INFORMATION PROCESSING 83, R.E.A. Mason (ed.)
Elsevier Science Publishers B.V. (North-Holland)

Types, Abstraction and Parametric Polymorphism*

John C. Reynolds
Syracuse University
Syracuse, New York, USA

Invited Paper

Abstract

We explore the thesis that type structure is a syntactic discipline for maintaining levels of abstraction. Traditionally, this view has been formalized algebraically, but the algebraic approach fails to encompass higher-order functions. For this purpose, it is necessary to generalize homomorphic functions to relations; the result is an "abstraction" theorem that is applicable to the typed lambda calculus and various extensions, including user-defined types.

Finally, we consider polymorphic functions, and show that the abstraction theorem captures Strachey's concept of parametric, as opposed to ad hoc, polymorphism.

1. A Fable

Once upon a time, there was a university with a peculiar tenure policy. All faculty were tenured, and could only be dismissed for moral turpitude. What was peculiar was the definition of moral turpitude: making a false statement in class. Needless to say, the university did not teach computer science. However, it had a renowned department of mathematics.

One semester, there was such a large enrollment in complex variables that two sections were scheduled. In one section, Professor Descartes announced that a complex number was an ordered pair of reals, and that two complex numbers were equal when their corresponding components were equal. He went on to explain how to convert reals into complex numbers, what "i" was, how to add, multiply, and conjugate complex numbers, and how to find their magnitude.

In the other section, Professor Bessel announced that a complex number was an ordered pair of reals the first of which was nonnegative, and that two complex numbers were equal if their first components were equal and either the first components were zero or the second components differed by a multiple of 2π. He then told an entirely different story about converting reals, "i", addition, multiplication, conjugation, and magnitude.

Then, after their first classes, an unfortunate mistake in the registrar's office caused the two sections to be interchanged. Despite this, neither Descartes nor Bessel ever committed moral turpitude, even though each was judged by the other's definitions. The reason was that they both had an intuitive understanding of type. Having defined complex numbers and the primitive operations upon them, thereafter they spoke at a level of abstraction that encompassed both of their definitions.

The moral of the fable is that:

> Type structure is a syntactic discipline for enforcing levels of abstraction.

For instance, when Descartes introduced the complex plane, this discipline prevented him from saying $\texttt{Complex} = \texttt{Real} \times \texttt{Real}$, which would have contradicted Bessel's definition. Instead, he defined the mapping $f : \texttt{Real} \times \texttt{Real} \to \texttt{Complex}$ such that $f(x, y) = x + i \times y$, and proved that this mapping is a bijection.

More subtly, although both lecturers introduced the set $\texttt{Int}^*$ of sequences of integers, and spoke of sets such as $\texttt{Int}^* + \texttt{Complex}$, $\texttt{Int}^* \times \texttt{Complex}$ and $\texttt{Int}^* \to \texttt{Complex}$, they never mentioned $\texttt{Int}^* \cup \texttt{Complex}$ or $\texttt{Int}^* \cap \texttt{Complex}$. Intuitively, they thought of sequences of integers and complex numbers as entities so immiscible that the union and intersection of $\texttt{Int}^*$ and $\texttt{Complex}$ are undefined.

More precisely, there is not such thing as *the* set of complex numbers. Instead, the type "$\texttt{Complex}$" denotes an abstraction that can be realized or represented by a variety of sets, with varying unions and intersections with $\texttt{Int}^*$ or $\texttt{Real} \times \texttt{Real}$.

A second moral of our fable is that types are not limited to computation. Thus (in the absence of recursion) they should be explicable without invoking constructs, such as Scott domains, that are peculiar to the theory of computation. Descartes and Bessel would be baffled by an explanation of their intuition that introduced undefined or approximate complex numbers.

What computation has done is to create the necessity of formalize type disciplines, to the point where they can be enforced mechanically. The idea that type disciplines enforce abstraction clearly underlies such languages as CLU {1} and ALPHARD {2}, and such papers as {3} and {4}. More recently, however, many formalizations have treated types as predicates or other entities denoting specific subsets of some universe of values {5–9}. This work has stemmed from Scott's discovery of how to construct sufficiently rich universes without encountering Russell's paradox. But it obscures the idea of abstraction, e.g. if types denote specific

*Work supported by National Science Foundation Grant MCS-8017577.

subsets of a universe then their unions and intersections are well-defined.

The major exception to this trend is the algebraic view of types {10–12}, in which a type, with its primitive operations, is an abstraction over a variety of algebras. Roughly speaking, type discipline is the limitation of language to terms of a free algebra. Each term is uniquely interpretable within every algebra of the variety, and its interpretations are related by the homomorphisms between the algebras.

Unfortunately, the algebraic approach is intrinsically first-order; if primitive operations are to be operations of an algebra they cannot be higher-order functions on the carrier of the algebra. It might be possible to extend algebra to encompass higher-order functions if homomorphic functions between carriers induced higher-order functions between functions on these carriers. Moreover, the basic homomorphic equation

$$\rho(f(x_1, \ldots, x_n)) = f'(\rho x_1, \ldots, \rho x_n)$$

shows how a homomorphism ρ from A to A' induces a relation between functions $f \in |A|^n \to |A|$ and $f' \in |A'|^n \to |A'|$. However, this relation is usually not a function.

The way out of this impasse is to generalize homomorphisms from functions to relations.

2. An Extended Typed Lambda Calculus

To characterize abstraction in a setting that permits higher-order functions, we will define an extension of the explicitly typed lambda calculus, which constant types, products, and generic conditional expressions, and show that its classical set-theoretic semantics obeys an "abstraction theorem" formalizing our thesis that type structure preserves abstraction.

The expressions of this language are divided into type expressions and ordinary expressions. To define the syntax of type expressions, we assume that we are given

C: A set of *type constants*, containing at least the type constant `Bool`.
T: An infinite countable set of *type variables*.

Then Ω, the set of *type expressions*, is the least set such that

$$\textit{If } \kappa \in C \textit{ then } \kappa \in \Omega, \tag{Ω1}$$
$$\textit{If } \tau \in T \textit{ then } \tau \in \Omega, \tag{Ω2}$$
$$\textit{If } \omega, \omega' \in \Omega \textit{ then } \omega \to \omega' \in \Omega, \tag{Ω3}$$
$$\textit{If } \omega, \omega' \in \Omega \textit{ then } \omega \times \omega' \in \Omega. \tag{Ω4}$$

We also distinguish Ω_C, the subset of Ω consisting of fixed type expressions, i.e. type expressions that do not contain occurrences of members of T.

Next, we assume that we are given

V: An infinite countable set of *ordinary* variables.

Then a *type assignment* is a function from some finite subset of V to Ω. We write

$$\Omega^* = \{\pi \mid \pi \in F \to \Omega \textit{ for some finite } F \subseteq V\}$$

for the set of type assignments.

To define the syntax of ordinary expressions, we assume that, for each $\omega \in \Omega_C$, we are given

K_ω: A set of *ordinary constants* of fixed type ω.

Then we define the family

$$\langle E_{\pi\omega} \mid \pi \in \Omega^*, \omega \in \Omega \rangle$$

of sets, in which $E_{\pi\omega}$ is the set of those ordinary expressions whose free ordinary variables belong to the domain of π and which take on the type ω under the assignment of types to variables given by π. This is the least family of sets satisfying

$$\textit{If } k \in K_\omega \textit{ then } k \in E_{\pi\omega}, \tag{Ea}$$
$$\textit{If } v \in \mathrm{dom}\ \pi \textit{ then } v \in E_{\pi,\pi v}, \tag{Eb}$$
$$\textit{If } e_1 \in E_{\pi,\omega\to\omega'} \textit{ and } e_2 \in E_{\pi\omega} \textit{ then } e_1(e_2) \in E_{\pi\omega'}, \tag{Ec}$$
$$\textit{If } e \in E_{[\pi|v:\omega],\omega'} \textit{ then } \lambda v : \omega.e \in E_{\pi,\omega\to\omega'}, \tag{Ed}$$
$$\textit{If } e \in E_{\pi\omega} \textit{ and } e' \in E_{\pi\omega'} \textit{ then } \langle e, e' \rangle \in E_{\pi,\omega\times\omega'}, \tag{Ee}$$
$$\textit{If } e \in E_{\pi,\omega\times\omega'} \textit{ then } e.1 \in E_{\pi\omega} \textit{ and } e.2 \in E_{\pi\omega'}, \tag{Ef}$$
$$\textit{If } b \in E_{\pi,\mathtt{Bool}} \textit{ and } e, e' \in E_{\pi\omega} \textit{ then} \quad \mathtt{if}\ b\ \mathtt{then}\ e\ \mathtt{else}\ e' \in E_{\pi\omega}. \tag{Eg}$$

Here dom π denotes the domain of π, and $[\pi \mid v : \omega]$ denotes the function with domain dom $\pi \cup \{v\}$ such that $[\pi \mid v : \omega]v' = \mathsf{if}\ v = v'\ \mathsf{then}\ \omega\ \mathsf{else}\ \pi v'$.

