. Then U_T has a left adjoint F_T and an associated natural transformation η_T such that
>
> $$F_T(S,\tau) = F_A(S),\hat{\tau}$$
> $$\eta_T(S,\tau) = \eta_A(S) \ ,$$
>
> where $\hat{\tau} \ \epsilon \ F_A(S) \underset{ALG}{\to} T$ is the unique morphism such that
>
> $$\begin{tikzcd} S \arrow[r, "\eta_A(S)"] \arrow[dr, "\tau"'] & U_A(F_A(S)) \arrow[d, "U_A(\hat{\tau})"] \\ & T \end{tikzcd}$$
>
> commutes in SET.

Informally, a type assignment to a set can be extended to the free Δ-algebra generated by that set by using the specification Γ to interpret the operators in Δ.

Our final (and most complicated) task is to construct an adjunction from ALG↓T to CALG. Let U_C be a functor from CALG to ALG↓T whose action on objects is given by:

$$\begin{aligned}
&U_C(B',\gamma') = \langle R',\sigma'\rangle,\tau' \quad \text{where} \\
&\qquad R' = \{\omega,x' \mid \omega \in |\Omega| \text{ and } x' \in B'(\omega)\} \,, \\
&\qquad \sigma'_\delta \in R'^n \rightarrow R' \text{ is the function such that} \\
&\qquad\qquad \sigma'_\delta(\langle\omega_1,x'_1\rangle, \ldots , \langle\omega_n,x'_n\rangle) = \\
&\qquad\qquad\qquad \Gamma_\delta(\omega_1, \ldots , \omega_n),\ \gamma'_\delta(\omega_1, \ldots , \omega_n)(x'_1, \ldots , x'_n) \,, \\
&\qquad \tau' \in R' \rightarrow |\Omega| \text{ is the function such that } \tau'(\omega,x') = \omega \,.
\end{aligned}\tag{1}$$

(The variables in this definition have been primed to facilitate its application to later developments.) The reader may verify that τ' is an homomorphism from R',σ' to T, so that $\langle R',\sigma'\rangle,\tau'$ is an object of ALG↓T. Intuitively, the action of U_C on objects is to forget the morphism part of B' (i.e., the implicit conversion functions) and to collapse the object part of B' into a disjoint union R' of its components, with a type assignment τ' which remembers which component of B' was the source of each member of R'.

To specify the action of U_C on morphisms, suppose $\theta \in B,\gamma \xrightarrow[\text{CALG}]{} B',\gamma'$, and let $\langle R,\sigma\rangle,\tau = U_C(B,\gamma)$ and $\langle R',\sigma'\rangle,\tau' = U_C(B',\gamma')$. Then

$$\begin{aligned}
&U_C(\theta) \in R \rightarrow R' \text{ is the function such that} \\
&\qquad U_C(\theta)(\omega,x) = \omega,\theta(\omega)(x) \,.
\end{aligned}$$

The reader may verify that $U_C(\theta)$ is an homomorphism from R,σ to R',σ' (which depends upon the fact that θ is an homomorphism from B,γ to B',γ'), that

$$\begin{tikzcd}
R,\sigma \arrow[rr, "U_C(\theta)"] \arrow[dr, "\tau"'] & & R',\sigma' \arrow[dl, "\tau'"] \\
& T &
\end{tikzcd}$$

commutes in ALG, so that $U_C(\theta) \in \langle R,\sigma\rangle,\tau \xrightarrow[\text{ALG}\downarrow T]{} \langle R',\sigma'\rangle,\tau'$, and that U_C preserves composition and identities.

