INFORMATION AND COMPUTATION 93, 93–113 (1991)

Finitely Stratified Polymorphism

DANIEL LEIVANT

School of Computer Science,
Carnegie Mellon University, Pittsburgh, Pennsylvania 15213

We consider predicative type-abstraction disciplines based on type quantification with finitely stratified levels. These lie in the vast middle ground between quantifier-free parametric abstraction and full impredicative abstraction. Stratified polymorphism has an unproblematic set-theoretic semantics, and may lend itself to new approaches to type inference, without sacrificing useful expressive power. Our main technical result is that the functions representable in the finitely stratified polymorphic λ-calculus are precisely the super-elementary functions, i.e., the class $\mathscr{E}_4$ in Grzegorczyk's subrecursive hierarchy. This implies that there is no super-elementary bound on the length of optimal normalization sequences, and that the equality problem for finitely stratified polymorphic λ-expressions is not super-elementary. We also observe that finitely stratified polymorphism augmented with type recursion admits functional algorithms that are not typable in the full second order λ-calculus.

INTRODUCTION

Type disciplines for programming languages attempt to strike a balance between three often conflicting aims: expressive power, simplicity and methodological coherence, and user friendly implementability. The trade-off between these aims can be seen in the contrast between two main paradigms of polymorphic typing: parametric quantifier-free polymorphism, as in ML, vs. Girard and Reynolds's impredicative quantificational discipline F_2 (Girard, 1972; Reynolds, 1974). The former is user friendly by virtue of its (in practice) fast type inference mechanism, but it lacks the power of full type quantification, and it suffers from certain anomalies (Mycroft, 1984; Peyton-Jones, 1987). The latter has great expressive power, well beyond current programming needs, but it is probably too powerful to allow computationally feasible user friendly facilities, such as type inference.

We discuss here another potential ingredient in the design of type disciplines for programming languages, namely *stratification of type abstraction*, which engenders a whole spectrum of disciplines between quantifier-free parametric polymorphism and full quantificational polymorphism. It

643/93/1-7

0890-5401/91 \$3.00

therefore has the potential both of clarifying theoretical issues concerning polymorphic typing, and of serving as an ingredient in language design.

The idea of stratifying abstraction into levels goes back to the Ramified Type Theory of (Russell, 1908; Whitehead and Russell, 1910), whose purpose was to circumvent the antinomies of Naive Set Theory. It was revived in the 1950's (e.g., (Kreisel, 1960; Wang, 1954, 1962)) in relation to *Predicative Analysis*. Stratification of type abstraction in the polymorphic λ-calculus (and related typed programming languages) seems to originate with (Statman, 1981).

The purpose of stratification is to avoid impredicative abstraction: a second order type $\tau = \forall t.\sigma$ has t ranging over all types, including τ itself. To circumvent this circularity, one stipulates that types fall into levels, with the base level consisting exactly of those types whose definition involves no type quantification. The next level consists of types whose definition may use quantification over types of the base level, and so on. This eliminates circularity, since in a type $\tau = \forall t^n.\sigma$ the type variable t^n ranges over types of level n, excluding τ since $level(\tau) > level(t^n) = n$. The construction of levels can proceed into transfinite ordinals, by taking at limit ordinals ξ the union over lower levels: in $\forall t^\xi.\sigma$ the variable t^ξ ranges over types of levels $< \xi$. This extension, albeit transfinite, has natural fragments with potentially useful finite presentations (Leivant, 1989). In this paper we focus on finite stratification, deferring to a future paper the treatment of transfinite stratification and other transfinite type constructions (Leivant, 1990b).

Our main technical result (Theorem 22) is that the numeric functions representable in the finitely stratified polymorphic λ-calculus are precisely the super-elementary functions. In Section 2 we show that every super-elementary function is representable, and in Section 3 we show the converse. An outline of the proof appeared in (Leivant, 1989).

In Section 4 we derive limitative results on finitely stratified polymorphism from the characterization above: there is no super-elementary bound on the length of optimal reduction sequences (Theorem 24), and the equality problem for the finitely stratified λ-calculus is not super-elementary (Theorem 25).

In the final Section 5 we consider stratified polymorphism with recursive types. It is known that, in spite of the computational strength of $\mathbf{F}_2$, certain simple numeric functional algorithms, such as Maurey's algorithm for branching on inequality, cannot be typed in it (Krivine, 1987). We point out that Maurey's example can be typed in the finitely stratified calculus augmented by recursive types.

1. The Finitely Stratified Polymorphic λ-Calculus

1.1. *Stratification*

The *finitely stratified polymorphic lambda calculus*, $\mathbf{SF}_2$, is similar to Girard and Reynolds's second order lambda calculus $\mathbf{F}_2$ (Girard, 1972; Reynolds, 1974) except that types are classified into levels 0, 1, ... Type expressions τ and their levels $L(\tau)$ are defined inductively:

- For each level $k = 0, 1, ...$ there is a denumerable supply of *type variables of level* k: $t^k, t_1^k, t_i^k, ...$ (We omit the level superscript when it is irrelevant or clear from the context.) A type variable of level k is also a *type expression* of level k.

- If σ and τ are type expressions, of levels p and q respectively, then $\sigma \to \tau$ is a type expression of level $\max(p, q)$.

- If τ is a type expression of level p, then $\forall t^q.\tau$ is a type expression of level $\max(p, q+1)$.

Thus, the level of a type expression τ is the largest of $L(t)$ for t free in τ and $L(t)+1$ for t bound in τ.

Expressions E and their types $type(E)$ are defined inductively:

- For each type expression τ there is a denumerable supply of *object variables of type* τ: $x^\tau, x_0^\tau, ..., x_i^\tau, ...$ τ is *the type of* x^τ. (We omit type superscripts when irrelevant or clear from the context.) An object variable of type τ is also an expression of type τ.

- If E is an expression of type σ, then $\lambda x^\tau.E$ is an expression of type $\tau \to \sigma$.

- If E is an expression of type $\tau \to \sigma$, and F an expression of type τ, then EF is an expression of type σ.

- If E is an expression of type τ, then $\Lambda t.E$ is an expression of type $\forall t.\tau$.

- If E is an expression of type $\forall t^k.\tau$, and $L(\sigma) \leqslant k$, then $E\sigma$ is an expression of type $\tau[\sigma/t]$. ($\tau[\sigma/t]$ is the result of simultaneously substituting σ for all free occurrences of t in τ, after renaming bound variables in τ to avoid binding of variables free in σ.) Note that if $L(\sigma) > k$ then $E\sigma$ is not legal.

We define the *level* $L(E)$ of a λ-expression E as $L(type(E))$. For $n = 0, 1, ...$, $\mathbf{S}^n\mathbf{F}_2$ denotes the restriction of $\mathbf{SF}_2$ to expressions of level $\leqslant n$. Thus, $\mathbf{S}^0\mathbf{F}_2$ allows no type quantification, and is equivalent to the simply typed λ-calculus, $\mathbf{F}_1$. Clearly, the quantifier-free parametric polymorphism of ML, as well as its extension defined in (Kfoury, Tyurin, and Urzyczyn, 1988) (without recursive types, in both cases), is contained in $\mathbf{S}^1\mathbf{F}_2$.

A set theoretic model theory for $\mathbf{SF}_2$ is fairly straightforward, and does not face the complicatons of providing a semantic for $\mathbf{F}_2$ (Reynolds, 1984; Reynolds and Plotkin, 1990). A semantics for a fragment of $\mathbf{SF}_2$ is described in (Mitchell and Harper, 1988).

1.2. *Reductions and Normalization*

Like $\mathbf{F}_2$, $\mathbf{SF}_2$ has object β-reductions: $(\lambda x.E)F$ reduces to $E[F/x]$, and type β-reductions: $(\Lambda t.E)\sigma$ reduces to $E[\sigma/t]$. It is easy to verify, by induction on expressions, that object and type β-reductions, as well as η-reductions, preserve the correctness of expressions with respect to the stratification condition on type application.

