Computation on Abstract Data Types. The Extensional Approach, with an Application to Streams

Solomon Feferman[*]

⋆⋆⋆

With profound gratitude to Stephen C. Kleene [†]

⋆⋆⋆

Abstract

In this paper we specialize the notion of abstract computational procedure previously introduced for intensionally presented structures to those which are extensionally given. This is provided by a form of generalized recursion theory which uses schemata for explicit definition, conditional definition and least fixed point (LFP) recursion in functionals of type level ≤ 2 over any appropriate structure. It is applied here to the case of potentially infinite (and more general partial) streams as an abstract data type.

1 Introduction

This paper is a continuation of the work of Feferman [5], [6] which initiated an approach through a form of *generalized recursion theory* (g.r.t.) to computation on *abstract data types* (ADTs), including intensionally presented types, in a sense explained below. In this paper, we separate out the extensional part of the theory and show how it may be applied to

[*]Research supported by the National Science Foundation.

[†]Kleene was not my mentor, official or otherwise, but through his exceptional development of our subject I learned as much from him as if he had been.

computation on streams as an ADT. One of the main new contributions here is an explanation of how this is to be done for finite "non-terminating" streams as well as infinite streams, and even more general partial ("gappy") streams.

From the logical point of view, what are called abstract data types in theoretical computer science are simply classes of structures closed under isomorphism. Among these, one is mainly interested in structures determined by categorical or relatively categorical conditions. A paradigm example is provided by the A-list structures, where A is an arbitrary set. Each such consists of two domains, A and L (the finite lists over A), and three operations, $Cons : A \times L \to L$, $Head : L—\{nil\} \to A$, $Tail : L —\{nil\} \to L$, where $Cons$ appends an element of A to the head of a list, nil is a member of L representing the empty list, and $Head$ and $Tail$ are operations inverse to $Cons$; we also assume given a test on L for equality to the nil element. The structure is characterized up to isomorphism relative to A by saying, in addition, that L is the least set containing nil and closed under the $Cons$ operation. Computations on lists can be given by abstract algorithms which do not depend on the way that the members of the underlying set A are specified or represented (be they numbers or names, if they are specified at all) nor on the way that lists over A are actually implemented. Examples of such are the operations of concatenation, reversal, length, and term (at a given position); we can also test for equality of lists and being a sublist. As a final example, we can define the Map functional which takes any $f : A \to A$ and any member ℓ of L and forms the result of replacing each term a of ℓ by $f(a)$.

The idea of A-streams is that of (possibly) infinitely proceeding sequences of elements of A. There are actually three notions to be considered here: (i) infinite streams, (ii) finite terminating streams, and (iii) finite non-terminating streams. Those under (ii) are just another form of finite lists. Those under (iii) arise naturally from both mathematical computations and physical phenomena. An example of the former is provided by the process of filtering a given infinite stream according to some decision function f, to form the substream of all elements a satisfying $f(a) = 0$, when we don't know whether there are infinitely many such terms. An example of the latter is provided by signals from some extraterrestrial source, when we don't know at any point whether or not there will be any further signals. Streams of kind (ii) can also be subsumed under these, if we reserve a special element of A as a signal for termination. Altogether, then, these are called *potentially infinite* streams. An appropriate structure for the *infinite* A-streams as an abstract data type is provided by two domains A and S and total operations $Cons: A \times S \to S$, $Head: S \to A$, and $Tail: S \to S$; in the case of potentially infinite streams $Head$ must be taken to be a partial operation satisfying some additional conditions. Note that equality of streams is not decidable in the informal sense, and no test for it is included in the basic structure. In order to characterize the infinite stream structure up to isomorphism, we add a second-order functional Sim which associates with each f from the natural numbers $\mathbb{N}$ into A an element $Sim(f)$ of S whose nth term for each n is the same as $f(n)$ (so, $Sim(f)$ *simulates* f as a stream). Without this, we could only ensure the existence of eventually constant streams. For potentially infinite streams we do the same, but now where Sim transforms any f defined up to a certain point into an element of S whose nth term is defined just in case $f(n)$ is defined, in which case they are equal. Some operations we would expect to be able to obtain as abstract algorithms on potentially infinite streams are the term at any given position (if defined), the mesh of two streams, the map functional, and the general filtering procedure.

Our general theory of computation is thus designed to apply to many-sorted functional structures

$$\mathcal{A} = (A_0, A_1, \ldots, A_n, F_0, \ldots, F_m) \tag{1}$$

where each F_j is an object of type level ≤ 2 over the A_is, i.e. where each of these is either an individual in some A_i or a partial function or partial functional of type level 2 of specified arity, and where we always take A_0 to be the Boolean set $\{tt, ff\}$. For any given signature Σ of such structures, the *abstract computational procedures* (ACPs) of signature Σ are formal objects $\mathbf{F}$ of type level ≤ 2 generated by schemata for explicit definition, conditional definition and least fixed point (LFP) recursion from initial $\mathbf{F}_j (j = 0, \ldots, m)$ of the specified arity in Σ. Each such $\mathbf{F}$ determines for each structure $\mathcal{A}$ as in (1) an object $\mathbf{F}^{\mathcal{A}}$ of the same type as $\mathbf{F}$, by means of the obvious semantics. These $\mathbf{F}$ are abstract in the sense that whenever $\mathcal{A}$ and $\mathcal{A}'$ are isomorphic then $\mathbf{F}^{\mathcal{A}}$ corresponds to $\mathbf{F}^{\mathcal{A}'}$ under the given isomorphism. It is shown in the present paper how the examples of operations on lists and streams indicated above fall out as special cases of ACPs on the respective structures. Indeed, for both we derive quite general distinctive principles of recursion which appear to cover all cases met in practice. For the case of lists there is nothing surprising in this. But for the case of streams, the prior accounts of computation schemata on streams of which I am aware treat them within the framework of co-inductive definitions, which is quite special to them (cf. Mendler [14], Geuvers [7], and Paulson [18], among others). Here, in contrast, computation on streams is subsumed under a general theory of computation for arbitrary structures (1), once we settle on an appropriate structure for them. Moreover, the prior work in the co-inductive framework deals only with infinite or finite terminating streams, and it is by no means obvious how to extend that approach to deal with computation on finite non-terminating streams. (For comparison with other approaches, see the further discussion below in this introduction.)

In outline, the contents of the present paper are as follows. Sec. 2 reviews the basic functional notions and notation for many sorted structures from [5], in which, as in (1), we ignore any intensional equality relations on the basic domains. The schemata for ACPs are introduced in Sec. 3; it is shown there that they preserve monotonicity and are preserved under isomorphism. Sec. 4 deals with the relation of ACPs on a structure $\mathcal{A}$ with that on a substructure $\mathcal{B}$ of $\mathcal{A}$, i.e. for which the basic function(al)s of $\mathcal{B}$ are the restriction to $\mathcal{B}$ of those in $\mathcal{A}$. It is shown that for each ACP $\mathbf{F}$, the function(al) $\mathbf{F}^{\mathcal{B}}$ is the restriction to $\mathcal{B}$ of $\mathbf{F}^{\mathcal{A}}$. This natural (and apparently novel) result proves to have a number of uses in the following. Still working within the general theory, Sec. 5 deals with the notion of continuity of functionals, and it is shown that this property is preserved by the schemata. Then in Sec. 6 we look at first-order structures, i. e. where all the initial F_j are either individuals or (partial) functions on the basic domains. We automatically have continuity of the ACPs for first-order structures, and it is also possible to eliminate one of the schemes for such. Sec. 7 then takes up structures $\mathcal{A}$ which are represented in the natural numbers, i. e. where each A_i is a subset of $\mathbb{N}$. First it is shown for the structure $\mathcal{N} = (\mathbb{N}, Sc, Pd, 0, Eq_0)$, that the ACPs of type 1 over $\mathcal{N}$ are exactly the partial recursive functions, and those of type 2 are all partial recursive functionals and they are exactly the partial recursive functionals when applied to total function arguments, but not necessarily when applied to partial function arguments; the reason for this last is that our schemata admit sequential reduction procedures even on partial arguments, and so such partial recursive functionals as the strong (parallel) OR are not obtainable by them. A (not necessarily first order) structure with basic sets A_i contained in $\mathbb{N}$ is said to be partial recursive if each initial F_j is the restriction of some partial recursive $F_j^\star$ on $\mathbb{N}$; then the same holds for each ACP over $\mathcal{A}$. All this is illustrated in Sec. 8 with the case of A-lists where the structures are as indicated above with two basic domains A and L. These are first-order, and when A is contained in $\mathbb{N}$, we can define L as a subset of $\mathbb{N}$ and the operations $Cons$, $Head$, and $Tail$ in a uniform way, independent of A, so that the result is a partial recursive structure. Many examples are given in Sec. 8 to show how various expected procedures on lists are obtainable as ACPs.

We turn, finally, to streams in Sec. 9. That is devoted to foundational considerations, in particular, to demonstrating why it is insufficient to treat even infinite streams as a first-order structure; the reason, simply, is that then the eventually constant streams would form a substructure, and so we would not have closure under the recursion schemata which are distinctive for streams. We then move on in that section to show why it is useful to treat finite non-terminating streams as well as infinite ones. It turns out that the simplest theory of computation for potentially infinite streams is obtained if we allow completely *partial* or "gappy" streams. Then Sec. 10 gives a full development of ACPs on partial-stream structures, showing how one derives very general distinctive schemata which can then be specialized under certain conditions to potentially infinite as well as infinite streams. Many examples of computational procedures are then given, including the ones indicated above. The final section 11 is devoted to a recursion-theoretic interpretation of ACPs on A-partial streams when A is contained in $\mathbb{N}$. Here the standard structure is obtained by taking the domain S to consist of all partial functions from $\mathbb{N}$ into A, with the identity functional for Sim; this is a substructure of the standard structure for $\mathbb{N}$-partial streams, so it is sufficient to establish a recursive interpretation for the latter. Since the objects of sort S in this are of type level 1, and our schemata requires use of functionals of type level ≤ 2 over the basic domains, we need a recursive interpretation in type levels ≤ 3. This is provided by a theory of partial recursion over hereditarily partial continuous functionals of arbitrary finite type due to Eršov [2], for which closure under the ACP schemata holds. It follows that all our ACPs from streams to streams are partial recursive and, of course, continuous. Moreover, the partial recursive streams form a substructure of the standard structure, so ACPs on the former are simply the restriction of those on the latter.

There are three Appendices. The first of these concerns the relation of our schemata to Moschovakis' Formal Language of Recursion (FLR) in his papers [15], [16]; that is provided by a system of terms built by explicit definition, conditional definition and a single application of a form of simultaneous least fixed point (SLFP) recursion. A proof that the schemata here yield the same class of procedures as FLR is sketched, and there is a brief discussion of the comparative advantages of the two approaches. Appendix 2 gives a comparison of the present approach with that of Tucker and Zucker [21] for schemes of computation on stream algebras. In the same spirit as our approach, the latter treats computation on streams as a special chapter in a general theory; however, there are significant points of difference both as to the general approach and the special case. Some obvious relationships are indicated in the appendix; beyond these there are some interesting open questions. Finally, Appendix 3 contains some necessary corrections to the functional notions for computation on structures with intensional equality relations in [5]. The point of dealing with the latter is that when we finally come down to actual computation with data objects of one sort or another, we must deal with them under some system of representation, and that may require having many representations of the same object. In some cases (lists again provide a paradigm example), we don't have to face this problem, since we can provide a system of unique representations on which to carry out our computations. But that is not possible if we want to compute on algebraic word structures given by a presentation which does not have a decidable word problem or distinguished normal forms. And it is definitely not possible when we turn to work on streams or "infinite-precision" real numbers (i.e. some form or other of Cauchy sequences of rational numbers with moduli-of-convergence information). In each such case we have a natural equivalence relation which our ACPs must respect; the aim of [5] was to show how this could be done both abstractly and concretely. Unfortunately, the resulting generality led to some confusions which need to be straightened out, both as to a few of the basic definitions and as to the applications to computation on streams and reals indicated there; for the definitions, that is done in Appendix 3. What the present paper shows is how far we can go in the case of streams with a purely extensional theory, where no such confusions arise. But if we are to treat ACPs on infinite precision reals, it is necessary to return to the intensional theory; that is planned for a sequel to the present paper.

To conclude this introduction I want, first, to say something briefly about how this fits in as a chapter in generalized recursion theory (g.r.t.). The main precursor to the approach here is to be found in the dissertation of Platek [19], which developed a notion — over fairly arbitrary basic structures — of partial recursion in all finite types, where the objects are hereditarily monotonic functionals, using schemata for explicit and conditional definition and LFP recursion. That in turn provided an elegant new treatment of recursion in finite types over $\mathbb{N}$ initiated in the remarkable fundamental paper by Kleene [9]. In any case, Platek showed in his dissertation that partial recursion in objects of type level ≤ 2 can be carried out without going above that level in the application of his schemata; that is one justification for the restriction to those type levels here. The reason for the adjective 'fairly arbitrary' applied above to Platek's underlying sets is that he assumed a pairing structure for those, which in effect allows building in recursion on the natural numbers. The approach here frees us of that assumption and allows us further to treat computation on such structures as streams and infinite precision reals in which pairing is not built in.[1]

Finally, one may rightly ask how this work relates to that of the domain-theoretic approach, both in general and for the specific case of streams. As to the general approach, there is a vast literature to which initial pointers can be found in the chapters by Mosses and by Scott and Gunter in the handbook [22], as well as in the references to the text by Winskel [24] — a literature of which (to be frank) I am largely ignorant. However, one central point of difference stands out: there, LFP is applied only to *continuous* monotonic functions on CPOs. That restriction excludes application to structures containing discontinuous functionals. There have been extensive developments in g.r.t. of computation on such, beginning with the demonstration by Kleene [9] of the significance for definability theory of computation relative to the type 2 functional $\exists\mathbb{N}$, for existential quantification over $\mathbb{N}$. What Platek brought out in his dissertation is that monotonicity suffices for a g.r.t. based on the LFP scheme, and thus is applicable both to suitable discontinuous functionals and continuous ones.

As to the domain-theoretic treatment of streams, I understand from Gordon Plotkin that considerable work has been done since the early 70s by Kahn, MacQueen, Park and others on potentially infinite streams. An indication of how such may be treated as a domain can be found in Winskel [24], secs. 8.2 and 12.1. There may well be a translation from the results of the domain-theoretic approach to those of ACPs in the present case. But, as far as I know, there is no comparable domain-theoretic treatment of computation on infinite-precision reals, for which this work constitutes a preparation.

[1]Arguments for that step and for not treating higher types as the *means* for developing a g.r.t., but rather as *one case* to which g.r.t. is to be applied, were made in [3].

2 Functional notions and notations

The following notations are relevant to any finite sequence of non-empty sets $A = (A_0, \ldots, A_n)$. The letters $a, b, c, u, v, w, x, y, z$ range over individuals in $A_0 \cup \ldots \cup A_n$, as well as over finite sequences of such, as in $x = (x_1, \ldots, x_\nu)$. The letters i, j, k, ℓ range over the set $\{0, \ldots, n\}$ of sort indices; x is said to be sort i if $x \in A_i$. Then $\overline{i}$, $\overline{j}$, $\overline{k}$, $\overline{\ell}$, are used to range over finite – possibly empty – sequences of sort indices. For $\overline{i} = (i_1, \ldots, i_\nu)$ with $\ell h(\overline{i}) = \nu \geq 1$, take $A_{\overline{i}} = A_{i_1} \times \ldots \times A_{i_\nu}$; when $\nu = 0$ we may identify $A_{\overline{i}}$ with a one-element set. For simplicity of notation, we do not use corresponding overbars to indicate finite sequences x of individuals but rely on contexts such as $x \in A_{\overline{i}}$ instead to avoid ambiguity.

If t is a possibly undefined expression for an individual (of some sort i) we write $t \downarrow$ if t is defined and $t \uparrow$ otherwise. $t_1 = t_2$ is written only when both $t_1 \downarrow$ and $t_2 \downarrow$ and their values are equal. $t_1 \simeq t_2$ means $(t_1 \downarrow \vee t_2 \downarrow \Rightarrow t_1 = t_2)$.

For any two sets B and C, we write $f : B \overset{\sim}{\to} C$ if f is a partial function from B to C; as usual, $f : B \to C$ means that f is total from B to C.

Relative to any given $A = (A_0, \ldots, A_n)$, the letters $\varphi, \psi, \chi, \theta$ are used to range over partial functions from some $A_{\overline{i}}$ into an A_j, where $\ell h(\overline{i}) \geq 1$. By a type symbol of level one is meant a formal combination $\overline{i} \overset{\sim}{\to} j$ where $\ell h(\overline{i}) \geq 1$; the letters σ, τ are used to range over such type symbols. Then for $\sigma = (\overline{i} \overset{\sim}{\to} j)$ we take $A_\sigma = \{\varphi | \varphi : A_{\overline{i}} \overset{\sim}{\to} A_j\}$. For φ, ψ partial functions of the same type, $\varphi \subseteq \psi$ means that φ is a subfunction of ψ. Given a sequence $\overline{\sigma} = (\sigma_1, \ldots, \sigma_\mu)$ of type symbols, take $A_{\overline{\sigma}} = A_{\sigma_1} \times \ldots \times A_{\sigma_\mu}$; again this is identified with a one-element set if $\ell h(\overline{\sigma}) = 0$. Typical elements of $A_{\overline{\sigma}}$ are $\varphi = (\varphi_1, \ldots, \varphi_\mu)$ with $\varphi_k \in A_{\sigma_k}$; then for $\varphi, \psi \in A_{\overline{\sigma}}$, $\varphi \subseteq \psi$ is defined by $\varphi_k \subseteq \psi_k$ for each k. Once more, for simplicity of notation, we don't use overbars to indicate finite sequences of partial functions but rely instead on context to avoid ambiguity.