The presence of ω in $\lambda v : \omega.e$ is what we mean by "explicit" typing; it assures that, under a given type assignment π, every ordinary expression has a unique type, i.e. $E_{\pi\omega}$ and $E_{\pi\omega'}$ are disjoint when $\omega \neq \omega'$.

To specify the semantics of type expressions, we assume that we are given

CS: A mapping from C to the class of sets, such that $CS(\mathtt{Bool}) = \{\mathtt{true}, \mathtt{false}\}$,

and we define a *set assignment* to be a mapping from T to the class of sets. Then we use # to denote the following extension of set assignments from type variables to type expressions: If S is a set assignment then $S^\#$ is the mapping from Ω to the class of sets such that

$$\textit{If } \kappa \in C \textit{ then } S^\#\ \kappa = CS\ \kappa \tag{S1}$$
$$\textit{If } \tau \in T \textit{ then } S^\#\ \tau = S\ \tau, \tag{S2}$$
$$\textit{If } \omega, \omega' \in \Omega \textit{ then } S^\#(\omega \to \omega') = S^\#\omega \to S^\#\omega', \tag{S3}$$
$$\textit{If } \omega, \omega' \in \Omega \textit{ then } S^\#(\omega \times \omega') = S^\#\omega \times S^\#\omega', \tag{S4}$$

where $s \to s'$ denotes the set of all functions from s to s' and $s \times s'$ denotes the set of pairs $\langle x, x' \rangle$ such that $x \in s$ and $x' \in s'$. Then the set $S^\#\omega$ is the meaning of the type

expression ω under the set assignment S. Note that, when $\omega \in \Omega_C$, $S^{\#}\omega$ is independent of the set assignment S.

We use $*$ to denote a further extension from type expressions of type assignments: If S is a set assignment then $S^{\#*}$ is the mapping from Ω^* to the class of sets such that

$$S^{\#*} = \prod_{v \in \mathrm{dom}\ \pi} S^{\#}(\pi v). \tag{S*}$$

Then $S^{\#*}\pi$ is the set of environments appropriate to π and S.

A conventional semantics for ordinary expressions would be obtained by fixing some set assignment S and defining a family of semantic functions from $E_{\pi\omega}$ to $S^{\#*}\pi \to S^{\#}\omega$. However, to capture abstraction properties we will need to relation the meanings of an expression under different set assignments. For this reason, we will treat set assignments as explicit parameters of the semantic functions. Specifically, for each $\pi \in \Omega^*$ and $\omega \in \Omega$ we will define a semantic function

$$\mu_{\pi\omega} \in E_{\pi\omega} \to \prod_{S \in \mathcal{S}} (S^{\#*}\pi \to S^{\#}\omega),$$

where $\mathcal{S}$ denotes the class of all set assignments.

We assume we are given, for each $\omega \in \Omega_C$, a function

$$\alpha_\omega \in K_\omega \to S^{\#}\omega$$

providing meanings (independent of S) to the ordinary constants of type ω. Then the semantic functions are defined by

$$\textit{If } k \in K_\omega \textit{ then } \mu_{\pi\omega}[\![k]\!]\ S\ \eta = \alpha_\omega\ k, \tag{Ma}$$

$$\textit{If } v \in \mathrm{dom}\ \pi \textit{ then } \mu_{\pi,\pi v}[\![v]\!]\ S\ \eta = \eta\ v, \tag{Mb}$$

$$\textit{If } e_1 \in E_{\pi,\omega\to\omega'} \textit{ and } e_2 \in E_{\pi\omega} \textit{ then} \tag{Mc}$$
$$\mu_{\pi\omega'}[\![e_1(e_2)]\!]\ S\ \eta = \mu_{\pi,\omega\to\omega'}[\![e_1]\!]\ S\ \eta\ (\mu_{\pi\omega}[\![e_2]\!]\ S\ \eta),$$

$$\textit{If } e \in E_{[\pi|v:\omega],\omega'} \textit{ then} \tag{Md}$$
$$\mu_{\pi,\omega\to\omega'}[\![\lambda v : \omega.e]\!]\ S\ \eta = f$$
$$\textit{where } f \in S^{\#}\omega \to S^{\#}\omega' \textit{ is such that}$$
$$f\ x = \mu_{[\pi|v:\omega],\omega'}[\![e]\!]\ S\ [\eta \mid v : x],$$

$$\textit{If } e \in E_{\pi\omega} \textit{ and } e' \in E_{\pi\omega'} \textit{ then} \tag{Me}$$
$$\mu_{\pi,\omega\times\omega'}[\![\langle e, e'\rangle]\!]\ S\ \eta = \langle \mu_{\pi,\omega}[\![e]\!]\ S\ \eta, \mu_{\pi,\omega'}[\![e']\!]\ S\ \eta\rangle,$$

$$\textit{If } e \in E_{\pi,\omega\times\omega'} \textit{ then} \tag{Mf}$$
$$\mu_{\pi\omega}[\![e.1]\!]\ S\ \eta = [\mu_{\pi,\omega\times\omega'}[\![e]\!]\ S\ \eta]_1$$
$$\mu_{\pi\omega}[\![e.2]\!]\ S\ \eta = [\mu_{\pi,\omega\times\omega'}[\![e]\!]\ S\ \eta]_2,$$

$$\textit{If } b \in E_{\pi,\mathsf{Bool}} \textit{ and } e, e' \in E_{\pi\omega} \textit{ then} \tag{Mg}$$
$$\mu_{\pi\omega}[\![\texttt{if } b \texttt{ then } e \texttt{ else } e']\!]\ S\ \eta =$$
$$\textsf{if } \mu_{\pi,\mathsf{Bool}}[\![b]\!]\ S\ \eta = \texttt{true}$$
$$\textsf{then } \mu_{\pi,\omega}[\![e]\!]\ S\ \eta$$
$$\textsf{else } \mu_{\pi,\omega}[\![e']\!]\ S\ \eta.$$

3. The Abstraction Theorem

We now want to formulate an abstraction theorem that connects the meanings of an ordinary expression under different set assignments. The underlying idea is that the meanings of an expression in "related" environments will be "related" values. But here "related" must denote a different relation for each type expression and type assignment. Moreover, while the relation for each type variable is arbitrary, the relations for compound type expressions and type assignments must be induced in a specific way. In other words, we must specify how an assignment of relations to type variables is extended to type expressions and type assignments.

This can be formalized by defining a "relation semantics" for type expressions that parallels their set-theoretic semantics. For sets s_1 and s_2, we introduce the set

$$\mathrm{Rel}(s_1, s_2) = \{r \mid r \subseteq s_1 \times s_2\}$$

of binary relations between s_1 and s_2, and we write

$$\mathsf{I}(s) = \{\langle x, x\rangle \mid x \in s\} \in \mathrm{Rel}(s, s)$$

for the identity relation on a set s. For $r \in \mathrm{Rel}(s_1, s_2)$ $r' \in \mathrm{Rel}(s_1', s_2')$, we write $r \to r'$ for the relation in $\mathrm{Rel}(s_1 \to s_1', s_2 \to s_2')$ such that

$$\langle f_1, f_2\rangle \in r \to r' \textit{ iff } (\forall \langle x_1, x_2\rangle \in r)\ \langle f_1\ x_1, f_2\ x_2\rangle \in r',$$

and $r \times r'$ for the relation in $\mathrm{Rel}(s_1 \times s_1', s_2 \times s_2')$ such that

$$\langle\langle x_1, x_1'\rangle, \langle x_2, x_2'\rangle\rangle \in r \times r' \textit{ iff } \langle x_1, x_2\rangle \in r \textit{ and } \langle x_1', x_2'\rangle \in r'.$$

In other words, functions are related if they map related arguments into related results, and pairs are related if their corresponding components are related.

For set assignments S_1 and S_2, a member of

$$\prod_{\tau \in T} \mathrm{Rel}(S_1\tau, S_2\tau)$$

is called a (binary) *relation assignment* between S_1 and S_2. Having defined $\to$ and $\times$ for relations we can extend relation assignments from T to Ω and Ω^* in essentially the same way as we extended set assignments. If R is a relation assignment between S_1 and S_2 then

$$R^{\#} \in \prod_{\omega \in \Omega} \mathrm{Rel}(S_1^{\#}\omega, S_2^{\#}\omega)$$

is such that

$$\textit{If } \kappa \in C \textit{ then } R^{\#}\ \kappa = \mathsf{I}(CS\ \kappa), \tag{R1}$$

$$\textit{If } \tau \in T \textit{ then } R^{\#}\ \tau = R\ \tau, \tag{R2}$$

$$\textit{If } \omega, \omega' \in \Omega \textit{ then } R^{\#}(\omega \to \omega') = R^{\#}\omega \to R^{\#}\omega', \tag{R3}$$

$$\textit{If } \omega, \omega' \in \Omega \textit{ then } R^{\#}(\omega \times \omega') = R^{\#}\omega \times R^{\#}\omega', \tag{R4}$$

and

$$R^{\#*} \in \prod_{\pi \in \Omega^*} \mathrm{Rel}(S_1^{\#*}\pi, S_2^{\#*}\pi)$$

is such that

$$\langle \eta_1, \eta_2 \rangle \in R^{\#*}\pi \text{ iff } (\forall v \in \text{dom } \pi)\ \langle \eta_1 v, \eta_2 v \rangle \in R^{\#}(\pi v). \quad (\text{R}^*)$$

It is easily seen that $\rightarrow$ and $\times$ preserve identity relations. Thus we have the

Identity Extension Lemma. *Suppose IA is a relation assignment such that* $IA\ \tau = \mathsf{I}(S\ \tau)$ *for all* $\tau \in T_0 \subseteq T$. *Then*

$$IA^{\#}\omega = \mathsf{I}(S^{\#}\omega) \text{ for all } \omega \text{ such that } \mathsf{F}(\omega) \subseteq T_0,$$

and

$$IA^{\#*}\pi = \mathsf{I}(S^{\#*}\pi) \text{ for all } \pi \text{ such that } (\forall v \in \text{dom } \pi)\ \mathsf{F}(\pi v) \subseteq T_0.$$

Here $\mathsf{F}(\omega)$ denotes the set of (free) variables occurring in ω.

