RECURSION

YIANNIS N. MOSCHOVAKIS

Department of Mathematics
University of California, Los Angeles
and *MPLA, University of Athens*

January 2006

CONTENTS

CHAPTER 1

RECURSIVE EQUATIONS

Recursive definitions and arguments about them permeate pure and applied mathematics, logic, and especially computer science, but the general theory of **recursion** is not customarily viewed as a separate field of inquiry. We will introduce it here with a few elementary examples, which illustrate some of its basic notions and applications and hint at our approach to it. Along the way, we will also review some useful, basic definitions and facts and set notation and terminology for the sequel.[1]

Example 1. The Fibonacci numbers. These are defined recursively by the equations

$$F_0 = 0,\ F_1 = 1,\ F_{n+2} = F_n + F_{n+1}, \tag{1-1}$$

so that $F_2 = 0 + 1 = 1$, $F_3 = 1 + 1 = 2$, etc. They have been studied extensively, partly because of some applications (which do not concern us here), but also because they provide a simple and elegant example of a recursive definition which is not immediately equivalent to an explicit one.

1.1. **Proposition.** *If $\rho = \frac{1}{2}(1+\sqrt{5})$ is the positive root of the quadratic equation $x + 1 = x^2$, then for all $n \geq 2$, $F_n \geq \rho^{n-2}$.*

PROOF is by induction on $n \geq 2$, the base cases $n = 2, 3$ being easily verified by direct computation. For the induction step,

$$F_{n+2} = F_n + F_{n+1} \tag{1-2}$$
$$\geq \rho^{n-2} + \rho^{n-1} \qquad \text{by ind. hyp.} \tag{1-3}$$
$$= \rho^{n-2}(1+\rho) \tag{1-4}$$
$$= \rho^{n-2}\rho^2 = \rho^n \qquad \text{because } \rho + 1 = \rho^2. \tag{1-5}$$

⊣

[1] Most readers of these notes will know most of the results in this chapter, so they should just skim through it on first reading and come back to it when they hit upon an unfamiliar notation or idea.

This simple proposition is typical of many in mathematics: a function $f : \mathbb{N} \to W$ on the natural numbers

$$\mathbb{N} = \{0, 1, \dots\}$$

is first defined by giving a general procedure for computing each value $f(n)$ from the sequence $(f(i) \mid i < n)$ of preceding values, i.e.,

$$\begin{aligned} F_n = &\text{ if } n = 0 \text{ then } 0 \\ &\text{else if } n = 1 \text{ then } 1 \\ &\text{else } F_{n-2} + F_{n-1} \end{aligned} \tag{1-6}$$

in the case of the Fibonacci sequence, or

$$f(n) = \Phi((f(0), \dots, f(n-1))) \tag{1-7}$$

in general. A **recurrence equation** of this form obviously determines a value $f(n)$ for each n, and so it serves as a **recursive definition** of the function f; and then we can establish that some proposition $P(n)$ about f holds of all n *by induction*, i.e., by showing the implication

$$(\forall i < n)P(i) \Longrightarrow P(n),$$

using (1-7). The method is used to define and establish the fundamental properties of familiar number theoretic functions (addition, multiplication, exponentiation, etc.), and it is the basic technique of elementary number theory, where it often provides the basic architecture for deep and difficult arguments far removed from the simplicity of this example.

Example 2. The merge-sort algorithm. Suppose L is a set with a fixed total ordering $\leq$ on it. A **string** (finite sequence, word)[2]

$$v = v_0 v_1 \cdots v_{n-1} = (v_0, \dots, v_{n-1}) \in L^*$$

is **sorted** (in non-decreasing order), if $v_0 \leq v_1 \leq \cdots \leq v_{n-1}$, and for each $u \in L^*$, $\mathrm{sort}(u)$ is the sorted "rearrangement" of u,

$$\begin{aligned} \mathrm{sort}(u) =_{\mathrm{df}} &\text{ the unique, sorted } v \in L^* \text{ such that for some} \\ &\text{bijection } \pi : \{0, \dots, n-1\} \rightarrowtail\!\!\!\!\!\twoheadrightarrow \{0, \dots, n-1\}, \\ &v = (u_{\pi(0)}, u_{\pi(1)}, \dots, u_{\pi(n-1)}). \end{aligned} \tag{1-8}$$

The efficient computation of $\mathrm{sort}(u)$ is of paramount importance in many computing applications, and the construction and analysis of sorting algorithms is a small cottage industry. We will consider here just one of these

[2]Sometimes we denote strings by simply listing their elements

$$v_0 v_1 \cdots v_{n-1} = (v_0, v_1, \dots, v_{n-1}),$$

especially when we think of them as "words" from some alphabet L of "symbols"; and in such cases, we often use "$\equiv$" to denote the equality relation on words, since "$=$" is often one of the symbols in the alphabet.

algorithms, which is easily expressed by a system of two, simple, recursive equations.

In dealing with strings, we use the following functions and notations:

$$|(u_0,\dots,u_{m-1})| = m \quad (\text{with } |\emptyset| = 0),$$
$$\text{tail}((u_0,\dots,u_{m-1})) = (u_1,\dots,u_{m-1}) \quad (\text{with } \text{tail}(\emptyset) = \emptyset),$$
$$(u_0,\dots,u_{m-1}) * (v_0,\dots,v_{n-1}) = (u_0,\dots,u_{m-1},v_0,\dots,v_{n-1}).$$

The merge-sort uses as a "subroutine" an algorithm for **merging** two strings, which is defined as follows.

1.2. **Proposition.** *The equation*

$$\begin{aligned}(1\text{-}9)\quad \text{merge}(w,v) = {} & \text{if } (|w|=0) \text{ then } v\\ & \text{else if } (|v|=0) \text{ then } w\\ & \text{else if } (w_0 \le v_0) \text{ then } (w_0) * \text{merge}(\text{tail}(w), v)\\ & \text{else } (v_0) * \text{merge}(w, \text{tail}(v))\end{aligned}$$

determines a value $\text{merge}(w,v)$ *for all strings* $w, v \in L^*$*, and if* w *and* v *are both sorted, then*

$$\text{merge}(w,v) = \text{sort}(w * v). \tag{1-10}$$

Moreover, the value $\text{merge}(w,v)$ *can be computed by successive applications of* (1-9)*, using no more than* $|w| + |v| \dot{-} 1$ *comparisons.*[3]

Proof. That (1-9) determines a function and that (1-10) holds are both trivial, by induction on $|w| + |v|$. For the comparison counting, notice first that (1-9) computes $\text{merge}(w,v)$ using no comparisons at all, if one of w or v is the empty sequence; if both $|w| > 0$ and $|v| > 0$, we make one initial comparison to decide whether $w_0 \le v_0$, and no more than $|w| + |v| - 2$ additional comparisons after that (by the induction hypothesis, in either case), for a total of $|w| + |v| - 1$. ⊣

We did not define precisely what it means *to compute* $\text{merge}(w,v)$ *by successive applications of* (1-9) (or *from* (1-9), as we will sometimes say), but the procedure is obvious; for example, when $L = \mathbb{N}$ with the natural ordering:

$$\begin{aligned}\text{merge}((3,1),(2,4)) &= (2) * \text{merge}((3,1),(4))\\ &= (2,3) * \text{merge}((1),(4))\\ &= (2,3,1) * \text{merge}(( ),(4))\\ &= (2,3,1,4).\end{aligned}$$

[3] *Arithmetic subtraction* is defined by $k \dot{-} i =$ if $k < i$ then 0 else $k - i$.

1.3. **Exercise.** Prove that if $x > v_0 > v_1 > \cdots > v_{n-1}$, then the computation of merge$((x), v)$ by (1-9) will require n comparisons.

For each sequence u with $|u| = m > 1$ and $k = \lfloor \frac{m}{2} \rfloor$ the integer part of $\frac{1}{2}|u|$, let:

$$\begin{aligned} \mathrm{half}_1(u) &=_{\mathrm{df}} (u_0, \ldots, u_{k-1}) \\ \mathrm{half}_2(u) &=_{\mathrm{df}} (u_k, \ldots, u_{m-1}), \end{aligned}$$

and for $|u| \leq 1$, set

$$\begin{aligned} \mathrm{half}_1(\emptyset) &=_{\mathrm{df}} \emptyset, & \mathrm{half}_2(\emptyset) &=_{\mathrm{df}} \emptyset, \\ \mathrm{half}_1((x)) &=_{\mathrm{df}} \emptyset, & \mathrm{half}_2((x)) &=_{\mathrm{df}} (x), \end{aligned}$$

so that in any case

$$u = \mathrm{half}_1(u) * \mathrm{half}_2(u)$$

and each of the two halves of u has length within 1 of $\frac{1}{2}|u|$. We will (occasionally) need the (real) base-2 logarithm of a number $m > 0$,

$$\ln_2 m =_{\mathrm{df}} \text{the unique real number } y \text{ such that } 2^y = m,$$

and (more often) its integer version,

$$\log_2 m =_{\mathrm{df}} \text{the least } k \text{ such that } m \leq 2^k.$$

It is clear that $\ln_2 2^k = \log_2 2^k = k$, and that in general,

$$\ln_2 m \leq \log_2 m < \ln_2 m + 1. \tag{1-11}$$

1.4. **Proposition.** *The sort function satisfies the equation*

$$\begin{aligned} \mathrm{sort}(u) = \text{if } |u| \leq 1 \text{ then } u \\ \text{else } \mathrm{merge}(\mathrm{sort}(\mathrm{half}_1(u)), \mathrm{sort}(\mathrm{half}_2(u))) \end{aligned} \tag{1-12}$$

and it can be computed from (1-9) *and* (1-12) *using no more than* $|u| \log_2 |u|$ *comparisons.*

PROOF. The validity of (1-12) is immediate, by induction on $|u|$. To prove the bound on comparisons, also by induction, note that it is trivial when $|u| \leq 1$, and suppose that $\log_2 |u| = k + 1$, so that (easily) both halves of u have length $\leq 2^k$. Thus, by the induction hypothesis and Proposition 1.2, we can compute sort(u) using no more than

$$k2^k + k2^k + 2^k + 2^k - 1 \leq (k+1)2^{k+1}$$

comparisons. ⊣

1.5. **Recursive equations and systems, (1)**. A **recursive equation** is (generally) an identity

$$f(u) = \Phi(u, f) \qquad (u \in U), \tag{1-13}$$

where $f : U \to W$ is some function and $\Phi : U \times (U \to W) \to W$ is a **functional**; and a **system of recursive equations** is a finite set

$$\begin{aligned} f_1(u_1) &= \Phi_1(u_1, f_1, \ldots, f_n) \\ &\vdots \\ f_n(u_n) &= \Phi_n(u_n, f_1, \ldots, f_n) \end{aligned} \tag{1-14}$$

of identities with the obvious restrictions on the domains and ranges of the functions f_i and the functionals Φ_i, so that they make sense. We can use a recursive equation (or a system) to *define* a function, if we can prove that it has exactly one solution, as is the case when it is a simple recurrence like (1-7). In this example, however, we did not use the system (1-9) and (1-12) as a definition, since the sorting function is defined directly and explicitly by (1-8); we used instead the fact that the functions merge(u, v) and sort(u) satisfy the system (1-9), (1-12) *to extract an algorithm* for computing sort(u), and then to derive a fundamental *complexity property* of that algorithm. This is another basic application of recursive equations.

Example 3. Dumb search. Suppose $R \subseteq \mathbb{N}$ is some set of natural numbers, and consider the recursive equation

$$p(m) = \text{if } m \in R \text{ then } m \text{ else } p(m+1). \tag{1-15}$$

It does not look much like a definition, since it defines $p(m)$ in terms of $p(m+1)$, and it may have many solutions—for example, if $R = \emptyset$, then every constant function $p : \mathbb{N} \to \mathbb{N}$ satisfies it. Still, we can extract a computation procedure from it as in the preceding examples. If, for instance, the smallest member of R is 4 and we use (1-15) to compute values of "the solution" as above, we get

$$p(0) = p(1) = p(2) = p(3) = p(4) = 4;$$

and if the next member of R is 7, then, similarly, $p(5) = p(6) = p(7) = 7$. In fact, when R is infinite, then (easily) (1-15) has only one solution, the function

$$p(m) = \text{the least } n \geq m \text{ such that } [m \in R]. \tag{1-16}$$

Now, it is natural to read (1-15) as defining this "canonical" solution, obtained by using it as a recipe for computations, even when R is not infinite. To make this precise and to characterize the preferred solution, we need to introduce partial functions.

1.6. A **partial function** $p : A \rightharpoonup B$ is any (total, ordinary) function $p : A \to B_\perp$, where

$$B_\perp = B \cup \{\perp\},$$

and $\perp$ (read "bottom") is some fixed object not in B. The idea is that $\perp$ is some kind of ideal element which stands for "the undefined," so that

$p(x) = \perp$ simply means that no value has been assigned to $p(x)$. We will be using several common notations in connection with partial functions, including:

$$p(x)\downarrow \iff_{\rm df} p(x) \neq \perp \iff p(x) \textbf{ converges} \text{ or } \textbf{is defined},$$
$$p(x)\uparrow \iff_{\rm df} p(x) = \perp \iff p(x) \textbf{ diverges} \text{ or } \textbf{is undefined}.$$

Finally, we let

$$(A \rightharpoonup B) = \{p \mid p : A \rightharpoonup B\}$$

be the set of all partial functions on A to B, and on $(A \rightharpoonup B)$ we define the relation

$$p \sqsubseteq q \iff (\forall x \in A)[p(x)\downarrow \Longrightarrow p(x) = q(x)]$$

which suggests interpreting each $p : A \rightharpoonup B$ as giving "partial information" about some total extension $\overline{p} : A \to B$, and that $p(x) = \perp$ means that no value $\overline{p}(x)$ *is determined by the approximation* p.[4]

A partial function $p : A \rightharpoonup B$ is **finite**, if its domain of convergence $\{x \in A \mid p(x)\downarrow\}$ is finite, and when it is, we let

$$|p| =_{\rm df} |\{x \in A \mid p(x)\downarrow\}| \tag{1-17}$$

be the number of elements in its domain. In particular, if

$$\emptyset(x) = \perp \quad (x \in A)$$

is the partial function which never converges, then $|\emptyset| = 0$.

Each partial function $p : A \rightharpoonup B$ has a natural extension to a function $\tilde{p} : A_\perp \to B_\perp$,

$$\tilde{p}(\perp) = \perp,\ x \in A \Longrightarrow \tilde{p}(x) = p(x) \quad (p : A \rightharpoonup B) \tag{1-18}$$

which in turn determines p uniquely. Composition of partial functions is defined using these so-called **strict liftups**: if $p : A \rightharpoonup B$, $q : B \rightharpoonup C$, then

$$(q \circ p)(x) =_{\rm df} \tilde{q}(p(x)) \quad (x \in A),$$

so that if $p(x)\uparrow$, then $q(p(x))\uparrow$. We think of p and $\tilde{p}$ as "two aspects" of the same object and will tend to blur the distinction between them, sometimes using the same symbol for both of them and letting context decide which is which.