Functionals of type level two are partial maps from some $A_{\overline{\sigma}} \times A_{\overline{i}}$ to some A_j where $\ell h(\overline{\sigma}) > 0$; we use F, G, H to range over such, as in $F : A_{\overline{\sigma}} \times A_{\overline{i}} \overset{\sim}{\to} A_j$. When $\varphi = (\varphi_1, \ldots, \varphi_\mu)$

and $x = (x_1, \ldots, x_\nu)$, $F(\varphi, x)$ is written for $F(\varphi_1, \ldots, \varphi_\mu, x_1, \ldots, x_\nu)$. If $\mu = 0$ and $\nu > 0$, F is identified with an element of $A_{\overline{i}} \overset{\sim}{\rightarrow} A_j$, i.e. it is a partial function of type level one, and we write $F(x)$ in this case. Finally, when $\mu = \nu = 0$, F is identified with an element c of A_j.

Given $F : A_{\overline{\sigma}} \times A_{\overline{i}} \overset{\sim}{\rightarrow} A_j$ of type level two, F is said to be *A-monotonic* if

$$(1) \qquad \forall \varphi, \psi \in A_{\overline{\sigma}}\, \forall x \in A_{\overline{i}} [F(\varphi, x)\downarrow \;\&\; \varphi \subseteq \psi \Rightarrow F(\varphi, x) = F(\psi, x)] \ .$$

Note that this condition is vacuously satisfied if $\ell h(\overline{\sigma}) = 0$, i.e. if F reduces to a partial function or an individual. Under certain combinations of types, each level two monotonic F has a *least fixed point* associated with it, as follows. The simplest case is that of a functional $F : A_\sigma \times A_{\overline{i}} \overset{\sim}{\rightarrow} A_j$, where $\sigma = (\overline{i} \overset{\sim}{\rightarrow} j)$. Then F induces the total map $\widehat{F} : A_\sigma \rightarrow A_\sigma$ given by $\widehat{F} = \lambda\varphi\lambda x.F(\varphi, x)$. If F is monotonic so is $\widehat{F}$ in the ordinary sense that $\forall \varphi, \psi \in A_\sigma \left[\varphi \subseteq \psi \Rightarrow \widehat{F}(\varphi) \subseteq \widehat{F}(\psi)\right]$. Then the least fixed point of $\widehat{F}$, $\mathrm{LFP}(\widehat{F})$ is determined as usual by:

$$(2) \qquad \begin{array}{ll} \text{(i)} & \mathrm{LFP}(\widehat{F}) = \cup_\alpha \varphi^{(\alpha)}\ (= \varphi^{(\infty)}) \text{ where} \\ \text{(ii)} & \varphi^{(\alpha)} = \bigcup\limits_{\beta<\alpha} \widehat{F}(\varphi^{(\beta)}) \quad \text{for each ordinal } \alpha\ . \end{array}$$

(It is understood here that $\varphi^{(0)}$ is the empty function.) More generally, we shall deal with least fixed points relative to function and individual parameters as follows. Suppose $\overline{\sigma} = (\sigma_1, \ldots, \sigma_\mu)$, $\overline{i} = (i_1, \ldots, i_\nu)$, $\overline{k} = (k_1, \ldots, k_{\nu'})$ and $\tau = (\overline{k} \overset{\sim}{\rightarrow} j)$; we write (σ, τ) for $(\sigma_1, \ldots, \sigma_\mu, \tau)$ and $(\overline{i}, \overline{k})$ for the concatenation of $\overline{i}$ and $\overline{k}$. Thus an $F : A_{(\overline{\sigma}, \tau)} \times A_{(\overline{i}, \overline{k})} \overset{\sim}{\rightarrow} A_j$ induces $\widehat{F}_{\varphi,x} : A_\tau \rightarrow A_\tau$ for each $\varphi \in A_{\overline{\sigma}}$, $x \in A_{\overline{i}}$, by

$$(3) \qquad \widehat{F}_{\varphi,x} = \lambda\psi\lambda y.F(\varphi, \psi, x, y)\ ,$$

and $\widehat{F}_{\varphi,x}$ has an LFP when F is monotonic. More perspicuously,

$$(4) \qquad \mathrm{LFP}(\widehat{F}_{\varphi,x}) = \psi \text{ where } \psi \text{ is least with } \psi(y) \simeq F(\varphi, \psi, x, y) \text{ for all } y \in A_{\overline{k}}\ .$$

3 Schemata for computational procedures on many-sorted functional structures.

Here we deal with structures of the form

(1) $$\mathcal{A} = (A_0, A_1, \ldots, A_n, F_0, \ldots, F_m)$$

where

(2) $$A_0 = \mathbb{B} = \{tt, ff\}$$

is the Boolean sort (with $tt \neq ff$), and each F_k is an object of type level ≤ 2 over $A = (A_0, \ldots, A_n)$, i.e.

(3) $$F_k : A_{\overline{\sigma}_k} \times A_{\overline{i}_k} \overset{\sim}{\rightarrow} A_{j_k} \quad (\ell h(\overline{\sigma}_k) \geq 0 \text{ and } \ell h(\overline{i}_k) \geq 0 \text{ for } k = 0, \ldots, m)\ .$$

Thus F_k is of type level 2, 1, or 0, according as $\ell h(\overline{\sigma}_k) > 0$, $\ell h(\overline{\sigma}_k) = 0$ & $\ell h(\overline{i}_k) > 0$, or $\ell h(\overline{\sigma}_k) = \ell h(\overline{i}_k) = 0$ (as explained in section 2). It is further assumed that

(4) each F_k of type level 2 is A-monotonic, and

(5) the constants tt and ff are among the F_k's .

By the *signature* Σ for such $\mathcal{A}$ is meant the pair $(n, \langle \overline{\sigma}_k, \overline{i}_k, j_k \rangle_{k \leq m})$. With each Σ is associated the following formal schemata for computation procedures on structures of signature Σ.

I. (Initial function(al)s) $\quad \mathbf{F}(\varphi, x) \simeq \mathbf{F}_k(\varphi, x) \quad (k = 0, \ldots, m)$

II. (Identity functions) $\quad \mathbf{F}(x) = x$

III. (Application functionals) $\quad \mathbf{F}(\varphi, x) \simeq \varphi(x)$

IV. (Conditional definition) $\quad \mathbf{F}(\varphi, x, b) \simeq [\text{ if } b = tt \text{ then } \mathbf{G}(\varphi, x) \text{ else } \mathbf{H}(\varphi, x)]$

V. (Structural) $\quad \mathbf{F}(\varphi, x) \simeq \mathbf{G}(\varphi_f, x_g)$

VI. (Individual Substitution) $\quad \mathbf{F}(\varphi, x) \simeq \mathbf{G}(\varphi, x, \mathbf{H}(\varphi, x))$

VII. (Function substitution) $\quad \mathbf{F}(\varphi, x) \simeq \mathbf{G}(\varphi, \lambda y.\mathbf{H}(\varphi, x, y), x)$

VIII. (Least fixed point) $\quad \mathbf{F}(\varphi, x, y) \simeq [\text{LFP}(\lambda\psi.\lambda z.\mathbf{G}(\varphi, \psi, x, z)](y)$

In each of these schemata, the conditions to be met on the types of the arguments and sorts of the values should be fairly evident. We note only that in I, $\mathbf{F}$ is supposed to be of type $\overline{\sigma}_k \times \overline{i}_k \overset{\sim}{\rightarrow} j_k$, to accord with Σ; in IV, b is a Boolean (sort 0) variable; in the structural scheme V, $f : \{1, \ldots, \mu'\} \rightarrow \{1, \ldots, \mu\}$, $g : \{1, \ldots, \nu'\} \rightarrow \{1, \ldots, \nu\}$ and the scheme itself abbreviates $\mathbf{F}(\varphi_1, \ldots, \varphi_\mu, x_1, \ldots, x_\nu) \simeq \mathbf{G}(\varphi_{f(1)}, \ldots, \varphi_{f(\mu')}, x_{g(1)}, \ldots, x_{g(\nu')})$, thus accounting for expansion, identification and permutation of arguments; finally, in the LFP scheme VIII, $\mathbf{G}$ is supposed to be of a type $(\overline{\sigma}, \tau) \times (\overline{i}, \overline{k}) \overset{\sim}{\rightarrow} j$ where $\tau = (\overline{k} \overset{\sim}{\rightarrow} j)$.

With each structure $\mathcal{A}$ (satisfying (1)–(5)) of signature Σ and each $\mathbf{F}$ generated by these schemata is associated $\mathbf{F}^{\mathcal{A}}$ on A of the same type as $\mathbf{F}$ in the obvious way; again, this may be of type level 2,1, or 0. (Only the interpretation of the scheme for conditional definition is subject to ambiguity; this is to be understood in the strong sense that $\mathbf{F}^{\mathcal{A}}(\varphi, x, \mathit{tt}) \simeq \mathbf{G}^{\mathcal{A}}(\varphi, x)$ and $\mathbf{F}^{\mathcal{A}}(\varphi, x, \mathit{ff}) \simeq \mathbf{H}^{\mathcal{A}}(\varphi, x)$.) In order for the LFP scheme to make sense it must be verified that the $\mathbf{F}^{\mathcal{A}}$ associated with $\mathbf{F}$ is A-monotonic. This is proved by induction on the schemata: it holds by assumption (4) for the scheme I, is immediate for the schemata II and III, and is easily verified to hold for $\mathbf{F}$ if it holds for $\mathbf{G}$ and $\mathbf{H}$ in the schemata III–VII. Finally, if it holds for $\mathbf{G}$ it holds for $\mathbf{F}$ in VIII as a result of the following.

Lemma *Suppose $G : A_{(\overline{\sigma},\tau)} \times A_{(\overline{i},\overline{k})} \overset{\sim}{\rightarrow} A_j$ is A-monotonic, where $\tau = (\overline{k} \overset{\sim}{\rightarrow} j)$. Then for $F(\varphi, x, y) \simeq [\mathrm{LFP}(\lambda\psi\lambda z.G(\varphi, \psi, x, z)](y)$, we have that F is A-monotonic.*

Proof. Using the notation of §2(3),

$$F(\varphi, x, y) \simeq [\mathrm{LFP}(\widehat{G}_{\varphi,x})](y), \text{ where } \widehat{G}_{\varphi,x} = \lambda\psi\lambda z.G(\varphi, \psi, x, z).$$

By monotonicity of G, the functionals $\widehat{G}_{\varphi,x} : A_\tau \rightarrow A_\tau$ are monotonic in the usual sense. Then by definition, $\mathrm{LFP}(\widehat{G}_{\varphi,x}) = \bigcup_\alpha \psi^{(\alpha)}_{\varphi,x}$ where $\psi^{(0)}_{\varphi,x}$ is the empty function and for $\alpha > 0$, $\psi^{(\alpha)}_{\varphi,x} = \bigcup_{\beta<\alpha} \widehat{G}_{\varphi,x}(\psi^{(\beta)}_{\varphi,x})$. Now suppose that $\varphi \subseteq \theta$. We prove by induction on α that $\psi^{(\alpha)}_{\varphi,x} \subseteq \psi^{(\alpha)}_{\theta,x}$; if this holds for each $\beta < \alpha$ then $\widehat{G}_{\varphi,x}(\psi^{(\beta)}_{\varphi,x}) \subseteq \widehat{G}_{\varphi,x}(\psi^{(\beta)}_{\theta,x}) \subseteq \widehat{G}_{\theta,x}(\psi^{(\beta)}_{\theta,x})$ for each such β, so it holds for α. (The second of these inclusions is by monotonicity of G in the parameter φ.) Hence $\mathrm{LFP}(\widehat{G}_{\varphi,x}) \subseteq \mathrm{LFP}(\widehat{G}_{\theta,x})$, which establishes monotonicity of F.

Suppose $\mathcal{A}$ and $\mathcal{A}'$ are isomorphic structures of signature Σ, given by a pair (h, h') of sequences of 1-1 functions $h = (h_0, \ldots, h_n), h' = (h'_0, \ldots, h'_n)$ with $h_i : A_i \to A'_i,\ h'_i : A'_i \to A_i$ inverse to each other. Then it is proved by a straightforward induction on the $\mathbf{F}$ generated by the schemata that $\mathbf{F}^{\mathcal{A}}$ and $\mathbf{F}^{\mathcal{A}'}$ correspond to each other under (h, h'). For the LFP scheme this can be proved by a subsidiary transfinite induction on the approximations to the least fixed point, or directly by using its "leastness" property. Thus *the schemata are invariant under isomorphism*, and this justifies calling the $\mathbf{F}$'s *abstract computation procedures.* For this reason we denote by ACP(Σ) the collection of all $\mathbf{F}$'s generated by the schemata for signature Σ. Then for any particular $\mathcal{A}$ of signature $\Sigma_{\mathcal{A}}$, we take ACP($\mathcal{A}$) to be the collection of all $\mathbf{F}^{\mathcal{A}}$ for $\mathbf{F}\in$ ACP($\Sigma_{\mathcal{A}}$), and say that F is an ACP over $\mathcal{A}$ if $F = \mathbf{F}^{\mathcal{A}}$ for some such $\mathbf{F}$.

4 The substructure theorem

Given $A = (A_0, \ldots, A_n)$ and $B = (B_0, \ldots, B_n)$ with $B_0 = A_0 = \mathbb{B}$, we write $B \subseteq A$ if each $B_i \subseteq A_i$. For $\bar{\imath} = (i_0, \ldots, i_\nu)$ and $x \in A_{\bar{\imath}}$, we say x is in B if $x \in B_{\bar{\imath}}$. Then for $\varphi \in A_\sigma$ with $\sigma = (\bar{\imath} \overset{\sim}{\to} j)$, write $\varphi \restriction B$ for the function $\lambda x \in B_{\bar{\imath}}.\varphi(x)$. This need not belong to B_σ; if it does, we say that B *is closed under* φ. Equivalently that holds if

$$\text{(1)} \qquad \text{for all } x \text{ in } B, \varphi(x)\downarrow \Rightarrow \varphi(x) \text{ in } B\ .$$

By direction extension, B is said to be closed under $\varphi = (\varphi_1, \ldots, \varphi_\mu)$ if it is closed under each $\varphi_k (k = 1, \ldots, \mu)$, and we write $\varphi \restriction B$ for $(\varphi_1 \restriction B, \ldots, \varphi_\mu \restriction B)$. Now for $F : A_{\bar{\sigma}} \times A_{\bar{\imath}} \overset{\sim}{\to} A_j$, we say that B *is closed under* F if

$$\text{(2)} \qquad \text{for all } \varphi \text{ and } x, B \text{ closed under } \varphi \text{ and } x \text{ in } B \text{ and } F(\varphi, x)\downarrow \Rightarrow F(\varphi, x) \text{ in } B\ .$$

Write $F \restriction B$ for the functional $\lambda\theta \in B_{\bar{\sigma}} \lambda x \in B_{\bar{\imath}}.F(\theta, x)$; then (2) implies that $F \restriction B : B_{\bar{\sigma}} \times B_{\bar{\imath}} \overset{\sim}{\to} B_j$, but the converse does not necessarily hold, unless we know $F \restriction B(\varphi \restriction B, x) \simeq F(\varphi, x)$ for φ, x as in (2).

Suppose now that $\mathcal{A} = (A_0, \ldots, A_n, F_0, \ldots, F_m)$ meets the conditions of sec. 3. We say that $B = (B_0, \ldots, B_n)$ *determines a substructure of* $\mathcal{A}$ if $B_0 = A_0 = \mathbb{B}$ and $B \subseteq A$ and for each $k = 0, \ldots, m$, B is closed under F_k.

Theorem 1 *(Substructure Theorem) Suppose B determines a substructure of $\mathcal{A}$. Then for each* ACP $\mathbf{F}$ *and for $F = \mathbf{F}^{\mathcal{A}}$ we have:*

(i) B is closed under F, and

(ii) for any φ, x, if B is closed under φ and x is in B then $F(\varphi{\restriction}B, x) \simeq F(\varphi, x)$.

Proof. This proceeds by induction on the schemata for ACP's. By hypothesis it is immediate for the scheme I, and it is straightforward for the schemata II–V. The substitution schemata VI–VII require only a little more attention, as follows. Suppose B is closed under G, H where $G = \mathbf{G}^{\mathcal{A}}, H = \mathbf{H}^{\mathcal{A}}$. In the scheme VI,

$$F(\varphi, x) \simeq G(\varphi, x, H(\varphi, x)) \ . \tag{.1}$$

To show B is closed under F, suppose it is closed under φ and that x is in B. Then $H(\varphi, x)$ is in B so $G(\varphi, x, H(\varphi, x))$ is in B. Also, $F(\varphi{\restriction}B, x) \simeq G(\varphi{\restriction}B, x, H(\varphi{\restriction}B, x)) = G(\varphi{\restriction}B, x, H(\varphi, x)) \simeq G(\varphi, x, H(\varphi, x))$ by hypothesis. In the scheme VII,

$$F(\varphi, x) \simeq G(\varphi, \lambda y.H(\varphi, x, y), x) \ . \tag{.2}$$

Again, suppose B is closed under φ and that x is in B. Then for each y in B, $H(\varphi, x, y)$ is in B. Hence B is closed under $\psi = \lambda y H(\varphi, x, y)$ so $G(\varphi, \psi, x)$ is in B. Also,

$$\begin{aligned} F(\varphi{\restriction}B, x) &\simeq G(\varphi{\restriction}B,\ \lambda y H(\varphi{\restriction}B, x, y), x) \\ &\simeq G(\varphi{\restriction}B,\ \lambda y H(\varphi{\restriction}B, x, y){\restriction}B, x) \\ &\simeq G(\varphi{\restriction}B,\ \lambda y H(\varphi, x, y){\restriction}B, x) \\ &\simeq G(\varphi,\ \lambda y H(\varphi, x, y), x) \end{aligned} \tag{.3}$$

whenever B is closed under φ and x is in B.

