THE BULLETIN OF SYMBOLIC LOGIC
Volume 11, Number 3, Sept. 2005

TIERING AS A RECURSION TECHNIQUE

HAROLD SIMMONS

Abstract. I survey the syntactic technique of tiering which can be used to restrict the power of a recursion scheme. I show how various results can be obtained entirely proof theoretically without the use of a model of computation.

§1. Introduction. Recently (since 1990) various recursion schemes have been refined to restrict their power. Syntactic conditions, which have no semantic content, are imposed on the scheme. One such refinement is tiered recursion. We use several kinds of natural numbers, of different colours say. In any function composition $g \circ f$ the colour of the output from f must match the colour of the input to g. When the colours are removed the original scheme reappears, and more composites are possible. With the colour coding fewer functions can be produced. (I will use this colour coding analogy several times.) Here are versions of the two best known tiering results.

THEOREM 1.1 (Bellantoni and Cook). *A tiered version $\lambda\mathsf{B}$ of Primitive Recursive Arithmetic produces only the class $\mathcal{H}$ of humble functions.*

(Leivant) *A tiered version $\lambda\mathsf{L}$ of Gödel's T produces only the class $\mathcal{L}$ of latent functions.*

Theorem 1.1 is not exactly what Bellantoni and Cook or Leivant proved. There are some technical differences which I will discuss later when I say what the two classes $\mathcal{H}$ and $\mathcal{L}$ are. Other people have produced variants of these results.

Many tiering proofs use a model of computation. Here I use only the crudest proof theoretic method, the exhaustive search technique. The basic idea is to translate descriptions in some system into a system $\lambda\mathsf{A}$ with little strength. We then apply the following.

THEOREM 1.2.

(0) *The class of first order functions represented in $\lambda\mathsf{A}$ is $\mathcal{T}$.*
(1) *The class of first order functions uniformly simulated throughout $\lambda\mathsf{A}$ is $\mathcal{H}$.*
(2) *The class of first order functions simulated throughout $\lambda\mathsf{A}$ is $\mathcal{L}$.*

Received March 13, 2001; revised June 26, 2002.

© 2005, Association for Symbolic Logic
1079-8986/05/1103-0001/$4.00

Here $\mathcal{T}$ is the class of translations. The notions of representation, uniform simulation, and simulation will be explained later.

In the strict sense this paper contains no new results. It is a survey, and not even a *comprehensive* survey of tiering. I will comment on the omitted topics in the section 9. If there is any novelty in the presentation, it is the use of some structure which seems to support the results, rather than relying on a collection of tricks.

Many functions have a computational complexity; the cost of evaluation. This refers to a model of computation to estimate the amount of time, space, or other resource needed to evaluate the function. We are not concerned with these measures here.

Many functions have an intrinsic complexity independent of any model of computation. The least coarse such measure is the very syntax required to specify the function. If we describe a function in Gödel's T then the syntax used tells us something about its complexity. Tiering helps to organize this finest syntactic measure. In practice, we use a coarser measure, an ordinal, which can be obtained from the syntax. The functions specified in Gödel's T can be matched with the long Grzegorczyk hierarchy (up to ε_0). The multi-coloured version of Gödel's T reduces this ordinal strength from ε_0 to 3. The ordinal strength ω (the primitive recursive functions) reduces to something less than 2.

Section 2 describes the general syntactic machinery. Section 3 gives some examples of tiered systems.

Section 4 begins the analysis of tiering. A geometric structure is attached to a tiered system which seems to determine much of its strength. Section 5 indicates how this attached structure can be used to produce several families of functions.

Sections 6 and 7 consider the classes $\mathcal{H}$ and $\mathcal{L}$, various kinds of recursions, and the way the syntax can be used to limit recursions.

Section 8 describes a translation technique for comparing two systems. The translation respects the internal workings of the systems but does *not* respect the external semantics.

The first version of this paper was read by two referees who made some constructive suggestions. I thank them for that, and I have incorporated almost all their suggestions.

One said my use of 'intrinsic' above was not quite right. We tend not to distinguish between a function and its 'specification', the way it is described. Strictly speaking it is this specification that carries the complexity, not the function, and different specifications can have different complexities; but I do not want to develop that theme here.

§2. Numeric applied λ-calculi.

We review the syntax and workings of a typical tiered system $\lambda\mathsf{T}$, an applied λ-calculus.

Judgement	Context	Space	Derivation
$\Gamma \vdash t : \tau$	$\Gamma = x_l : \sigma_l, \ldots, x_1 : \sigma_1$	$S_l \times \cdots \times S_1 \longrightarrow T$	$(\nabla) \quad \Gamma \vdash t : \tau$

TABLE 1. Preamble

We are mostly concerned with first order numeric functions of a type $\mathbb{N}^l \to \mathbb{N}$ for $l < \omega$. It is often more convenient to view these functions in Curried form, that is, receiving the inputs in sequence rather than as a tuple. We generate the spaces $\mathbb{N}(l)$, as on the left.

$$\mathbb{N}(0) = \mathbb{N}, \quad \mathbb{N}(1+l) = \mathbb{N} \to \mathbb{N}(l), \quad \mathbb{N} \to \cdots \to \mathbb{N} \to \mathbb{N}.$$

Thus $\mathbb{N}(l)$ has the form on the right where there are $l+1$ occurrences of $\mathbb{N}$ (l input sets and one eventual target set.) A function $f : \mathbb{N}(l)$ can receive inputs

$$f x_l, \quad f x_l x_{l-1}, \quad \ldots, \quad f x_l \ldots x_2, \quad f x_l \ldots x_2 x_1$$

in sequence to produce several auxiliary functions and an eventual value $f x_l \ldots x_1$.

As well as first order functions we use higher order functions, so the syntax must be able to handle these. First we generate names for the function spaces.

DEFINITION 2.1. The types are built up from a given stock of atoms using arrow (function space) and cross (product space) formation. Each atom is a type, if π and σ are types then so is $(\pi \to \sigma)$, if σ and ρ are types then so is $(\sigma \times \rho)$.

The use of products is not just a convenience; it makes applied λ-calculi more expressive, but does involve more work. More importantly, some results are quite sensitive to the existence or not of products. We will return to this point several times.

The brackets are part of the syntax (to ensure unique parsing) but in practice we often omit some using the Currying conventions. A small piece of notation is useful. For a type τ we let τ' abbreviate $(\tau \to \tau)$ and then τ'' abbreviates $\tau' \to \tau'$ and so on.

Throughout this paper we use a prime '$\cdot'$' in two ways; one to indicate the successor $x' = x + 1$ of a natural number x; and one to indicate the successor of a type τ, as above.

Each system $\lambda\mathsf{T}$ has at least one atom $\mathcal{N}$, the standard name for the set $\mathbb{N}$ of natural numbers. For an untiered numeric system this is the only atom. Here we allow other names for $\mathbb{N}$ of different colours. We think of $\mathcal{N}$ as naming the monochrome natural numbers, but also allow names for the red, blue, green, ... natural numbers. There is always the monochrome atom $\mathcal{N}$ and, depending on $\lambda\mathsf{T}$, an indexed family $(\mathcal{N}[i] \mid i \in \mathbb{I})$ of coloured atoms

Rule	Shape	Proviso
Axiom	$\Gamma \vdash \mathsf{k} : \kappa$	Γ legal and $\mathsf{k} : \kappa$ an axiom
Projection	$\Gamma \vdash x : \sigma$	Γ legal and $x : \sigma$ occurs in Γ
Weakening	$\dfrac{\Gamma \vdash t : \tau}{\Gamma, x : \sigma \vdash t : \tau}$	$\Gamma, x : \sigma$ legal
Introduction	$\dfrac{\Gamma, x : \sigma \vdash r : \rho}{\Gamma \vdash t : \tau}$	$t = \lambda x : \sigma . r, \quad \tau = \sigma \to \rho$
Elimination	$\dfrac{\Gamma \vdash q : \pi \to \tau \quad \Gamma \vdash p : \pi}{\Gamma \vdash qp : \tau}$	

TABLE 2. The construction rules for derivations

where the index set $\mathbb{I}$ may be arbitrary. For the monochrome systems $\mathbb{I}$ is empty. The set $\mathbb{I}$ indexes the other colours used.

Each type τ names a set $[\![\tau]\!]$, its orthodox interpretation. On the atoms we have

$$[\![\mathcal{N}]\!] = \mathbb{N}, \quad [\![\mathcal{N}[i]]\!] = \mathbb{N}$$

and $[\![\cdot]\!]$ moves through the types in the obvious way. This process loses the colour coding.

By following the construction of a type we produce a descent path.

DEFINITION 2.2. For types τ, γ we write $\tau \downarrow \gamma$ and say τ descends to γ if either $\tau = \gamma$, or $\tau = \pi \to \sigma$ where $\sigma \downarrow \gamma$, or $\tau = \sigma \times \rho$ where $\sigma \downarrow \gamma$ or $\rho \downarrow \gamma$ for types π, σ, ρ. This is a specification by recursion over τ. For each τ there is at least one atom ξ with $\tau \downarrow \xi$.

Types name function spaces. To name functions we use terms.

DEFINITION 2.3. The raw terms are generated from a stock of constants and a stock of identifiers. Each constant and each identifier is a term, if q and p are terms then (qp) is a term, if x is an identifier and σ is a type and r is a term then $(\lambda x : \sigma . r)$ is a term.

The brackets are part of the syntax but we may omit some using the Currying conventions (which, of course, are not the same as the conventions for types). For terms t, s we generate the formal iterates $t^m s$ by $t^0 s = s$ and $t^{1+m} s = t(t^m s)$.

Some of these raw terms are grammatically incorrect. A derivation system is used to extract the meaningful terms. To explain this we need a bit of a preamble.

As in Table 1, a judgement consists of a context Γ, a term t, and a type τ. The context is a list of declarations, $x : \sigma$, where each x is an identifier and

For each pair σ, ρ of types there are three pairing gadgets.	$\mathsf{Pair}_{\sigma,\rho}\colon \sigma \to \rho \to \sigma \times \rho$ $\mathsf{Left}_{\sigma,\rho}\colon \sigma \times \rho \to \sigma$ $\mathsf{Right}_{\sigma,\rho}\colon \sigma \times \rho \to \rho$
Each atom ξ comes with its zero and its successor.	$0_\xi\colon \xi \quad \mathsf{S}_\xi\colon \xi'$
There are iterators for certain atoms ξ and types τ.	$\mathsf{I}_{\tau,\xi}\colon \xi \to \tau''$

TABLE 3. Axioms and constants

each σ is a type. The context is legal if the declared identifiers are pairwise distinct. By setting $S_i = [\![\sigma_i]\!]$ for each $1 \le i \le l$ and $T = [\![\tau]\!]$ we obtain a function space, as shown. The judgement is attempting to name a function in this space where the term t describes its behaviour. However, to do that the judgement must be built according to certain rules.

DEFINITION 2.4. A derivation ∇ is a finite rooted tree of judgements grown according to the rules of Table 2.

Thus, each derivation is either a leaf, which is an Axiom (using an axiom) or a Projection, or is obtained from smaller derivations by one of the rules Weakening, Introduction, or Elimination. As in Table 1 we often write

$$(\nabla) \quad \Gamma \vdash t\colon \tau$$

to indicate the ∇ is a derivation with root judgement $\Gamma \vdash t\colon \tau$.

For a tiered system the constants and axioms are of a limited form, as in Table 3. A tiered system $\lambda\mathsf{T}$ is determined by the index set $\mathbb{I}$ and the battery of iterators. Given $\mathbb{I}$ there are two extremes; with no iterators, or with an iterator for all (ξ, τ).

Each constant $\mathsf{k}\colon \kappa$ is interpreted $[\![\mathsf{k}]\!]$ as a member of the set $[\![\kappa]\!]$. Pairing gadgets have the obvious interpretation. For each atom ξ we let $[\![0_\xi]\!]$ be zero and $[\![\mathsf{S}_\xi]\!]$ be the successor function on $[\![\xi]\!] = \mathbb{N}$. For an iterator $\mathsf{I}\colon \xi \to \tau''$ we have $[\![\mathsf{I}]\!]\colon \mathbb{N} \longrightarrow \mathbb{T}''$ where $\mathbb{T} = [\![\tau]\!]$ and we let $[\![\mathsf{I}]\!]mts = t^m s$ for all $m \in \mathbb{N}, t \in \mathbb{T}', s \in \mathbb{T}$. Thus $[\![\mathsf{I}]\!]$ is a real world iterator.

By passing through a derivation ∇ we produce its meaning

$$[\![\Gamma]\!] \xrightarrow{[\![\nabla]\!]} [\![\tau]\!]$$

a function in the space associated with the root. Notice that the passage from ∇ to $[\![\nabla]\!]$ loses a lot of information. In particular, it loses all the colour coding.