Finally, we can state the

Abstraction Theorem. *Let R be a relation assignment between set assignments* S_1 *and* S_2. *For all* $\pi \in \Omega^*$, $\omega \in \Omega$, $e \in E_{\pi\omega}$, *and* $\langle \eta_1, \eta_2 \rangle \in R^{\#*}\pi$,

$$\langle \mu_{\pi\omega}[\![e]\!]\ S_1\ \eta_1, \mu_{\pi\omega}[\![e]\!]\ S_2\ \eta_2 \rangle \in R^{\#}\omega.$$

In essence, the semantics of any expression maps related environments into related values. For the case where e is a constant, this theorem depends upon the fact that, by the identity extension lemma with T_0 empty, $R^{\#}\omega$ is an identity relation whenever $\omega \in \Omega_C$. The remainder of the proof is by structural induction on the syntax of ordinary expressions.

The abstraction theorem is closely related to Proposition 1 in {13}. This relationship will be discussed in Section 9.

4. Type Definition

Several programming languages, of which CLU {1} is perhaps the earliest, permit one to introduce a new type by defining its representation in terms of builtin or previously defined types and defining its primitive operations in terms of procedures that act upon the representation. Within the scope of such a definition, the new type is *opaque*, i.e. the meaning of the scope is an abstraction over appropriately related definitions of the type.

In this section, we extend our illustrative language to provide this kind of type definition, and derive a consequence of the abstraction theorem that formalizes type opacity.

First, we introduce the notion of type substitution. For $\tau \in T$ and $\omega, \omega' \in \Omega$, we write $\omega'/\tau \rightarrow \omega$ to denote the type expression obtained from ω' by substitution ω for all (free) occurrences of τ, and for $\pi \in \Omega^*$ we write $\pi/\tau \rightarrow \omega$ to denote the type assignment such that $(\pi/\tau \rightarrow \omega)\ v = (\pi v)/\tau \rightarrow \omega$ for all $v \in \text{dom } \pi$.

Since the definitions of # and $*$ are algebraic in nature, it is easy to show that

$$S^{\#}(\omega'/\tau \rightarrow \omega) = [S \mid \tau : S^{\#}\omega]^{\#}\omega',$$
$$S^{\#*}(\pi/\tau \rightarrow \omega) = [S \mid \tau : S^{\#}\omega]^{\#*}\pi,$$
$$R^{\#}(\omega'/\tau \rightarrow \omega) = [R \mid \tau : R^{\#}\omega]^{\#}\omega',$$
$$R^{\#*}(\pi/\tau \rightarrow \omega) = [R \mid \tau : R^{\#}\omega]^{\#*}\pi.$$

Second, for $\pi \in \Omega^*$ and $\tau \in T$, we define

$$\pi - \tau = \pi \restriction \{v \mid v \in \text{dom } \pi \text{ and } \tau \notin \mathsf{F}(\pi v)\}$$

where $\mathsf{F}(\pi v)$ denotes the set of type variables occurring (free) in the type expression πv, and $\pi \restriction F$ denotes the restriction of π to F. Then $\pi - \tau$ is the least restriction of π that is unchanged by all substitutions for τ:

$$(\pi - \tau)/\tau \rightarrow \omega = \pi - \tau.$$

thus, if $\eta \in S^{\#*}\pi$ then

$$\eta \restriction \text{dom}(\pi - \tau) \in S^{\#*}(\pi - \tau) = S^{\#*}((\pi - \tau)/\tau \rightarrow \omega)$$
$$= [S \mid \tau : S^{\#}\omega]^{\#*}(\pi - \tau).$$

In conjunction with

$$S^{\#}(\omega'/\tau \rightarrow \omega) = [S \mid \tau : S^{\#}\omega]^{\#}\omega',$$

this justifies the following extension of ordinary expressions, which provides the pure definition of types (without primitive operation):

$$\textit{If } \tau \in T, \omega, \omega' \in \Omega, \pi \in \Omega^*, \textit{ and } e \in E_{\pi-\tau,\omega'} \textit{ then} \quad (\text{Eh})$$
$$\texttt{lettype}\ \tau = \omega\ \texttt{in}\ e \in E_{\pi,(\omega'/\tau\rightarrow\omega)},$$
$$\textit{If } \tau \in T, \omega, \omega' \in \Omega, \pi \in \Omega^*, \textit{ and } e \in E_{\pi-\tau,\omega'} \textit{ then} \quad (\text{Mh})$$
$$\mu_{\pi,(\omega'/\tau\rightarrow\omega)}[\![\texttt{lettype}\ \tau = \omega\ \texttt{in}\ E]\!]\ S\ \eta =$$
$$\mu_{\pi-\tau,\omega'}[\![e]\!]\ [S \mid \tau : S^{\#}\omega]\ (\eta \restriction \text{dom}(\pi - \tau)).$$

Notice that the condition $e \in E_{\pi-\tau,\omega'}$ limits the ordinary variables occurring free in e to variables whose types do not depend on τ.

This extension preserves the abstraction theorem. Moreover, the abstraction theorem implies a relationship between the meanings of `lettype` $\tau = \omega$ `in` e for different representations ω:

Pure Type Definition Theorem 1. *Let S be a set assignment,* $\omega_1, \omega_2 \in \Omega$, *and r be a relation between* $S^{\#}\omega_1$ *and* $S^{\#}\omega_2$. *For all* $\pi \in \Omega^*$, $\tau \in T$, $\omega' \in \Omega$, $e \in E_{\pi-\tau,\omega'}$, *and* $\eta \in S^{\#*}\pi$,

$$\langle \mu_{\pi,(\omega'/\tau\rightarrow\omega_1)}[\![\texttt{lettype}\ \tau = \omega_1\ \texttt{in}\ e]\!]\ S\ \eta,$$
$$\mu_{\pi,(\omega'/\tau\rightarrow\omega_2)}[\![\texttt{lettype}\ \tau = \omega_2\ \texttt{in}\ e]\!]\ S\ \eta\rangle$$
$$\in [IA \mid \tau : r]^{\#}\omega',$$

where IA is the relation assignment such that $IA\ \tau = \mathsf{I}(S\ \tau)$ *for all* $\tau \in T$.

Proof: By the identity extension lemma, $[IA \mid \tau : r]^{\#*}(\pi - \tau)$ is an identity relation, and thus contains $\langle \eta \restriction \mathrm{dom}(\pi - \tau), \eta \restriction \mathrm{dom}(\pi - \tau)\rangle$. The rest of the theorem follows by applying the abstraction theorem to the definition of the `lettype` construct. □

Since our language is higher-order, it is trivial to extend the pure `lettype` construction to include the definition of primitive operations; the extension is semantically insignificant and can be defined as syntactic sugar. For simplicity, we only consider defining a single primitive with name v_p, abstract type ω_p, and definition e_p:

If $\tau \in T, \omega, \omega', \omega_p \in \Omega, \pi \in \Omega^*, e \in E_{[\pi-\tau|v_p:\omega_p],\omega'}$,
and $e_p \in E_{\pi,(\omega_p/\tau\to\omega)}$ *then*
$\quad$ `lettype` $\tau = \omega$ `with` $v_p : \omega_p = e_p$ `in` e
is a member of $E_{\pi,(\omega'/\tau\to\omega)}$ *with the same meaning as*
$\quad (\texttt{lettype}\ \tau = \omega\ \texttt{in}\ \lambda v_p : \omega_p.e)(e_p)$.

Notice that, unlike the algebraic approach to type definition, the operation type ω_p can be higher-order. The more complicated case, where several primitive operations, or even several types, are defined is conceptually similar.

