Theoretical Computer Science 2 (1976) 133–145.

COMPUTABILITY CONCEPTS FOR PROGRAMMING LANGUAGE SEMANTICS

Herbert EGLI

Forschungsinstitut für Mathematik, ETH-Z, 8006 Zürich, Switzerland

Robert L. CONSTABLE

Department of Computer Science, Cornell University, Ithaca, N.Y. 14850, USA

Communicated by E. Engeler
Received February 1975

Abstract. This paper is about mathematical problems in programming language semantics and their influence on recursive function theory. We define a notion of computability on continuous higher types (for all types) and show its equivalence to effective operators. This result shows that our computable operators can model mathematically (i.e. extensionally) everything that can be done in an operational semantics. These new recursion theoretic concepts which are appropriate to semantics also allow us to construct Scott models for the λ-calculus which contain all and only computably elements. Depending on the choice of the initial cpo, our general theory yields a theory for either strictly determinate or else arbitrary non-deterministic objects (parallelism). The formal theory is developed in Part II of this paper. Part I gives motivation and comparision with related work.

Part I

I.1. Introduction

We develop a theory of computable objects which is appropriate for the description of the semantics of programming languages. New recursion theoretic concepts are introduced. For instance, we no longer use the convention that a partial recursive function is undefined if one of its arguments is. This generalization is imposed by our wish to model extensionally all functions that can be defined computably (in other words, we investigate "mathematical semantics" vs "operational semantics"; see [14, 4]). In fact, in computing — as opposed to pure mathematical considerations (classical recursion theory) — we have to take into account that arguments of functions are not always given by canonical (e.g. decimal) representation but can be given by an arithmetic expression or even a whole program. Since an operational semantics works on the representation of the argument rather than on its value, the evaluation procedure may return a result even if the argument did not evaluate to a value. For an extensional description of the semantics, therefore, we are led to consider nonstrict functions $(\mathbf{N} \cup \{\perp\})^n \to \mathbf{N} \cup \{\perp\}$, where $\perp$ stands for undefined. With the partial ordering defined by $\perp \sqsubseteq n$ for all $n \in \mathbf{N}$, the set $\mathbf{N} \cup \{\perp\}$ becomes a

cpo (complete partially ordered set) $\mathbf{D}$ (see Section II.1). It is easy to see (and will follow from our general theorem) that functions arising in computing are modelled by *continuous* functions $\mathbf{D}^n \to \mathbf{D}$. But mathematical semantics not only asks for these functions which have simple types $0 \times \ldots \times 0 \to 0$. It also asks for functions of arbitrary types τ (even for infinite type; see [14, 4], also Section II.5). It seems natural then to consider the hereditarily continuous functions of finite types (for instance, this guarantees that we obtain a fixed point operator for all types). But there arise two questions.

(1) What are the "computable" elements among all continuous functions of type τ?

(2) Can we express everything that we need in semantics by such continuous computable operators?

We investigate these questions and give precise answers in this paper.

I.2. The basic ideas

We consider objects of all finite types. There is one type for individuals, denoted by 0. If τ and σ are types, then $\tau \to \sigma$ is the type of a function from type τ to type σ objects.

Analogously to type $0 \to 0$, we have two possibilities to represent an argument for a higher type function. We can either give (an index of) a program of appropriate type or else an extensional representation of the argument. We investigate both approaches and define

Computable operators (of type τ): They take arguments of appropriate type in extensional representation, i.e. arguments are represented by their "graph" which is appropriate for mathematical semantics. Since the "graph" of such an object is infinite in general, we approximate it by finite pieces (basis elements) and require the function to be continuous with respect to this approximation. The computable objects then are those represented by (an index of) an r.e. set which enumerates the desired basis elements.

Effective operators (of type τ): They work directly on programs, i.e. they are given by partial recursive functions which have the same values on equivalent indices. Effective operators describe operational semantics.

Comparing the two notions of higher type operators, we observe:

(a) Computable operators seem to be more restricted because they are restricted to *continuous arguments.*

(b) Effective operators seem to be more restricted because they take only other *effective operators as arguments* (computable operators have $2^{\aleph_0}$-many arguments, i.e. also non-computable ones).

However, we will show (main theorem, Section II.4) that the two notions are in fact equivalent. This theorem has been proved by Myhill and Shepherdson for *determinate functionals* (Rogers calls them *recursive functionals* [12]). Harrison [6]

gave a proof for all *functionals* (types $n+1 : n \to 0$) on *total inputs* (thus generalizing the theorem of Kreisel et al. [9] for type $(0 \to 0) \to 0$ on total inputs). Our theorem now applies to *all types* on *partial inputs*. It also applies to *non-determinate operators* (which include the *partial recursive operators and functionals* in Rogers' terminology) as we discuss next.