By a technical trick, when the context is empty the meaning of a derived judgement $\vdash t\colon \tau$ produces a member of $[\![\tau]\!]$. In particular, for atoms $\xi(0), \ldots, \xi(l)$, each derivation

$$(F) \quad \vdash \ulcorner f \urcorner\colon \xi(l) \to \cdots \to \xi(1) \to \xi(0)$$

For each redex $(\lambda x\colon \sigma.r)s$ there is a redex removal, where $r[x := s]$ is the result of substituting s for x in r.	$(\lambda x\colon \sigma.r)s \rhd r[x := s]$
For each pair of types σ, ρ there are two kinds of instances. The subscripts have been omitted here.	$\mathsf{Left}(\mathsf{Pair}\ lr) \rhd l$ $\mathsf{Right}(\mathsf{Pair}\ lr) \rhd r$
For each iterator $\mathsf{I}\colon \xi \to \tau''$ over an atom ξ, there are two kinds of instances using its furnishings.	$\mathsf{I}0_\xi ts \rhd s$ $\mathsf{I}(\mathsf{S}_\xi r)ts \rhd t(\mathsf{I}rts)$

TABLE 4. One-step reductions

---

names a function $f = [\![\ulcorner f \urcorner]\!] \in \mathbb{N}(l)$. By definition, the strength of a tiered system $\lambda\mathsf{T}$ is the class of all first order functions so named in the system.

As well as naming the function the term $\ulcorner f \urcorner$ can be used to evaluate f.

Each atom ξ comes with a pair of constants $0_\xi\colon \xi$ and $\mathsf{S}_\xi\colon \xi'$. For each $m \in \mathbb{N}$ we set $\ulcorner m_\xi \urcorner = \mathsf{S}_\xi^m 0_\xi$ to obtain the authentic ξ-numerals. (Why 'authentic' will be explained later.) Note that $\vdash \ulcorner m_\xi \urcorner\colon \xi$ and $[\![\ulcorner m_\xi \urcorner]\!] = m$. These are the numerals of different colours.

For $m(1), \ldots, m(l) \in \mathbb{N}$ let $m(0) = fm(l) \ldots m(1)$. Using (F) we generate a term

$$\vdash \ulcorner f \urcorner \ulcorner m(l)_{\xi(l)} \urcorner \ldots \ulcorner m(1)_{\xi(1)} \urcorner\colon \xi(0)$$

which names $m(0)$. However, this term is not just $\ulcorner m(0)_{\xi(0)} \urcorner$, it can be rather complicated. We employ the reduction mechanism to transform it into the numeral $\ulcorner m(0)_{\xi(0)} \urcorner$.

We use three relations

$$t^- \rhd t^+, \quad t^- \rhd\!\!\rhd t^+, \quad t^- \rhd\!\!\rhd\!\!\rhd t^+$$

between terms t^-, t^+. We set up the 1-step reduction $\rhd$ by explicitly listing the pairs of terms for which it holds. In our case these are as in Table 4.

We generate the reduction relation $\rhd\!\!\rhd$ as the the transitive, compositional closure of the 1-step relation. We are allowed to move from term t^- to term t^+ in a finite number of steps. At each step use an instance $s^- \rhd s^+$ of the 1-step reduction to replace a subterm s^- of the current term by s^+. A simple example of this is $\mathsf{I}_\xi \ulcorner m_\xi \urcorner ts \rhd\!\!\rhd t^m s$ for each atom ξ, $m \in \mathbb{N}$, and terms t, s. You should convince yourself that reduction preserves meaning, that is if $t^- \rhd\!\!\rhd t^+$ then $[\![t^-]\!] = [\![t^+]\!]$. By intention the relation $\rhd\!\!\rhd$ is unidirectional and irreflexive. However, it is sometimes convenient to work with the reflexive version $\rhd\!\!\rhd\!\!\rhd$ of $\rhd\!\!\rhd$. Thus we let $t^- \rhd\!\!\rhd\!\!\rhd t^+$ mean ($t^- = t^+$ or $t^- \rhd\!\!\rhd t^+$).

A term t is normal if there are no reductions from t; if t has no subterm which matches the left hand side of one of the 1-step reductions. For example, each numeral is normal.

There are three general properties of tiered systems we need.

(Normalization): For each derivation, $\Gamma \vdash t^- : \tau$, there is a normal term t^+ with $t^- \rhd\!\rhd\!\rhd t^+$.

(Confluence): For each divergent wedge of reductions $t^- \rhd\!\rhd\!\rhd l$, $t^- \rhd\!\rhd\!\rhd r$ from a common source t^-, there is a convergent wedge $l \rhd\!\rhd\!\rhd t^+$, $r \rhd\!\rhd\!\rhd t^+$ to a common sink t^+.

(Influence): For each derivation $\Gamma \vdash t^- : \tau$ and reduction $t^- \rhd\!\rhd\!\rhd t^+$, there is a derivation $\Gamma \vdash t^+ : \tau$. This is also known as subject reduction.

We do not verify these properties here. All of the examples of Section 3 have these properties, but it is possible other more exotic examples do not.

THEOREM 2.5. *Let ξ be an atom. For each $m \in \mathbb{N}$, the term $\ulcorner m_\xi \urcorner$ is normal. Conversely, for each derivation $\vdash t : \xi$ where t is normal, there is some $m \in \mathbb{N}$ with $t = \ulcorner m_\xi \urcorner$.*

We do not prove this here. The first part is trivial. The second part uses the exhaustive search technique. We enumerate all possible derivations that can have the required root. This involves looking at more general derivations, and the trick is to select the appropriate induction hypothesis. In section 6 we will look at the details of a more interesting example.

How can we evaluate the function f named by a term $\ulcorner f \urcorner$, as in (F) above? Using the general properties the proof of the following takes no more than half a dozen lines.

THEOREM 2.6. *When $\ulcorner f \urcorner$ names $f : \mathbb{N}(l)$, the reduction holds for all $m_1, \ldots, m_l \in \mathbb{N}$.*

$$\vdash \ulcorner f \urcorner : \xi(l) \to \cdots \to \xi(1) \to \xi(0), \quad \ulcorner f \urcorner \ulcorner m_l \urcorner \cdots \ulcorner m_1 \urcorner \rhd\!\rhd\!\rhd \ulcorner f\, m_l \cdots m_1 \urcorner.$$

§3. Some examples of tiered systems. A tiered system $\lambda\mathsf{T}$ is an applied λ-calculus. It has an atom $\mathcal{N}$ with furnishings $\mathsf{0} : \mathcal{N}$ and $\mathsf{S} : \mathcal{N}'$. The rest of the system is designed to give various forms of recursion. The set $\mathbb{I}$ indexes the coloured atoms $\mathcal{N}[i]$, and so fixes the language of $\lambda\mathsf{T}$. All atoms are furnished and there are three pairing gadgets for all product types. The battery of iterators $\mathsf{I}_{\tau,\xi} : \xi \to \tau''$ determines which recursions are available, and so fixes the strength of $\lambda\mathsf{T}$. Only selected pairs ξ, τ support an iterator, and ξ is always an atom.

$\lambda\mathsf{A}$ The simplest tiered system has just one atom $\mathcal{N}$ and no iterators. This is essentially the simply typed λ-calculus, and is a part of every other tiered system.

$\lambda\mathsf{B}$ We enriched $\lambda\mathsf{A}$ by adding one new atom $\mathcal{N}[\star]$ and an iterator $\mathsf{I} : \mathcal{N}[\star] \to \mathcal{N}''$. This can be viewed as the system underlying the Bellantoni and Cook result.

$\lambda\mathsf{C}$ We enriched $\lambda\mathsf{A}$ by adding an atom and an iterator $\mathsf{I}_\tau : \mathcal{N}[\tau] \to \tau''$ for each type τ generated from $\mathcal{N}$. This includes $\lambda\mathsf{B}$ since $\mathcal{N}[\star] = \mathcal{N}[\mathcal{N}]$ and $\mathsf{I} = \mathsf{I}_\mathcal{N}$.

λD This is a more economical version of λC. We add to λA one new atom $\mathcal{N}[\star]$ and a whole battery of iterators $\mathsf{I}_\tau : \mathcal{N}[\star] \to \tau''$ for each type τ generated from $\mathcal{N}$.

λE We may iterate the trick of λD. We add a sequence $(\mathcal{N}[i] \mid i < \omega)$ of atoms together with iterators $\mathsf{I}_{\tau,i} : \mathcal{N}[i] \to \tau''$ for each $i < \omega$ and each type τ built up from $\mathcal{N}, \mathcal{N}[0], \mathcal{N}[1], \ldots, \mathcal{N}[i-1]$. Thus λE is layered with λD appearing as layer $i = 0$.

λF This is a coalesced version of λB. We add to λA one iterator $\mathsf{I} : \mathcal{N} \to \mathcal{N}''$ (without the coloured atom $\mathcal{N}[\cdot]$). This is an enfeebled version of Primitive Recursive Arithmetic.

λG This is a coalesced version of λC and of λD. There are no coloured atoms. We add to λA an iterator $\mathsf{I}_\tau : \mathcal{N} \to \tau''$ for each type τ. This is the term calculus of Gödel's T.

λL This has an intricate type structure. Begin to generate the types from the atom $\mathcal{N}$. As each type τ is generated throw in a new atom $\mathcal{N}[\tau]$ from which further types can be generated. For each type τ (which may contain nested uses of atoms) there is an iterator $\mathsf{I}_\tau : \mathcal{N}[\tau] \to \tau''$ on the new atom. This is the system used by Leivant.

The systems λA, λF, and λG are well understood, but it may not be clear how λC and λD differ. The multi-purpose atom $\mathcal{N}[\star]$ makes λD more difficult to analyse. Which of λE and λL, do you think is the stronger? We take λA as a source system against which other tiered systems can be tested. The exhaustive search technique gives the following.

THEOREM 3.1. *The strength of* λA *is the class* $\mathcal{T}$ *of translations*

$$f x_l \ldots x_1 = x_i + m, \quad f x_l \ldots x_1 = m,$$

those functions $f \in \mathbb{N}(l)$ *of one of the indicated forms for some* $m \in \mathbb{N}$ *and* $1 \leq i \leq l$.

§4. The network of numerals. Each atom ξ is furnished with constants 0_ξ and S_ξ from which we generate the authentic ξ-numerals $\ulcorner m_\xi \urcorner = \mathsf{S}_\xi^m 0_\xi$, the canonical names of natural numbers. For each type τ we set

$$\overline{m_\tau} = \lambda y : \tau', x : \tau . y^m x$$

to obtain the simulated τ-numerals (Church numerals) with $\vdash \overline{m_\tau} : \tau''$. In the pure λ-calculus these are used to capture the natural numbers. However, $\overline{m_\tau}$ *does not name a natural number*, it names the m-fold iterator on $[\![\tau]\!]$. This difference leads to a geometric object which seems to be an important indicator of the strength of a tiered system.

For an authentic numeral $\ulcorner m_\xi \urcorner : \xi$ and an iterator $\mathsf{I} : \xi \to \tau''$, the compound $\mathsf{I} \ulcorner m_\xi \urcorner$ names the m-fold iterator on $[\![\tau]\!]$. But $\overline{m_\tau}$ does this already, so why bother with $\ulcorner m_\xi \urcorner$?

$\tau = \pi \to \sigma$	$\tau = \sigma \times \rho$	$\tau = \sigma \times \rho$	
Lifted to	$\mathsf{D}_{\sigma,\tau} \colon \tau \to \sigma$	$\mathsf{D}_{\sigma,\tau} \colon \tau \to \sigma$	$\mathsf{D}_{\rho,\tau} \colon \tau \to \rho$
lifted	$\lambda v \colon \tau \,.\, v\mathsf{0}_\pi$	$\mathsf{Left}_{\sigma,\rho}$	$\mathsf{Right}_{\sigma,\rho}$
Simulated to	$\mathsf{E}_{\sigma,\tau} \colon \tau'' \to \sigma''$	$\mathsf{E}_{\sigma,\tau} \colon \tau'' \to \sigma''$	$\mathsf{E}_{\rho,\tau} \colon \tau'' \to \rho''$
simulated	$w(\mathsf{B}y)(\mathsf{K}x)\mathsf{0}_\pi$	$\mathsf{Left}_{\sigma,\rho}\,\big(w(\mathsf{A}_L y)(\mathsf{J}_L x)\big)$	$\mathsf{Right}_{\sigma,\rho}\,\big(w(\mathsf{A}_R y)(\mathsf{J}_R x)\big)$

TABLE 5. Some numeral converters

DEFINITION 4.1. For each type τ the lifted furnishings of τ, that is the terms $\mathsf{0}_\tau \colon \tau$ and $\mathsf{S}_\tau \colon \tau''$, are generated by recursion over the structure of τ as follows.

