Bar recursion is not T+*min* definable

John Longley

February 2, 2015

Abstract

This note supplies the proof of a theorem stated in the forthcoming book by Longley and Normann: namely, that Spector's *bar recursion* functional is not representable by a left-well-founded sequential procedure, and hence is not definable in the language T + *min* (that is, Gödel's System T augmented with the minimization operator), for instance within the Kleene–Kreisel model of total continuous functionals.

1 Introduction

In the forthcoming book by Longley and Normann [3], it is stated without proof (as Theorem 6.3.28[1]) that Spector's *bar recursion* functional is not computable by any *left-well-founded* nested sequential procedure, and hence is not expressible in the programming language T + *min* (i.e. Gödel's System T augmented with the minimization or 'μ' operator familiar from basic computability theory). The present note, intended as a supplement to [3], supplies the proof of this theorem.

More precisely, the theorem states that within the *nested sequential procedure* (NSP) model SP^0 as studied in Chapter 6 of [3], no left-well-founded element can play the role of a Spector-style bar recursor, or even a *restricted* one (see Definition 5 below). Since every T + *min* definable procedure is left-well-founded (Theorem 6.3.15), it follows that no (restricted) bar recursor in SP^0 is T + *min* definable. As noted by Corollary 6.3.33, and shown in more detail in Section 3 below, this readily implies that in the total type structure Ct of Kleene–Kreisel continuous functionals (for instance), the bar recursion operator is not μ-*computable*, or even μ-computable relative to the System T recursors. Since it is well known that bar recursion is Kleene S1–S9 computable within Ct (Theorem 8.3.1), we thus see that bar recursion offers a particularly striking example of a phenomenon first established by Bergstra [1]: namely, the existence of Kleene computable functionals in Ct that are not μ-computable (see Section 8.5.2).

[1] Throughout this note, three-part reference numbers for subsections, definitions, theorems etc. will always refer to [3].

We shall begin by recalling a few points of notation and terminology, although we shall rely heavily on [3] for complete definitions and for the basic theory of the model SP^0.

We work with the class T of *simple types* freely generated from the single base type $\mathtt{N}$ (for natural numbers) via the binary type constructor $\to$. We write $\overline{k}$, or sometimes just k, for the *pure type* of level k given by $\overline{0} = \mathtt{N}$ and $\overline{k+1} = \overline{k} \to \overline{0}$.

As explained in Subsection 3.2.5 and Section 6.1 of [3], *nested sequential procedures* (or *NSPs*) are infinitary terms generated by the following grammar, construed coinductively:

$$\begin{array}{rrcl} \textit{Procedures:} & p, q & ::= & \lambda x_0 \ldots x_{r-1}.\, e \\ \textit{Expressions:} & d, e & ::= & \bot \mid n \mid \mathtt{case}\ a\ \mathtt{of}\ (i \Rightarrow e_i \mid i \in \mathbb{N}) \\ \textit{Applications:} & a & ::= & x\, q_0 \ldots q_{r-1} \end{array}$$

We shall often use vector notation $\vec{x}, \vec{q}$ for sequences $x_0 \ldots x_{r-1}$ and $q_0 \ldots q_{r-1}$. The notions of free variable and (infinitary) α-equivalence are defined in the expected way, and we shall work with terms only up to α-equivalence.

If variables are considered as annotated with types from T, there is an evident notion of a *well-typed* procedure term $p : \sigma$, where $\sigma \in \mathsf{T}$. For each type σ, we let $\mathsf{SP}(\sigma)$ be the set of well-typed procedures of type σ, and $\mathsf{SP}^0(\sigma)$ for the set of *closed* such procedures (note that $\mathsf{SP}^0(\mathtt{N}) \cong \mathbb{N}_\bot$). We will sometimes take notational liberties: e.g. for $n \in \mathbb{N}$, we will often write the procedure $\lambda.n \in \mathsf{SP}^0(\mathtt{N})$ simply as n.

It is shown in Section 6.1 of [3] how to define a (total) *application* operation $\cdot : \mathsf{SP}(\sigma \to \tau) \times \mathsf{SP}(\sigma) \to \mathsf{SP}(\tau)$ for each $\sigma, \tau \in \mathsf{T}$, thus making the sets $\mathsf{SP}(\sigma)$ into an applicative structure SP. Since closed procedures are closed under application, we likewise obtain an applicative substructure SP^0. A fundamental though non-trivial result is that SP^0 is a (typed) *$\lambda\eta$-algebra* (Theorem 6.1.18). Furthermore, it is readily shown that SP^0 admits a natural interpretation of the constants of $\mathrm{T} + \mathit{min}$ (Theorems 6.3.5 and 6.3.15), and even those of Plotkin's PCF (Example 7.1.5), thus making SP^0 an adequate model for these languages.

The focus of the present note will be a certain substructure of SP^0 representing a more restricted concept of sequential higher-order computation: the submodel consisting of *left-well-founded* procedures. Suppose that p is a sequential procedure and π is some path through its syntax tree. If $\mathtt{case}\ a\ \mathtt{of}\ (i \Rightarrow e_i)$ is any subterm appearing along π, we shall say π takes a *left branch* at this subterm if it descends into a, and a *right branch* if it descends into some e_i. We say p is *left-well-founded* (LWF) if no path π through its syntax tree involves infinitely many left branches. Equivalently, p is LWF if the tree of application subterms within p is well-founded.

It can be shown (Theorem 6.3.14) that the left-well-founded procedures of SP^0 are closed under application, and so constitute an applicative substructure $\mathsf{SP}^{0,\mathrm{lwf}}$ of SP^0. Furthermore, $\mathsf{SP}^{0,\mathrm{lwf}}$ is a sub-$\lambda\eta$-algebra of SP^0 and contains the interpretations of the constants of $\mathrm{T} + \mathit{min}$, so that the interpretation of any $\mathrm{T} + \mathit{min}$ term remains within $\mathsf{SP}^{0,\mathrm{lwf}}$. Significantly, however, the interpretations of the PCF fixed point operators Y_σ in SP^0 fall outside $\mathsf{SP}^{0,\mathrm{lwf}}$.

Our theorem will state that no LWF procedure of relevant type can play the role of a *bar recursor*. Here, the property of being a bar recursor is framed in terms of having a certain extensional behaviour when applied to 'total' arguments. In fact, since several concepts of 'totality' are possible in a higher-order setting, we shall limit our attention to a restricted class of *strongly total* arguments which can be expected to be 'total' in all reasonable senses—this will enable us to formulate our theorem in a strong form that transfers readily to many other models and many concepts of totality. We now proceed to identify a class of strongly total elements that will be convenient for our purposes.

First, we recall that SP^0 admits a natural interpretation of the language $\mathsf{Klex}^{\mathrm{prim}}$ of *Kleene primitive recursion*: in effect, the simply typed lambda calculus with constants $\widehat{0} : \mathtt{N}$, $Suc : \mathtt{N} \to \mathtt{N}$ and $Primrec : \mathtt{N} \to \mathtt{N} \to (\mathtt{N} \to \mathtt{N} \to \mathtt{N}) \to \mathtt{N}$.[2] The interpretation of these constants as NSPs is as follows:

$$\begin{array}{rcl} \widehat{0} & = & \lambda.0 \\ Suc & = & \lambda x.\ \mathtt{case}\ x\ \mathtt{of}\ (i \Rightarrow i+1) \\ Primrec_j[x,f] & = & \mathtt{case}\ x\ \mathtt{of}\ i_0 \Rightarrow \mathtt{case}\ f0i_0\ \mathtt{of}\ i_1 \Rightarrow \cdots \\ & & \qquad \mathtt{case}\ f(j-1)i_{j-1}\ \mathtt{of}\ i_j \Rightarrow i_j\ , \\ Primrec & = & \lambda xfn.\ \mathtt{case}\ n\ \mathtt{of}\ (j \Rightarrow Primrec_j[x,f])\ . \end{array}$$

It is reasonable to expect that any $\mathsf{Klex}^{\mathrm{prim}}$ definable procedure will be 'total' in all senses of interest.