Clearly, every sequence of successive reductions in $\mathbf{SF}_2$ is finite (and terminates with a normal expression), by Girard's Strong Normalization Theorem for $\mathbf{F}_2$ (Girard, 1972), since every expression of $\mathbf{SF}_2$ becomes an expression of $\mathbf{F}_2$ when stripped of level labels. We write $norm(E)$ for the normal form of E. In Section 3 we prove directly a normalization theorem for $\mathbf{SF}_2$, with far sharper computational bounds.

1.3. *The Scope of* $\mathbf{SF}_2$

This paper focuses on representation of numeric functions in $\mathbf{SF}_2$. An orthogonal question is the delineation of the λ-expressions that can, individually, be assigned types in $\mathbf{SF}_2$ or in $\mathbf{S}^n\mathbf{F}_2$ $(n \geqslant 0)$. This issue has been tackled by Pawel Urzyczyn, who has announced the following results (private communication, July 1990):

- The typing power of $\mathbf{SF}_2$ (for individual expressions) is strictly weaker than that of $\mathbf{F}_2$: The expression $(\lambda x.xyx)(\lambda z.zyz)$ can be typed in $\mathbf{F}_2$ but not in $\mathbf{SF}_2$.
- For each n, $\mathbf{S}^{n+1}\mathbf{F}_2$ has greater typing power than $\mathbf{S}^n\mathbf{F}_2$: Let $G_1 =_{\mathrm{df}} \lambda.xx$, $G_{n+1} =_{\mathrm{df}} \lambda y.\, yG_n y$; then G_{n+1} is typable in $\mathbf{S}^{n+1}\mathbf{F}_2$, but not in $\mathbf{S}^n\mathbf{F}_2$.

2. The Super-Elementary Functions Are Representable

2.1. *Function Representation*

The *Church numerals* in the untyped λ-calculus are the expressions

$$\bar{n} =_{\mathrm{df}} \lambda s \lambda z.s^{[n]}z, \qquad n = 0, 1, ...,$$

where the bracketed superscript denotes iteration. In every typed λ-calculus there are, for each type τ, *Church numerals over* τ,

$$\bar{n}^{\nu[\tau]} =_{\mathrm{df}} \lambda s^{\tau\to\tau}\lambda z^{\tau}.s^{[n]}z, \qquad n = 0, 1, ...$$

These expressions are of type $\nu[\tau] =_{\mathrm{df}} (\tau \to \tau) \to (\tau \to \tau)$. We write $\nu^*[\tau]$ for the sequence of types $\nu^0[\tau] =_{\mathrm{df}} \nu[\tau], \ldots, \nu^{i+1}[\tau] =_{\mathrm{df}} \nu^i[\tau] \to \nu^i[\tau] \equiv \nu[\nu^{i-1}[\tau]]$. (We let $\nu^{-2}[\tau] =_{\mathrm{df}} \tau$ and $\nu^{-1}[\tau] =_{\mathrm{df}} \tau \to \tau$.)

In $\mathbf{SF}_2$ there are, for each $k \geqslant 0$, *level-k polymorphic numerals*

$$\bar{n}^{\nu_k} =_{\mathrm{df}} \Lambda t^k \lambda s^{t \to t} \lambda z^t. s^{[n]} z,$$

of type $\nu_k =_{\mathrm{df}} \forall t^k. \nu[t]$. These polymorphic numerals are stratified variants of the polymorphic numerals of Fortune (1979) and O'Donnell (1979). We write ν_* for the set $\{\nu_k \mid k \geqslant 0\}$.

An expression E of type $\sigma_1 \to \cdots \to \sigma_p \to \tau$ *represents* a p-ary recursive function F (*with inputs of types* $\sigma_1, \ldots, \sigma_p$ *and output of type* τ) if the conditions $Fn_1 \cdots n_p = m$ and $E(\bar{n}_1)^{\sigma_1} \cdots (\bar{n}_p)^{\sigma_p} =_{\beta\eta} \bar{m}^\tau$ are equivalent. If $\sigma_1 = \cdots = \sigma_p = \tau$ we say that the representation of $\mapsto$ is τ.

If $\mathbf{L}$ is a typed λ-calculus (that contains the rules of $\mathbf{F}_1$), and if each one of T and S is a type, a sequence of types, or a set of types, then $Rep_{\mathbf{L}}(T; S)$ will denote the set of functions representable in $\mathbf{L}$ with inputs of types out of T, and output of type out of S. We say that a function in $Rep_{\mathbf{SF}_2} = Rep_{\mathbf{SF}_2}(\nu_*; \nu_*)$ is, simply, *representable.*

2.2. *Representation of Basic Functions*

LEMMA 1. *Z (the constant zero function), S (successor), $+$, and $\times$ are in $Rep_{\mathbf{F}_1}(\nu[0]; \nu[0])$.*

The proof is well-known and goes back to Church (see, e.g., (Fortune, Leivant, and O'Donnell, 1983)).

A function f is *defined by recurrence from g and h* if

$$f(0, \vec{x}) = g(\vec{x}),$$
$$f(Sy, \vec{x}) = h(f(y, \vec{x}), y, \vec{x}).$$

If y is not a direct argument of h in the second equation, i.e., $f(Sy, \vec{x}) = h(f(y, \vec{x}), \vec{x})$, then f is said to be defined from g and h *by iteration.*

LEMMA 2. *Suppose f is defined by iteration from $g, h \in Rep_{\mathbf{L}}(\tau; \tau)$. Then $f \in Rep_{\mathbf{L}}(\tau, \nu[\tau]; \tau)$.*

Proof. Suppose G and H represent g and h in $\mathbf{L}$, with inputs and output of type τ. Then f is represented by the expression $F =_{\mathrm{df}} \lambda y^{\nu[\tau]} \lambda \vec{x}^\tau. y(\lambda u^\tau. Hu\vec{x})(G\vec{x})$. ∎

From Lemmas 1 and 2 we obtain:

LEMMA 3. *If f is defined by one iteration from Z, S, $+$, and $\times$, then $f \in Rep_{\mathbf{F}_1}(\nu^*[0]; \nu[0])$.*

2.3. *Type Uniformization*

LEMMA 4. $Rep_{\mathbf{F}_1}(v^*[0]; v[0]) \subseteq Rep_{\mathbf{S}^1\mathbf{F}_2}(v_0; v_0)$.

Proof. Suppose a function f is represented by an expression $\lambda x_1 \cdots \lambda x_m.E$, of type $v^{j_1}[0] \to \cdots \to v^{j_m}[0] \to v[0]$. Let $y_1, ..., y_m$ be fresh variables of type $v_0 = \forall t^0.v[t]$. Then $Y_i =_{\text{df}} y_i(v^{j_i-2}[t])$ is a correctly typed expression, of type $v[v^{j_i-2}[t]] = v^{j_i}[t]$ $(i = 1, ..., m)$. Let E' be the same as E, except that every free occurrence of x_i is replaced by Y_i. Then E' is a correctly typed expression (by induction on E), of type $v[t]$. Hence $\Lambda t.E'$ is of type $\forall t.v[t] = v_0$, and $\lambda y_1 \cdots \lambda y_m.\Lambda t.E'$ is an expression that represents f over v_0. ∎

LEMMA 5. $Rep_{\mathbf{SF}_2}(v^*[v_0]; v_0) \subseteq Rep_{\mathbf{SF}_2}(v_1; v_0)$.