To conclude the proof we now turn to the LFP scheme VIII. Here B is assumed to be closed under G where $G = \mathbf{G}^{\mathcal{A}}$,

$$F(\varphi, x, y) \simeq \mathrm{LFP}(\lambda\psi\lambda u.G(\varphi, \psi, x, u))(y)\ . \tag{.4}$$

Writing $\widehat{G}_{\varphi,x}$ for $\lambda\psi\lambda y.G(\varphi, \psi, x, y)$ as before, we have

$$\begin{aligned} F(\varphi, x, y) &\simeq \psi_{\varphi,x}(y) \text{ where } \psi_{\varphi,x} = \bigcup_{\alpha} \psi^{(\alpha)}_{\varphi,x} \text{ and each} \\ \psi^{(\alpha)}_{\varphi,x} &= \bigcup_{\beta<\alpha} \widehat{G}_{\varphi,x}(\psi^{(\beta)}_{\varphi,x}), \end{aligned} \tag{.5}$$

i.e.

$$\psi^{(\alpha)}_{\varphi,x}(y) \simeq z \Leftrightarrow \exists\beta < \alpha \left[G\left(\varphi, \psi^{(\beta)}_{\varphi,x}, x, y\right) \simeq z\right]\ . \tag{.6}$$

Now suppose that B is closed under φ and that x is in B. Then we prove by induction on α that B is closed under $\psi^{(\alpha)}_{\varphi,x}$. If this is true for each $\beta < a$ then for y in B, $G(\varphi, \psi^{(\beta)}_{\varphi,x}, x, y)$ is in B by hypothesis on G, so $\psi^{(\alpha)}_{\varphi,x}(y)$ is in B by (.6). Thus B is closed under $\psi_{\varphi,x}$ and so $F(\varphi, x, y)$ is in B by (.5). Finally, to show

$$F(\varphi{\restriction}B, x, y) \simeq F(\varphi, x, y) \tag{.7}$$

for φ closed under B and x, y in B, we show by induction on α that

$$\psi^{(\alpha)}_{\varphi|B,x}(y) \simeq \psi^{(\alpha)}_{\varphi,x}(y) \text{ for each } y \text{ in } B\ , \tag{.8}$$

and hence

$$\psi^{(\alpha)}_{\varphi|B,x}{\restriction}B = \psi^{(\alpha)}_{\varphi,x}{\restriction}B\ . \tag{.9}$$

Corollary. *Suppose B determines a substructure of $\mathcal{A} = (A_0, \ldots, A_n, F_0, \ldots, F_m)$. Let $\mathcal{B} = (B_0, \ldots, B_n, F_0{\restriction}B, \ldots, F_m{\restriction}B)$; then for each* ACP $\mathbf{F}$ *and $F = \mathbf{F}^{\mathcal{A}}$ we have*

$$F{\restriction}B = \mathbf{F}^{\mathcal{B}}\ .$$

In other words, the interpretation of $\mathbf{F}$ *in $\mathcal{B}$ is the restriction to B of its interpretation in $\mathcal{A}$.*

5 Continuity

Suppose $\mathcal{A} = (A_0, A_1, \dots, A_n, F_0, \dots, F_m)$ satisfies the conditions of Sec. 3. A functional F on $A = (A_0, \dots, A_n)$ of type level 2 is said to be *continuous* if

(1) whenever $F(\varphi, x) \simeq y$ then there exists finite $\widetilde{\varphi} \subseteq \varphi$ such that $F(\widetilde{\varphi}, x) \simeq y$.

For F of type level ≤ 1, F is automatically continuous.

Theorem 2 *If each F_k in $\mathcal{A}$ is continuous then for each* ACP $\mathbf{F}$*, the functional $\mathbf{F}^{\mathcal{A}}$ is continuous.*

Proof. By induction on the generation of $\mathbf{F}$. Only the schemata VII and VIII require any special attention. Suppose $G = \mathbf{G}^{\mathcal{A}}, H = \mathbf{H}^{\mathcal{A}}$ and that $F = \mathbf{F}^{\mathcal{A}}$ where $\mathbf{F}$ is obtained from $\mathbf{G}$, $\mathbf{H}$ by VII, i.e.

$$F(\varphi, x) \simeq G(\varphi, \lambda y.H(\varphi, x, y), x) \ . \tag{.1}$$

Let $\psi = \lambda y.H(\varphi, x, y)$. If $F(\varphi, x) \simeq z$ then there exist finite $\widetilde{\varphi}^{(0)} \subseteq \varphi$, $\widetilde{\psi} \subseteq \psi$ with $G(\widetilde{\varphi}^{(0)}, \widetilde{\psi}, x) \simeq z$. Let $\mathrm{dom}(\widetilde{\psi}) = \{y^{(1)}, \dots, y^{(p)}\}$. For each $i = 1, \dots, p$, there exists finite $\widetilde{\varphi}^{(i)} \subseteq \varphi$ with $H(\widetilde{\varphi}^{(i)}, x, y^{(i)}) \simeq \widetilde{\psi}(y^{(i)})$. Let $\widetilde{\varphi} = \widetilde{\varphi}^{(0)} \cup \dots, \cup \widetilde{\varphi}^{(p)}$. Then by monotonicity, $H(\widetilde{\varphi}, x, y^{(i)}) \simeq \widetilde{\psi}(y^{(i)})$ for each $i = 1, \dots, p$, i.e. $\widetilde{\psi} \subseteq \lambda y.H(\widetilde{\varphi}, x, y)$. Hence again by monotonicity $G(\widetilde{\varphi}, \lambda y.H(\widetilde{\varphi}, x, y), x) \simeq z$, so $F(\widetilde{\varphi}, x) \simeq z$. Now for the scheme VIII, where

$$F(\varphi, x, y) \simeq [\mathrm{LFP}(\lambda\psi\lambda u.G(\varphi, \psi, x, u)](y) \simeq [\mathrm{LFP}(\widehat{G}_{\varphi,x})](y) \tag{.2}$$

we first show that

$$F(\varphi, x, y) \simeq \psi^{(\omega)}_{\varphi,x} \tag{.3}$$

where $\psi^{(\alpha)}_{\varphi,x} = \bigcup_{\beta<\alpha} \widehat{G}_{\varphi,x}(\psi^{(\beta)}_{\varphi,x})$. This is by the usual argument to show that

$$\widehat{G}_{\varphi,x}(\psi^{(\omega)}_{\varphi,x}) \subseteq \psi^{(\omega)}_{\varphi,x} \ . \tag{.4}$$

For if $G(\varphi, \psi^{(\omega)}_{\varphi,x}, x, y) \simeq z$, there are finite $\widetilde{\varphi} \subseteq \varphi$ and $\widetilde{\psi} \subseteq \psi^{(\omega)}_{\varphi,x}$ with $G(\widetilde{\varphi}, \widetilde{\psi}, x, y) \simeq z$. Then $\widetilde{\psi} \subseteq \widetilde{\psi}^{(n)}_{\varphi,x}$ for some n, so $G(\varphi, \psi^{(n)}_{\varphi,x}, x, y) \simeq z$, i.e. $\psi^{(n+1)}_{\varphi,x}(y) \simeq z$ and, finally, $\psi^{(\omega)}_{\varphi,x}(y) \simeq z$. Hence $\psi^{(\omega)}_{\varphi,x}$ is the LFP of $\widehat{G}_{\varphi,x}$.

It remains to show that F is continuous. We prove by induction on n that for each y there is a finite $\widetilde{\varphi} \subseteq \varphi$ with

$$\psi^{(n)}_{\widetilde{\varphi},x}(y) \simeq \psi^{(n)}_{\varphi,x}(y) \; . \tag{.5}$$

For $n = 0$, this is trivial, since $\psi^{(0)}_{\varphi,x}$ is empty by definition. Suppose for n. Then

$$\psi^{(n+1)}_{\varphi,x}(y) \simeq G(\varphi, \psi^{(n)}_{\varphi,x}, x, y) \; . \tag{.6}$$

If this is defined and its value is z, then by continuity of G there exist finite $\widetilde{\varphi}^{(0)}$ and $\widetilde{\psi}$ with $\widetilde{\varphi}^{(0)} \subseteq \varphi, \widetilde{\psi} \subseteq \psi^{(n)}_{\varphi,x}$ and $G(\widetilde{\varphi}^{(0)}, \widetilde{\psi}, x, y) \simeq z$. Let $\mathrm{dom}(\widetilde{\psi}) = \{y^{(1)}, \ldots, y^{(p)}\}$. By induction hypothesis, for each i we can find finite $\widetilde{\varphi}^{(i)} \subseteq \varphi$ with $\psi^{(n)}_{\widetilde{\varphi}^{(i)},x}(y^{(i)}) \simeq \psi^{(n)}_{\varphi,x}(y^{(i)}) \simeq \widetilde{\psi}(y^{(i)})$, for each $i = 1, \ldots, p$. Let $\widetilde{\varphi} = \widetilde{\varphi}^{(0)} \cup \widetilde{\varphi}^{(1)} \cup \ldots, \cup \widetilde{\varphi}^{(p)}$. By monotonicity, also $\psi^{(n)}_{\widetilde{\varphi},x}(y^{(i)}) \simeq \widetilde{\psi}(y^{(i)})$ for each $i = 1, \ldots, p$ so $\widetilde{\psi} \subseteq \psi^{(n)}_{\widetilde{\varphi},x}$. From $G(\widetilde{\varphi}^{(0)}, \widetilde{\psi}, x, y) \simeq z$ it follows that $G(\widetilde{\varphi}, \psi^{(n)}_{\widetilde{\varphi},x}, x, y) \simeq z$, so $\psi^{(n+1)}_{\widetilde{\varphi},x}(y) \simeq z$. This completes the inductive step. Now, finally, if $F(\varphi, x, y) \simeq z$ then $\psi^{(\omega)}_{\varphi,x}(y) \simeq z$ so for some n, $\psi^{(n)}_{\varphi,x}(y) \simeq z$. Then by what has just been proved there exists finite $\widetilde{\varphi} \subseteq \varphi$ with $\psi^{(n)}_{\widetilde{\varphi},x}(y) \simeq z$, hence $\psi^{(\omega)}_{\varphi,x}(y) \simeq z$ and $F(\widetilde{\varphi}, x, y) \simeq z$. Thus F is indeed continuous.

6 Computation procedures on first-order structures

The signature $\Sigma = (n, \langle \overline{\sigma}_k, \overline{i}_k, j_k \rangle_{k \le m})$ is said to be *first-order if* $\ell h(\overline{\sigma}_k) = 0$ for each $k \le m$. Structures $\mathcal{A} = (A_0, \ldots, A_n, F_0, \ldots, F_m)$ of signature Σ are then first-order in the sense that each F_k is of type level ≤ 1, i.e. is a specified partial function or a constant. In this case they are vacuously monotonic, so each ACP over $\mathcal{A}$ is monotonic by Sec. 3. There is a useful simplification that can be made in the ACP schemata for first-order Σ, namely we can omit the scheme VII for function substitution. Write ACP_0 for ACP minus scheme VII.

Theorem 3 *If* Σ *is first order then* $\mathrm{ACP}_0(\Sigma)$ *is closed under scheme VII.*

Proof. Consider $\mathcal{A}$ of signature Σ. We prove the required closure condition in a more general form, as follows. Suppose $\mathbf{F}\in$ ACP$_0(\Sigma)$, $\mathbf{F}$ of type $\overline{\sigma}\times\overline{\imath} \overset{\sim}{\to} j, \overline{\sigma} = (\sigma_1, \ldots, \sigma_\mu)$, with arguments (φ, x) or $(\varphi_1, \ldots, \varphi_\mu, x)$ for $\varphi = (\varphi_1, \ldots, \varphi_\mu)$. Suppose also that $\mathbf{H}_1, \ldots, \mathbf{H}_\mu \in$ ACP$_0(\Sigma)$, where $\mathbf{H}_k$ has arguments $(\varphi, x, y^{(k)})$ for $k = 1, \ldots, \mu$. Then the functional $\mathbf{F}^\star$ given by the scheme

$$\mathbf{F}^\star(\varphi, x) \simeq \mathbf{F}(\lambda y^{(1)}\mathbf{H}_1(\varphi, x, y^{(1)}), \ldots, \lambda y^{(\mu)}\mathbf{H}_\mu(\varphi, x, y^{(\mu)}), x)$$

is also in ACP$_0(\Sigma)$. This is proved by a straightforward induction on $\mathbf{F}$, for arbitrary $\mathbf{H}_1, \ldots, \mathbf{H}_\mu$. The more general statement is needed for $\mathbf{F}$ introduced by the structural scheme V. It is trivially satisfied for the scheme I, since the initial $\mathbf{F}_k$ have no function arguments in a first-order signature.

Theorem 4 *If Σ is first-order and $\mathcal{A}$ is of signature Σ then $\mathbf{F}^\mathcal{A}$ is continuous for each* ACP $\mathbf{F}$.

Proof. This is a corollary of Theorem 2 in the preceding section, since each initial F_k is vacuously continuous in the case of first-order $\mathcal{A}$. Note that by Theorem 3, in the case of first-order structures the proof of Theorem 2 can be simplified, so that the essential point is only to verify that LFP preserves continuity, as shown there.

7 Computation on structures in the natural numbers

Consider, first, the *ur*-structure for recursion theory,

(1) $$\mathcal{N} = (\mathbb{N}, Sc, Pd, 0, Eq_0)$$

where $\mathbb{N}$ is the set of natural numbers with $Sc(x) = x' = x + 1$, $Pd(x) = x \dot{-} 1$ and $Eq_0(x) =$ (if $x = 0$ then 0 else 1). Here we identify $\mathbb{B}$ with $\{0, 1\}$ and *tt* with 0, *ff* with 1. Let $\Sigma_\mathcal{N}=$ the signature of $\mathcal{N}$; this is first-order.

Theorem 5

(i) The functions of type level 1 in ACP($\mathcal{N}$) *are exactly the partial recursive functions.*

(ii) The functionals of type level 2 in ACP($\mathcal{N}$) are coextensive with the partial recursive functionals when restricted to total function arguments.

(iii) Every functional of type level 2 in ACP($\mathcal{N}$) is partial recursive on partial function arguments but not conversely.

Proof.(Sketch)

1°. First show the forward implications in (i)–(iii) by associating with each $\mathbf{F}$ in $\mathrm{ACP}_0(\Sigma_{\mathcal{N}})$ a system of equations defining $\mathbf{F}^{\mathcal{N}}$ in the Herbrand-Gödel-Kleene equation calculus. For type 2 $\mathbf{F}(\varphi, x)$, these derivations at any specific partial function arguments $\varphi = (\varphi_1, \ldots, \varphi_\mu)$ and numerals for $x = (x_1, \ldots, x_\nu)$ are made from the equations for $\mathbf{F}^{\mathcal{N}}$ together with the formal diagrams for the φ_k.

2°. To show that every partial recursive function f is in ACP($\mathcal{N}$), use the Kleene normal form representation,

$$f(x_1, \ldots, x_n) \simeq U(\mu y.T_n(e, x_1, \ldots, x_n, y))$$

for suitable e, with U, T_n primitive recursive. Closure of $\mathrm{ACP}_0(\mathcal{N})$ under the scheme of primitive recursion and under the μ operator are both obtained as usual by the LFP scheme.

3°. To show that every partial recursive functional $F(\varphi, x_1, \ldots, x_n)$ of *total* arguments φ is in ACP($\mathcal{N}$), use Kleene [8] p. 330,

$$F(\varphi, x_1, \ldots, x_n) \simeq U(\mu y.T_n^{\varphi}(e, x_1, \ldots, x_n, y))$$

with T_n^{φ} primitive recursive uniformly in φ, hence in $\mathrm{ACP}_0(\mathcal{N})$.

4°. As noted by Platek [19], an example of a partial recursive functional of *partial* arguments which is not obtainable by his LFP schemata is the *strong* (parallel) *or* functional OR^+

(also denoted SO) given by

$$OR^+(\varphi_1, \varphi_2) \simeq 0 \Leftrightarrow \varphi_1(0) \simeq 0 \text{ or } \varphi_2(0) \simeq 0 \ ;$$

OR^+ is undefined otherwise. The functional OR^+ is non-deterministic, while all $\mathbf{F}^{\mathcal{N}}$ obtained in $\mathrm{ACP}_0(\mathcal{N})$ can be computed by a deterministic (sequential) reduction procedure as shown by Platek [19]. (Related procedures are to be found in Vuillemin [24], Manna [13], pp. 386ff and Moschovakis [16].)