Another simple example of a manifestly 'total' NSP, seemingly not definable within $\mathsf{Klex}^{\mathrm{prim}}$, is the following 'strong definition by cases' operator $Cases : \mathtt{N} \to (\mathtt{N} \to \mathtt{N}) \to (\mathtt{N} \to \mathtt{N}) \to \mathtt{N}$:[3]

$$Cases \quad = \quad \lambda xfg.\ \mathtt{case}\ x\ \mathtt{of}\ (0 \Rightarrow f0 \mid i+1 \Rightarrow g0)\ .$$

(Here, for example, $f0$ abbreviates $\mathtt{case}\ f0\ \mathtt{of}\ (i \Rightarrow i)$.) As in Subsection 6.3.4, we shall write $\mathsf{Klex}^{\mathrm{prim}+}$ for the simply typed λ-calculus with constants $\widehat{0}$, *Suc*, *Primrec* and *Cases*, and shall consider an NSP to be *strongly total* iff it is definable by a closed term of $\mathsf{Klex}^{\mathrm{prim}+}$. (We do not claim any canonical status for this definition—the choice of the language $\mathsf{Klex}^{\mathrm{prim}+}$ is purely a matter of technical convenience.)

Abstracting from this situation, we may introduce the following general definition (cf. Exercise 6.3.21):

Definition 1 *Suppose* $\mathbf{A}$ *is any simply typed* λ*-algebra over* $\mathbb{N}$ *or* $\mathbb{N}_\perp$ *equipped with a choice of elements Suc, Primrec, Cases, with types as above, such that the following hold in* $\mathbf{A}$ *for all* $n, m \in \mathbb{N}$ *and all* x, f, g *of appropriate types.*

[2] In the language $\mathsf{Klex}^{\mathrm{prim}}$ as defined in Subsection 5.1.5, *Suc* and *Primrec* are treated as built-in language constructs rather than simply as constants, but this is an inessential difference.

[3] Note that *Cases* is called D in [3] (see Exercise 6.3.21). We have not actually proved that *Cases* is not definable in $\mathsf{Klex}^{\mathrm{prim}}$—this may be an interesting question for further work.

(The equations involving $\bot$ apply only when $\mathbf{A}(\mathtt{N}) = \mathbb{N}_\bot$.)

$$\begin{array}{rcl} \mathit{Suc} \cdot n & = & n+1 \\ \mathit{Primrec} \cdot x \cdot f \cdot 0 & = & x \\ \mathit{Primrec} \cdot x \cdot f \cdot n & = & m \implies \\ \mathit{Primrec} \cdot x \cdot f \cdot n & = & \bot \implies \\ \mathit{Cases} \cdot 0 \cdot f \cdot g & = & f0 \\ \mathit{Cases} \cdot n+1 \cdot f \cdot g & = & g0 \end{array} \qquad \begin{array}{rcl} \mathit{Suc} \cdot \bot & = & \bot \\ \mathit{Primrec} \cdot x \cdot f \cdot \bot & = & \bot \\ \mathit{Primrec} \cdot x \cdot f \cdot n+1 & = & f \cdot n \cdot m \\ \mathit{Primrec} \cdot x \cdot f \cdot n+1 & = & \bot \\ \mathit{Cases} \cdot \bot \cdot f \cdot g & = & \bot \end{array}$$

Relative to this choice, an element of $\mathbf{A}$ *is* strongly total *if it is* λ*-definable from* 0*,* *Suc, Primrec, Cases.*

In SP^0 in particular, it follows easily from results of Section 6.3 in [3] that every strongly total element is well-founded and contains no occurrences of $\bot$. Moreover, the class of strongly total procedures forms a total computability model over $\mathbb{N}$. In our main proof, we shall sometimes claim that a certain procedure is strongly total, but leave it as an easy exercise to verify that this is indeed the case.

In order for our general notions of bar recursor to make sense, we shall work in the setting of a model $\mathbf{A}$ satisfying the hypotheses of Definition 1, along with the additional hypothesis that $\mathbf{A}$ is *extensional at type* $\overline{1}$: that is, if $f, g \in \mathbf{A}(1)$ and $f \cdot x = g \cdot x$ for all $x \in \mathbf{A}(0)$, then $f = g$. It is clear that our model SP^0 satisfies these conditions, as do many other models of interests such as the Scott-Ershov model PC of *partial continuous functionals* and the Kleene–Kreisel model Ct of *total continuous functionals*. Clearly, in any such model, any primitive recursive function $f : \mathbb{N} \to \mathbb{N}$ has a canonical representative $\widetilde{f} \in \mathbf{A}(1)$ such that $\widetilde{f} \cdot n = f(n)$ for each $n \in \mathbb{N}$ and (when $\mathbf{A}(0) = \mathtt{N}_\bot$) $\widetilde{f} \cdot \bot = \bot$.

A little more notation and terminology will be helpful. If $x = \langle x_0, \ldots, x_{r-1} \rangle$, where $x, x_i \in \mathbb{N}$ and $\langle \cdots \rangle$ is a coding for finite sequences of natural numbers, we shall write $|x|$ for the length of the coded sequence (namely r), and $x.z$ for $\langle x_0, \ldots, x_{r-1}, z \rangle$ where $z \in \mathbb{N}$. We also write $x_0, \ldots, x_{r-1}, j^\omega$ or $\vec{x}\, j^\omega$ for the primitive recursive function $\mathbb{N} \to \mathbb{N}$ sending i to x_i if $i < r$, and to j if $i \geq r$; we also use the same notation for the canonical representative of this function within $\mathbf{A}(1)$.

We shall in fact consider two variants of bar recursion, due respectively to Spector [4] and to Kohlenbach [2]. Both of these exploit the idea that any *continuous* function $\mathbb{N}^\mathbb{N} \to \mathbb{N}$ can be construed as representing a well-founded countably branching tree, but they differ as regards the way in which it does so:

Definition 2 *(i) Given any* $F \in \mathbf{A}(2)$*, let us say a sequence* $\vec{x} = x_0, \ldots, x_{r-1}$ *is a* Spector leaf *in* F *if* $F(\vec{x}\, 0^\omega) < |x|$*, and is a* Kohlenbach leaf *in* F *if* $F(\vec{x}\, 0^\omega) = F(\vec{x}\, 1^\omega)$*. A Spector [resp. Kohlenbach] leaf* $\vec{x}$ *is* minimal *if no proper prefix of* $\vec{x}$ *is also a Spector [resp. Kohlenbach] leaf in* F*.*

(ii) The Spector tree *of* F*, written* $Tr^S(F)$*, is the set of sequence numbers* $x = \langle \vec{x} \rangle$ *such that no proper prefix of* $\vec{x}$ *is a Spector leaf in* F*. The* Kohlenbach tree $Tr^K(F)$ *is defined analogously.*

The key property here that both $Tr^S(F)$ and $Tr^K(F)$ are *well-founded* trees if F acts continuously on functions $g : \mathbb{N} \to \mathbb{N}$. This is because if F is continuous, then for any such g there exists r such that $g(0), \ldots, g(r-1)$ is a Spector leaf in F, and likewise for Kohlenbach leaves.

