Discrete Polymorphism

Summary

Daniel Leivant
School of Computer Science
Carnegie-Mellon University

1 Introduction

Polymorphic typing comes in two flavors: parametric — as in ML and in Girard-Reynolds's second order λ-calculus 2λ, and discrete — as in Forsyth [Rey88] and Coppo-Dezani's λ-calculus with intersection types, $J\lambda$. At first blush, intersection types might look as the ultimate form of discrete polymorphism, since all normalizable λ-expressions are (suitably) typable. However, the typings obtained in $J\lambda$ for normalizable λ-expressions may be highly non-uniform. For instance, if P is a λ-expression representing a unary numeric function f, then $P\bar{n}$ will be typable for every Church numeral $\bar{n}$, but there may be no type τ common to *all* numerals and such that P has a type of the form $\tau \to \cdots$ (see Theorem 14 below). Thus, while intersection-types serve well as guarantor of functional well-behavior, for each individual input, they are not all-powerful as compile-time checks for the functional well-behavior of procedures as a whole.

We consider here a discipline $J^\infty\lambda$ with *infinite* type intersection, for which we outline a mathematical theory. This has several advantages:

1. Conceptually, $J^\infty\lambda$ conveys more fully than $J\lambda$ the intuitive idea of multiple typing.
2. The potential for uniform typing is far greater in $J^\infty\lambda$ than in $J\lambda$.
3. $J^\infty\lambda$ is a natural master formalism in which several typing paradigms, such as stratified quantificational polymorphism, can be interpreted and related.
4. One important rationale for typing, the absence of infinite regression in evaluating recursion-free procedures, is preserved: all typed expressions are strongly normalizable.
5. $J^\infty\lambda$ has the (literal) subtype property: every derived typing for a normal expression can be derived using only subtypes of the derived typing. This property fails for 2λ, even for its stratified variant.
6. Although the types and type derivations of $J^\infty\lambda$ are infinite, significant fragments have effective, finite, and transparent representations. Its infinite nature is conceptual, and is not *per se* an impediment to implementation.
7. The discipline naturally stratifies into subdisciplines, allowing type formation only up to certain levels. This potentially permits striking a balance between programming needs and efficiency of implementation.
8. $J^\infty\lambda$ is computationally far more interesting than $J\lambda$. We establish natural relations between the size of types allowed and the computational nature of the representable functions. Note that functions are represented here by *finite* λ-expressions: infinite are only the typing properties of these expressions.

Acknowledgment: I am grateful to Robert Harper and Benjamin Pierce for comments on an early draft. Research partially supported by ONR grant N00014-84-K-0415 and by DARPA grant F33615-81-K-1539.

2 Discrete polymorphic type disciplines

In this section we describe a semantic (i.e. implicit) calculus for a typed λ-calculus with infinite type intersection. We start with a description of semantic-style disciplines for the simply typed, the finite-intersection typed, and the parametrically polymorphic λ-calculi, providing common grounds for comparisons and interpretations.

2.1 Finite type intersection

Discrete polymorphism in programming languages was recognized already by Strachey [Str67], who referred to it as "ad hoc" polymorphism, in the sense that it permits a polymorphic function or procedure to have different, unrelated, behaviors at different types, as in type overloading of function identifiers. A partial mathematical rendition of the concept was given by Coppo and Dezani in their typed λ-calculus with type intersection, $J\lambda$, which extends the simply typed (first order) λ-calculus 1λ [Cop80,CD80].

1λ captures the essence of type disciplines of algol-like languages. The simple types and their ranks are defined recursively: o is a type of rank 0; if τ is a type of rank t and σ is a type of rank s, then $\tau \to \sigma$ is a type of rank $\max(1+t, s)$. We associate $\to$ to the right, so $\tau \to \sigma \to \rho$ abbreviates $\tau \to (\sigma \to \rho)$.

A *type statement* is an expression $E : \tau$, where τ is a type and E is a λ-expression. An *environment* is a finite

function η from λ-variables to types. We write $\eta, x:\tau$ for $\eta \cup \langle x, \tau \rangle$. A *typing* is an expression $\eta \vdash E : \tau$, where η is an environment. $\vdash E : \tau$ stands for $\emptyset \vdash E : \tau$.

The *derived typings* are generated by the *type inference rules:*

Initial: If $\eta(x) = \tau$ then $\eta \vdash x : \tau$ is derived.

$\rightarrow$-Intro: If $\eta, x : \sigma \vdash E : \tau$ is derived, then so is $\eta \vdash (\lambda x.E) : (\sigma \rightarrow \tau)$.

$\rightarrow$-Elim: If $\eta \vdash E : (\sigma \rightarrow \tau)$ and $\eta \vdash F : \sigma$ are derived, then so is $\eta \vdash (EF) : \tau$.

A *type derivation* can be represented as a finite tree Δ of typings where each node is derived from its children by one of the inference rules above, and the root $\eta \vdash E : \tau$ of Δ is the derived typing. We write $exp(\Delta) =_{df} E$, $type(\Delta) =_{df} \tau$.

$\mathbf{J}\lambda$ is an extension $\mathbf{1}\lambda$. An additional clause in the definition of types is: if τ and σ are types, then so is $\tau \wedge \sigma$ (the *intersection* or *meet* of τ and σ).[1] Additional inference rules are:

$\wedge$-Intro: If $\eta \vdash E : \tau$ and $\eta \vdash E : \sigma$ are derived, then so is $\eta \vdash E : \tau \wedge \sigma$.

$\wedge$-Elim: If $\eta \vdash E : \tau \wedge \sigma$ is derived, then so are $\eta \vdash E : \tau$ and $\eta \vdash E : \sigma$.

$\mathbf{J}\Omega\lambda$ is an extension of $\mathbf{J}\lambda$ with a designated type identifier Ω, and the inference rule:

Univ: $\eta \vdash E : \Omega$ is derived (for any η and E).

A remarkable property of intersection types is that they characterize computational properties of λ-expressions. Call a type τ *proper* [*p-proper*] {*sp-proper*} if Ω has no [positive] {strictly-positive} occurrence therein. A type derivation is proper if all types therein are proper. Then a λ-expression is strongly normalizable iff it has a proper type derivation, it is normalizable iff it has a derived p-proper typing, and it is solvable iff it has a derived sp-proper typing ([CDV81], or see [Lei86]).

2.2 Parametric polymorphism

The standard bearer of parametric polymorphism is the Girard-Reynolds second order λ-calculus, $\mathbf{2}\lambda$ [Gir72, Rey74]. One posits a denumerable supply TV of *type variables*. Types are generated as in $\mathbf{1}\lambda$, with the added clause: if τ is a type and $t \in TV$, then $\forall t.\tau$ is a type. The inference rules of $\mathbf{1}\lambda$ are augmented with:

$\forall$-Intro: If $\eta \vdash E : \tau$ is derived, and $t \in TV$ is not free in the range of η, then $\eta \vdash E : \forall t.\tau$ is derived.

$\forall$-Elim: If $\eta \vdash E : \forall t.\tau$ is derived, and σ is free for t in τ (in the usual sense), then $\eta \vdash F : \tau[\sigma/t]$ is derived. ($\tau[\sigma/t]$ is the result of simultaneously substituting σ for all free occurrences of t in τ.) t and σ are the *eigen-variable* and the *eigen-type* of the inference, respectively.

[1] The phrase "type conjunction" may be misleading, because the type corresponding to conjunction rules, under the Curry-Howard isomorphism, is the cartesian product of types, not type intersection.

Type quantification in $\mathbf{2}\lambda$ is "impredicative", in that $\forall$ ranges over all types, including the ones with occurrences of $\forall$. This self referential nature of type quantification implies that the functions representable by λ-expressions typed in $\mathbf{2}\lambda$ include all the provably recursive functions of full (impredicative) second order arithmetic [Gir72], and that the discipline has no set-theoretic models [Rcy84, RP88].