τ	$\mathsf{0}_\tau$	S_τ
atom ξ	$\mathsf{0}_\xi$	S_ξ
$\pi \to \sigma$	$\lambda p \colon \pi \,.\, \mathsf{0}_\sigma$	$\lambda v \colon \tau,\, p \colon \pi \,.\, \mathsf{S}_\sigma(vp)$
$\sigma \times \rho$	$\mathsf{Pair}_{\sigma,\rho}\,\mathsf{0}_\sigma\mathsf{0}_\rho$	$\lambda v \colon \tau \,.\, \mathsf{Pair}_{\sigma,\rho}(\mathsf{S}_\sigma(\mathsf{Left}\,v))(\mathsf{S}_\rho(\mathsf{Right}\,v))$.

For each type τ and $m \in \mathbb{N}$ we set $\ulcorner m_\tau \urcorner = \mathsf{S}_\tau^m \mathsf{0}_\tau$ to obtain the lifted τ-numerals.

For an atom τ these furnishings and numerals are just the authentic gadgets. For a non-atom τ the term $\ulcorner m_\tau \urcorner$ is not normal and we do not have $[\![\ulcorner m_\tau \urcorner]\!] = m$. These terms are designed to link the authentic and the simulated, not to capture natural numbers.

For a type τ let $\widehat{\tau}$ be τ'' or τ, and for each $m \in \mathbb{N}$ set $\widehat{m_\tau} = \ulcorner m_\tau \urcorner$ or $\widehat{m_\tau} = \overline{m_\tau}$ to match the choice. Similarly for a type γ let $\widehat{\gamma}$ be γ'' or γ where this choice is independent of the first. For $m \in \mathbb{N}$ set $\widehat{m_\gamma} = \ulcorner m_\gamma \urcorner$ or $\widehat{m_\gamma} = \overline{m_\gamma}$ to match the second choice.

DEFINITION 4.2. Continuing with the notation above, a numeral converter is a term $\vdash \mathsf{L} \colon \widehat{\tau} \to \widehat{\gamma}$ such that $\mathsf{L}\widehat{m_\tau} \rhd\!\rhd \widehat{m_\gamma}$ for all $m \in \mathbb{N}$.

There are four kinds of numeral converters; simulated to lifted, lifted to lifted, simulated to simulated, and lifted to simulated. For a type τ the term $\mathsf{C}_{\tau,\tau} \colon \tau'' \to \tau$ where

$$\mathsf{C}_{\tau,\tau} = \lambda w \colon \tau''.w\mathsf{S}_\tau\mathsf{0}_\tau$$

is an example of the first kind. Let us look at some other examples. These will form part of the proof of Theorem 4.3.

Consider Table 5. In the upper block there are three terms $\mathsf{D}_{\gamma,\tau} \colon \tau \to \gamma$ where γ is a component of τ and the term is given below its type display. A simple induction gives

$$\ulcorner m_\tau \urcorner \mathsf{0}_\pi \rhd\!\rhd \ulcorner m_\sigma \urcorner, \quad \mathsf{Left}_{\sigma,\rho}\ulcorner m_\tau \urcorner \rhd\!\rhd \ulcorner m_\sigma \urcorner, \quad \mathsf{Right}_{\sigma,\rho}\ulcorner m_\tau \urcorner \rhd\!\rhd \ulcorner m_\rho \urcorner,$$

respectively, and hence $\mathsf{D}_{\gamma,\tau}$ is a numeral converter of the indicated type and kind.

The lower block leads to three terms $\mathsf{E}_{\gamma,\tau} : \tau'' \to \gamma''$ for the same component γ of τ. Let BODY be the term given in the table using auxiliaries $\mathsf{B}, \ldots, \mathsf{J}_R$ constructed to satisfy

$$\begin{array}{lll} \mathsf{K}sp \rhd\!\!\rhd s & \mathsf{Left}(\mathsf{J}_L x) \rhd\!\!\rhd x & \mathsf{Right}(\mathsf{J}_R x) \rhd\!\!\rhd x \\ \mathsf{B}tvp \rhd\!\!\rhd t(vp) & \mathsf{Left}(\mathsf{A}_L yv) \rhd\!\!\rhd y(\mathsf{Left}\, v) & \mathsf{Right}(\mathsf{A}_R yv) \rhd\!\!\rhd y(\mathsf{Right}\, v) \end{array}$$

for terms v and p. In each case $\mathsf{E}_{\gamma,\tau}$ is $\lambda w : \tau'', y : \gamma', x : \gamma.$BODY and we have

$$\mathsf{E}_{\sigma,\tau}\overline{m_\tau} ts \rhd\!\!\rhd (\mathsf{B}t)^m(\mathsf{K}s)\mathsf{0}_\pi \rhd\!\!\rhd t^m s, \quad \mathsf{E}_{\sigma,\tau}\overline{m_\tau} ts \rhd\!\!\rhd \mathsf{Left}\big((\mathsf{A}_L t)^m(\mathsf{J}_L s)\big) \rhd\!\!\rhd t^m s$$

for arbitrary terms $t : \gamma'$ and $s : \gamma$ (with a similar reduction for the right-hand version $\mathsf{E}_{\rho,\tau}$). Thus $\mathsf{E}_{\gamma,\tau}$ is a numeral converter of the indicated type and kind.

THEOREM 4.3. *For each descent path $\tau \downarrow \gamma$ there are numeral converters*

$$\mathsf{C}_{\gamma,\tau} : \tau'' \to \gamma, \quad \mathsf{D}_{\gamma,\tau} : \tau \to \gamma, \quad \mathsf{E}_{\gamma,\tau} : \tau'' \to \gamma''$$

taking simulated to lifted, lifted to lifted, simulated to simulated, respectively.

PROOF. The term $\mathsf{C}_{\tau,\tau}$ above, and the identity terms $\mathsf{D}_{\tau,\tau} = \mathsf{Id}_\tau$ and $\mathsf{E}_{\tau,\tau} = \mathsf{Id}_{\tau''}$ are numeral converters. For the one step cases the required terms $\mathsf{D}_{\gamma,\tau}$ and $\mathsf{E}_{\gamma,\tau}$ are given above. The composite of two compatible numeral converters is a numeral converter. ⊣

What about converting lifted numerals to simulated numerals? Roughly speaking iterators and authentic to simulated numeral converters are the same thing. Suppose we have either an iterator I or a numeral converter L from authentic to simulated as indicated.

$$\begin{array}{llll} \text{Iterator} & \mathsf{I} : \xi \to \tau'' & \text{gives converter} & \mathsf{L} = \lambda u : \xi.\lambda y : \tau', x : \tau.\mathsf{I}uyx \\ \text{Converter} & \mathsf{L} : \xi \to \tau'' & \text{gives iterator} & \mathsf{I} = \lambda u : \xi.\lambda y : \tau', x : \tau.\mathsf{L}uyx \end{array}$$

In each case by taking the corresponding compound term we obtain, respectively, a numeral converter or (a term that behaves point-wise as) an iterator.

LEMMA 4.4. *Suppose $\lambda\mathsf{T}$ has a numeral converter $\mathsf{L} : \xi \to \xi''$ (to simulated numerals) for some atom ξ. Then $\lambda\mathsf{T}$ is as least as strong as $\lambda\mathsf{F}$.*

The network of numerals and converters may have other interesting properties.

§5. Imitating functions in a tiered system. Recall that a function named by a term can be evaluated by the reduction mechanism.

DEFINITION 5.1. Let $\tau_0, \ldots, \tau_l$ be types and select $\widehat{\tau_i} = \tau_i''$ or $\widehat{\tau_i} = \tau_i$ for each $0 \le i \le l$. These choices are independent. A term $\widehat{f}$, as indicated below, imitates a function $f : \mathbb{N}(l)$ if the reduction holds for all $m_0, m_1, \ldots, m_l \in \mathbb{N}$ with $f m_l \ldots m_1 = m_0$.

$$\vdash \widehat{f} : \widehat{\tau_l} \to \cdots \to \widehat{\tau_1} \to \widehat{\tau_0}, \quad \widehat{f}\,\widehat{m_l} \ldots \widehat{m_1} \rhd\!\!\rhd \widehat{m_0}.$$

When each τ is an atom and $\widehat{\tau} = \tau$, such an imitation is a representation. When $\widehat{\tau_i} = \tau_i''$ for each i, such an imitation is a simulation.

By Theorem 2.6 naming and representing are two sides of the same coin. Other forms of imitation are not so clear cut. A term may name one function but simulate another.

EXAMPLE 5.2. For a type τ generate the powers $\tau^{(i)}$ by $\tau^{(0)} = \tau$ and $\tau^{(i')} = \tau^{(i)\prime}$. Let

$$\beth_l m_l \dots m_1 m_0 = m_0^{m_1^{\cdot^{\cdot^{m_l}}}}$$

for $m_0, \dots, m_l \in \mathbb{N}$ to give an $(l+1)$-placed stacking function $\beth_l$. The term

$$\mathsf{B}(\tau, l) = \lambda u_l \colon \tau^{(l+2)}, \dots, u_1 \colon \tau^{(3)}, u_0 \colon \tau^{(2)}.\lambda y \colon \tau', x \colon \tau.u_l \dots u_1 u_0 y x$$

names an identity function but simulates $\beth_l$.

By composing with converters we can change the kind of imitation of a function.

LEMMA 5.3. *For each imitation we may arrange that each input is simulated and the output is authentic. Furthermore, we may arrange that all the input types are the same.*

The strength of a tiered system $\lambda\mathsf{T}$ is concerned with first order functions represented in $\lambda\mathsf{T}$. The 'hidden' strength is concerned with the first order functions imitated in $\lambda\mathsf{T}$.

DEFINITION 5.4. The function $f \in \mathbb{N}(l)$ is uniformly simulated throughout the tiered system $\lambda\mathsf{T}$ if for each type ζ there is a term (U) which simulates f.

$$(U) \quad \vdash \overline{f} \colon \zeta'' \to \dots \to \zeta'' \to \zeta'', \qquad (S) \quad \vdash \overline{f} \colon \zeta_l'' \to \dots \to \zeta_1'' \to \zeta''$$

The function f is simulated throughout $\lambda\mathsf{T}$ if for each type ζ there are types $\zeta_1, \dots, \zeta_l$ and a term (S) which simulates f.

Let $\mathcal{R}(\lambda\mathsf{T})$, $\mathcal{U}(\lambda\mathsf{T})$, $\mathcal{S}(\lambda\mathsf{T})$ be, respectively, the class of functions represented, uniformly simulated throughout, and simulated throughout $\lambda\mathsf{T}$.

Using this notation we see that the various parts of Theorems 1.1 and 1.2 are

$$(1.1\,(B)) \quad \mathcal{R}(\lambda\mathsf{B}) = \mathcal{H} \quad (1.1\,(L)) \quad \mathcal{R}(\lambda\mathsf{L}) = \mathcal{L}$$
$$(1.2\,(0)) \quad \mathcal{R}(\lambda\mathsf{A}) = \mathcal{T} \quad (1.2\,(1)) \quad \mathcal{U}(\lambda\mathsf{A}) = \mathcal{H} \quad (1.2\,(2)) \quad \mathcal{S}(\lambda\mathsf{A}) = \mathcal{L}$$

respectively. Of these $(1.2(0))$ is Theorem 3.1. The rest have yet to be proved.

§6. The class $\mathcal{H}$ of humble functions. Theorem 1.2(1) is an old result going back at least to [16]. However, the details are worth looking at, for there are still things to learn. We verify a cycle of inclusions

$$\mathcal{E} \subseteq \mathcal{U}(\lambda\mathsf{A}) \subseteq \mathcal{H} \subseteq \mathcal{M} \subseteq \mathcal{E}$$

between four classes three of which have yet to be defined.

The class $\mathcal{E}$ of extended polynomials is the compositional closure of the projections, the zero function Z, successor S, addition A, multiplication M, and numeric negation N given by $N0 = 1$ and $Nx' = 0$ (for $x \in \mathbb{N}$).

LEMMA 6.1. *We have $\mathcal{E} \subseteq \mathcal{U}(\lambda\mathsf{A})$.*

PROOF. Trivially, the projections can be simulated, and the terms

$$\begin{aligned}
\overline{\mathsf{Z}_\zeta} &= \lambda u\colon \zeta''.\lambda y\colon \zeta', x\colon \zeta.x && \zeta'' \to \zeta'' \\
\overline{\mathsf{S}_\zeta} &= \lambda u\colon \zeta''.\lambda y\colon \zeta', x\colon \zeta.y(uyx) && \zeta'' \to \zeta'' \\
\overline{\mathsf{A}_\zeta} &= \lambda v,u\colon \zeta''.\lambda y\colon \zeta', x\colon \zeta.(vy)(uyx) && \zeta'' \to \zeta'' \to \zeta'' \\
\overline{\mathsf{M}_\zeta} &= \lambda v,u\colon \zeta''.\lambda y\colon \zeta', x\colon \zeta.v(uy)x && \zeta'' \to \zeta'' \to \zeta'' \\
\overline{\mathsf{N}_\zeta} &= \lambda u\colon \zeta''.\lambda y\colon \zeta', x\colon \zeta.u(\lambda v\colon \zeta\,.\,x)(yx) && \zeta'' \to \zeta''
\end{aligned}$$

simulate the initial functions, Z, S, A, M, N. Closure under composition is immediate. ⊣

The inclusion $\mathcal{U}(\lambda\mathsf{A}) \subseteq \mathcal{H}$ is an immediate consequence of the following.