We are now ready at last to present our definitions of bar recursors (cf. Section 7.3.3 of [3]):

Definition 3 *A* Spector bar recursor *(for type* $\mathtt{N}$*) is an element*

$$\mathsf{BR}^S \in \mathbf{A}(((\mathtt{N} \to \mathtt{N}) \to \mathtt{N}) \to (\mathtt{N} \to \mathtt{N}) \to (\mathtt{N} \to (\mathtt{N} \to \mathtt{N}) \to \mathtt{N}) \to (\mathtt{N} \to \mathtt{N}))$$

such that the following hold for all strongly total $F, L, G \in \mathbf{A}$ *of appropriate types and for all* $x = \langle \vec{x} \rangle \in Tr^S(F)$*:*

$$\begin{array}{lcl} \mathsf{BR}^S(F,L,G)(x) & = & L(x) \quad \textit{if } x \textit{ is a Spector leaf in } F\,, \\ \mathsf{BR}^S(F,L,G)(x) & = & G(x,\, \lambda z : \overline{0}.\, \mathsf{BR}^S(F,L,G)(x.z)) \quad \textit{otherwise}\,. \end{array}$$

The notion of a Kohlenbach bar recursor BR^K *is defined analogously, using the set* $Tr^K(F)$ *and the notion of Kohlenbach leaf.*

(By the notation $(\lambda z.\, \mathsf{BR}(F,L,G)(x.z))$, we here mean the canonical representative of this function within $\mathbf{A}(1)$. This exists because $\mathbf{A}$ is a λ-algebra and the function $(x,z) \mapsto x.z$ is primitive recursive, hence present in $\mathbf{A}$.)

The condition that x belongs to the relevant tree is not a standard part of this definition (and is not included e.g. in Section 7.3.3 of [3]). This condition makes our notions of bar recursor slightly weaker than usual (which in turn makes our main theorem slightly stronger), but its main purpose is to facilitate Proposition 4 below. Clearly, both Spector and Kohlenbach bar recursors exists in SP^0: e.g. the above recursive definition of BR^S can be readily recast as an NSP for BR^S featuring BR^S itself, and by expanding this ad infinitum we obtain an infinitely deep NSP with the required properties. (Crucially, however, these 'canonical' bar recursors are not LWF.) If BR is any Spector [resp. Kohlenbach] bar recursor in SP^0, it is easy to show by continuity of SP^0 and meta-level bar induction that for any strongly total F, L, G, x, the value of $\mathsf{BR} \cdot F \cdot G \cdot x$ is a numeral and is uniquely determined by the clauses of Definition 3.

For our main proof, we shall find it marginally more convenient to work with the Kohlenbach definition, but our theorem will transfer readily to the Spector version in view of the following relative definability result. We shall here state this just for SP^0; it will be seen that it also holds for other models with reasonable computational closure and continuity properties, although we shall not bother to formulate the relevant general conditions precisely.

Proposition 4 *If* BR^S *is any Spector bar recursor in* SP^0*, then a Kohlenbach bar recursor* BR^K *is* $\mathrm{T} + \mu$ *definable relative to* BR^S*. Hence if a left-well-founded Spector bar recursor exists in* SP^0*, so does a left-well-founded Kohlenbach bar recursor.*

PROOF We first construct a T + *min* definable element $U \in \mathsf{SP}^0(2 \to 2)$ such that for any $F \in \mathsf{SP}^0(2)$ in which the empty sequence is not a Kohlenbach leaf, the minimal Spector leaves in $U \cdot F$ are precisely the minimal Kohlenbach leaves in F. We may achieve this by defining

$$U = \lambda F.\lambda g.\ (\mathit{min}\ r.\ F(g(0), \ldots, g(r-1), 0^\omega) = F(g(0), \ldots, g(r-1), 1^\omega)) - 1\ .$$

Using this, we may define

$$\mathsf{BR}^K\ =\ \lambda FLGx.\ \mathit{if}\ F(0^\omega) = F(1^\omega)\ \mathit{then}\ L\langle\rangle\ \mathit{else}\ \mathsf{BR}^S(U(F), L, G)(x)\ .$$

It is now easy to check by bar induction on nodes in $Tr^K(F)$ that BR^K is a Kohlenbach bar recursor. □

The converse implication can also be proved similarly. In fact, a somewhat subtler argument (appearing in Kohlenbach [2]) shows that Spector bar recursion is definable relative to Kohlenbach bar recursion in System T alone (the converse to this does not hold).

One final piece of terminology is needed. To ease notation, and also to present our theorem in a slightly stronger form, we shall consider more specialized forms of bar recursion that are easily obtained from the general forms above. We here revert to the general setting of a model $\mathbf{A}$ satisfying the conditions of Definition 1 along with extensionality at type $\overline{1}$.

Definition 5 *A* restricted Spector bar recursor *is an element*

$$\mathsf{BR}^{S,r} \in \mathbf{A}(2 \to 2 \to 1)$$

such that the following hold for all strongly total $F, G \in \mathbf{A}$ of appropriate types and for all $x = \langle \vec{x} \rangle \in Tr^S(F)$:

$$\begin{array}{lcl} \mathsf{BR}^{S,r}(F,G)(x) & = & 2x+1 \quad \mathit{if}\ x\ \mathit{is\ a\ Spector\ leaf\ in}\ F\ , \\ \mathsf{BR}^{S,r}(F,G)(x) & = & G(\lambda z : \overline{0}.\, \mathsf{BR}^{S,r}(F,G)(x.z)) \quad \mathit{otherwise}\ . \end{array}$$

The notion of restricted Kohlenbach bar recursor $\mathsf{BR}^{K,r}$ *is defined analogously.*

It is easily seen that a restricted (Spector or Kohlenbach) bar recursor is primitive recursively definable from an ordinary one by specializing the leaf function L to $\lambda x.2x+1$ and by eschewing the dependence of G on an argument x. The proof of Proposition 4 clearly also yields the following:

Proposition 6 *If $\mathsf{BR}^{S,r}$ is a restricted Spector bar recursor in SP^0, then a restricted Kohlenbach bar recursor $\mathsf{BR}^{K,r}$ is* T + *min definable relative to $\mathsf{BR}^{S,r}$. Hence if a left-well-founded restricted Spector bar recursor exists in SP^0, so does a left-well-founded restricted Kohlenbach bar recursor.* □

2 The theorem

We are now ready to state our main theorem:

Theorem 7 Within SP^0, a restricted Kohlenbach bar recursor cannot be LWF, and hence cannot be T + *min* definable.

The corresponding statement for restricted Spector bar recursion (which appears as Theorem 6.3.28 in [3]) follows immediately by Proposition 6. It also follows a fortiori that no ordinary Spector or Kohlenbach bar recursor in the sense of Definition 3 can be LWF.

The proof of the Theorem follows a general method introduced in [3], where it is used to show that the System T recursor rec_1 is not definable in $\mathsf{Klex}^{\min}$, the language of μ-recursion (Theorem 6.3.27). The proof of that result is already complex, and that of the present Theorem even more so. The reader is therefore strongly urged to study the proof of Theorem 6.3.27 in detail before tackling the present one.

We start with an informal outline of our argument. Suppose that B is any genuine restricted Kohlenbach bar recursor in SP^0. Suppose also, for contradiction, that B' is a *left-well-founded* restricted bar recursor in SP^0; our task is to construct particular strongly total F, G such that $B' \cdot F \cdot G \cdot \langle\rangle \neq B \cdot F \cdot G \cdot \langle\rangle$. We shall write $P \in \mathsf{SP}^0(2, 2 \to 0)$ for the procedure $\lambda F^2 G^2 . B' F G \langle\rangle$; clearly P is LWF.

As far as G is concerned, our strategy is to begin by analysing in detail the computations arising when P and various subterms thereof are applied to $G_0 = \lambda g^1 . 2g(0)$; let the result of applying P to G_0 be c. We then concoct some strongly total G_1 which is sufficiently close to G_0 that all these computations proceed in the same way when G_0 is replaced by G_1, but such that the true value of B at G_1 is different from c. (In this respect our proof is similar to that of Theorem 6.3.27.)