Remark. It has been shown by Sazonov [20] that the partial recursive functionals of partial function arguments are just those obtained from schemata like $\mathrm{ACP}_0(\mathcal{N})$ when OR^+ is added. It is of course an option to include OR^+ as a basic functional with $\mathcal{N}$, for that purpose.

By a *partial recursive structure* in $\mathbb{N}$ is meant one of the form

(2) (i) $\mathcal{A} = (A_0, A_1, \ldots, A_n, F_0, \ldots, F_m)$, where

(ii) each $A_i \subseteq \mathbb{N}$, $A_0 = \mathbb{B} = \{0, 1\}$, and

(iii) each F_k is the restriction to $A = (A_0, A_1, \ldots, A_n)$ of a partial recursive functional $F_k^\star$ on $\mathbb{N}$.

Note that since each type 2 $F_k^\star$ is monotonic on arbitrary partial function arguments, its restriction F_k to A is monotonic on partial functions whose arguments and values lie in A.

Theorem 6 *Suppose $\mathcal{A}$ is a partial recursive structure in $\mathbb{N}$ of signature Σ. Then for each $\mathbf{F}$ in $\mathrm{ACP}(\Sigma)$ and for $F = \mathbf{F}^{\mathcal{A}}$ we have that F is the restriction to $\mathcal{A}$ of a partial recursive functional $F^\star$, and F is continuous.*

Proof. Note that $\mathcal{A}$ satisfying (2) can be considered to be a substructure of

$$\mathcal{N}^\star = (\mathbb{B}, \mathbb{N}, \ldots, \mathbb{N}, F_0^\star, \ldots, F_m^\star)$$

of signature Σ. Then by the Substructure Theorem (§4, Theorem 2), F is the restriction of $F^\star = \mathbf{F}^{\mathcal{N}^\star}$ to $\mathcal{A}$. Now by an extension of the argument 1° for Theorem 5(iii), we show (by induction on $\mathbf{F}$) that $F^\star$ is equationally definable, hence is a partial recursive function(al). Then $F^\star$ is continuous, so its restriction to $\mathcal{A}$ is continuous.

This result provides another version of the theorem in §11 of [5], which was interpreted as telling us how computation on ADTs could, in suitable cases, be interpreted as ordinary computation (in the sense of recursion theory). If the ADT K is strict (i.e., K is an isomorphism type) and contains a partial recursive structure $\mathcal{A}$ on $\mathbb{N}$, then abstract computational procedures on any structure of K transfer under isomorphism to partial recursive function(al)s on the "implementation" or "realization" of K via $\mathcal{A}$ in $\mathbb{N}$.

Most examples of abstract data types K which contain partial recursive structures are those whose domains are generated by finitely many finitary operations, or are obtained from such by restriction, such as lists, finite sets, finite trees, records, etc. When treated as *relative* ADTs, such as lists-of-A's, the elements of the domains to which they are relativized need not be finitary, but can still lead to partial recursive structures. Thus if $A = \{a_0, a_1, \ldots, a_n, \ldots\}$ is any countable set, we can realize lists-of-A's as a partial recursive structure, no matter how A is identified as a subset of $\mathbb{N}$; this will be demonstrated in the next section. For example, A might be a countable set of functions or other set-theoretical objects. Or A might be a non-recursive subset of $\mathbb{N}$, such as the set of Gödel numbers of total recursive functions, or the set of constructive ordinal notations. That is why no restriction was made on the A_i's in the definition above of partial recursive structures other than that they be subsets of $\mathbb{N}$.

8 Computation on list structures.

The case of abstract computational procedures on (relativized) list structures is paradigmatic for finitary data types in many respects, and is useful for comparison with computation on infinitary data types, of which streams form the main example in this paper.

There are actually several kinds of data types to be considered, depending on whether the sets A from which we are forming lists come with additional structure or not, and also whether list computations may take natural number arguments and values (e.g. the term and length functions). We begin with the core case: let LIST be the collection of all structures

(1) $\mathcal{L} = (A, L, Cons, Hd, T\ell, nil, Eq_{nil})$ where

(i) $A \neq \phi$

(ii) $Cons : A \times L \to L, \quad Hd : L - \{nil\} \to A,$
$T\ell : L - \{nil\} \to L,\ nil \in L, Eq_{nil} : L \to \mathbb{B},$

(iii) $\forall x \in L[x = nil \Leftrightarrow Eq_{nil}(x) = tt]$

(iv) $\forall a \in A \forall \ell \in L[Cons(a, \ell) \neq nil \ \&\ Hd(Cons(a, \ell)) = a \ \&\ T\ell(Cons(a, \ell)) = \ell]$

(v) $\forall X \subseteq L[nil \in X \ \&\ Cons : A \times X \to X \Rightarrow X = L]$.

These are evidently first-order structures. In (1), L is thought of as the A-lists, *nil* as the empty list, and '$Cons$', 'Hd' and '$T\ell$' abbreviate the usual cons, head and tail operations. Condition (v) is the basis of proof by induction on L and thence definition by recursion (cf. Theorem 7 below). Note that we have suppressed the Boolean part $(\mathbb{B}, tt, ff)$ of $\mathcal{L}$; this is assumed to be implicitly given here and throughout the following.

The core structures $\mathcal{L}$ may be augmented by some new basic domains and/or new basic (monotonic) function(al)s or distinguished elements. For example, with $\mathcal{N} = (\mathbb{N}, Sc, Pd, 0, Eq_0)$, we write $(\mathcal{L}, \mathcal{N})$ for the structure $\mathcal{L}$ augmented by the set $\mathbb{N}$, the operations Sc, Pd, Eq_0 and the element 0. In general (possibly) expanded structures are indicated by $\mathcal{L}^+ = (\mathcal{L}, \ldots)$. In all cases, $\mathcal{L}$ is supposed to satisfy the conditions in (1) above. The basic scheme for recursive definition on any such structure is given by the following.

Theorem 7 *Suppose $\mathcal{L}^+ = (\mathcal{L}, \ldots)$ with $\mathcal{L} \in LIST$ as in (1) and that C is a subset of one of the basic sets in $\mathcal{L}^+$. Suppose also that G, H are ACPs over $\mathcal{L}^+$ with $G \in C$ and $H : A \times L \times C \to C$. Then we can find an* ACP *F over $\mathcal{L}^+$ satisfying*

(i) $F : L \to C$

(ii) $F(nil) = G$, *and*

(iii) $F(Cons(a, \ell)) = H(a, \ell, F(\ell))$ *for each* $\ell \in L$.

Moreover, the same holds uniformly in any parameters φ, x.

Proof. Simply take F to be the LFP ψ of

$$\psi(\ell) \simeq [\text{ if } \ell = nil \text{ then } G \text{ else } H(Hd(\ell), T\ell(\ell), \psi(T\ell(\ell))].$$

Then (i)–(iii) are proved by induction on ℓ.

Corollary *If $\mathcal{L}' = (A', L', Cons', Hd', T\ell', nil', Eq_{nil'})$ satisfies the conditions* (1)(i)–(v)*, and if $I : A \cong A'$ then $F : \mathcal{L} \cong \mathcal{L}'$, where*

(i) $F(nil) = nil'$, and

(ii) $F(Cons(a, \ell)) = Cons'(I(a), F(\ell))$.

Given A, now write $\mathcal{L}(A)$ for any structure of the form (1) and LIST(A) for all such structures; this is a strict ADT. We show next how to produce a partial recursive structure in LIST(A) uniformly for all $A \subseteq \mathbb{N}$. Namely, let P be a (primitive) recursive pairing operation on the natural numbers, with inverses P_1 and P_2 and with 0 not in its range.

(2)
(i) $P : \mathbb{N}^2 \to \mathbb{N} - \{0\}$,

(ii) $P_1(P(n, m)) = n \;\&\; P_2(P(n, m)) = m$

Further identify $\mathbb{B}$ with $\{0, 1\}$, *tt* with 0 and *ff* with 1, and take

(3) $$Eq_0 = \lambda n.[\text{ if } n = 0 \text{ then } 0 \text{ else } 1].$$

Finally, given $A \subseteq \mathbb{N}$, let

(4) $$L(A) = \bigcap X \subseteq \mathbb{N}[0 \in X \;\&\; \forall a \in A\, \forall x \in X(P(a, x) \in X)]$$

Then

(5) $$(A, L(A), P\restriction A \times L(A), P_1\restriction L(A) - \{0\}, P_2\restriction L(A) - \{0\}, 0,\ Eq_0\restriction L(A))$$

is a partial recursive structure in LIST (A).

We turn now to abstract computational procedures on list structures in general. For greater perspicuity, we write $\langle a; \ell\rangle$ for $Cons(a, \ell)$ in the following. First, working simply on the core structure $\mathcal{L}$, we obtain operations for concatenation, one-termed lists and reversal as follows:

(6) (i) $Concat : L \times L \to L$ is given as follows, where we write $\ell \star \ell'$ for $Concat(\ell, \ell')$:

$nil \star \ell' = \ell'$ and $\langle a; \ell\rangle \star \ell' = \langle a; \ell \star \ell'\rangle$.

(ii) $One : A \to L,$ for which we write $One(a) = \langle a\rangle,$ is given by $\langle a\rangle = \langle a; nil\rangle$.

(iii) $Rev : L \to L$ is given by $Rev(nil) = nil$ and $Rev(\langle a; \ell\rangle) = Rev(\ell) \star \langle a\rangle$.

We can now write $\langle a\rangle \star \ell$ for $\langle a; \ell\rangle$ or $Cons(a, \ell)$. The Map functional, which sends A-lists pointwise into A-lists via any given φ, is provided by:

(7) $Map : (A \to A) \times L \to L,$ where

$Map(\varphi, nil) = nil$ and $Map(\varphi, \langle a\rangle \star \ell) = \langle\varphi(a)\rangle \star Map(\varphi, \ell)$.

This can obviously be generalized to $Map : (A \to A') \times L \to L'$ in expanded structures $(\mathcal{L}, \mathcal{L}')$. Other interesting ACPs defined by recursions of the form given by Theorem 7 with function parameters are: (i) $Filter(\varphi, \ell)$ which, for $\varphi : A \to \mathbb{B}$, forms the sublist of all terms a in ℓ such that $\varphi(a) = tt$; (ii) $Repl(\varphi, \psi, \ell)$ which, for $\varphi : A \to \mathbb{B}, \psi : A \to A$, forms the list obtained by replacing each term a in ℓ satisfying $\varphi(a) = tt$ by $\psi(a)$; and (iii) $Del(\varphi, \ell)$ which, again for $\varphi : A \to \mathbb{B}$, forms the sublist of ℓ obtained by deleting all terms a with $\varphi(a) = tt$. For example, for the first of these we have:

(8) $Filter : (A \to \mathbb{B}) \times L \to L$ is given by

$Filter(\varphi, nil) = nil$ and

$Filter(\varphi, \langle a\rangle \star \ell) = (\text{if } \varphi(a) = tt \text{ then } \langle a\rangle \star Filter(\varphi, \ell) \text{ else } Filter(\varphi, \ell))$.

Turning now to the expansion $(\mathcal{L}, \mathcal{N})$ of the core list structure $\mathcal{L}$ by the natural number structure $\mathcal{N}$, we can define the length function by:

(9) $Lh : L \to \mathbb{N},$ where

$$Lh(nil) = 0 \text{ and } Lh(\langle a\rangle \star \ell) = Lh(\ell) + 1 \ .$$

To define the term in position n of a list ℓ for $n < Lh(\ell)$ – which is a partial function – we return to the LFP scheme:

(10)
$$Tm : \mathbb{N} \times L \overset{\sim}{\to} A, \text{ where}$$
$$Tm(n, \ell) \simeq \{ \text{ if } \ell = nil \text{ then } U(0) \text{ else}$$
$$[\text{ if } n = 0 \text{ then } Hd(\ell) \text{ else } Tm(n-1, T\ell(\ell))]\} \ ,$$

where $U : \mathbb{N} \overset{\sim}{\to} A$ is the nowhere defined function (obtained, e.g., as the LFP of $U(k) \simeq U(k+1)$). In order to make Tm total, we would have to choose some ad hoc element $a_0 \in A$ as value for $Tm(n, \ell)$ when $n \geq Lh(\ell)$. In any case, (10) does not fall under Theorem 7 since the parameter n is varied in the recursion on ℓ. Alternatively, (10) can be considered as given by recursion on n with the parameter ℓ varied. Related operations obtained in either of these ways are: (i) $Del(n, \ell)$ which deletes $Tm(n, \ell)$ from ℓ, and (ii) $\mathrm{Ins}(b, n, \ell)$ which inserts an element b of A in ℓ following $Tm(n, \ell)$. We shall not spell out these recursions. From now on we write $(\ell)_n$ for $Tm(n, \ell)$ when $n < Lh(\ell)$, so $(\ell)_0 = Hd(\ell)$ when $\ell \neq nil$. Also we write $\ell^{\rightarrow}$ for $T\ell(\ell)$, so that $(\ell)_n = (\ell^{\rightarrow})_{n-1}$ when $0 < n < Lh(\ell)$.

The general scheme analogous to that for Theorem 7 for recursion on lists ℓ in which given parameters x may be varied is as follows:

(11)
$$F(x, \ell) \simeq \{ \text{ if } \ell = nil \text{ then } G(x) \text{ else } H(x, \ell, F(K(x), \ell^{\rightarrow}))\}$$
$$\text{where } x' = K(x) \text{ is of the same arity as } x \ .$$

More generally, we may vary function parameters as well. The proof that such F is total when G, H are total on suitable sets proceeds by induction on $Lh(\ell)$. Another interesting example of such recursive definition with varied parameters is provided by the test for equality on A-lists. This requires augmenting $\mathcal{L}$ by a test for equality $Eq_A : A \times A \to \mathbb{B}$ on A, i.e. such that $Eq_A(a, a') = tt \Leftrightarrow a = a'$. Then we take:

(12)
$$Eq_L(\ell, \ell') \simeq [\text{ if } \ell = nil \text{ then } Eq_{nil}(\ell') \text{ else}$$

$$\neg Eq_{nil}(\ell') \wedge Eq_A((\ell)_0, (\ell')_0) \wedge Eq_L(\ell^{\rightarrow}, (\ell')^{\rightarrow})] \ ,$$

where $\neg : \mathbb{B} \to \mathbb{B}$ and $\wedge : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ are the usual Boolean operations of negation and conjunction. Then $Eq_L : L \times L \to \mathbb{B}$ by induction on $Lh(\ell)$ and we have $Eq_L(\ell, \ell') = tt \Leftrightarrow Lh(\ell) = Lh(\ell') \ \&\ \forall n < Lh(\ell)[Eq_A((\ell)_n, (\ell')_n) = tt]$. More generally, any "decidable" relation $\rho : A \times A \to \mathbb{B}$ on A can be extended pointwise to lists uniformly by:

$$(13) \qquad Ext(\rho, \ell, \ell') \ \simeq \ [\text{ if } \ell = nil \text{ then } Eq_{nil}(\ell') \text{ else}$$
$$\neg Eq_{nil}(\ell') \wedge \rho((\ell)_0, (\ell')_0) \wedge Ext(\rho, \ell^{\rightarrow}, (\ell')^{\rightarrow})] \ .$$

To treat sorting on lists as an ACP, we need to augment A by a less-than relation on A, $Less_A : A \times A \to \mathbb{B}$. Then any of the usual sorting procedures such as bubble-sort, merge-sort, quick-sort, etc., can be turned into an ACP over $\mathcal{L}(A)$. Again, more generally, each of these can be considered as obtained uniformly from a relation $\rho : A \times A \to \mathbb{B}$.

Each of the abstract computational procedures considered here determines a partial recursive function(al) when specialized to partial recursive list structures. Moreover, they are obtained uniformly from the operations P, P_1, P_2 and Eq_0 of (2), (3) above. Thus they may be considered as *polymorphic* operations on partial recursive list structures. Another way of looking at this uniformity is through the substructure theorem, with each substructure $\mathcal{L}(A)$ in (5) considered as a substructure of:

$$(14) \qquad \mathcal{L}(\mathbb{N}) = (\mathbb{B}, \mathbb{N}, \mathbb{N}, P, P_1, P_2, 0, Eq_0) \ .$$

Then for each ACP $\mathbf{F}$ and each $A \subseteq \mathbb{N}$, the restriction of the partial recursive function(al) $\mathbf{F}^{\mathcal{L}(\mathbb{N})}$ to $\mathcal{L}(A)$ is co-extensive with $\mathbf{F}^{\mathcal{L}}(A)$. The same applies to the ACPs for the various augmented structures considered above.

Remarks

1. Abstract data types and computation on them are supposed to be independent of their implementation. But some forms of implementation are more suitable for efficiency of

operations than others. For example, in the case of lists, the operations of deletion and insertion at a given position in a list are much more efficiently performed if these are implemented as linked lists rather than as arrays. In a linked A-list, each item is at a given location in "memory" and consists of a member a of A together with a "pointer" to another memory location. For these an appropriate ADT would contain besides A, L also a domain M interpreted as the memory locations.

2. There is a sense in which one can compose ADTs. For example we can substitute for the domain A in lists-of-As the domain lists-of-Bs, so that the L in $\mathcal{L}(A)$ is interpreted as lists-of-lists-of-Bs. An example of an operation which can be defined on this composed structure is *Flatten*, which takes a list-of-lists-of-Bs into lists of Bs in the usual way. We shall not try to develop the formation of composed structures as a general notion here. It should be clear how the result of any particular composition of structures can be treated as a new structure to which the notion of ACP is then appropriately applied. Note that the special case lists-of-list works out nicely in the partial recursive interpretation (5) simply by taking $A = L(B)$ as defined in (4), and we can use the same operations in $\mathcal{L}(A)$ as in $\mathcal{L}(B)$.