Proof. The proof is the same as for Lemma 4, except for the type abstraction. Suppose f is represented by some expression $\lambda x_1 \cdots x_m.E$, of type $v^{j_1}[v_0] \to \cdots \to v^{j_m}[v_0] \to v_0$. Let $y_1 \cdots y_m$ be fresh variables, of type v_1. Then $Y_i =_{\text{df}} y_i(v^{j_i-2}[v_0])$ is a correctly typed expression (since $L(v^{j_i-2}[v_0])) = 1$), of type $v[v^{j_i-2}[v_0]] = v^{j_i}[v_0]$. Let E' be E with each x_i replaced by Y_i. Then $\lambda y_1 \cdots y_m.E'$ is an expression that represents f with inputs of type v_1 and output of type v_0. ∎

LEMMA 6. *If f is defined by two iterations from Z, S, $+$, and $\times$, then $f \in Rep_{\mathbf{S}^1\mathbf{F}_1}(v_1; v_0)$.*

Proof. Suppose f is defined by iteration from functions g, h, that are in turn defined by iteration from Z, S, $+$, and $\times$. Then $g, h \in Rep_{\mathbf{F}_1}(v^*[0]; v[0])$, by Lemma 3; so $g, h \in Rep_{\mathbf{S}^1\mathbf{F}_2}(v_0; v_0)$, by Lemma 4. Therefore, by Lemma 2, $f \in Rep_{\mathbf{S}^1\mathbf{F}_2}(v^*[v_0]; v_0)$, from which $f \in Rep_{\mathbf{S}^1\mathbf{F}_2}(v_1; v_0)$, by Lemma 5. ∎

2.4. *Closure of Representable Functions under Elementary Operations*

The proof of Lemma 2 can be refined, to apply to additional forms of recurrence, as follows. For types τ, σ, define

$$(\tau, \sigma) =_{\text{df}} \forall t^l.(\tau \to \sigma \to t) \to t, \qquad \text{where} \quad l = \max(L(\tau), L(\sigma)).$$

LEMMA 7. *Suppose g is representable (in $\mathbf{SF}_2$) with inputs of types $\vec{\rho}$ and output of type τ, and h is representable with inputs of types $\tau, \sigma, \vec{\rho}$ (where σ is $v[\xi]$ for some ξ or v_l for some l) and output of type τ. Then the function f defined by recurrence from g, h is representable with inputs of types $v[(\tau, \sigma)]$, $\vec{\rho}$ and output of type τ.*

Proof. The proof builds on Kleene's representation of the predecessor function (see, e.g., Fortune, Leivant, and O'Donnell, 1983)). We use polymorphism to define a pairing function for expressions of different type. For type τ, σ, let

$$P^{\tau\sigma} =_{\mathrm{df}} \lambda x^{\tau}, y^{\sigma}. \Lambda t. \lambda u^{\tau \to \sigma \to t}. uxy \qquad (P^{\tau\sigma} \text{ is of type } \tau \to \sigma \to (\tau, \sigma)).$$

If A, B are expressions of types τ, σ, respectively, then we write $\langle A, B \rangle$ for $P^{\tau\sigma}AB$. For an expression E of type (τ, σ) we let $(E)_1$ abbreviate $E\tau(\lambda x^{\tau} y^{\sigma}. x)$, and $(E)_2$ abbreviate $E\sigma(\lambda x^{\tau} y^{\sigma}. y)$. Then $(\langle A, B \rangle)_1 =_{\beta} A$, and $(\langle A, B \rangle)_2 =_{\beta} B$.

Let G and H represent g and h, respectively, with inputs and outputs as stipulated in the lemma. Let $\mathbf{s}$ represent the successor function over σ. Define

$$\begin{aligned}
F =_{\mathrm{df}} \lambda y^{\nu[(\tau,\sigma)]}, \vec{x}.& \\
&(y \\
&\qquad \lambda q^{(\tau,\sigma)}. \langle H(q)_1 \, (q)_2 \, \vec{x}, \mathbf{s}((q)_2) \rangle \\
&\qquad \langle G\vec{x}, \bar{0}^{\sigma} \rangle \\
&)_1.
\end{aligned}$$

Then F represents f as required. ▮

Note that the proof above only requires that the output type of H be the same as the type of its first input. We conclude that the schemas of bounded iterated sum and bounded iterated product preserve representability:

Lemma 8. *If $a \in Rep_{\mathrm{SF}_2}$, then Σ_a, $\Pi_a \in Rep_{\mathrm{SF}_2}$, where $\Sigma_a(y, \vec{x}) \equiv \sum_{i<j} a(i, \vec{x})$, and $\Pi_a(y, \vec{x}) \equiv \prod_{i<j} a(i, \vec{x})$.*

Proof. We have

$$\begin{aligned}
\Sigma_a(0, \vec{x}) &= 0 \\
\Sigma_a(y+1, \vec{x}) &= \Sigma_a(y, \vec{x}) + a(y, \vec{x}), \\
\Pi_a(0, \vec{x}) &= 1 \\
\Pi_a(y+1, \vec{x}) &= \Pi_a(y, \vec{x}) \cdot a(y, \vec{x}).
\end{aligned}$$

The "recurrence functions" $h(z, y, \vec{x})$ used in these schemas are, respectively, $z + a(y, \vec{x})$, and $z \cdot a(y, \vec{x})$. Suppose a is representable by A, with inputs of types σ, $\vec{\rho}$ and output of type τ. Let $H =_{\mathrm{df}} \lambda z^{\tau} y^{\sigma} \vec{x}. Fz(Ay\vec{x})$, where F represents addition over τ. Then H represents $z + a(y, \vec{x})$, with output and first input of type τ. By Lemma 7, it follows that Σ_a is represented. The proof for Π_a is similar. ▮

2.5. *Closure of Representable Functions under Composition*

LEMMA 9. *If* $f \in Rep_{\mathbf{SF}_2}(\nu_l; \nu_k)$, *and* $d \geq 0$, *then* $f \in Rep_{\mathbf{SF}_2}(\nu_{l+d}; \nu_{k+d})$.

Proof. A straightforward induction on expressions shows that lifting all levels by d preserves legal typing. Hence, if E represents f with inputs of type ν_l and output of type ν_k, and E' arises from E by replacing each level label q by $q+d$, then E' represents f with inputs of type ν_{l+d} and output of type ν_{k+d}. ∎

LEMMA 10. *Suppose* $f \in Rep_{\mathbf{SF}_2}(\nu_{l_1} \cdots \nu_{l_m}; \nu_0)$. *Let* $q \geq l_i$ $(i=1, ..., m)$. *Then* $f \in Rep_{\mathbf{SF}_2}(\nu_q; \nu_0)$.

Proof. Suppose that f is represented by $\lambda x_1 \cdots x_m.E$, of type $\nu_{l_1} \to \cdots \to \nu_{l_m} \to \nu_0$. Let $y_1 \cdots y_m$ be fresh variables of type ν_q. Set $Y_i =_{\mathrm{df}} \Lambda t^{l_i}. y_i t$, and let $E' =_{\mathrm{df}} E[Y_1/x_1 \cdots Y_m/x_m]$. By induction on E, E' is seen to be a legal expression, and $E[\bar{n}^{\nu_{l_i}}/x_i] =_\beta E'[\bar{n}^{\nu_q}/y_i]$, for all $n \geq 0$. Thus $\lambda y_1 \cdots y_m.E'$ is a legal expression that represents f with inputs of type ν_q and output of type ν_0. ∎

LEMMA 11. *Suppose that* $f(\vec{x}) = h(\vec{g}(\vec{x}))$, *where* $\vec{x} = (x_1 \cdots x_n)$, $\vec{g} = (g_1 \cdots g_k)$, $g_i \in Rep_{\mathbf{SF}_2}$ $(i=1, ..., k)$, *and* $h \in Rep_{\mathbf{SF}_2}$. *Then* $f \in Rep_{\mathbf{SF}_2}$.