---

[4] If $f : A \rightharpoonup B$, we call A the **input set** and B the **output set** of f; $\{x \in A \mid f(x)\downarrow\}$ is the **domain of convergence** of f and $\{f(x) \mid f(x)\downarrow\}$ is the **image** of f. This avoids confusion, especially between the input set and the domain of convergence of a partial function. Notice that if $f : A \rightharpoonup B$, $A \subseteq A'$ and $B \subseteq B'$, then $f : A' \rightharpoonup B'$, so that the input and output sets of f are not determined by f (if we view functions as sets of ordered pairs).

We will represent each relation $R \subseteq A$ on a set A by its **characteristic function**

$$\chi_R(x) = \begin{cases} \text{tt}, & \text{if } R(x), \\ \text{ff}, & \text{otherwise}, \end{cases} \tag{1-19}$$

where tt and ff are some once-and-for-all fixed objects representing "truth" and "falsity". By extension, a **partial relation** is any partial function $p : A \rightharpoonup \{\text{tt}, \text{ff}\}$.

1.7. **Exercise.** Show that the relation $\sqsubseteq$ is a *partial ordering* of the space $(A \rightharpoonup B)$, i.e., for all $p, q, r \in (A \rightharpoonup B)$,

(1) $p \sqsubseteq p$.
(2) $p \sqsubseteq q \;\&\; q \sqsubseteq r \Longrightarrow p \sqsubseteq r$.
(3) $[p \sqsubseteq q \;\&\; q \sqsubseteq p] \Longrightarrow p = q$.

1.8. **Recursive equations and systems, (2).** It is now natural to consider recursive equations and systems as in 1.5 but on partial functions:

$$p(u) = \Phi(u, p) \quad (p : U \rightharpoonup W), \tag{1-20}$$

and

$$\begin{aligned} p_1(u_1) &= \Phi_1(u_1, p_1, \ldots, p_n) \quad (p_1 : U_1 \rightharpoonup W_1) \\ &\vdots \\ p_n(u_n) &= \Phi_n(u_n, p_1, \ldots, p_n) \quad (p_n : U_n \rightharpoonup W_n) \end{aligned} \tag{1-21}$$

And with these definitions, we can prove that (1-16) (correctly understood) defines the least (partial) solution of the recursive equation (1-15).

1.9. **Proposition.** *For each set $R \subseteq \mathbb{N}$, the partial function*

$$\begin{aligned} \overline{p}(m) &= (\mu n \geq m)[n \in R] \\ &= \begin{cases} (\text{the least } n \geq m)[n \in R], & \text{if } (\exists n \geq m)[n \in R], \\ \perp, & \text{otherwise} \end{cases} \end{aligned}$$

satisfies the recursive equation (1-15)*, and it is the least solution of* (1-15) *in the following sense: if $q : \mathbb{N} \rightharpoonup \mathbb{N}$ and for all m,*

$$q(m) = \text{if } m \in R \text{ then } m \text{ else } q(m+1), \tag{1-22}$$

then $\overline{p} \sqsubseteq q$.

Proof. That $\overline{p}$ satisfies (1-15) is immediate, and it is a bit easier to see that it is the least solution by taking cases on whether R is finite or infinite.

Case 1. *R is infinite.* Now $\overline{p}$ is a total function, and it is the only solution of (1-15); because if q satisfies (1-22) and $\overline{p}(m) = m$, then $m \in R$ and so $q(m) = m$; while if $\overline{p}(m) = m + k$ for some $k > 0$, then by k applications of (1-22), $q(m) = q(m+1) = \cdots = q(m+k) = m + k = \overline{p}(m)$.

Case 2. *R is finite.* Let M be the least number greater than all the members of R, so that

$$\overline{p}(x)\downarrow \iff x < M,$$

and every q which satisfies (1-22) must agree with $\overline{p}$ below M, as in *Case* 1; this means that $\overline{p} \sqsubseteq q$, by the definition of the partial ordering $\sqsubseteq$ on the function space $(\mathbb{N} \rightharpoonup \mathbb{N})$. ⊣

In this example, the recursive equation (1-15) may have many solutions, when R is finite, but $\overline{p}$ stands out as the most natural one, and we will call it **the canonical solution** of (1-15). It is characterized by being the least solution, and it arises by reading (1-15) as determining a recipe for computations (an algorithm), rather than just a condition which a partial function may or may not satisfy.

Example 4. Recursion on the powerset. Quite often in mathematics we need *the least set* with certain properties, and we justify the existence of such a set by appealing to the following, simple result. Here

$$\mathcal{P}(\Omega) = \{X \mid X \subseteq \Omega\}$$

is the powerset operation, and a mapping (function) $\Phi : \mathcal{P}(\Omega) \to \mathcal{P}(\Omega)$ is **monotone** if it respects the inclusion relation,

$$X \subseteq Y \Longrightarrow \Phi(X) \subseteq \Phi(Y).$$

1.10. **Theorem.** *If $\Phi : \mathcal{P}(\Omega) \to \mathcal{P}(\Omega)$ is a monotone mapping on the powerset of a set Ω, then the intersection*

$$I = \mathbf{fix}(\Phi) = \bigcap\{X \subseteq \Omega \mid \Phi(X) \subseteq X\} \tag{1-23}$$

of all sets which are "pressed down" by Φ is the least (under $\subseteq$) solution of the recursive equation

$$X = \Phi(X), \tag{1-24}$$

i.e., it is a fixed point of Φ, and it is a subset of every fixed point of Φ. We say that I is defined by the **recursive equivalence**

$$x \in I \iff x \in \Phi(I). \tag{1-25}$$

PROOF. If $\Phi(X) \subseteq X$, then $I \subseteq X$, by the definition of I, and hence $\Phi(I) \subseteq \Phi(X)$, since Φ is monotone; thus $\Phi(I) \subseteq X$ for every X which is pressed down, and, by the definition of I,

$$\Phi(I) \subseteq I.$$

Now, by monotonicity again, we get from this that $\Phi(\Phi(I)) \subseteq \Phi(I)$, which means that $\Phi(I)$ is pressed down, and hence $I \subseteq \Phi(I)$, which together with the last displayed inclusion gives the required $\Phi(I) = I$. That I is a subset of every other solution of (1-24) is immediate from its definition, since every fixed point of Φ is pressed down. ⊣

1.11. **Recursive equations and systems, (3)**. We called (1-24) a recursive equation, although it is not, strictly speaking in the "functional form" (1-20), for the simple reason that it relates sets rather than functions. More generally, we will call any *fixed point equation*

$$x = \Phi(x) \qquad (x \in X, \Phi : X \to X) \tag{1-26}$$

a *recursive equation*, no matter what the set X or the function Φ; and we will also talk of *recursive systems*

$$\begin{aligned} x_1 &= \Phi_1(x_1, \ldots, x_n) \\ &\vdots \\ x_n &= \Phi_n(x_1, \ldots, x_n), \end{aligned} \tag{1-27}$$

where $x_i \in X_i$ and $\Phi_i : X_1 \times \cdots \times X_n \to X_i$ for $i = 1, \ldots n$. Notice that (1-20) can also be written in this general form using the λ-operator,

$$p = \lambda(u)\Phi(u, p) \quad (p : U \rightharpoonup W),$$

and the same can be done for the system in (1-21). Whether such equations and systems have solutions (or canonical solutions) depends on the circumstances, of course, and we will need to prove it for interesting equations and classes of equations which come up in applications; Theorem 1.10 is one such result.

1.12. **Exercise** (Function closure)**.** Suppose A is a set, $f_1, \ldots, f_k$ are functions on A of various arities, $f_i : A^{n_i} \to A$, $A_0 \subseteq A$ is a subset of A, and define $\pi : \mathcal{P}(A) \to \mathcal{P}(A)$ by

$$\pi(X) =_{\text{df}} A_0 \bigcup \{f_i(x_1, \ldots, x_{n_i}) \mid x_1, \ldots, x_{n_i} \in X, i = 1, \ldots, k\}.$$

Show that π is monotone on $\mathcal{P}(A)$, and its least fixed point $\overline{A}_0 = \mathbf{fix}(\pi)$ is the **closure** of A_0 under $f_1, \ldots, f_k$, i.e., the least set $I \subseteq A$ such that

$$A_0 \subseteq I \;\&\; (\forall i, x_1, \ldots, x_{n_i})[x_1, \ldots, x_{n_i} \in I \Longrightarrow f_i(x_1, \ldots, x_{n_i}) \in I].$$

Show also that for every $x \in \overline{A}_0$, either $x \in A_0$, or there is some f_i and $x_1, \ldots, x_{n_i} \in \overline{A}_0$ such that $x = f_i(x_1, \ldots, x_{n_i})$.

The syntactic categories (terms, formulas, programs, proofs, theorems etc.) of formal and programming languages are typically introduced by simple recursive definitions, which (when not entirely trivial) are justified by appealing to Exercise 1.12. Primarily to set up notation, we review here and in the problems some examples of this procedure.

1.13. **The Propositional Calculus**. In the succinct notation preferred by computer scientists, the formulas of the propositional calculus are defined by the scheme

$$A :\equiv p \mid \neg(A_1) \mid (A_1 \;\&\; A_2) \mid (A_1 \vee A_2) \mid (A_1 \to A_2)$$

where p is chosen from a specified list $\{\mathtt{v}_i \mid i = 0, 1, \dots\}$ of formal variables, A, A_1 and A_2 stand for formulas, and it is tacitly assumed that the remaining reserved symbols are all distinct, from each other and from the variables. The interpretation here is that the set Prop of propositional formulas is the least fixed point of the monotone mapping

$$\pi(X) = \{\mathtt{v}_i \mid i = 0, 1, \dots\} \cup \{\neg(A) \mid A \in X\} \cup \{(A_1 \,\&\, A_2) \mid A_1, A_2 \in X\}$$
$$\cup \{(A_1 \vee A_2) \mid A_1, A_2 \in X\} \cup \{(A_1 \to A_2) \mid A_1, A_2 \in X\},$$

in the powerset of the set Σ^* of strings from the *propositional alphabet*

$$\Sigma = \{\mathtt{v}_i \mid i = 0, 1, \dots\} \cup \{\neg, (,), \&, \vee, \to\};$$

in the notation of (1-25), Prop is the solution of the recursive equivalence

$$A \in \text{Prop} \iff A \in \pi(\text{Prop}). \tag{1-28}$$

Recursive definitions of syntax like this can be quite complex, and must often be accompanied by the proof of various properties of the syntactic objects, not always trivial. A key, first result is often a **Parsing Lemma**, see Problems x1.22 and x1.23.

Partly to use in the next example, but also because of its intrinsic interest, we introduce here a natural and useful extension of the notion of "string".

1.14. **Streams**. For any two disjoint sets M and A, an **M-stream from A** is a finite or infinite sequence from A, or a sequence of the form

$$u = (a_0, \dots, a_n, w) \quad (n \geq 0), \tag{1-29}$$

where $w \in M$. The finite or infinite sequences from A are called **divergent** or **non-terminating** streams, while u in (1-29) is a **convergent** or **terminating** stream which **returns the value** $w \in M$. In applications, A typically represents "acts" of some sort, and a stream stands for a sequence of executions of these acts, ending (perhaps) with the "return" of a value from M. We let

$$\text{Streams}(A)_M =_{\text{df}} \{u \mid u \text{ is an } M\text{-stream from } A\}. \tag{1-30}$$

In this chapter we will be concerned only with the simplest case, where $M = \{\mathbf{t}\}$ is a singleton and

$$\text{Streams}(A) = \text{Streams}(A)_{\{\mathbf{t}\}} \tag{1-31}$$

is a kind of *completion* of the set A^* of finite strings from A, Exercise 1.16. The (divergent) empty stream is the same as the empty string, i.e., $\emptyset$; it is distinct from the "empty" convergent stream $(\mathbf{t})$.

Streams carry the following, natural operations, some of them extending the corresponding operations on strings:

$$|u| = \begin{cases} \text{the length of } u, & \text{if } u \text{ is finite,} \\ \infty, & \text{if } u \text{ is infinite,} \end{cases}$$

$$\text{tail}(u) = \text{if } (|u| \leq 1) \text{ then } \emptyset \text{ else } \lambda(i)u_{i+1},$$

$$b \frown u = \text{prepend}(b, u) = \lambda(i)[\text{if } (i = 0) \text{ then } b \text{ else } u_{i-1}]$$

$$\text{head}(u) = \text{if } (|u| > 0) \text{ then } u_0 \text{ else } \perp,$$

Notice that convergent streams have length > 0, and that

$$\text{head} : \text{Streams}(A)_M \to (A \cup M)_\perp,$$

We also set

$$\text{Symbol}(u) \iff \text{lh}(u) = 2 \;\&\; (u)_1 = \mathbf{t} \quad (u \in \text{Streams}(A),$$

so that each $a \in A$ is represented by the stream $(a, \mathbf{t})$.

Finally, streams are partially ordered by the "initial segment" relation,

$$u \sqsubseteq v \iff_{\text{df}} (\forall i < |u|)[u_i = v_i]. \tag{1-32}$$

1.15. **Exercise.** Show that the binary relation $\sqsubseteq$ is a partial ordering of $\text{Streams}(A)_M$, as in 1.7 whose *maximal* points are the convergent streams.

1.16. **Exercise.** Show that every non-decreasing sequence of streams

$$u_0 \sqsubseteq u_1 \sqsubseteq \dots$$

has a **least upper bound** u, i.e., there is a unique stream

$$u = \lim_n u_n$$

such that

(1) For every n, $u_n \sqsubseteq u$.
(2) If v is any stream such that for all n, $u_n \sqsubseteq v$, then $u \sqsubseteq v$.

1.17. **Exercise.** Show that the operations $\text{tail}(u)$, $\text{prepend}(b, u)$ and the partial operation $\text{head}(u)$ are monotone, with respect to the initial segment partial ordering $\sqsubseteq$, i.e.,

$$u \sqsubseteq v \Longrightarrow \text{tail}(u) \sqsubseteq \text{tail}(v),$$

and similarly with prepend.

1.18. **Exercise.** Show that the recursive equation

$$u\,;v = \text{if } \text{head}(u) = \mathbf{t} \text{ then } v$$
$$\text{else } \text{prepend}(\text{head}(u), \text{tail}(u)\,;v)$$

on $\text{Streams}(A)$ has a unique solution, the operation of **sequential execution**. Show also that

$$\text{if } u \text{ is divergent, then } u\,;v = u,$$

and that ; is associative, i.e.,

$$u\,;(v\,;w) = (u\,;v)\,;w.$$

Example 5. Deterministic, imperative programming languages. Consider the following program, in a toy "command language" which we will soon make precise:

```
Program A:
    0. Read(X)                      5. goto 2
    1. Y:=1                         6. Print(Y)
    2. if (X=0) goto 6              7. Ring
    3. X:=X-1                       8. end
    4. Y:=Y+Y
```

In the obvious reading of the notation, `X` and `Y` are *number variables* which can hold a (natural) number, and the *assignment command* `Y:=` n stores the number n in `Y`; `Read(X)` requests input and stores the received number in `X`; `Ring` rings some bell; `Print(Y)` prints (in some standard format) the value stored in `Y`; `end` terminates execution; and the basic *conditional* and `goto` commands are self-explanatory. The algorithm expressed by the program `A` receives the input x, *initializes* `Y` to 1, and then successively decrements `X` and doubles `Y` until (the value stored in) `X` becomes 0, at which time it prints the value of `Y` (which is now 2^x), rings the bell and quits. Ultimately the program `A` defines a function which assigns to each input x a stream of "observable" acts,

$$\mathrm{den}(\mathtt{A})(x) = (\mathtt{Print}(2^x), \mathtt{Ring}, \mathbf{t}), \tag{1-33}$$

typically called the **denotation** of `A`; the sequence of "internal" (not observable) acts which in the end produce this result is the **computation** of `A` on the input x, and it can be made precise in many ways.