As far as F is concerned, the idea is that since P is LWF, it will only be prepared to nest calls to G to finite depth along any given branch of the computation. We therefore wish to construct an F representing a well-founded tree that goes deeper along some branch than P is willing to explore. Rather than specifying such an F up front, however, we shall construct one by a process of successive approximation in tandem with our analysis of the computation tree for $P \cdot F \cdot G_0$; the idea is that our choice of F is based on looking at how deep P actually goes in the computation in question. (This is a new ingredient of our argument not present in the proof of Theorem 6.3.27.)

Our proof will be structured as follows. First, we let $G_0 \in \mathsf{SP}^0(2)$ be the evident procedure for $\lambda g . 2g(0)$, and analyse the computation of $P \cdot F \cdot G_0$ for various F. At 'top level', this will consist of a finite sequence of calls to F or G_0 followed by the return of a final result; but we will go on to analyse the type 1 procedures that are passed to F and G_0, at least for numerical arguments below a certain bound. These subsidiary computations will themselves consist of a finite sequence of calls to F or G_0, so we can recursively apply a similar

analysis to these calls. The hypothesis that P is LWF will ensure that this whole process terminates at some finite depth.

At the end of this analysis, we are left with two things. First, in the course of the analysis, the value of F we are considering will have been refined via an approximating process, and at the end we are able to fix on the definitive value F_∞ that we shall use to obtain a contradiction. Second, our analysis generates a set of sufficient conditions on a procedure G (all satisfied by G_0) which guarantee that the computation of $P \cdot F_\infty \cdot G$ will proceed exactly as for $P \cdot F_\infty \cdot G_0$, and will yield the same result. These sufficient conditions may be encapsulated by a certain *neighbourhood* $\mathcal{G} \subseteq \mathsf{SP}^0(2)$, consisting of procedures sufficiently close to G_0 that the same computation works.

To complete the proof, we then construct a certain procedure G_1 (again drawing on the above analysis), and show that $G_1 \in \mathcal{G}$ (so that $P \cdot F_\infty \cdot G_1 = P \cdot F_\infty \cdot G_0$), but that $B \cdot F_\infty \cdot G_1 \cdot \langle\rangle \neq P \cdot F_\infty \cdot G_1$. We thus conclude that B' is not a restricted bar recursor after all.

We now proceed to the more formal outworking of this method. We shall write $Tr^K(F)$ simply as $Tr(F)$.

Computation analysis: the top level

To begin, we define strongly total procedures

$$\begin{aligned} G_0 &= \lambda g.\, \mathtt{case}\ g(0)\ \mathtt{of}\ (i \Rightarrow 2i)\ , \\ F_0^+ &= \lambda f.\, \mathtt{case}\ f(0)\ \mathtt{of}\ (i \Rightarrow \langle i \rangle)\ . \end{aligned}$$

Note that $\langle\rangle$ is not a leaf in $Tr(F_0^+)$, but that each $\langle x_0 \rangle$ is a leaf. Suppose that $P \cdot F_0^+ \cdot G_0 = c \in \mathbb{N}$.[4] By continuity of application, we may pick $k^0 \in \mathbb{N}$ large enough that $P \cdot F_0 \cdot G_0 = c$, where

$$F_0 \ = \ \lambda f.\, \mathtt{case}\ f(0)\ \mathtt{of}\ (i < k^0 \Rightarrow \langle i \rangle \mid i \geq k^0 \Rightarrow \bot)$$

(extending our notation for `case` expressions in an obvious way). Note that $F_0 \sqsubseteq F_0^+$. We shall use F_0 as the first step in our approximative construction of a suitable F.

Let us now look at the computation of $P \cdot F_0 \cdot G_0 = c$. At 'top level', this will consist of a finite sequence of calls to F_0 or G_0 (in any order), corresponding to a rightward path through the syntax tree of P leading to the result c (recall that P has the form $\lambda FG. \cdots$). For example, such a path might have the form

$$\lambda FG.\, \mathtt{case}\ F(f_0^0)\ \mathtt{of}\ u_0^0 \Rightarrow \mathtt{case}\ G(g_0^0)\ \mathtt{of}\ v_0^0 \Rightarrow \mathtt{case}\ F(f_1^0)\ \mathtt{of}\ u_1^0 \Rightarrow \cdots \Rightarrow c\ .$$

Here the f_i^0 and g_i^0 are themselves type 1 procedures which appear syntactically within P and which may contain F, G as free variables (the superscript indicates that we are here analysing the computation at 'level 0'). Let $f_0^0, \ldots, f_{l^0-1}^0$ be the complete list of such procedures appearing as arguments to F along this computation path, with $u_0^0, \ldots, u_{l^0-1}^0$ the corresponding outcomes when F, G

[4] One can actually show that $P \cdot F_0^+ \cdot G_0 = 4\langle 0 \rangle + 2$, but we shall not need this fact.

are instantiated to F_0, G_0. Likewise, let $g^0_0, \ldots, g^0_{n^0-1}$ be the list of procedures appearing as arguments to G on this path, with $v^0_0, \ldots, v^0_{n^0-1}$ the corresponding outcomes. In general, if h is a procedure possibly containing F, G free, and F', G' are closed procedures of appropriate types, we shall use the notation $h[F', G']$ for the closed procedure obtained from h by instantiating F, G to F', G' and normalizing (formally, $h[F', G'] = (\lambda FG.h) \cdot F' \cdot G'$).

Of course, when $F = F_0$ and $G = G_0$, the type 1 procedures f^0_i and g^0_i will be interrogated only on the argument 0. However, in order to secure the same evaluation behaviour when we later replace G_0 by our contrary example G_1, we shall also need to analyse the behaviour of each g^0_i on arguments up to some *modulus* m^0. Choose

$$m^0 > k^0 + n^0 + 1 \; .$$

(This constraint is rather hard to motivate at this stage; the reason for it will emerge during the construction of G_1, at the point where we select a number x_0 with certain properties.)

In order to proceed further, we need to extend our approximation to F. First, extend the procedure F_0 to a strongly total F^+_1:

$$\begin{aligned} F^+_1 \;\; = \;\; \lambda f. \;\; & \texttt{case}\ f(0)\ \texttt{of}\ (i_0 < k^0 \Rightarrow \langle i_0 \rangle \mid i_0 \geq k^0 \Rightarrow \\ & \texttt{case}\ f(1)\ \texttt{of}\ (i_1 \Rightarrow \langle i_0, i_1 \rangle)) \; . \end{aligned}$$

(We leave it as an easy exercise to verify that F^+_1 is indeed strongly total.) The idea is that $\langle x_0 \rangle$ will be a leaf node in $Tr(F^+_1)$ when $x_0 < k^0$, but elsewhere $Tr(F^+_1)$ will have depth 2.

Now consider the computation of $P \cdot F^+_1 \cdot G_0$. Since $F^+_1 \sqsupseteq F_0$, this has the same shape as before and features the same type 1 procedures f^0_i and g^0_i and outcomes u^0_i, v^0_i. Furthermore, we claim that $g^0_i[F^+_1, G_0] \cdot z$ yields some natural number r^0_{iz}. To show this, suppose for contradiction that $g^0_i[F^+_1, G_0](z) = \bot$ for some i, z, and let $G'_0 = \lambda g.\ \texttt{case}\ g(z)\ \texttt{of}\ (j \Rightarrow G_0(g))$. Clearly G'_0 is strongly total. Also $G'_0 \preceq G_0$ in the extensional preorder on NSPs (see Exercise 6.1.21), so

$$(G(g^0_i))[F^+_1, G'_0] \;\sqsubseteq\; G'_0(g^0_i[F^+_1, G_0]) \;=\; \bot \; .$$

Moreover, for each application $F(f^0_j)$ (respectively $G(g^0_j)$) occurring before $G(g^0_i)$ in the path under consideration, we have $(F(f^0_j))[F^+_1, G'_0] \sqsubseteq u^0_j$ (respectively $(G(g^0_j))[F^+_1, G'_0] \sqsubseteq v^0_j$), whence it is easy to see that $P \cdot F^+_1 \cdot G'_0$ is undefined. This contradicts the fact that P is defined on all strongly total arguments.