When applied to this extension, the pure type definition theorem leads to the

General Type Definition Theorem. *Let*

S *be a set assignment*, $\omega_1, \omega_2, \omega', \omega_p \in \Omega$,
r *be a relation between* $S^{\#}\omega_1$ *and* $S^{\#}\omega_2$,
$\pi \in \Omega^*, \eta \in S^{\#*}\pi, \tau \in T, v_p \in V$,
$e \in E_{[\pi-\tau|v_p:\omega_p],\omega'}$,
$e_1 \in E_{\pi,(\omega_p/\tau\to\omega_1)}, e_2 \in E_{\pi,(\omega_p/\tau\to\omega_2)}$

be such that

$$\langle \mu_{\pi,(\omega_p/\tau\to\omega_1)}[\![e_1]\!]\ S\ \eta,\ \mu_{\pi,(\omega_p/\tau\to\omega_2)}[\![e_2]\!]\ S\ \eta\rangle \in [IA \mid \tau : r]^{\#}\omega_p,$$

where IA is the relation assignment such that $IA\ \tau = \mathsf{I}(S\ \tau)$ *for all* $\tau \in T$. *Then*

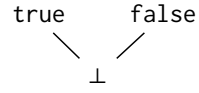
$$\langle \mu_{\pi,(\omega'/\tau\to\omega_1)}[\![\texttt{lettype}\ \tau = \omega_1\ \texttt{with}\ v_p : \omega_p = e_1\ \texttt{in}\ e]\!]\ S\ \eta,\ \mu_{\pi,(\omega'/\tau\to\omega_2)}[\![\texttt{lettype}\ \tau = \omega_2\ \texttt{with}\ v_p : \omega_p = e_2\ \texttt{in}\ e]\!]\ S\ \eta\rangle \in [IA \mid \tau : r]^{\#}\omega'.$$

It can be argued that, to achieve complete type opacity, the extended `lettype` ... `with` construction should be restricted by requiring that τ not occur (free) in ω'. In this case, $[IA \mid \tau : r]^{\#}\omega'$ in the conclusion of the general type definition theorem can be replaced by the identity relation $IA^{\#}\omega'$, so that $\mu_{\pi,\omega'}[\![\texttt{lettype}\ \tau = \omega_i\ \texttt{with}\ v_p : \omega_p = e_i\ \texttt{in}\ e]\!]\ S\ \eta$ is independent of i.

5. From Sets to Domains

Although this paper is primarily concerned with set-theoretic semantics, in this section and the next we digress to consider the interpretation of types as domains rather than sets. We define a domain to be a complete partial ordering, i.e. a partial ordering containing a least element $\perp$ and least upper bounds of all directed subsets.

The essential changes in the semantics of our language are that the class of sets is replaced by the class of domains, set assignments become domain assignments, $\to$ between domains is redefined to denote a pointwise-ordered set of continuous functions, and the products in (S4) and (S*) are also pointwise ordered. Since constant types must denote domains, $CS(\texttt{Bool})$ becomes the domain



and the semantics of the conditional expression is extended to

If $b \in E_{\pi,\texttt{Bool}}$ *and* $e, e' \in E_{\pi,\omega}$ *then* (Mg')
$\quad \mu_{\pi\omega}[\![\texttt{if}\ b\ \texttt{then}\ e\ \texttt{else}\ e']\!]\ S\ \eta =$
$\qquad$ if $\mu_{\pi,\texttt{Bool}}[\![b]\!]\ S\ \eta = \texttt{true}$
$\qquad$ then $\mu_{\pi\omega}[\![e]\!]\ S\ \eta$
$\qquad$ else if $\mu_{\pi,\texttt{Bool}}[\![b]\!]\ S\ \eta = \texttt{false}$
$\qquad$ then $\mu_{\pi\omega}[\![e']\!]\ S\ \eta$
$\qquad$ else $\perp_{S^{\#}\omega}$.

Then the ordinary expressions can be extended to include a generic fixed-point operator:

If $e \in E_{\pi,\omega\to\omega}$ *then* $\mathsf{Y}\ e \in E_{\pi\omega}$, (Ei)
If $e \in E_{\pi,\omega\to\omega}$ *then* $\mu_{\pi\omega}[\![\mathsf{Y}\ e]\!]\ S\ \eta$ *is the least* (Mi)
$\quad$ *fixed-point of* $\mu_{\pi,\omega\to\omega}[\![e]\!]\ S\ \eta$ *in the domain* $S^{\#}\omega$.

The addition of a fixed-point operator, as well as the extension of the meaning of conditionals, causes the abstraction theorem to fail. To save the situation, we must require the relations denoted by type expressions and assignments to be *complete* relations.

A subset C of a domain D is *complete* iff C contains $\perp_D$ and, for every directed $X \subseteq D$, if $X \subseteq C$ then the least upper bound of X belongs to C. A relation between domains s_1 and s_2 is *complete* iff it is a complete subset of the pointwise-ordered product $s_1 \times s_2$.

Identity relations are complete, and completeness is preserved by the action of $\to$ and $\times$ upon relations. Thus it is consistent to restrict the sets $\mathsf{Rel}(s_1, s_2)$ to complete relations. This restriction is sufficient to regain the abstraction theorem.

(Further difficulties arise if domains are taken to be complete lattices rather than complete partial orderings, because of the behavior of the conditional construct for the overdefined element $\top_{\mathsf{Bool}}$. These difficulties can be resolved by taking $R^{\#}\mathsf{Bool}$ to be the partial identity relation that does not relate $\top$ to itself, or by using a doubly strict conditional and strengthening the definition of a complete subset to require it to contain $\top$. These complications are typical of the vagaries of overdefined elements.)

As an aside, we note that ideals are a special case of complete subsets: A complete subset C of a domain D is an *ideal* iff it is downward closed, i.e. $(\forall x \in D)(\forall y \in C)\ x \sqsubseteq y$ implies $x \in C$. The operations $\rightarrow$ and $\times$ preserve ideal relations, but binary identity relations are not ideals. However, if one works with unary, rather than binary relations (which is a special case of the extension to multinary relations discussed in Section 9) then identity relations are trivially ideals, and our operations $\rightarrow$ and $\times$ become the operations $\Rightarrow$ and $\boxtimes$ used in {7}.

Finally, we note that the partial orderings of domains are complete relations that are preserved by $\rightarrow$ and $\times$, i.e.

$$\begin{aligned} \sqsubseteq_s \ \rightarrow\ \sqsubseteq_{s'} \ &= \ \sqsubseteq_{s \rightarrow s'}\ , \\ \sqsubseteq_s \ \times\ \sqsubseteq_{s'} \ &= \ \sqsubseteq_{s \times s'}\ . \end{aligned}$$

In conjunction with reflexivity, this implies that both the identity extension lemma and the abstraction theorem remain true if identity relations on domains are replaced by the partial orderings, i.e. if $\mathsf{I}(s)$ is redefined to be $\sqsubseteq_s$. This "order-relation" semantics of type expressions will be used in the next section.

6. Representations

In {4}, the abstraction properties of types were characterized by a "representation" theorem. In this section we show that this theorem is a special case of the abstraction theorem.

For domains s_1 and s_2, a representation between s_1 and s_2 is a pair $\langle \phi, \psi \rangle$ of continuous functions such that

$$\begin{aligned} \phi &\in s_1 \rightarrow s_2, & \psi &\in s_2 \rightarrow s_1, \\ \psi \cdot \phi &\sqsupseteq \mathsf{I}_{s_1}, & \phi \cdot \psi &\sqsubseteq \mathsf{I}_{s_2}, \end{aligned}$$

where $\cdot$ denotes functional composition, i.e. $(\psi \cdot \phi)\ x = \psi(\phi\ x)$, and I_s denotes the identity function on s. (If the partial orderings s_1 and s_2 are regarded as categories, then a representation is an adjunction, with ϕ being the left adjoint of ψ.)

We write $\mathsf{Rel}(s_1, s_2)$ for the set of representations between s_1 and s_2. Domains and the representations between them form a category REP in which composition is $\langle \phi, \psi \rangle \cdot \langle \phi', \psi' \rangle = \langle \phi \cdot \phi', \psi \cdot \psi' \rangle$ and the identity representation for a domain s is $\mathsf{IP}(s) = \langle \mathsf{I}_s, \mathsf{I}_s \rangle$.

For $\langle \phi, \psi \rangle \in \mathsf{Rep}(s_1, s_2)$ and $\langle \phi', \psi' \rangle \in \mathsf{Rep}(s_1', s_2')$, we define

$$\begin{aligned} & \langle \phi, \psi \rangle \rightarrow \langle \phi', \psi' \rangle \in \mathsf{Rep}(s_1 \rightarrow s_1', s_2 \rightarrow s_2') \\ \textit{and} \quad & \langle \phi, \psi \rangle \times \langle \phi', \psi' \rangle \in \mathsf{Rep}(s_1 \times s_1', s_2 \times s_2') \\ \textit{by} \quad & \langle \phi, \psi \rangle \rightarrow \langle \phi', \psi' \rangle = \langle \Phi, \Psi \rangle \\ & \qquad \textit{where}\ \Phi\ f_1 = \phi' \cdot f_1 \cdot \psi \\ & \qquad\qquad \Psi\ f_2 = \psi' \cdot f_2 \cdot \phi \\ \textit{and} \quad & \langle \phi, \psi \rangle \times \langle \phi', \psi' \rangle = \langle \Phi, \Psi \rangle \\ & \qquad \textit{where}\ \Phi\ \langle x_1, x_1' \rangle = \langle \phi\ x_1, \phi' x_1' \rangle \\ & \qquad\qquad \Psi\ \langle x_2, x_2' \rangle = \langle \psi\ x_2, \psi' x_2' \rangle \end{aligned}$$

(Thus $\rightarrow$ and $\times$ are functors from $\mathsf{REP} \times \mathsf{REP}$ to REP.)