9 Foundations of computation on stream structures.

In the framework of computation on ADTs, streams over a set A are to be treated as a basic set S in a suitable structure $\mathcal{S}$ analogous to that for A-lists. Intuitively, an A-stream is an *infinite sequence* $s = \langle s_0, \ldots, s_n, \ldots \rangle$ of members of A, or a *potentially infinite sequence* of such, in a sense to be explained below. Thus, though the standard interpretation of S consists of *second-order* objects, in the present approach they are to be treated as *first-order* objects in $\mathcal{S}$. On the other hand, as we shall now argue, it is insufficient for the intended applications to construe $\mathcal{S}$ itself as a first-order structure in the sense of Sec. 6; that is, $\mathcal{S}$ will have to include a functional of type level 2 among its basic F_k.

If we try to treat the structures for infinite streams as being first-order in the sense of Sec. 6, the obvious form for these to take would be as follows (where we use the superscript '1' to distinguish these from second-order structures):

(1)
(i) $\mathcal{S}^{(1)} = (A, S, Cons, Hd, T\ell)$ where

(ii) $A \neq \phi$

(iii) $Cons : A \times A \to S, Hd : S \to A, T\ell : S \to S$

(iv) $\forall a \in A\, \forall s \in S[Hd\,(Cons\,(a,s)) = a \,\&\, T\ell(Cons(a,s)) = s]$.

The main point against this is that these (and similar) conditions do not uniquely determine $\mathcal{S}^{(1)}$ up to isomorphism, given A. Two non-isomorphic structures are obtained by interpreting S in the first instance to be the set $(\mathbb{N} \to A)$ of *all* functions from $\mathbb{N}$ to A, and in the second instance to be the subset $(\mathbb{N} \underset{fin}{\to} A) = \{f \in \mathbb{N} \to A | \exists n \forall m \geq n(f(m) = f(n))\}$ of eventually constant functions. In both cases we take $Cons(a,f) = \lambda n.[$ if $n = 0$ then a else $f(n-1)], Hd(f) = f(0)$ and $T\ell(f) = \lambda n.f(n+1)$, for $f : \mathbb{N} \to A$. The first-order structure (1) with $S = (\mathbb{N} \to A)$ is denoted $\mathcal{S}^{(1)}_{\mathbb{N}\to A}$ and the second $\mathcal{S}^{(1)}_{\mathbb{N}\underset{fin}{\to} A}$; obviously the latter is a substructure of the former.

Secondly, the conditions (1) do not guarantee closure under the expected computation procedures. Schemes of definition by recursion on streams have been proposed and studied by Mendler [14], Geuvers [7] and Paulson [18], among others, in the approach to streams as a *co-inductive type* (in a sense to be explained below). The simplest associated scheme, called *co-recursion*, is supposed to yield F from G, H defined on a set C, satisfying:

(2)
$$F : C \to S \text{ with } F(x) = Cons(G(x), F(H(x))) \text{ for } x \in C,$$
$$\text{when } G : C \to A, H : C \to C\ .$$

It will be shown in the next section how such F (among others) can be obtained as an ACP on suitable *second-order* stream structures $\mathcal{S}^{(2)}$. In particular, this will permit us to define a function F from $\mathbb{B}$ to $\mathbb{B}$-streams (as S) by:

(3)
$$F : \mathbb{B} \to S \text{ with } F(u) = Cons(G(u), F(H(u)),$$

$$\text{where } G(u) = u,\ H(u) = \neg u,\ \text{for } u \in \mathbb{B}\ .$$

If this F could be defined as an ACP over first-order stream structures $\mathcal{S}^{(1)}$, then in $\mathcal{S}^{(1)}_{\mathbb{N}\to\mathbb{B}}$, $F(\mathit{tt})$ would be the stream $\langle \mathit{tt}, \mathit{ff}, \mathit{tt}, \mathit{ff}\ldots\rangle$. But this F is *not* a map from $\mathbb{B}$ to S in the $\mathcal{S}^{(1)}_{\mathbb{N}\underset{fin}{\to}\mathbb{B}}$ interpretation. By the substructure theorem of §4 it follows that F cannot be defined as an ACP over $\mathcal{S}^{(1)}_{\mathbb{N}\to\mathbb{B}}$, because the substructure $\mathcal{S}^{(1)}_{\mathbb{N}\underset{fin}{\to}\mathbb{B}}$ is not closed under F. Moreover, there is no obvious expansion of (1) by first-order operations for which the latter is not still a substructure of the former.

The need to somehow expand the structure (1) by a type 2 functional (or functionals) has been argued from another point of view in [5], [6]. Namely, if we are to characterize the infinite A-streams up to isomorphism relative to A, we need some kind of second-order expansion of (1) which ensures the completeness of the structure. However, unlike the case of lists in the preceding sections, there is no evident way to achieve this simply by adding a second-order condition to those of (1), though the approach through co-inductive types mentioned above might seem to suggest that. There, informally, the set S of A-streams is identified as the *largest* X such that $X \subseteq \Phi(X)$ where Φ is the monotonic operation given by $\Phi(X) = \{Cons(a,x) | a \in A \ \&\ x \in X\}$. (This is dual to the identification of the A-lists as the *smallest* X such that $\Phi(X) \subseteq X$). But while such S always exists as a subset of a universe V on which we have an operation $Cons : A \times V \to V$ (simply take $S = \bigcup X \subseteq V[X \subseteq \Phi(X)]$), its value depends very much on what V is, even if we have Hd and $T\ell$ operations with $Hd(Cons(a,x)) = a$ and $T\ell(Cons(a,x)) = x$. For example, for $A = \mathbb{N}$, here are three essentially different examples where this S turns out to be V: (i) take $V_1 = \mathbb{N}$, $Cons_1$ a pairing operation from $\mathbb{N} \times \mathbb{N}$ onto $\mathbb{N}$, and $Hd_1, T\ell_1$ its projections; (ii) take $V_2 = (\mathbb{N} \to \mathbb{N}), Cons_2(a,f) = \lambda n.\ [\text{if } n = 0 \text{ then } a \text{ else } f(n-1)], Hd_2(f) = f(0)$ and $T\ell_2(f) = \lambda n.f(n+1)$; (iii) take $V_3 = (\mathbb{N} \to \mathbb{N}) \times \mathbb{N}, Cons_3(a,(f,m)) = (Cons_2(a,f),m), Hd_3((f,m)) = Hd_2(f) = f(0)$ and $T\ell_3((f,m)) = (T\ell_2(f),m)$. Note that we do not have closure under the scheme (2) in case (i), and while we have closure under it in case (iii), the resulting F is not uniquely determined. In the articles on the co-inductive approach to streams mentioned above, the largest $X \subseteq \phi(X)$ is singled out by reference to a "suitable" categorical or set-theoretical framework which is external to the structure $\mathcal{S}$ that we are trying to characterize. For an internal characterization in the next section, we simply add instead a type 2 functional Sim which is used to ensure that every (total) $\varphi : \mathbb{N} \to A$ can be simulated by a stream $Sim(\varphi) \in S$ considered as a type 0 object.

Remark. This discussion is not meant as a criticism of the co-inductive approach, which has both heuristic value and independent interest. However, the resulting notions of computation on streams appear rather specific to that approach rather than fall out as a special case of a general notion of computation on arbitrary structures as here.

Granted, now, that we must take something like the type 2 Sim functional as basic in structures for infinite streams, we turn next to the extension of these ideas to *finite non-terminating streams*, i.e. those for which there is no signal for termination. Streams which are either infinite or finite non-terminating are called here *potentially infinite.* These arise naturally both from mathematical computations and physical phenomena. An example of the first is provided by the formation of $Filter(\varphi, s)$ — which is supposed to be the substream of the stream s consisting of all terms $(s)_n$ for which $\varphi\left((s)_n\right) = tt$ — when we don't know in advance whether there exist infinitely many (or even *any*) n for which $\varphi\left((s)_n\right) = tt$. An example of the second is provided by irregularly received signals from some extraterrestrial source, when we don't know at any point whether or not there will be any further signals. Such physical examples can even be considered as giving rise to "gappy" streams, where we regard $(s)_n$ as being undefined if there is no signal at time n (allowing for the fact that there may have been such but it was too weak to be received). In any case, it turns out that the ADTs for *partial streams* and the abstract computational procedures on them are simpler to describe than for potentially infinite streams, while we can easily extract from them the ACP's for the latter as special cases.

We are thus led to consider structures of the form

(4) $\mathcal{S} = (A, S, Cons, Hd, T\ell, Sim, \mathcal{N})$ where

(i) $A \neq \phi$

(ii) $Cons : A \times S \to S,\ Hd : S \overset{\sim}{\to} A, T\ell : S \to S,\ Sim : (\mathbb{N} \overset{\sim}{\to} A) \to S,$

(iii) $\forall a \in A \, \forall s \in S \, [Hd(Cons(a, s)) = a \ \& \ T\ell(Cons(a, s) = s)]$, and

(iv) $\forall \varphi \in (\mathbb{N} \overset{\sim}{\to} A) \, \forall n \in \mathbb{N}[Hd \, (T\ell^n \, (Sim \, (\varphi)) \,) \simeq \varphi(n)]$.

Here $\mathbb{N}$ is built in as a basic domain, along with the structure $\mathcal{N} = (\mathbb{N}, Sc, Pd, 0, Eq_0)$ that it carries; this is necessary in order to make sense of Sim and to produce useful φ to which Sim can be applied. Note that if we were just to deal with infinite streams, we would take Hd to be total, i.e. $Hd : S \to A$, and Sim to be of type $Sim : (\mathbb{N} \to A) \to S$, otherwise, no change in (4) would be necessary. But, having argued above for the value of working with potentially infinite and even more general partial streams, we shall take (4) in the above form as the starting point of the next section. However, we meet one new problem in taking this added step of generality, namely that the Sim functional is *not* monotonic in the sense of Sec. 2, i.e. we do *not* have $\varphi \subseteq \psi \Rightarrow Sim(\varphi) = Sim(\psi)$. For, the nth term of $Sim(\varphi)$ is defined only when $\varphi(n)$ is defined, by (iv) above. Write $(s)_n \simeq Hd(T\ell^n(s))$ for $s \in S$, $n \in \mathbb{N}$, and $s \subseteq_S s'$ for $\forall n[(s)_n \downarrow \Rightarrow (s')_n = (s)_n]$. What we *do* have is that Sim *is* monotonic in the weaker sense that $\varphi \subseteq \psi \Rightarrow Sim(\varphi) \subseteq_S Sim(\psi)$.

Now, in order to accommodate computation on partial streams in terms of the functional schemata of Sec. 3 (in particular, of the LFP scheme) we must generalize our basic framework to assume given with each set A_i a relation $\subseteq_{A_i}$, so that monotonicity of $F : A_{\overline{\sigma}} \times A_{\overline{i}} \overset{\sim}{\to} A_j$ is taken to mean that if the arguments of F increase in $A_{\overline{\sigma}} \times A_{\overline{i}}$ then its values increase (under $\subseteq_j$) in A_j. In the case at hand, $\subseteq_{A_i}$ will still be the identity relation on A and $\mathbb{N}$, but will be taken to be $\subseteq_S$ on S. Such a generalization would not be necessary if we were to restrict our attention to infinite streams, since there $Sim : (\mathbb{N} \to A) \to S$ is trivially monotonic in the sense of Sec. 2. However, the recursion schemata for partial streams (such as needed for the *Filter* operation) come out much more simply than they do for total streams. It turns out that little modification of our basic framework is necessary in order to deal with the generalized notion of monotonic functional. In outline, this is done as follows.

First, returning to Sec. 2, assume fixed for each $i = 0, \ldots, n$ a chain-complete partial ordering $\subseteq_{A_i}$ (also written $\subseteq_i$) on A_i. For $X \subseteq A_i$ non-empty and linearly ordered by $\subseteq_i$, write $\bigcup X$ for ℓ. u. b. (X). Given $x, y \in A_{\bar{i}}$, $\bar{i} = (i_1, \ldots, i_\nu)$, $x = (x_1, \ldots, x_\nu)$, $y = (y_1, \ldots, y_\nu)$, put $x \subseteq_{\bar{i}} y \Leftrightarrow x_k \subseteq_{i_k} y_k$ for $k = 1, \ldots, \nu$. Now by $A_{\overline{\sigma}}$ we mean the set of all $\varphi : A_{\bar{i}} \overset{\sim}{\to} A_j$ which are monotonic in the sense that $\forall x, y \in A_{\bar{i}} [\varphi(x) \downarrow \& \ x \subseteq_{\bar{i}} y \Rightarrow \varphi(y) \downarrow \& \varphi(x) \subseteq_j \varphi(y)]$. For $\sigma = (\bar{i} \overset{\sim}{\to} j)$ and $\varphi, \psi \in A_\sigma$, take $\varphi \subseteq_\sigma \psi \Leftrightarrow \forall x \in A_{\bar{i}} [\varphi(x) \downarrow \Rightarrow \psi(x) \downarrow \& \ \varphi(x) \subseteq_j \psi(x)]$. This relation is extended term-wise to $\subseteq_{\overline{\sigma}}$ for $\overline{\sigma} = (\sigma_1, \ldots, \sigma_\mu)$. Finally, for $F : A_{\overline{\sigma}} \times A_{\bar{i}} \overset{\sim}{\to} A_j$ of type level 2, we take F to be monotonic (in the generalized sense) if

$$\forall \varphi, \psi \in A_{\overline{\sigma}} \forall x, y \in A_{\bar{i}} [F(\varphi, x) \downarrow \ \& \ \varphi \subseteq_{\overline{\sigma}} \psi \ \& \ x \subseteq_{\bar{i}} y \Rightarrow F(\psi, y) \downarrow \& F(\varphi, x) \subseteq_j F(\psi, y)] \ . \tag{5}$$

In the following, we shall omit the subscripts on the various inclusion relations when these are determined by the context.

The next step is to re-examine the definition of LFP in Sec. 2. Here, the basic point to be observed is that each A_σ is also chain-complete: if $\mathcal{X} \subseteq A_\sigma$ is any non-empty linearly ordered collection then $\mathcal{X}$ has a ℓ.u.b. $\bigcup \mathcal{X}$ defined by

$$(\bigcup \mathcal{X})(x) = \left(\bigcup_{\varphi \in \mathcal{X}} \varphi \right)(x) = \bigcup_{\varphi \in \mathcal{X}} \varphi(x) \ . \tag{6}$$

Actually, completeness is needed only for well-ordered chains (as also for the basic $\subseteq_i$), in order to define LFP for monotonic F in the above sense. This is done exactly as in Sec. 2, for $\widehat{F} = \lambda\varphi\lambda x.F(\varphi, x)$:

$$\begin{aligned} \mathrm{LFP}(\widehat{F}) &= \bigcup_\alpha \varphi^{(\alpha)} \text{ where } \varphi^{(0)} \text{is the empty function and} \\ \varphi^{(\alpha)} &= \bigcup_{\beta < \alpha} \widehat{F}(\varphi^{(\beta)}) \text{ for each } \alpha > 0 \ . \end{aligned} \tag{7}$$

Again, this is extended to LFP of $F(\varphi, \psi, x, y)$ relative to parameters φ, x by taking $\widehat{F}_{\varphi,x} = \lambda\psi\lambda y.F(\varphi, \psi, x, y)$. All that then needs to be proved is that if $G(\varphi, \psi, x, y)$ is monotonic in the sense of (5) above, then so also is

(8) $$F(\varphi, x, y) \simeq [LFP(\lambda\psi\lambda z.G(\varphi, \psi, x, z)](y)\ .$$

This is established by the same argument as for the Lemma of Sec. 3. The schemata for ACPs of Sec. 3 then make sense for any structure $\mathcal{A}$ on basic domains $(A_i, \subseteq_i)$ for which the basic functions or functionals F_k are monotonic in the above sense. (Note, in particular, that the application functional $F(\varphi, x) \simeq \varphi(x)$ is monotonic by our requirement that we restrict attention to monotonic φ.) This suffices for the treatment of ACPs on partial streams which we take up next.

10 Computation on partial-stream structures.

Let P-STREAM ('P' for 'partial') be the collection of all structures

(1) $\mathcal{S} = (A, S, Cons, Hd, T\ell, Sim, \mathcal{N})$ where

(i) $A \neq \phi$,

(ii) $Cons : A \times S \to S, Hd : S \xrightarrow{\sim} A, T\ell : S \to S, Sim : (\mathbb{N} \xrightarrow{\sim} A) \to S$,

(iii) $\forall a \in A\, \forall s \in S\, [Hd(Cons(a, s) = a\, \&\, T\ell(Cons(a, s)) = s]$, and

(iv) $\forall \varphi \in (\mathbb{N} \xrightarrow{\sim} A) \forall n \in \mathbb{N}[Hd(T\ell^n(Sim(\varphi)) \simeq \varphi(n)]$.

The following notation will be used for any such structure. Given $s \in S$, write

(2) (i) $(s)_n \simeq Hd(T\ell^n(s))$, and

(ii) $s \subseteq_S s' \Leftrightarrow \forall n[(s)_n \downarrow \Rightarrow (s')_n = (s)_n]$.