This task of interpretation is trivial for this simple program `A`, but we can use arithmetical operations, assignments, conditionals and "goto" commands to write much more complex programs, whose behavior is by no means obvious. The task of "program semantics" is to give general rules of interpretation for all programs of a given programming language, which make precise the "meanings" of programs and also help us to reason efficiently and rigorously about them. Our limited aim here is to see how recursive definitions come up naturally when we try to do this.

1.19. **Syntax of the programming language** L_1. The (arithmetical) **expressions** of L_1 are defined by the scheme

$$\mathtt{E} :\equiv \mathtt{X} \mid \mathtt{0} \mid \mathtt{1} \mid (\mathtt{E}_1 + \mathtt{E}_2) \mid (\mathtt{E}_1 - \mathtt{E}_2) \mid (\mathtt{E}_1 \times \mathtt{E}_2)$$

where `X` is any member of some specified, infinite set V of **variables**, e.g., the set of all strings of English letters.

The **commands** (or **statements**) of L_1 are similarly defined by the scheme

$$C :\equiv \texttt{X} := \texttt{E} \mid \texttt{if } (\texttt{E}_1 = \texttt{E}_2) \texttt{ then goto } j \mid \texttt{if EOF then goto } j$$
$$\mid \texttt{Ring} \mid \texttt{Read(X)} \mid \texttt{Print(E)} \mid \texttt{end}$$

where X is any variable, E, $\texttt{E}_1$, $\texttt{E}_2$ are expressions, and j is any number, in decimal notation.

Finally, a **program** of L_1 is a finite sequence of commands

$$\texttt{P} = (C_0, C_1, \dots, C_n)$$

of length $n + 1 > 0$, in which all the numbers which occur (in the conditional commands) are between 0 and n. Typically we name programs and we number the commands, as we did in Program A, for easy reading and reference, but none of this is necessary. We will also introduce some obvious abbreviations, e.g.,

$$\texttt{goto } j \text{ stands for } \texttt{if } (0 = 0) \texttt{ then goto } j,$$

for "unconditional" jumps.

EOF stands for "end-of-file", and the following program illustrates the meaning of the command `if EOF then goto` which did not occur in Program A.

```
Program B:
    0. Read(X)              3. goto 0
    1. if EOF goto 4        4. end
    2. Print(X)
```

Program B repeatedly requests input, which it simply prints out, and it could go on forever; to stop it, whatever agent is producing input (perhaps George, on the keyboard) signals that no more is coming (typically by hitting <Control>-Z, if it is George), which satisfies the EOF test and execution jumps to command 4 and ends.

1.20. **Exercise.** Write a program in L_1 which requests two inputs, m and n and prints out a 1 or a 0 accordingly as $m \geq n$ or $m < n$.

1.21. **Exercise.** Write a program in L_1 which requests two inputs, m and n; prints out a 0 and quits if $m < n$ or $n = 0$; and if $m \geq n \geq 1$, then prints out a q and an r such that $0 \leq r < n$ and $m = nq + r$.

We need to interpret—i.e., assign mathematical objects to—the expressions and the programs of L_1, in agreement, of course, with our intuitive, computational understanding of L_1, so that, for example, we can prove rigorously that the programs in Exercises 1.20 and 1.21 actually do what we designed them for. For the expressions, the job is easy: each of them

denotes a number, which depends, however, on some given interpretation of the variables.

1.22. **Semantics of the expressions of L_1.** A **store** (or **valuation**) is any function

$$\pi : V \to \mathbb{N}$$

which assigns a natural number to each variable, and each expression $\mathtt{E}$ defines a function

$$\mathrm{den}(\mathtt{E}) : \mathrm{Stores} \to \mathbb{N}$$

as follows:

$$\begin{aligned}
\mathrm{den}(\mathtt{X})(\pi) = \pi(\mathtt{X}),\ \mathrm{den}(\mathtt{0})(\pi) = 0,\ \mathrm{den}(\mathtt{1})(\pi) = 1 \\
\mathrm{den}((\mathtt{E}_1 + \mathtt{E}_2))(\pi) = \mathrm{den}(\mathtt{E}_1)(\pi) + \mathrm{den}(\mathtt{E}_2)(\pi) \\
\mathrm{den}((\mathtt{E}_1 - \mathtt{E}_2))(\pi) = \mathrm{den}(\mathtt{E}_1)(\pi) \dot{-} \mathrm{den}(\mathtt{E}_2)(\pi) \\
\mathrm{den}((\mathtt{E}_1 \times \mathtt{E}_2))(\pi) = \mathrm{den}(\mathtt{E}_1)(\pi) \cdot \mathrm{den}(\mathtt{E}_2)(\pi)
\end{aligned}$$

1.23. **States and computations.** A **state** for a program

$$\mathtt{P} = (C_0, \ldots, C_n)$$

is a triple

$$s = (i, \pi, u),$$

where i is the number of a command ("to be executed next"), and so $i \leq n$; π is a store; and u is a stream of natural numbers ("the part of the input not yet used up"). A state is *terminal* if program execution ends with it,

$$\mathrm{Terminal}(s) \iff C_i \equiv \mathtt{end}.$$

Execution of command C_i may change the state and may also produce an observable act, as described in Table 1, where i^+ is the label of "the next command in the program", i.e.,

$$i^+ = \text{if } (i < n) \text{ then } i + 1 \text{ else } n,$$

and $\pi\{\mathtt{X} := n\}$ is the **update** of the store π which changes it only on the variable $\mathtt{X}$:

$$\pi\{\mathtt{X} := n\}(\mathtt{Y}) = \begin{cases} n, & \text{if } \mathtt{Y} \equiv \mathtt{X}, \\ \pi(\mathtt{Y}) & \text{otherwise.} \end{cases} \tag{1-34}$$

The Table defines two natural functions associated with the program $\mathtt{P}$:

$$\begin{aligned}
\mathrm{next}(s) &= \text{the next state} \\
&= \text{the NEXT STATE entry for } s \text{ in the Table,} \\
\mathrm{act}(s) &= \text{the ACTION entry for } s \text{ in the Table,}
\end{aligned}$$

Command	Action	Next State
`X:=E`	*skip*	$(i^+, \pi\{\mathtt{X} := \mathrm{den}(\mathtt{E})(\pi)\}, u)$
`if (`$\mathtt{E}_1$`=`$\mathtt{E}_2$`) then goto` j	*skip*	if $(\mathrm{den}(\mathtt{E}_1)(\pi) = \mathrm{den}(\mathtt{E}_2))(\pi))$ then (j, π, u) else (i^+, π, u)
`if EOF then goto` j	*skip*	if $u = (\mathbf{t})$ then (j, π, u) else (i^+, π, u)
`Read(X)`	*skip*	$(i^+, \pi\{\mathtt{X} := \mathrm{head}(u)\}, \mathrm{tail}(u))$
`Ring`	*ring*	(i^+, π, u)
`Print(E)`	$print(\mathrm{den}(\mathtt{E})(\pi))$	(i^+, π, u)
`end`	*skip*	(i, π, u)

Table 1. Transition and action of program `P` at state (i, π, u).

where act(s) is *skip*, *ring* or *print*(n) for some number n. Finally, the **computation** of `P` starting from a state s is the unique stream of states

$$\mathrm{comp}(\mathtt{P})(s) = (s_0, s_1, \dots) \tag{1-35}$$

of length at least 2, such that:

(1) $s_0 = s$.
(1) If s_n is terminal, then $s_{n+1} = \mathbf{t}$.
(3) If s_n is not terminal, then $s_{n+1} = \mathrm{next}(s_n)$.

Note that

$$\mathrm{comp}(\mathtt{P}) : \mathrm{States}(\mathtt{P}) \to \mathrm{Streams}(\mathrm{States}(\mathtt{P})),$$

where States(`P`) is the set of states of `P`. In using all these functions and relations, we will normally skip the explicit reference to `P`, when it is clear from the context which program we are talking about.

1.24. **Exercise.** Prove that comp(`P`)(s) is always a convergent or infinite, divergent stream, and give examples of convergent and divergent computations of Program `B`.

Recall the standard notation for the *iterates* of a function $\tau : X \to X$:

$$\tau^0(s) = s,$$
$$\tau^{n+1}(s) = \tau(\tau^n(s)).$$

1.25. **Lemma.** *For each state s of a program* `P`,

$$\mathrm{comp}(\mathtt{P})(s) = \text{if Terminal}(s) \text{ then } (s, \mathbf{t}) \text{ else } s \frown \mathrm{comp}(\mathtt{P})(\mathrm{next}(s)).$$

PROOF. Fix the program P. Directly from the definition,

$$\text{Terminal}(s) \Longrightarrow \text{comp}(s) = (s, \mathbf{t}),$$

and if s is not terminal, then

$$\text{comp}(s) = (s, \text{next}^1(s), \text{next}^2(s), \dots),$$

and comp(s) converges and has length

$$|\,\text{comp}(s)| = (\mu n)\text{Terminal}(\text{next}^n(s)) + 2$$

exactly if some next$^n(s)$ is terminal. Applying this to next(s), we get

$$\text{comp}(\text{next}(s)) = (\text{next}^1(s), \text{next}^2(s), \text{next}^3(s), \dots),$$

from which it follows that

$$\neg\text{Terminal}(s) \Longrightarrow \text{comp}(s) = s \frown \text{comp}(\text{next}(s))$$

and completes the proof of the Lemma. ⊣

1.26. **Theorem.** *For each program* P*, the function* comp(P) *is the unique solution of the recursive equation*

$$f(s) = \text{if Terminal}(s) \text{ then } (s, \mathbf{t}) \text{ else } s \frown f(\text{next}(s)). \tag{1-36}$$

PROOF. It suffices, by the Lemma, to prove that (1-36) has at most one solution.

First we observe that it f is any solution of (1-36), then

$$\neg\text{Terminal}(s) \Longrightarrow |f(s)| = |f(\text{next}(s))| + 1. \tag{1-37}$$

Suppose now that f_1 and f_2 both satisfy (1-36): we will complete the proof by showing by induction on n that

$$(\forall s)[n < |f_1(s)| \Longrightarrow f_1(s)_n = f_2(s)_n].$$

This is immediate at the basis, since

$$f_1(s)_0 = s = f_2(s)_0,$$

and it is also true for all n when s is terminal, since then

$$f_1(s) = (s, \mathbf{t}) = f_2(s).$$

Finally, if s is not terminal and $n + 1 < |f_1(s)|$, then

$$\begin{aligned} f_1(s)_{n+1} &= f_1(\text{next}(s))_n && \text{by (1-36) for } f_1 \\ &= f_2(\text{next}(s))_n && \text{by ind. hyp.} \\ &= f_2(s)_{n+1} && \text{by (1-36) for } f_2, \end{aligned}$$

where the induction hypothesis applies in the crucial step because either next(s) is terminal, in which case $f_1(\text{next}(s)) = f_2(\text{next}(s))$, outright, or it is not, and then by (1-37)

$$n + 1 < |f_1(s)| \Longrightarrow n < |f_1(\text{next}(s))|. \tag{1-38}$$

⊣

1.27. **Denotational semantics for L_1.** The **denotation** of a program P is the (possibly divergent, perhaps empty) stream of observable acts which it produces, as a function of an arbitrary input stream of integers u, so

$$\mathrm{den}(\mathtt{P}) : \mathrm{Streams}(\mathbb{N}) \to \mathrm{Streams}(A),$$

where A is the set of *ring* and *print*(n) acts. Execution starts on the **initial state**

$$s_{in}(u) = (0, \pi_{in}, u), \tag{1-39}$$

where $\pi_{in}(\mathtt{X}) = 0$ for every variable X. Thus, the only relevant computation is the one which starts on $s_{in}(u)$, but it is useful to associate a denotation $\mathrm{den}(\mathtt{P})(s)$ with every state, and finally set

$$\mathrm{den}(\mathtt{P})(u) = \mathrm{den}(\mathtt{P})(s_{in}(u)). \tag{1-40}$$

Finally, $\mathrm{den}(\mathtt{P})(s)$ is just the stream of *ring* and *print*(n) acts "executed" by the computation $\mathrm{comp}(\mathtt{P})(s)$, i.e.,

$$\mathrm{den}(\mathtt{P})(s) = h(s_0) * h(s_1) * \cdots$$

where $*$ is the operation of concatenation on strings and

$$h(s) = \begin{cases} (\mathbf{t}) & \text{if } s \text{ is terminal,} \\ (\ ) & \text{otherwise, if } \mathrm{act}(s) = \mathit{skip}, \\ (\mathrm{act}(s)) & \text{otherwise.} \end{cases} \tag{1-41}$$

Together with Lemma 1.25, this implies that

$$\neg\mathrm{Terminal}(s) \Longrightarrow \mathrm{den}(\mathtt{P})(s) = h(s) * \mathrm{den}(\mathtt{P})(\mathrm{next}(s)), \tag{1-42}$$

which leads us to the fundamental recursive equation for the denotation function $\mathrm{den}(\mathtt{P})$.

1.28. **Theorem.** *For each program* P, $\mathrm{den}(\mathtt{P})$ *is the least solution of the recursive equation*

$$g(s) = \begin{cases} (\mathbf{t}) & \text{if } \mathrm{Terminal}(s), \\ g(\mathrm{next}(s)) & \text{otherwise, if } \mathrm{act}(s) = \mathit{skip}, \\ \mathrm{act}(s) \frown g(\mathrm{next}(s)) & \text{otherwise, if} \\ & \mathrm{act}(s) = \mathit{ring} \text{ or } \mathrm{act}(s) = \mathit{print}(n). \end{cases} \tag{1-43}$$

PROOF. That $\mathrm{den}(\mathtt{P})$ satisfies (1-43) is immediate from its definition when s is terminal and from (1-42) when it is not. To prove its minimality, notice that if g satisfies (1-43), then for every s,

$$g(s) = h(s) * g(\mathrm{next}(s)); \tag{1-44}$$

if we apply this n times, for any $n < |\,\mathrm{comp}(s)|$: we get

$$g(s) = h(s_0) * h(s_1) * \ldots h(s_{n-1}) * g(s_n),$$

which implies $h(s_0) * h(s_1) * \ldots h(s_{n-1}) \sqsubseteq g(s)$, and, in the limit yields the required $\mathrm{den}(\mathtt{P})(s) \sqsubseteq g(s)$. ⊣

1.29. **Exercise.** Give an example of a program P for which equation (1-43) has many solutions.

The language L_1 is very simple, and so it might appear that the fussy distinction between *computations* and *denotations* and the recursive characterizations of the functions comp(P) and den(P) do not contribute much to our understanding of it. On the other hand, such characterization of comp(P) and den(P) as, respectively, the unique and least solutions of **two related recursive equations** is part of the basic methodology in the theory of deterministic programming languages.

Next we look (very briefly) at one, simple example of **non-deterministic** languages, where the method is severely tested.