By the same argument, for each $i < l^0$ and for $z = 0$, $f^0_i[F^+_1, G_0] \cdot z$ yields a natural number, which we call q^0_{iz}. (This apparently superfluous use of z is intended to mesh with the more general situation treated below.)

At this point, it is convenient to record the information gleaned so far in the form of certain *neighbourhoods* of G_0. For each $i < n^0$, define

$$\begin{aligned} V^0_i \;\; &= \;\; \{g \in \mathsf{SP}^0(1) \mid \forall z < m^0.\, g \cdot z = r^0_{iz}\} \; , \\ \mathcal{G}^0_i \;\; &= \;\; \{G \in \mathsf{SP}^0(2) \mid \forall g \in V^0_i.\, G \cdot g = v^0_i\} \; . \end{aligned}$$

Clearly $G_0 \in \mathcal{G}_i^0$ for each i, because G_0 interrogates its argument only at 0 (so in fact even the single condition $g \cdot 0 = r_{i0}^0$ suffices to guarantee that $G \cdot g = v_i^0$). This completes our analysis of the computation at top level; we shall refer to this as the *depth* 0 *analysis*.

The idea is that the sets $\mathcal{G}_i^0$ will form part of a system of neighbourhoods recording all the necessary information about G_0; we will then be free to select any G from the intersection of these neighbourhoods knowing that the computation will proceed as before. (The particular neighbourhood system we give will be carefully selected to allow for the construction of a certain G_1 with the desired properties.) As things stand, the neigbourhoods $\mathcal{G}_i^0$ do not achieve this: for an arbitrary G in all these neigbourhoods, there is no guarantee that the meaning of each $g_i^0(z)$ at F_1^+ and G will agree with its meaning at F_1^+ and G_0 — and similarly for each $f_i^0(0)$. We therefore need a deeper analysis of these subcomputations in order to nail down the precise information about G_0 that these rely on.

Computation analysis: the step case

The idea is that we now repeat our analysis for each of the computations of

$$f_i^0(z)[F_1^+, G_0] \;\; (i < l^0,\ z < 1)\,, \qquad g_i^0(z)[F_1^+, G_0] \;\; (i < n^0,\ z < m^0)\,.$$

The analysis at this stage is in fact illustrative of the general analysis at depth w, assuming we have completed the analysis at depth $w-1$. For notational simplicity, however, we shall concentrate here on the depth 1 analysis, adding a few brief remarks on the depth 2 analysis in order to clarify how the construction works in general.

First, since each of the above computations yields a numeral q_{iz}^0 or r_{iz}^0 as appropriate, we may choose k^1 such that all these computations yield the same results when F_1^+ is replaced by

$$\begin{aligned} F_1 \;\; = \;\; \lambda f.\;\; & \texttt{case}\ f(0)\ \texttt{of}\ (i_0 < k^0 \Rightarrow \langle i_0 \rangle \mid i_0 \geq k^0 \Rightarrow \\ & \texttt{case}\ f(1)\ \texttt{of}\ (i_1 < k^1 \Rightarrow \langle i_0, i_1 \rangle \mid i_1 \geq k^1 \Rightarrow \bot))\,. \end{aligned}$$

Note in passing that F_0 no longer suffices here: there will be computations of values for $g_i^0(z)$ that did not feature anywhere in the original computation of $P \cdot F_0 \cdot G_0$.

Everything we have said about the main computation and its subcomputations clearly goes through with F_1^+ replaced by F_1. So let us consider the shape of the computations of

$$f_i^0(z)[F_1, G_0] \;\; (i < l^0,\ z < 1)\,, \qquad g_i^0(z)[F_1, G_0] \;\; (i < n^0,\ z < m^0)\,.$$

At top level, each of these consists of a finite sequence of applications of F_1 and G_0 (in any order), leading to the result q_{iz}^0 or r_{iz}^0. Taking all these computations together, let $f_0^1, \ldots, f_{l^1-1}^1$ and $g_0^1, \ldots, g_{n^1-1}^1$ respectively denote the (occurrences of) type 1 procedures to which F_1 and G_0 are applied, with $u_0^1, \ldots, u_{l^1-1}^1$ and

$v_0^1, \ldots, v_{n^1-1}^1$ the corresponding outcomes. Although we will not explicitly track the fact in our notation, we should consider each of the f_j^1 and g_j^1 as a 'child' of the procedure f_i^0 or g_i^0 from which it arose. Note that if g_j^1 is a child of f_i^0 (for example), then just as Ff_i^0 appears as a subterm within the syntax tree of P, so Gg_j^1 appears as a subterm within the syntax tree of f_i^0. Thus, each of the f_j^1 and g_j^1 corresponds to a path in P with at least two left branches.

We now select a suitable modulus for our analysis of the g_i^1. Choose

$$m^1 > k^1 + n_0 + n_1 + 2 \,, \quad m^1 \geq m^0 \,.$$

(Again, the reason for this choice will emerge from the construction of G_1.) Extend F_1 to F_2^+ so that $F_2^+(f) = \langle f(0), f(1), f(2) \rangle$ when $f(0) \geq k^0$ and $f(1) \geq k^1$. Replacing F_1 by F_2^+ preserves all the structure established so far, and as before we have that $g_i^1(z)$ at F_2^+, G_0 yields a numeral r_{iz}^1 for each $i < n^1$ and $z < m^1$; similarly $f_i^1(z)$ at F_2^+, G_0 yields a numeral q_{iz}^1 for each $i < l^1$ and $z < 2$. (In fact, it is superfluous to consider $f_i^1(1)$ in cases where $f_i^1(0) < k^0$, but it simplifies notation to use 2 here as our uniform modulus of inspection for the f_i^1.) We may now augment our collection of neighbourhoods by defining

$$\begin{array}{rcl} V_i^1 & = & \{g \in \mathsf{SP}^0(1) \mid \forall z < m^1 . \, g \cdot z = r_{iz}^1\} \,, \\ \mathcal{G}_i^1 & = & \{G \in \mathsf{SP}^0(2) \mid \forall g \in V_i^1 . \, G \cdot g = v_i^1\} \,. \end{array}$$

for each $i < n^1$; note once again that $G_0 \in \mathcal{G}_i^1$. This completes our analysis of the computation at depth 1.

At the next stage, we choose k^2 so that the above all holds with F_2^+ replaced by

$$\begin{array}{rcl} F_2 & = & \lambda f. \quad \mathtt{case}\ f(0)\ \mathtt{of}\ (i_0 < k^0 \Rightarrow \langle i_0 \rangle \mid i_0 \geq k^0 \Rightarrow \\ & & \qquad \mathtt{case}\ f(1)\ \mathtt{of}\ (i_1 < k^1 \Rightarrow \langle i_0, i_1 \rangle \mid i_1 \geq k^1 \Rightarrow \\ & & \qquad\quad \mathtt{case}\ f(2)\ \mathtt{of}\ (i_2 < k^2 \Rightarrow \langle i_0, i_1, i_2 \rangle \mid i_2 \geq k^2 \Rightarrow \bot))) \,. \end{array}$$

We now repeat our analysis for each of the computations of

$$f_i^1(z)[F_2, G_0] \ \ (i < l^1, \ z < 2) \,, \qquad g_i^1(z)[F_2, G_0] \ \ (i < n^1, \ z < m^1) \,.$$

Having identified the relevant type 1 procedures $f_0^2, \ldots, f_{l^2-1}^2$ and $g_0^2, \ldots, g_{n^2-1}^2$ that feature as arguments to F and G, we pick

$$m^2 > k^2 + n^0 + n^1 + n^2 + 3 \,, \quad m^2 \geq m^1 \,,$$

and use this to define suitable sets $V_i^2, \mathcal{G}_i^2$ for $i < n^2$. By this point, it is clear how our construction may be continued to arbitrary depth.