A predicative variant $\mathbf{P2}\lambda$ of $\mathbf{2}\lambda$ is obtained by successively expanding the collection of type expressions, starting with the types of $\mathbf{1}\lambda$. Let $TV_0, TV_1, \ldots$ be disjoint denumerable sets of identifiers; the elements of TV_n are the *level n type variables*. For $n \geq 0$, the set T_n of *level n types* is defined like the set of types of $\mathbf{2}\lambda$, except that free type variables must be of level $\leq n$ and bound ones of level $< n$. The inference rules are as for $\mathbf{2}\lambda$, except that $\forall$-Elim is allowed only when the eigen-type is in T_n, where n is the level of the eigen-variable. $\mathbf{P}^k\mathbf{2}\lambda$ is the fragment of $\mathbf{P2}\lambda$ with $\forall$-Elim further limited to eigen-types in T_k. In particular, $\mathbf{P}^0\mathbf{2}\lambda$ allows $\forall$-Elim only for first order eigen-types. $\mathbf{P2}\lambda$ was defined and studied in [Sta81,Lei89]. Transfinite extensions $\mathbf{P}^\alpha\mathbf{2}\lambda$ of $\mathbf{P2}\lambda$, for countable ordinals α as levels, are considered in [Lei89].

2.3 Infinite type intersection

The idea that an object can have different "concrete" (denotable) types can be more fully formalized in a discipline $\mathbf{J}^\infty\lambda$, in which arbitrarily long type intersections are allowed. The types of $\mathbf{J}^\infty\lambda$ are generated as in $\mathbf{1}\lambda$ with the added clause: if T is a countable (finite or infinite) set of types, then so is $\bigwedge T$. (When $T = \{\tau_\iota | \iota \in I\}$, we also write $\bigwedge_{\iota \in I} \tau_\iota$ for $\bigwedge T$.) For example, $\bigwedge T_1$, where T_1 is the set of types of $\mathbf{1}\lambda$, is a legitimate type.

The inference rules are those of $\mathbf{1}\lambda$, augmented with:

$\bigwedge$-Intro: If $\eta \vdash E : \tau$ is derived for every $\tau \in T$, then so is $\eta \vdash E : \bigwedge T$.[2]

$\bigwedge$-Elim: If $\eta \vdash E : \bigwedge T$ is derived, then so is $\eta \vdash E : \tau$, for each $\tau \in T$.

A *type derivation* is a well-founded tree of typings (i.e. every branch terminates), where each node is derived from its children by one of the inference rules, as above.

Well-founded countably-branching trees T, like the types and derivations of $\mathbf{J}^\infty\lambda$, can be calibrated by countable ordinals, $\alpha(T)$: if T is a singleton, then $\alpha(T) =_{df} 0$; if the immediate subtrees of T are T_i, $i < n \leq \omega$, then $\alpha(T) =_{df} \sup_{i<n}(\alpha(T_i) + 1)$. The definition of type rank is extended: $rank(o) =_{df} 0$, $rank(\sigma \rightarrow \rho) =_{df} \max(rank(\rho), rank(\sigma) + 1)$, $rank(\bigwedge T) =_{df} \sup_{\tau \in T} rank(\tau)$. We denote by $\mathbf{J}^\alpha\lambda$ the fragment of $\mathbf{J}^\infty\lambda$ that allows only types of rank $\prec \alpha$. Thus $\mathbf{J}^\omega\lambda$ is a notational variant of $\mathbf{J}\lambda$.

It will be convenient to consider two additional inference rules, for β-expansion, and for repeated typings:

β-Exp: If $\eta \vdash E[F/x] : \tau$ is derived, then so is $\eta \vdash (\lambda x.E)F : \tau$.

Rep: The conclusion is the same as the premise.

We say that a type derivation with possible uses of [Exp] {Rep} is a [β-]{r-}derivation.

[2] The infinite $\bigwedge$-intro rule is unrelated to ω-rules for the λ-calculus (see [Bar81],§4.1.10)), as it leaves the λ-expression unaltered.

The repetition rule allows a streamlined treatment of normalization of infinite derivation, and was used for the same purpose in [Min78]. The β-expansion rule is needed to permit an uncomplicated description of arrow reductions (see §4.1 below). A β-(r-)derivation Δ is *acceptable* if there is a bound on the size of λ-expressions therein. [3] *exp-size*(Δ) will then denote the largest size of λ-expressions in Δ.

3 Basic properties of the calculus of infinite type intersection

3.1 Finitary representation

Though the types and derivations of $\mathbf{J}^\infty\lambda$ are generally infinite, the ones of interest are easily represented by finitary descriptions. First, we may restrict attention to recursively described types and type derivations, as follows. Types and derivations of $\mathbf{J}^\infty\lambda$ are syntactic trees, i.e. functions D that assign to certain elements of $\mathbf{N}^*$ (the *nodes* of the universal spread) syntactic objects: type identifiers, $\rightarrow$ and $\bigwedge$ for finite types, type statements for derivations, or a flag $\perp$ for unused nodes. These functions are numeric, once given a canonical numeric codings of $\mathbf{N}^*$ and of the syntactic objects considered. Then functions D as above can be converted into numeric functions. We say that a type or derivation is recursive, primitive recursive, etc., if the numeric function representing it as above is. If $\mathcal{C}$ is a class of functions, and λ an infinitary type discipline, then we write $\lambda(\mathcal{C})$ for the fragment of λ in which all infinite syntactic objects are coded as functions in $\mathcal{C}$.

It is also easy to set up simple formal languages for describing infinite types and type derivations. Here is one which suffices for all useful types of order $< \omega^\omega$. We generate inductively a set ST of *set-types description terms*. We write $V(e)$ for the set denoted by e, given an assignment of sets of types to the free variables in e.

- There is a denumerable set of type-set variables $\theta_1, \theta_2, \ldots$ in ST.
- $O \in ST$, with $V(O) = \{o\}$.

If $e, e' \in ST$ then:

- $e \cup e' \in ST$, and $V(e \cup e') = V(e) \cup V(e')$.
- $(e \rightarrow e') \in ST$, and $V(e \rightarrow e') = \{\tau \rightarrow \tau' \mid \tau \in V(e), \tau' \in V(e')\}$.
- $e[e'] \in ST$, and $V(e[e']) = \{\tau[\tau'/o] \mid \tau \in V(e), \tau' \in V(e')\}$.
- $\wedge(e, e') \in ST$, and $V(\wedge(e, e')) = \{\bigwedge\{\tau[\tau'/o] \mid \tau' \in V(e')\} \mid \tau \in V(e)\}$.
- $\kappa\theta_1 \ldots \theta_k.\, e \in ST$, and $V(\kappa\theta_1 \ldots \theta_k.\, e) = \cup_i V(d_i)$, where $d_0 =_{\mathrm{df}} O$, $d_{i+1} = \kappa\theta_2 \ldots \theta_k.e[d_i]$. Thus κ is an ω-closure operator[4].

For example, $V(\kappa\theta.\theta) = \{o\}$; $V(\kappa\theta.(\theta \rightarrow \theta)) = \{\tau_i\}_i$, where $\tau_0 = (o \rightarrow o)$, and $\tau_{i+1} = (\tau_i \rightarrow \tau_i)$; if $e_0 =_{\mathrm{df}} \kappa\theta.(\theta \cup (\theta \rightarrow \theta))$, then $V(e_0) =$ the set of first order types; if $e_1 =_{\mathrm{df}} \kappa\theta.(\theta \cup (\theta \rightarrow \theta) \cup \wedge(\theta, e_0))$ then $V(e_1) =$ the set of types generated by $\rightarrow$ and conjunction over all first order types. Let $e_{i+1} =_{\mathrm{df}} \kappa\theta.(\theta \cup (\theta \rightarrow \theta) \cup \wedge(\theta, e_i))$, $e_\omega =_{\mathrm{df}} \kappa\theta_1, \theta_0.(\theta_0 \cup (\theta_0 \rightarrow \theta_0) \cup \wedge(\theta_0, \theta_1))$; then $V(e_\omega) =$ the set of types generated by $\rightarrow$ and conjunction over all types in $V(e_i)$, $i = 1, 2 \ldots$. Generally, for ordinals $\alpha < \omega^\omega$ we define $e_{\beta+\omega^k} =_{\mathrm{df}} \kappa\theta_k \cdots \theta_0.(\theta_0 \cup (\theta_0 \rightarrow \theta_0) \cup \wedge(\theta_0, \wedge(\theta_1, \cdots \wedge (\theta_k, e_\beta)\cdots))$. Then $rank(e_\alpha) = \alpha$.