I.3. Non-determinate objects

We can introduce a second symbol for undefined, $\top$ (= overdefined) and look at continuous functionals, operators etc. over $\hat{\mathbf{D}} := \{\bot\} \cup \mathbf{N} \cup \{\top\}$, a cpo with $\bot \sqsubseteq n \sqsubseteq \top$ for all $n \in \mathbf{N}$. We call those *non-determinate operators*. Rogers' partial recursive functionals and operators can be modelled by such continuous functions. But we get more than those because it becomes possible now to consider "*partially overdefined*" objects. An example of such an operator which is not allowed in Rogers' theory is $\lambda\varphi \cdot \lambda x \cdot F_{r(x)}(\varphi)$, where r is any recursive function and F_j is the non-determinate functional with index j. More generally, this means that the category of non-determinate operators is cartesian closed, a property that is clearly most desirable (even necessary for a general theory of computability).

Using non-determinate partial computable functions, we can define non-determinate effective operators and show their equivalence to non-determinate computable operators.

The uniform treatment of all types makes the analogies between objects of different types transparent and explains the seemable fundamental differences between function theory and higher type theory that Rogers points out (for instance, the fact that his recursive operators are recursively enumerable). From our point of view, it is simply an unfortunate choice of words. His recursive functionals actually correspond to partial recursive functions and his partial recursive functionals correspond to non-determinate functions, a notion which is usually not introduced in recursion theory. However, it is useful to consider the notion of *non-determinate operators* (which leads to our "*theory over* $\hat{\mathbf{D}}$") because

(a) they arise in a general formalism with non-deterministic constructs, and determinacy in a reasonably rich non-deterministic language is undecidable;

(b) already at type 2, non-determinate operators are more powerful than determinate ones (there is a function f which is the image of a partial function φ under a non-determinate operator but no determinate operator can establish this correspondence).

On the other hand, we might want to have a separate theory for *determinate objects* (*theory over* $\mathbf{D}$) because

(a) partial recursive functions that are usually considered are determinate type-1-operators;

(b) although the determinate objects are a subclass of the objects over $\hat{\mathbf{D}}$, we can not single them out because their set of indices is not r.e.

Our theory captures both notions. It is based on a cpo D with additional properties common to both $\mathbf{D}$ and $\hat{\mathbf{D}}$ (D is required to have a recursive basis, see Section II.2). This gives us at the same time a theory over $\mathbf{D}$ and a theory over $\hat{\mathbf{D}}$. It should be noted at this point that it is an essential feature of our developments that we are working in the category of cpo's and not lattices, which would be too rich a structure for the theory of determinate objects.

Finally, let us mention that our theory, as it stands, is applicable also to the cpo $\tilde{\mathbf{D}} := 2^{\mathbf{N}}$ (theory over $\tilde{\mathbf{D}}$). In this setting, different output values are not collapsed into a single element "overdefined" but they are retained. The typed operators over $\tilde{\mathbf{D}}$ are equivalent to the "typed functions" in Scott's model $P\omega$ (see [15]).

Part II

II.1. Preliminaries for the general theory

We first recall some basic notions that we are going to use (types, cpo's, continuous functions, etc.). A more detailed presentation can be found for instance in [3] or [10].

(1) The *types* that we are going to use are defined by:

(a) 0 is a type (it is the only basic type),

(b) if τ, σ are types then $(\tau \to \sigma)$ is a type,

(c) there are no other types.

We have selected the *integer types* to be the subcollection of types defined inductively by $n+1 := n \to n$ (and not $n+1 = n \to 0$ which are the functional types used sometimes in the literature [8]). We abbreviate type $(\tau \to (\sigma \to \rho))$ by $\tau \to \sigma \to \rho$.

(2) *Complete partially ordered sets* and their use in programming language semantics (fixed point semantics) have been discussed in [11, 4]. We review only the basic definitions and properties that we need in this paper.

(a) A *cpo* (complete partially ordered set) is a partially ordered set with the properties that:

(i) each ascending chain has a least upper bound,

(ii) there is a least element, denoted by $\perp$ (bottom).

(b) The appropriate functions between cpo's D and D' are the functions that respect least upper bounds of (nonempty) chains. They are called *continuous functions*. The set of all continuous functions from D to D' is itself a cpo under the pointwise induced partial ordering and is denoted by $[D, D']$. Given a cpo D^0, we can define for each type $\tau \to \sigma$ a cpo $D^{\tau \to \sigma}$ inductively by $D^{\tau \to \sigma} := [D^\tau, D^\sigma]$.

(c) The product of any number of cpo's is obtained by taking the cartesian product of the underlying sets with the component-wise induced partial ordering.

(d) Continuous functions with several arguments can be viewed as functions on iterated function spaces since there is a continuous natural isomorphism $[D \times D', D''] \cong [D, [D', D'']]$ for any cpo's D, D' and D''. We will therefore allow writing $\alpha(x_1, \ldots, x_n)$ instead of $\alpha(x_1)(x_2)\ldots(x_n)$, but we do not introduce explicitly a "many-argument type" since it is easier for the formal treatment to have only one-argument functions to consider.

(3) All our notions of computable objects will be *based on an indexing of r.e. sets*, $\{W_i\}$. We do not specify how this indexing is obtained. However, we require that a canonical procedure for enumerating each W_i be given.

II.2. Computable operators

We said earlier that we wanted to approximate a computable operator by "finite pieces of its graph" (basis elements). The properties that we require below for a recursive basis B of a cpo D are such that

(a) B is recursively enumerable and generates D,

(b) we can compute with elements represented by an r.e. sequence of basis elements, and

(c) the space of continuous functions between two cpo's with recursive bases has itself a recursive basis.