Computation analysis: the bottom level

The crucial observation is that this entire construction eventually bottoms out. Indeed, using h as a symbol that can ambivalently mean either f or g (and

likewise H for F or G), we have that for any sequence $h^0_{i^0}, h^1_{i^1}, \ldots$ of type 1 procedures where each $h^{w+1}_{i^{w+1}}$ is a child of $h^w_{i^w}$, the syntax tree of P contains the descending sequence of subterms $H^0 h^0_{i^0}, H^1 h^1_{i^1}, \ldots$. Since P is LWF by assumption, any such sequence must eventually terminate. Moreover, the tree of all such procedures h^w_i is finitely branching, so by König's lemma it is finite altogether.

Let us see explicitly what happens at the last stage of the construction. For some depth d, we will have constructed the $f^d_i, g^d_i, u^d_i, v^d_i$ as usual, along with m^d, F^+_{d+1}, the numbers r^d_{iz}, q^d_{iz} and the neighbourhoods $\mathcal{G}^d_i$, but will then discover that $l^{d+1} = n^{d+1} = 0$: that is, none of the relevant computations of $f^d_i(z)$ or $g^d_i(z)$ (relative to F^+_{d+1} and G_0) themselves perform calls to F or G.

At this point, we may settle on F^+_{d+1} as the definitive version of F to be used in our counterexample, and henceforth call it F_∞. Explicitly:

$$\begin{aligned} F_\infty \;=\; \lambda f.\;\; & \texttt{case } f(0) \texttt{ of } (i_0 < k^0 \Rightarrow \langle i_0 \rangle \mid i_0 \geq k^0 \Rightarrow \\ & \texttt{case } f(1) \texttt{ of } (i_1 < k^1 \Rightarrow \langle i_0, i_1 \rangle \mid i_1 \geq k^1 \Rightarrow \\ & \cdots \\ & \texttt{case } f(d) \texttt{ of } (i_d < k^d \Rightarrow \langle i_0, \cdots, i_d \rangle \mid i_d \geq k^d \Rightarrow \\ & \texttt{case } f(d+1) \texttt{ of } (i_{d+1} \Rightarrow \langle i_0, \cdots, i_{d+1} \rangle)) \cdots)) \,. \end{aligned}$$

Clearly F_∞ is strongly total and $F_\infty \sqsupseteq F_w$ for $w \leq d$. Note that if $f(0) \geq k^0, \ldots, f(d) \geq k^d$ then $F_\infty \cdot f = \langle f(0), \ldots, f(d+1) \rangle$; indeed $\langle f(0), \ldots, f(d+1) \rangle$ is a minimal leaf node in $Tr(F)$. It is this portion of the tree, not visited by any of the computations described so far, that we shall exploit when we construct our counterexample G_1.

The critical neighbourhood of G_0

We may now define the *critical neighbourhood* $\mathcal{G} \subseteq \mathsf{SP}^0(2)$ by

$$\mathcal{G} \;=\; \bigcap_{w \leq d,\; i < n^w} \mathcal{G}^w_i \,.$$

Clearly $G_0 \in \mathcal{G}$. We claim that the following hold for all $G \in \mathcal{G}$ and all $w \leq d$:

1. $f^w_i(z)[F_\infty, G] = q^w_{iz}$ for all $i < l^w$ and $z \leq w$.
2. $g^w_i(z)[F_\infty, G] = r^w_{iz}$ for all $i < n^w$ and $z < m^w$.
3. $F(f^w_i)[F_\infty, G] = u^w_i$ for all $i < l^w$.
4. $G(g^w_i)[F_\infty, G] = v^w_i$ for all $i < n^w$.
5. $P \cdot F_\infty \cdot G = c$.

In other words, $\mathcal{G}$ provides a tight enough constraint to ensure that all the computations we have considered run for any $G \in \mathcal{G}$ just as they did for G_0.

We prove claims 1–4 simultaneously by downwards induction on w. For $w = d$, claims 1 and 2 hold because the computations in question make no use of F_∞ or G. For any w, claim 1 implies claim 3: F_w was chosen so that (among other things) $F_w(f_i^w[F_w, G_0]) = u_i^w$ is defined; moreover, F_w interrogates its argument only on $0, \ldots, w$ at most, so the established values of $f_i^w(z)[F_\infty, G]$ for $z \leq w$ suffice to ensure that $F_w(f_i^w[F_\infty, G]) = u_i^w$, and hence that $F_\infty(f_i^w[F_\infty, G]) = u_i^w$. Likewise, claim 2 implies claim 4, since $G \in \mathcal{G}_i^w$ by hypothesis, and the established values of g_i^w secure that $g_i^w \in V_i^w$ (at F_∞ and G).

Assuming claims 3 and 4 hold for $w+1$, it is easy to see that claims 1 and 2 hold for w: the relevant top-level computation may be reconstituted from left to right leading to the result q_{iz}^w or r_{iz}^w. Applying the same argument one last time also yields claim 5.

The counterexample G_1

It remains to construct our contrary example $G_1 \in \mathcal{G}$. The idea is that G_1 will be chosen so that according to the definition of restricted bar recursion, some 'large' value $K \neq c$ will be propagated from a leaf node at depth $d+1$ up to the surface of the computation. We work with paths beyond the horizon defined by $k^0, k^1, \ldots$ to ensure that we do not encounter a leaf prematurely, and use the moduli m^w to ensure that the type 1 functions at intermediate levels steer clear of the sets V_i^w.

Recall that B is assumed to be a genuine restricted bar recursor within SP^0, and set $B_0 = B \cdot F_\infty \cdot G_0 \in \mathsf{SP}^0(1)$. Since $G_0 = \lambda g.2g(0)$ and the leaf function has been fixed at $x \mapsto 2x+1$, we have that for any sequence number x, $B_0 \cdot x$ will take one of the values

$$2x+1\ ,\quad 2(2(x.0)+1)\ ,\quad 4(2(x.0.0)+1)\ ,\quad 8(2(x.0.0.0)+1)\ ,\quad \ldots\ ,$$

according to where a leaf for F_∞ first appears in the sequence x, $x.0$, $x.0.0$, ... In particular, for any fixed j, if we know that $|x| > j$, we can recover x_j from $B_0 \cdot x$ and even from $\theta(B_0 \cdot x)$, where $\theta(n)$ denotes the unique odd number such that $n = 2^t.\theta(n)$ for some t. We shall write $x.0^t$ for the result of appending t occurrences of 0 to the sequence number x.

We construct a finite path $x_0, x_1, \ldots, x_d$ through the tree for F_∞ in the following way, along with some associated numbers $y_0, y_1, \ldots, y_d, y_{d+1}$. Start by setting $y_0 = B_0 \cdot \langle 0 \rangle$. By the foregoing remarks, the mapping $z \mapsto \theta(B_0 \cdot \langle z, 0 \rangle)$ is injective, so by our choice of m^0 we may pick x_0 with $k^0 \leq x_0 < m^0$ such that $y_1 = B_0 \cdot \langle x_0, 0 \rangle$ differs from $g_i^0(0)$ (more precisely from r_{i0}^0) for each $i < n^0$, and also $\theta(y_1)$ differs from $\theta(y_0)$. Likewise, the mapping $z \mapsto \theta(B_0 \cdot \langle x_0, z, 0 \rangle)$ is injective, so by our choice of m^1 we may pick x_1 with $k^1 \leq x_1 < m^1$ such that $y_2 = B_0 \cdot \langle x_0, x_1, 0 \rangle$ is different from all r_{i0}^0 and $r_{i'0}^1$ where $i < n^0$, $i' < n^1$, and also $\theta(y_2)$ is different from $\theta(y_0)$ and $\theta(y_1)$. In general, we pick x_w with $k^w \leq x_w < m^w$ such that $y_{w+1} = B_0 \cdot \langle x_0, \ldots, x_w, 0 \rangle$ is different from all r_{i0}^u with $u \leq w$ and $i < n^u$, and also $\theta(y_{w+1})$ is different from $\theta(y_0), \ldots, \theta(y_w)$. Since $x_w \geq k^w$ for each $w \leq d$, we have that $\langle x_0, \ldots, x_d, 0 \rangle$ is a minimal leaf node for F_∞.