Example 6. Do this or that; non-deterministic recursion. The language L_2 is obtained by adding the non-deterministic "autonomous choice" construct or, so that its expressions are those of L_1 and its commands are defined by

$C :\equiv \mathtt{X} := \mathtt{E} \mid \mathtt{if}\ (\mathtt{E}_1 = \mathtt{E}_2)\ \mathtt{then\ goto}\ j \mid \mathtt{if\ EOF\ then\ goto}\ j \mid \mathtt{goto\ i\ or\ j}$
$\mid \mathtt{Ring} \mid \mathtt{Read(X)} \mid \mathtt{Print(E)} \mid \mathtt{end}$

For example:

```
Program C:
    0. goto 1 or 3
    1. Print(0)          3. Print(1)
    2. goto 0            4. goto 0
```

In the most natural understanding of these commands, the execution of C starts with a choice, after which it prints a 0 or a 1 and returns to the initial instruction 0, to make another choice, and so on, indefinitely: the execution never terminates, and produces some arbitrary infinite, binary sequence, depending on the choices made at each execution of the choice command 0. There are many "computations"—infinite streams in this case—which produce many outputs, and so it is natural to identify *the denotation* of C with the set of all infinite, binary sequences,

(1-45) $$\mathrm{den}(\mathtt{C}) = (\mathbb{N} \to \{0, 1\}).$$

To give precise semantics for L_2, we add to Table 1 the additional line

goto j or j	*skip*	(i, π, u) or (j, π, u)

which describes the (trivial) action and two, possible transitions for the new command. It is no longer the case that for each program P, each state

s has a unique next state s', but there is an obvious *relation* between states

$$\text{next}(\mathtt{P})(s,s') \iff s' \text{ is a possible next state to } s;$$

and for each state s, we have an associated set of computations of P,

$$\text{comp}(\mathtt{P})(s) = \{(s_0, s_1, \dots) \mid s_0 = s \;\&\; (\forall i)[\text{next}(\mathtt{P})(s_i, s_{i+1})]\}.$$

Finally, the denotations of programs are also sets of streams:

$$\text{den}(\mathtt{P})(s) = \{h(s_0) * h(s_1) * \cdots \mid (s_0, s_1, \cdots) \in \text{comp}(\mathtt{P})(s)\},$$

where h is defined by (1-41) above. Notice that both $\text{comp}(\mathtt{P})(s)$ and $\text{den}(\mathtt{P})(s)$ are *non-empty sets of streams*, although there are obvious cases where the denotation is the singleton of the empty stream

$$\text{den}(\mathtt{P})(s) = \{\emptyset\},$$

for example in the program with the single instruction

(D) $$0. \qquad \mathtt{goto}\ 0$$

If we try to get recursive, fixed-point characterizations of these set-valued functions $\text{comp}(\mathtt{P})$ and $\text{den}(\mathtt{P})(s)$, however, we encounter problems. For the simple program C (and skipping the state s, since the executions of P do not depend on the state), we are naturally led to the equation

(1-46) $$x = (0 \frown x) \cup (1 \frown x) \qquad (\emptyset \neq x \subseteq \text{Streams}(\{0,1\})),$$

where the prepend operation on sets of streams is defined in the obvious way, by **distribution**,

$$b \frown x = \{b \frown u \mid u \in x\}.$$

Now the correct denotation (1-45) of C satisfies this equation, but it is not "the least solution" of (1-46) under any plausible definition of *least* because of the following:

1.30. **Exercise.** Show that for $i = 0, 1$, the set of binary streams

$$x_i = \{u \in \text{Streams}(\{0,1\}) \mid \text{for all sufficiently large } n, u_n = i\}$$

satisfies (1-46), while

$$x_0 \cap x_1 = \emptyset.$$

At this point, one might observe that $\text{den}(\mathsf{C})$ is *the largest* (under $\subseteq$) solution of (1-46), so perhaps the denotations of L_2 can be characterized using largest fixed points—which sounds initially plausible, since these (at least) exist in $\mathcal{P}(\text{Streams}(\text{acts})) \setminus \{\emptyset\}$ by Example 4. But Program D above eliminates this possibility, since the recursive equation that one would naturally associate with its denotation is the trivial

(1-47) $$x = x,$$

whose largest solution is $(\mathbb{N} \to \{0,1\})$, while the implementations give $\mathrm{den}(\mathtt{D}) = \{\emptyset\}$, which is actually the least solution of equation (1-47) in $\mathcal{P}(\mathrm{Streams}(\mathrm{acts})) \setminus \{\emptyset\}$.

It is actually possible—and *useful*—to characterize the denotations of L_2 by *canonical solutions* of the natural recursive equations associated with its programs, but this requires some non-trivial foundational analysis of nondeterministic recursion, and the development of some interesting mathematics. We consider some of the relevant facts in Problems x1.31 – *x1.32.

Problems for Chapter 1

x1.1. Prove that if $\rho = \frac{1}{2}(1+\sqrt{5})$ and $\sigma = \frac{1}{2}(1-\sqrt{5})$ are the roots of the quadratic $x+1 = x^2$, then

$$F_n = \tfrac{1}{\sqrt{5}}(\rho^n - \sigma^n) \quad (n \in \mathbb{N}).$$

x1.2. Addition on the natural numbers may be defined by the recursion

$$\begin{aligned} m + 0 &= m, \\ m + Sn &= S(m+n), \end{aligned} \tag{1-48}$$

where $Sn = n+1$ is the successor operation. Prove that it is *associative*,

$$k + (m+n) = (k+m) + n$$

and *commutative*,

$$m + n = n + m.$$

HINT: The idea is to use only the equations (1-48) in these proofs. To show commutativity you will need to prove the proposition

$$(\forall m)(\forall n)[m + n = n + m]$$

by induction on m; and both the basis : $(\forall n)[0 + n = n + 0]$ and the induction step : $(\forall n)[Sm+n = n+Sm]$ of this proof are shown by induction on n. (Note that to show $Sm + Sn = Sn + Sm$ in the induction step of the induction on n, within the induction step of the "master induction" on m, you have two *induction hypotheses* available: you may use $(\forall k)[m + k = k + m]$, and also $Sm + n = n + Sm$. This is a typical example of "proof by double induction.")

Recall the definition of the *greatest common divisor* of two natural numbers:

$$\gcd(m,n) = \text{ the largest } d \text{ such that } d \,|\, m \text{ and } d \,|\, n \quad (m \geq n \geq 1) \tag{1-49}$$

where

$$\begin{aligned} \mathrm{rm}(m,n) &= \text{ the unique } r < n \text{ such that, for some } q,\ m = nq + r, \\ n \,|\, m &\iff \mathrm{rm}(m,n) = 0. \end{aligned}$$

The *Euclidean algorithm* for computing $\gcd(m,n)$, first described in Book VII of the *Elements*, is (perhaps) the most ancient and still one of the most important algorithms in mathematics. Here we consider the "iterated division" rather than the "iterated subtraction" version of the algorithm.

x1.3 (**The Euclidean algorithm**). Prove that the function $\gcd(m,n)$ satisfies the recursive equation

$$\gcd(m,n) = \text{if } n \mid m \text{ then } n \qquad (m \geq n \geq 1) \\ \text{else } \gcd(n, \mathrm{rm}(m,n)), \tag{1-50}$$

and use (1-50) to compute $\gcd(231,165)$ by repeated division.

HINT: Show that the pairs $\{m,n\}$ and $\{n, \mathrm{rm}(m,n)\}$ have exactly the same common divisors.

x1.4 (**Bezout 's Lemma**). Prove that there exist functions

$$\alpha, \beta : \mathbb{N} \times \mathbb{N} \to \mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$$

which satisfy the following system of recursive equations, for $m \geq n \geq 1$:

$$\begin{aligned} \alpha(m,n) &= \text{ if } (n \mid m) \text{ then } 0 \text{ else } \beta(n, \mathrm{rm}(m,n)), \\ \beta(m,n) &= \text{ if } (n \mid m) \text{ then } 1 \\ &\qquad \text{else } \alpha(n, \mathrm{rm}(m,n)) - \mathrm{quot}(m,n)\beta(n, \mathrm{rm}(m,n)), \end{aligned}$$

where, for $m \geq n \geq 1$,

$$\mathrm{quot}(m,n) = \text{the unique } q \text{ such that for some } r < n, m = nq + r$$

is the (natural number) quotient of m by n. Show also that if α and β satisfy this system, then

$$\gcd(m,n) = \alpha(m,n)m + \beta(m,n)n,$$

and use this to express $\gcd(231,165)$ as an integer, linear combination of 231 and 165.

x1.5. Prove that for all $m \geq n \geq 1$, there are infinitely many choices of rational integers α and β such that

$$\gcd(m,n) = \alpha m + \beta n,$$

and give an algorithm which finds the pair with least $\alpha \geq 0$.

To measure the (most natural) complexity of the computation of $\gcd(m,n)$ by the Euclidean algorithm, let

$$c(m,n) = \text{the number of divisions required to compute} \\ \gcd(m,n) \text{ from (1-50)} \quad (m \geq n \geq 1), \tag{1-51}$$

so that, directly from (1-50),

$$c(m,n) = \text{ if } (n \mid m) \text{ then } 1 \text{ else } 1 + c(n, \mathrm{rm}(m,n)). \tag{1-52}$$

x1.6. Prove that for the Fibonacci sequence $(F_k \mid k \in \mathbb{N})$ defined in (1-1) and all $k \geq 2$, $\gcd(F_{k+1}, F_k) = 1$ (i.e., F_k and F_{k+1} are relatively prime) and $c(F_{k+1}, F_k) = k - 1$.

*x1.7 (**Lamé's Lemma**). Prove that if $n \leq F_k$ (the kth Fibonacci number) with $k \geq 2$, then, for every $m \geq n$, $c(m, n) \leq k - 1$.

HINT: Use induction on $k \geq 2$, checking separately the two basis cases $k = 2, 3$; the analysis required for $k = 3$ suggests the induction argument for all $k + 2$, with $k \geq 2$.

*x1.8. Prove that if $m \geq n \geq 2$, then

$$c(m, n) \leq 2 \ln_2(n),$$

where ρ is the positive root of $x + 1 = x^2$. (This says that the number of divisions required to compute $\gcd(m, n)$ by the Euclidean algorithm grows only logarithmically with n.)

In the next two problems we define and analyze a simple algorithm for sorting, which is much less efficient than the merge-sort.

x1.9. Prove that the equation

$$\begin{aligned} \text{(1-53)} \qquad \text{insert}(x, u) = {} & \text{if } (|u| = 0) \text{ then } (x) \\ & \text{else if } x \leq u_0 \text{ then } (x) * u \\ & \text{else } (u_0) * \text{insert}(x, \text{tail}(u)) \end{aligned}$$

determines a value $\text{insert}(x, u) \in L^*$ for any $x \in L$ and $u \in L^*$, and if u is sorted, then

$$\text{(1-54)} \qquad \text{insert}(x, u) = \text{sort}((x) * u).$$

Moreover, $\text{insert}(x, u)$ can be computed from (1-53) using no more than $|u|$ comparisons.

x1.10 (**The insert-sort algorithm**). Prove that the sort function satisfies the equation

$$\begin{aligned} \text{(1-55)} \qquad \text{sort}(u) = {} & \text{if } |u| \leq 1 \text{ then } u \\ & \text{else insert}(u_0, \text{sort}(\text{tail}(u))), \end{aligned}$$

and can be computed from (1-55) and (1-53) using no more than $\frac{1}{2}|u|(|u|-1)$ comparisons. Illustrate the computation with some examples, and show also that if u is inversely ordered, then this computation of $\text{sort}(u)$ requires exactly $\frac{1}{2}|u|(|u| - 1)$ comparisons.

To see the difference between the merge-sort and the insert-sort, note that when $|u| = 64 = 2^6$, then the insert-sort may need as many as 2016 comparisons, while the merge-sort will need no more than 384. On the other hand, as the next two problems show, there is nothing wrong with

the idea of sorting by repeated inserting—it is only that (1-53) expresses a very inefficient algorithm for insertion.

*x1.11 (**Binary insertion**). Prove that the equation

$$\begin{aligned}(1\text{-}56)\quad \mathrm{binsert}(x,u) = {} & \text{if } (|u|=0) \text{ then } (x)\\ & \text{else if } (x \le \mathrm{half}_2(u)_0)\\ & \qquad \text{then } \mathrm{binsert}(x,\mathrm{half}_1(u)) * \mathrm{half}_2(u)\\ & \text{else } \mathrm{half}_1(u) * \mathrm{half}_2(u)_0 \frown \mathrm{binsert}(x,\mathrm{tail}(\mathrm{half}_2(u)))\end{aligned}$$

determines a value $\mathrm{binsert}(x,u) \in L^*$ for any $x \in L$ and $u \in L^*$, and if u is sorted, then

$$\mathrm{binsert}(x,u) = \mathrm{insert}(x,u) = \mathrm{sort}((x) * u).$$

Moreover, $\mathrm{binsert}(x,u)$ can be computed from (1-56) using (for $|u|>0$) no more than $b(|u|)$ comparisons, where

$$b(m) = \begin{cases} \log_2 m + 1, & \text{if } m \text{ is a power of 2,}\\ \log_2 m, & \text{otherwise.}\end{cases}$$

*x1.12 (**Binary-insert-sort**). Prove that the sort function satisfies the equation

$$\begin{aligned}(1\text{-}57)\qquad \mathrm{sort}(u) = {} & \text{if } |u| \le 1 \text{ then } u\\ & \text{else } \mathrm{binsert}(u_0, \mathrm{sort}(\mathrm{tail}(u))),\end{aligned}$$

and can be computed from (1-57) and (1-56) using no more than $s(|u|)$ comparisons, where for $m>0$,

$$(1\text{-}58)\qquad s(m) = \ln_2((m-1)!) + (m-1) \le \log_2((m-1)!) + (m-1).$$

*x1.13. For the function $s(m)$ defined in (1-58), prove that

$$\lim_{m\to\infty} \frac{s(m)}{\log_2(m!)} = 1.$$

By Stirling's formula,

$$\lim_{m\to\infty} \frac{m\log_2 m}{\log_2(m!)} = 1,$$

and so the merge-sort and the binary-insert-sort algorithms are asymptotically equally efficient, from the point of view of the required number of comparisons. In fact, they are asymptotically best possible, but without a general definition of "algorithm" we can only prove this fact for specific models of computation, e.g., Turing machines:

*x1.14 (**Lower bound for sorting**). A Turing machine M with an oracle is a *Turing sorter for sequences of length* $n \geq 2$, if for every ordering $\leq$ on the set $\mathbb{N}_n = \{0, 1, \ldots, n-1\}$, if the oracle answers every question "*is* $i \leq j$?" during the computation of M truthfully for the given ordering, then the computation terminates and the numbers $0, 1, \ldots, n-1$ are printed on the tape as ordered by $\leq$. Prove that every such Turing sorter will call the oracle for at least $\log_2(|n|!)$ times, for at least one ordering of $\mathbb{N}_n$.

HINT: The computation of M proceeds deterministically until the first question to the oracle, after which it may split depending on whether the answer is "yes" or "no"—and then it may split again on the second question, along each of these paths, etc. Consider the tree of computations which is constructed in this way, and argue that it must have exactly $|n|!$ terminating branches, one for each ordering of $\mathbb{N}_n$; now use the fact that if the largest number of oracle calls along a single branch is k, then the number of distinct, terminating branches cannot be larger than 2^k.