DEFINITION 6.2. A function $f : \mathbb{N}(l)$ is humble if it is imitated in $\lambda\mathsf{A}$ by a derivable term

$$(F) \quad \vdash \widehat{f}\colon \widehat{\mathcal{N}} \to \cdots \to \widehat{\mathcal{N}} \to \mathcal{N}$$

where each $\widehat{\mathcal{N}}$ is $\mathcal{N}''$ or $\mathcal{N}$ with independent choices. We abreviate the type of $\widehat{f}$ as $\widehat{\mathcal{N}}(l)$. Let $\mathcal{H}$ be the class of humble functions.

Using $\mathsf{C}\colon \mathcal{N}'' \to \mathcal{N}$, each derivation (F) gives a derivation $\vdash \widetilde{f}\colon \mathcal{N}''(l)$ which imitates the same function f. But the step $\widehat{f} \mapsto \widetilde{f}$ loses some important information. Using (F) we divide the inputs of f into the simulated and the authentic. It is convenient to write

$$f(y_s, \ldots, y_1; x_a, \ldots, x_1)$$

for the value of f at the simulated inputs $y_1, \ldots, y_s$ and the authentic inputs $x_1, \ldots, x_a$. We say f has arity (s, a) and write $f(\mathsf{y};\mathsf{x})$ when the details of y and x are not needed. Roughly speaking these correspond to what are often called normal and safe inputs. However, as we will see, it is better to think of these as control and standard inputs.

A decision function D is an extended polynomial which takes only the values 0 and 1. Decision functions $D_1, \ldots, D_a, D$ form a partition if each has the same list y of inputs, and for each such y exactly one of $D_1\mathsf{y}, \ldots, D_a\mathsf{y}, D\mathsf{y}$ is 1 and the others are 0.

DEFINITION 6.3. A function f of arity (s, a) is meek if it has the form

$$f(\mathsf{y};\mathsf{x}) = \begin{cases} x_i + A_i\mathsf{y} & \text{if } D_i\mathsf{y} = 1, \\ A\mathsf{y} & \text{if } D\mathsf{y} = 1 \end{cases}$$

where $A_1, \ldots, A_a, A$ are extended polynomials and $D_1, \ldots, D_a, D$ form a partition. Let $\mathcal{M}$ be the class of meek functions.

In other words, a meek function is a step-wise translation where the steps are controlled by the simulated inputs.

We will show $\mathcal{H} \subseteq \mathcal{M}$, and more. From the derivation (F) we obtain a description of f as a meek function where *the status of the inputs—simulated or authentic—is as in the term* $\widehat{f}$. For this we need some closure properties of meek functions. The first corresponds to what is sometimes called safe composition.

Suppose we have a meek function h with indicated arity.

$$h(v_s, \ldots, v_1; u_a, \ldots, u_1) \quad k_s\mathsf{y}, \ldots, k_1\mathsf{y} \quad g_a(\mathsf{y};\mathsf{x}), \ldots, g_1(\mathsf{y};\mathsf{x}).$$

Suppose also we have matching lists of meek functions with the indicated arities.

LEMMA 6.4. *For the above data the composite function f given by*

$$f(\mathsf{y};\mathsf{x}) = h(k_s\mathsf{y}, \ldots, k_1\mathsf{y}; g_a(\mathsf{y};\mathsf{x}), \ldots, g_1(\mathsf{y};\mathsf{x}))$$

is meek with the indicated arity.

Suppose $h(\mathsf{y};\mathsf{x}, x)$ is a meek function with indicated arity. Select one of the authentic inputs x (and place it at the right hand end). By fixing the other inputs we obtain a 1-placed function $h(\mathsf{y};\mathsf{x}, \cdot)$ which can be iterated. Thus we define a new function f by

$$f(r, \mathsf{y};\mathsf{x}, x) = h(\mathsf{y};\mathsf{x}, \cdot)^r x$$

where the exponent r is a simulated input and the argument x is an authentic input.

LEMMA 6.5. *For the above data the function f is meek with indicated arity.*

PROOF. We are given

$$h(\mathsf{y};\mathsf{x}, x) = \begin{cases} x + A\mathsf{y} & \text{if } D\mathsf{y} = 1, \\ x_i + B_i\mathsf{y} & \text{if } E_i\mathsf{y} = 1, \\ C\mathsf{y} & \text{if } F\mathsf{y} = 1 \end{cases}$$

where A, B_i, C are extended polynomials and D, E_i, F form a partition. Here i ranges over the length of x. Making use of the numeric negation N

we have

$$h(\mathsf{y};\mathsf{x},\cdot)^r x = \left\{ \begin{array}{rl} x & \text{if } Nr = 1, \\ x + r \times (A\mathsf{y}) & \text{if } N^2 r = 1 \text{ and } D\mathsf{y} = 1, \\ x_i + B_i\mathsf{y} & \text{if } N^2 r = 1 \text{ and } E_i\mathsf{y} = 1, \\ C\mathsf{y} & \text{if } N^2 r = 1 \text{ and } F\mathsf{y} = 1 \end{array} \right\}$$

$$= \left\{ \begin{array}{rl} x + G(r,\mathsf{y}) & \text{if } H(r,\mathsf{y}) = 1, \\ x_i + B_i\mathsf{y} & \text{if } I_i(r,\mathsf{y}) = 1, \\ C\mathsf{y} & \text{if } J(r,\mathsf{y}) = 1 \end{array} \right.$$

where the components

$$G(r,\mathsf{y}) = r \times (A\mathsf{y}) \times (N^2 r) \qquad I_i(r,\mathsf{y}) = (N^2 r) \times (E_i\mathsf{y})$$
$$H(r,\mathsf{y}) = Nr + (N^2 r) \times (D\mathsf{y}) \qquad J(r,\mathsf{y}) = (N^2 r) \times (F\mathsf{y})$$

are extended polynomials and the H, I_i, J form a partition. ⊣

We will need a mixture of Lemmas 6.4 and 6.5.

LEMMA 6.6. *Suppose the functions $h(\mathsf{y};\mathsf{x},\cdot)$ and $g(\mathsf{y};\mathsf{x})$ are meek with indicated arities. Suppose x is one of the simulated inputs. The function f given by*

$$f(\mathsf{y};\mathsf{x}) = h(\mathsf{y};\mathsf{x},\cdot)^x g(\mathsf{y};\mathsf{x})$$

is meek with indicated arity.

We use the exhaustive search technique to analyse derivations (F), but this time we will actually prove the result. We must analyse a more extensive class of derivations

$$(\nabla) \quad \Gamma \vdash t : \tau$$

in $\lambda\mathsf{A}$ which meet the following restrictions.

(Γ) The context Γ is a list of declarations $z : \widehat{\mathcal{N}}$.
(t) The term t is normal.
(τ) The type τ is either $\widehat{\mathcal{N}}(l)$ (for some l) or is a product type.

The reason for including product types in (τ) will become clear.

THEOREM 6.7. *For each derivation ∇, restricted as above, the following hold.*

(func): *If τ is a function type $\widehat{\mathcal{N}}(l)$ then the function produced by ∇ is meek.*
(prod): *If τ is a product type then t is* Pair *lr for some terms l, r.*

PROOF. We proceed by induction over ∇. We survey all the different ways that ∇ can arise, remembering the restrictions imposed.

(As in most proofs of this kind, there are many cases to consider. Almost all of these are straight forward. In this particular proof, we save the interesting one until last.)

Firstly, ∇ may be a leaf, either an Axiom or a Projection.

(Axiom) Since we are working in λA there are just five possibilities for $t : \tau$.

(i) $\mathsf{0} : \mathcal{N}$ (iii) $\mathsf{Pair} : \sigma \to \rho \to \sigma \times \rho$ (iv) $\mathsf{Left} : \sigma \times \rho \to \sigma$

(ii) $\mathsf{S} : \mathcal{N}'$ (v) $\mathsf{Right} : \sigma \times \rho \to \rho$

Both (i,ii) fall into case (func) and the function produced is a translation. The axioms (iii,iv,v) do not arise (since they match neither (func) nor (prod)).

(Projection) Because of the nature of Γ we see that $t : \tau$ is either $y : \mathcal{N}''$ or $x : \mathcal{N}$ depending on the choice of $\widehat{\mathcal{N}}$. The first of these matches neither (func) nor (prod), so only the second can arise. This produces a projection.

Secondly, ∇ may arise by a use of (Weak), (Intro), of (Elim) from one or two shorter derivations. The first two of these are easily dealt with.

(Weak) Here we have a shorter derivation

$$(\nabla^-) \quad \Sigma \vdash t : \tau$$

where $\Gamma = \Sigma, z : \widehat{\mathcal{N}}$. The induction hypothesis applies to ∇^- to give either an appropriate function or to show that t is a product term. The function produced by ∇ is a composite of the function produced by ∇^- with a projection (to insert a dummy argument).

(Intro) Here we have a shorter derivation

$$(\nabla^-) \quad \Gamma, z : \sigma \vdash r : \rho$$

where $t = \lambda z : \sigma.r$ and $\tau = \sigma \to \rho$. In particular, τ is not a product type, and hence must be some $\widehat{\mathcal{N}}(l)$, so that σ is $\widehat{\mathcal{N}}$ and ρ is a function type. This ensures that ∇^- is of restricted form, and we may invoke the induction hypothesis. This gives the required result since ∇^- and ∇ produce the same function.

Lastly, ∇ may arise by a use of (Elim). This splits into several subcases.

(Elim) Tracking ∇ through all uses of (Elim) and (Weak) we find shorter derivations

$$(Q) \quad \Xi \vdash q : \pi_k \to \cdots \to \pi_1 \to \tau, \qquad (P_i) \quad \Pi_i \vdash p_i : \pi_i, \quad i \le i \le k,$$

where Ξ and each Π_i is a left part of Γ, and $t = q p_k \ldots p_1$ with q and each p_i normal. Let

$$\chi = \pi_k \to \cdots \to \pi_1 \to \tau$$

be the type of (Q), and consider how (Q) can arise. We can discount the use of (Elim) or (Weak). Thus Q is either an Axiom or a Projection or arises by a use of Introduction.

(Q Axiom) There are five possibilities for $q : \chi$, namely (i,ii,iii,iv,v) as listed earlier.

Case (i) can not arise, for χ must be composite.

For case (ii) we must have

$$(Q) \quad \Xi \vdash \mathsf{S} : \mathcal{N}', \qquad (P) \quad \Pi \vdash p : \mathcal{N}$$

with $t = \mathsf{S}p$ and $\tau = \mathcal{N}$. In particular, (P) meets the restrictions, and so we may invoke the induction hypothesis. The derivation P produces an appropriate function, and then ∇ produces a composite of this with the successor function and some projections.

For case (iii) we have

$$(Q) \quad \Xi \vdash \mathsf{Pair} : \sigma \to \rho \to \sigma \times \rho$$

and hence, since no further uses of (Elim) are possible, one of

$$\tau = \rho \to \sigma \times \rho, \quad t = \mathsf{Pair}\, l, \quad \Pi \vdash l : \sigma, \quad \tau = \sigma \times \rho,$$
$$t = \mathsf{Pair}\, lr, \quad \Pi \vdash l : \sigma, \quad \Lambda \vdash r : \rho$$

must hold. The first cannot arise, for this τ has the wrong shape. The second must arise, and hence $t = \mathsf{Pair}\ lr$, which falls under (prod).

For case (iv) we have at least

$$(Q) \quad \Xi \vdash \mathsf{Left} : \sigma \times \rho \to \sigma, \qquad (P) \quad \Pi \vdash p : \sigma \times \rho$$

where $\mathsf{Left}\ p$ is a subterm of t. The induction hypothesis applied to (P) gives $p = \mathsf{Pair}\, lr$ for some term l, r. But now $\mathsf{Left}(\mathsf{Pair}\, lr)$ is a subterm of t. Since this subterm reduces (to l) and t is normal, this case cannot arise.

For case (v) a similar argument shows that it cannot arise.

(Q Introduction) Here we have $q = (\lambda z : \pi.r)$, so that $(\lambda z : \pi.r)p$ is a subterm of t. Since t is normal, this case cannot arise.