Proof. By Lemma 10 there is a sufficiently large q such that each g_i is represented by an expression G_i with inputs of type ν_q and output of type ν_0 $(i=1, ..., k)$, and with h represented by an expression H with inputs of type ν_q and output of type ν_0. By Lemma 9 there are expressions G'_i representing g_i with inputs of type ν_{2q} and output of type ν_q. Thus,

$$F =_{\mathrm{df}} \lambda x_1^{\nu_{2q}} \cdots x_n^{\nu_{2q}}. H(G'_1 \vec{x}) \cdots (G'_k \vec{x})$$

represents f with inputs of type ν_{2q} and output of type ν_0. ∎

2.6. *All Super-Elementary Functions are Representable*

The Grzegorczyk class $\mathscr{E}_k$ $(k \geq 0)$ is generated by composition and bounded recurrence from Z, S, the projection functions, and the function F_k, where $F_0 =_{\mathrm{df}} S$, $F_1 =_{\mathrm{df}} \lambda x.2x$, $F_2 =_{\mathrm{df}} \lambda x.x^2$, and $F_{k+1}(x) =_{\mathrm{df}} F_k^{[x]}(x)$ for $k \geq 2$ ($F^{[n]}$ being the nth iterate of F). $\mathscr{E}_3$ is Kalmar's class of *elementary* functions, and the functions in $\mathscr{E}_4$ are dubbed *super-elementary*. We have $\mathscr{PR} = \bigcup_k \mathscr{E}_k$ (Grzegorczyk, 1953). (For details see, e.g., (Rose, 1984).)

The following is stated in (Statman, 1981) without proof.

LEMMA 12. *Every super-elementary function is representable in* $\mathbf{SF}_2$.

Proof. The predecessor function is in $Rep_{\mathbf{F}_1}(\nu^*[0]; \nu[0])$ (see (Fortune, Leivant, and O'Donnell, 1983)), so, by Lemma 4, also in $Rep_{\mathbf{SF}_2}(\nu_0; \nu_0)$. By

Lemma 2 the cut-off subtraction function is then in $Rep_{\mathrm{SF}_2}(v^*[v_0]; v_0)$, and so also in $Rep_{\mathrm{SF}_2}(v_1; v_0)$, by Lemma 5.

The initial primitive recursive functions are trivially representable, as is addition (Lemma 1). By Lemma 8, the class of representable functions is closed under bounded iterated sum and product, and by Lemma 11 also under composition. Since $\mathscr{E}_3$ is the same as the class of functions generated from the initial functions, $+$, and $\dot{-}$, by composition, bounded iterated sum, and bounded iterated product (Grzegorczyk, 1953), it follows that all elementary functions are representable.

Since F_4 is defined from addition by two iterations, it follows from Lemma 6 that F_4 is also representable.

A standard construction shows that bounded recurrence can be defined in terms of composition with elementary functions, bounded minimalization and bounded quantification. (The construction is essentially due to Kleene; see, e.g., (Rose, 1984, proof of Theorem 1.3.1, p. 11), where bounded product is also used.) Bounded minimalization and bounded quantification are easily definable in terms of elementary functions and bounded sum and product (see, e.g., (Rose, 1984, Sect. I)). It follows, by Lemma 8, that the class of representable functions is closed under bounded recurrence.

The lemma now follows from the definition above of $\mathscr{E}_4$. ∎

3. The Representable Functions Are Super-Elementary

3.1. *Complexity of Cuts*

For a λ-expression E, a sub-expression F of E is a *cut* if F is the left immediate sub-expression of a redex FG or $F\sigma$ in E. We write $cut(E)$ for the set of cut sub-expressions of E.

Define the following functions on expressions E and types τ:

$$CL(E) =_{\mathrm{df}} \max\{L(F) \mid F \in cut(E)\}$$

$D_l(\tau) =_{\mathrm{df}}$ negative-nesting count in τ of subtypes of level $\geqslant l$; i.e.,

$$D_l(t^k) =_{\mathrm{df}} \begin{cases} 0 & \text{if } k < l \\ 1 & \text{otherwise} \end{cases}$$

$$D_l(\sigma \to \tau) =_{\mathrm{df}} \max(\dot{s}D_l(\sigma), D_l(\tau))$$

for $l > 0$, where

$$\dot{s}x =_{\mathrm{df}} \text{if } x = 0 \text{ then } 0 \text{ else } x + 1$$

$$D_l(\forall t^k.\tau) =_{\mathrm{df}} \begin{cases} \max(1, D_l(\tau)) & \text{if } k+1 \geqslant l \\ D_l(\tau) & \text{otherwise} \end{cases}$$

$$D_l(E) =_{\mathrm{df}} D_l(type(E))$$

$$CD_l(E) =_{\mathrm{df}} \max\{D_l(F) \mid F \in cut(E)\}$$

$$CD(E) =_{\mathrm{df}} CD_l(E), \qquad \text{where} \quad l = CL(E)$$

$$\delta_{ld}(E) =_{\mathrm{df}} \begin{cases} 1 & \text{if } E \text{ is a redex } G\alpha \text{, with } L(G) \geqslant l \text{ and } D_l(G) \geqslant d \\ 0 & \text{otherwise} \end{cases}$$

$M_{ld}(E) =_{\mathrm{df}}$ the maximal length of any chain of nested redexes $G\alpha$ with $L(G) \geqslant l$ and $D_l(G) \geqslant d$; i.e.,

$$M_{ld}(x) =_{\mathrm{df}} 0$$

$$M_{ld}(GH) =_{\mathrm{df}} \delta_{ld}(GH) + \max(M_{ld}(G), M_{ld}(H)),$$

$$M_{ld}(G\sigma) =_{\mathrm{df}} \delta_{ld}(G\sigma) + M_{ld}(G)$$

$$M_{ld}(\lambda x . G) =_{\mathrm{df}} M_{ld}(G)$$

$$M_{ld}(\Lambda t . G) =_{\mathrm{df}} M_{ld}(G)$$

$$M(E) =_{\mathrm{df}} M_{ld}(E), \qquad \text{where} \quad l = CL(E) \text{ and } d = CD(E).$$

Note that $D_k(\tau) \leqslant D_l(\tau)$ for $k \geqslant l$, by the definition of D_l.

3.2. *Preservation of Cut-Complexity under Substitution*

LEMMA 13. *Suppose that $CL(E)$, $CL(F)$, $L(F) \leqslant l$, and $CD_l(E)$, $CD_l(F)$, $D_l(F) < d$. Let $E' \equiv_{\mathrm{df}} E[F/x]$. Then $CL(E') \leqslant l$, and $CD_l(E') < d$.*

Proof. Induction on E, by cases.

1. E is a variable y. If y is x, then $E' \equiv F$; otherwise $E' \equiv E$. In either case the lemma is immediate.

2. E is of the form $E_0 E_1$, so $E' \equiv E'_0 E'_1$ (where $E'_i \equiv_{\mathrm{df}} E_i[F/x]$). By induction assumption $CL(E'_i) \leqslant l$ and $CD_l(E'_i) < d$ $(i = 0, 1)$.

There are three sub-cases.

2(i) E' is not a redex. Then $CL(E') = \max(CL(E'_0), CL(E'_1)) \leqslant l$, and $CD_l(E') = \max(CD_l(E'_0), CD_l(E'_1)) \leqslant d$.

2(ii) E is a redex. Then $L(E_0) \leqslant l$, $D_l(E_0) < d$. Since $type(E'_0) = type(E_0)$, these imply $L(E'_0) \leqslant l$, $D_l(E'_0) < d$. Hence $CL(E') = \max(CL(E'_0), CL(E'_1), L(E'_0)) \leqslant l$, $CD_l(E') = \max(CD_l(E'_0), CD_l(E'_1), D_l(E'_0)) < d$.

2(iii) E' is a redex, but E is not a redex. Then $E_0 \equiv x$ and $E' \equiv FE'_1$. Since $L(F) \leqslant l$ and $D_l(F) < d$, $CL(E') = \max(CL(E'_0), CL(E'_1), L(F)) \leqslant l$, and $CD_l(E') = \max(CD_l(E'_0), CD_l(E'_1), D_l(F)) < d$.

3. E is of the form $\lambda u.E_0$. Then $CL(E) = CL(E_0) \leqslant l$ and $CD_l(E) = CD_l(E_0) < d$, so $CL(E') = CL(E'_0) \leqslant l$ and $CD_l(E') = CD_l(E'_0) < d$, by induction assumption.