For domain assignments S_1 and S_2,

$$\prod_{\tau \in T} \mathsf{Rep}(S_1\tau, S_2\tau)$$

is the set of representation assignments between S_1 and S_2. If P is such an assignment then

$$P^{\#} \in \prod_{\omega \in \Omega} \mathsf{Rep}(S_1^{\#}\omega, S_2^{\#}\omega)$$

is such that

$$\textit{If}\ \kappa \in C\ \textit{then}\ P^{\#}\ \kappa = \mathsf{IP}(CS\ \kappa), \tag{P1}$$

$$\textit{If}\ \tau \in T\ \textit{then}\ P^{\#}\ \tau = P; \tau, \tag{P2}$$

$$\textit{If}\ \omega, \omega' \in \Omega\ \textit{then}\ P^{\#}(\omega \rightarrow \omega') = P^{\#}\omega \rightarrow P^{\#}\omega', \tag{P3}$$

$$\textit{If}\ \omega, \omega' \in \Omega\ \textit{then}\ P^{\#}(\omega \times \omega') = P^{\#}\omega \times P^{\#}\omega', \tag{P4}$$

and

$$P^{\#*} \in \prod_{\pi \in \Omega^*} \mathsf{Rep}(S_1^{\#*}\pi, S_2^{\#*}\pi)$$

is such that

$$\begin{aligned} P^{\#*}\pi &= \langle \Phi, \Psi \rangle \qquad (\mathrm{P}^*) \\ \textit{where}\ \Phi\ \eta_1\ v &= \phi_v(\eta_1 v) \\ \Psi\ \eta_2\ v &= \psi_v(\eta_2 v) \\ \textit{and}\ \langle \phi_v, \psi_v \rangle &= P^{\#}(\pi v). \end{aligned}$$

This "representation semantics" of type expressions is closely related to their order-relation semantics. For each $\langle \phi, \psi \rangle \in \mathsf{Rep}(s_1, s_2)$, we define $\overline{\langle \phi, \psi \rangle} \in \mathsf{Rel}(s_1, s_2)$ to be the complete relation such that

$$\langle x_1, x_2 \rangle \in \overline{\langle \phi, \psi \rangle}\ \textit{iff}\ x_1 \sqsubseteq_{s_1} \psi\ x_2,$$

or equivalently

$$\langle x_1, x_2 \rangle \in \overline{\langle \phi, \psi \rangle}\ \textit{iff}\ \phi\ x_1 \sqsubseteq_{s_2} x_2.$$

Then

$$\begin{aligned} & \overline{\mathsf{IP}(s)} = \ \sqsubseteq_s \ = \mathsf{I}(s), \\ & \overline{\langle \phi, \psi \rangle \rightarrow \langle \phi', \psi' \rangle} = \overline{\langle \phi, \psi \rangle} \rightarrow \overline{\langle \phi', \psi' \rangle}, \\ & \overline{\langle \phi, \psi \rangle \times \langle \phi', \psi' \rangle} = \overline{\langle \phi, \psi \rangle} \times \overline{\langle \phi', \psi' \rangle}, \end{aligned}$$

so that

$$\begin{aligned}&\textit{If } R\ \tau = \overline{P\ \tau} \textit{ for all } \tau \in T\\ &\textit{then } R^{\#}\omega = \overline{P^{\#}\omega} \textit{ for all } \omega \in \Omega^*\\ &\textit{and } R^{\#*}\pi = \overline{P^{\#*}\pi} \textit{ for all } \pi \in \Omega^*.\end{aligned}$$

Thus the abstraction theorem implies the

Representation Theorem. *Let P be a representation assignment between domain assignments S_1 and S_2. For all $\pi \in \Omega^*$, $\omega \in \Omega$, $e \in E_{\pi\omega}$, and $\langle \eta_1, \eta_2\rangle \in P^{\#*}\pi$,*

$$\langle \mu_{\pi\omega}[\![e]\!]\ S_1\ \eta_1, \mu_{\pi\omega}[\![e]\!]\ S_2\ \eta_2\rangle \in \overline{P^{\#}\omega}.$$

7. Polymorphic Functions

In {4} we proposed an extension of the typed lambda calculus in which the binding of type variables was introduced to permit the construction of polymorphic functions. (A similar but more general extension of the typed lambda calculus was developed independently in {14}, with the entirely different motivation of extending the connection between typed lambda calculus and intuitionistic logic.)

The basic idea is that an ordinary expression e of type ω can be abstracted on a type variable τ to give $\Lambda\tau.e$ of type $\Delta\tau.\omega$, which can be thought of as the type of polymorphic functions that, when applied to a type τ, yield a value of type ω. Then an ordinary expression p denoting a polymorphic function of type $\Delta\tau.\omega'$ can be applied to a type expression ω to give an ordinary expression $p[\omega]$ of type $\omega'/\tau \to \omega$. The intent is that $(\lambda\tau.e)[\omega]$ should have the same meaning as `lettype` $\tau = \omega$ `in` e, or equivalently as the result of substituting ω for τ in all type expressions occurring in e.

For example,

$$\Lambda\tau.\lambda f : \tau \to \tau.\lambda x : \tau.f(f(x))$$

denotes a polymorphic doubling function of type

$$\Delta\tau.(\tau \to \tau) \to (\tau \to \tau),$$

and

$$(\Lambda\tau.\lambda f : \tau \to \tau.\lambda x : \tau.f(f(x)))[\mathtt{Bool}]$$

has the same meaning as

$$\mathtt{lettype}\ \tau = \mathtt{Bool}\ \mathtt{in}\ \lambda f : \tau \to \tau.\lambda x : \tau.f(f(x))$$

or as

$$\lambda f : \mathtt{Bool} \to \mathtt{Bool}.\lambda x : \mathtt{Bool}.f(f(x)).$$

To make these ideas precise, we extend the syntax of type expressions by

$$\textit{If } \tau \in T \textit{ and } \omega \in \Omega \textit{ then } \Delta\tau.\omega \in \Omega. \qquad (\Omega 5)$$

The operator $\Delta\tau$ binds the occurrences of τ in ω, so that alpha conversion (i.e. renaming) is applicable to type expressions. We will regard alpha-variants of type expressions as identical and assume that the definition of substitution for type variables includes the use of alpha conversion to avoid collisions of type variables.

Then the syntax of ordinary expressions is extended by

$$\textit{If } e \in E_{\pi-\tau,\omega} \textit{ then } \Lambda\tau.e \in E_{\pi,\Delta\tau.\omega}, \qquad \text{(Ej)}$$

$$\textit{If } e \in E_{\pi,\Delta\tau.\omega'} \textit{ then } e[\omega] \in E_{\pi,(\omega'/\tau\to\omega)}. \qquad \text{(Ek)}$$

In (Ej), notice that $e \in E_{\pi-\tau,\omega}$ prevents e from containing free (though not bound) occurrences of any ordinary variables v whose type πv contains free occurrences of τ. This reflects the fact that the meaning of τ in $\Lambda\tau.e$ is different from its meaning in the surrounding context.

The concept of polymorphism was first recognized by Strachey {15}, who distinguished between "parametric" and "ad hoc" polymorphism. Intuitively, a parametric polymorphic function is one that behaves the same way for all types, while an ad hoc polymorphic function may have unrelated meanings for different types. For example, an ad hoc function might add integers, "or" Boolean values, and compose functions. In {4} our intention was to permit only parametric polymorphism. Thus, for example, the language provides no way of branching on types.

8. Semantics of Polymorphism

Several authors {5–9} have developed domain-theoretic semantics for the language described in the previous section, or for closely related languages. However, these models do not describe parametric polymorphism, since the domains associated with polymorphic types include ad hoc functions.

I am convinced that a satisfactory model should exclude ad hoc polymorphism. Moreover, based on the intuition that types are not limited to computation, I believe that, in the absence of recursive definitions of either values or types, it should be possible to give a set-theoretic semantics. This section and the next summarize the progress that has been made towards this goal. At present (March 1983) success has not been achieved, but the current results are encouraging.

The most naive definition one might give for the set associated with a polymorphic type is the collection of functions that accept sets and give values of the appropriate type. However, this collection is far too large to be a set. Moreover, it includes ad hoc functions.

Thus we define the set associated with a polymorphic type to be

$$\begin{aligned}&\textit{If } \tau \in T \textit{ and } \omega \in \Omega \textit{ then} \qquad (S5)\\ &\quad S^{\#}(\Delta\tau.\omega) = \{p \mid \mathrm{dom}(p) = \mathrm{SET}\\ &\qquad \textit{and } (\forall s \in \mathrm{SET})\ p(s) \in [S \mid \tau : s]^{\#}\omega\\ &\qquad \textit{and } \mathrm{parametric}_{S\tau\omega}(p)\},\end{aligned}$$

where SET is the class of all sets and $\mathrm{parametric}_{S\tau\omega}(p)$, to be defined later, excludes ad hoc functions. We hope that

this exclusion is so severe that $S^{\#}(\Delta\tau.\omega)$ is (isomorphic to) a set; whether this is the case for all ω is the central unsolved question about our proposed model.