(Q Projection) This is the crucial case. Because of the nature of Ξ we have either

$$\Xi \vdash x : \mathcal{N}'' \text{ with } \chi = \mathcal{N}'' \quad \text{or} \quad \Xi \vdash y : \mathcal{N} \text{ with } \chi = \mathcal{N}$$

depending on the declaration projected. But χ must be compound, so only the first of these can arise. Thus we must have one of

$$\Xi \vdash x : \mathcal{N}'' \quad \Pi \vdash p : \mathcal{N}' \qquad \text{with } t = xp$$
$$\Xi \vdash x : \mathcal{N}'' \quad \Pi \vdash p : \mathcal{N}' \quad \Lambda \vdash r : \mathcal{N} \quad \text{with } t = xpr$$

where Ξ, Π, and Λ (when it occurs) are left hand parts of Γ.

When moving from ∇ to Q we removed a number of uses of (Weak) and at least one use of (Elim). We may reinstate these uses of (Weak) to produce

$$(X) \quad \Gamma \vdash x : \mathcal{N}'', \qquad (P) \quad \Gamma \vdash p : \mathcal{N} \to \mathcal{N}, \qquad (R) \quad \Gamma \vdash r : \mathcal{N},$$

where these derivations are still shorter than ∇ (because of the missing use of (Elim)).

We may apply the induction hypothesis to (P) to obtain a meek function h. In a typical value $h(\mathsf{y}; \mathsf{x}, x)$ of h the right hand authentic input corresponds to the input type $\mathcal{N}$ in the type of p. All the other inputs of h arise from the context Γ.

Consider $\Gamma \vdash yp\colon \mathcal{N}'$ where yp is a subterm of t. Think of the function f produced by this derivation. (Note that we are *not* invoking the induction hypothesis to claim that this function is meek.) The identifier y corresponds to one of the simulated inputs of h. Thus, remembering how simulated numerals work, we have

$$f(\mathsf{y};\mathsf{x},x) = h(\mathsf{y};\mathsf{x},\cdot)^y x$$

to show, by Lemma 6.6, that, in fact, f is meek with indicated arity.

When $t = yp$ this is the function produced by ∇.

When $t = ypr$ we apply the induction hypothesis to (R) to obtain a function $g(\mathsf{y};\mathsf{x},x)$ which is meek with indicated arity. Then the composite

$$f(\mathsf{y};\mathsf{x},x) = h(\mathsf{y};\mathsf{x},\cdot)^y g(\mathsf{y};\mathsf{x},x)$$

is the function produced by ∇. By Lemma 6.6 this is meek. ⊣

We will need a particular case of this result.

COROLLARY 6.8. *Each derivation* $\vdash \widehat{f}\colon \widehat{\mathcal{N}}(l)$ *of* $\lambda\mathsf{A}$ *imitates a meek function.*

We have $\mathcal{E} \subseteq \mathcal{U}(\lambda\mathsf{A}) \subseteq \mathcal{H} \subseteq \mathcal{M}$. In fact, extensionally these four classes are the same. To complete the cycle consider any meek function $f \in \mathcal{M}$. By Definition 6.3, we have

$$f(\mathsf{y};\mathsf{x}) = (A\mathsf{y}) \times (D\mathsf{y}) + \sum_i (x_i + A_i\mathsf{y}) \times (D_i\mathsf{y})$$

to give $f \in \mathcal{E}$. It can be shown that each meek function can be imitated as a humble function without changing the status of the inputs. I will not give the details here.

§7. The class $\mathcal{L}$ of latent functions. In this section we start the proof of Theorem 1.2(2). As with part (1) of that result we begin with a rather unsatisfying definition of the class $\mathcal{L}$.

DEFINITION 7.1. A function $f \in \mathbb{N}(l)$ is latent if it is imitated in $\lambda\mathsf{A}$ by a derivable term

$$\vdash \widehat{f}\colon \zeta(l)'' \to \cdots \to \zeta(1)'' \to \mathcal{N}$$

for some types $\zeta(1),\ldots,\zeta(l)$. Let $\mathcal{L}$ be the class of latent functions.

As with Definition 6.2 this does not tell us all we want to know. We will obtain more concrete information about the class $\mathcal{L}$, but not with the same success as we had with $\mathcal{H}$.

We relate $\mathcal{L}$ to three other classes. Remembering Definition 5.4 we have the following.

LEMMA 7.2. *We have* $\mathcal{S}(\lambda\mathsf{A}) \subseteq \mathcal{L} \subseteq \mathcal{R}(\lambda\mathsf{C}) \subseteq \mathcal{R}(\lambda\mathsf{L})$.

PROOF. The first inclusion follows by a use of the numeral converter $\mathsf{C}\colon \mathcal{N}'' \to \mathcal{N}$ available in $\lambda\mathsf{A}$. The second inclusion follows by a use of the numeral converters (iterators) $\mathsf{I}\colon \mathcal{N}(\zeta) \to \zeta''$ available in $\lambda\mathsf{C}$. The third inclusion is trivial. $\dashv$

Theorem 1.2(2) will follow from an inclusion $\mathcal{R}(\lambda\mathsf{L}) \subseteq \mathcal{S}(\lambda\mathsf{A})$. We postpone a proof of that until section 8 where we develop a technique suitable for the job. In the remainder of this section we describe some of the properties of these classes.

We know that $\mathcal{E} = \mathcal{U}(\lambda\mathsf{A}) \subseteq \mathcal{S}(\lambda\mathsf{A})$, so the first thing we should ascertain is that $\mathcal{S}(\lambda\mathsf{A})$ is a strictly bigger class. To do this consider the predecessor function $\delta \in \mathbb{N}'$ given by $\delta 0 = 0$ and $\delta x' = x$ for $x \in \mathbb{N}$. You should convince yourself that this is not an extended polynomial. However, we can show that it does belong to the various classes.

EXAMPLE 7.3. Let $\zeta = \mathcal{N} \times \mathcal{N}$. Working in $\lambda\mathsf{A}$ set

$$\widehat{\delta} = \lambda x\colon \zeta''.\, \mathsf{Right}(x \ulcorner D \urcorner(\mathsf{Pair}\, 00)),$$

where

$$\ulcorner D \urcorner = \lambda w\colon \zeta.\, \mathsf{Pair}(\mathsf{S}(\mathsf{Left}\, w))(\mathsf{Left}\, w)$$

using the pairing gadgets associated with ζ. Once you are let into the secret it is easy to check that $\widehat{\delta}\colon \zeta'' \to \mathcal{N}$ imitates δ, and hence $\delta \in \mathcal{L}$. Using the iterator $\mathsf{I}\colon \mathcal{N}[\zeta] \to \zeta''$ we can convert $\widehat{\delta}$ into a name $\ulcorner \delta \urcorner\colon \mathcal{N}[\zeta] \to \mathcal{N}$ for δ in $\lambda\mathsf{C}$. In fact, δ can be simulated in the pure simply typed λ-calculus without any numeric or pairing gadgets.

This example is a variant of the way Kleene obtained δ in the λ-calculus. I have left it unexplained for we are going to look at a more general technique. The class $\mathcal{L}$ has several closure properties some of which look quite strong. To get a feel for this let us see how many forms of recursion can be replaced by iteration, even those that go way beyond $\mathcal{L}$.

EXAMPLES 7.4. Let $\mathbb{P}$ be a space of parameters. We may think of $\mathbb{P}$ as $\mathbb{N}^l$ for some l.

(a) Using the data functions

$$g\colon \mathbb{P} \to \mathbb{N}, \quad h\colon \mathbb{N} \to \mathbb{N} \to \mathbb{P} \to \mathbb{N}, \quad d\colon \mathbb{N} \to \mathbb{P} \to \mathbb{P},$$

let $f\colon \mathbb{N} \to \mathbb{P} \to \mathbb{N}$ be the function specified by the recursion

$$f0p = gp, \quad fx'p = hxtp, \quad \text{where } t = fxq, \quad \text{where } q = dxp$$

for $x\colon \mathbb{N}$ and $p\colon \mathbb{P}$. (Remember that $x' = x + 1$.) To replace such a recursion by an iteration let $\mathbb{G} = \mathbb{P} \to \mathbb{N}$ and consider $H\colon \mathbb{N} \times \mathbb{G} \to \mathbb{N} \times \mathbb{G}$ defined by

$$H(x, \phi) = (x', \psi), \quad \text{where } \psi p = hxtp, \quad \text{where } t = \phi q, \quad \text{where } q = dxp$$

for $x\colon \mathbb{N}$, $\phi\colon \mathbb{G}$, and $p\colon \mathbb{P}$. In particular $H(x, fx) = (x', fx')$ and hence $H^x(0, g) = (x, fx)$ from which we may extract f.

(b) Using the data functions

$$g\colon \mathbb{P} \to \mathbb{N}, \quad h\colon \mathbb{N} \to \mathbb{N} \to \mathbb{P} \to \mathbb{N}, \quad d\colon \mathbb{N} \to \mathbb{P} \to \mathbb{P}, \quad e\colon \mathbb{N} \to \mathbb{P}$$

let $f\colon \mathbb{N} \to \mathbb{P} \to \mathbb{N}$ be the function specified by the recursion

$$f0p = gp, \quad fx'p = hxtp,$$
$$\text{where } t = fxq, \quad q = dsp, \quad s = fxb, \quad b = e(fxp)$$

for $x\colon \mathbb{N}$ and $p\colon \mathbb{P}$. In this recursion the parameters vary *and* there is a nesting of f. Such a recursion can be replaced by an iteration using the same trick. To do this think of the step $\phi \mapsto \psi$ where $\phi = fx$ and $\psi = fx'$. I will leave you to work out the details.

We take this idea and mimic it in λA using simulated numerals. For this product types and pairing gadgets are essential (to create the terms in parallel). Also we need a way of notating the parameters p, but in practice (when $\mathbb{P} = \mathbb{N}^l$) this is not a problem. The main obstacle is that the types involved must have a certain compatibility.

Before you read the proof of the next result you should try what may seem a rather childish exercise. Examples 7.4(a) is done in monochrome. In the coloured version there are potentially seven different colours of numbers. Try re-doing the calculation in colour.

How can we mimic the recursion of Examples 7.4(a)? Suppose we require the values of f to be simulated in ζ'' for some type ζ. The parameters will not be a problem, and we can suppose they are notated in some type π. We must simulate f in a type $\theta'' \to \pi \to \zeta''$ for some θ. We assume the three data functions are simulated with certain types. Within these there are five possible colours of natural numbers, but some of these must be ζ''.

THEOREM 7.5. *Consider an instance of the recursion of Examples* 7.4(a). *Suppose the three data functions are simulated by terms*

$$\overline{g}\colon \pi \to \zeta'', \quad \overline{\overline{h}}\colon \eta'' \to \zeta'' \to \pi \to \zeta'', \quad \overline{\overline{\overline{d}}}\colon \eta'' \to \pi \to \pi$$

for some types ζ, η and π. Then the function f is simulated by a term

$$\check{f}\colon \theta'' \to \pi \to \zeta''$$

where the new source type is $\theta = \eta'' \times \gamma$ for $\gamma = \pi \to \zeta''$.