A formalism of finite descriptions for infinite derivations of height $< \omega^\omega$ can be constructed analogously. Also, one additional constructor would allow natural finite descriptions for useful types of order up to ϵ_0 (the first ordinal closed under exponentiation).

3.2 Correctness of descriptions

The statement that a function F is a correct description of a type (or derivation) is the conjunction of two conditions: (1) the *local correctness* of the infinite syntax tree, i.e. nodes are related correctly to their children nodes; and (2) the *well-foundedness* of F (every branch is finite). Local correctness is a Π^0_1 sentence, whereas well-foundedness is Π^1_1.

The well-foundedness of trees whose ordinal-height α is recursive can be reduced to the well-foundedness of a suitable canonical recursive well-ordering $\prec_\alpha$ of the natural numbers, of order type α (see e.g. [Schü77]). All trees of height $\prec \alpha$ can be codified as functions whose domain is a subdomain of the field of $\prec_\alpha$. The well foundedness of a tree is then assured by the well foundedness of $\prec_\alpha$.

We would also like to refer to *provably correct* derivations. Let $\mathbf{T}$ be a given number theory, in which all elementary functions are definable. Write $\mathbf{T}'$ for the theory in a two-sorted language extending the language of $\mathbf{T}$ with function variables, and where $\mathbf{T}'$ is $\mathbf{T}$ augmented with function-existence axioms for the functions definable in $\mathbf{T}$. It is well-known that $\mathbf{T}'$ is a conservative extension of $\mathbf{T}$. We write $\mathbf{J}^\alpha\lambda[\mathbf{T}]$ for the subformalism of $\mathbf{J}^\alpha\lambda$ where

- each type τ used is definable (as a function) in $\mathbf{T}$, and transfinite induction over τ is provable in $\mathbf{T}'$ for all formulas in the language of $\mathbf{T}'$;
- each derivation Δ used is definable (as a function) in $\mathbf{T}$, and the well-foundedness of Δ is provable in $\mathbf{T}'$.

That is, we can, within $\mathbf{T}$, compute along definable branches of derivations, and use transfinite induction over types.

3.3 Interpretation of predicative parametric polymorphism in discrete polymorphism

The stratified second order λ-calculus $\mathbf{P2}\lambda$ has a simple and natural embedding in $\mathbf{J}^\infty\lambda$. Define a mapping D from types of $\mathbf{P2}\lambda$ to types of $\mathbf{J}^\infty\lambda$, as follows. $D(\tau) =_{\mathrm{df}} \tau$ for quantifier free τ; $D(\tau \rightarrow \sigma) =_{\mathrm{df}} D(\tau) \rightarrow D(\sigma)$; and, for $t \in TV_k$, $D(\forall t.\tau) =_{\mathrm{df}} \bigwedge\{D(\tau)[\sigma/t] \mid order(\sigma) \prec \omega \cdot k\}$.

Theorem 1 *If $\vdash E : \tau$ is derived in $\mathbf{P}^k\mathbf{2}\lambda$, then $\vdash E : D(\tau)$ is derived in $\mathbf{J}^{\omega \cdot k}\lambda$.*

More generally, if $\vdash E : \tau$ is derived in $\mathbf{P}^\alpha\mathbf{2}\lambda$ for $\alpha \prec \epsilon$ (see [Lei89]) then $\vdash E : D(\tau)$ is derived in $\mathbf{J}^{\omega \cdot \alpha}\lambda$.

Theorem 1 shows that discrete polymorphism captures predicative parametric polymorphism. The advantage of infinite discrete polymorphism are, first, the ability to refer to type intersections far more general than the ones implicit in stratified parametric polymorphism, and second, a more uniform and transparent treatment of infinite constructs.

[3] From the Normalization Theorem below it follows that all β-r-derivations are acceptable. Size of expression is no issue in the absence of β-Exp, because no inference other than Exp permits the λ-expressions in premises to be larger than the one in the conclusion.

[4] We avoid using μ, because the ω-limit here is not a fixpoint.

3.4 Normal derivations and the subtype property

A node in a derivation is a *detour* (or *cut*) if it is derived by $\bigwedge$-Intro and is the premise of $\bigwedge$-Elim, or is derived by $\rightarrow$-Intro and is the first ("major") premise of $\rightarrow$-Elim. The type in a detour's type statement is the *eigen-type* of the detour. The *rank* of the detour is the rank of its eigen-type. The *detour-rank* of a type derivation Δ, $rank(\Delta)$, is the ordinal $\sup\{\rho \mid \rho$ is the rank of a detour in $\Delta\}$. A type derivation is *normal* if it has no detour, i.e. if its detour rank is 0.

A type discipline T has the *subtype property* when, if a λ-expression E has a type derivation, then it has a type derivation that uses only (literal) subtypes of the derived type. T has the *relative subtype property* when the above holds for E normal. [5]

The relative subtype property holds for the Second Order Typed λ-calculus if the subtypes of a type $\forall t.\tau$ are defined to include all types $\tau[\sigma/t]$. This reading is rarely of interest, since the "subtypes" of a type τ are here in no way shorter or simpler than τ.

In contrast, we have:

Theorem 2 *If Δ is a normal type derivation for $\{x_i : \sigma_i\}_i \vdash E : \tau$, where E is normal, then all types in Δ are literal subtypes of τ or of some σ_i.*

We shall prove below (Theorem 7) that every derived typing for a normal λ-expression has a normal derivation, and that every derived typing has a normal β-r-derivation. Hence,

Corollary 3 *Type derivations of $\mathbf{J}^\infty\lambda$ have the relative subtype property: Every derived typing θ for a normal λ-expression has a derivation in which all types are literal subtypes of the types in θ.*

Hence, β-derivations of $\mathbf{J}^\infty\lambda$ have the subtype property: Every derived typing θ has a β-derivation in which all types are literal subtypes of the types in θ.

4 Normalization

In this section we prove a strong-normalization theorem for $\mathbf{J}^\infty\lambda$. We show that every type derivation for a typing $\eta \vdash E : \tau$ can be transformed effectively into a type derivation for a typing $\eta \vdash E' : \tau$, where E' is β-normal and β-equal to E. We also obtain information on the computational nature of the transformation of a given derivation into a normal one.

4.1 Reductions

The basic derivation transformations are *reductions*, in the style of Prawitz [Pra65]. An *intersection reduction* maps a β-r-derivation

$$\begin{array}{c} \Delta_\tau \\ \{\eta \vdash E : \tau\}_{\tau \in T} \\ \eta \vdash E : \bigwedge T \\ \eta \vdash E : \tau_0 \\ \Pi \end{array}$$

to the β-r-derivation

$$\begin{array}{c} \Delta_{\tau_0} \\ \eta \vdash E : \tau_0 \\ \eta \vdash E : \tau_0 \\ \eta \vdash E : \tau_0 \\ \Pi \end{array}$$

The Rep inferences in the latter derivation prevent the output from being shorter than the input, and leave unchanged the position in the derivation tree of typing statements in Π and Δ_{τ_0}.