Given a cpo D we define:

(a) a subset $X \subseteq D$ is called *compatible* (what we actually mean is that the elements of X are compatible) if X has an upper bound in D;

(b) a subset $B \subseteq D$ is called a *recursive basis* of D if

(i) B is an r.e. subset of D $(\beta_0 = \bot, \beta_1, \beta_2, \ldots)$,

(ii) each element $x \in D$ is the lub of a chain of basis elements,

(iii) for all chains $\{x_i\} \subseteq D$ and all $j \in \mathbf{N}$, $\sqcup x_i \sqsupseteq \beta_j \Rightarrow$ there is an i s.t. $x_i \sqsupseteq \beta_j$,

(iv) it is decidable whether for $i_1, \ldots, i_r \in \mathbf{N}$,

$\beta_{i_1} \sqsubseteq \beta_{i_2}$,

$\{\beta_{i_1}, \ldots, \beta_{i_r}\}$ is compatible,

(v) for all $i_1, \ldots, i_r \in \mathbf{N}$,

$\{\beta_{i_1}, \ldots, \beta_{i_r}\}$ is compatible $\Leftrightarrow$

$\sqcup\{\beta_{i_1}, \ldots, \beta_{i_r}\} \in B$ exists effectively.

Remarks. (i) If a cpo has a recursive basis, then it is uniquely determined (up to its enumeration).

(ii) The cpo's $\mathbf{D}$, $\hat{\mathbf{D}}$ and $\tilde{\mathbf{D}}$ (see Part I) have recursive bases.

From a recursive basis B^0 of a cpo D^0 we can construct a recursive basis B^τ of D^τ for all types τ. This follows from:

Theorem. *Let D and D' be cpo's with recursive bases B and B'. Then we can construct a recursive basis $\bar{B}$ for $\bar{D} := [D, D']$.*

Proof. We think of basis elements as generalized finite functions. They can be described by a finite collection of functions which "only have a value on one argument", let us say on $b \in B$ we want the value $b' \in B'$. However, in order to make such a simple function continuous we have to require at least that for all $x \sqsupseteq b$ we get also the value b'.

Formally now, we define

$$(b, b') := \lambda x \in D.\ \textit{If } x \sqsupseteq b \textit{ then } b' \textit{ else } \perp',$$

and prove some properties about this definition.

Lemma 1. *For all $b \in B$, $b' \in B'$, (b, b') is an element of $\bar{D}$.*

Lemma 2. *Each monotonic function $m : B \to D'$ extends uniquely to a continuous function $\Phi : D \to D'$.*

Let $\bar{b}_i := (b_i, b'_i)$ for $i = 1, \ldots, n$.

Lemma 3. *The following statements are equivalent:*

(a) $\{\bar{b}_1, \ldots, \bar{b}_n\}$ *is compatible,*

(b) *for all* $i_1, \ldots, i_r$, $\{b_{i_1}, \ldots, b_{i_r}\}$ *is compatible* $\Rightarrow$ $\{b'_{i_1}, \ldots, b'_{i_r}\}$ *is compatible,*

(c) $\sqcup\{\bar{b}_1, \ldots, \bar{b}_n\}$ *exists.*

We leave the proofs of these lemmas to the reader. A recursive basis $\bar{B}$ for $\bar{D}$ is now obtained by enumerating all $\sqcup\{(\beta_{i_1}, \beta'_{j_1}), \ldots, (\beta_{i_n}, \beta'_{j_n})\}$ that exist (whether such a lub exists or not is decidable by Lemma 3; the axioms of a recursive basis are easily verified).

Starting with a cpo D^0 which has a recursive basis, we thus have a recursive basis for each D^τ. The computable elements of D^τ are going to be those which are represented by an r.e. set of basis elements, i.e. those which are the lub of an r.e. set of basis elements. If we think of D^0 as being $\hat{\mathbf{D}}$ or $\tilde{\mathbf{D}}$ then each such set has a lub. However, if D^0 is $\mathbf{D}$, this is not true. So for the general theory we have to single-value the W_j's with respect to the enumeration of the basis B^τ. The single-valuing procedure gives us for each W_j an r.e. set $W_{s_\tau(j)}$ which we will write as U_j^τ.

It is defined by:

(1) enumerate $W_j : x_1, x_2, \ldots;$

(2) U_j^τ: $y_1, y_2, \ldots$ is obtained by

(a) $y_1 = x_1$,

(b)

$$y_{n+1} = \begin{cases} x_{n+1} & \text{if } \{\beta^\tau_{y_1}, \ldots, \beta^\tau_{y_n}, \beta^\tau_{x_{n+1}}\} \text{ is compatible,} \\ y_n & \text{otherwise.} \end{cases}$$

U_j^τ is clearly r.e. and $\{\beta_k^\tau \mid k \in U_j^\tau\}$ is compatible (because each finite subset is compatible). If $\{\beta_k \mid k \in W_j\}$ happens to be compatible, then $U_j^\tau = W_j$.