4. E is of the form $\Lambda t.E_0$ or of the form $E_0\sigma$. These are similar to case (3). ∎

LEMMA 14. *Suppose $L(E) \leqslant l$, $D_l(E) < d$, and $L(\sigma) < l$. Let $E' \equiv_{\mathrm{df}} E[\sigma/t]$. Then $L(E') \leqslant l$ and $D_l(E') < d$.*

Proof. If τ is the type of a cut in E, then $\tau' =_{\mathrm{df}} \tau[\sigma/t]$ is the type of the corresponding cut in E'. If $L(\tau) < l$, then $L(\tau') < l$. If $L(\tau) = l$, then, by a trivial induction on τ, $L(\tau') = L(\tau) \leqslant l$, and $D_l(\tau') = D_l(\tau)$, so $D_l(E') < d$. ∎

3.3. *Canonical Reductions*

Let E be a λ-expression. A redex $G\alpha$ in E (where α is a type or a λ-expression) is *canonical* if it is an innermost cut of the largest level-degree complexity in E; that is, $l =_{\mathrm{df}} L(G) = CL(E)$, $d =_{\mathrm{df}} D_l(G) = CD(E)$, $M_{ld}(G) = 0$, and, if α is a λ-expression, $M_{ld}(\alpha) = 0$.

E reduces canonically to E', $E \Rightarrow_{\mathrm{c}} E'$, if E' is the result of reducing all canonical redexes of E (the order makes no difference, since no canonical redex occurs within another).

LEMMA 15. *Suppose $E \Rightarrow_{\mathrm{c}} E'$, $CL(E) = l$, $CD(E) = d$. Then $CL(E') \leqslant l$, $CD_l(E') \leqslant d$, and $M_{ld}(E') < M_{ld}(E)$.*

Proof. By induction on E. The only non-trivial case is where E is a (unique) canonical redex of itself. We have two cases, corresponding to the two sorts of redex.

Case 1. E is of the form $(\lambda x^\tau.E_0^\sigma)F$, and $E' \equiv E_0[F/x]$. Since E is a critical redex, $L(\tau \to \sigma) = l$, $D_l(\tau \to \sigma) = d$, and $M_{ld}(E_0) = M_{ld}(F) = 0$. We claim that $D_l(F) < d$: if $L(F) = L(\tau) < l$, then $D_l(F) = 0 < d$ ($d > 0$, by definition of CD); if $L(F) = l$, then $D_l(\tau) < D_l(\tau \to \sigma) = d$. Thus, by Lemma 13, $CL(E') \leqslant l$ and $CD_l(E') < d$, so $M_{ld}(E') = 0 < 1 = M_{ld}(E)$.

Case 2. E is of the form $(\Lambda s.E_0)\sigma$, and $E' \equiv E_0[\sigma/t]$. By the stratification condition on type application, $L(\sigma) \leqslant L(s) < l$. Hence, by Lemma 14, $L(E') \leqslant l$ and $D_l(E') < d$, so $M_{ld}(E') = 0 < 1 = M_{ld}(E)$. ∎

For a λ-expression E, let $\mu(E) =_{\mathrm{df}} (CL(E), CD(E), M(E))$.

LEMMA 16. *If $E \Rightarrow_{\mathrm{c}} E'$, then $\mu(E') \prec \mu(E)$, where $\prec$ is the lexicographic ordering.*

Proof. Let $l = CL(E)$, $d = CD(E)$, $m = M(E)$, $l' = CL(E')$, $d' = CD(E')$, and $m' = M(E')$.

By Lemma 15, $l' \leqslant l$. If $l' < l$, then $\mu(E') \prec \mu(E)$. If $l' = l$, then $d' = CD_{l'}(E') = CD_l(E') \leqslant CD(E) = d$, by Lemma 15. If $d' < d$, then $\mu(E') \prec \mu(E)$. If $d = d'$, then $m' = CM(E') = CM_{l'd'}(E') = CM_{ld}(E') < CD(E) = m$, again by Lemma 15. ▮

3.4. *Super-Elementary Bounds on Length of Normal Forms*

For an expression E, let

$$GD(E) =_{\mathrm{df}} \max\{D_l(F) \mid F \text{ a sub-expression of } E, l \leqslant L(E)\},$$

$$|E| =_{\mathrm{df}} \text{the height of the applicative part of } E,$$

i.e.,

$$|x| = 0$$

$$|FG| = \max(|F|, |G|) + 1$$

$$|F\tau| = |F| + 1$$

$$|\lambda x.F| = |\lambda t.F| = |F|.$$

We collect some trivial properties of these measures in the following:

LEMMA 17. 1. *$D_l(E) \leqslant GD(E)$ for all l;*

2. *$M_{ld}(E) \leqslant |E|$ for all l, d;*

3. *If $E \Rightarrow_{\mathrm{c}} E'$, then $|E'| \leqslant 2 \cdot |E|$ (and so $M(E') \leqslant 2 \cdot |E|$), and $GD(E') \leqslant GD(E)$.*

We define primitive recursive functions h_l, $l \geqslant 0$, by the following recursions with parameter substitution (cf., e.g., (Rose, 1984, Sect. 1.3):

$$h_0(0, 0, x, g) = x$$

$$h_l(d, m+1, x, g) = h_l(d, m, 2x, g)$$

$$h_l(d+1, 0, x, g) = h_l(d, x, x, g)$$

$$h_{l+1}(0, 0, x, g) = h_l(g, x, x, g).$$

Clearly, each h_l is non-decreasing in each one of its arguments, since we use in the definitions only non-decreasing functions. Also, $h_k(\vec{a}) \geqslant h_l(\vec{a})$ for $k > l$. (Detailed proofs are by nested inductions on l, d, g, and m.)

LEMMA 18. *h_l is super-elementary for all l.*

Proof. Let

$$\eta(0, m, x) = 2^m \cdot x$$

$$\eta(d+1, m, x) = 2^{\eta(d,m,x)} \cdot \eta(d, m, x).$$

The function η is defined by a single recurrence from elementary functions, and is therefore super-elementary (see, e.g., (Rose, 1984) or (Schwichtenberg, 1969)).

Claim 1. $\eta(d, m+1, x) = \eta(d, m, 2x)$ for all arguments. The proof is straightforward by induction on d.

Claim 2. $\eta(d, x, x) = \eta(d+1, 0, x)$ for all arguments. Again, a straightforward induction on d.

Claim 3. $h_0(d, m, x, g) = \eta(d, m, x)$ for all arguments. The proof is by main induction on d, using Claim 2, and secondary induction on m, using Claim 1.

Claim 4. $h_{l+1}(d, m, x, g) = h_l(g, \eta(d, m, x), \eta(d, m, x), g)$ for all l and all arguments. The proof is by main induction on d and secondary induction on m. We have

$$\begin{aligned}
&h_{l+1}(0, 0, x, g) \\
&\qquad = h_l(g, x, x, g) \\
&\qquad = h_l(g, \eta(0, 0, x), \eta(0, 0, x), g), \\
&h_{l+1}(d, m+1, x, g) \\
&\qquad = h_{l+1}(d, m, 2x, g) \\
&\qquad = h_l(g, \eta(d, m, 2x), \eta(d, m, 2x), g) && \text{by induction assumption} \\
&\qquad = h_l(g, \eta(d, m+1, x), \eta(d, m+1, x), g) && \text{by Claim 1,}
\end{aligned}$$

and

$$\begin{aligned}
&h_{l+1}(d+1, 0, x, g) \\
&\qquad = h_{l+1}(d, x, x, g) \\
&\qquad = h_l(g, \eta(d, x, x), \eta(d, x, x), g) && \text{by induction assumption} \\
&\qquad = h_l(g, \eta(d+1, 0, x), \eta(d+1, 0, x), g) && \text{by Claim 2.}
\end{aligned}$$

It now follows that every h_l is super-elementary, by induction on l. Claim 3 establishes the induction's basis. h_{l+1} is defined by composition from η and h_l, which by the induction assumption is super-elementary; hence h_{l+1} is super-elementary. ∎

LEMMA 19. *If* $\mu(E) = (l, d, m)$ *then* $|norm(E)| \leqslant h_l(d, m, |E|, GD(E))$.

Proof. By (course-of-value) induction on (l, d, m), i.e., main induction on l, secondary induction on d, and ternary induction on m.