The semantics of ordinary expressions is more straightforward:

$$\begin{aligned}&\textit{If } e \in E_{\pi-\tau,\omega} \textit{ then} && \text{(Mj)}\\ &\quad \mu_{\pi,\Delta\tau.\omega}[\![\Lambda\tau.e]\!]\ S\ \eta = p \in S^{\#}(\Delta\tau.\omega) \textit{ is such that}\\ &\quad p(s) = \mu_{\pi-\tau,\omega}[\![e]\!]\ [S \mid \tau : s]\ (\eta \restriction \mathsf{dom}(\pi-\tau)),\\ &\textit{If } e \in E_{\pi,\Delta\tau.\omega'} \textit{ then} && \text{(Mk)}\\ &\quad \mu_{\pi,(\omega'/\tau\to\omega)}[\![e[\omega]]\!]\ S\ \eta = \mu_{\pi,\Delta\tau.\omega'}[\![e]\!]\ S\ \eta\ (S^{\#}\omega).\end{aligned}$$

Next, we specify the relation semantics of $\Delta\tau.\omega$. The following definition preserves the validity of the abstraction theorem (and is essentially determined by this requirement):

$$\begin{aligned}&\textit{If } \tau \in T \textit{ and } \omega \in \Omega \textit{ then} && \text{(R5)}\\ &\quad R^{\#}(\Delta\tau.\omega) \in \mathsf{Rel}(S_1^{\#}(\Delta\tau.\omega), S_2^{\#}(\Delta\tau.\omega))\\ &\quad \textit{is the relation such that } \langle p_1, p_2\rangle \in R^{\#}(\Delta\tau.\omega) \textit{ iff}\\ &\quad (\forall s_1, s_2 \in \mathsf{SET})(\forall r \in \mathsf{Rel}(s_1, s_2))\\ &\qquad \langle p_1 s_1, p_2 s_2\rangle \in [R \mid \tau : r]^{\#}\omega.\end{aligned}$$

However, the identity extension lemma raises a problem. Let S be a set assignment and IA the relation assignment such that $IA\ \tau = \mathsf{I}(S\ \tau)$ for all $\tau \in T$. Then (R5) implies (taking r to be an identity relation) that $IA^{\#}(\Delta\tau.\omega)$ is a partial identity relation. But for $IA^{\#}(\Delta\tau.\omega)$ to be a total identity relation, all p in $S^{\#}(\Delta\tau.\omega)$ must satisfy

$$(\forall s_1, s_2 \in \mathsf{SET})(\forall r \in \mathsf{Rel}(s_1, s_2))\ \langle p\ s_1, p\ s_2\rangle \in [IA \mid \tau : r]^{\#}\omega.$$

Thus, if the identity extension lemma is to remain true, $\mathsf{parametric}_{S\tau\omega}$ must be at least as restrictive as the following definition:

$$\begin{aligned}&\mathsf{parametric}_{S\tau\omega}(p) = && \text{(PAR)}\\ &\quad (\forall s_1, s_2 \in \mathsf{SET})(\forall r \in \mathsf{Rel}(s_1, s_2))\ \langle p\ s_1, p\ s_2\rangle \in [IA \mid \tau : r]^{\#},\\ &\quad \textit{where } IA\ \tau = \mathsf{I}(S\ \tau) \textit{ for all } \tau \in T.\end{aligned}$$

However, we must be sure that this definition is permissible, i.e. that all ordinary expressions of polymorphic type denote functions that are parametric. Let S be a set assignment, $\pi \in \Omega^*$, $\tau \in T$, $\omega \in \Omega$, $e \in E_{\pi,\Delta\tau.\omega}$ and $\eta \in S^{\#*}\pi$. Then the identity extension lemma gives

$$\langle \eta, \eta\rangle \in IA^{\#*}\pi,$$

the abstraction theorem gives

$$\langle \mu_{\pi,\Delta\tau.\omega}[\![e]\!]\ S\ \eta, \mu_{\pi,\Delta\tau.\omega}[\![e]\!]\ S\ \eta\rangle \in IA^{\#}(\Delta\tau.\omega)$$

and (R5) gives

$$\begin{aligned}&(\forall s_1, s_2 \in \mathsf{SET})(\forall r \in \mathsf{Rel}(s_1, s_2))\\ &\quad \langle \mu_{\pi,\Delta\tau.\omega}[\![e]\!]\ S\ \eta\ s_1, \mu_{\pi,\Delta\tau.\omega}[\![e]\!]\ S\ \eta\ s_2\rangle \in [IA \mid \tau : r]^{\#}\omega,\end{aligned}$$

so that $\mathsf{parametric}_{S\tau\omega}(\mu_{\pi,\Delta\tau.\omega}\ S\ \eta)$ holds.

This is the essential link between abstraction and parametric polymorphism: The abstraction theorem guarantees that, in an environment in which all polymorphic functions are parametric, the meaning of any ordinary expression will be parametric.

We are currently investigating the question of whether $S^{\#}(\Delta\tau.\omega)$ is small enough to be a set, and have obtained an affirmative answer for "low-order" ω's without any embedded Δ's.

These results are obtained from (PAR), but would also hold for any more restrictive definition of $\mathsf{parametric}_{S\tau\omega}$ that was still permissible. They are all based on the algebraic nature of low-order types.

For nonnegative integers $n_1, \ldots, n_k$, the collection

$$S^{\#}(\Delta\tau.(\tau^{n_1} \to \tau) \times \cdots \times (\tau^{n_k} \to \tau) \to \tau)$$

is isomorphic to the carrier of the initial one-sorted algebra whose signature has operators $1, \ldots, k$ with arities $n_1, \ldots, n_k$.

As special cases of this result,

$$S^{\#}(\Delta\tau.\tau \times \tau \to \tau)$$

is isomorphic to a two-element set (e.g. of truth values) and

$$S^{\#}(\Delta\tau.(\tau \to \tau) \times \tau \to \tau)$$

is isomorphic to the set of natural numbers.

This result generalizes to multiple binders and occurrences of free type variables. Let

$$\Omega_0 = T$$

$$\Omega_{r+1} = T \cup \{\omega_1 \times \cdots \times \omega_k \to \tau \mid \omega_1, \ldots, \omega_k \in \Omega_r \textit{ and } \tau \in T\}$$

and

$$T_b = \tau_1, \ldots, \tau_N$$

be a finite subset of T. If $\omega \in \Omega_2$ then

$$S^{\#}(\Delta\tau_1 \ldots \Delta\tau_N.\omega)$$

is isomorphic to a set that can be defined algebraically.

Specifically, ω can be written in the form

$$\omega_1 \times \cdots \times \omega_m \times \omega_1' \times \cdots \times \omega_n' \to \tau$$

where each ω_i has the form τ or $\cdots \to \tau$ for some $\tau \in T_b$ and each ω_i' has the form τ or $\cdots \to \tau$ for some $\tau \in T - T_b$. Let Σ be the many-sorted signature with sorts T and operators $1, \ldots, m$ of arities $\omega_1, \ldots, \omega_m$. Let S_1 be the set assignment mapping $\tau \in T_b$ into $\{\}$ and $\tau \in T - T_b$ into S_τ, let F be the free Σ-algebra generated by S_1, and let S_2 be the set assignment mapping $\tau \in T$ into the τ-component of the carrier of F. Then $S^{\#}(\Delta\tau \ldots \Delta\tau_N.\omega)$ is isomorphic to

$$S_2^{\#}(\omega_1' \times \cdots \times \omega_n' \to \tau).$$

Interesting special cases include

$$S^{\#}(\Delta\tau.\tau \to \alpha) \simeq S^{\#}\alpha,$$

$$S^{\#}(\Delta\tau.(\alpha \to \tau) \to \tau) \simeq S^{\#}\alpha,$$

$$S^{\#}(\Delta\tau.(\alpha \to \tau) \times (\beta \to \tau) \to \tau) \simeq S^{\#}\alpha + S^{\#}\beta,$$
$$S^{\#}(\Delta\tau.(\alpha \times \tau \to \tau) \times \tau \to \tau) \simeq (S^{\#}\alpha)^{*},$$

where α and β are free type variables, $\simeq$ denotes isomorphism, + denotes disjoint union, and s^* denotes the set of finite sequences of members of s. (The last isomorphism was suggested by a functional encoding of lists devised by C. Böhm.)

We conjecture that, when $\Delta\tau_1 \ldots \Delta\tau_N.\omega$ is closed and ω contains no Δ's, $S^{\#}(\Delta\tau_1 \ldots \tau_N.\omega)$ is isomorphic to the subset of $E_{\langle\rangle,\omega}$ (where $\langle\rangle$ is the empty type assignment) whose members are in normal form. However, even if this conjecture, or the more general conjecture that $S^{\#}\omega$ is always isomorphic to some set, is false, a set-theoretic semantics of polymorphism may still be viable. The key may be to formulate a more general abstraction theorem and resulting definition of parametric$_{S\tau\omega}$. This possibility is explored in the next section.