We now can define the operator with Gödel number j of type τ, Φ_j^τ, by

$$\Phi_j^\tau := \sqcup\{\beta_k^\tau \mid k \in U_j^\tau\}.$$

We call those the *computable operators of type* τ,

$$\mathrm{CD}^\tau := \{\Phi_j^\tau \mid j \in \mathbf{N}\}.$$

For computing with such an operator $\Phi_j^{\tau\to\sigma}$, we first have to decide on how we want to represent an input of type τ. If the argument is a Φ_i^τ then we know that the index i defines the single-valued r.e. set U_i^τ which defines Φ_i^τ by $\Phi_i^\tau := \sqcup\{\beta_k^\tau \mid k \in U_i^\tau\}$. But now remember that the basis B^τ generates D^τ, thus all elements of D^τ are of the form $\Sigma^\tau = \sqcup\{\beta_k^\tau \mid k \in S\}$, where $S \subseteq \mathbf{N}$ is some (singe-valued) subset of $\mathbf{N}$.

Equally, for each j, the operator $\Omega^\sigma := \Phi_j^{\tau\to\sigma}(\Sigma^\tau)$ is of the form $\Omega^\sigma = \sqcup\{\beta_k^\sigma \mid k \in O\}$ for some $O \subseteq \mathbf{N}$. So the question is: how do we "compute O from S"? We use the following model (which is used also by Rogers to define enumeration operators). We think of S as being presented element by element. Then from each finite piece of S we obtain a finite piece of the "output set" O as follows. Suppose that at time n we know $\{s_1, \ldots, s_m\} \subseteq S$ and $\{k_1, \ldots, k_r\} \subseteq U_j^{\tau\to\sigma}$. We then add to the output set an index l with the property

$$\beta_l^\sigma = (\sqcup\{\beta_{k_1}^{\tau\to\sigma}, \ldots, \beta_{k_r}^{\tau\to\sigma}\})(\sqcup\{\beta_{s_1}^\tau, \ldots, \beta_{s_m}^\tau\}).$$

Remarks. (i) The output set O obviously depends on the order in which S is presented. However, the operator described by O depends only on the operator described by S. If we cared for a canonical representation of all elements, we could easily represent an element Σ^τ by its "full graph", the set $\{k \mid \beta_k^\tau \sqsubseteq \Sigma^\tau\}$, and modify the above procedure so that it produces the full graph of the result.

(ii) If the input set S is an r.e. set, then we can find effectively from its index i an index k of the output set (which is thus r.e. also). This can be done for all types. In addition, if the result is of type O, we can compute more than just the index of a type O program. We can "partially compute" the element of D^0 that is denoted by the program. Computation in a cpo D^0 means that we compute better and better approximations. In general, such computations are infinite. If $D^0 = \mathbf{D}_\perp$ then this notion of computability corresponds exactly to the usual notion of partial computability.

II.3. Effective operators

The idea of effective operators is to let higher type objects be induced by partial computable functions on the indices (programs) of the arguments. We define such partial computable functions with values in a cpo D^0 first. Again, we use the

indexing $\{W_i\}$ of r.e. sets that we used before. In addition, we assume that a pairing function $\langle \cdot, \cdot \rangle : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ is given and define inductively

$$\langle x_1, \ldots, x_{n+1} \rangle := \langle \langle x_1, \ldots, x_n \rangle, x_{n+1} \rangle.$$

For computable operators of type τ we had to single-value the W_i's with respect to the basis B^τ of D^τ; here we have to single-value the W_i's with respect to the basis B^0 of D^0 and $(n+1)$-tuples given by the pairing function. The r.e. set obtained from single-valuing W_i in this way is denoted by V_i^n.

It is defined by: $V_i^n : y_1, y_2, \ldots$ is obtained from $W_i : x_1, x_2, \ldots$ by

(a) $y_1 = x_1$,

(b)

$$y_{m+1} = \begin{cases} y_m & \text{if } x_{m+1} = \langle p_1, \ldots, p_n, k \rangle \text{ and } \exists l \leq m,\ y_l = \langle p_1, \ldots, p_n, k' \rangle \\ & \text{with } \{\beta_k^0, \beta_{k'}^0\} \text{ not compatible,} \\ x_{m+1} & \text{otherwise.} \end{cases}$$

This single-valuing has the effect that for all $p_1, \ldots, p_n \in \mathbf{N}$,

$$\sqcup \{\beta_k^0 \mid \langle p_1, \ldots, p_n, k \rangle \in V_i^n\}$$

exists. This allows us to define $\psi_i^n : \mathbf{N}^n \to D^0$ by

$$\psi_i^n(p_1, \ldots, p_n) := \sqcup \{\beta_k^0 \mid \langle p_1, \ldots, p_n, k \rangle \in V_i^n\}.$$

Remarks. (i) If $D^0 = \mathbf{D}$, then $\{\psi_i^n\}$ is a Gödel numbering of the partial recursive functions.