If $m=0$, then E is normal and $E=E'$, $l=d=0$. We have $|norm(E)| = |E| = h_0(0, 0, |E|, g)$ for any g.

Suppose $M(E)=m+1$. Let $E \Rightarrow_c E'$, so $M_{ld}(E')=m$, and $|E'| \leqslant 2 \cdot |E|$.

Case 1. $L(E')=l$ and $D(E')=d$, so $M(E')=M_{ld}(E')=m$.

$$\begin{aligned} |norm(E)| &= |norm(E')| \\ &\leqslant h_l(d, m, |E'|, GD(E')) && \text{by induction assumption} \\ &\leqslant h_l(d, m, 2 \cdot |E|, GD(E)) && \text{since } |E'| \leqslant 2 \cdot |E| \\ & && \text{and } GD(E') \leqslant GD(E) \\ &= h_l(d, m+1, |E|, GD(E)). \end{aligned}$$

Case 2. $L(E')=l$ and $d' =_{\text{df}} D(E') < d$, so $m=0$.

$$\begin{aligned} |norm(E)| &= |norm(E')| \\ &\leqslant h_l(d', M(E'), |E'|, GD(E')) && \text{by induction assumption} \\ &\leqslant h_l(d-1, 2 \cdot |E|, 2 \cdot |E|, GD(E)) && \text{since} \quad d' \leqslant d-1 \\ &= h_l(d, 0, 2 \cdot |E|, GD(E)) && \text{by definition of } h_l \\ &= h_l(d, 1, |E|, GD(E)) \\ &= h_l(d, m+1, |E|, GD(E)). \end{aligned}$$

Case 3. $l' =_{\text{df}} L(E') < l$, so $m=0$.

$$\begin{aligned} |norm(E)| &= |norm(E')| \\ &\leqslant h_{l'}(GD(E'), M(E'), |E'|, GD(E')) && \text{by induction assumption} \\ &\leqslant h_{l-1}(GD(E), 2 \cdot |E|, 2 \cdot |E|, GD(E)) && \text{since} \quad h_{l-1} \geqslant h_{l'} \\ &= h_l(0, 0, 2 \cdot |E|, GD(E)) && \text{by definition} \\ &= h_l(0, 1, |E|, GD(E)) \\ &\leqslant h_l(d, m+1, |E|, GD(E)). \quad \blacksquare \end{aligned}$$

3.5. *Super-Elementary Normalization Functions*

We turn to exact normalization functions for $\mathbf{SF}_2$. For each $l \leqslant 0$ we show that the normalization function for $\mathbf{S}^l\mathbf{F}_2$, as a function on codes of expressions, is super-elementary.

Fix a canonical (Gödel-) coding of expressions, $E \mapsto \# E$, with elementary functions $\hat{l}$, $\hat{d}$, $\hat{m}$, $\hat{a}$, and $\hat{r}$, such that for every expression E, $\hat{l}(\# E) = CL(E)$, $\hat{d}(\# E) = CD(E)$, $\hat{m}(\# E) = M(E)$, $\hat{a}(\# E) = |E|$, and if $E \Rightarrow_c E'$

then $\hat{r}(\#E) = \#(E')$. Such functions can easily be defined so as to return 0 when the argument is not the code of an expression. For $l \geqslant 0$ we define the function $\hat{n}_l$ by

$$\hat{n}_l(d, m, x) = 0 \qquad \text{if either } x \text{ is not the code of an expression,}$$
$$l > \hat{l}(x),\ d > \hat{d}(x),\ \text{or } m > \hat{m}(x);$$

Otherwise:

$$\hat{n}_l(d, m, x) = \hat{n}_{l-1}(d, m, x) \qquad \text{if} \quad \hat{l}(x) < l;$$

Otherwise:

$$\hat{n}_0(0, 0, x) = x$$
$$\hat{n}_l(d, m+1, x) = \hat{n}_l(d, m, \hat{r}(x))$$
$$\hat{n}_l(d+1, 0, x) = \hat{n}_l(\hat{d}(\hat{r}(x)), \hat{m}(\hat{r}(x)), \hat{r}(x))$$
$$\hat{n}_{l+1}(0, 0, x) = \hat{n}_l(\hat{d}(\hat{r}(x)), \hat{m}(\hat{r}(x)), \hat{r}(x)).$$

LEMMA 20.

If $\mu(E) = (l, d, m)$ *then* $\#norm(E) = \hat{n}(CL(E), CD(E), M(E), \#E)$.

Proof. Straightforward, by nested course-of-value induction on l, d, and m. ∎

Let

$$N_l(x) =_{\mathrm{df}} \begin{cases} \hat{n}_{\hat{l}(x)}(\hat{d}(x), \hat{m}(x), x) & \text{if } x = \#E \text{ for some } E \text{ with } CL(E) \leqslant l, \\ 0 & \text{otherwise} \end{cases}$$

LEMMA 21. *For each* $l \geqslant 0$, *the function* N_l *is super-elementary.*

Proof. For each $l \geqslant 0$, N_l is defined from elementary functions by composition and course-of-value recursion with parameter substitution. The latter can be converted to instances of (simple) recurrence (see, e.g., (Rose, 1984, Sect. 1.3)). Moreover, all these recurrences are bounded by functions elementary in h_l, by Lemma 19. Since, by definition, $\mathscr{E}_4$ is closed under bounded recurrence, it follows that N_l is super-elementary. ∎

3.6. *The Representable Functions are Super-Elementary*

THEOREM 22. $Rep_{\mathbf{SF}_2} = \mathscr{E}_4$.

Proof. We have $Rep_{\mathbf{SF}_2} \supseteq \mathscr{E}_4$ by Lemma 12.

For the converse, suppose that E represents in $\mathbf{S}^l\mathbf{F}_2$ an m-ary function f, with inputs of type $v_{l_1} \cdots v_{l_m}$ and output of type v_0 $(l_1 \cdots l_m < l)$. Then, for every $k_1 \cdots k_m \geqslant 0$, $norm(E\bar{k}_1^{\nu_{l_1}} \cdots \bar{k}_m^{\nu_{l_m}})$ is $\bar{v}^{\nu_0}$, where $v =_{\mathrm{df}} f(k_1, ..., k_m)$. Note that $|\bar{v}^{\nu_0}| = f(k_1, ..., k_m)$. Let $c(k_1, ..., k_m) =_{\mathrm{df}} \#(E\bar{k}_1 \cdots \bar{k}_m)$, which is an elementary function. Then $f(k_1, ..., k_m) = \hat{a}(N_l(c(k_1, ..., k_m)))$. Thus, by Lemma 21, f is the composition of super-elementary functions, and so it is super-elementary. ▮

4. Limitative Properties of the Stratified Calculus

4.1. *Length of Reduction Sequences*

The representability of all super-elementary functions implies that there is no super-elementary function that bounds the length of reduction sequences.

Lemma 23. *There is no super-elementary function b such that, for every expression E of* $\mathbf{SF}_2$, *$b(|E|) \geqslant$ the length of the shortest reduction sequence starting with E.*

Proof. Suppose b were a function as above; then $c(x) =_{\mathrm{df}} 2^{b(x+1)} \cdot (x+2)$ is also super-elementary, and therefore represented by some expression C. Then, for any $k \geqslant |C|$,

$$b(k+1) = b(|C\bar{k}|) \geqslant \text{the length of the shortest reduction sequence starting with } C\bar{k}.$$

Since a reduction on an expression E at most doubles $|E|$, this implies that

$$\begin{aligned} c(k) &> 2^{b(k+1)} \cdot (k+1) && \text{by definition of } c \\ &= 2^{b(k+1)} \cdot |C\bar{k}| \\ &\geqslant |norm(C\bar{k})| && \text{by the property above and the assumption on } b \\ &= c(k) && \text{since } C \text{ represents } c, \end{aligned}$$

a contradiction. ▮

Theorem 24. *There is no super-elementary function B such that, for every expression E of* $\mathbf{SF}_2$, *$B(\#E) \geqslant$ the length of the shortest reduction sequence starting with E.*

Proof. Suppose B were a function as above. Let

$$b(x) =_{\mathrm{df}} \max\{B(e) \mid e = \#E \text{ for some } E \text{ with no vacuous abstractions, and with } |E| \leqslant x\}.$$

The number of expressions E as above is exponential in x, so b is elementary in B, and if B is super-elementary then so is b. Since vacuous abstractions have no effect on reductions, we have, for every expression E,

$$b(|E|) \geqslant \text{the length of the shortest reduction sequence starting with } E,$$

contradicting Lemma 23. ▮

4.2. *Complexity of Equality*

Given a λ-calculus $\mathbf{L}$, the *equality problem* for $\mathbf{L}$, $Eq[\mathbf{L}]$, is the problem of deciding, given two expressions of $\mathbf{L}$, whether they are β-equal. Statman (1979) showed that $Eq[\mathbf{F}_1] \in \mathscr{E}_4 - \mathscr{E}_3$.