PROOF. We need to take some care with the typing discipline. Thus we write

$$\widetilde{p}\colon \pi, \quad \dot{\overline{m}}\colon \zeta'', \quad \ddot{\overline{m}}\colon \eta'', \quad \dddot{\overline{m}}\colon \theta''$$

for the notated parameters in π, and for the numerals over ζ, η and θ. We are given

$$\overline{g}\,\widetilde{p} \rhd\!\!\rhd \overline{\dot{g}p}, \quad \overline{\overline{h}}\,\ddot{\overline{m}}\,\dot{\overline{n}}\,\widetilde{p} \rhd\!\!\rhd \overline{\dot{hmnp}}, \quad \overline{\overline{\overline{d}}}\,\ddot{\overline{m}}\,\widetilde{p} \rhd\!\!\rhd \widetilde{dmp}$$

for $m, n \in \mathbb{N}$ and $p \in \mathbb{P}$. We generate a sequence $(\,\phi_m \mid m < \omega\,)$ of terms by

$$\phi_0 = \overline{g}, \quad \phi_{m'} = \lambda p\colon \pi.\overline{\overline{h}}\,\ddot{\overline{m}}(\phi_m q)p \quad \text{where } q = \overline{\overline{\overline{d}}}\,\ddot{\overline{m}}\,p$$

and check that $\vdash \phi_m\colon \gamma$ and $\phi_m\widetilde{p} \rhd\!\!\rhd\!\!\rhd \overline{\dot{fmp}}$. With $\theta = \eta'' \times \gamma$ we use the pairing functions

$$\mathsf{Pair}\colon \eta'' \to \gamma \to \theta, \quad \mathsf{Left}\colon \theta \to \eta'', \quad \mathsf{Right}\colon \theta \to \gamma$$

to produce some auxiliary terms. With identifiers $v\colon \theta$ and $p\colon \pi$ we let

$$l = \mathsf{Left}\,w, \quad \phi = \mathsf{Right}\,w, \quad q = \overline{\overline{\overline{d}}}\,lp, \quad t = \phi q, \quad r = \lambda p\colon \pi.\overline{\overline{h}}ltp$$

so that $l\colon \eta''$, $\phi\colon \gamma$, $q\colon \pi$, $t\colon \zeta''$, $r\colon \gamma$. Let $\overline{\overline{\mathsf{S}}}\colon \eta'''$ simulate the successor function, and set

$$G = \mathsf{Pair}\,\ddot{\overline{0}}\overline{g}, \qquad H = \lambda v\colon \theta.\,\mathsf{Pair}(\overline{\overline{\mathsf{S}}}l)r$$

to produce $G\colon \theta$ and $H\colon \theta'$ as auxiliary terms. With these we check that

$$\dddot{\overline{m}}HG \rhd\!\!\rhd H^m G \rhd\!\!\rhd\!\!\rhd \mathsf{Pair}\,\ddot{\overline{m}}\phi_m, \quad \mathsf{Right}(\dddot{\overline{m}}HG)\widetilde{p} \rhd\!\!\rhd \overline{\dot{fmp}}$$

for each $m \in \mathbb{N}$ where the second uses the calculation above. Finally, we set

$$\check{f} = \lambda w\colon \theta'', p\colon \pi.\,\mathsf{Right}(wHG)p$$

to produce the required simulation. ⊣

This result seems to say that any reasonable form of recursion can be mimicked using simulated numerals. In fact, things are worse (or better). There is a corresponding result which deals with Example 7.4(b). Furthermore, if you write down any recursion and then replace it by an iteration, you will find there is a corresponding simulation result.

There is a catch. The colours of certain numerals must match; certain types must be the same. In Theorem 7.5 the types of the data functions can use only two colours; the three ζs and the two ηs. This compatibility is needed to produce the sequence $(\,\phi_m \mid m < \omega\,)$. Attemp to mimic the construction of the stacking function $\beth$ as a primitive recursive function but using simulated numerals. After a couple of steps the required typing restrictions fail. It is these extra requirements that tiering is designed to highlight.

Classes of simulated functions can be closed under many forms of recursion *provided some tiering-like restrictions are met*. As far as I know there has been no comprehensive investigation of just what these restrictions entail, but much information can be found in the literature. Let us look at a simple example of such a closure property. This also illustrates how we handle the notating of parameters.

LEMMA 7.6. *For a 2-placed function $k \in \mathcal{E}$ and a $(1+r)$-placed function $l \in \mathcal{S}(\lambda\mathsf{A})$ consider the $(1+r)$-placed function f specified by the recursion*

$$f\,0\mathsf{y} = \text{constant}, \quad f\,x'\mathsf{y} = kt(lx\mathsf{y}), \quad \textit{where } t = f\,x\mathsf{y}$$

where the constant is fixed and y *is an arbitrary list from* $\mathbb{N}$ *of length* r. *Then* $f \in \mathcal{S}(\lambda\mathsf{A})$.

PROOF. Consider any type ζ. We must produce a simulation of f with target ζ''. Since $k \in \mathcal{E}$ and $l \in \mathcal{S}(\lambda\mathsf{A})$, there are terms

$$\overline{k}\colon \zeta'' \to \zeta'' \to \zeta'', \quad \overline{l}\colon \eta'' \to \chi(r)'' \to \cdots \to \chi(1)'' \to \zeta''$$

which simulate k and l, respectively. Here $\chi(1), \ldots, \chi(l), \eta$ are some types which depend on ζ and l. In particular, we have little control over what they can be. It is convenient to collapse the parameters types $\chi(1), \ldots, \chi(l)$ to one. Thus let

$$\pi = \chi(l)'' \times \cdots \times \chi(1)''$$

so, with a bit a licence, we have

$$\overline{l}\colon \eta'' \to \pi \to \zeta''$$

as the simulating term. In the same way we may rephrase the specification as

$$f\,0p = gp, \quad f\,x'p = hxtp, \quad \text{where } t = f\,x\mathsf{y}$$

where g is the appropriate constant function and

$$hxtp = kt(lxp)$$

for all $x, t \in \mathbb{N}$ and parameters p. This specification fits into Examples 7.4(a) where d is a projection function. Using the given simulations $\overline{k}$ and $\overline{l}$ we may construction a term

$$\overline{\overline{h}}\colon \eta'' \to \zeta'' \to \pi \to \zeta''$$

which simulates h. Thus we may invoke Theorem 7.5. ⊣

The important point here is that in the construction $(\overline{k}, \overline{l}) \longmapsto \check{f}$ only the type of the active argument is changed. The types of the parameters and the target are unchanged. Returning to the uncollapsed notation we have

$$\begin{gathered}\overline{l}\colon \eta'' \to \chi(r)'' \to \cdots \to \chi(1)'' \to \zeta'' \\ \check{f}\colon \theta'' \to \chi(r)'' \to \cdots \to \chi(1)'' \to \zeta'' \\ \theta = \eta'' \times (\chi(r)'' \to \cdots \to \chi(1)'' \to \zeta'')\end{gathered}$$

which succinctly indicates what is happening.

Let us use this result to show how bounded sums and bounded products can be handled.

Consider any function $l : \mathbb{N} \to \mathbb{P} \to \mathbb{N}$ where, in practice, $\mathbb{P}$ is a power of $\mathbb{N}$. The two functions $\sum[l]$ and $\prod[l]$, of the same type as l, are defined by

$$\sum[l]xp = \sum_{i<x} lip, \quad \prod[l]xp = \prod_{i<x} lip$$

for $x \in \mathbb{N}, p \in \mathbb{P}$. Using either addition or multiplication for k we can specify these by a recursion as in Lemma 7.6. Thus we obtain the following.

COROLLARY 7.7. *Suppose* $l : \mathbb{N} \to \mathbb{P} \to \mathbb{N}$ *has a simulation* $\overline{l} : \eta'' \to \gamma$ *where* $\gamma = \pi \to \zeta''$. *Then both* $f = \sum[l]$ *and* $f = \prod[l]$ *have simulations with type* $\theta'' \to \gamma$ *where* $\theta = (\eta'' \times \gamma)$.

In particular, the class $\mathcal{S}(\lambda\mathsf{A})$ *is closed under bounded sums and bounded products.*

In section 8 we show that $\mathcal{L} = \mathcal{S}(\lambda\mathsf{A})$, which means $\mathcal{L}$ will seem familiar. The Kalmár elementary functions form the the least class $\mathcal{K}$ which is closed under composition and bounded sums and products, includes $\mathcal{E}$, and contains 2-placed modified subtraction Δ given by $\Delta yx = x - y$ if $x \leq y$ and 0 otherwise. It is known that $\mathcal{L} \subseteq \mathcal{K}$ but Δ is not in $\mathcal{L}$. The precise nature of the class $\mathcal{L}$ is not at all clear (at least to me).

§8. The implementation technique. In this section we prove all the outstanding results. These are: Theorem 1.1; the remaining part of Theorem 1.2, namely part (2); and an improvement of Lemma 7.2 to an equality. We also show how a natural restriction on a tiered system bounds its strength to $\mathcal{L}$.

We use a technique which enables us to put an upper bound on the strength of certain tiered systems $\lambda\mathsf{T}$. We compare $\lambda\mathsf{T}$ with a system $\lambda\mathsf{S}$, whose strength is known, using an implementation of $\lambda\mathsf{T}$ in $\lambda\mathsf{S}$, a syntactic translation which respects the typing discipline and almost respects the reduction mechanism, but does *not* respect the semantics.

We use a simple version of this technique where $\lambda\mathsf{S}$ is $\lambda\mathsf{A}$ and $\lambda\mathsf{T}$ satisfies a certain restriction. This will certainly apply to $\lambda\mathsf{B}, \lambda\mathsf{C}, \lambda\mathsf{L}$ (but not to some of the other listed systems). This version uses an analogue of Example 5.2. We use a term which for $\lambda\mathsf{T}$ names an iterator, but for $\lambda\mathsf{A}$ names an identity function and simulates an iterator.

Before we give the particular details let us look at the general idea.

We build the implementation of $\lambda\mathsf{T}$ in $\lambda\mathsf{A}$ by translating the syntactic categories. We translate the types $\tau \longmapsto |\tau|$ and the terms $t \longmapsto |t|$ in the following way.

$$\begin{array}{llll} |\mathcal{N}| = \mathcal{N} & |\xi| = \text{as selected} & |\sigma \to \rho| = |\sigma| \to |\rho| & |\sigma \times \rho| \quad = |\sigma| \times |\rho| \\ |x| \ = x & |\mathsf{k}| = \text{as selected} & |qp| \quad = |q||p| & |\lambda x : \sigma \,.\, r| = \lambda x : |\sigma| . |r|. \end{array}$$

On the top line we find the translation of types. The monochrome atom $\mathcal{N}$ of $\lambda\mathsf{T}$ is also an atom of $\lambda\mathsf{A}$, so that is sent to itself. A coloured atom ξ of

$\lambda\mathsf{T}$ is not part of $\lambda\mathsf{A}$, so it must be sent to some type $|\xi|$ of $\lambda\mathsf{A}$. Thereafter the translation follows the syntactic structure. Thus, the translation of types $\tau \longmapsto |\tau|$ works by merely replacing each coloured atom by an appropriate compound. The translation of terms $t \longmapsto |t|$ works in a similar way. We follow the syntax, so that only the translation $|\mathsf{k}|$ of a constant k needs some thought. Some of these constants will be a part of $\lambda\mathsf{A}$, and these are translated as themselves. The heart of the matter is the translation of those constants of $\lambda\mathsf{T}$ which are not part of $\lambda\mathsf{A}$.

- Each pairing gadget is translated into the corresponding pairing gadget.

$$|\,\mathsf{Pair}_{\sigma,\rho}\,| = \mathsf{Pair}_{|\sigma|,|\rho|}, \quad |\,\mathsf{Left}_{\sigma,\rho}\,| = \mathsf{Left}_{|\sigma|,|\rho|}, \quad |\,\mathsf{Right}_{\sigma,\rho}\,| = \mathsf{Right}_{|\sigma|,|\rho|}\,.$$

- The atom $\mathcal{N}$ is its furnishings $\mathsf{0}\colon \mathcal{N}$ and $\mathsf{S}\colon \mathcal{N}'$ are translated as themselves.
- For each coloured atom ξ we select $|\xi| = \zeta''$ for some $\lambda\mathsf{A}$-type ζ. This is done later.
- For the furnishings $\mathsf{0}_\xi\colon \xi$ and $\mathsf{S}_\xi\colon \xi'$ of a coloured atom we have $|\xi| = \zeta''$ for some $\lambda\mathsf{A}$-type ζ. We set $|\mathsf{0}_\xi| = \overline{\mathsf{0}_\zeta}$ and $|\mathsf{S}_\xi| = \overline{\mathsf{S}_\zeta}$ using the simulated furnishings of ζ''.

As can be expected, we will need to translate plenty of authentic numerals $\ulcorner m_\xi \urcorner$. Trivially, each monochrome numeral $\ulcorner m \urcorner = \ulcorner m_\mathcal{N} \urcorner$ translates into itself. The coloured ones do not.

EXAMPLE 8.1. Let ξ be a coloured atom with $|\xi| = \zeta''$. Then

$$|\ulcorner m_\xi \urcorner| = |\mathsf{S}_\xi^m \mathsf{0}_\xi| = |\mathsf{S}_\xi|^m |\mathsf{0}_\xi| = \overline{\mathsf{S}_\zeta}^m \overline{\mathsf{0}_\zeta} \rhd\!\!\rhd \overline{m_\zeta}$$

for each $m \in \mathbb{N}$.

The crunch comes when we translate iterators. For this we need a restriction on $\lambda\mathsf{T}$.

RESTRICTION 8.2. For each coloured atom ξ of $\lambda\mathsf{T}$ there is a unique type τ and iterator $\mathsf{I}\colon \xi \to \tau''$. The atom ξ does not occur in τ, and every iterator arises in this way.

This says that the coloured atoms ξ and the iterators I of $\lambda\mathsf{T}$ come in matched pairs. You cannot have one without the other; each determines the other. There is also a restriction on the target type τ of I. It must be created without the use of ξ. You will see why shortly.