9. A More General Abstraction Theorem

There are two ways in which the abstraction theorem can be generalized. First, binary relations can be replaced by multinary relations. Second, and less trivially, individual relations can be replaced by families of relations of the kind used by Kripke {16} to model intuitionistic logic.

These generalizations are suggested by recent work by G. Plotkin {13}. Proposition 1 in {13} is essentially the generalization of our abstraction theorem to multinary relations, and Proposition 2 is the further generalization to Kripke-like families of relations. Actually, these propositions deal with the pure typed lambda calculus and assume a fixed arbitrary set assignment, but their extension to our illustrative language and to varying set assignments is straightforward.

Moreover, Plotkin showed that his Proposition 2 completely characterizes the meanings of the typed lambda calculus, in the sense that every meaning satisfying this proposition is the meaning of some ordinary expression. This result encourages the hope that his generalization will lead to a definition of parametric$_{S\tau\omega}$ so restrictive that the collections of parametric polymorphic functions will be sets.

To generalize from binary to multinary relations, we assume we are given a fixed index set I of arbitrary cardinality. If $\mathcal{s} = \langle \mathcal{s}_i \mid i \in I\rangle$ is an I-indexed family of sets, then

$$\mathsf{Rel}(\mathcal{s}) = \{r \mid r \subseteq \prod_{i \in i} \mathcal{s}_i\}$$

is the set of I-ary relations among the $\mathcal{s}_i$. For a set s, the I-ary identity relation on s is

$$\mathsf{I}(s) = \{\langle x \mid i \in I\rangle \mid x \in s\} \in \mathsf{Rel}(\langle s \mid i \in I\rangle).$$

To generalize further to Kripke semantics, we assume we are given a fixed partial ordering W (of "alternative worlds"), and we define a *Kripke relation* among the $\mathcal{s}_i$ to be a function from W to $\mathsf{Rel}(\mathcal{s})$ that is monotone when $\mathsf{Rel}(\mathcal{s})$ is ordered by inclusion. Thus

$$\mathsf{Krel}(\mathcal{s}) = \{f \mid f \in W \xrightarrow{\text{mon}} \mathsf{Rel}(\mathcal{s})\}$$

is the set of Kripke relations among the $\mathcal{s}_i$. For a set s, the identity Kripke relation on s is $IK(s) \in \mathsf{Krel}(\langle s \mid i \in I\rangle)$ such that

$$IK(s)\ w = \mathsf{I}(s) \textit{ for all } w \in W.$$

For $k \in \mathsf{Krel}(\mathcal{s})$ and $k' \in \mathsf{Krel}(\mathcal{s}')$ we define

$$k \to k' \in \mathsf{Krel}(\langle \mathcal{s}_i \to \mathcal{s}'_i \mid i \in I\rangle)$$

to be the Kripke relation such that

$$f \in (k \to k')\ w \textit{ iff}$$
$$(\forall w' \geq w)(\forall x \in k\ w')\ \langle f_i x_i \mid i \in I\rangle \in k'w',$$

and

$$k \times k' \in \mathsf{Krel}(\langle \mathcal{s}_i \times \mathcal{s}'_i \mid i \in I\rangle)$$

to be the Kripke relation such that

$$\langle\langle x_i, x'_i\rangle \mid i \in I\rangle \in (k \times k')\ w \textit{ iff } x \in k\ w \textit{ and } x' \in k'w.$$

If $S = \langle S_i \mid i \in I\rangle$ is an I-indexed family of set assignments, then

$$\prod_{\tau \in T} \mathsf{Krel}(\langle S_i\tau \mid i \in I\rangle)$$

is the set of *Kripke relation assignments* among the S_i. If K is such an assignment then

$$K^{\#} \in \prod_{\omega \in \Omega} \mathsf{Krel}(\langle S_i^{\#}\omega \mid i \in I\rangle)$$

is such that

$$\textit{If } \kappa \in C \textit{ then } K^{\#}\ \kappa = IK(CS\ \kappa), \tag{K1}$$
$$\textit{If } \tau \in T \textit{ then } K^{\#}\ \tau = K\ \tau, \tag{K2}$$
$$\textit{If } \omega, \omega' \in \Omega \textit{ then } K^{\#}(\omega \to \omega') = K^{\#}\omega \to K^{\#}\omega', \tag{K3}$$
$$\textit{If } \omega, \omega' \in \Omega \textit{ then } K^{\#}(\omega \times \omega') = K^{\#}\omega \times K^{\#}\omega', \tag{K4}$$
$$\textit{If } \tau \in T \textit{ and } \omega \in \Omega \textit{ then} \tag{K5}$$
$$K^{\#}(\Delta\tau.\omega) \in \mathsf{Krel}(\langle S_i^{\#}(\Delta\tau.\omega) \mid i \in I\rangle)$$
$$\textit{is the Kripke relation such that}$$
$$p \in K^{\#}(\Delta\tau.\omega) \textit{ iff } (\forall \mathcal{s} \in \mathsf{SET}^I)(\forall k \in \mathsf{Krel}(\mathcal{s}))$$
$$\langle p_i \mathcal{s}_i \mid i \in I\rangle \in [K \mid \tau : k]^{\#}\omega\ w,$$

and

$$K^{\#*} \in \prod_{\pi \in \Omega^*} \mathsf{Krel}(\langle S_i^{\#*}\pi \mid i \in I\rangle)$$

is such that

$$\eta \in K^{\#*}\pi\ w \textit{ iff} \tag{K*}$$
$$(\forall v \in \mathrm{dom}\ \pi)\ \langle \eta_i v \mid i \in I\rangle \in K^{\#}(\pi v)\ w.$$

Then we have the

Generalized Abstraction Theorem. *Let K be a Kripke relation assignment among set assignments S_i. For all $\pi \in \Omega^*$, $\omega \in \Omega$, $e \in E_{\pi\omega}$, $w \in W$, and $\eta \in K^{\#*}\pi\ w$,*

$$\langle \mu_{\pi\omega}[\![e]\!]\ S_i\ \eta_i \mid i \in I\rangle \in K^{\#}\omega\ w.$$

Moreover, the

Generalized Identity Extension Lemma. *Suppose that IA is a Kripke relation assignment such that $IA\ \tau = IK(S\ \tau)$ for all $\tau \in T_0 \subseteq T$. Then*

$$IA^{\#}\omega = IK(S^{\#}\omega) \text{ for all } \omega \text{ such that } \mathsf{F}(\omega) \subseteq T_0, \text{ and}$$
$$IA^{\#*}\pi = IK(S^{\#*}\pi) \text{ for all } \pi \text{ such that}$$
$$(\forall v \in \mathsf{dom}\ \pi)\ \mathsf{F}(\pi v) \subseteq T_0,$$

will hold if $\mathsf{parametric}_{S\tau\omega}$ is defined by

$$\mathsf{parametric}_{S\tau\omega}(p) = (\forall \mathcal{I} \in \mathsf{SET}^I)(\forall k \in \mathsf{Krel}(\mathcal{I}))(\forall w \in W)$$
$$\langle p_i \mathcal{I}_i \mid i \in I\rangle \in [IA \mid \tau : k]^{\#}\omega\ w,$$
$$\text{where } IA\ \tau = IK(S\ \tau) \text{ for all } \tau \in T,$$

and the generalized abstraction theorem shows that this definition is permissible.

10. Existential Type Quantification

In {14}, J.-Y. Girard proposed an extension of the typed lambda calculus based on an analogy with intuitionistic logic, in which a type expression is interpreted as a proposition, and an ordinary expression as a proof of the proposition that is its type. Specifically, the type operators $\rightarrow$, $\times$, and $+$ (disjoint union) are interpreted as implication, conjunction, and disjunction. This analogy led Girard to introduce new type operations corresponding to universal and existential quantification. His universal quantifier $\forall$ coincides with our Δ, but his existential quantifier $\exists$ goes beyond the language proposed in {4}.

Gordon Plotkin and John Mitchell have recently suggested that existential types capture the idea of encapsulating primitive operations with abstract types; essentially an object of type $\exists\tau.\omega_p$ consists of a representation type ω along with a primitive operation of type $\omega_p/\tau \rightarrow \omega$.

The following is a syntactic description that recasts their ideas in the notation of this paper: The syntax of type expressions is extended by

$$\text{If } \tau \in T \text{ and } \omega \in \Omega \text{ then } \exists\tau.\omega \in \Omega, \tag{Ω6}$$

where $\exists\tau$ binds the occurrences of τ in ω (and alpha-variants are regarded as identical). Then the syntax of ordinary expressions is extended by

$$\text{If } \pi \in \Omega^*, \tau \in T, \omega, \omega_p \in \Omega, \text{ and } e_p \in E_{\pi,(\omega_p/\tau\rightarrow\omega)} \tag{El}$$
$$\text{then } \langle \omega, e_p : \exists\tau.\omega_p\rangle \in E_{\pi,\exists\tau.\omega_p},$$
$$\text{If } \pi \in \Omega^*, \tau \in T, v_p \in V, \omega', \omega_p \in \Omega, \tag{Em}$$
$$e' \in E_{\pi,\exists\tau.\omega_p}, e \in E_{[\pi-\tau|v_p:\omega_p],\omega'}, \text{ and } \tau \notin \mathsf{F}(\omega')$$
$$\text{then } \mathtt{abstract}\ \tau, v_p = e'\ \mathtt{in}\ e \in E_{\pi\omega'}.$$

In (El), notice that τ and ω_p must occur explicitly in $\langle \omega, e_p : \exists\tau.\omega_p\rangle$ to ensure that this expression has a unique type.