Now let K be some natural number larger than any that has featured in the construction so far, and define

$$\begin{array}{rcll} G_1 & = & \lambda g. & \texttt{case}\ g(0)\ \texttt{of}\ (\\ & & & \quad y_{d+1} \Rightarrow K \\ & & & \mid\ y_d \Rightarrow \texttt{case}\ g(x_d)\ \texttt{of}\ (K \Rightarrow K \mid j \Rightarrow 2i) \\ & & & \mid\ \cdots \\ & & & \mid\ y_1 \Rightarrow \texttt{case}\ g(x_1)\ \texttt{of}\ (K \Rightarrow K \mid j \Rightarrow 2i) \\ & & & \mid\ y_0 \Rightarrow \texttt{case}\ g(x_0)\ \texttt{of}\ (K \Rightarrow K \mid j \Rightarrow 2i) \\ & & & \mid\ i \Rightarrow 2i \\ & & &) \, . \end{array}$$

Here we understand i, j as 'pattern variables' that catch all cases not handled by the preceding clauses. In particular, the clauses $j \Rightarrow 2i$, $i \Rightarrow 2i$ mean that unless g possesses some special property explicitly handled by some other clause, we will have $G_1 \cdot g = 2(g \cdot 0) = G_0 \cdot g$. Again, we leave it as an exercise to check that G_1 is strongly total.

Properties of G_1

Let us first check that $G_1 \in \mathcal{G}$. Suppose $w \leq d$ and $i < n^w$; we will show $G_1 \in \mathcal{G}_i^w$. Consider an arbitrary $g \in V_i^w$ (note that g is not assumed to represent a total function); we want to show that $G_1 \cdot g = v_i^w$. From the definition of V_i^w we have $g \cdot 0 = r_{i0}^w$, so for $u > w$, we have $g \cdot 0 \neq y_u$ by choice of y_u. If also $g \cdot 0 \neq y_u$ for each $u \leq w$ then $G_1(g) = 2(g \cdot 0) = G_0(g) = v_i^w$ as required. If $g \cdot 0 = y_u$ for some $u \leq w$, then $G_1(g) = \texttt{case}\ g(x_u)\ \texttt{of}\ (K \Rightarrow K \mid i \Rightarrow 2(g \cdot 0))$. However, since $x_u < m^u \leq m^w$ we have $g(x_u) = r_{ix_u}^w$, and K was assumed to be larger than this, so once again $G_1(g) = 2(g \cdot 0) = v_i^w$.

We now work towards showing that $B \cdot F_\infty \cdot G_1 \cdot \langle\rangle = K$. Set $B_1 = B \cdot F_\infty \cdot G_1$, and for $w \leq d+1$, denote $\langle x_0, \ldots, x_{w-1} \rangle$ by x^w.

Claim: $B_1 \cdot (x^w.0) = y_w$ for all $w \leq d+1$.

Proof of claim: Recall that $y_w = B_0 \cdot (x^w.0)$; which has the form $2^t.s$ where $s = \theta(y_w) = B_0 \cdot (x^w.0.0^t)$ and t is minimal such that $x^w.0.0^t$ is a leaf for F_∞. Note also that if $0 < t' \leq t$ then $B_0 \cdot (x^w.0.0^{t'}) = 2^{t-t'}.s$, which is distinct from y_w (this is the point of the doubling in the definition of G_0), and also from all the other y_u since $\theta(y_0), \ldots, \theta(y_{d+1})$ are all distinct.

We may now see by reverse induction on $t' \leq t$ that $B_1 \cdot (x^w.0.0^{t'}) = 2^{t-t'}.s$. When $t' = t$, this holds because $x^w.0.0^t$ is a leaf for F_∞ so $B_1 \cdot (x^w.0.0^t) = B_0 \cdot (x^w.0.0^t)$. Assuming this holds for $t'+1$ with $t' < t$, because $x^w.0.0^{t'}$ is not a leaf we have

$$\begin{array}{rcl} B_1 \cdot (x^w.0.0^{t'}) & = & G_1 \cdot (\lambda z. B_1 \cdot (x^w.0.0^{t'}.z)) \\ & = & \texttt{case}\ B_1 \cdot (x^w.0.0^{t'}.0)\ \texttt{of}\ (\cdots \mid i \Rightarrow 2i) \\ & = & \texttt{case}\ 2^{t-(t'+1)}.s\ \texttt{of}\ (\cdots \mid i \Rightarrow 2i) \\ & = & 2^{t-t'}.s \, , \end{array}$$

using the observation that $2^{t-(t'+1)}.s$ is distinct from all of the y_u.

In particular, $B_1 \cdot (x^w.0) = 2^t.s = y_w$, so the claim is established.

Next, we show by reverse induction that $B_1 \cdot x^w = K$ for all $w \leq d+1$. For the case $w = d+1$, we have by the above claim that $B_1 \cdot (x^{d+1}.0) = y_{d+1}$, and since x^{d+1} is not a leaf for F_∞, we have

$$\begin{aligned} B_1 \cdot x^{d+1} &= G_1(\lambda z.\, B_1 \cdot (x^{d+1}.z)) \\ &= \mathtt{case}\ B_1 \cdot (x^{d+1}.0)\ \mathtt{of}\ (y_{d+1} \Rightarrow K \mid \cdots) \\ &= K\ . \end{aligned}$$

For $w < d+1$, again we have by the claim that $B_1 \cdot (x^w.0) = y_w$, and the induction hypothesis gives us $B_1 \cdot (x^w.x_w) = B_1 \cdot (x^{w+1}) = K$. Since x^w is not a leaf for F_∞, we have

$$\begin{aligned} B_1 \cdot x^w &= G_1(\lambda z.\, B_1 \cdot (x^w.z)) \\ &= \mathtt{case}\ B_1 \cdot (x^w.0)\ \mathtt{of} \\ &\qquad (\cdots \mid y_w \Rightarrow \mathtt{case}\ B_1 \cdot (x^w.x_w)\ \mathtt{of}\ (K \Rightarrow K \mid \cdots) \mid \cdots) \\ &= K\ . \end{aligned}$$

In particular, when $w = 0$ we have $B_1 \cdot \langle\rangle = B_1 \cdot x^0 = K$. Since K is assumed to be larger than c, we thus have

$$B' \cdot F_\infty \cdot G_1 \cdot \langle\rangle \;=\; P \cdot F_\infty \cdot G_1 \;=\; c \;\neq\; K \;=\; B \cdot F_\infty \cdot G_1 \cdot \langle\rangle\ .$$

Since this argument applies for B any genuine restricted bar recursor, we may conclude that B' is not a restricted bar recursor after all. This completes the proof.

3 Other type structures

As mentioned in [3] (Corollary 6.3.33), it follows readily from our main theorem that bar recursion fails to be $\mathrm{T} + \mathit{min}$ definable in many other models besides SP^0. The proof follows the same method as that of Corollary 6.3.32, but for the sake of completeness we spell it out here.