(ii) The functions ψ_i^n have the s-m-n-property. All we really need are the functions s_n^1. Remember that these are recursive functions with the property that for all $j, p_0, \ldots, p_n$,

$$\psi_j^{n+1}(p_0, p_1, \ldots, p_n) = \psi^{n}_{s_n^1(j, p_0)}(p_1, \ldots, p_n).$$

For the inductive definition of the effective operators of type τ we have to know what kind of arguments this type expects. Notice that each type τ has a unique decomposition

$$\tau = \tau_1 \to \tau_2 \to \ldots \to \tau_p \to 0.$$

We call p the length of τ, $|\tau| := p$; it is the number of arguments that τ expects. In the following, τ_i for $i \leq |\tau|$ will refer always to the decomposition of τ. We can now define $E^\tau \subseteq \mathbf{N}$, *the indices of effective operators of type* τ, and $e \equiv_\tau e'$, the *equivalence relation* between indices of effective operators of type τ, inductively by:

(i) $E^0 = \mathbf{N}$; (ii) $e \equiv_0 e' \leftrightarrow \psi_e^0 = \psi_{e'}^0$.

(i) $e \in E^{\tau \to \sigma} \subseteq \mathbf{N}$ iff

(a) $\forall x \in E^\tau, S^1_{|\sigma|}(e, x) \in E^\sigma$,

(b) $\forall x, x' \in E^\tau,\ x \equiv_\tau x' \leftrightarrow S^1_{|\sigma|}(e, x) \equiv_\sigma S^1_{|\sigma|}(e, x')$.

(ii) For $e, e' \in E^{\tau\to\sigma}$,

$$e \equiv_{\tau\to\sigma} e' \leftrightarrow \forall x \in E^{\tau},\quad S^1_{|\sigma|}(e,x) \equiv_\sigma S^1_{|\sigma|}(e',x).$$

The *effective operators of type* τ, ED^τ, are obtained as follows:

(a) $ED^0 = \{\Psi^0_e \in D^0 \mid e \in E^0\}$ where $\Psi^0_e = \psi^0_e$,

(b) $ED^{\tau\to\sigma} = \{\Psi^{\tau\to\sigma}_e : ED^\tau \to ED^\sigma \mid e \in E^{\tau\to\sigma}\}$ where $\Psi^{\tau\to\sigma}_e(\Psi^\tau_x) := \Psi^\sigma_{S^1_{|\sigma|}(e,x)}$.

Remarks. (a) For $e \in E^\tau$, $x_i \in E^{\tau_i}$ $(\tau = \tau_1 \to \ldots \to \tau_n \to 0)$,

$$\Psi^\tau_e(\Psi^{\tau_1}_{x_1})\ldots(\Psi^{\tau_n}_{x_n}) = \psi^n_e(x_1,\ldots,x_n).$$

(b) For $e, e' \in E^\tau$,

$$e \equiv_\tau e' \leftrightarrow \forall x_i \in E^{\tau_i} : \psi^n_e(x_1,\ldots,x_n) = \psi^n_{e'}(x_1,\ldots,x_n).$$

II.4. Main theorem

Let D^0 be a cpo with a recursive basis B^0. The theorem below states that the systems $\{CD^\tau\}$ and $\{ED^\tau\}$ are effectively isomorphic (with respect to application).

Main theorem. *For each type* τ *there are recursive functions*

$$\begin{array}{ccc} \mathbf{N} & \xrightarrow{g_\tau} & \mathbf{N} \\ \cup & \swarrow{\scriptstyle \Psi^\tau} & \\ E & & \end{array}$$

with the properties:

(I) *For all* $e \in E^{\tau\to\sigma}$, $x \in E^\tau$, $y \in E^\sigma$,

$$\Psi^{\tau\to\sigma}_e(\Psi^\tau_x) = \Psi^\sigma_y \to \Phi^{\tau\to\sigma}_{g_{\tau\to\sigma}(e)}(\Phi^\tau_{g_\tau(x)}) = \Phi^\sigma_{g_\sigma(y)}.$$

(II) *For all* $j, i, k \in \mathbf{N}$,

$$\Phi^{\tau\to\sigma}_j(\Phi^\tau_i) = \Phi^\sigma_k \to \Psi^{\tau\to\sigma}_{h_{\tau\to\sigma}(j)}(\Psi^\tau_{h_\tau(i)}) = \Psi^\sigma_{h_\sigma(k)}.$$

(III) g_τ *and* h_τ *induce an equivalence between* ED^τ *and* CD^τ.

Proof. The proof is by induction on the structure of the types. For type 0, we have $\Phi^0_j = \Psi^0_j$, so g_0 and h_0 are chosen to be the indentities on $\mathbf{N} = E^0$. Now assume that the theorem is true for τ, σ, σ_i, where $\sigma = \sigma_1 \to \ldots \to \sigma_n \to 0$, and show that it is true for type $\tau \to \sigma$. The easy part is to show that each computable operator gives rise to an effective operator. We do this first:

(a) *Definition of* $h_{\tau\to\sigma}$. Set $h_{\tau\to\sigma}(j) :=$ index of ψ^{n+1} which satisfies

$$\psi^{n+1}(x, x_1, \ldots, x_n) := \Phi^{\tau\to\sigma}_j(\Phi^\tau_{g_\tau(x)}, \Phi^{\sigma_1}_{g_{\sigma_1}(x_1)}, \ldots, \Phi^{\sigma_n}_{g_{\sigma_n}(x_n)}).$$

and show:

(i) $h_{\tau\to\sigma}(j)\in E^{\tau\to\sigma}$,

(ii) (II) of the theorem.