*x1.15. For any partial function $g : \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$, consider the recursive equation

$$p(y, \vec{x}) = (\text{if } (g(y, \vec{x}) = 0) \text{ then } y \text{ else } p(y+1, \vec{x})). \tag{1-59}$$

(1) Prove that the least solution of (1-59) is the partial function

$$\begin{aligned} f(y, \vec{x}) &= (\mu t \geq y)[g(t, \vec{x}) = 0] \\ &= \text{the least } t \geq y \text{ such that } [g(t, \vec{x}) = 0 \\ &\qquad \& \ (\forall s)[y \leq s < t \Longrightarrow g(s, \vec{x})\downarrow \neq 0]. \end{aligned} \tag{1-60}$$

(2) Show that for different choices of g, (1-59) may have one or infinitely many solutions.

(3) Show that there is no choice of g for which (1-59) has finitely many but more than one solutions.

x1.16. Consider the recursive equation

$$p(x, y) = \text{if } (x = 0) \text{ then } 1 \text{ else } p(Pd(x), p(x, y)) \tag{1-61}$$

on $\mathbb{N}$

(1) Solve (1-61)—i.e., prove that there is a least partial function which satisfies it and give an explicit description of this partial function.

(2) Give an explicit description of the unique total function which satisfies (1-61).

*x1.17. Prove that (1-61) is satisfied by uncountably many distinct partial functions.

x1.18. Suppose $\mathcal{F} \subseteq \mathcal{P}(\Omega)$ is a family of sets such that $\emptyset \in \mathcal{F}$ and $\mathcal{F}$ is closed under arbitrary unions, i.e.,

$$\mathcal{X} \subseteq \mathcal{F} \Longrightarrow \bigcup \mathcal{X} \in \mathcal{F}.$$

Let $\Phi : \mathcal{F} \to \mathcal{F}$ be a monotone operator. Prove that the set

$$J = J_\Phi = \bigcup \{X \in \mathcal{F} \mid X \subseteq \Phi(X)\}$$

is the largest fixed point of Φ, i.e., $\Phi(J) = J$, and for every $X \in \mathcal{F}$, if $\Phi(X) = X$, then $X \subseteq J$.

In particular, every monotone operator $\Phi : \mathcal{P}(\Omega) \to \mathcal{P}(\Omega)$ has a largest fixed point.

x1.19. Fix a closed set C of real numbers, define $\Phi : \mathcal{P}(C) \to \mathcal{P}(C)$ by

$$\Phi(X) = \{x \in X \mid x \text{ is a limit point of } X\},$$

and let J_C be the largest fixed point of Φ. Prove that J_C is the *kernel* of C, i.e., the unique perfect subset F of C such that $C \setminus F$ is countable.

x1.20. Suppose $\leq$ is a wellordering of a set A. A subset $I \subseteq A$ is an *initial segment* (of A) if

$$[x \in I \ \& \ y < x] \Longrightarrow y \in I,$$

and for each $x \in A$, $I_x = \{y \in A \mid y < x\}$ is the initial segment "with top" x. It is easy to check that every initial segment is either A or I_x for some (unique) x.

Suppose $F : (A \rightharpoonup B) \to (A \rightharpoonup B)$ is an operation defined on all the partial functions on A to B, let

$$\mathcal{F} = \{p : A \rightharpoonup B \mid \text{Domain}(p) \text{ is an initial segment and for all } x \in \text{Domain}(p), p(x) = F(p \restriction I_x)\},$$

and define $\Phi : \mathcal{F} \to \mathcal{F}$ by

$$\Phi(p) = \begin{cases} p, & \text{if Domain}(p) = A, \\ p \cup \{(x, F(p))\}, & \text{if Domain}(p) = I_x. \end{cases}$$

Prove that (as a subset of $\mathcal{P}(A \times B)$) $\mathcal{F}$ is closed under arbitrary unions, $\Phi : \mathcal{F} \to \mathcal{F}$, Φ is monotone, and the largest fixed point f of Φ guaranteed by Problem x1.18 is the unique $f : A \to B$ such that

$$f(x) = F(f \restriction I_x) \qquad (x \in A).$$

Infer that f is also the least fixed point of Φ.

Note. This is the basic theorem of set theory which justifies definition by recursion on a wellordering, and the point of the problem is to show that it is a special case of the existence of largest fixed points of monotone operators on classes of sets. (In Chapter 4 we will derive a much simpler reduction of transfinite recursion to least-fixed-point recursion.)

x1.21. For each $G \subseteq \mathcal{P}(\Omega)$, define a monotone operator

$$\Phi_G : \mathcal{P}(\mathcal{P}(\Omega)) \to \mathcal{P}(\mathcal{P}(\Omega))$$

such that $B(G) = \mathbf{fix}(\Phi_G)$ is the *σ-field* generated by G, i.e., the least collection of subsets of Ω which contains all the sets in G and is closed under complementation, countable unions and countable intersections. (If G is the set of all *open* subsets of some topological space Ω, then $B(G)$ is the set of all *Borel* subsets of Ω.)

x1.22. For each string A of symbols from the alphabet of propositional formulas defined in 1.13, let

$$\begin{aligned} \mathrm{left}(A) &= \text{the number of left parentheses ``(" occurring in } A, \\ \mathrm{right}(A) &= \text{the number of right parentheses ``)" occurring in } A, \end{aligned}$$

and show that:

(a) If A is a formula, then $\mathrm{left}(A) = \mathrm{right}(A)$, and if X is an initial segment of some formula A, then $\mathrm{left}(X) \geq \mathrm{right}(X)$. (The parentheses *balance*.)

(b) No proper initial segment of a formula is a formula.

(c) **The Parsing Lemma**. For each propositional formula A, *exactly one* of the following cases applies:

1. $A \equiv \mathsf{v}_i$, with a uniquely determined v_i.
2. $A \equiv \neg(A_1)$, with a uniquely determined formula A_1.
3. $A \equiv (A_1 \,\&\, A_2)$, with uniquely determined formulas A_1 and A_2.
4. $A \equiv (A_1 \vee A_2)$, with uniquely determined formulas A_1 and A_2.
5. $A \equiv (A_1 \to A_2)$, with uniquely determined formulas A_1 and A_2.

HINT: Use the fixed-point characterization (1-28) of the set $\mathrm{Prop} = \mathbf{fix}(\pi)$ of formulas of the propositional calculus, and induction on the length of formulas.

x1.23. The terms and formulas of predicate logic (with identity) relative to a fixed **vocabulary**

$$\tau = (C, F, R)$$

are defined by the following two recursions:

$$\begin{aligned} A &:\equiv \mathsf{v} \mid \mathsf{c} \mid \mathsf{f}(A_1, \ldots, A_{n_\mathsf{f}}) \\ \phi &:\equiv A_1 = A_2 \mid \mathsf{R}(A_1, \ldots, A_{n_R}) \mid \neg(\phi_1) \mid (\phi_1 \,\&\, \phi_2) \mid (\exists \mathsf{v})\phi_1 \mid (\forall \mathsf{v})\phi_1 \end{aligned}$$

Here v is from a specified (infinite) set of variables; c is from the specified set C of *constant symbols* given by τ; f is from the specified set F of *function symbols* with associated arity n_f, as given by τ; R is from the specified set R of relation symbols determined by τ, each with a specified integer arity n_R; A, A_i stand for terms; and ϕ, ϕ_i stand for formulas.

(a) As in 1.13 and x1.22 for the propositional calculus, make this precise and formulate and prove a Parsing Lemma for the terms and the formulas.

(b) Show that the Parsing Lemma does not hold if we omit the parentheses in the conjunction clause $(\phi_1 \ \& \ \phi_2)$.

(c) Show that the Parsing Lemma holds if we omit the parentheses in the negation clause $\neg\phi_1$ and also allow n-ary conjunctions, for any n, i.e., the formation clause $(A_1 \ \& \ \dots \ \& \ A_n)$.

x1.24. Solve the following recursive equation on the set Streams(A) from an arbitrary set A:

$$f(u) = \text{if } (|u| \leq 2) \text{ then } u \text{ else } f(\text{tail}(u))\,;(\text{head}(u), \mathbf{t}).$$

(The problem asks for an explicit description of the least solution of this equation, with a proof that it is the least solution.)

1.31. **Definition.** A partial function $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ is *L_1-computable* if there is a program P such that for every n-tuple of integers $\vec{x} = (x_1, \dots, x_n)$,

(1) $f(\vec{x})\!\downarrow \iff \text{den(P)}((\vec{x}, \mathbf{t}))\!\downarrow$.

(2) $f(\vec{x})\!\downarrow \Longrightarrow \text{den(P)}(((\vec{x}, \mathbf{t})) = (\mathit{print}(f(\vec{x})), \mathbf{t})$.

In the problems which follow we outline a proof that the L_1-computable functions on the integers are exactly the Turing computable partial functions, using the classical identification of Turing computability with μ-recursion. The problems are simple (and somewhat boring) if you know a lot about programming and recursive functions; they can be made more challenging if you strive to give very simple proofs, using as much as possible the recursive characterization of denotations.

x1.25. Prove that the class of L_1-computable partial functions includes the successor $S(x) = x + 1$, every constant $C_q(\vec{x}) = q$, and every projection

$$P_i^n(\vec{x}) = x_i \qquad (1 \leq i \leq n).$$

x1.26. Prove that the class of L_1-computable partial functions is closed under definition by substitution,

$$f(\vec{x}) = g(h_1(\vec{x}), \dots, h_k(\vec{x})).$$

x1.27. Prove that the class of L_1-computable partial functions is closed under definition by primitive recursion,

$$\begin{aligned} f(0, \vec{x}) &= g(\vec{x}) \\ f(y+1, \vec{x}) &= h(f(y, \vec{x}), y, \vec{x}). \end{aligned}$$

x1.28. Prove that the class of L_1-computable partial functions is closed under minimalization,

$$\begin{aligned} f(\vec{x}) &= (\mu t)[g(t, \vec{x}) = 0] \\ &= (\mu t)[(\forall s < t)(\exists w)[g(t, \vec{x}) = w + 1] \ \& \ g(t, \vec{x}) = 0]. \end{aligned}$$

Infer that every Turing computable (μ-recursive) partial function is L_1-computable.

x1.29. Prove that every L_1-computable partial function on the integers is Turing computable.

*x1.30. Show that equation (1-46) has $2^{\aleph_0}$ solutions in the set of non-empty sets of infinite, binary sequences.

1.32. It is sometimes convenient to imbed Streams(A) into the set of infinite sequences $(\mathbb{N} \to A \cup \{\mathbf{t}, \infty\})$ where (just like $\mathbf{t}$), ∞ is some object not in A and not the same as $\mathbf{t}$. The embedding is the identity on all infinite streams, and (quite trivially), for the finite ones,

$$\pi((u_0, \ldots, u_{n-1})) = (u_0, \ldots, u_{n-1}, \infty, \infty, \cdots),$$

i.e., we simply extend them by adding an infinite sequence of ∞'s at the end. Now the image $\pi[\text{Streams}(A)]$ under this embedding is a *closed subset* of the *metric space* $(\mathbb{N} \to A \cup \{\mathbf{t}, \infty\})$, by the following definitions.

x1.31. (1) Show that for any set $B \neq \emptyset$, the function

$$d(\alpha, \beta) = \begin{cases} 0, & \text{if } \alpha = \beta, \\ \dfrac{1}{1 + \text{ the least } n \text{ such that } \alpha(n) \neq \beta(n)}, & \text{otherwise} \end{cases}$$

is a *metric* on the set of infinite sequences $(\mathbb{N} \to B)$, i.e.,

$$d(\alpha, \beta) = d(\beta, \alpha); \quad d(\alpha, \beta) = 0 \iff \alpha = \beta; \quad d(\alpha, \gamma) \leq d(\alpha, \beta) + d(\beta, \gamma).$$

(2) Show that for every non-empty B, the metric space $(\mathbb{N} \to B)$ is *complete*, i.e., every sequence $\{\alpha_n\}_n$ which has the Cauchy property converges.

x1.32. For the metric space $(\mathbb{N} \to B)$ with non-empty B:
(1) Show that a set $F \subseteq (\mathbb{N} \to B)$ is closed, if and only if there is a *tree* $T \subseteq B^*$ such that

$$F = [T] = \{\alpha \mid (\forall n)[\alpha \restriction n \in T]\}.$$

(A tree is a set of finite sequences closed under initial segments, and $[T]$ is called the *body* of the tree T.)
(2) Show that a set $F \subseteq (\mathbb{N} \to B)$ is *compact* if and only if $F = [T]$ for a *finitely splitting* tree, i.e., a tree T such that for each $u \in B^*$, the set $u * \{t\} \cap T$ of one-point extensions of u in T is finite.

*x1.33. Show that for each program P of L_2 and each state s, the set $\pi[\text{den}(\mathtt{P}(s)]$ is a compact subset of $(\mathbb{N} \to A \cup \{\mathbf{t}, \infty\})$, where π is the imbedding of Streams(A) defined in 1.32. HINT: Show that the set of computations comp(P)(s) is compact (in this sense), and appeal to the topological fact that continuous images of compact sets are compact.

CHAPTER 2

FIST-ORDER RECURSION AND COMPUTATION

Consider a first order structure

(2-1) $$\mathbf{M} = (M, c_1, \ldots, c_N, R_1, \ldots, R_L, f_1, \ldots, f_L)$$

as we study these in logic, with **universe** some non-empty set M and **given** distinguished members of M and relations and functions on M. A function $f : M^n \to M$ is **explicit** if it is defined by a term of the associated language of first-order logic; in this chapter we will introduce the **recursive functions** of $\mathbf{M}$, which are (roughly) the "canonical solutions" of systems of recursive term equations. This approach to recursion on first-order structures generalizes the work of McCarthy [1963], who established a very elegant and mathematically simple characterization of the usual (Turing)-computable functions on the natural numbers.

2A. Recursion in partial algebras

To apply model-theoretic methods to recursion theory, we need to make two adjustments, and the first of these is to allow partial functions and relations of all arities among the givens of a structure. In fact, it is convenient to adopt a somewhat eccentric notion of "n-ary partial function on M" which combines both of these, as follows.

2A.1. **Definition.** For each set M, we fix three, distinct objects $\mathtt{tt}$ (**truth**), $\mathtt{ff}$ (**falsity**) and $\perp$ (**bottom**, **divergence**) not in M, we set

$$M_{\mathbb{B}} = M \cup \{\mathtt{tt}, \mathtt{ff}\},$$

and we define the **partial function space** $(M^n \rightharpoonup M_{\mathbb{B}})$ by the following recursion on n:

$$(M^0 \rightharpoonup M_{\mathbb{B}}) = M_{\mathbb{B}} \cup \{\perp\} = M \cup \{\mathtt{tt}, \mathtt{ff}, \perp\},$$
$$(M^{n+1} \rightharpoonup M) = (M \to (M^n \rightharpoonup M_{\mathbb{B}}))$$

We refer to the objects in $(M^n \rightharpoonup M_{\mathbb{B}})$ as "n-ary partial functions on M" (even when $n = 0$, in which case p is **nullary**), and we write interchangeably

$$p \in (M^n \rightharpoonup M_{\mathbb{B}}) \iff p : M^n \rightharpoonup M_{\mathbb{B}},$$
$$p(x, y) = p(x)(y) \quad (p : M^2 \rightharpoonup M_{\mathbb{B}}),$$

and similarly for partial functions of arity greater than 2. We compose these partial function "strictly", treating values in $\{\mathrm{tt}, \mathrm{ff}\}$ as it they were $\perp$, e.g.,

$$f(g(x)) = w \iff (\exists u \in M)[g(x) = u \;\&\; f(u) = w] \quad (x \in M, w \in M_{\mathbb{B}}).$$

Thus $(M^n \rightharpoonup M_{\mathbb{B}})$ includes the usual partial functions $p : M^n \to M \cup \{\perp\}$ and **partial relations** $p : M^n \to \{\mathrm{tt}, \mathrm{ff}, \perp\}$ of arity n, but also some weird (but harmless) partial functions which take some values in M and some in $\{\mathrm{tt}, \mathrm{ff}\}$.[5]

2A.2. **Definition.** A **partial algebra** is any structure

$$\mathbf{M} = (M, f_1, \dots, f_L), \tag{2-2}$$

where M is a set, and for $i = 1, \dots, L$,

$$f_i : M^{n_i} \rightharpoonup M_{\mathbb{B}}$$

is a partial function of arity n_i. The **characteristic** (or **signature**) of $\mathbf{M}$ is the sequence of numbers

$$\chi(\mathbf{M}) = (n_1, \dots, n_L) \tag{2-3}$$

which codes the arities of the givens of $\mathbf{M}$.