An *arrow reduction* maps a β-r-derivation of the form

$$\dfrac{\begin{array}{cc} \Delta & \\ \eta, x : \tau \vdash E : \sigma & \Sigma \\ \eta \vdash \lambda x.E : (\tau \rightarrow \sigma) & \eta \vdash F : \tau \end{array}}{\eta \vdash (\lambda x.E)F : \sigma}$$
$$\Pi$$

to the derivation

$$\begin{array}{c} \Delta[\Sigma/x] \\ \eta \vdash E[F/x] : \sigma \\ \eta \vdash E[F/x] : \sigma \\ \eta \vdash (\lambda x.E)F : \sigma \\ \Pi \end{array}$$

Here $\Delta[\Sigma/x]$ comes from Δ by

1. deleting x from the domain of environments (unless x is in the domain of η, and assuming, without loss of generality, that x has no unrelated occurrences in the given derivation);
2. substituting F for x, and
3. at each initial node of Δ of the form $\eta \cup \eta' \vdash x : \tau$, grafting $\Sigma[\eta'] =_{df}$ the result of extending each environment of Σ with η'.

4.2 Reductions operators

A standard normalization procedure for derivations eliminates detours from derivations in decreasing order of detour rank. This method cannot be used (at least not directly) where the complexity of detours form infinite ascending chains, as they might in infinite derivations. We adapt instead the well-known strong normalization technique of Tait and Prawitz.

A *pseudo-derivation* is defined like a typing derivation, except that well-foundedness is not required. Let $U = u_1, u_2 \ldots$ be a sequence, finite or infinite, without repetitions, of nodes in $\mathbf{N}^*$. U induces a transformation $\tilde{U}$ on pseudo-derivation Δ, namely the limit of Δ_i, where $\Delta_0 = \Delta$, and Δ_{i+1} arises from Δ_i by a reduction at u_i, if Δ_i has a detour at u_i, and $\Delta_{i+1} = \Delta_i$ otherwise. The definition of the reductions, using Rep inferences, implies that this limit is always defined: for every $u \in \mathbf{N}^*$, $(\tilde{U}\Delta)(u) = \Delta_i(u)$, where u_i is the last entry of U at or below u. Moreover, $\tilde{U}\Delta$ is a pseudo-derivation for every derivation Δ. We call an operation $\tilde{U}$ a *reduction operator*.

A reduction operator as above is *fair for* Δ if for every $u \in \mathbf{N}^*$, if Δ_i has a detour at u, then $u_j = u$ for some $j \geq i$.

If U is an enumeration of all of $\mathbf{N}^*$, that respects the initial-sequence relation on $\mathbf{N}^*$, then $\tilde{U}$ is a fair for all derivations. We say that such operators are *universal*.

[5] The subtype property is equivalent, under Curry-Howard formula-as-type isomorphism, to the *subformula property* for natural deduction logical calculi: each provable formula φ has a derivation that uses only subformulas of φ. No proof formalism exists for First Order Arithmetic, for Second order Logic, or for Dynamic Logic of programs, that has this property [Kre65], and perhaps the most important rationale for infinitary proof theory is to recover the subformula property.

Proposition 4 *Let Δ be a pseudo-derivation. If $\tilde{U}$ is a reduction operator fair for Δ, then $\tilde{U}\Delta$ is a detour-free pseudo-derivation.*

Moreover, if $\tilde{U}$ and $\tilde{U}'$ are reduction operators fair for Δ, then $\tilde{U}\Delta = \tilde{U}'\Delta$.

4.3 The Strong Normalization Theorem

A derivation Δ is *strongly normalizable (SN)* if $\tilde{U}\Delta$ is a derivation for every reduction operator $\tilde{U}$. For countable ordinals α, we define the notions of α-stability, and of α-improper reductions, by simultaneous recurrence on α. An *intersection α-improper reduction* maps a β-r-derivation of the form

$$\begin{array}{c}\Delta_\tau \\ \{\eta \vdash E : \tau\}_{\tau \in T} \\ \eta \vdash E : \bigwedge T\end{array}$$

to any one of the β-r-derivations

$$\begin{array}{c}\Delta_\tau \\ \eta \vdash E : \tau\end{array}$$

$\tau \in T$.

An *arrow α-improper reduction* maps a derivation of the form

$$\begin{array}{c}\Delta \\ \eta, x : \tau \vdash E : \sigma \\ \eta \vdash \lambda x.E : \tau \to \sigma\end{array}$$

where $rank(\tau \to \sigma) \leq \alpha$, to any derivation of the form

$$\begin{array}{c}\Delta[\Sigma/x] \\ \eta \vdash E[F/x] : \sigma\end{array}$$

provided Σ is a stable β-r-derivation, where $\beta = rank(\sigma) < \alpha$. Note that improper reductions operate on the last inference of derivations. Let Δ be a β-r-derivation whose derived type is of rank $\prec\alpha$. Δ is *α-stable* iff every finite sequence of reductions and of γ-improper reductions ($\gamma \leq \alpha$) yields a SN derivation. A derivation Δ is *stable* if it is α-stable, where $\alpha = rank(type((\Delta))$. We have the trivial

Lemma 5 *Every stable derivation is SN.*

A derivation Δ with a derived typing $x_1 : \sigma_1, \ldots, x_k : \sigma_k \vdash E : \tau$ is *stable under substitutions* if, for each stable derivations $\Pi_1 \ldots \Pi_k$, $\Delta[\Pi_1/x_1 \ldots \Pi_k/x_k]$ is stable.

Another trivial observation is:

Lemma 6 *Every derivation stable under substitutions is stable.*

Theorem 7 [Strong Normalization] *Every derivation Δ is stable under substitutions.*

Hence, by Lemmas 6 and 5, every derivation is SN.

The proof of Theorem 7 is analogous to [Pra71], and proceeds by transfinite induction on $rank(\Delta)$.

4.4 Increase of expression size under normalization

We are interested in the size increase of λ-expressions in a type derivation Δ when it is normalized. The determining factor is the detour rank of Δ, but we are able to give sharper bounds for derivations that are provably correct.

A *value expression* in a type derivation Δ is a λ-expression in Δ above which Δ has no β-expansions. Observe that in a normal β-r-derivation a value expression is always normal.

For an acceptable type derivation Δ, let $val(\Delta)$ be the first of its value expressions, under some canonical ordering of λ-expressions.

Theorem 8 *There is a functional N, of an ordinal argument, a function argument, and a numeric argument, defined from elementary functions by transfinite recursion on its first argument, and such that, if Δ is an acceptable β-r-derivation of $\mathbf{J}^\infty\lambda$, with $exp\text{-}size(\Delta) \leq n$ and $rank(\Delta) \preceq \theta$, and Δ^N is the normal form of Δ, then $exp\text{-}size(\Delta^N) = N(\theta, \Delta, n)$.*

Corollary 9 *For every θ there is a function N_θ, defined from elementary functions by transfinite recursion up to ω^θ, and such that, if Δ is an acceptable recursive β-r-derivation of $\mathbf{J}^\infty\lambda$, with $exp\text{-}size(\Delta) = n$ and $rank(\Delta) \preceq \theta$, Δ^N is the normal form of Δ, and d is a code for Δ, then $exp\text{-}size(\Delta^N) = N_\theta(d, n)$.*

The corollary follows from Theorem 8 by [Tai61].

The results above require no uniformity or provable correctness of the derivations. The following theorem does, but yields sharper bounds. We do not know whether the provability conditions are necessary to obtain these improved bounds. Let **EA** be Elementary Arithmetic (see [Ros86]), and let **EA'** be defined from **EA** as in §3.2 above.

Theorem 10 *Theorems 7 and 8 for $\mathbf{J}^\alpha\lambda$ are provable in **EA'**, augmented with transfinite induction over the types used.*

*Hence, if **T** is a number theory in which **EA** is interpretable, then a type derivation of $\mathbf{J}^\infty\lambda[\mathbf{T}]$, with set D of detour types, normalizes to a derivation in $\mathbf{J}^\infty\lambda[\mathbf{T}+Ind(D)]$, where $Ind(D)$ is (transfinite) induction over the types in D.*

5 Lambda representation of functions

A fundamental yardstick for calibrating the computational nature of a type discipline is the class of functions representable in the pure λ-calculus typed by the given discipline. We revisit this much-traveled territory to stress the distinction between strict, skewed, and non-uniform λ-representations of functions.