Next, let F_C be the functor from ALG↓T to CALG such that

$$
\begin{aligned}
&F_C(\langle R,\sigma\rangle,\tau) = B,\gamma \quad \text{where} \\
&\qquad B(\omega) = \{r,\iota \mid r \in R \text{ and } \iota \in \tau(r) \underset{\Omega}{\to} \omega\} \ , \\
&\qquad B(\rho \in \omega \underset{\Omega}{\to} \omega') \in B(\omega) \to B(\omega') \text{ is the function such that} \\
&\qquad\qquad B(\rho)(r,\iota) = r,(\iota;_{\Omega}\rho) \ , \\
&\qquad \gamma_\delta(\omega_1, \ldots , \omega_n) \in B(\omega_1) \times \ldots \times B(\omega_n) \to B(\Gamma_\delta(\omega_1, \ldots , \omega_n)) \\
&\qquad\qquad \text{is the function such that} \\
&\qquad\qquad \gamma_\delta(\omega_1, \ldots , \omega_n)(\langle r_1,\iota_1\rangle, \ldots , \langle r_n,\iota_n\rangle) = \\
&\qquad\qquad\qquad\qquad \sigma_\delta(r_1, \ldots , r_n),\Gamma_\delta(\iota_1, \ldots , \iota_n) \ .
\end{aligned}
\tag{2}
$$

To see that $\gamma_\delta(\omega_1, \ldots , \omega_n)$ is a function of the correct type, suppose that, for $1 \le i \le n$, $\langle r_i,\iota_i\rangle \in B(\omega_i)$. Then each $\iota_i \in \tau(r_i) \underset{\Omega}{\to} \omega$. Thus $\Gamma_\delta(\iota_1, \ldots , \iota_n) \in \Gamma_\delta(\tau(r_1), \ldots , \tau(r_n)) \underset{\Omega}{\to} \Gamma_\delta(\omega_1, \ldots , \omega_n)$. But since τ is an homomorphism from R,σ to $T = |\Omega|,\Gamma_{ob}$, this set is also $\tau(\sigma_\delta(r_1, \ldots , r_n)) \underset{\Omega}{\to} \Gamma_\delta(\omega_1, \ldots , \omega_n))$. Thus $\langle\sigma_\delta(r_1, \ldots , r_n), \Gamma_\delta(\iota_1, \ldots , \iota_n)\rangle \in B(\Gamma_\delta(\omega_1, \ldots , \omega_n))$. The reader may also verify that B is a functor from Ω to SET and γ_δ is a natural transformation from $B^n;\times^{(n)}$ to $\Gamma;B$.

Intuitively, one can think of τ as assigning a "minimal" type to each member of R, and of a member of $B(\omega)$ as a member of R paired with an implicit conversion from its minimal type to ω.

For any object $\langle R,\sigma\rangle,\tau$ of ALG↓T,

$$
\begin{aligned}
&U_C(F_C(\langle R,\sigma\rangle,\tau)) = \langle \overline{R},\overline{\sigma}\rangle,\overline{\tau} \quad \text{where} \\
&\qquad \overline{R} = \{\omega,\langle r,\iota\rangle \mid \omega \in |\Omega| \text{ and } r \in R \text{ and } \iota \in \tau(r) \underset{\Omega}{\to} \omega\} \ , \\
&\qquad \overline{\sigma}_\delta \in \overline{R}^n \to \overline{R} \text{ is the function such that} \\
&\qquad\qquad \overline{\sigma}_\delta(\langle\omega_1,\langle r_1,\iota_1\rangle\rangle, \ldots , \langle\omega_n,\langle r_n,\iota_n\rangle\rangle) = \\
&\qquad\qquad\qquad \Gamma_\delta(\omega_1, \ldots , \omega_n), \langle\sigma_\delta(r_1, \ldots , r_n),\Gamma_\delta(\iota_1, \ldots , \iota_n)\rangle \ , \\
&\qquad \overline{\tau} \in \overline{R} \to |\Omega| \text{ is the function such that } \overline{\tau}(\omega,\langle r,\iota\rangle) = \omega \ .
\end{aligned}
$$

Let

$\eta_C(\langle R,\sigma\rangle,\tau) \in R \to \overline{R}$ be the function such that

$$\eta_C(\langle R,\sigma\rangle,\tau)(r) = \tau(r),\langle r,I^{\Omega}_{\tau(r)}\rangle \ .$$