The systems $\lambda\mathsf{B}$, $\lambda\mathsf{C}$, and $\lambda\mathsf{L}$ meet this restriction. The systems $\lambda\mathsf{D}$ and $\lambda\mathsf{E}$ do not since, in these systems, for each coloured atom ξ there are iterators $\mathsf{I}_{\xi,\tau}\colon \xi \to \tau''$ for several types τ. The system $\lambda\mathsf{F}$ fails this test, for the restriction insists that only coloured atoms support iterators. The system $\lambda\mathsf{G}$ gets nowhere near meeting the restriction.

From now on we assume that $\lambda\mathsf{T}$ satisfies Restriction 8.2.

How do we translate coloured atoms ξ and iterators I? By Restriction 8.2 these occur in matched pairs and we have $\mathsf{I}\colon \xi \to \tau''$ for a unique type τ. The

atom ξ does not occur in τ, so we may calculate $|\tau|$ without knowing $|\xi|$. Thus the pair I, ξ determines a $\lambda\mathsf{A}$-type $\zeta = |\tau|$, and we select $|\xi| = |\tau''| = \zeta''$. We translate the iterator as

$$|\mathsf{I}| = \lambda w : \zeta'', y : \zeta', x : \zeta . wyx$$

which is the name of an identity function.

A re-reading of the translations of the primitives gives the following.

LEMMA 8.3. *For each axiom* $\mathsf{k} : \kappa$ *of the restricted system* $\lambda\mathsf{T}$ *there is a* $\lambda\mathsf{A}$*-derivation*

$$(\nabla(\mathsf{k})) \quad \vdash |\mathsf{k}| : |\kappa|$$

in the empty context using the translation $|\kappa|$ *of the housing type.*

This give a base for the translation of the next syntactic category, the derivations. Starting at the root of a $\lambda\mathsf{T}$-derivation

$$(\nabla) \quad \Gamma \vdash t : \tau$$

and working upwards towards the leaves, we allow $|\cdot|$ to pass across the construction rules for derivations in the obvious way. This will almost produce a $\lambda\mathsf{A}$-derivation. Only the treatment of an Axiom, as on the left, may be a problem.

$$(\text{Axiom}) \quad \Gamma \vdash \mathsf{k} : \kappa, \qquad (|\text{Axiom}|) \quad |\Gamma| \vdash |\mathsf{k}| : |\kappa|.$$

Its translation, on the right, need not be a $\lambda\mathsf{A}$-derivation. However, using the $\lambda\mathsf{A}$-derivation $\nabla(\mathsf{k})$ we may grow a $\lambda\mathsf{A}$-derivation above $|\text{Axiom}|$. Thus we have the following.

LEMMA 8.4. *Each* $\lambda\mathsf{T}$*-derivation* ∇ *translates into a* $\lambda\mathsf{A}$*-derivation* $|\nabla|$

$$(\nabla) \quad \Gamma \vdash t : \tau \quad \longmapsto \quad (|\nabla|) \quad |\Gamma| \vdash |t| : |\tau|$$

obtained by growing extra branches above the Axiom leaves of ∇.

This sets up the translation

$$\lambda\mathsf{T} \xrightarrow{\ |\cdot|\ } \lambda\mathsf{A}$$

for any tiered system $\lambda\mathsf{T}$ which meets Restriction 8.2. The translation respects the typing discipline, that is derivations are sent to derivations, but the semantics is certainly not respected (unless $\lambda\mathsf{T}$ happens to be $\lambda\mathsf{A}$). Indeed, the whole idea is that it should *not* preserve the meaning of a derivation. However, as we will see, $|\cdot|$ does preserve strength.

How does the translation $|\cdot|$ interacts with reductions. The obvious 'result' is that for $\lambda\mathsf{T}$-terms t^-, t^+ a reduction $t^- \twoheadrightarrow t^+$ gives $|t^-| \twoheadrightarrow |t^+|$, This is not quite right.

LEMMA 8.5. *For each pair* t^-, t^+ *of* $\lambda\mathsf{T}$*-terms and a reduction, as on the left*

$$t^- \twoheadrightarrow t^+, \quad |t^-| \twoheadrightarrow\!\!\!\!\twoheadrightarrow t^0, \quad |t^+| \twoheadrightarrow\!\!\!\!\twoheadrightarrow t^0$$

there is a $\lambda\mathsf{A}$*-term* t^0 *and a pair of reductions, as on the right.*

PROOF. It suffices to check what happens to each 1-step reduction $t^- \rhd t^+$ in $\lambda\mathsf{T}$. In fact, by traipsing through the cases, we see that almost all of these translate into a 1-step reduction $|t^-| \rhd |t^+|$ in $\lambda\mathsf{A}$. There is just a couple we should look at.

Consider the case of a redex removal, as to the left. The crucial observation

$$(\lambda x\colon \sigma.r)s \rhd r[x\colon = s] \quad |r[x\colon = s]| = |r|[x\colon = |s|]$$

(which can be verified by a tedious induction), is that the translation commutes with substitution (as to the right). In particular, a redex removal does translate into a redex removal.

The case that does not translate in the obvious fashion is that of an iterator reduction

$$t^- = \mathsf{I}(\mathsf{S}_\xi r)ts, \quad t^+ = t(\mathsf{I}rts)$$

with a successor argument. To deal with this let $T = \mathsf{I}rts$ and $A = |r||t||s|$ (so that T is a $\lambda\mathsf{T}$-term and A is a $\lambda\mathsf{A}$-term). Since $|\mathsf{I}|$ is an identity term we have $|T| = |\mathsf{I}||r||t||s| \rhd\!\rhd A$ in $\lambda\mathsf{A}$. Thus $|t^+| = |t||T| \rhd\!\rhd |t|A$ in $\lambda\mathsf{A}$. Next, remembering the shape of $|\mathsf{S}_\xi|$, we have

$$|t^-| = |\mathsf{I}|(|\mathsf{S}_\xi||r|)|t||s| \rhd\!\rhd |\mathsf{S}_\xi||r||t||s| \rhd\!\rhd |t|(|r||t||s|) = |t|A$$

in $\lambda\mathsf{A}$. Thus $|t^-| \rhd\!\rhd |t|A$ and $|t^+| \rhd\!\rhd |t|A$ to show that $t^0 = |t|A$ is the required term. ⊣

We are now in a position to obtain the variant of the Bellantoni and Cook result.

THEOREM 8.6. *The strength of* $\lambda\mathsf{B}$ *is exactly* $\mathcal{H}$.

PROOF. By Definition 6.2, for each humble function $f\colon \mathbb{N}(l)$, there is a term of $\lambda\mathsf{A}$

$$\vdash \widehat{f}\colon \widehat{\mathcal{N}_l} \to \cdots \to \widehat{\mathcal{N}_1} \to \mathcal{N}$$

which imitates f. It is convenient to view f as a meek function with the simulated and authentic inputs in the canonical position. Thus we have a $\lambda\mathsf{A}$-derivable term

$$\text{(a)} \qquad \vdash \overline{f}\colon \mathcal{N}'' \to \cdots \to \mathcal{N}'' \to \mathcal{N} \to \cdots \to \mathcal{N} \to \mathcal{N}$$

which imitates essentially the same function f. Here the are s occurrences of $\mathcal{N}''$ and $a+1$ occurrences of $\mathcal{N}$. This is also a term of $\lambda\mathsf{B}$. Using the iterator $\mathsf{I}\colon \mathcal{N}[\star] \to \mathcal{N}''$ we may produce a numeral converter $\mathsf{L}\colon \mathcal{N}[\star] \to \mathcal{N}''$ with

$$\mathsf{L}\ulcorner n_\star \urcorner \rhd\!\rhd \overline{n}$$

for each $n \in \mathbb{N}$. Here $\ulcorner n_\star \urcorner$ is the authentic numeral over $\mathcal{N}[\star]$ and $\overline{n}$ is the simulated numeral over $\mathcal{N}$. We set

$$\ulcorner f \urcorner = \lambda v_s; \mathcal{N}[\star], \ldots, v_1\colon \mathcal{N}[\star], u_a\colon \mathcal{N}, \ldots, u_1\colon \mathcal{N}.\overline{f}(\mathsf{L}v_s)\ldots(\mathsf{L}v_l)u_a \ldots u_1$$

to obtain a λB-derivable term

$$\text{(b)} \qquad \vdash \ulcorner f \urcorner : \mathcal{N}[\star] \to \cdots \to \mathcal{N}[\star] \to \mathcal{N} \to \cdots \to \mathcal{N} \to \mathcal{N}$$

which, by a simple sequence of reductions, we see names f.

Conversely, consider any function $f : \mathbb{N}(l)$ which is named in λB. By shuffling around the inputs we may suppose it is named by a term $\ulcorner f \urcorner$, as in (b) above. Since $|\mathcal{N}[\star]| = \mathcal{N}''$ this λB-derivation translates into a λA derivation (a) where $\overline{f} = |\ulcorner f \urcorner|$. We show that $\overline{f}$ imitates f, and hence $f \in \mathcal{H}$. To prevent subscript nausea let us suppose there is one input of each kind. Thus

$$\text{(a)} \qquad \vdash \overline{f} : \mathcal{N}'' \to \mathcal{N} \to \mathcal{N}$$

$$\text{(b)} \qquad \vdash \ulcorner f \urcorner : \mathcal{N}[\star] \to \mathcal{N} \to \mathcal{N}$$

are the two derivable terms, and we are given

$$\ulcorner f \urcorner \ulcorner n_\star \urcorner \ulcorner m \urcorner \rhd\!\!\rhd \ulcorner f n m \urcorner$$

for each $m, n \in \mathbb{N}$. Lemma 8.5 gives a term t^0 such that both

$$\overline{f}\,|\ulcorner n_\star \urcorner|\ulcorner m \urcorner = |\ulcorner f \urcorner \ulcorner n_\star \urcorner \ulcorner m \urcorner| \rhd\!\!\rhd\!\!\rhd\ t_0, \quad \ulcorner f n m \urcorner = |\ulcorner f n m \urcorner| \rhd\!\!\rhd\!\!\rhd\ t^0$$

hold. But $\ulcorner f n m \urcorner$ is normal and hence $t_0 = \ulcorner f n m \urcorner$. By Example 8.1 we have

$$\overline{f}\,|\ulcorner n_\star \urcorner|\ulcorner m \urcorner \rhd\!\!\rhd \overline{f}\,\overline{n}\ulcorner m \urcorner$$

so that a use of confluence and the normality of t_0 gives

$$\overline{f}\,\overline{n}\ulcorner m \urcorner \rhd\!\!\rhd \ulcorner f n m \urcorner,$$

as required. $\dashv$

As a corollary the strength of λB is exactly the class $\mathcal{E}$ of extended polynomials. As a scholium the strength of λB is exactly the class $\mathcal{M}$ of meek functions where the status of the inputs—simulated or authentic—is preserved by the representation.

A similar argument gives the variant of the Leivant result.

THEOREM 8.7. *We have* $\mathcal{S}(\lambda\mathsf{A}) = \mathcal{L} = \mathcal{R}(\lambda\mathsf{C}) = \mathcal{R}(\lambda\mathsf{L})$.

PROOF. By Lemma 7.2 it suffices to show $\mathcal{R}(\lambda\mathsf{L}) \subseteq \mathcal{S}$.

Consider any l-placed function $f \in \mathcal{R}(\lambda\mathsf{L})$. This is named by a derivable term

$$\vdash \ulcorner f \urcorner : \xi(l) \to \cdots \to \xi(1) \to \xi(0)$$

of λT for atoms $\xi(0), \ldots, \xi(l)$. We arrange $\xi(0) = \mathcal{N}$ without changing the other $\xi(i)$.

Consider any coloured atom ξ. We have an iterator $\mathsf{I} : \xi \to \tau''$ for a unique type τ. Also $\tau \downarrow \chi$ for some atom χ together with a numeral converter $\mathsf{C} : \tau'' \to \chi$. Using these we can massage the name $\ulcorner f \urcorner$ into a name

$$\vdash \ulcorner f \urcorner : \xi(l) \to \cdots \to \xi(1) \to \chi(0)$$

of the same function for some 'lower' atom $\chi(0)$. By making sufficiently many descents of this kind we replace $\xi(0)$ be the monochrome atom $\mathcal{N}$, and modify $\ulcorner f \urcorner$ accordingly. Thus we obtain a 'canonical' name

$$\vdash \ulcorner f^{+} \urcorner \colon \xi(l) \to \cdots \to \xi(1) \to \mathcal{N}$$

of the function f.

We now use the homogeneity of λL. Anything that happens above $\mathcal{N}$ is replicated above each coloured atom $\mathcal{N}[\zeta]$. Using $\ulcorner f^{+} \urcorner$ as a template we produce a different name for the same function where the target type is $\xi(0) = \mathcal{N}[\zeta]$ where ζ is any λA-type of our choice. This process generates λL-types $\zeta(1), \ldots, \zeta(l)$ such that the new source types are $\xi(i) = \mathcal{N}[\zeta(i)]$ for $1 \leq i \leq l$. In other words we now view '$\mathcal{N}$' as a variable and make a substitution $\mathcal{N} \mapsto \mathcal{N}[\zeta]$ throughout. This is a kind of external polymorphism in use.