5.1 Uniform vs. non-uniform function representation

A canonical method for obtaining λ-representations for free algebras is due to Böhm and Berarducci [BB85].[6] Let A be a free algebra, generated from the constructors $\mathbf{c}_1 \ldots \mathbf{c}_m$, of arities $r_1 \ldots r_m \geq 0$, respectively. Let us curry the generators of A, so that $\mathbf{c}$ of arity r is re-read as an identifier c of type $\tilde{r}$, where $\tilde{0} =_{df} o$, $\widetilde{r+1} =_{df} o \to \tilde{r}$. When terms of A are given in this curried form, then a trivial λ-representation of an A-term e is e itself, with the identifiers $c_1 \ldots c_m$ considered as λ-variables. A closed form is obtained by abstracting

[6] It was also obtained indirectly in [Lei83].

with respect to the constructors, yielding the representation $\rho(e) =_{df} \lambda c_1 \ldots c_m.e$. In 1λ all these expressions can be assigned type $\alpha_A =_{df} \tilde{r}_1 \to \cdots \to \tilde{r}_m \to o$, the *canonical type for A*, as well as $\alpha[\tau] \equiv \alpha_A[\tau/o]$ for any type τ. [7] In particular, if A is the algebra N of the numerals, whose constructors are 0 and s, then $\rho(n)$ is Church's n'th numeral, and $\alpha_{\mathbf{N}} = (o \to o) \to o \to o$.

Let $t\lambda C$ be a typed λ-calculus, A a free algebra, f a k-ary function over A. We say that a λ-expression E *represents* f *non-uniformly* if, for all $e_1 \ldots, e_k \in A$, $E\rho(e_1)\cdots\rho(e_k)$ can be typed in $t\lambda C$, and $E\rho(e_1)\cdots\rho(e_k) =_{\beta\eta} \rho(f(e_1, \ldots, e_k))$. A moderate uniformity condition is that a type be fixed for each one of the k inputs, and for the output: E *represents* f *skewly* in $t\lambda C$ if there are types $\tau_1 \ldots \tau_k$, such that, for all $e_1 \ldots, e_k \in A$, $E\rho(e_1)\cdots\rho(e_k)$ can be typed in $t\lambda C$, with type τ_i assigned to $\rho(e_i)$ $(i = 1 \ldots k)$, and $E\rho(e_1)\cdots\rho(e_k) =_{\beta\eta} \rho(f(e_1, \ldots, e_k))$. We call $\tau_1 \ldots \tau_k$ the *input types*, and the type of $\rho(f(e_1, \ldots, e_k)$ (as determined by the type of E) the *output type*. If the input types and the output type are all the same, we say that E *represents* f *strictly (at the common type,* τ_1*)*.

5.2 Simply typed λ-representations

We show that for $t\lambda C = 1\lambda$ the various representations of functions reduce (at least over most algebras) to two: skewed representation, and strict representation with input and output of canonical type.

Proposition 11 *Suppose a free algebra A has at most one 0-ary constructor, or has at least one k-ary constructor with* $k > 1$. *If a function over A has a non-uniform* ρ*-representation in* 1λ, *then it has a skewed representation.*

Proof sketch. Suppose A is an algebra that satisfies the proposition's condition. If A is degenerated (no 0-ary constructor or only 0-ary constructors) then A is finite, and all functions over A are represented strictly.

If A is non-degenerated we exhibit an $e \in A$ such that the only possible types for $\rho(e)$ are of the form $\alpha_A[\tau]$. Then, if f is represented non-uniformly by E, and $E\rho(e)\cdots$ can be typed, then the type assigned to $\rho(e)$ can be assigned to $\rho(e')$ for any $e' \in A$, and so the representation has a skewed typing.

Suppose A has one 0-ary constructor z, and the remaining constructors are $c_1 \ldots c_m$. Let $c_i(e)$ abbreviate $c_i(e, e, \ldots, e)$ (where number of arguments = $arity(c_i)$). Let $e =_{df} c_1(c_1(c_2(c_2(\cdots c_m(c_m(z))\cdots))))$. If σ is the type assigned to $\rho(e)$, and τ is the type assigned to z, then σ must be $\alpha_A[\tau]$.

Suppose A has a k-ary constructor b $(k > 1)$, and let $z_1 \ldots z_q$ be the 0-ary constructors of A. Form a term w that has subterms of the form $b(z_i, \ldots)$ for all $i = 1 \ldots q$ (this is possible since there is at least one non-unary constructor). Set $e =_{df} c_1(c_1(c_2(c_2(\cdots c_m(c_m(w))\cdots))))$. If a type is assigned to $\rho(e)$, with τ the type assigned to some z_i, then b must be assigned a type of the form $\tau \to \cdots$, so τ must be assigned to all z_i's $(i = 1 \ldots q)$, and the type assigned to $\rho(e)$ must be $\alpha_A[\tau]$. ⊣

Our definition of strict representation does not insist on using the canonical typing α_A. The following proposition shows that this is inconsequential.

[7] Note that $rank(\alpha_A) = 2$ (except for degenerated cases). Conversely, every order 2 type α corresponds to a free algebra A_τ, and every closed term of type α is $\beta\eta$ equal to a term of the form above.

Proposition 12 *Let f be a function over a free algebra A. If f is strictly representable in* 1λ *then it is strictly representable with* α_A *as the type of the inputs and output.*

Proof sketch. Let E be a normal λ-expression representing f strictly in 1λ, i.e. $E\rho(e_1)\cdots\rho(e_k) =_{\beta\eta} \rho(f(e_1 \ldots e_k))$, with E typed by $\alpha_A[\tau]^k \to \alpha_A[\tau] = \alpha_A[\tau] \to \cdots \to \alpha_A[\tau] \to \alpha_A[\tau]$, for all $e_1 \ldots e_k \in A$. Since E is normal and closed, the type of each of its subexpressions is a subtype of $\alpha_A[\tau]^k \to \alpha_A[\tau]$. Call a typed expression F τ*-centered* if the type assigned to any subexpression is either a subtype of τ or a supertype of τ (i.e. of the form $\sigma[\tau]$). Thus E is τ-centered.

If F is a τ-centered expression, call a subexpression G of F a *center* if it is assigned τ, is not a subexpression of any expression assigned a subtype of τ, and is an abstraction term (so the type of its immediate subexpression is a strict subtype of τ). Clearly, no center can be a subexpression of another. Let $\check{F}$ arise from F by replacing each center by a fresh variable of type τ. It is easy to see that if F $\beta\eta$-reduces to F', then $\check{F}$ $\beta\eta$-reduces to $\check{F}'$. It follows that $\check{E}$ also represents f.

The type of each subexpression of $\check{E}$ is $\sigma[\tau]$ for some σ. Replacing each $\sigma[\tau]$ by σ, we obtain a typed expression representing f with α_A as the type of all inputs and output. ⊣

The functions strictly representable in 1λ (modulo ρ) are characterized in [Zai90, Zaiα, Leiα]. The skewly represented functions of 1λ include the predecessor and exponentiation functions, but not subtraction [FLO83]. For example, exponentiation is representable by
$\lambda nms.nms$, where n is assigned type
$\alpha_{\mathbf{N}}(o \to o) = ((o \to o) \to (o \to o)) \to (o \to o) \to (o \to o)$,
m is assigned $\alpha_{\mathbf{N}}$, and s is assigned $o \to o$.

5.3 Intersection typed λ-representations

Proposition 13 *Every total recursive function is representable non-uniformly in (the p-proper typed fragment of)* $J\Omega\lambda$.