A partial algebra is **total** if the givens $f_1, \dots, f_L$ are total functions.

Examples of partial algebras include the standard structure of (Peano) arithmetic

$$\mathbf{N} = (\mathbb{N}, 0, 1, +, \cdot, =); \tag{2-4}$$

the simpler, **basic structure of arithmetic**

$$\mathbf{N}_0 = (\mathbb{N}, 0, S, Pd, =_0) \tag{2-5}$$

with $S(x) = x + 1$, $Pd(x) = \text{if } (x = 0) \text{ then } 0 \text{ else } (x - 1)$, and

$$=_0 (x) = \text{if } (x = 0) \text{ then tt else ff}$$

is the characteristic function of equality with 0; and the algebras

$$\mathbf{Z} = (\mathbb{Z}, 0, 1, +, -, \cdot, =), \qquad \mathbf{Q} = (\mathbb{Q}, 0, 1, +, -, \cdot, \div, =)$$

where $\mathbb{Z} = \{0, 1, -1, 2, -2, \dots\}$ is the set of positive and negative "whole" numbers, $\mathbb{Q}$ is the set of fractions, the operation $a \div b$ of division in $\mathbf{Q}$ is

[5] Gödel used the term "notion" to refer indefinitely to a function or a relation, and we can think of these objects as still more general, indeterminate mixtures of the two.

set $= 0$ when $b = 0$ and $=$ is represented by its characteristic function $\chi_=$ per (1-19). Another example is the structure

$$\mathbf{R} = (\mathbb{R}, 0, 1, +, \cdot, \leq)$$

of the real numbers viewed as an ordered field, where the ordering relation is again represented by its characteristic (total) function $\chi_\leq$. Similarly, **graphs** and **groups** are (total) algebras

$$\mathbf{H} = (H, \rightarrow), \quad \mathbf{G} = (G, e, \cdot, ^{-1}, =) \tag{2-6}$$

where $x \rightarrow y$ is the edge relation on H (i.e., its characteristic function), and for groups we take as givens the unit, multiplication, inversion and the equality relation.

2A.3. **Expansions**. If $g : M^m \rightharpoonup M_{\mathbb{B}}$, then the partial algebra

$$(\mathbf{M}, g) = (M, f_1, \ldots, f_L, g)$$

is the **expansion** of $\mathbf{M}$ by g, and it has characteristic

$$\chi(\mathbf{M}, g) = \tau * (m) = (n_1, \ldots, n_L, m).$$

We do not, in general, assume that the characteristic function $\chi_=$ of the identity relation on M is among the givens, and so we often need to expand a structure to insure that it is in:

$$(\mathbf{M}, =) = (M, f_1, \ldots, f_L, \chi_=).$$

We will also consider expansions $(\mathbf{M}, g_1, \ldots, g_k)$ of $\mathbf{M}$ by any number of partial functions on the same universe.

2A.4. A **many-sorted partial algebra** is a structure

$$\mathbf{M} = (M_1, \ldots, M_n, f_1, \ldots, f_L), \tag{2-7}$$

where $M_1, \ldots, M_n$ are pairwise disjoint sets and each f_i is a partial function with input set some finite product of the universes and output set some M_j. An obvious example is a vector space

$$\mathbf{V} = (K, V, 0_K, 1_K, +_K, -_K, \cdot_K, \div_K, 0_V, +_V, -_V, \cdot_{K,V})$$

where K is the scalar field, the subscripted symbols denote the usual operations on K and V, and $\cdot_{K,V} : K \times V \rightarrow V$ is the multiplication of vectors by scalars. We will represent each such $\mathbf{M}$ as a single-universe partial algebra

$$\mathbf{M}' = (M_1 \cup \cdots \cup M_n, f'_1, \ldots, f'_L, \chi_1, \ldots, \chi_n), \tag{2-8}$$

whose universe is the union of the given n parts, each f'_i is defined on M' by setting its value equal to $\perp$ when one of its arguments is not in the proper part, and each χ_i is the characteristic function of M_i as a subset of M'. An important example of this form is **second order arithmetic**

$$\mathbf{N}^2 = (\mathbb{N}, \mathcal{N}, 0, 1, +, \cdot, \mathrm{ap}), \tag{2-9}$$

where $\mathcal{N} = (\mathbb{N} \to \mathbb{N})$ is **Baire space**, the set of all unary functions on $\mathbb{N}$, and $\mathrm{ap} : \mathcal{N} \times \mathbb{N} \to \mathbb{N}$ is **function application**,

$$\mathrm{ap}(\alpha, x) = \alpha(x).$$

2A.5. **The formal language $\mathsf{R}(\tau)$: syntax**. A **vocabulary** is any sequence of distinct symbols

$$\tau = (\mathsf{f}_1, \ldots, \mathsf{f}_L) \qquad (\mathrm{arity}(\mathsf{f}_i) = n_i)$$

with attached arities which determine its *characteristic*,

$$\chi(\tau) = (n_1, \ldots, n_L),$$

and for each vocabulary we specify a formal language $\mathsf{R}(\tau)$ with the following symbols:

individual variables:	$\mathsf{v}_0, \mathsf{v}_1, \ldots,$
individual constants:	$\mathbb{t}, \mathrm{ff}$
function constants:	$\mathsf{f}_1, \ldots, \mathsf{f}_L \quad (\mathrm{arity}(\mathsf{f}_i) = n_i)$
symbols for branching:	if then else
punctuation symbols:	, ( )
equality:	$=$

The second deviation from model theory that we need is to include **conditional definitions** in the terms of R.

The set $T(\tau)$ of **explicit terms** of $\mathsf{R}(\tau)$ is the smallest set of *words* (finite sequences of symbols) with the following properties:

(T1) The individual constants $\mathbb{t}, \mathrm{ff}$ and all individual variables v_i are terms.
(T2) If $A_1, \ldots, A_{n_i}$ are terms, then so is the word $\mathsf{f}_i(A_1, \ldots, A_{n_i})$.
(T3) If A_1, A_2, A_3 are terms, then so is (if A_1 then A_2 else A_3).

In the summary way of describing such recursive definitions of syntactic objects,

$$A :\equiv \mathbb{t} \mid \mathrm{ff} \mid \mathsf{v}_i \mid \mathsf{f}_i(A_1, \ldots, A_{n_i}) \mid (\text{if } A_1 \text{ then } A_2 \text{ else } A_3) \tag{2-10}$$

A term is **closed** if no individual or function variable occurs in it.

Each term is uniquely in one of the forms of (2-10), and then with uniquely determined **subterms** which have smaller length; this justifies proofs by **structural induction** for properties of terms and definition by **structural recursion** of operations on terms.

As usual, we will rarely write out "grammatically correct" terms. In practice we will use the familiar mathematical symbols instead of their formal versions, e.g., $x, y, \ldots$ rather than $\mathsf{v}_1, \mathsf{v}_2, \ldots$ for individual variables, $f, g, +, \cdot, \ldots$ instead of $\mathsf{f}_1, \mathsf{f}_2, \ldots$ for function constants, etc. We will also

skip parentheses or replace them with braces and blank space when this helps readability, and write (for example)

$$(x+y)\cdot z \text{ instead of } \cdot(+(\mathsf{v}_1,\mathsf{v}_2),\mathsf{v}_3)$$

2A.6. **The formal language $\mathsf{R}(\tau)$: denotational semantics**. A **τ-algebra** is a partial algebra $\mathbf{M}$ as in (2-2) with $\chi(\mathbf{M})=\chi(\tau)$, so that the terms of $\mathsf{R}(\tau)$ can be naturally interpreted in $\mathbf{M}$.

A **valuation**[6] in a τ-algebra $\mathbf{M}$ is any function

$$\pi : \{\mathsf{v}_0,\mathsf{v}_1,\dots\}\to M$$

which assigns a member of $M\cup\{\mathtt{tt},\mathtt{ff}\}$ to each individual variable v_i. The **value** (or **denotation**)

$$\mathrm{den}(A,\pi)=\mathrm{den}(\mathbf{M},A,\pi)$$

of a term A of $\mathsf{R}(\tau)$ in $\mathbf{M}$ is determined by the following recursion on terms:

$$\mathrm{den}(\mathtt{tt},\pi)=\mathtt{tt};\quad \mathrm{den}(\mathtt{ff},\pi)=\mathtt{ff};\quad \mathrm{den}(\mathsf{v}_i,\pi)=\pi(\mathsf{v}_i)$$
$$\mathrm{den}(\mathsf{f}_i(A_1,\dots,A_{n_i}),\pi)=f_i(\mathrm{den}(A_1,\pi),\dots,\mathrm{den}(A_{n_i},\pi))$$

$\mathrm{den}(\text{if } A_1 \text{ then } A_2 \text{ else } A_3,\pi)$

$$=\begin{cases}\mathrm{den}(A_2,\pi), & \text{if } \mathrm{den}(A_1,\pi)=\mathtt{tt},\\ \mathrm{den}(A_3,\pi), & \text{if } \mathrm{den}(A_1,\pi)\downarrow \ \&\neq \mathtt{tt},\\ \bot, & \text{otherwise, i.e., if } \mathrm{den}(A_1,\pi)\uparrow.\end{cases}$$

It is clear that $\mathrm{den}(A,\pi)$ need not always converge, as we have allowed partial functions among the givens of $\mathbf{M}$. We write

$$\mathbf{M},\pi\models A=B \iff \mathrm{den}(\mathbf{M},A,\pi)=\mathrm{den}(\mathbf{M},B,\pi),$$
$$\mathbf{M}\models A=B \iff \text{for all valuations } \pi,\ \mathbf{M},\pi\models A=B.$$

If $\mathbf{M}\models A=B$, we say that the term equation $A=B$ is **valid** in $\mathbf{M}$.

2A.7. **Term replacement**. For each term A, each individual variable x and each term B, we put

$$A\{\mathsf{x}:\equiv B\}\equiv \text{the result of replacing } \mathsf{x} \text{ by } B \text{ in } A,$$

and, more generally, for $\vec{\mathsf{x}}=\mathsf{x}_1,\dots,\mathsf{x}_m$, $\vec{B}=B_1,\dots,B_m$,

$$A\{\vec{\mathsf{x}}:\equiv\vec{B}\}\equiv A\{\mathsf{x}_1:\equiv B_1\}\cdots\{\mathsf{x}_m:\equiv B_m\}. \tag{2-11}$$

[6] These are called *assignments* in logic, but we will call them valuations here to avoid confusion with the assignment commands of programming languages. **Updates** of valuations are defined in the obvious way: if $t\in M$, then

$$\pi\{x:=t\}(\mathsf{v}_i)=\begin{cases}t, & \text{if } \mathsf{v}_i\equiv x,\\ \pi(\mathsf{v}_i), & \text{otherwise.}\end{cases}$$

For example,

$$\mathsf{f}_1(\mathsf{v}_5,\mathsf{v}_1)\{\mathsf{v}_1 :\equiv B\} \equiv \mathsf{f}_1(\mathsf{v}_5,B), \quad \mathsf{f}_2(\mathsf{v}_5,\mathsf{v}_1)\{\mathsf{v}_1 :\equiv B, \mathsf{v}_5 :\equiv C\} \equiv \mathsf{f}_2(C,B)$$

2A.8. **Lemma.** *For each partial algebra* $\mathbf{M}$*, all terms* A, B*, each variable* x*, and all valuations in* M*:*

if $\mathrm{den}(B,\pi) = w \in M$*, then* $\mathrm{den}(A\{\mathsf{x} :\equiv B\},\pi) = \mathrm{den}(A,\pi\{\mathsf{x} := w\})$;

if $\mathrm{den}(B,\pi) = \mathtt{tt}$*, then* $\mathrm{den}(A\{\mathsf{x} :\equiv B\},\pi) = \mathrm{den}(A\{\mathsf{x} \equiv \mathtt{tt}\},\pi)$*; and if* $\mathrm{den}(B,\pi) = \mathtt{ff}$*, then* $\mathrm{den}(A\{\mathsf{x} :\equiv B\},\pi) = \mathrm{den}(A\{\mathsf{x} \equiv \mathtt{ff}\},\pi)$.

It follows that for any terms A, B, C*,*

$$\text{(2-12)} \quad \text{if } \mathbf{M} \models A = C \text{ and } \mathrm{den}(B,\pi)\downarrow, \quad \text{then } \mathrm{den}(A\{\mathsf{x} :\equiv B\},\pi) = \mathrm{den}(C\{\mathsf{x} :\equiv B\},\pi).$$

PROOF. The first claim is easy, by induction on the term A, Problem x2A.1, and the second follows from it: for example, if $\mathrm{den}(B,\pi) = w \in M$, then

$$\begin{aligned}\mathrm{den}(A\{\mathsf{x} :\equiv B\},\pi) &= \mathrm{den}(A,\pi\{\mathsf{x} := w\}) \\ &= \mathrm{den}(C,\pi\{\mathsf{x} := w\}) = \mathrm{den}(C\{\mathsf{x} :\equiv B\},\pi).\end{aligned}$$ ⊣

The hypothesis $\mathrm{den}(B,\pi)\downarrow$ is necessary for the second claim, which may fail if $\mathrm{den}(B,\pi)\uparrow$, Problem x2A.2. The most substantial applications of this Lemma are when B is a closed, convergent term, which then determines a specific member of M, independent of any valuation.

2A.9. **The programming language** $\mathsf{R}(\tau)$**; terms**. To use $\mathsf{R}(\tau)$ as a programming language, first we enrich it with (partial) **function variables**

$$\zeta_0^n, \zeta_1^n, \ldots \quad (n = 0, 1, \ldots, \qquad \mathrm{arity}(\zeta_i^n) = n),$$

infinitely many for each arity $n \geq 0$. From the point of view of syntax, function variables are treated exactly like the function constants $\mathsf{f}_1, \ldots, \mathsf{f}_L$: the terms are now defined by the recursion

$$\text{(2-13)} \quad A :\equiv \mathtt{tt} \mid \mathtt{ff} \mid \mathsf{v}_i \mid \mathsf{f}_i(A_1, \ldots, A_{n_i}) \mid \zeta_i^n(A_1, \ldots, A_n) \mid (\text{if } A_1 \text{ then } A_2 \text{ else } A_3)$$

and they have all the usual properties—unique readability, etc. If all the function variables that occur in a term A are in the list $\zeta_1, \ldots, \zeta_n$, then A is a term in the expanded language

$$\mathsf{R}(\tau, \zeta_1, \ldots, \zeta_n) = \mathsf{R}(\mathsf{f}_1, \ldots, \mathsf{f}_L, \zeta_1, \ldots, \zeta_n),$$

and it is naturally interpreted in expansions

$$(\mathbf{M}, p_1, \ldots, p_n) = (M, f_1, \ldots, f_L, p_1, \ldots, p_n)$$

of τ-algebras $\mathbf{M}$, by considering each ζ_i as a function constant denoting the partial function p_i.