The reader may verify that $\eta_C(\langle R,\sigma\rangle,\tau)$ is an homomorphism from R,σ to $\overline{R},\overline{\sigma}$ (which depends upon the fact that τ is an homomorphism from R,σ to $T = |\Omega|,\Gamma_{ob}$), and that

$$\begin{tikzcd}
R,\sigma \arrow[rr, "\eta_C(\langle R,\sigma\rangle,\tau)"] \arrow[dr, "\tau"'] & & \overline{R},\overline{\sigma} \arrow[dl, "\overline{\tau}"] \\
& T &
\end{tikzcd}$$

commutes in ALG. Thus $\eta_C(\langle R,\sigma\rangle,\tau) \in \langle R,\sigma\rangle,\tau \underset{ALG\downarrow T}{\to} \langle\overline{R},\overline{\sigma}\rangle,\overline{\tau} = \langle R,\sigma\rangle,\tau \underset{ALG\downarrow T}{\to} U_C(F_C(\langle R,\sigma\rangle,\tau))$.

Now we will show that F_C is a left adjoint of U_C, with associated natural transformation η_C. Let $\langle R,\sigma\rangle,\tau$ be an object of ALG↓T, let B',γ' be an object of CALG, and let h be a morphism in ALG↓T from $\langle R,\sigma\rangle,\tau$ to $U_C(B',\gamma')$, where $U_C(B',\gamma') = \langle R',\sigma'\rangle,\tau'$ is described by (1).

Since h is a function from R to R', the definition of R' implies that $h(r)$ will be a pair ω,x', where $x' \in B'(\omega)$. Moreover, since h is a morphism in ALG↓T,

$$\begin{tikzcd}
R,\sigma \arrow[rr, "h"] \arrow[dr, "\tau"'] & & R',\sigma' \arrow[dl, "\tau'"] \\
& T &
\end{tikzcd}$$

must commute in ALG, so that $\tau(r) = \tau'(h(r)) = \tau'(\omega,x') = \omega$. Thus $[h(r)]_1 = \tau(r)$ and $[h(r)]_2 \in B'(\tau(r))$.

Now suppose $\hat{h}$ is any morphism in $F_C(\langle R,\sigma\rangle,\tau) \underset{CALG}{\to} B',\gamma'$, where $F_C(\langle R,\sigma\rangle,\tau) = B,\gamma$ is described by (2), and consider the diagram

$$\begin{tikzcd}
\langle R,\sigma\rangle,\tau \arrow[r, "\eta_C(\langle R,\sigma\rangle,\tau)"] \arrow[dr, "h"'] & U_C(F_C(\langle R,\sigma\rangle,\tau)) \arrow[d, "U_C(\hat{h})"] \\
& U_C(B',\gamma')
\end{tikzcd} \qquad (D)$$

in ALG↓T.

From the definitions of η_C and of the action of U_C on morphisms, we have

$$U_C(\hat{h})(\eta_C(\langle R,\sigma\rangle,\tau)(r)) = U_C(\hat{h})(\tau(r),\langle r,I^{\Omega}_{\tau(r)}\rangle) = \tau(r),\hat{h}(\tau(r))(r,I^{\Omega}_{\tau(r)}) \ .$$

Thus the diagram (D) will commute if and only if, for all $r \in R$,

$$\hat{h}(\tau(r))(r,I^{\Omega}_{\tau(r)}) = [h(r)]_2 \ .$$

Moreover, since $\hat{h}$ is a category-sorted homomorphism from B,γ to B',γ', it is a natural transformation from B to B'. Thus for all $r \in R$, $\omega \in |\Omega|$, and $\iota \in \tau(r) \underset{\Omega}{\rightarrow} \omega$,

$$\begin{tikzcd}
B(\tau(r)) \arrow[r, "\hat{h}(\tau(r))"] \arrow[d, "B(\iota)"] & B'(\tau(r)) \arrow[d, "B'(\iota)"] \\
B(\omega) \arrow[r, "\hat{h}(\omega)"] & B'(\omega)
\end{tikzcd}$$

commutes in SET. In conjunction with the action of B on morphisms, this gives

$$\hat{h}(\omega)(\langle r,\iota\rangle) = \hat{h}(\omega)(B(\iota)(r,I^{\Omega}_{\tau(r)})) = B'(\iota)(\hat{h}(\tau(r))(r,I^{\Omega}_{\tau(r)})) \ .$$