These are straightforward verifications.

(b) *Definition of* $g_{\tau\to\sigma}$. For $e\in E^{\tau\to\sigma}$, $\Phi^{\tau\to\sigma}_{g_{\tau\to\sigma}(e)}$ should be equivalent to $\Psi^{\tau\to\sigma}_e$. If we apply $\Psi^{\tau\to\sigma}_e$ to effective operators corresponding to basis elements of type τ, we can determine exactly what $\Phi^{\tau\to\sigma}_{g_{\tau\to\sigma}(e)}$ should be. What is not obvious however is that $\Psi^{\tau\to\sigma}_e$ induces a *continuous* element. We first prove two lemmas which express exactly this fact.

Let $c(k)$ be a canonical index for $\{k\}$, i.e. $W_{c(k)}=\{k\}$, therefore $\Phi^\tau_{c(k)}=\beta^\tau_k$.

Lemma 1 (monotonicity). *For all* $k,j\in\mathbf{N}$, $e\in E^{\tau\to\sigma}$, $x_i\in E^{\sigma_i}$,

$$\beta^\tau_k\sqsubseteq\Phi^\tau_j\to\psi^{n+1}_e(h_\tau(c(k)),x_1,\ldots,x_n)\sqsubseteq\psi^{n+1}_e(h_\tau(j),x_1,\ldots,x_n).$$

Lemma 2 (compactness). *For all* $j,s\in\mathbf{N}$, $e\in E^{\tau\to\sigma}$, $x_i\in E^{\sigma_i}$,

$$\beta^0_s\sqsubseteq\psi^{n+1}_e(h_\tau(j),x_1,\ldots,x_n)\to[\exists k,\beta^\tau_k\sqsubseteq\Phi^\tau_j \text{ and } \beta^0_s\sqsubseteq\psi^{n+1}_e(h_\tau(c(k)),x_1,\ldots,x_n)].$$

An immediate consequence of these two lemmas is:

Corollary (continuity). *For all* $j\in\mathbf{N}$, $e\in E^{\tau\to\sigma}$, $x_i\in E^{\sigma_i}$,

$$\psi^{n+1}_e(h_\tau(j),x_1,\ldots,x_n)=\sqcup\{\psi^{n+1}_e(h_\tau(c(k)),x_1,\ldots,x_n)\mid\beta^\tau_k\sqsubseteq\Phi^\tau_j\}.$$

Proofs of the lemmas. We show that the negation of each of the lemmas allows us to solve the halting problem. Therefore the proofs are not constructively valid.

Proof of Lemma 1. The negation of Lemma 1 implies that there exists $k,j,s\in\mathbf{N}$, $e\in E^{\tau\to\sigma}$, $x_i\in E^{\sigma_i}$ such that $\beta^\tau_k\sqsubseteq\Phi^\tau_j$ and $\beta^0_s\sqsubseteq\psi^{n+1}_e(h_\tau(c(k)),x_1,\ldots,x_n)$, but $\beta^0_s\not\sqsubseteq\psi^{n+1}_e(h_\tau(j),x_1,\ldots,x_n)$. We define an r.e. set $W_{r(z)}$ for each $z\in\mathbf{N}$ by $W_{r(z)}$: output k. *If* $\varphi_z(z)\downarrow$ *then* output U^τ_j.

Properties:

(a) $U^\tau_{r(z)}=W_{r(z)}$,

(b) $\Phi^\tau_{r(z)}=\Phi^\tau_j\leftrightarrow\varphi_z(z)\downarrow$,

(c) $\Phi^\tau_{r(z)}=\beta^\tau_k\leftrightarrow\varphi_z(z)\uparrow$.

For deciding whether $\varphi_z(z)\downarrow$ we can ask simultaneously

(a) $\varphi_z(z)\downarrow$?

(b) $\beta^0_s\sqsubseteq\psi^{n+1}_e(h_\tau(r(z)),x_1,\ldots,x_n)$? $(\leftrightarrow\varphi_z(z)\uparrow)$.

Proof of Lemma 2. Assume that there exist $j,s\in\mathbf{N}$, $e\in E^{\tau\to\sigma}$, $x_i\in E^{\sigma_i}$ such that $\beta^0_s\sqsubseteq\psi^{n+1}_e(h_\tau(j),x_1,\ldots,x_n)$ and

$$[\forall k\in\mathbf{N},\beta^\tau_k\sqsubseteq\Phi_j\to\beta^0_s\not\sqsubseteq\psi^{n+1}_e(h_\tau(c(k)),x_1,\ldots,x_n)].$$

This means that no "finite segment" of Φ_j^τ is sufficient for ψ_e^{n+1}, therefore $U_j^\tau : k_1, k_2, \ldots$ must be infinite.

For each $z \in \mathbf{N}$ we define an r.e. set $W_{r(z)}$ by:

For $n \in \mathbf{N}$ *do*

If $\varphi_z(z)\not\downarrow \leqslant n$ *then* output k_n.