The intent is that, for $\pi \in \Omega^*$, $\tau \in T$, $v_p \in V$, $\omega, \omega', \omega_p \in \Omega$, $e_p \in E_{\pi,(\omega_p/\tau\rightarrow\omega)}$, $e \in E_{[\pi-\tau|v_p:\omega_p]}$, and $\tau \notin \mathsf{F}(\omega')$,

$$\mathtt{abstract}\ \tau, v_p = \langle \omega, e_p : \exists\tau.\omega_p\rangle\ \mathtt{in}\ e \tag{$*$}$$

should have the same meaning as

$$\mathtt{lettype}\ \tau = \omega\ \mathtt{with}\ v_p : \omega_p = e_p\ \mathtt{in}\ e.$$

As discussed in Sections 4 and 7, this `lettype` expression has the alternative desugaring

$$(\Lambda\tau.\lambda v_p : \omega_p.e)[\omega](e_p) \tag{$**$}$$

(which is actually more general, since the restriction $\tau \notin \mathsf{F}(\omega')$ can be dropped). But the present approach embodies the idea of "packaging" the representation type ω with the implementation of the primitive operation (or tuple of primitive operations) e_p.

The semantics of $\exists$ is no harder (or easier) than that of Δ, since the existential constructs can be defined in terms of the universal ones. The essential idea, suggested to the author by D. Leivant, is that $\exists\tau.\omega$ should have the same meaning as $\Delta\tau'.(\Delta\tau.(\omega \rightarrow \tau')) \rightarrow \tau'$, where τ' is distinct from τ and does not occur free in ω. Thus we define

$$\text{If } \tau \in T \text{ and } \omega \in \Omega \text{ then} \tag{S6}$$
$$S^{\#}(\exists\tau.\omega) = S^{\#}(\Delta\tau'.(\Delta\tau.(\omega \rightarrow \tau')) \rightarrow \tau')$$
$$\text{where } \tau' \neq \tau \text{ and } \tau' \notin \mathsf{F}(\omega),$$

and similarly for $R^{\#}$ and $K^{\#}$. Then the semantics of the new constructions for ordinary expressions is:

$$\text{If } \pi \in \Omega^*, \tau \in T, \omega, \omega_p \in \Omega, \text{ and} \tag{Ml}$$
$$e_p \in E_{\pi,(\omega_p/\tau\rightarrow\omega)} \text{ then}$$
$$\langle \omega, e_p : \exists\tau.\omega_p\rangle$$
$$\text{has the same meaning as}$$
$$\Lambda\tau'.\lambda p : \Delta\tau.(\omega_p \rightarrow \tau').p[\omega](e_p)$$
$$\text{where } p \notin \mathsf{dom}\ \pi, \tau' \neq \tau, \text{ and } \tau' \text{ does not occur free in } \omega_p, \omega,$$
$$\pi v \text{ for any } v \in \mathsf{dom}\ \pi, \text{ or any type expression occurring in } e_p.$$

$$\text{If } \pi \in \Omega^*, \tau \in T, v_p \in V, \omega', \omega_p \in \Omega, e' \in E_{\pi,\exists\tau.\omega_p}, \tag{Mm}$$
$$e \in E_{[\pi-\tau|v_p:\omega_p],\omega'}, \text{ and } \tau \notin \mathsf{F}(\omega') \text{ then}$$
$$\mathtt{abstract}\ \tau, v_p = e'\ \mathtt{in}\ e$$
$$\text{has the same meaning as}$$
$$e'[\omega'](\Lambda\tau.\lambda v_p : \omega_p.e).$$

Then ($*$) has the same meaning as

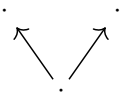
$$(\Lambda\tau'.\lambda p : \Delta\tau.(\omega_p \rightarrow \tau').p[\omega](e_p))[\omega'](\Lambda\tau.\lambda v_p : \omega_p.e),$$

which reduces to ($**$).

11. Future Directions

Beyond the central problem of determining if $S^{\#}(\Delta\tau.\omega)$ is always a set, the following are promising directions for further research:

(a) Connections with category theory should be explored. For example, binary relations can be made into a Cartesian closed category by defining suitable morphisms between relations. Moreover, this category is (isomorphic to) a full subcategory of the Cartesian closed category SET^{Σ}, where Σ is the partial ordering



A similar story can probably be told about Kripke relations.

(b) To deal with recursive definitions, of both values and types, a domain-theoretic model of parametric polymorphism should be sought. While several domain-theoretic models of polymorphism have been proposed {5–9}, none of them provides domains containing only parametric functions.

(c) Polymorphism and its models should be extended to encompass some concept of subtype, including noninjective implicit conversions, as discussed in {17}.

(d) The theory should be applied to imperative Algol-like languages. In this connection, the distinction, made in {18} and {19}, between data types and phrase types is germane. We have recently started exploring the practical potential of phrase-type definitions for data representation structuring in the sense of {20, Chapter 5}.

Acknowledgement

I am grateful to Gordon Plotkin and Daniel Leivant for their contagious enthusiasm and patient explanations, as well as numerous specific ideas.

References

[1] Barbara Liskov, Russell R. Atkinson, Toby Bloom, Eliot B. Moss, J. Craig Schaffert, Robert Schaffert, and Alan Snyder. CLU Reference Manual. *Lecture Notes in Computer Science*, 114, 1979.

[2] William A. Wulf, Ralph L. London, and Mary Shaw. Abstraction and Verification in Alphard: Introduction to Language and Methodology. Technical report, USC Information Sciences Institute, Los Angeles, 1976.

[3] James H. Morris. Types are not Sets. In *Proceedings of the 1st ACM Symposium on Principles of Programming Languages*, pages 120–124, 1973.

[4] John C. Reynolds. Towards a Theory of Type Structure. In *Programming Symposium: Proceedings, Colloque sur la Programmation*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425, New York, 1974. Springer-Verlag.

[5] Nancy J. McCracken. *An Investigation of a Programming Language with a Polymorphic Type Structure.* Ph. d. dissertation, Syracuse University, June 1979.

[6] Nancy J. McCracken. A Finitary Retract Model for the Polymorphic Lambda-Calculus. Technical report, Syracuse University, 1982. Submitted to *Information and Control.*

[7] D. B. MacQueen and Ravi Sethi. A Semantic Model of Types for Applicative Languages. In *Proceedings of the 1982 ACM Symposium on LISP and Functional Programming*, pages 243–252, 1982.

[8] Christopher T. Haynes. A Theory of Data Type Representation Independence. Technical Report 82-04, Department of Computer Science, University of Iowa, December 1982.

[9] Daniel Leivant. Structural Semantics for Polymorphic Data Types (Preliminary Report). In *Proceedings of the 10th ACM Symposium on Principles of Programming Languages*, pages 155–166, January 1983.

[10] J. V. Guttag and J. J. Horning. The Algebraic Specification of Abstract Data Types. *Acta Informatica*, 10(1):27–52, 1978.

[11] Joseph A. Goguen, James W. Thatcher, and Eric G. Wagner. Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types. *Current Trends in Programming Methodology*, 4: Data Structuring, 1978.

[12] Deepak Kapur. Towards a Theory for Abstract Data Types. Technical Report TR-237, Laboratory for Computer Science, MIT, May 1980.

[13] Gordon D. Plotkin. Lambda-Definability in the Full Type Hierarchy. In Jonathan P. Seldin and James R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 363–373. Academic Press, 1980.

[14] Jean-Yves Girard. *Interprétation Fonctionelle et Élimination des Coupures dans l'Arithmétique d'Ordre Supérieur.* Thèses de Doctorat d'État, Université Paris VII, Paris, 1972.

[15] Christopher Strachey. Fundamental Concepts in Programming Languages, August 1967. Lecture Notes, International Summer School in Computer Programming, Copenhagen.

[16] Saul A. Kripke. Semantical Analysis of Intuitionistic Logic I. In *Formal Systems and Recursive Functions, Proceedings of the 8th Logic Colloquium, Oxford, 1963*, Amsterdam, 1965. North-Holland.

[17] John C. Reynolds. Using Category Theory to Design Implicit Conversion and Generic Operators. In N. D. Jones, editor, *Semantics-Directed Compiler Generation, Proceedings of a Workshop*, volume 94 of *Lecture Notes in Computer Science*, New York, January 1980. Springer-Verlag.

[18] John C. Reynolds. The Essence of Algol. In J. W. de Bakker and J. C. van Vliet, editors, *Algorithmic Languages*, pages 345–372. North-Holland, Amsterdam, 1981.

[19] Frank J. Oles. *A Category-Theoretic Approach to the Semantics of Programming Languages.* Ph. d. dissertation, Syracuse University, August 1982.

[20] John C. Reynolds. *The Craft of Programming.* Prentice-Hall International, London, 1981.