We work in the setting of a *total type structure* $\mathbf{A}$ over $\mathbb{N}$: that is, an extensional simply-typed λ-algebra $\mathbf{A}$ with $\mathbf{A}(\mathtt{N}) = \mathbb{N}$. We shall moreover assume that $\mathbf{A}$ models Kleene primitive recursion—that is, $\mathbf{A}$ contains elements *Suc* and *Primrec* satisfying the relevant equations from Definition 1. In such a model, there is a unique functional *Cases* satisfying the relevant equations, and it is definable as

$$\mathit{Cases} \;=\; \lambda x f g.\ \mathit{Primrec}\ (f0)\ (\lambda y z.g0)\ x\ .$$

This means that $\mathbf{A}$ satisfies all the relevant conditions from Section 1, so the definitions of the various notions of bar recursor make sense for $\mathbf{A}$. The leading example we have in mind is the type structure Ct of total continuous functionals;

however, other models such as the *hereditarily majorizable functionals* may also hold some interest.

As explained in Section 5.1 of [3], any model $\mathbf{A}$ satisfying these conditions admits a *partial* interpretation $[\![-]\!]^{\mathbf{A}}_{-}$ of the language $\mathsf{Klex}^{\min}$, which we here regard as the simply-typed λ-calculus with constants 0, *Suc*, *Primrec* and an additional construct

$$\frac{\Gamma \vdash M : \overline{1}}{\Gamma \vdash \mathit{Min}(M) : \overline{0}}$$

(It is important here that we do not treat *Min* simply as a constant of type $\overline{2}$, since such a constant would have no suitable interpretation in the total models of interest.) The qualifier 'partial' here means that not all $\mathsf{Klex}^{\min}$ terms need receive denotations in $\mathbf{A}$; nevertheless, we may say that an element $x \in \mathbf{A}(\sigma)$ is *μ-computable* if x arises as the denotation of some closed $\mathsf{Klex}^{\min}$ term $M : \sigma$.

In Definition 6.2.1, we also presented an interpretation of $\mathsf{Klex}^{\min}$ terms in SP. In that interpretation, variables were interpreted by themselves (more precisely, x was interpreted by the procedure x^η). For our present purposes, we adjust this definition to yield an interpretation $[\![-]\!]^{\mathsf{SP}}_{-}$ in which variables might also be interpreted via a valuation ν in SP^0. Specifically, the clauses involving variables in the definition of $[\![-]\!]^{\mathsf{SP}}_{-}$ are as follows:

- If $x \in \mathrm{dom}\ \nu$, then $[\![x]\!]^{\mathsf{SP}}_{\nu} = \nu(x)$; otherwise $[\![x]\!]^{\mathsf{SP}}_{\nu} = x^\eta$.
- $[\![\lambda x.M]\!]^{\mathsf{SP}}_{\nu} = \lambda x.[\![M]\!]^{\mathsf{SP}}_{\nu}$ (assuming by renaming that $x \notin \mathrm{dom}\ \nu$).

From the compositionality of this interpretation, it is clear that $[\![\lambda x.M]\!]^{\mathsf{SP}}_{\nu} \cdot p = [\![M]\!]^{\mathsf{SP}}_{\nu, x \mapsto p}$.

With all this in mind, we may frame the following consequence of our main theorem:

Corollary 8 *Within any total type structure $\mathbf{A}$ satisfying the conditions above, no restricted Spector or Kohlenbach bar recursor (and hence no ordinary one either) can be μ-computable, or even μ-computable relative to the System T recursors if these exist in $\mathbf{A}$.*

Proof We show this for restricted Kohlenbach bar recursors, the argument being precisely analogous for the others. We proceed by setting up a logical relation Z between SP^0 and $\mathbf{A}$: at type $\mathtt{N}$, we simply take $Z(n, n)$ for all $n \in \mathbb{N}$, and we lift this to higher types in the standard way. The following facts about Z may be easily checked with reference to the appropriate definitions from [3]:

1. Any System T recursors that exist in $\mathbf{A}$ are related via Z to the standard System T recursors in SP^0.
2. If $\Gamma \vdash M : \sigma$ is a $\mathsf{Klex}^{\min}$ term, ν, ν' are valuations of Γ in $\mathsf{SP}^0, \mathbf{A}$ respectively with $Z(\nu, \nu')$ elementwise, and $[\![M]\!]^{\mathbf{A}}_{\nu'}$ is defined, then $[\![M]\!]^{\mathsf{SP}}_{\nu}$, $[\![M]\!]^{\mathbf{A}}_{\nu'}$ are Z-related. (This is similar to the logical relations lemma, but the proof requires a separate induction case for the *Min* construct.)

3. If $\mathbb{N}$ is a closed λ-term possibly involving 0, *Suc*, *Primrec*, *Cases*, then $[\![N]\!]^{\mathsf{SP}}$, $[\![N]\!]^{\mathbf{A}}$ are Z-related. Hence every strongly total element in SP^0 is Z-related to a strongly total element in $\mathbf{A}$ and vice versa.

4. For any $\vec{x}$ and j, the elements $\vec{x}\, j^\omega$ in $\mathsf{SP}^0(1)$ and $\mathbf{A}(1)$ are Z-related.

5. If $Z(F, F')$ where $F \in \mathsf{SP}^0(2)$ and $F' \in \mathbf{A}(2)$, then F, F' have the same Kohlenbach leaves.

6. The canonical representatives of the operation $(x, z) \mapsto x.z$ in SP^0 and $\mathbf{A}$ are Z-related.

Now suppose for contradiction that a restricted Kohlenbach bar recursor B' is μ-computable, possibly relative to some System T recursors. Then there exist a $\mathsf{Klex}^{\min}$ term M and a (possibly empty) valuation ν' of certain variables as System T recursors such that $[\![M]\!]^{\mathbf{A}}_{\nu'} = B'$. Writing ν for the valuation of the same variables as the standard System T recursors in SP^0, and taking $B = [\![M]\!]^{\mathsf{SP}}_{\nu}$, we have by facts 1 and 2 above that $Z(B, B')$. We claim that B satisfies the axioms for a restricted Kohlenbach bar recursor, i.e. for all strongly total F, G we have:

- $B \cdot F \cdot G \cdot x = 2x + 1$ if x is a Kohlenbach leaf,
- $B \cdot F \cdot G \cdot x = G \cdot (\lambda z.\, B \cdot F \cdot G \cdot (x.z))$ otherwise.

But these follow readily from the corresponding properties for B' in the light of facts 3–6 above.

On the other hand, it is easy to see that $B = [\![M]\!]^{\mathsf{SP}}_{\nu}$ is LWF: if $\vec{x}$ are the variables of ν and $\vec{R}$ the corresponding System T recursors in SP^0, then $B = [\![\lambda\vec{x}.M]\!]^{\mathsf{SP}} \cdot \vec{R}$, where $[\![\lambda\vec{x}.M]\!]$ is left-bounded (and hence LWF) by Theorem 6.3.18, and the $\vec{R}$ are of course LWF; hence B is LWF by Theorem 6.3.14 We thus have a contradiction with Theorem 7, and the proof is complete. □

In a similar vein, one may readily show that in type structures over $\mathbb{N}_\perp$ such as the model PC of partial continuous functionals, no bar recursor can be T + *min* definable; we leave the outworking of the details to the interested reader.

References

[1] Bergstra, J.: Continuity and Computability in Finite Types. PhD thesis, University of Utrecht (1976)

[2] Kohlenbach, U.: Theory of Majorizable and Continuous Functionals and their Use for the Extraction of Bounds from Non-Constructive Proofs: Effective Moduli of Uniqueness for Best Approximations from Ineffective Proofs of Uniqueness. PhD thesis, Frankfurt (1990)

[3] Longley, J. and Normann, D.: Higher-Order Computability. To appear in 'Computability in Europe' series, Springer (2015)

[4] Spector, C.: Provably recursive functionals of analysis: a consistency proof of analysis by an extension of principles formulated in current intionistic mathematics. In: Dekker, J. (ed.) Recursive Function Theory, Proceedings of Symposia in Pure Mathematics, Volume 5, pp. 1-27. AMS, Providence (1962)