Thus diagram (D) will commute if and only if

$$\hat{h}(\omega)(\langle r,\iota\rangle) = B'(\iota)([h(r)]_2)$$

holds for all $r \in R$, $\omega \in |\Omega|$, and $\iota \in \tau(r) \underset{\Omega}{\rightarrow} \omega$.

Since this equation completely determines $\hat{h}$, the adjunction property will hold if the resulting $\hat{h}$ is actually a category-sorted homomorphism from B,γ to B',γ'. We leave it to the reader to verify that $\hat{h}(\omega) \in B(\omega) \rightarrow B'(\omega)$, and that, because of the action of B on morphisms, $\hat{h}$ is a natural transformation from B to B'. The one nontrivial property to be shown is that $\hat{h}$ satisfies the homomorphic relationship with the interpretations γ and γ', i.e., that for all $n \geq 0$, $\delta \in \Delta_n$, and $\omega_1, \ldots , \omega_n \in |\Omega|$,

$$\begin{tikzcd}
B(\omega_1) \times \dots \times B(\omega_n) \arrow[r, "\gamma_\delta(\omega_1, \dots , \omega_n)"] \arrow[d, "\hat{h}(\omega_1) \times \dots \times \hat{h}(\omega_n)"] & B(\Gamma_\delta(\omega_1, \dots , \omega_n)) \arrow[d, "\hat{h}(\Gamma_\delta(\omega_1, \dots , \omega_n))"] \\
B'(\omega_1) \times \dots \times B'(\omega_n) \arrow[r, "\gamma'_\delta(\omega_1, \dots , \omega_n)"] & B'(\Gamma_\delta(\omega_1, \dots , \omega_n))
\end{tikzcd}$$

commutes in SET.

To see this, suppose $\langle r_1,\iota_1\rangle, \dots , \langle r_n,\iota_n\rangle \in B(\omega_1) \times \dots \times B(\omega_n)$. Then

$$\hat{h}(\Gamma_\delta(\omega_1, \dots , \omega_n))(\gamma_\delta(\omega_1, \dots , \omega_n)(\langle r_1,\iota_1\rangle, \dots , \langle r_n,\iota_n\rangle))$$

$$= \hat{h}(\Gamma_\delta(\omega_1, \dots , \omega_n))(\sigma_\delta(r_1, \dots , r_n), \Gamma_\delta(\iota_1, \dots , \iota_n))$$

$$= B'(\Gamma_\delta(\iota_1, \dots , \iota_n))([h(\sigma_\delta(r_1, \dots , r_n))]_2)$$

$$= B'(\Gamma_\delta(\iota_1, \dots , \iota_n))([\sigma'_\delta(h(r_1), \dots , h(r_n))]_2)$$

since h is an homomorphism from R,σ to R',σ'

$$= B'(\Gamma_\delta(\iota_1, \dots , \iota_n))(\gamma'_\delta(\tau(r_1), \dots , \tau(r_n))([h(r_1)]_2, \dots , [h(r_n)]_2))$$

by the definition of σ'_δ given in (1)

$$= \gamma'_\delta(\omega_1, \dots , \omega_n)(B'(\iota_1)([h(r_1)]_2), \dots , B'(\iota_n)([h(r_n)]_2))$$

since γ'_δ is a natural transformation from $B'^n;\times^{(n)}$ to $\Gamma_\delta;B'$

$$= \gamma'_\delta(\omega_1, \dots , \omega_n)(\hat{h}(\omega_1)(r_1,\iota_1), \dots , \hat{h}(\omega_n)(r_n,\iota_n))$$