Proof. If E λ-represents a total (unary) f, then $E\tilde{n}$ is a normalizable expression for every $n \geq 0$, and has therefore a p-proper type derivation in $J\Omega\lambda$, by [CDV81]. ⊣

Using a different λ-representation, $n \mapsto \tilde{n}$, of natural numbers we can replace $J\Omega\lambda$ by $J\lambda$: let E represent f in the λI-calculus, modulo a suitable representation $n \mapsto \tilde{n}$ of the numerals (see [Bar81]); $E\tilde{n}$ is normalizable, and, being in the λI-Calculus, is therefore strongly normalizable, whence typable in $J\lambda$.

In contrast, we have:

Theorem 14 *Every numeric function skewly* ρ*-representable in* $J\lambda$ *is elementary.*

Proof sketch. This is a special case of Theorem 22 below. For a more direct proof, suppose E is a normal expression representing f uniformly. Let Δ be a type derivation in $J\lambda$ of $E : \tau \to \sigma$. For each n, let Θ_n be a normal type derivation of $\tilde{n} : \tau$. So Δ and Θ_n combine to a derivation of $E\tilde{n}$ of rank $d \leq length(\tau)$.

As in [Lei89], one proves that all λ-expressions of length $\leq l$, and with a $J\lambda$ type derivation of detour rank $\leq d$, normalize to expressions of length $K_d(l)$, where K_d is an elementary function. Thus, each $E\tilde{n}$ normalizes to an expression of length $\leq K_d(n)$. Therefore, E represents a primitive

recursive function bounded by the elementary function K_d. Hence f is elementary (see [Ros84]). ⊣

The proof in [Plo82] that subtraction and equality are not skewly ρ-representable in 1λ seems to apply also to $J\lambda$. We conjecture that the functions skewly [strictly] representable in $J\lambda$ are skewly [strictly] representable already in 1λ.

6 Functions representable using infinite type intersection

6.1 Computability of functions represented using infinite type intersection

The definition of skew and strict function representation in $J^\infty\lambda$ are the same as for the other type disciplines.

For a class of functions $\mathcal{C}$, and a free algebra A, we say that a type τ is *representable for A in $\mathcal{C}$* if there is a function in $\mathcal{C}$ that yields, for each $e \in A$, a canonical code of a type inference for $\vdash \rho(e) : \tau$. If f is a function over A, we say that E represents f skewly in $\mathbf{J}^\infty\lambda(\mathcal{C})$ if it is representable skewly in the sense above, with the added proviso that the types for inputs and output are representable in $\mathcal{C}$ for A.

Let $\mathbf{T}$ be a theory in which $\mathbf{EA}$ is interpretable. We say that a typing τ is *representable in $\mathbf{T}$ for A* if there is a proof of $\mathbf{T}$ that $\rho(e) : \tau$ is derivable for all $e \in A$. If f is a function over A, we say that E represents f skewly in $\mathbf{J}^\infty\lambda[\mathbf{T}]$ if there is a proof in $\mathbf{T}$ that E represents f skewly in $\mathbf{J}^\infty\lambda$.

The limits on function representability in $\mathbf{J}^\infty\lambda$ can be assessed using Theorems 9 and 10.

Theorem 15 *Let $\mathbf{T}$ be a number theory in which $\mathbf{EA}$ is interpretable. If f is skewly representable in $\mathbf{J}^\infty\lambda[\mathbf{T}]$, then f is provably computable in $\mathbf{T}$.*

Proof. Suppose E represents skewly a k-ary function f in $\mathbf{J}^\infty\lambda[\mathbf{T}]$. Then, by Theorem 10, there is a proof that $E\rho(e_1)\cdots\rho(e_k)$ has a normal type derivation, for each $e_1 \ldots e_k \in A$. By Theorem 10, there is a function F provably computable in $\mathbf{T}$, and such that for each $\vec{e} \in A$ $F\vec{e}$ is the (unique) value expression in a normal β-r-derivation for $E\rho(e_1)\cdots\rho(e_k)$. That value expression is a β-normal λ-expression, β-equal to $E\rho(e_1)\cdots\rho(e_k)$, so f is elementary in F. ⊣

If $\mathcal{C}$ is a class of functions, and θ a recursive ordinal (given as a recursive well-ordering), let $\mathcal{C}^\theta$ denote $\mathcal{C}\cup\mathcal{E}^3$ augmented with transfinite recursion up to θ.

Theorem 16 *If $\mathcal{C}$ is a class of functions, θ a recursive ordinal, then every function representable in $\mathbf{J}^\theta\lambda(\mathcal{C})$ is in $\mathcal{C}^\theta$.*

The proof is simlilar to the proof of Theorem 15.

6.2 Representing functions using infinite type intersection

A numeric function f is defined by *iteration from g* if

$$\begin{aligned} f(0,\vec{x}) &= x_1 \\ f(n+1,\vec{x}) &= g(f(n,\hat{x}),\hat{x}) \end{aligned}$$

Lemma 17 *Suppose that a unary function g is represented askew by a λ-expression E_g, with input of type $\sigma[\rho]$ and output type ρ. Then E_g represents g strictly, with input and output of type $\bigwedge_{i<\omega}\tau_i$, where $\tau_0 =_{\mathrm{df}} \rho$, $\tau_{2n+1} =_{\mathrm{df}} \sigma[\tau_{2n}]$, $\tau_{2n+2} =_{\mathrm{df}} \rho[\tau_{2n+1}]$.*

More generally, if g is of arity $r-1$, and g is represented askew by a λ-expression E_g, with inputs of types $\sigma_1[\rho],\ldots,\sigma_{r-1}[\rho]$ and output of type ρ, then E_g represents g strictly, with input and output of type $\bigwedge_{i<\omega}\tau_i$, where $\tau_0 =_{\mathrm{df}} \rho$, $\tau_{rn+j} =_{\mathrm{df}} \sigma_j[\tau_{rn}]$ $(1 \le j < r)$, and $\tau_{rn+r} =_{\mathrm{df}} \rho[\bigwedge_{1\le j<r}\tau_{rn+j}]$.

Lemma 18 *If g is a numeric function skewly represented in $\mathbf{J}^\alpha\lambda(\mathcal{C})$, with input types $\tau,\sigma_1\ldots\sigma_r$ and output type τ, and f is defined by iteration (with respect to the first variable) from g, then f is skewly represented in $\mathbf{J}^{\alpha+2}\lambda(\mathcal{C})$, with input types $\nu(\tau),\sigma_1\ldots\sigma_r$ and output type τ.*

The lemma holds also for functions over any free algebra A, for the natural generalization of the iteration schema to free algebras (see [Lciα]).

Combining Lemmas 17 and 18, we have

Lemma 19 *If a numeric function g is represented strictly in $\mathbf{J}^\alpha\lambda(\mathcal{C})$, where $\mathcal{C}$ is closed under composition with elementary functions, and f is defined from g by iteration, then f is represented strictly in $\mathbf{J}^{\alpha\cdot\omega}\lambda(\mathcal{C})$.*

Similarly for $\mathbf{J}^\alpha[\mathbf{T}]$, for any theory $\mathbf{T}$ in which $\mathbf{EA}$ is interpretable.

Lemma 20 *For each α, and each class $\mathcal{C}$ of functions closed under composition with elementary functions, the class of functions representable strictly in $\mathbf{J}^\alpha\lambda(\mathcal{C})$ is closed under composition.*

Lemma 21 *Suppose $\mathcal{C}$ contains all functions defined from $0, S, +$ and $\times$ by two iterations, and is closed under composition with such functions. Then $\mathcal{C}$ is closed under bounded recursion.*

Proof. See [Ros84] §1.3, proof of Theorem 3.1. ⊣

Theorem 22 *For $k \ge 2$, every function in $\mathcal{E}^{k+2}$ is representable in $\mathbf{J}^{\omega^k}\lambda(\mathcal{E}^3)$ and in $\mathbf{J}^{\omega^k}\lambda[\mathbf{EA}]$.*

Hence, every primitive recursive function is representable in $\mathbf{J}^{\omega^\omega}\lambda(\mathcal{E}^3)$ and in $\mathbf{J}^{\omega^\omega}\lambda[\mathbf{EA}]$.