This will be the basic relation assumed given on the domain S, while the relations $\subseteq_A$ and $\subseteq_\mathbb{N}$ are taken to be equality on A and $\mathbb{N}$, resp. (We here identify $\mathbb{B}$ with $\{0, 1\}$.) Then, according to the definition (2), the basic function(al)s of $\mathcal{S}$ are automatically monotonic, in the sense of the preceding section, for we have

(3) (i) $(Cons(a,s))_n \simeq [\text{ if } n=0 \text{ then } a \text{ else } (s)_{n-1}]$,

(ii) $Hd(s) \simeq (s)_0$

(iii) $(T\ell(s))_n \simeq (s)_{n+1}$, and

(iv) $(Sim(\varphi))_n \simeq \varphi(n)$.

In the following we shall write $s \subseteq s'$ for $s \subseteq_S s'$. To make the notation more perspicuous we shall also write $\langle a; s\rangle$ for $Cons(a,s)$ and $s^{\rightarrow}$ for $T\ell(s)$. Note that $(\langle a; s\rangle)_0 = a$ is always defined, given $a \in A$. It is only when we form $\langle e; s\rangle$ with e an expression (for an element of A) that may fail to be defined, that $(\langle e; s\rangle)_0$ may fail to be defined.

It was emphasized in the preceding section that something like the Sim functional is needed to characterize stream structures up to isomorphism. This is now provided directly by the following:

Theorem 10 *Suppose $\mathcal{S}' = (A', S', Cons', Hd', T\ell', Sim', \mathcal{N})$ also satisfies the conditions of* (1)(i)–(v)*, and that $I : A \cong A'$. Then $F : \mathcal{S} \cong \mathcal{S}'$ for $F(s) = Sim'(\lambda n.I((s)_n))$.*

By stream recursion we mean any general computational scheme for producing streams as values. The following suffices for all our applications here, but more general schemata are derivable, as will be indicated below. Here $\mathcal{S}^+ = (\mathcal{S}, \dots,)$ is any expanded structure with $\mathcal{S}$ as in (1).

Theorem 11 *Let $\mathcal{S}^+ = (\mathcal{S}, \dots)$ with $\mathcal{S}$ in* P-STREAM. *Suppose C is a subset of one of the basic sets in $\mathcal{S}^+$ and that G, H_0, H_1, D are* ACP*s over $\mathcal{S}^+$ with $G : C \overset{\sim}{\rightarrow} A$, $H_0 : C \rightarrow C$, $H_1 : C \rightarrow C$ and $D : C \rightarrow \mathbb{B}$. Then we can find an* ACP *F over $\mathcal{S}^+$ satisfying:*

(i) *$F : C \rightarrow S$ with*

(ii) *$F(c) = [$ if $D(c) = tt$ then $\langle G(c); F(H_0 c)\rangle$ else $F(H_1 c)]$ for all $c \in C$, and*

(iii) *if $F' : C \rightarrow S$ is any other function satisfying* (ii) *then $F(c) \subseteq F'(c)$ for all $c \in C$.*

The same holds uniformly in any parameters φ, x.

Proof. While F solves a fixed-point equation (ii), it cannot be described as its LFP, since that is the completely undefined function. Here, in contrast, F is total and is characterized by (iii) among all total solutions of (ii) as the one which is least pointwise on C. Instead we take

(.1) $$F(c) = Sim(\lambda n.\psi(c,n))$$

where

(.2) $$\psi : C \times \mathbb{N} \overset{\sim}{\to} A \text{ is the LFP of}$$
$$\psi(c,n) \simeq \begin{cases} G(c) \text{ if } D(c) = tt \text{ and } n = 0 \\ \psi(H_0 c, n-1) \text{ if } D(c) = tt \text{ and } n > 0 \\ \psi(H_1 c, n) \text{ otherwise.} \end{cases}$$

By (1) (iv),

(.3) $$(F(c)_n) \simeq \psi(c,n) \text{ for all } c,n \text{ and } F(c) \in S \text{ for all } c.$$

Hence

(.4) $$(F(c))_n \simeq \begin{cases} G(c) \text{ if } D(c) = tt \text{ and } n = 0 \\ (F(H_0 c))_{n-1} \text{ if } D(c) = tt \text{ and } n > 0 \\ (F(H_1 c))_n \text{ otherwise .} \end{cases}$$

This shows that F satisfies (ii) in the statement of our theorem. Then if F' also satisfies (ii) and we take $\theta(c,n) \simeq (F'(c))_n$, we have that θ satisfies (.2) in place of ψ, so $\psi \subseteq \theta$ by its definition as LFP of (.2); hence $F(c) \subseteq F'(c)$.

More general such recursions can be justified with the same conclusion, e. g.

(4) $$F(c) = [\text{ if } D_0(c) = tt \text{ then } \langle G(c);\ \text{if } D_1(c) = tt \text{ then } F(H_0 c) \text{ else } K(c)\rangle$$
$$\text{else } F(H_1 c)]$$

when $D_1 : C \to \mathbb{B}$ and $K : C \to S$. Similarly we can add more 'else' clauses at the end, according to suitable cases. However, as stated above, the scheme of Theorem 11 suffices for the applications here. Our next step is to see when we can strengthen the conclusion about the values of $F(c)$ for $c \in C$.

A stream $s \in S$ is said to be *infinite* (or *total*) if $\forall n[(s)_n \downarrow]$, and *potentially infinite* (or *non-gappy*) if $\forall n, m[(s)_n \downarrow \; \& \; m < n \Rightarrow (s_m) \downarrow]$. We denote by S_{inf}, S_{potinf} the subsets of S consisting of these s, resp. A stream s is said to be *finite* (*non-terminating*) if $s \in S_{\text{potinf}} - S_{\text{inf}}$.

Theorem 12 *Under the same hypothesis as Theorem 11, if* $G : C \to A$ *then* $F : C \to S_{\text{potinf}}$.

Proof. Returning to the proof of Theorem 11, we have

$$(.1) \qquad \psi(c, 0) \simeq \begin{cases} G(c) \text{ if } D(c) = tt \\ \psi(H_1 c, 0) \text{ otherwise} \end{cases}$$

and

$$(.2) \qquad \psi(c, n+1) \simeq \begin{cases} \psi(H_0 c, n) \text{ if } D(c) = tt \text{ and } n > 0 \\ \psi(H_1 c, n+1) \text{ otherwise .} \end{cases}$$

It follows from (.1) that

$$(.3) \qquad \psi(c, 0) \downarrow \Leftrightarrow \exists m(D(H_1^m c) = tt).$$

For if $D(H_1^m c) = tt$ and m is the least such then

$$\psi(c, 0) \simeq \psi(H_1 c, 0) \simeq \ldots \simeq \psi(H_1^m c, 0) \simeq G(H_1^m c).$$

But G is total so $\psi(c, 0) \downarrow$. Conversely, if $\forall m(D(H_1^m c) = ff)$ then we have $\psi(c, 0) \simeq \psi(H_1 c, 0) \simeq \ldots \simeq \psi(H_1^m c, 0) \simeq \ldots$ for all m, and the least solution ψ makes $\psi(c, 0) \uparrow$.

$$(.4) \qquad \psi(c, n+1) \downarrow \Leftrightarrow \exists m[D(H_1^m c) = tt \,\&\, \forall k < m(D(H_1^k c) = ff) \,\&\, \psi(H_0 H_1^m c, n) \downarrow].$$

For suppose $\forall m[D(H_1^m c) = ff]$, then by the same argument just given, $\psi(c, n+1) \uparrow$. Hence if $\psi(c, n+1) \downarrow$ the least m with $D(H_1^m c) = tt$ satisfies $\psi(c, n+1) \simeq \psi(H_1^m c, n+1) \simeq \psi(H_0 H_1^m c, n)$, so $\psi(H_0 H_1^m c, n) \downarrow$. Conversely, tracing this back makes $\psi(c, n+1) \downarrow$.

$$(.5) \qquad \forall n \forall c[\psi(c, n+1) \downarrow \Rightarrow \psi(c, n) \downarrow] \text{ .}$$

This is proved by induction on n. For $n = 0$ this follows directly from (.3) and (.4). Suppose it holds for n. To show for $n + 1$ we apply (.4) to $n + 1$ in place of n:

(.6) $\quad \psi(c, n+2)\downarrow \Leftrightarrow \exists m[D(H_1^m c) = tt \& \forall k < m(D(H_1^k c) = ff) \& \psi(H_0 H_1^m c, n+1)\downarrow]$.

By induction hypothesis $[\psi(H_0 H_1^m c, n+1)\downarrow \Rightarrow \psi(H_0 H_1^m c, n)\downarrow]$; Hence from (.4) and (.6), $[\psi(c, n+2)\downarrow \Rightarrow \psi(c, n+1)\downarrow]$. Since $(F(c))_n \simeq \psi(c, n)$ for all n, (.5) shows us that $(F(c))_{n+1}\downarrow \Rightarrow (F(c))_n\downarrow$, so $F(c) \in S_{\text{potinf}}$.

We cannot strengthen this further to $F : C \to S_{\text{inf}}$ under the given general conditions but that *can* be established for less general forms of recursion such as the following.

Theorem 13 *Suppose $\mathcal{S}^+ = (\mathcal{S}, \dots)$ with $\mathcal{S}$ in* P-STREAM. *Suppose C is a subset of one of the domains in $\mathcal{S}^+$ and that G, H are* ACP*s over $\mathcal{S}^+$ with $G : C \to A$ and $H : C \to C$. Then we can find an* ACP *F over $\mathcal{S}^+$ satisfying:*

(i) $F : C \to S_{\text{inf}}$ *and*

(ii) $F(c) = \langle G(c); F(Hc)\rangle$ *for all $c \in C$.*

Proof. This may be regarded as a special case of Theorem 12 with $D(c) = tt$ for all c. At any rate, here $(F(c))_n \simeq \psi(c, n)$ where ψ is the LFP of

(.1)
$$\psi(c, n) \simeq \begin{cases} G(c) \text{ if } n = 0 \\ \psi(Hc, n-1) \text{ if } n > 0. \end{cases}$$

It is then proved by induction on n that $\forall c[\psi(c, n)\downarrow]$. Hence $F(c) \in S_{\text{inf}}$.

The same conclusion can be drawn for slightly more general recursions of the form:

(5)
$$F(c) = \langle G(c);\ \text{if } D(c) = tt \text{ then } F(Hc) \text{ else } K(c)\rangle$$

when $G : C \to A$, $D : C \to \mathbb{B}$, $H : C \to C$ and $K : C \to C$. (This is the form that corresponds to the scheme of corecursion in [7]).

We now turn to examples of the recursive stream definitions in Theorems 11–13. Most of these are drawn from Abelson et al [1], Sec. 3.4 (cf. especially 3.4.4 for operations on and to infinite streams). First, for $\mathcal{S}, \mathcal{S}'$ both in P-STREAM, we can define the ACP

(6) $$MAP : (A \to A') \times S \to S' \text{ with}$$
$$MAP(\varphi, s) = \langle \varphi((s)_0);\ MAP(\varphi, s^{\to}) \rangle'.$$

This is obtained from Theorem 11 uniformly in φ, with $\mathcal{S}^+ = (\mathcal{S}, \mathcal{S}')$, and S' in place of S there, $C = S, G(s) \simeq (s)_0, H(s) = s^{\to}$ and $D(s) = tt$ all s, for any given φ. We can then apply Theorem 13 to conclude

(7) $$\text{if } s \in S_{\text{inf}} \text{ then } MAP(\varphi, s) \in S'_{inf},$$

by restricting C to S_{inf}, on which $G(s) = (s)_0$ becomes total. Note that if φ is a 1–1 correspondence between the domain A of S and A' of S' then $MAP(\varphi)$ induces an isomorphism of $\mathcal{S}$ with $\mathcal{S}'$, so this gives another route to Theorem 10.

The next example extends any operation $\varphi : A \times A \to A$ pointwise to A-streams,

(8) $$Op : (A \times A \to A) \times S \times S \to S \text{ with}$$
$$Op(\varphi, s, s') = \langle \varphi((s)_0, (s')_0);\ Op(\varphi, T\ell(s), T\ell(s')) \rangle.$$

This is by Theorem 11 uniformly in φ with $C = S \times S$, $G(s, s') \simeq \varphi((s)_0, (s')_0)$, $H(s, s') = (s^{\to}, (s')^{\to})$, $D(s, s') = tt$ all s, s'. Again, by Theorem 13,

(9) $$\text{if } s, s' \in S_{\text{inf}} \text{ then } Op(\varphi, s, s') \in S_{\text{inf}}.$$

The procedure of meshing or interleaving two A-Streams is given by

(10) $$Mesh : S \times S \to A \text{ with } Mesh(s, s') = \langle (s)_0;\ Mesh(s', s^{\to}) \rangle,$$

taking $C = S \times S$, $G(s, s') \simeq (s)_0$, $H(s, s') = (s', s^{\to})$ and $D(s, s') = tt$, in Theorem 11. Again, by Theorem 13,

(11) $$\text{if } s, s' \in S_{\text{inf}} \text{ then } Mesh(s, s') \in S_{\text{inf}}\ .$$

More interesting, next, is the general procedure of filtering with respect to a predicate $\varphi : A \to \mathbb{B}$. We have

(12)
$$Filter : (A \to \mathbb{B}) \times S_{\text{inf}} \to S_{\text{potinf}} \text{ with}$$
$$Filter(\varphi, s) = \begin{cases} \langle (s)_0; Filter(\varphi, s^{\to}) \rangle \text{ if } \varphi((s)_0) = \mathit{tt} \\ \\ Filter(\varphi, s^{\to}) \text{ otherwise.} \end{cases}$$

This falls under Theorem 12 uniformly in φ, with $C = S_{\text{inf}}, G(s) = (s)_0,\ H(s) = s^{\to}$ and $D(s) = \varphi((s)_0)$. (The restriction of C to S_{inf} is needed to make G total). Clearly,

(13)
$$\text{if } \forall n \exists m \geq n[\varphi((s)_n) = \mathit{tt}] \text{ and } s \in S_{\text{inf}} \text{ then } Filter(\varphi, s) \in S_{\text{inf}}.$$

However, if the hypothesis of (13) is not known, we can only treat $Filter(\varphi, s)$ as a potentially infinite stream. If $\exists n \forall m \geq n[\varphi((s)_n) = \mathit{ff}]$ but we have no proof of that, then $Filter(\varphi, s)$ is an example of a finite non-terminating stream.

In the present approach there is no obvious general way to represent the filtering process computationally which ensures that it will always lead from infinite streams to infinite streams. One way has been suggested in the coinductive approach by Leclerc and Paulin-Mohring [12], in the framework of the *Coq* language. Their idea is to build in a proof of the hypothesis of (13) as part of the data parameters. One particular case is examined op. cit. (The Sieve of Eratosthenes, see next), and the general treatment is only suggested. In any case, this only shifts the problem from providing an external proof to that of providing its formalization as internal data, a step which will only make the filtering procedure more cumbersome. Our view here is that for programming purposes, the computational procedure of filtering should be represented in as simple a way as possible (here, as in (12)), and that while its application to a specific (φ, s) may call for a proof of the hypothesis of infinitude of $\{n : \varphi((s)_n) = \mathit{tt}\}$, it need not require it. For example, we may want to filter the predicate of being a twin prime up to a point, say $n = 10^9$, in order to provide experimental data about twin primes. If one insisted on internalizing a proof of infinitude as part of the data, this would never get off the ground, at least not as a stream procedure.

Filtering can also be carried out relative to a parameterized predicate $\varphi : A \times U \to \mathbb{B}$, by taking

$$(14) \qquad Filter(\varphi, u, s) = Filter(\lambda a.\varphi(a, u), s) \ .$$

The general sieving procedure is then defined by:

$$(15) \qquad Sieve : (A \times U \to \mathbb{B}) \times S_{\text{inf}} \to S_{\text{potinf}} \text{ with}$$

$$Sieve(\varphi, s) = \langle (s)_0; Sieve(Filter(\varphi, (s)_0, s^{\to})\rangle.$$

In particular, the Sieve of Eratosthenes applied to a stream s of natural numbers is given by $Sieve(\varphi_{\overline{div}}, s)$ where

$$(16) \qquad \varphi_{\overline{div}} : \mathbb{N} \times \mathbb{N} \to \mathbb{B} \text{ with } \varphi_{\overline{div}}(a, u) = tt \Leftrightarrow u \not| \, a.$$

$Sieve(\varphi_{\overline{div}}, s)$ acts by filtering out all multiples of $(s)_0$ before proceeding on to the sieve of $s^{\to}$. Hence

$$(17) \qquad Prime = Sieve(\varphi_{\overline{div}}, \lambda n.(n + 2)) \ .$$

On the other hand for example, $Sieve(\varphi_{\overline{div}}, \lambda n.2)$ is undefined following its first term. Of course, (17) is only one of many methods for generating the sequence of primes as a stream.

A nice example of an infinite number-theoretical stream produced without filtering is the Fibonacci sequence (cf. [1] p. 267);

$$(18) \qquad Fib = Fib(0, 1) \text{ where } Fib(n, m) = \langle n; Fib(m, n + m)\rangle \ .$$

By Theorem 13, $Fib \in S(\mathbb{N})_{\text{inf}}$ (i.e. is an infinite $\mathbb{N}$- stream). Another nice example comes from the theory of divergent series: the Cesáro $(C, 1)$ summation method leads from any infinite stream s of rational numbers (or, more generally, real numbers) to the infinite stream $Ces(s)$ of partial averages given by

$$(19) \qquad (Ces(s))_n = \frac{1}{n+1} \sum_{i=0}^{n} (s)_i \ .$$

In terms of the stream operations, this can be analyzed as the composition of pointwise division by the stream of positive integers with the operation $Sum : S \to S$ given by

$$(20) \qquad Sum(s) = \langle (s)_0; (s)_0 \dot{+} Sum(s^{\rightarrow}) \rangle$$

where $x \dot{+} s$ is the stream obtained from s by pointwise addition by x, i.e. $(x \dot{+} s)_n = x + (s)_n$. The operation Ces itself can be iterated any number of times.