In summary, we have constructed the adjunctions

$$\begin{tikzcd}
\mathrm{SET}{\downarrow}|\Omega| \arrow[r, shift left, "F_T"] & \mathrm{ALG}{\downarrow}T \arrow[l, shift left, "U_T"] \arrow[r, shift left, "F_C"] & \mathrm{CALG} \arrow[l, shift left, "U_C"]
\end{tikzcd}$$

with associated natural transformations η_T and η_C. The adjunction used in the main text is the composition of these adjunctions:

$$U = U_C;U_T$$

$$F = F_T;F_C$$

$$\eta(S,\tau_S) = \eta_T(S,\tau_S);_{\mathrm{SET}\downarrow|\Omega|}U_T(\eta_C(F_T(S,\tau_S))) .$$

The free $\Omega\Delta\Gamma$-algebra $F(S,\tau_S)$ generated by S,τ_S is given explicitly by (2), where R,σ is the free Δ-algebra generated by S and $\tau \in R \to |\Omega|$ is the unique homomorphism such that $\eta_A(S);\tau = \tau_S$.

In the special case where Ω is a preordered set, there is at most one $\iota \in \tau(r) \underset{\Omega}{\to} \omega$, so that (2) is isomorphic to the much simpler definition:

$$B(\omega) = \{r \mid r \in R \text{ and } \tau(r) \leq \omega\}$$

$B(\omega \leq \omega')$ is the identity inclusion from $B(\omega)$ to $B(\omega')$,

$$\gamma_\delta(\omega_1, \ldots , \omega_n)(r_1, \ldots , r_n) = \sigma_\delta(r_1, \ldots , r_n).$$

In this case, $B(\omega)$ is simply the subset of the terms of the ordinary free Δ-algebra whose minimal sort is a subsort of ω, the implicit conversion functions are all identity inclusions, and the operators are interpreted the same way as in the ordinary free algebra.

REFERENCES

1. Goguen, J. A., "Order Sorted Algebras: Exceptions and Error Sorts, Coercions and Overloaded Operators", Semantics and Theory of Computation Report #14, Computer Science Department, U.C.L.A., (December 1978). To appear in Journal of Computer and Systems Science.

2. Goguen, J. A., Thatcher, J. W., Wagner, E. G., and Wright, J. B., "Initial Algebra Semantics and Continuous Algebras", Journal ACM 24 (1) pp. 68-95 (January 1977).

3. Burstall, R. M., and Landin, P. J., "Programs and Their Proofs: An Algebraic Approach", in Machine Intelligence 4, B. Meltzer and D. Michie, Eds., Edinburgh University Press, pp. 17-43 (1969).

4. Birkhoff, G., and Lipson, J. D., "Heterogeneous Algebras", Journal of Combinatorial Theory 8, pp. 115-133 (1970).

5. Higgins, P. J., "Algebras with a Schema of Operators", Math. Nachr. 27, pp. 115-132 (1963).

6. Morris, J. H., "Types are not Sets", Proc. ACM Symposium on Principles of Programming Languages, pp. 120-124, Boston (1973).

7. Reynolds, J. C., "Syntactic Control of Interference", Proc. Fifth ACM Symposium on Principles of Programming Languages, pp. 39-46, Tucson (1978).

8. Reynolds, J. C., The Craft of Programming, in preparation.

9. MacLane, S., Categories for the Working Mathematician, Springer-Verlag, New York (1971).

10. Reynolds, J. C., "GEDANKEN - A Simple Typeless Language Based on the Principle of Completeness and the Reference Concept", Comm. ACM 13 (5), pp. 308-319 (May 1970).

11. Arbib, M. A., and Manes, E. G., Arrows, Structures, and Functors - The Categorical Imperative, Academic Press, New York (1975).