Proof sketch. By induction on k. Exponentiation is in $\mathbf{J}^\omega\lambda(\mathcal{E}^3)$. Hence the super-exponential function F_4 is representable in $\mathbf{J}^{\omega^2}\lambda(\mathcal{E}^3)$, by Lemma 19, and therefore so is every $f \in \mathcal{E}^4$, by Lemmas 20 and 21. The induction step follows from Lemma 19.

The proof for $\mathbf{J}^{\omega^k}\lambda[\mathbf{EA}]$ is similar. ⊣

From Theorem 15, 16 and 22 we conclude:

Theorem 23 *Let $\mathcal{C}$ be a class of primitive recursive functions containing all elementary functions, and closed under composition with elementary functions. Then the functions represetable in $\mathbf{J}^{\omega^\omega}\lambda(\mathcal{C})$ are precisely the primitive recursive functions.*

Let $\mathbf{PRA}$ be primitive recursive arithmetic (formulated, say, as Peano's Arithmetic with induction restrcited to Σ^0_1 formulas), and let $\mathbf{T}$ be a theory interpretable in $\mathbf{EA}$, and in which $\mathbf{PRA}$ is interpretable. Then the functions represetable in $\mathbf{J}^{\omega^\omega}\lambda[\mathbf{T}]$ are precisely the primitive recursive functions.

6.3 Representability of functionals

We refer to a weak notion of functional representability in λ calculi. The *(hereditarily) total functionals of (simple) finite types* are defined by: the total functionals of type o are the natural numbers; a total functional of type $\sigma \to \rho$ is a function that maps total functionals of type σ to total functionals of type ρ.

The scheme of primitive recursion can be used to generate functionals of arbitrary functionality type. For instance, an iteration functional of type $(0 \to 0) \to 0 \to (0 \to 0)$ is defined by $\Phi(f,0) =_{\mathrm{df}} \lambda x.x$, $\Phi(f,sy) =_{\mathrm{df}} \lambda x.f(\Phi(f,y)(x))$. The Ackermann-like function A defined by $A(0,x) =_{\mathrm{df}} 2x$, $A(sy,0) =_{\mathrm{df}} 1$, $A(sy,sz) =_{\mathrm{df}} A(y,A(sy,z))$, is then defined by the primitive recursion: $A(x,y) =_{\mathrm{df}} \tilde{A}(x)(y)$, where $\tilde{A}(0) =_{\mathrm{df}} \lambda x.2x$, $\tilde{A}(sy) =_{\mathrm{df}} \lambda x.\Phi(\tilde{A}(y),x)(1)$.

A λ-expression E *represents (internally)* a total functional Φ of type τ if either $\tau = 0$ and E is β-equal to a Church-numeral, or $\tau = \sigma \to \rho$, and for each expression F, if F represents a total functional of type σ, EF represents a total functional of type ρ.

For example, the functional $\tilde{A}$ is represented by $\lambda yx.yFDx$, where D represents $\lambda x.2x$, and $F =_{\mathrm{df}} \lambda f.\lambda u.(uf\bar{1})$ represents a primitive recursive functional of type $(0\to 0)\to(0\to 0)$. The types assigned to the latter functional must be such that u is legally applied to f, and that the type of the mapping $v \mapsto vf\bar{1}$ is a type for f. This cannot be obtained by the iterative construction described above for transforming a skew representation of functions into a strict one. Rather, that construction has to be iterated ω times. The types obtained have height ω, ω^2, ω^3, $\cdots$, so that the conjunction of all of them, which can now be assigned to both f and u, has height ω^ω.

Recall that ϵ_o denotes the first ordinal closed under exponentiation, that is $\epsilon_o =_{\mathrm{df}} \lim_i(\omega_i)$, where $\omega_1 =_{\mathrm{df}} \omega$, $\omega_{i+1} =_{\mathrm{df}} \omega^{\omega_i}$. The outline above can be generalized to yield the following:

Theorem 24 *The numeric functionals (of any functionality type) defined by primitive recursion for functionals up to order k are representable in $\mathbf{J}^{\omega_{k+1}}\lambda(\mathcal{E}^3)$, and in $\mathbf{J}^{\omega_{k+1}}\lambda[\mathbf{EA}]$.*

Hence, all numeric functionals defined by primitive recursion are representable in $\mathbf{J}^{\epsilon_0}\lambda(\mathcal{E}^3)$ and in $\mathbf{J}^{\epsilon_0}\lambda[\mathbf{EA}]$.

6.4 Representability of the provably recursive functions

An important computational complexity class above the primitive recursive functions is the class of computable functions that are provably total in First Order (Peano) Arithmetic **PA**. This class is related, in a number of ways, to the ordinal ϵ_o, and to natural recursion schemas. Some conditions equivalent to a recursive function f being provably total in **PA** are:

1. (Kreisel) f is provably total in the constructive variant of **PA**.
2. (Gödel) f is definable by iteration and composition for all finite types.
3. (Schütte) f is definable by transfinite recursion over ordinals $\prec \epsilon_o$.
4. (Gentzen) f is provably total in induction free **PA** augmented by transfinite induction up to ϵ_o for quantifier free formulas.
5. (Löb & Wainer) f is in the extended Grzegorczyk hierarchy, $\{\mathcal{E}^\alpha\}_{\alpha\prec\epsilon_o}$, where $\mathcal{E}^\alpha$ consists of the functions defined by composition with elementary functions from the function F_α. Here $F_0(x) =_{\mathrm{df}} x+1$, $F_{\beta+1}(0) =_{\mathrm{df}} 2$, $F_{\beta+1}(x+1) =_{\mathrm{df}} F_\beta^{[x+1]}(0) \equiv F_\beta(F_{\beta+1}(x))$, and for limit ξ, $F_\xi(x) =_{\mathrm{df}} F_{\xi(x)}(x)$, where $\xi(x)$ is the x'element of Cantor's fundamental sequence for ξ ([LW70] modified by Rose).
6. ([Lei89]) f is representable in $\mathbf{P}^{\epsilon_0}2\lambda$.
7. ([Leiβ]) f preserves the natural numbers, provably in second order logic with comprehension restricted to computational (= strict Π^1_1) formulas.

From Theorems 15 and 24 we obtain yet another characterization:

Theorem 25 *A numeric function f is provably recursive in **PA** iff it is representable in $\mathbf{J}^{\epsilon_o}\lambda(\mathcal{E}^3)$, and also iff it is representable in $\mathbf{J}^{\epsilon_o}\lambda[\mathbf{EA}]$.*

An alternative proof of the theorem, of independent interest, uses the extended Grzegorczyk hierarchy. The uniformity of the definition of the functions F_α permits the representation by a uniform expression E_α in the (untyped) λ-calculus, as follows.