2A.10. **Recursive equations and programs**. As a programming language, $\mathsf{R}(\tau)$ has two more categories of syntactic objects other than terms: recursive equations and programs.

A **recursive equation** of $\mathsf{R}(\tau)$ is a term equation of the form

$$\zeta(\mathsf{x}_1, \dots, \mathsf{x}_n) = A, \tag{e}$$

where ζ is a function variable; $\mathsf{x}_1, \dots, \mathsf{x}_n$ are distinct individual variables; and A is a term in which the only individual variables which (may) occur are in the list $\mathsf{x}_1, \dots, \mathsf{x}_n$. The equation is **explicit** if no function variables occur in A. If $\mathbf{M}$ is a τ-algebra, then we also refer to an $\mathsf{R}(\tau)$-equation as a recursive equation of $\mathbf{M}$.

For example (and with simplified notation)

$$\zeta(x) = \text{if } R(x) \text{ then ff else tt}$$

is an explicit equation in every vocabulary with the relation symbol R;

$$\zeta(x) = \text{if } (x = 0) \text{ then } 0 \text{ else } S(\zeta(Pd(x)))$$

is a recursive, not explicit equation of $\mathbf{N}_0$ (with $(x = 0)$ viewed as an abbreviation of $=_0 (x)$); and

$$\zeta(x) = S(y)$$

is not a recursive equation because y occurs on the right but not on the left.

Finally a **recursive program** of $\mathsf{R}(\tau)$ (or of any τ-algebra $\mathbf{M}$) is any system of recursive equations

$$(E) \qquad \begin{array}{lrcl} (e_0) & \zeta_0(\vec{x}_0) & = & E_0 \\ & & \vdots & \\ (e_k) & \zeta_k(\vec{x}_k) & = & E_k \end{array}$$

where the function variables $\zeta_0, \dots, \zeta_k$ are distinct and they are the only function variables which occur in the terms $E_0, \dots, E_k$. The equations of E are called (mutual, recursive) **definitions** of the function variables $\zeta_0, \dots, \zeta_k$, and ζ_0 is the **principal function variable** of E.

2A.11. **Computational semantics of $\mathsf{R}(\tau)$ (1)**. Fix again a τ-algebra $\mathbf{M}$. We will show how to assign to each function variable ζ_i with $\text{arity}(\zeta_i) = k_i$ of a program E as above a partial function

$$\overline{\zeta}_i : M^{k_i} \rightharpoonup M_{\mathbb{B}} \quad (\text{arity}(\zeta_i) = k_i),$$

so that the partial functions $\overline{\zeta}_1, \dots, \overline{\zeta}_k$ **satisfy** E, i.e.,

$$(\mathbf{M}, \overline{\zeta}_0, \dots, \overline{\zeta}_k) \models \zeta_i(\vec{x}_i) = E_i \quad (i = 0, \dots, k, \vec{x}_i \in M^{k_i}).$$

Before we give the precise definition of this correspondence

$$(\mathbf{M}, E) \mapsto (\overline{\zeta}_0, \dots, \overline{\zeta}_k),$$

we consider a few examples.

The definition

$$(E_1) \qquad \zeta(x,y) = S(y)$$

is (by itself) a program of $\mathbf{N}_0$, which defines (explicitly) the binary function variable ζ. The semantics should give us

$$\overline{\zeta}(x,y) = \mathrm{den}(\mathbf{N}_0, S(y)) = y+1.$$

The equation

$$(E_2) \qquad \zeta(y,\vec{x}) = \text{if } (g(y,\vec{x}) = 0) \text{ then } y \text{ else } \zeta(S(y),x)$$

is a program of $(\mathbf{N}_0, g)$ for any $g : \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$, and there are examples of g for which (E_2) may have many solutions, but for each g, there is a least solution of (E_2) by Problem *x1.14, namely

$$\overline{\zeta}(y,\vec{x}) = (\mu i \geq y)[g(i,\vec{x}) = 0].$$

This should be the solution $\overline{\zeta}$ which the computational semantics of $\mathsf{R}(\tau, g)$ will assign to the function variable ζ in $(\mathbf{N}_0, g)$.

Finally, consider the following trivial program, with the single equation

$$(E_3) \qquad \zeta(x) = \zeta(x),$$

on any partial algebra $\mathbf{M}$. Equation (E_3) is satisfied by all partial functions; the computational semantics will yield

$$\overline{\zeta}(x) = \emptyset(x) = \bot,$$

i.e., again the least solution of (E_3), as in the previous example (E_2).

The basic idea of the computational semantics of $\mathsf{R}(\tau)$ is to associate with each τ-algebra $\mathbf{M}$, each program E, and each function variable ζ_i of E a **machine**

$$\mathcal{M} = \mathcal{M}(\mathbf{M}, E, \zeta_i)$$

which **computes** a partial function

$$\overline{\zeta}_i : M^{k_i} \rightharpoonup M_{\mathbb{B}};$$

we will then show that the partial functions

$$(\overline{\zeta}_0, \ldots, \overline{\zeta}_k)$$

satisfy the equations of E in $\mathbf{M}$—and, indeed, they are the least solutions of E in $\mathbf{M}$.

2A.12. **Definition.** A **transition system** is any triple

$$\mathcal{T} = (S, \rightarrow, T),$$

where:

(a) S is a non-empty set, the *states* of $\mathcal{T}$.

(b) $\to$ is a binary *transition relation* on S.
(c) $T \subseteq S$ is the set of *terminal states*, and it is such that

$$s \in T \Longrightarrow (\forall s')[s \not\to s']. \tag{2-14}$$

A state which satisfies (2-14) but is not terminal is **stuck** (or **inactive**), and a state which is neither terminal nor stuck is **active**: thus s is active if there is at least one s' such that $s \to s'$.

A system $\mathcal{T}$ is **deterministic** if each state has at most one *next state*, i.e.,

$$[s \to s' \ \& \ s \to s''] \Longrightarrow s' = s''. \tag{2-15}$$

For example, let

$$m \to_1 n \iff m > n, \qquad m \to_2 n \iff m = n+1; \tag{2-16}$$

the system $(\mathbb{N}, \to_1, \{0\})$ is non-deterministic, while $(\mathbb{N}, \to_2, \{0\})$ is deterministic.

2A.13. **Computations**. A **partial computation** of a transition system $\mathcal{T}$ is any finite *path*

$$Y = (s_0 \to s_1 \to \cdots \to s_n) \tag{2-17}$$

in the *graph* $(S, \to)$; Y is **convergent** if the last state s_n is terminal and **stuck** if s_n is stuck. An **infinite divergent computation** is any infinite path

$$Y = (s_0 \to s_1 \to \cdots)$$

of the graph $(S, \to)$. The *length* $|Y|$ of a finite, partial computation as in (2-17) is $n+1$.

A transition system $\mathcal{T}$ *computes* a partial function $\pi : S \rightharpoonup T$ if for all $s \in S$ and $t \in T$,

$$\pi(s) = t \iff (\exists (s_0, \dots, s_n) \in C(\mathcal{T}))[s_0 = s \ \& \ s_n = t], \tag{2-18}$$

where

$$C(\mathcal{T}) = \text{the set of all convergent coputations of } \mathcal{T}. \tag{2-19}$$

It follows that

$$\pi(s)\downarrow \iff (\exists (s_0, \dots, s_n) \in C(\mathcal{T}))[s_0 = s].$$

Every deterministic transition system computes exactly one partial function $\pi : S \rightharpoonup T$ which is defined by (2-18). In addition, if $\mathcal{T}$ is deterministic, then for each state s there is exactly one convergent, stuck or infinite *complete* computation

$$\mathrm{comp}(s) = \mathrm{comp}_{\mathcal{T}}(s) = (s \to s_1 \to s_2, \dots) \tag{2-20}$$

which cannot be extended, and which is defined by the recursion

$$s_0 = s,$$
$$s_{n+1} = \begin{cases} \text{the unique } s' \text{ such that } s_n \to s', & \text{if one exists,} \\ \bot, & \text{otherwise.} \end{cases}$$

To compute a partial function $f : A \rightharpoonup B$ with some transition system $\mathcal{T}$, we must enrich $\mathcal{T}$ with some method of introducing arguments from A and extracting values in B.

Next we want to associate a transition system with each program E of $\mathsf{R}(\tau)$ and each τ-algebra $\mathbf{M}$, and to do this we will need to enrich $\mathsf{R}(\tau)$ with names for the members of M.

2A.14. **M-terms**. For each τ-algebra $\mathbf{M}$, the M-terms are defined by the recursion

$$A :\equiv \mathtt{tt} \mid \mathtt{ff} \mid x \mid \mathsf{v}_i \mid \mathsf{f}_i(A_1, \ldots, A_{n_i}) \mid \zeta_i^n(A_1, \ldots, A_n) \mid (\text{if } A_1 \text{ then } A_2 \text{ else } A_3) \tag{2-21}$$

where $x \in M$; thus we use the members of M as names of themselves, and we will also refer to them as **individual constants**.[7]

An M-term is **closed** if no individual variable occurs in it.

2A.15. **Recursive machines**. For each τ-algebra $\mathbf{M}$ and each program E of $\mathsf{R}(\tau)$, we define a transition system

$$\mathcal{T}(E) = \mathcal{T}(E, \mathbf{M})$$

as follows.

(a) The states of $\mathcal{T}(E)$ are all words s of the form

$$\alpha_0 \ \ldots \ \alpha_{m-1} : \beta_0 \ \ldots \ \beta_{n-1}$$

where the "symbols" $\alpha_0, \ldots, \alpha_{m_1}, \beta_0, \ldots, \beta_{n-1}$ of s satisfy the following conditions:

- Each α_i is a (constant or variable) function symbol, or a closed M-term, or the special symbol ?, and
- each β_j is an individual constant, i.e., a member of $M_{\mathbb{B}}$.

The states of $\mathcal{T}(E, \mathbf{M})$ are the same for all programs of $\mathbf{M}$, and so we can call them the **M-states**.

For example, the word

$$\zeta_1^2 \ 3 \ \mathsf{S}(3) \ 1 \ \text{if} \ \ ? : 3 \ 0 \ 1$$

is a state of $\mathbf{N}_0$, as are the words

$$\mathsf{Pd} \ 1 \ 3 \ \zeta_1^2(\mathsf{S}(2)) : \qquad : 0 \ 23$$

[7] Assuming the obvious—that no symbol of $\mathsf{R}(\tau)$ is a member of M.

or the simplest

$$:$$

(b) The terminal states of $\mathcal{T}(E)$ are all states of the form

$$: w$$

i.e., those with no symbols to the left of : and just one $w \in M_{\mathbb{B}}$ on the The recursive machines which will be associated with programs of $\mathbf{M}$ will all have the same output function,

$$\text{output}(: w) = w.$$

(c) The transition function of $\mathcal{T}(E)$ is defined by seven cases in the Transition Table 2: i.e., $s \to s'$ if it is a special case of one of the lines in the Table. Notice that the external calls (e-call) are the only transitions which depend on $\mathbf{M}$ (they "call" the givens), while the internal calls (i-call) are the only transitions which depend on the program E.

It is clear that $\mathcal{T}(E)$ is a deterministic transition system.

For each n-ary function variable ζ_i of E, the **recursive machine** $\mathcal{T}(E, \zeta_i)$ is derived from $\mathcal{T}(E)$ by adding the input function

$$\text{input}(\vec{x}) \equiv \zeta_i : \vec{x}$$

and it computes the partial function $\overline{\zeta}_i : M^n \rightharpoonup M_{\mathbb{B}}$, where

$$\overline{\zeta}_i(\vec{x}) = w \iff \zeta_i : \vec{x} \to s_1 \to \cdots \to : w. \tag{2-22}$$

Another useful notation is

$$\mathbf{M}, E \vdash \zeta_i(\vec{x}) = w \iff \overline{\zeta}_i(\vec{x}) = w \quad (\vec{x} \in M^n, w \in M_{\mathbb{B}}), \tag{2-23}$$

as it reveals the dependence of $\overline{\zeta}$ on the τ-algebra $\mathbf{M}$ and the program E. In practice, we will simply refer to the partial function $\overline{\zeta}_i$, when the specific partial algebra $\mathbf{M}$ and program E are clear from the context.

The **main symbol** of a program E is the function variable ζ_0 in its first equation, and E **computes in** $\mathbf{M}$ the partial function $\overline{\zeta}_0$.

For example, if one of the equations of E in $\mathbf{N}_0$ is

$$\zeta(x) = S(S(x)),$$

then the computation

$$\zeta : x \ \to \ S(S(x)) : \to \ S \ S(x) : \to \ S \ S \ x : \\ \to S \ S \ : \ x \to S \ : \ x+1 \to : \ x+2$$

verifies that for every x, $\overline{\zeta}(x) = x + 2$.

For a more interesting example, we notice that addition is defined recursively in $\mathbf{N}_0$ by the single equation

$$\zeta(i, x) = \text{if } (i = 0) \text{ then } x \text{ else } S(\zeta(Pd(i), x)).$$

(pass)	$\alpha\ \underline{x :}\ \beta$	$\to$	$\alpha\ \underline{: x}\ \beta \quad (x \in M)$
(e-call)	$\alpha\ \underline{f_i : \vec{x}}\ \beta$	$\to$	$\alpha\ \underline{: f_i(\vec{x})}\ \beta$
(i-call)	$\alpha\ \underline{\zeta_i : \vec{x}}\ \beta$	$\to$	$\alpha\ \underline{E_i\{\vec{x}_i :\equiv \vec{x}\} :}\ \beta$
(comp)	$\alpha\ \underline{h(A_1, \dots, A_n) :}\ \beta$	$\to$	$\alpha\ \underline{h\ A_1\ \cdots\ A_n :}\ \beta$
(br)	$\alpha\ \underline{(\text{if } A \text{ then } B \text{ else } C) :}\ \beta$	$\to$	$\alpha\ \underline{B\ C\ ?\ A :}\ \beta$
(br0)	$\alpha\ \underline{B\ C\ ? : \Bbbk}\ \beta$	$\to$	$\alpha\ \underline{B :}\ \beta$
(br1)	$\alpha\ \underline{B\ C\ ? : y \neq \Bbbk}\ \beta$	$\to$	$\alpha\ \underline{C :}\ \beta$

- The underlined words are those which change in each transition.
- $\vec{x} = x_1, \dots, x_n$ is an n-tuple of individual constants.
- In the *external call* (e-call), f_i is one of the given partial functions of $\mathbf{M}$, with $\text{arity}(f_i) = n_i = n$.
- In the *internal call* (i-call), ζ_i is an n-ary function variable of the program E which is defined by the equation $\zeta_i(\vec{x}) = E_i$, and $\vec{x} \in M^{k_i}$.
- In the *composition transition* (comp), h is a (constant or variable) function symbol with $\text{arity}(h) = n$.