Remark: $\varphi_z(z)\not\downarrow \leqslant n$ means: "$\varphi_z(z)$ does not converge in at most n steps".

Properties:

(a) $U_{r(z)}^\tau = W_{r(z)}$,

(b) $\Phi_{r(z)} = \Phi_j^\tau \leftrightarrow \varphi_z(z)\uparrow$,

(c) $B^\tau \ni \Phi_{r(z)}^\tau \sqsubset \Phi_j^\tau \leftrightarrow \varphi_z(z)\downarrow$.

For deciding whether $\varphi_z(z)\downarrow$ we can ask simultaneously

(a) $\varphi_z(z)\downarrow$?

(b) $\beta_s^0 \sqsubseteq \psi_e^\tau(h_\tau(r(z)), x_1, \ldots, x_n)$? $(\leftrightarrow \varphi_z(z)\uparrow)$.

We now define $g_{\tau\to\sigma}(e)$ in the way we have already outlined briefly. For $\Phi_{g_{\tau\to\sigma}(e)}^{\tau\to\sigma}$, we enumerate all basis elements that we obtain from considering $\Psi_e^{\tau\to\sigma}$ applied to effective operators corresponding to basis elements of type τ. Formally, we define $W_{g_{\tau\to\sigma}(q)}$ for all $q \in \mathbf{N}$ by $W_{g_{\tau\to\sigma}(q)}$: enumerate all $i \in \mathbf{N}$ such that

$$\beta_i^{\tau\to\sigma} = (\beta_k^\tau, \beta_r^\sigma) \quad \text{with} \quad r \in U_{p(k)}^\sigma$$

where $p(k) = g_\sigma(s_n^1(q, h_\tau(c(k))))$.

Proposition. *For all* $e \in E^{\tau\to\sigma}$,

$$U_{g_{\tau\to\sigma}(e)}^{\tau\to\sigma} = W_{g_{\tau\to\sigma}(e)}.$$

Remark. This proposition tells us that all basis elements that we have enumerated are indeed available for $\Phi_{g_{\tau\to\sigma}(e)}^{\tau\to\sigma}$.

Proof. We have to show that each finite set of basis elements with indices from $W_{g_{\tau\to\sigma}(e)}$ is compatible. Due to Lemma 3 in II.2, this follows immediately from the following:

Lemma. *Let* $\beta_{i_j}^{\tau\to\sigma} = (\beta_{k_j}^\tau, \beta_{r_j}^\sigma)$ *with* $i_j \in W_{g_{\tau\to\sigma}(e)}$ *for* $j = 1, \ldots, m$. *Assume that* $\{\beta_{k_1}^\tau, \ldots, \beta_{k_m}^\tau\}$ *is compatible. Then* $\{\beta_{r_1}^\sigma, \ldots, \beta_{r_m}^\sigma\}$ *is compatible.*

Proof. Let k be such that $\beta_k^\tau = \sqcup\{\beta_{k_1}^\tau, \ldots, \beta_{k_m}^\tau\}$. Set $p(k_j) = g_\sigma(s_n^1(e, h_\tau(c(k_j))))$ for $j = 1, \ldots, m$. The definition of $W_{g_{\tau\to\sigma}(e)}$ tells us that $\beta_{r_j}^\sigma \sqsubseteq \Phi_{p(k_j)}^\sigma$. Lemma 1 allows us to show that $\Phi_{p(k_j)}^\sigma \sqsubseteq \Phi_{p(k)}^\sigma$. Thus $\beta_{r_j}^\sigma \sqsubseteq \Phi_{p(k)}^\sigma$ for $j = 1, \ldots, m$, i.e. $\{\beta_{r_1}^\sigma, \ldots, \beta_{r_m}^\sigma\}$ is compatible.

So now we know that for $e \in E^{\tau\to\sigma}$,

$$\Phi_{g_{\tau\to\sigma}(e)}^{\tau\to\sigma} = \sqcup\{\beta_k^{\tau\to\sigma} \mid k \in W_{g_{\tau\to\sigma}(e)}\}.$$

We have to show that $g_{\tau\to\sigma}$ satisfies (I) of the theorem which is equivalent to saying:

(*) For all $e \in E^{\tau\to\sigma}$, $x \in E^{\tau}$, $x_i \in E^{\sigma_i}$,

$$\Phi^{\tau\to\sigma}_{g_{\tau\to\sigma}(e)}(\Phi^{\tau}_{g_\tau(x)})(\vec{\Phi}) = \psi_e^{n+1}(x, x_1, \ldots, x_n),$$

where $\vec{\Phi} := (\Phi^{\sigma_1}_{g_{\sigma_1}(x_1)}, \ldots, \Phi^{\sigma_n}_{g_{\sigma_n}(x_n)})$. Since

$$\Phi^{\tau\to\sigma}_{g_{\tau\to\sigma}(e)}(\Phi^{\tau}_{g_\tau(x)}) = \bigsqcup\{\Phi^{\tau\to\sigma}_{g_{\tau\to\sigma}(e)}(\beta^\tau_k) \mid \beta^\tau_k \sqsubseteq \Phi^\tau_{g_\tau(x)}\},$$