THEOREM 25. $Eq[\mathbf{SF}_2] \in \mathscr{E}_5 - \mathscr{E}_4$.

Proof. Let $H(l, \vec{x}) =_{\mathrm{df}} h_l(\vec{x})$. By Lemma 18, H is defined by course-of-value recurrence with parameter substitution from $\eta \in \mathscr{E}_4$:

$$H(0, d, m, x, g) = \eta(d, m, x)$$

$$H(l+1, d, m, x, g) = H(l, g, \eta(d, m, x), \eta(d, m, x), g).$$

So $H \in \mathscr{E}_5$ (see (Rose, 1984) or (Schwichtenberg, 1969)). Let $N'(x) =_{\mathrm{df}} N_{f(x)}(x)$; then $N' \in \mathscr{E}_5$, since N' is definable by recurrences bounded by functions elementary in H. It follows that the function

$$eq(x, y) =_{\mathrm{df}} \begin{cases} 1 & \text{if } \; N'(x) = N'(y) \\ 0 & \text{otherwise} \end{cases}$$

is in $\mathscr{E}_5$, and decides β-equality of expressions of $\mathbf{SF}_2$. Thus $Eq[\mathbf{SF}_2] \in \mathscr{E}_5$.

Suppose $Eq[\mathbf{SF}_2] \in \mathscr{E}_4$. Let $\{E_n\}_n$ be an elementary enumeration of all λ-expressions of $\mathbf{SF}_2$. The assumption implies that the function

$$f(n) =_{\mathrm{df}} \begin{cases} 1 & \text{if } \; E_n \bar{n} =_\beta \bar{0} \\ 0 & \text{otherwise} \end{cases}$$

is in $\mathscr{E}_4$, hence representable by some $E_k \in \mathbf{SF}_2$. But then $E_k \bar{k} =_\beta \bar{0}$ iff $E_k \bar{k} =_\beta \bar{1}$, a contradiction. ▮

5. STRATIFIED POLYMORPHISM WITH TYPE RECURSION

5.1. *Recursive Types*

Suppose τ is a type expression of $\mathbf{F}_2$ in which the type variable t has no free negative occurrences (an occurrence is negative if it is in the negative scope of an odd number of $\rightarrow$). Then $t \mapsto \tau$, understood as a set theoretic operation, is positive, and has a minimal fixpoint (Aczel, 1977; Mendler, 1987). Let $\mu t.\tau$ be a new type expression, intended to denote that minimal fixpoint. (Mendler, 1987; Leivant, 1990a) discuss several calculi in which $\mathbf{F}_2$ is augmented with constants and reduction rules, intended to convey that meaning of $\mu t.\tau$. We briefly describe the stratified variants of two of these.

Let $\mathbf{F}_2\mathbf{I}$ be $\mathbf{F}_2$ augmented with type expressions $\mu t.\tau$ for every τ and t non-negative in τ; with, for each such $\delta \equiv \mu t.\tau$, a *closure constant* $\mathbf{C}_\delta$, of type $\tau[\delta] \rightarrow \delta$, and an *induction constant* $\mathbf{I}_\delta$, of type $\forall s.(\forall t.((t \rightarrow s) \rightarrow \tau \rightarrow s) \rightarrow \delta \rightarrow s)$; and with a new *closure reduction*, mapping $\mathbf{I}_\delta \sigma E(\mathbf{C}_\delta F)$ to $E\delta(\mathbf{I}_\delta \sigma E)F$ (σ an arbitrary type, $\tau[\delta] =_{\mathrm{df}} \tau[\delta/t]$, E of type $\forall t.((t \rightarrow \sigma) \rightarrow \tau \rightarrow \sigma)$, and F of type $\tau[\delta]$).

PROPOSITION 26. *A stratified version* $\mathbf{SF}_2\mathbf{I}$ *of* $\mathbf{F}_2\mathbf{I}$ *must have* $L(\mu t^l.\tau) = l = L(\tau)$.

Proof. If the type of $\mathbf{I}_\delta$ is $\forall s^m.(\forall t^l.((t \rightarrow s) \rightarrow \tau \rightarrow s) \rightarrow \delta \rightarrow s)$, then the type of E in a reduction as above is $\forall t^l.((t \rightarrow \sigma) \rightarrow \tau \rightarrow \sigma)$. Since δ is an argument of E, $L(\delta) \leqslant l$. On the other hand, except for the trivial case where t is not free in τ, τ is of level $\geqslant l$. Hence, to permit the type $\mu t^l.\tau$, with t free in τ, we should have $L(\mu t^l.\tau) = l$. ∎

Proposition 26 states that an inductively generated type has the same level as the level of the operator defining it. This bit of impredicativity is implicit in a number of foundational contexts, notably in the justification of induction (Leivant, 1990c). We conjecture that, as a result, there are numeric functions representable in $\mathbf{SF}_2\mathbf{I}$ that are not representable in $\mathbf{SF}_2$. This would be in contrast with the innocuous computational effect of adding recursive types to $\mathbf{F}_2$: Every function representable in of $\mathbf{F}_2\mathbf{I}$ is provably recursive in second-order arithmetic (Mendler, 1987), and is therefore already representable in $\mathbf{F}_2$ (Girard, 1972).

Another extension of $\mathbf{F}_2$ with recursive types, $\mathbf{F}_2\boldsymbol{\mu}$, has recursive types $\delta = \mu t.\tau$ as above, but no new constants or reductions. Instead, $\mathbf{F}_2\boldsymbol{\mu}$ liberalizes the typing conditions of $\mathbf{F}_2$, as follows. Let $=_\mu$ be the symmetric and transitive closure of the relation that holds between types α and β if β results from replacing in α an ocurrence of δ by $\tau[\delta]$, for some $\delta = \mu t.\tau$. If $E: \sigma \rightarrow \rho$, and $F: \sigma'$, with $\sigma =_\mu \sigma'$, then we let EF be a legal expression, of type ρ. $\mathbf{F}_2\boldsymbol{\mu}$ is consistent with $\mu t.\tau$ being interpreted as any fixpoint of $t \mapsto \tau$, not necessarily the minimal one. In a stratified version $\mathbf{SF}_2\boldsymbol{\mu}$ of $\mathbf{F}_2\boldsymbol{\mu}$, the requirement $L(\mu t^l.\tau) = l = L(\tau)$ is immediate, from the explicit identification of δ with $\tau[\delta]$ in the typing rules.

5.2. *Algorithms Representable Using Recursive Types*

Although adding recursive types to $\mathbf{F}_2$ does not result in new functions being representable, it does allow new *algorithms* to be typed. Consider the function **if** $y > x$ **then** b **else** a, an equational program for which is

$$f(\mathbf{s}(x), y, a, b) = f(y, x, b, a)$$
$$f(0, y, a, b) = a.$$

A λ-representation for this program, relative to Church numerals, was invented by Maurey (reported in (Krivine, 1987)): Let $F =_{\mathrm{df}} \lambda f, g.\, gf$, $A =_{\mathrm{df}} \lambda u.a$, and $B =_{\mathrm{df}} \lambda u.b$. Then

$$\begin{aligned} F^{[n+1]}A(F^{[m]}B) &= F(F^{[n]}A)(F^{[m]}B) \\ &=_\beta F^{[m]}B(F^{[n]}A), \\ F^{[0]}A(F^{[m]}B) &= A(F^{[m]}B) \\ &=_\beta a, \\ F^{[0]}B(F^{[m]}A) &= B(F^{[m]}A) \\ &=_\beta b. \end{aligned}$$

So f is represented by the expression $M = \lambda x, y, a, b.xFA(yFB)$.