Thus, starting from any name for f we produce a name

$$\vdash \ulcorner f \urcorner \colon \xi(l) \to \cdots \to \xi(1) \to \xi(0),$$

where

$$\xi(l) = \mathcal{N}[\zeta(l)], \ldots, \xi(1) = \mathcal{N}[\zeta(1)], \xi(0) = \mathcal{N}[\zeta]$$

and where ζ is an λA-type of out choice.

We translate this new derivation into a λA-derivation. We have $|\xi(0)| = \zeta(0)''$ where $\zeta(0) = \zeta$, and for each $1 \leq i \leq l$ there is some λA-type $\zeta(i)$ with $|\xi(i)| = \zeta(i)''$. Thus the new derivation translates into a λA-derivation

$$\vdash \overline{f} \colon \zeta(l)'' \to \cdots \to \zeta(1)'' \to \zeta(0)''$$

where $\overline{f} = |\ulcorner f \urcorner|$. We have chosen $\zeta(0) = \zeta$ at the outset, but we must accept $\zeta(1), \ldots, \zeta(l)$ as produced by the translation. We show that $\overline{f}$ simulates f.

Consider any $m(0), m(1), \ldots, m(l) \in \mathbb{N}$ where $f m(l) \ldots m(1) = m(0)$. We know that

$$\ulcorner f \urcorner \ulcorner m(l)_{\xi(l)} \urcorner \ldots \ulcorner m(1)_{\xi(1)} \urcorner \rhd\!\!\rhd \ulcorner m(0)_{\xi(0)} \urcorner$$

and so Example 8.1 and Lemma 8.5 give

$$\overline{f} |\ulcorner m(l)_{\xi(l)} \urcorner| \ldots |\ulcorner m(1)_{\xi(1)} \urcorner| \rhd\!\!\rhd \overline{f}\,\overline{m(l)_{\zeta(l)}} \ldots \overline{m(1)_{\zeta(1)}},$$
$$\overline{f} |\ulcorner m(l)_{\xi(l)} \urcorner| \ldots |\ulcorner m(1)_{\xi(1)} \urcorner| \rhd\!\!\rhd\!\!\rhd t^0$$

and

$$\overline{m(0)_{\zeta(0)}} \rhd\!\!\rhd\!\!\rhd t^0$$

for some term t^0. The numeral $\overline{m(0)_{\zeta(0)}}$ is normal, so that t^0 is $\overline{m(0)_{\zeta(0)}}$ (up to a change of bound identifiers). Finally, a use of confluence gives

$$\overline{f}\,\overline{m(l)_{\zeta(l)}} \ldots \overline{m(1)_{\zeta(1)}} \rhd\!\!\rhd \overline{m(0)_{\zeta(0)}},$$

as required. $\dashv$

This shows that the two systems $\lambda\mathsf{C}$ and $\lambda\mathsf{L}$ have the same strength, namely $\mathcal{L}$. At first sight this seems a little odd for we feel that $\lambda\mathsf{L}$ should be stronger.

Two tricks are used in this proof. We use numeral converters to massage a name into a canonical form, and we use the homogeneity to move around canonical names. What happens if we do not have this homogeneity? We make more use of numeral converters.

THEOREM 8.8. *Suppose* $\lambda\mathsf{T}$ *meets Restriction* 8.2. *Then its strength is no more than* $\mathcal{L}$.

PROOF. Consider any l-placed first order function $f : \mathbb{N}(l)$ which is named in $\lambda\mathsf{T}$. We must show that $f \in \mathcal{L}$. We are given a $\lambda\mathsf{T}$-term $\ulcorner f \urcorner$ and $\lambda\mathsf{T}$-derivation

$$(\lambda\mathsf{T}) \quad \vdash \ulcorner f \urcorner : \xi(l) \to \cdots \to \xi(1) \to \xi(0)$$

which names f. We have to worry about whether each atom $\xi(i)$ is $\mathcal{N}$ or coloured, and then we act accordingly. Let us consider the particular case

$$(\lambda\mathsf{T}) \quad \vdash \ulcorner f \urcorner : \mathcal{N} \to \xi \to \chi$$

where both ξ and χ are coloured. This is essentially the most complicated case, apart from possible duplications of the input problems.

The given $\lambda\mathsf{T}$-derivation translates into a $\lambda\mathsf{A}$-derivation

$$(|\lambda\mathsf{T}|) \quad \vdash |\ulcorner f \urcorner| : \mathcal{N} \to \zeta'' \to \eta''$$

where $|\xi| = \zeta''$ and $|\chi| = \eta''$. We massage this into the required $\lambda\mathsf{A}$-derivation.

Within $\lambda\mathsf{A}$ there is a pair

$$\mathsf{L} : \eta'' \to \mathcal{N} \quad \mathsf{R} : \mathcal{N}'' \to \mathcal{N}$$

of numeral converters such that

$$\mathsf{L}\overline{k_\eta} \rhd\!\!\rhd \ulcorner k \urcorner, \quad \mathsf{R}\overline{n} \rhd\!\!\rhd \ulcorner n \urcorner$$

for $k, n \in \mathbb{N}$. Let

$$\overline{f} = \lambda v : \mathcal{N}'', u : \zeta''.\mathsf{L}(|\ulcorner f \urcorner|(\mathsf{R}v)u)$$

to give a $\lambda\mathsf{A}$-derivation

$$(\lambda\mathsf{A}) \quad \vdash \overline{f} : \mathcal{N}'' \to \zeta'' \to \mathcal{N}$$

which we show is the required imitation. In other words, we show that

$$\overline{f}\,\overline{n}\,\overline{m_\zeta} \rhd\!\!\rhd \ulcorner k \urcorner$$

for each $m, n, k \in \mathbb{N}$ with $k = fnm$. For such m, n, k we have

$$\ulcorner f \urcorner \ulcorner n \urcorner \ulcorner m_\xi \urcorner \rhd\!\!\rhd \ulcorner k_\chi \urcorner$$

since $\ulcorner f \urcorner$ represents f. By Lemma 8.5 we have

$$|\ulcorner f \urcorner|\ulcorner n \urcorner|\ulcorner m_\xi \urcorner| \rhd\!\!\rhd\!\!\rhd t^0, \quad |\ulcorner k_\chi \urcorner| \rhd\!\!\rhd\!\!\rhd t^0$$

for some term t^0. By Example 8.1 we have

$$|\ulcorner f \urcorner|\ulcorner n \urcorner|\ulcorner m_\xi \urcorner| \rhd\!\!\rhd |\ulcorner f \urcorner|\ulcorner n \urcorner \overline{m_\zeta}, \quad |\ulcorner k_\chi \urcorner| \rhd\!\!\rhd\!\!\rhd \overline{k_\eta},$$

and hence several uses of confluence and the normality of $\text{—}\overline{k_\eta}$ shows that

$$|\ulcorner f \urcorner|\ulcorner n \urcorner \overline{m_\zeta} \rhd\!\!\rhd \overline{k_\eta}$$

holds. Thus

$$\overline{f}\,\overline{n}\,\overline{m_\zeta} \rhd\!\!\rhd \mathsf{L}(|\ulcorner f \urcorner|(\mathsf{R}\overline{n})\overline{m_\zeta}) \rhd\!\!\rhd \mathsf{L}(|\ulcorner f \urcorner|\ulcorner n \urcorner \overline{m_\zeta}) \rhd\!\!\rhd \mathsf{L}\overline{k_\eta} \rhd\!\!\rhd \ulcorner k \urcorner,$$

as required. ⊣

You must be getting tired by now.

§9. Final remarks. This paper is a survey of some of the aspects of tiering but, as mentioned in the Introduction, not a comprehensive survey. In this final section I will mention a few things that have been omitted.

This paper considers only recursion over the natural numbers. However, similar ideas can be applied to almost any free algebra, and it is instructive to work through some of the details. The two papers [13, 14] are a good starting point.

An interesting recent development is the use of methods from linear logic. In this approach there is no explicit tiering, but there are type constructors with 'linear' properties. The papers [5, 7, 8, 15] take this approach. There should be some relationship between the tiering and linear methods, but exactly what is not clear (at least to me).

If you compare the results give here with those in the literature, you might be puzzled by something. Where I use the class $\mathcal{L}$ of latent functions others use the strictly larger class $\mathcal{K}$ of Kalmár elementary function. For instance, this is the case with the Leivant version of Theorem 1.1 and many accounts of Theorem 1.2(2). This is because the systems used elsewhere have an explicit description of the predecessor function which can be iterated to produce subtraction and hence all of the required initial functions of $\mathcal{K}$. The precise relationship between $\mathcal{L}$ and $\mathcal{K}$ is not clear to me, and there seems to be very little published information available (although no doubt somebody out there knows something). The paper [1] obtains a characterization of $\mathcal{K}$ by imposing certain syntactic restriction on $\lambda\mathsf{G}$ with an explicit predecessor function. At first sight this is not a tiering restriction, but I suggest that the resulting system is related to $\lambda\mathsf{E}$. Of course, that proof uses a version of the bounding technique rather than the crude exhaustive search method.

You may have found this paper a little terse in parts. This was quite deliberate to keep it down to a reasonable size. A more leisurely account given in [18].

Finally, I thank Matthew Collinson who read a draft of this paper and, being true to his Aberdonian upbringing, went through it with a fine tooth comb picking up some errors and suggested several improvements.

REFERENCES

[1] A. Beckmann and A. Weiermann, *Characterizing the elementary recursive functions by a fragment of Gödel's T*, ***Archive for Mathematical Logic***, vol. 39 (2000), pp. 475–491.

[2] S. Bellantoni, ***Predicative recursion and computational complexity***, Ph.D. thesis, University of Toronto, 1992.

[3] ———, *Predicative recursion and the polytime hierarchy*, In Clote and Remmel [6], pp. 15–29.

[4] S. Bellantoni and S. Cook, *A new recursion-theoretic characterization of the poly-time functions*, ***Computational complexity***, vol. 2 (1992), pp. 97–110.

[5] S. J. Bellantoni and M. Hofmann, *A new 'feasible' arithmetic*, ***The Journal of Symbolic Logic***, vol. 67 (2002), pp. 104–116.

[6] P. Clote and J. Remmel (editors), ***Feasible mathematics II***, Perspectives in Computer Science, Birkhäuser, Boston, New York, 1995.

[7] M. Hofmann, *A mixed modal/linear lambda calculus with application to Bellantoni-Cook safe recursion*, ***CSL 97***, Lecture Notes in Computer Science, vol. 1414, Springer-Verlag, pp. 275–295.

[8] ———, *Linear types and non-size-increasing polynomial time computation*, ***Proceedings of the Symposium of the 14th LICS***, CS Press, 1999.

[9] ———, *Safe recursion with higher types and BCK-algebra*, ***Annals of Pure and Applied Logic***, vol. 104 (2000), pp. 113–166.

[10] D. Leivant, *Finitely stratified polymorphism*, ***Information and Computation***, vol. 93 (1991), pp. 93–113.

[11] ———, *Ramified recurrence and computational complexity I: Word recurrence and poly-time*, In Clote and Remmel [6], pp. 320–343.

[12] ———, *Ramified recurrence and computational complexity III: Higher type recurrence and elementary complexity*, ***Annals of Pure and Applied Logic***, vol. 96 (1999), pp. 209–229.

[13] D. Leivant and J. Y. Marion, *A λ-calculus characterization of poly-time*, ***Typed lambda calculi and applications*** (M. Bezem and J. F. Groote, editors), Lecture Notes in Computer Science, vol. 664, Springer-Verlag, 1993, pp. 274–288.

[14] ———, *Lambda calculus characterization of poly-time*, ***Fundamenta Informatica***, vol. 19 (1993), pp. 167–184.

[15] A. S. Murawski and C.-H. L. Ong, *Can safe recursion be interpreted in light linear logic?*, paper presented at ICC '00, 2000.

[16] H. Schwichtenberg, *Definierbare Funktionen im Lambda-Kalkül mit Typen*, ***Archiv für Logik und die Grundlagen der Mathematik***, vol. 17 (1976), pp. 113–114.

[17] H. Simmons, *The realm of primitive recursion*, ***Archive for Mathematical Logic***, vol. 27 (1988), pp. 177–188.

[18] ———, *Tiering for natural number systems*, `http://www.cs.man.ac.uk/~hsimmons/DOCUMENTS/papersandnotes.html`.

MATHEMATICAL FOUNDATIONS GROUP
THE UNIVERSITY OF MANCHESTER
OXFORD ROAD
MANCHESTER M13 9PL, ENGLAND
E-mail: hsimmons@manchester.ac.uk