We close this section with a few applications to combined structures of streams and lists. To begin with, a structure $(\mathcal{S}, \mathcal{L})$ in which $\mathcal{S}$ acts as the A-streams and $\mathcal{L}$ as the A-lists allows us to define

$$(21) \qquad \begin{aligned} &Append : L \times S \to S \text{ with} \\ &Append(\ell, s) = [\text{ if } \ell = nil \text{ then } s \text{ else } \langle Hd(\ell);\ Append(\ell^{\rightarrow}, s) \rangle\ . \end{aligned}$$

This is simply given by recursion on L. In the following we write $\ell \star s$ for Append (ℓ, s). Inversely, we define

$$(22) \qquad \begin{aligned} &Truncate : S \times \mathbb{N} \to L \text{ with} \\ &Truncate(s, n) = [\text{ if } n = 0 \text{ then } nil \text{ else } Truncate(s, n \dot{-} 1) \star \langle (s)_n \rangle], \end{aligned}$$

which is given by recursion on $\mathbb{N}$. Writing $\overline{s}(n)$ for $Truncate(s, n)$, we have $\overline{\ell \star s}(Lh(\ell)) = \ell$.

To treat streams of lists we can use a combined structure $(\mathcal{S}_A, \mathcal{S}_B, \mathcal{L}_B)$, where $\mathcal{S}_A, \mathcal{S}_B$ are the structures of A-streams and B-streams, resp., and $\mathcal{L}_B$ is the structure of B-lists and where, finally, it is assumed that

$$(23) \qquad A = L_B\ ,$$

i.e. that the members of A are exactly the B-lists. Hence, given $a \in A$ we can test (in $\mathcal{L}_B$) whether $a = nil_B$, and if $a \neq nil_B$, form $Hd_B(a)$ as an element of B and $T\ell_B(a)$ as a B-list, i. e. as a member of A. Now S_A, the A-streams, can be thought of as streams of B-lists. Hence we can define the ACP:

$$(24) \qquad Flatten : S_A \to S_B \text{ with}$$

$$Flatten(s) = [\ \text{if}(s)_0 \neq nil_B \text{ then } \langle Hd_B((s)_0);\ Flatten(T\ell_B((s)_0) \star T\ell_A(s))\rangle$$
$$\text{else } Flatten(T\ell_A(s))]\ .$$

This falls under Theorems 11 and 12, so that if $s \in (S_A)_{\text{inf}}$ then $Flatten(s) \in (S_B)_{\text{potinf}}$, and $Flatten(s) \in (S_B)_{\text{inf}}$ only if s has infinitely many non-nil terms.

Remark. Some authors, including Abelson et al [1] and Paulson [18] treat lists as finite streams. But these must be distinguished from finite non-terminating streams in the sense defined above. If a list is to be considered as a stream in the way these are dealt with here, it must be provided with a signal for termination. One way in which that can be accomplished is to reserve a specified element a_0 of A to serve as such a signal. Then, for example, lists could be identified with infinite streams s with the property that if $(s)_n = a_0$ then $(s)_m = a_0$ for all $m > n$; the list in the proper sense associated with s is then just $\overline{s}(n)$ for the least such n. Obvious alternative identifications are also workable.

11 Recursion-theoretic interpretation of computation on number-stream structures.

We shall deal here with computational procedures on structures for A-streams when $A \subseteq \mathbb{N}$. The *standard realization* for these will simply take

(1) $$S(A) = (\mathbb{N} \overset{\sim}{\to} A) \text{ for each } A \subseteq \mathbb{N}\ .$$

In particular, the standard realization for $A = \mathbb{N}$ is taken as:

(2) $$\mathcal{S}(\mathbb{N}) = (\mathbb{N}, S(\mathbb{N}), Cons, Hd, T\ell, Sim, \mathcal{N})$$

where

(i) $Cons : \mathbb{N} \times S(\mathbb{N}) \to S(\mathbb{N}),\ Hd : S(\mathbb{N}) \overset{\sim}{\to} \mathbb{N},\ T\ell : S(\mathbb{N}) \to S(\mathbb{N}),$

$Sim : (\mathbb{N} \overset{\sim}{\to} \mathbb{N}) \to S(\mathbb{N})$

are given by

(ii) $Cons(a,s) = \lambda n.[\text{ if } n=0 \text{ then } a \text{ else } s(n \dot{-} 1)]$

(iii) $Hd(s) \simeq s(0)$

(iv) $T\ell(s) = \lambda n.s(n+1)$

(v) $Sim(\varphi) = \varphi$.

The substructure induced by A is then

$$\mathcal{S}(A) = (A, S(A), Cons, Hd, T\ell, Sim, \mathcal{N}) \tag{3}$$

where $Cons$ is restricted to $A \times S(A)$, Hd and $T\ell$ are restricted to $S(A)$ and Sim is restricted to $\mathbb{N} \xrightarrow{\sim} A$. Clearly, $\mathcal{S}(A)$ is in P-STREAM for any A, and by the Categoricity Theorem 10, every member $\mathcal{S}$ of P-STREAM on A has $\mathcal{S} \cong \mathcal{S}(A)$.

In (1), (2) we are trying to maintain the distinction between $\mathbb{N} \xrightarrow{\sim} \mathbb{N}$ in it role as the set of all partial functions from $\mathbb{N}$ to $\mathbb{N}$ and its role as the interpretation of the individual domain $S(\mathbb{N})$ in $\mathcal{S}(\mathbb{N})$. For the former, we continue to use function letters $\varphi, \psi, \ldots,$, while for the latter, stream letters $s, s', \ldots$. The notation is not without ambiguity, though. In (2)(v), φ appears as a partial function on the left side of the equation and as a member of $S(\mathbb{N})$ on the right side. And in the equations (2)(ii)–(iv) with expressions of the form $s(e)$, the stream object s is treated as a partial function. We can lessen the latter ambiguity by writing $(s)_n$ for $s(n)$ when treating s as a stream object. Note that the inclusion relation $\subseteq$ for partial functions coincides with $\subseteq_{S(\mathbb{N})}$ in their guise as stream objects.

The following is an immediate consequence of the Substructure Theorem (and its corollary) in Sec. 3.

Theorem 14 *For each* ACP $\mathbf{F}$ *in signature* Σ (P-STREAM) *and for* $F = \mathbf{F}^{\mathcal{S}(\mathbb{N})}$ *we have:*

(i) $\mathcal{S}(A)$ *is closed under* F, *and*

(ii) $F \restriction \mathcal{S}(A) = \mathbf{F}^{\mathcal{S}(A)}$.

Thus for a recursion-theoretic description of ACPs over $\mathcal{S}(A)$ for $A \subseteq \mathbb{N}$ it is sufficient to describe the ACPs over $\mathcal{S}(\mathbb{N})$ and then form their restrictions. To simplify matters in doing so, we first replace $\mathcal{S}(\mathbb{N})$ by the structure

$$(4)\qquad \mathcal{E}(\mathbb{N}) = (\mathbb{N}, S(\mathbb{N}), Eval, Sim, Sc, Pd, 0, Eq_0),$$

where $Eval : S(\mathbb{N}) \times \mathbb{N} \overset{\sim}{\to} \mathbb{N}$ is given by $Eval(s,n) \simeq s(n)$. Every ACP over $\mathcal{S}(\mathbb{N})$ can be obtained as one over $\mathcal{E}(\mathbb{N})$ and conversely. For the former we use:

$$(5)\qquad \begin{array}{ll} \text{(i)} & Cons(a,s) = Sim(\lambda n.[\text{ if } n=0 \text{ then } a \text{ else } Eval(s, n\dot{-}1)]) \\ \text{(ii)} & Hd(s) \simeq Eval(s,0) \\ \text{(iii)} & T\ell(s) \simeq Sim(\lambda n.Eval(s,n+1)), \end{array}$$

while for the latter we use

$$(6)\qquad Eval(s,n) \simeq Term(s,n) \simeq Hd(T\ell^n(s)) \ .$$

Now, to see how the ACPs over $\mathcal{E}(\mathbb{N})$ work out, let us first return to the function and functional notation of Sec. 2. We have just two basic domains A_i to consider, namely $A_0 = \mathbb{N}$ and $A_1 = S(\mathbb{N})$. Hence the $A_{\bar{i}}$ are products $A_1^{\nu_1} \times A_0^{\nu_2}$, or $S(\mathbb{N})^{\nu_1} \times \mathbb{N}^{\nu_2}$. The type $\bar{i}$ is in this case written as $\bar{i} = 1^{\nu_1} \times 0^{\nu_2}$, and (s,x) is written for a typical element of $A_{\bar{i}}$, where $s = (s_1, \ldots, s_{\nu_1})$, $x = (x_1, \ldots, x_{\nu_2})$, and $\ell h(\bar{i}) = \nu = \nu_1 + \nu_2$. Next, the partial function types $\sigma = (\bar{i} \overset{\sim}{\to} j)$ reduce to those of the form $1^{\nu_1} \times 0^{\nu_2} \overset{\sim}{\to} 0$ or $1^{\nu_1} \times 0^{\nu_2} \overset{\sim}{\to} 1$. In the first case, a partial function $\varphi \in A_\sigma$ becomes a partial map $\varphi : S(\mathbb{N})^{\nu_1} \times \mathbb{N}^{\nu_2} \overset{\sim}{\to} \mathbb{N}$. Recall that at the end of Sec. 9 it was further required that φ preserves $\subseteq_S$ when applied to arguments in S. Hence:

(7) *For $\nu_1 > 0$, the partial functions $\varphi : S(\mathbb{N})^{\nu_1} \times \mathbb{N}^{\nu_2} \overset{\sim}{\to} \mathbb{N}$ of type level 1 which preserve $\subseteq_{S(\mathbb{N})}$ are identified with the monotonic partial functionals $\varphi : (\mathbb{N} \overset{\sim}{\to} \mathbb{N})^{\nu_1} \times \mathbb{N}^{\nu_2} \overset{\sim}{\to} \mathbb{N}$ of type level 2 over $\mathcal{N}$. In the case that $\nu_1 = 0$, φ remains of type level 1 over $\mathcal{N}$.*

Next, with each φ of type $\sigma = 1^{\nu_1} \times 0^{\nu_2} \overset{\sim}{\to} 1$ is associated φ^- of type $\sigma^- = 1^{\nu_1} \times 0^{\nu_2+1} \overset{\sim}{\to} 0$ by

$$(8)\qquad \varphi^-(s,x,n) \simeq Eval(\varphi(s,x),n) \ .$$

With φ required to preserve $\subseteq_{S(\mathbb{N})}$, the same holds for φ^{-}. Conversely, with each ψ of type $1^{\nu_1} \times 0^{\nu_2+1} \overset{\sim}{\to} 0$ is associated ψ^{+} of type $1^{\nu_1} \times 0^{\nu_2} \to 1$

(9) $$\psi^{+}(s,x) = \lambda n.\psi(s,x,n)\ .$$

Note that

(10) (i) $(\psi^{+})^{-} = \psi$, and

(ii) $\varphi \subseteq (\varphi^{-})^{+}$, with $\varphi(s,x) = (\varphi^{-})^{+}(s,x)$ when $\varphi(s,x)\downarrow$,

and $(\varphi^{-})^{+} =$ empty function when $\varphi(s,x)\uparrow$.

While by this we do not have a one-one match-up between A_σ and A_{σ^-}, for all computational purposes we can reduce A_σ to A_{σ^-}; the reason is that we are only concerned here with $\varphi(s,x)$ when $\varphi(s,x)\downarrow$. Thus:

(11) *For $\nu_1 > 0$, the partial functions $\varphi : S(\mathbb{N})^{\nu_1} \times \mathbb{N}^{\nu_2} \overset{\sim}{\to} S(\mathbb{N})$ of type level 1 over $\mathcal{E}(\mathbb{N})$ which preserve $\subseteq_{S(\mathbb{N})}$ are identified with the monotonic partial functionals $\varphi : (\mathbb{N} \overset{\sim}{\to} \mathbb{N})^{\nu_1} \times \mathbb{N}^{\nu_2+1} \overset{\sim}{\to} \mathbb{N}$ of type level 2 over $\mathcal{N}$. In the case that $\nu_1 = 0$, φ is identified with ψ of type level 1 over $\mathcal{N}$.*

It should now be clear how things will go with the functionals in our framework:

(12) *The monotonic partial functionals F of type level 2 over $\mathcal{E}(\mathbb{N})$ which have some partial function arguments that, in turn, have arguments in $S(\mathbb{N})$, are identified with monotonic partial functionals of type level 3 over $\mathcal{N}$. But if all partial function arguments of F have only numerical arguments, then such F are identified with monotonic partial functionals of type level 2 over $\mathcal{N}$.*

Thus for a recursion-theoretic interpretation of the ACPs over $S(\mathbb{N})$ or equivalently over $\mathcal{E}(\mathbb{N})$, we need an extension of the notion of partial recursiveness to functionals of type level 3 over $\mathcal{N}$. This is provided by the work of Eršov [2], which leads to a notion of *partial recursiveness for functionals of arbitrary finite type applied to hereditarily partial continuous* *arguments.* Eršov derives this via his theory of enumerated structures from an abstract theory of special kinds of topological spaces, called f-spaces. Subsequently, I presented in [3], [4] a direct concrete version of these notions which is analogous to that given for finite type functionals of hereditarily *total* continuous arguments in Kleene [10] and Kreisel [11]. This concrete version can be explained much more quickly than that due to Eršov. For simplicity, this is done for the *pure types* n where $(n+1) = (n \overset{\sim}{\to} 0)$. The idea is that objects of type $(n+1)$ are partial functions φ such that for each ψ of type n, the value of $\varphi(\psi)$ depends only on a finite amount of information about ψ. That information is represented by *formal neighborhoods.* The set of formal neighborhoods of type n, Nd^n, is defined inductively; we use letters $U^n, U_1^n, \ldots$ to range over members of Nd^n.

(13) (i) $Nd^0 = \mathbb{N}$

(ii) Nd^{n+1} consists of all finite sequences $\langle U_i^n, p_i\rangle_{i\leq m}$ such that $U_i^n \in Nd^n$ and $p_i \in \mathbb{N}$.

Then we define $\overset{n}{C^{\smile}}$ and $|U^n| \subseteq \overset{n_1}{C^{\smile}}$ inductively as follows, where we write $\varphi \in U^n$ for $\varphi \in |U^n|$; also superscripts are omitted when these are determined by the context.

(14) (i) $\overset{0}{C^{\smile}} = \mathbb{N}$ and $|p| = \{p\}$

(ii) $\overset{n+1}{C^{\smile}}$ is the set of all $\varphi : \overset{n_1}{C^{\smile}} \overset{\sim}{\to} \mathbb{N}$ such that

$$\forall\psi \in \overset{n_1}{C^{\smile}} \forall p[\varphi(\psi) = p \Rightarrow \exists U^n(\psi \in U \,\&\, \forall\chi \in U(\varphi(\chi) = p))] \ .$$

For $U^{n+1} = \langle U_i^n, p_i\rangle_{i\leq m}$,

$$|U^{n+1}| = \{\varphi | \forall_{i\leq m} \forall\psi \in U_i^n[\varphi(\psi) = p_i]\} \ .$$

Note that $\overset{1}{C^{\smile}}$ is the same as $\mathbb{N} \overset{\sim}{\to} \mathbb{N}$.

(15) $F : \overset{n_1}{C^{\smile}} \times \ldots \times \overset{n_k}{C^{\smile}} \overset{\sim}{\to} \mathbb{N}$ is said to be *partial recursively continuous*, and we write $F \in PR/C^{\smile}$, if

(i) F is continuous, i.e. whenever $F(\psi_1 \ldots, \psi_k) = p$ then

$$\exists U_1^{n_1}, \ldots, U_k^{n_k}[\psi_1 \in U_1 \& \ldots \& \psi_k \in U_k$$

$$\forall \chi_1, \ldots, \chi_k (\chi_1 \in U_1 \& \ldots \& \chi_k \in U_k \Rightarrow F(\chi_1, \ldots, \chi_k) = p), \text{ and}$$

(ii) there exists partial recursive $f : \mathbb{N}^k \overset{\sim}{\to} \mathbb{N}$ such that

$$F(\psi_1, \ldots, \psi_k) \simeq p \Leftrightarrow \exists U_1^{n_1}, \ldots U_k^{n_k}[\psi_1 \in U_1 \& \ldots \& \psi_k \in U_k \& f(U_1, \ldots, U_k) \simeq p] \ .$$

(In (ii), we assume formal neighborhoods coded by natural numbers.) By use of primitive recursive tupling functions, objects of type level n are represented as object of pure type n. The following result is a consequence of the work in [2], which is via a rather lengthy development through the theory of f-spaces. I plan to make available a much shorter direct proof for $PR/C^{\smile}$ as explained here.

Theorem 15 *$PR/C^{\smile}$ is closed under the extension of the* ACP *schemata to arbitrary finite types.*

Corollary.