While this expression cannot be typed, for Church numerals as input and output, in $\mathbf{F}_2$ (Krivine, 1987), we have:

PROPOSITION 27. *Maurey's algorithm can be typed, as a function over* v_0, *in* $\mathbf{S}^1\mathbf{F}_2\boldsymbol{\mu}$.

Proof. Let s be a type variable of level 0, $\sigma =_{\mathrm{df}} v[s]$, and $\delta =_{\mathrm{df}} \mu t^0.(t \to \sigma) \to \sigma$. So $\delta =_\mu (\delta \to \sigma) \to \sigma$. Hence $F_1 =_{\mathrm{df}} \lambda f^{\delta \to \sigma} g^{\delta}.\, gf$ is correctly typed, and has type $(\delta \to \sigma) \to \delta \to \sigma$, and $F_2 =_{\mathrm{df}} \lambda f^{\delta} g^{\delta \to \sigma}.\, gf$ is correctly typed, and has type $\delta \to (\delta \to \sigma) \to \sigma =_\mu \delta \to \delta$. Also, $A =_{\mathrm{df}} \lambda u^{\delta}.a^{v_0}s$ is of type $\delta \to \sigma$, and $B =_{\mathrm{df}} \lambda u^{\delta \to \sigma}.b^{v_0}s$ is of type δ.

It follows that the expression

$$\lambda x^{v_0} y^{v_0} a^{v_0} b^{v_0}.\Lambda s.x(\delta \to \sigma)\, F_1 A(y\delta F_2 B)$$

is a typed form of M, in $\mathbf{S}^1\mathbf{F}_2\boldsymbol{\mu}$, which represents Maurey's Algorithm over v_0. ∎

Acknowledgments

I am grateful to Pawel Urzyczyn for detailed comments on a preliminary version of this work. Research partially supported by ONR Grant N00014-84-K-0415 and by DARPA Grant F33615-87-C-1499, ARPA order 4976, Amendment 20.

RECEIVED February 21, 1990; FINAL MANUSCRIPT RECEIVED October 23, 1990

References

ACZEL, P. (1977), An introduction to inductive definitions, *in* "Handbook of Mathematical Logic" (J. Barwise, Ed.), pp. 739–782, North-Holland, Amsterdam.

FORTUNE, S., LEIVANT, D., AND O'DONNELL, M. (1983), The expressiveness of simple and second order type structures, *J. Assoc. Comput. Mach.* **30**, 151–185.

FORTUNE, F. (1979), Ph.D. Dissertation, Cornell University.

FRIEDMAN, H. (1975), Equality between functionals, *in* "Logic Colloquium" (R. Parikh, Ed.), pp. 23–37, Lecture Notes in Mathematics, Vol. 453, Springer-Verlag, Berlin.

GIRARD, J.-Y. (1972), "Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur," Thèse de Doctorat d'Etat, Paris.

GRZEGORCZYK, A. (1953), Some classes of recursive functions, *in* "Rozprawy Mat. IV," Warsaw.

KREISEL, G. (1960), La prédicativité, *Bull. Soc. Math. France* **88**, 371–391.

KRIVINE, J.-L. (1987), Un algorithme non typable dans le système *F*, *C.R. Acad. Sci. Paris Sér. I Math.* **304**, 123–126.

KFOURY, A. J., TYURIN, J., AND URZYCZYN, P. (1988), A proper extension of ML with an effective type assignment, *in* "Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages," pp. 58–69, Assoc. Comput. Mach., New York.

LEIVANT, D. (1989), Stratified polymorphism, *in* "Proceedings of the Fourth Annual Symposium of Logic in Computer Science," pp. 39–47, IEEE Computer Society, Washington, DC.

LEIVANT, D. (1990a), Contracting proofs to programs, *in* "Logic and Computer Science" (P. Odifreddi, Ed.), pp. 279–327, Academic Press, 1990.

LEIVANT, D. (1990b), Discrete polymorphism, *in* "Proceedings of the Sixth ACM Conference on LISP and Functional Programming," pp. 288–297.

LEIVANT, D. (1990c), Computationally based set existence principles, *in* "Logic and Computation" (W. Sieg, Ed.), pp. 197–211, Contemporary Mathematics, Vol. 106, Amer. Math. Soc., Providence, RI.

MENDLER, N. P. (1987), Recursive types and type constraints in second-order lambda calculus, *in* "Proceedings, Symposium on Logic in Computer Science," pp. 30–36, Computer Society Press of the IEEE, Washington, DC.

MITCHELL, J., AND HARPER, R. (1988), The essence of ML, *in* "Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages," pp. 28–46, Assoc. Comput. Mach., New York.

MYCROFT, A. (1984), Polymorphic type schemes and recursive definitions, *in* "International Symposium on Programming" (M. Paul and B. Robinet, Eds.), pp. 217–228, Lecture Notes in Computer Science, Vol. 167, Springer-Verlag, Berlin/New York.

O'DONNELL, M. (1979), A programming language theorem which is independent of Peano Arithmetic, *in* "Eleventh Annual ACM Symposium on Theory of Computing," Assoc. Comput. Mach., New York.

PEYTON-JONES, S. L. (1987), "The Implementation of Functional Programming Languages," Prentice–Hall, Englewood Cliffs, NJ.

REYNOLDS, J. (1974), Towards a theory of type structures, *in* "Conference on Programming" (J. Loecks, Ed.), pp. 408–425, Lecture Notes in Computer Science, Vol. 19, Springer-Verlag, Berlin.

REYNOLDS, J. (1984), Polymorphism is not set-theoretic, *in* "Semantics of Data Types" (G. Kahn, D. B. MacQueen, and G. D. Plotkin, Eds.), pp. 145–156, Lecture Notes in Computer Science, Vol. 173, Springer-Verlag, Berlin.

REYNOLDS, J., AND PLOTKIN, G. (1990), "On Functors Expressible in the Polymorphic Typed Lambda Calculus," Report CMU-CS-90-147, Carnegie–Mellon University. [Also to appear elsewhere]

ROSE, H. E. (1984), "Subrecursion," Clarendon (Oxford Univ. Press), Oxford.

RUSSELL, B. (1908), Mathematical logic as based on the theory of types, *Amer. J. Math.* **30**, 222–262. [Reprinted in "From Frege to Gödel" (H. van Heijenoort, Ed.), pp. 150–182, Harvard Univ. Press, Cambridge, MA.]

SCHWICHTENBERG, H. (1969), Rekursionszahlen und die Grzegorczyk-Hierarchie, *Arch. Math. Logik* **12**, 85.

STATMAN, R. (1979), The typed λ-calculus is not elementary recursive, *Theoret. Comput. Sci.* **9**, 73.

STATMAN, R. (1981), Number theoretic functions computable by polymorphic programs, *in* "Twenty Second Annual Symposium on Foundations of Computer Science" pp. 279–282, IEEE Computer Society, Los Angeles.

WANG, H. (1954), The formalization of mathematics, *J. Symbolic Logic* **19**, 241.

WANG, H. (1962), Some formal details on predicative set theories, *in* "A Survey of Mathematical Logic," Chap. 24, Science Press, Peking. Republished in 1964 by North Holland, Amsterdam. Republished in 1970 under the title "Logic, Computers, and Sets" by Chelsea, New York.

WHITEHEAD, A., AND RUSSELL, B. (1910), *Principia Mathematica*, Vol. 1, Cambridge Univ. Press, Cambridge, UK.