Define, for $k \geq 0$, $D_k =_{\mathrm{df}} \lambda a_k a_{k-1} \ldots a_0 x.\, x a_k a_{k-1} \cdots a_0 x$. Then, for example, if F_β is represented by E_β, then $F_{\beta+1}$ is represented by D_0E_β, and $F_{\beta+\omega}$ is represented by $D_1D_0E_\beta$. More generally, we have

Lemma 26 *F_{ω_k} is represented by $D_kD_{k-1}\cdots D_0E$, where E represents F_0.*

The alternative proof of Theorem 25 falls now out of the following

Theorem 27 *Every D_k is representable in $\mathbf{J}^{\omega_k+1}\lambda$.*

7 Some open problems and directions for research

1. It is likely that interesting new filter models for the λ-calculus are related to $\mathbf{J}^\infty\lambda$, generalizing the filter models of [BCD83] obtained from $\mathbf{J}\lambda$.
2. What are the principal type properties of $\mathbf{J}^\infty\lambda$ and its finitarily represented subdisciplines, analogous to the semi-principal typing for $\mathbf{J}\lambda$ [CDV80, RV84]? Are there natural relations to principal "constraints" for typings in $\mathbf{2}\lambda$, as discussed in [GR88]?
3. Neither $\mathbf{J}^\infty\lambda$ nor the stratified quantificational discipline $\mathbf{P}^\infty\mathbf{2}\lambda$ are inherently restricted to ordinals that are recursive, or even countable. The use of sufficiently large ordinals, ones that cannot be predicatively justified, should allow an interpretation in $\mathbf{J}^\infty\lambda$ of impredicative fragments of full $\mathbf{2}\lambda$, in particular of 2λ restricted to Π^1_1 type arguments. This would be analogous to proof theoretic ordinal calibrations of impredicative systems for analysis, as in [BS88].
4. Is $\mathbf{J}^\infty\lambda$ (or its subdisciplines) reducible to $\mathbf{J}^\alpha\lambda$ for some sufficiently large α? (This is the opposite of (3)!). Would uncountable type intersection increase the typing power of $\mathbf{J}^\infty\lambda$?

References

Bar81 Henk Barendregt, **The Lambda Calculus,** North Holland, Amsterdam, 1981, xiv+615pp.

BB85 Corrado Böhm and Allessandro Berarducci, *Automatic synthesis of typed λ-programs on term algebras,* **Theoretical Computer Science 39** (1985) 135–154.

BCD83 H. Barendregt, M. Coppo & M. Dezani-Ciancaglini, *A filter lambda-model and the completeness of type assignment,* **Journal of Symbolic Logic 48** (1983) 931-940.

BS88 W. Buchholz & K. Schütte, **proof Theory of Impredicative Subsystems of Analysis,** Bibliopolis, Napoli, 1988.

Cop80 Mario Coppo, *An extended polymorphic type system for applicative languages,* in P. Dembinski (ed.), **Mathematical Foundations of Computer Science,** Springer-Verlag (LNCS #88), berlin, 1980, 194–204.

CD80 Mario Coppo and Mariangiola Dezani-Ciancaglini, *An extension of basic functionality theory for λ-calculus,* **Notre-Dame Journal of Formal Logic 21** (1980) 685-693.

CDV80 Mario Coppo, Mariangiola Dezani-Ciancaglini and B. Veneri, *Principla type schemes and λ-calculus semantics,* in J.P. Seldin & J.R. Hindley, **To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism,** Academic Press, London, 1980, pp. 535–560.

CDV81 Mario Coppo, Mariangiola Dezani-Ciancaglini and B. Veneri, *Functional character of solvable terms,* **Zeitschr. f. math. Logik und Grundlagen d. Math 27** (1981) 45-58.

FLO83 Steven Fortune, Daniel Leivant, and Michael O'Donnell, *The expressiveness of simple and second order type structures,* **Journal of the ACM 30** (1983), pp 151-185.

GR88 P. Giannini & S. Ronchi Della Rocca, *Characterization of typings in polymorphic type discipline,* Proceedings, Third Annual Symposium on Logic in Computer Science, IEEE Computer Society, Los Angeles, 1988, 61–70.

Gir72 Jean-Yves Girard, *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre superieur,* Thèse de Doctorat d'Etat, 1972, Paris.

Kol25 Alexander Kolmogorov, *Sur le principe de tertium non datur,* **Recueil Math. de la Soc. Math. de Moscou (= Matématicéskij Sbornik) 32** (1925) 647–667. English translation: *On the principle of excluded middle,* in Jean van Heijenoort (editor), **From Frege to Godel,** Harvard university Press, Cambridge, 1967, pp. 414–437.

Kre65 Georg Kreisel, *Mathematical logic,* in T. Saaty (ed.), **Lectures on Modern Mathematics vol. III,** John Wiley, New York, 1965, 95–195.

Lei83 Daniel Leivant, *Reasoning about functional programs and complexity classes associated with type disciplines,* **Twenty-fourth Annual Symposium on Foundations of Computer Science** (1983) 460-469.

Lei85 Daniel Leivant, *Syntactic translations and provably recursive functions,* **Journal of Symbolic Logic 50** (1985) 682–688.

Lei86 Daniel Leivant, *Typing and computational properties of lambda expressions,* **Theoretical Computer Science 44** (1986) 51–68.

Lei89 Daniel Leivant, *Stratified polymorphism,* Proceedings, Fourth Annual Symposium on Logic in Computer Science (Pacific Grove, CA), IEEE Computer Society, Los Angeles, 1989, 39–47.

Leiα Daniel Leivant, *Subrecursion and lambda representation over free algebras,* in S. Buss and P. Scott (eds.), **Feasible Mathematics** (Proceedings of the June 1989 Workshop at Cornell), to appear.

Leiβ Daniel Leivant, *Predicativity levels of number theories,* to appear.

LW70 M.H. Löb & S.S. Wainer, *Hierarchies of number theoretic functions I, II,* **Arch. Math. Logik 13** (1970) 39–51, 97–113.

Min78 Gregori Mints, *Finite investigation of transfinite derivations,* **Journal of Soviet Mathematics 10** (1978) 548–596.

RP88 John Reynolds and Gordon Plotkin, **On functors expressible in the polymorphic typed lambda calculus,** Report *cmu-cs-88-125,* Carnegie-Mellon University, 1988 (also to appear elsewhere).

Plo82 G. Plotkin, *A note on the functions definable in the typed λ-calculus,* Manuscript, January 1982.

Pra65 Dag Prawitz, **Natural Deduction,** Almqvist and Wiskel, Uppsala, 1965.

Pra71 Dag Prawitz, *Ideas and results in proof theory,* in J.E. Fenstad (ed.), **Proceedings of the Second Scandinavian Logic Symposium,** North-Holland, Amsterdam (1971) pp. 235-308.

Rey74 John Reynolds, *Towards a theory of type structures,* in J. Loeckx (ed.), **Conference on Porgramming,** Springer-Verlag (LNCS #19), Berlin, 1974, pp. 408–425.

Rey84 John Reynolds, *Polymorphism is not set-theoretic,* in G. Kahn, D.B. MacQueen, and G.D. Plotkin (eds.), **Semantics of Data Types,** Springer-Verlag (LNCS #173), Berlin, 1984, pp. 145-156.

Rey87 John Reynolds, *Conjunctive types in functional and ALGOL-like languages,* Notes for an invited talk, Second Symposium on Logic in Computer Science, Ithaca, NY. Abstract in Proceedings, Second Annual Symposium on Logic in Computer Science, IEEE Computer Society, Los Angeles, 1987, 119.

Rey88 John Reynolds, *Preliminary Design of the Programming Language Forsythe,* Tech Report CMU-CS-88-159, Carnegie-Mellon University, June 1988.

RV84 S. Ronci della Rocca and B. Venneri, *Principal type schemes for an extended type theory,* **Theoretical Computer Science 28** (1984) 151–169.

Ros84 H.E. Rose, **Subrecursion**, Clarendon Press (Oxford University Press), Oxford, 1984.

Schü77 Kurt Schütte, **Proof Theory**, Springer-Verlag (GMW #225), Berlin, 1977.

Sta81 Richard Statman, *Number theoretic functions computable by polymorphic programs,* **Twenty Second Annual Symposium on Foundations of Computer Science**, IEEE Computer Society, Los Angeles, 1981, 279–282.

Str67 C. Strachey, *Fundamental concpets in programming languages,* Lecture notes for the International Summer School in Computer Programming, Copenhagen, August 1967.

Tai61 W.W. Tait, *Nested recursion,* **Math. Ann. 143** (1961) 236–250.

Zai90 Marek Zaionc, *A characterization of λ-definable tree operations,* to appear in **Information and Computation**.

Zaiα Marek Zaionc, *λ-definability on free algebras,* manuscript submitted for publication.