$$\Phi^{\tau\to\sigma}_{g_{\tau\to\sigma}(e)}(\beta^\tau_k) = \bigsqcup\{\Phi^\sigma_{p(l)} \mid \beta^\tau_l \sqsubseteq \beta^\tau_k\},$$

we deduce

$$\begin{aligned}\Phi^{\tau\to\sigma}_{g_{\tau\to\sigma}(e)}(\Phi^{\tau}_{g_\tau(x)})(\vec{\Phi}) &= \bigsqcup\{\Phi^\sigma_{p(l)}(\vec{\Phi}) \mid \beta^\tau_l \sqsubseteq \Phi^\tau_{g_\tau(x)}\} \\ &= \bigsqcup\{\psi_e^{n+1}(h_\tau(c(l)), x_1, \ldots, x_n) \mid \beta^\tau_l \sqsubseteq \Phi^\tau_{g_\tau(x)}\}.\end{aligned}$$

Our corollary finally tells us that this is equal to $\psi_e^{n+1}(x, x_1, \ldots, x_n)$ which concludes the proof of (*).

The only thing left in the proof of the theorem is (III), i.e. that $h_{\tau\to\sigma}$ and $g_{\tau\to\sigma}$ induce an equivalence between $CD^{\tau\to\sigma}$ and $ED^{\tau\to\sigma}$. This is immediate since both sets are defined with extensional equality and furthermore, inequality of two elements of $CD^{\tau\to\sigma}$ shows up on computable arguments (even on basis elements) of type τ.

II.5. Typeless operators

Part of our motivation to investigate computable operators of finite types came from an attempt to construct extensional λ-calculus models which (1) contain only computable objects, and (2) contain "all" computable objects of finite types. This is now almost trivially achieved. The set $\{\iota_n\beta^n_j \mid n, j \in \mathbf{N}\}$ is a recursive basis of the extensional λ-calculus model D over D^0 (see [13], or [3] for its definition in the category of cpo's; $\iota_\tau : D^\tau \to D$ are the embeddings of the finite type domains into D).

So we can single-value a W_j with respect to the (enumeration of the) basis B. Let us denote the resulting r.e. set by U_j. The jth typeless operator is then defined by $\Phi_j := \bigsqcup\{\beta_k \mid k \in U_j\}$ and $CD := \{\Phi_j \mid j \in \mathbf{N}\}$. Obviously, if we restrict the Φ_j's to computable arguments, the extensional equality on CD remains the same. This allows us to verify that CD is indeed an extensional λ-calculus model. That it contains all finite type computable operators is immediate from the definition. We can call it computable for several reasons. Not only can application be described by an effective function on the indices of the elements of CD, but the elements themselves can be subject to computations, for instance their 0-component in D^0 can be computed. More generally, the projections $\Pi_\tau : CD \to CD^\tau$ are all computable, so we can get a pretty good insight into the nature of these objects.

References

[1] J. M. Cadiou, Recursive definitions of partial functions and their computations, Ph.D. Thesis, AIM–163/CS–260, Dept. of Computer Science, Stanford University (1972).

[2] R. L. Constable and D. Gries. On classes of program schemata, *SIAM J. Comput.* 1 (1) (March 1972).

[3] H. Egli, An Analysis of Scott's λ-calculus models, TR 73–191, Cornell University (1973).

[4] H. Egli, Programming language semantics using extensional λ-calculus models. TR 74–206, Cornell University (1974).

[5] R. O. Gandy, Computable functionals of finite type I, in: *Sets, Models and Recursion Theory*, Crossley (ed.) (North-Holland, Amsterdam, 1967).

[6] J. Harrison, Equivalence of the effective operations and the hereditarily recursively continuous functionals, Foundations of Classical Analysis, Stanford Summer Seminar 1963, App. VC.

[7] S. C. Kleene, *Introduction to Meta-mathematics* (Van Nostrand, Princeton N.J., 1952).

[8] S. C. Kleene, Countable functionals, in: *Constructivity in Mathematics*, A. Heyting (ed.) (North-Holland, Amsterdam, 1959).

[9] G. Kreisel, D. Lacombe and J. R. Schönfield, Partial recursive functionals and effective operations, in: *Constructivity in Mathematics*, A. Heyting (ed.) (North-Holland, Amsterdam, 1959).

[10] R. Milner, Models of LCF, AIM–186/CS–332, Computer Science Department, Stanford University (1973).

[11] R. Milner, Implementation and application of Scott's logic for computable functions, *Proc. ACM Conf. on Proving Assertions about Programs*, Las Cruces, New Mexico (1972).

[12] H. Rogers, *Theory of Recursive Functions and Effective Computability* (McGraw-Hill, New York, 1967).

[13] D. Scott, Continuous lattices, *Proc. Dalhousie Conference on Toposes, Algebraic Geometry and Logic*, Springer Lecture Notes in Mathematics 274 (Springer, Berlin, 1972).

[14] D. Scott and C. Strachey, Toward a mathematical semantics for computer languages. *Proc. of a Symposium on Computer and Automata*, New York, Polytechnic Institute of Brooklyn (1971).

[15] D. Scott, Data types as lattices, Lecture Notes of the Kiel Summer Seminar (July 1974). To be published in Springer Lecture Notes.