(i) *The* ACPs *over $\mathcal{S}(\mathbb{N})$ are all partial recursively continuous.*

(ii) *If $F : S(\mathbb{N})^{\nu_1} \times \mathbb{N}^{\nu_2} \overset{\sim}{\to} \mathbb{N}$ with $\nu_1 > 0$ is an* ACP *over $\mathcal{S}(\mathbb{N})$, then F is a partial recursive functional in the usual sense; similarly for $F : S(\mathbb{N})^{\nu_1} \times \mathbb{N}^{\nu_2} \overset{\sim}{\to} S(\mathbb{N})$.*

(iii) *If $F \in S(\mathbb{N})$ is generated by the* ACP *schemata over $\mathcal{S}(\mathbb{N})$ then F is partial recursive.*

(iv) *The structure of partial recursive streams is closed under the* ACP *schemata over $\mathcal{S}(\mathbb{N})$.*

Remarks.

1. Re(iv): though in $\mathcal{S}(\mathbb{N})$, Sim is regarded as having domain $\mathbb{N} \overset{\sim}{\to} \mathbb{N}$, it can only be used in the schemata in the form $Sim(\lambda n.G(n, \ldots))$ where G is an ACP over $\mathcal{S}(\mathbb{N})$; thus it will only be applied to partial recursive function arguments.

2. If we expand $\mathcal{S}(\mathbb{N})$ by some specified $s_1, \ldots, s_m \in S(\mathbb{N})$ as imput data to $\mathcal{S}^+(\mathbb{N}) = (\mathcal{S}(\mathbb{N}), s_1, \ldots, s_m)$, then the results of the corollary hold for $\mathcal{S}^+(\mathbb{N})$ relative to $s_1, \ldots, s_m$.

3. In actual computation with streams, we cannot pass them as arguments or values in the extensional sense as partial functions, but can only deal with them via some method of representation, typically by Gödel numbers of partial recursive functions. Then partial recursive stream operations as in (ii) above are represented by *effective operations*, in the sense of Myhill & Shepherdson [17]. A basis for treating abstract computational procedures which would have direct such intensional interpretations was provided in [5], but its proposed application to streams there (op.cit. Sec. 12) needs to be corrected; that will be done elsewhere. Incidentally, the Myhill-Shepherdson theorem extends to $PR/C^{\smile}$ in all finite types.

4. The preceding remark can be relativized to any given input data streams such as come from external (*prima-facie*) non-recursive sources.

Appendix 1 Relation of ACPs to Moschovakis' FLR.

Moschovakis [15], [16] has developed a Formal Language of Recursion (FLR), which provides for any Σ a language of terms for computation procedures over structures of signature Σ. Besides the formation of terms by explicit definition, which correspond to the procedures obtained by our schemata I-VII, FLR features the formation of a term using simultaneous least fixed point (SLFP) recursion, to which the following scheme is analogous for appropriate combinations of types:

VIII$'$ (Simultaneous least fixed point)

$$\mathbf{F}(\varphi, x) \simeq [SLFP(\mathbf{G}_1, \ldots, \mathbf{G}_m) \text{ in } \mathbf{H}](\varphi, x),$$

interpreted in each structure $\mathcal{A}$ of signature Σ as follows, when $F = \mathbf{F}^{\mathcal{A}}, G_k = \mathbf{G}_k^{\mathcal{A}}(k = 1, \ldots, m), H = \mathbf{H}^{\mathcal{A}}$, with arguments $F(\varphi, x),\ G_k(\varphi, \psi_1, \ldots, \psi_m, x, y^{(k)}),\ H(\varphi, \psi_1, \ldots, \psi_m, x)$:

(1) $F(\varphi, x) \simeq H(\varphi, \widetilde{\psi}_1, \ldots, \widetilde{\psi}_m, x)$ where

$\widetilde{\psi}_1, \ldots, \widetilde{\psi}_m$ are the least simultaneous solution of the system of equations

$$\begin{cases} \psi_1(y^{(1)}) \simeq G_1(\varphi, \psi_1, \ldots, \psi_m, x, y^{(1)}) \\ \qquad\qquad\qquad \cdots \\ \psi_m(y^{(m)}) \simeq G_m(\varphi, \psi_1, \ldots, \psi_m, x, y^{(m)}). \end{cases}$$

Let ACP′(Σ) be the abstract computation procedures in signature Σ generated by I-VII plus VIII′ in place of VIII. Let ED(Σ) be the procedures for explicit definition i.e. those generated by the schemata I-VII without VIII (or VIII′).

Theorem

(i) ACP(Σ) is equivalent to ACP′(Σ) for each Σ.

(ii) ACP′(Σ) is equivalent to the procedures obtained by restricting VIII′ to $\mathbf{G}_1, \ldots, \mathbf{G}_m, \mathbf{H}$ in ED(Σ).

Proof. (Sketch) (i) Clearly the result of a single LFP by VIII can be treated as a special case of SLFP (VIII′), so ACP(Σ) is included in ACP′(Σ). Conversely, VIII′ can be obtained by a succession of single LFP's. This was proved in [5], §9; the idea is briefly as follows for $m = 2$, where we suppress the function and individual parameters (φ, x). To find the simultaneous LFP $\widetilde{\psi}_1, \widetilde{\psi}_2$ of

$$(.1) \qquad \begin{cases} \widetilde{\psi}_1(y^{(1)}) \simeq G_1(\psi_1, \psi_2, y^{(1)}) \\ \widetilde{\psi}_2(y^{(2)}) \simeq G_2(\psi_1, \psi_2, y^{(2)}), \end{cases}$$

define

$$(.2) \qquad K(\psi_1, y^{(2)}) \simeq [LFP(\lambda\psi_2\lambda z^{(2)}.G_2(\psi_1, \psi_2, z^{(2)}))](y^{(2)}) \ .$$

Then

$$(.3) \qquad \widetilde{\psi}_1(y^{(1)}) \simeq [LFP(\lambda\psi_1\lambda z^{(1)}.G_1(\psi_1, \lambda y^{(2)}K(\psi_1, y^{(2)})z^{(1)}))](y^{(1)})$$
$$\text{and } \widetilde{\psi}_2(y^{(2)}) \simeq K(\widetilde{\psi}_1, y^{(2)}) \ .$$

In other words, in the equation for ψ_1 we treat ψ_2 as a LFP uniformly in ψ_1.

(ii) The idea of the proof for this part is that the end result of two applications of VIII′ can be merged into a single application by combining the successive SLFPs into a single

SLFP. Hence any number of applications of VIII$'$ can be reduced to a single one preceded and followed only by explicit definitions.

It is the abstract computation procedures obtained as in (ii) which are directly analogous to the terms of FLR. Hence the two approaches yield the same class of procedures over each structure. The fact that the kind of definition VIII$'$ of SLFP can be reduced to a single application is an advantage of FLR over the use of successive applications of LFP via VIII. However, for the results in this paper, it would have been more complicated to work with VIII$'$ in place of VIII. This of course is irrelevant to Moschovakis' main purpose for FLR, which is his interesting proposal to use it to explain the intensional notion of *algorithm* and, via his normalization procedure (in [16]) for terms in FLR, the notion of *identity of algorithms.*

Appendix 2. Comparison with the work of Tucker and Zucker.

In a series of publications since 1988 (detailed in the references to [21]) J.V. Tucker and J.I. Zucker have explored various notions (or "models") of computation applied to rather general multi-sorted structures $\mathcal{A}$. In [21] they have extended these notions to structures $\overline{\mathcal{A}}$ for streams. In the same spirit as ours, that work treats computation on streams as a special chapter in their general theory; however, there are significant points of difference both as to the general approach and the special case. The following points initiate a comparison, but more work needs to be done to establish exact relationships.

1°. The general approach of Tucker and Zucker ('T-Z' in the following) is reviewed in [21] secs. 2, 3, which is what we follow here. That applies only to first-order structures $\mathcal{A}$. Moreover, in addition to assuming that a structure for the Booleans $\mathbb{B}$ is built in, it is also assumed there that a structure for the natural numbers $\mathbb{N}$ (equivalent to our $\mathcal{N}$) is built into $\mathcal{A}$. Thus, even for first-order structures, the T-Z approach is more restrictive than ours. There are two primary notions of computation studied for such structures: PR($\mathcal{A}$) and μPR($\mathcal{A}$). These make use of generalized schemata for primitive recursive functions extended in the second case by a scheme for the least number operator μ. (Note that both definitions by primitive recursion and by μ make essential use of the assumption that $\mathcal{N}$ is contained in $\mathcal{A}$). It is easy to see that for first-order $\mathcal{A}$ containing $\mathcal{N}$, every μPR partial computable function over $\mathcal{A}$ agrees with an ACP over $\mathcal{A}$. It is a question whether the converse is true. Also, there is no obvious comparison of PR($\mathcal{A}$) with a subset of the ACPs over $\mathcal{A}$.

2°. The T-Z approach also studies computation over structures $\mathcal{A}^\star$ associated with $\mathcal{A}$ as in 1°, where $\mathcal{A}^\star$ contains the domain $\mathcal{A}^\star$ of finite sequences (or "arrays") of elements of each A_i in $\mathcal{A}$, with the appropriate additional structure. Then PR($\mathcal{A}^\star$) and μPR($\mathcal{A}^\star$) determine, by restriction, notions of computation PR*($\mathcal{A}$) and μPR*($\mathcal{A}$), resp. Since $\mathcal{A}^\star$ is a first-order structure containing $\mathcal{N}$, we also have μPR($\mathcal{A}^\star$) $\subseteq$ ACP($\mathcal{A}^\star$). I conjecture that the reverse inclusion also holds, using the results here from Secs. 5–7 and using finite sequences to code (finite) computations in the LFP scheme.

3°. The notions of computations for streams studied in [21] apply to structures $\overline{\mathcal{A}}$ which contain with (some) A_i also the set $\overline{A_i} = (\mathbb{N} \to A_i)$. Thus, only infinite streams are treated there. The structure $\overline{\mathcal{A}}$ contains for each $\overline{\mathcal{A}}_i$ an evaluation function(al) $eval_i : \overline{A}_i \times \mathbb{N} \to A_i$. In addition to the schemata dealt with in the general situation (as described in 1°, 2° above), there is a special new scheme $\boldsymbol{\lambda}$, which allows one to pass from a function $g : D \times \mathbb{N} \to A_i$ to a function(al) $\lambda g : D \to \overline{A}_i$ in the canonical way. This leads to four new notions of computation: $\lambda PR(\overline{\mathcal{A}})$, $\lambda\mu PR(\overline{\mathcal{A}})$, $\lambda PR^\star(\overline{\mathcal{A}})$, $\lambda\mu PR^\star(\overline{\mathcal{A}})$. Our treatment of ACPs over stream structures in Sec. 10 extends in the obvious way to such structures $\overline{\mathcal{A}}$ and $\overline{\mathcal{A}}^\star$, when the appropriate second-order simulation functionals Sim_i are added; in order to make this explicit, we shall indicate the latter structures by $\overline{\mathcal{A}}[Sim]$ and $\overline{\mathcal{A}}^\star[Sim]$. Again it is easy to see that $\lambda PR(\overline{\mathcal{A}}) \subseteq$ACP ($\overline{\mathcal{A}}[Sim]$) and $\lambda\mu PR(\overline{\mathcal{A}}^\star) \subseteq$ACP($\overline{\mathcal{A}}^\star[Sim]$), since we have closure under the $\boldsymbol{\lambda}$ scheme using the Sim operators and the substitution scheme VII for ACPs. The interesting question here is whether all the stream operations obtained by ACPs over $\overline{\mathcal{A}}[Sim]$ or $\overline{\mathcal{A}}^\star[Sim]$ can be obtained by the $\lambda\mu PR$ schemes over $\overline{\mathcal{A}}$, resp. $\overline{\mathcal{A}}^\star$. This is perhaps possible in the latter case via the recursion-theoretic interpretation in Sec. 11. Finally, we remark that no recursion schemata distinctive for streams (as in Sec. 10 here) are studied in [21].

Appendix 3. Corrections to [5].

The following corrections are to be made to my paper [5].

p. 80, ℓ –10 and in ftn. 2, change '[4]' to '[3]' and '[5]' to '[4]'.

p. 83, (1). Using the notation of Sec. 6, assume given $=_{A_0}, =_{A_1}, \ldots, =_{A_n}$. Then change '$\varphi(x) = \psi(x)$' to '$\varphi(x) =_{A_j} \psi(x)$'.

p. 83, (3). Similarly, change '$F(\psi, x) = F(\varphi, x)$' to '$F(\psi, x) =_{A_j} F(\varphi, x)$'.

p. 83, (4). Similarly, change '$F_1(\varphi, x) = F_2(\varphi, x)$' to '$F_1(\varphi, x) =_{A_j} F_2(\varphi, x)$'.

(The need for these corrections on p. 83 op.cit. was brought to my attention by Scott Stoller.)

p. 93. The recursion-theoretic interpretation of computation on streams indicated in the next to the last paragraph of Sec. 12 op.cit. is incorrect as it stands. It is superseded by the work of Sec. 11 in the present paper combined with Remark 3 thereto. A direct treatment in terms of indices of partial recursive functions is also possible, by adapting the generalization of monotonicity and thence of our basic approach introduced here in the latter part of Sec. 9. Namely, we write $z \subseteq w$ if $\{z\}$ is a subfunction of $\{w\}$; then $sim^\star = \lambda z.z$ is trivially monotonic in the sense that $z \subseteq w \Rightarrow sim^\star(z) \subseteq sim^\star(w)$.

References

[1] H. Abelson, G.J. Sussman and J. Sussman, Structure and interpretation of computer programs, (M.I.T. Press, Cambridge, MA, 1985).

[2] Y.L. Eršov, Computable functionals of finite types, Algebra and Logic 11, (1972) 203–242, (translation from Algebra i Logika 11, No. 4 (1972) 367–437).

[3] S. Feferman, Inductive schemata and recursively continuous functionals, in: R.O. Gandy and J.M.E. Hyland, eds., Logic Colloquium, '76 (North-Holland, Amsterdam, 1977) 373–392.

[4] S. Feferman, Generating schemes for partial recursively continuous functionals, in: Colloque International de Logique, (Éditions de C.N.R.S., Paris, 1977) 191–198.

[5] S. Feferman, A new approach to abstract data types II. Computations on ADTs as ordinary computation, in: E. Börger, et al, eds., Computer Science Logic, Lecture notes in Computer Science, 626 (1991) 79–95.

[6] S. Feferman, A new approach to abstract data types I. Informal development, Mathematical Structures in Computer Science 2 (1992) 193–229.

[7] H. Geuvers, Inductive and co-inductive types with iteration and recursion, (notes of a talk at the BRA-LF meeting in Edinburgh, May 1991, (1992)).

[8] S.C. Kleene, Introduction to Metamathematics, (North-Holland, Amsterdam, 1952).

[9] S.C. Kleene, Recursive functionals and quantifiers of finite types I, Trans. Amer. Math. Soc. 91 (1959) 1–52.

[10] S.C. Kleene, Countable functionals, in: A. Heyting, ed., Constructivity in Mathematics, (North-Holland, Amsterdam 1959) 81–100.

[11] G. Kreisel, Interpretation of analysis by means of constructive functionals of finite types, in: A. Heyting, ed., Constructivity in Mathematics (North-Holland, Amsterdam 1959) 101–128.

[12] F. Leclerc and C. Paulin-Mohring, Programming with streams in *Coq* – a case study: The sieve of Eratasthenes, in: H. Barendregt and T. Nipkow, eds., Types for Proofs and Programs, Lecture Notes in Computer Science 806 (1994) 191–212.

[13] Z. Manna, Mathematical Theory of Computation, (McGraw-Hill, New York, 1974).

[14] N.P. Mendler, Recursive types and type-constraints in second-order lambda calculus, in: Second Annual Symposium on Logic and Computer Science, (IEEE Computer Society Press, 1987) 30–36.

[15] Y.N. Moschovakis, Abstract recursion as a foundation of the theory of recursive algorithms, in: M.M. Richter, et al., eds., Computation and Proof Theory, Lecture Notes in Mathematics 1104 (1984) 289–364.

[16] Y.N. Moschovakis, The formal language of recursion, J. Symbolic Logic 54 (1989) 1216–1252.

[17] J. Myhill and J. Shepherdson, Effective operations on partial recursive functions, Zeitschrift Math. Logik u. Grundlagen der Mathematik 1 (1955) 310–317.

[18] L. Paulson, Co-induction and co-recursion in higher-order logic, (unpublished paper, Computer Lab., Univ. of Cambridge, 1994).

[19] R.A. Platek, Foundations of Recursion Theory, Ph.D. Thesis, Stanford University, Stanford, CA (1966).

[20] V.Y. Sazonov, Degrees of parallelism in computations, in: Mathematical Foundations of Computer Science, Lecture Notes in Computer Science 45 (1976) 517–523.

[21] J.V. Tucker and J.I. Zucker, Computable functions on stream algebras, in: H. Schwichtenberg, ed. Proc. NATO Summer School in Proof and Computation, Marktoberdorf, 1993 (Springer-Verlag, Heidelberg), to appear.

[22] J. van Leeuwen, ed., Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (M.I.T. Press, Cambridge, MA and Elsevier, Amsterdam, 1990).

[23] J. Vuillemin, Proof Techniques for Recursive Programs, Ph.D. Thesis, Stanford University, Stanford, CA (1973).

[24] G. Winskel, The Formal Semantics of Programming Languages. An Introduction (M.I.T. Press, Cambridge, MA, 1993).