TABLE 2. Transitions of the system $\mathcal{T}(E, \mathbf{M})$.

In Figure 1 we show the computation of $\overline{\zeta}(2, 3) = 5$ by this program.

2A.16. **Definition (M-recursion).** A partial function $f : M^n \rightharpoonup M$ is **recursive** in the partial algebra $\mathbf{M}$ (or **M-recursive**), if $f = \overline{\zeta}$ for some program E of $\mathbf{M}$ and some function variable ζ of E, i.e., if

$$f(\vec{x}) = w \iff \mathbf{M}, E \vdash \zeta(\vec{x}) = w$$

with the notation in (2-23). Since the order in which the equations of a program are listed does not affect the definition of the partial functions $\overline{\zeta}$, *a partial function $f : M^n \rightharpoonup M$ is* $\mathbf{M}$*-recursive if and only if it is computed by some program of* $\mathbf{M}$. We set

(2-24) $$\mathbf{rec}_0(\mathbf{M}) = \{f : M^n \rightharpoonup M \mid f \text{ is } \mathbf{M}\text{-recursive}\}.$$

$$
\begin{array}{rll}
\zeta : 2\ 3 & & \text{(i-call)} \\
\text{if } (2 = 0) \text{ then } 3 \text{ else } S(\zeta(Pd(2), 3)) : & & \text{(br)} \\
3\ S(\zeta(Pd(2), 3))\ ?\ =_0 (2) : & & \text{(comp)} \\
3\ S(\zeta(Pd(2), 3))\ ?\ =_0 2 : & & \text{(pass)} \\
3\ S(\zeta(Pd(2), 3))\ ?\ =_0 : 2 & & \text{(e-call)} \\
3\ S(\zeta(Pd(2), 3))\ ? : \mathrm{ff} & & \text{(br1)} \\
S(\zeta(Pd(2), 3)) : & & \text{(comp)} \\
S\ \zeta(Pd(2), 3) : & & \text{(comp)} \\
S\ \zeta\ Pd(2)\ 3 : & & \text{(pass)} \\
S\ \zeta\ Pd(2) : 3 & & \text{(comp)} \\
S\ \zeta\ Pd\ 2\ : 3 & & \text{(pass)} \\
S\ \zeta\ Pd : 2\ 3 & & \text{(e-call)} \\
S\ \zeta : 1\ 3 & & \text{(i-call)} \\
S \text{ if } (1 = 0) \text{ then } 3 \text{ else } S(\zeta(Pd(1), 3)) : & & \text{(br), (comp), (pass), (e-call)} \\
S\ 3\ S(\zeta(Pd(1), 3))\ ? : \mathrm{ff} & & \text{(br1)} \\
S\ S(\zeta(Pd(1), 3)) : & & \text{(comp)} \\
S\ S\ \zeta(Pd(1), 3) : & & \text{(comp)} \\
S\ S\ \zeta\ Pd(1)\ 3 : & & \text{(pass)} \\
S\ S\ \zeta\ Pd(1) : 3 & & \text{(comp)} \\
S\ S\ \zeta\ Pd\ 1 : 3 & & \text{(pass)} \\
S\ S\ \zeta\ Pd : 1\ 3 & & \text{(e-call)} \\
S\ S\ \zeta : 0\ 3 & & \text{(i-call)} \\
S\ S \text{ if } (0 = 0) \text{ then } 3 \text{ else } S(\zeta(Pd(0), 3)) : & & \text{(br), (comp), (pass), (e-call)} \\
S\ S\ 3\ S(\zeta(Pd(0), 3))\ ? : \mathrm{tt} & & \text{(br0)} \\
S\ S\ 3 : & & \text{(pass)} \\
S\ S : 3 & & \text{(e-call)} \\
S : 4 & & \text{(e-call)} \\
: 5 & &
\end{array}
$$

FIGURE 1. The computation of $2 + 3$ by the program $\zeta(i, x) = \text{if } (i = 0) \text{ then } x \text{ else } S(\zeta(Pd(i), x))$.

A relation $P(\vec{x})$ on M is **M**-recursive if its characteristic function (1-19) is **M**-recursive, and a set $A \subseteq M$ is **M**-recursive if the associated membership

relation

$$R_A(x) \iff x \in A$$

is **M**-recursive. For obscure, historical reasons, the class of **M**-recursive relations is called the (recursive) **section** of **M**,

$$\mathbf{sec}(\mathbf{M}) = \{R \subseteq M^n \mid R \text{ is } \mathbf{M}\text{-recursive}\}. \tag{2-25}$$

Finally, a relation $P(\vec{x})$ on M is **M-semirecursive** if it is the domain of convergence of some **M**-recursive partial function,

$$P(\vec{x}) \iff f(\vec{x})\downarrow \quad (f \in \mathbf{rec}_0(\mathbf{M}),$$

and a set $A \subseteq M$ is **M-semirecursive** if

$$x \in A \iff f(x)\downarrow$$

for some **M**-recursive partial function. The class of these relations is the **envelope** of **M**,

$$\mathbf{env}(\mathbf{M}) = \{P \subseteq M^n \mid P \text{ is } \mathbf{M}\text{-semirecursive}\}. \tag{2-26}$$

Many of the simple properties of **M**-recursion can be established easily by using the following, natural extensions of the familiar notions of model theory to partial structures.

2A.17. **Homomorphisms and imbeddings**. Suppose **M** and $\mathbf{M}'$ are partial algebras of the same characteristic. A **homomorphism**

$$\rho : \mathbf{M} \to \mathbf{M}'$$

is any (total) function $\rho : M_{\mathbb{B}} \to M'_{\mathbb{B}}$ such that $\rho(\mathrm{tt}) = \mathrm{tt}, \rho(\mathrm{ff}) = \mathrm{ff}$, and for $i = 1, \ldots, K$,

$$f_i(x_1, \ldots, x_{n_i}) = w \Longrightarrow f'_i(\rho(x_1), \ldots, \rho(x_{n_i})) = \rho(w); \tag{2-27}$$

we call ρ an **imbedding** if it is an injection (one-to-one), and an **isomorphism** if it is a bijection and the inverse $\rho^{-1} : M' \to M$ is also an imbedding.

Notice that we do not demand of homomorphisms and imbeddings the converse of (2-27), which is equivalent to the more usual identity

$$\rho(f_i(x_1, \ldots, x_{n_i})) = f'_i(\rho(x_1), \ldots, \rho(x_{n_i}));$$

when that holds, we call ρ a **strong homomorphism** or **strong imbedding**.

If $M \subseteq M'$ and the identity $x \mapsto x$ is an imbedding of **M** into $\mathbf{M}'$, we say that **M** is a (partial) **subalgebra** of $\mathbf{M}'$, in symbols,

$$\mathbf{M} \subseteq_p \mathbf{M}' \iff \mathbf{M} \text{ is a partial subalgebra of } \mathbf{M}'.$$

This clearly holds if for $i = 1, \ldots, K$, $f_i \sqsubseteq f'_i$.

2A.18. **Lemma.** (1) *If $\rho : \mathbf{M}_1 \to \mathbf{M}_2$ is a homomorphism from one τ-algebra into another and E is a program of $\mathsf{R}(\tau)$, then for every function variable ζ of E and all $\vec{x} \in M_1, w \in M_1 \cup \{\mathrm{tt}, \mathrm{ff}\}$,*

$$\text{if } \mathbf{M}_1, E \vdash \zeta_i(\vec{x}) = w, \text{ then } \mathbf{M}_2, E \vdash \zeta_i(\rho(\vec{x})) = \rho w,$$

with $\rho(x_1, \dots, x_n) = (\rho(x_1), \dots, \rho(x_n))$. In particular, if $\mathbf{M} \subseteq_p \mathbf{M}'$, then

$$\text{if } \mathbf{M}, E \vdash \zeta_i(\vec{x}) = w, \text{ then } \mathbf{M}', E \vdash \zeta_i(\vec{x}) = w.$$

(2) *If $\rho : \mathbf{M} \to \mathbf{M}$ is a homomorphism of a partial algebra into itself and $f : M^n \rightharpoonup M_{\mathbb{B}}$ is $\mathbf{M}$-recursive, then for all $\vec{x} \in M^n, w \in M_{\mathbb{B}}$,*

$$\text{if } f(\vec{x}) = w, \text{ then } f(\rho(\vec{x})) = \rho(w).$$

PROOF. (1) We extend ρ to all M_1-terms by the obvious recursion on their definition,

$$\begin{aligned}
\rho \mathsf{v}_i &\equiv \mathsf{v}_i \\
\rho x &\equiv \rho(x) \quad (x \in M_1), \\
\rho \mathsf{f}_i(A_1, \dots, A_{n_i}) &\equiv \mathsf{f}_i(\rho A_1, \dots, \rho A_{n_i}) \\
\rho \zeta_i^n(A_1, \dots, A_n) &\equiv \zeta_i(\rho A_1, \dots, \rho A_n) \\
\rho(\text{if } A \text{ then } B \text{ else } C) &\equiv (\text{if } \rho A \text{ then } \rho B \text{ else } \rho C)
\end{aligned}$$

and then to the states of $\mathbf{M}_1$ by setting

$$\begin{gathered}
\rho ? \equiv ? \quad \rho \mathsf{f}_i \equiv \mathsf{f}_i, \rho \zeta_i^n \equiv \zeta_i^n \\
\rho(\alpha_0 \cdots \alpha_{n-1} : \beta_0 \cdots \beta_{m-1}) = \rho\alpha_0 \cdots \rho\alpha_{n-1} : \rho\beta_0 \cdots \rho\beta_{m-1}
\end{gathered}$$

Now ρs is a state of $\mathbf{M}_2$ for each state s of $\mathbf{M}_2$, and for the relevant transition systems, by inspection,

$$\text{if } s \to s' \text{ in } \mathbf{M}_1, \text{ then } \rho s \to \rho s' \text{ in } \mathbf{M}_2,$$

which gives the required result.

(2) follows immediately. ⊣

Several of the problems that follow (including x2A.8) can be solved easily using this simple Lemma.

Problems for Section 2A

x2A.1. Prove the first claim of Lemma 2A.8.

x2A.2. Prove that the second claim of Lemma 2A.8 is not true if we omit the hypothesis $\mathrm{den}(B, \pi)\downarrow$.

x2A.3. What are the partial computations of the transition systems (2-16), and which partial functions they compute?

x2A.4. For the following three recursive programs in $\mathbf{N}_0$:

(E_1) $$\zeta(x) = S(\zeta(x))$$

(E_2) $$\zeta(x) = \zeta(\eta(x))$$
$$\eta(x) = x,$$

(E_3) $$\zeta(x,y) = \zeta_1(\zeta(x,y),y).$$
$$\zeta_1(x,y) = x,$$

(1) Which partial functions satisfy them?

(2) Which partial functions do they compute, and in what way do their computations differ?

*x2A.5. (1) Show that for every program E in a *total* algebra $\mathbf{M}$, and every n-ary function variable ζ which is defined in E, there is no stuck computation of the form

(*) $$\zeta : x_1, \dots, x_n \to s_1 \to \cdots \to s_m.$$

(Stuck computations are defined in 2A.13.)

(2) Show that if $\mathbf{M}$ is a partial algebra, ζ is an n-ary function variable of a program E, and the finite computation (*) is stuck, then its last state is of the form

$$\alpha\ f_i : y_1, \dots, y_{n_i}\ \beta$$

where f_i is one of the given partial functions of $\mathbf{M}$ and $f_i(y_1, \dots, y_{n_i})\!\uparrow$.

x2A.6. Prove or give a counterexample: or each partial algebra $\mathbf{M}$ and each $x_0 \in M$, the constant, unary function $f(x) = x_0$ is $\mathbf{M}$-recursive.

2A.19. **Definition.** (a) A set $X \subseteq M$ is *closed for the givens* of a partial algebra $\mathbf{M}$, or **M-closed**, if

$$[x_1, \dots, x_{n_i} \in X \text{ and } f_i(x_1, \dots, x_{n_i}) = w] \Longrightarrow w \in X, \qquad (i = 1, \dots, L).$$

(b) For each $A \subseteq M$, define recursively

$$G_0(A) = A,$$
$$G_{m+1}(A) = G_m(A) \cup \{f_i(x_1, \dots, x_{n_i}) \in M \mid x_1, \dots, x_{n_i} \in G_m(A), i = 1, \dots, L\}.$$

The set $G_m(A)$ is the subset of M generated in m steps by A in $\mathbf{M}$, and the union

$$\overline{A}_{\mathbf{M}} = \bigcup_{m=0}^{\infty} G_m(A)$$

is the subset of M which is **generated** by A in $\mathbf{M}$.

x2A.7. Prove that for each partial algebra $\mathbf{M}$ and each $A \subseteq M$, the set $\overline{A}$ is the least $\mathbf{M}$-closed subset of M which contains A.

x2A.8. Prove that if $A \subseteq M$ and the partial function $f : M^n \rightharpoonup M$ is $\mathbf{M}$-recursive, then the $\mathbf{M}$-closure $\overline{A}$ of A is closed for f, i.e.,

$$[x_1, \dots, x_n \in \overline{A}, f(x_1, \dots, x_n) = w] \Longrightarrow w \in \overline{A}.$$

x2A.9. Suppose $\mathbf{M} = (M, R_1, \dots, R_L)$ is a partial algebra whose givens are all (possibly partial) relations. Prove that the only recursive function $f : M \to M$ is the identity $\mathrm{id}(x) = x$.

x2A.10. Suppose $\mathbf{G} = (G, e, \cdot, ^{-1}, =)$ is a group and $f : G \to G$ is a recursive, unary function; what are the possible values of $f(x)$?

x2A.11. Suppose $\mathbf{G} = (G, e, \cdot, ^{-1}, =)$ is a group and for each $x \in G$, let $[x]$ be the subgroup generated by x. Prove that the relation

$$R(t, x) \iff t \in [x]$$

is $\mathbf{G}$-semirecursive.

x2A.12. Suppose $\mathbf{G}$ is a group as in (2-6), and let

$$f(x, y) = x^n, \text{ where } n \text{ is the least } m \in \mathbb{N} \text{ such that } y^m = e.$$

Prove that the (partial) function $f(x, y)$ is $\mathbf{G}$-recursive.

2B. Computational soundness and least solutions

In this section, we will establish two simple but fundamental results, which yield a "structural" (computation-independent) characterization of the tuple of partial functions $(\overline{\zeta}_0, \dots, \overline{\zeta}_k)$ computed by a program E in a partial algebra $\mathbf{M}$ and imply easily the basic properties of $\mathbf{M}$-recursion. The keys to these theorems are the following two, basically trivial observations.

2B.1. **Lemma (Function variable relabelling).** *Suppose $\zeta_0, \dots, \zeta_k$ are the function variables defined in a program E of $\mathsf{R}(\tau)$ and $\zeta'_0, \dots, \zeta'_k$ are fresh, distinct function variables such that* $\mathrm{arity}(\zeta_i) = \mathrm{arity}(\zeta'_i)$, *and let E' be the system of equations obtained by replacing each ζ_i by ζ'_i in the equations of E. It follows that E' is also a program, and that for any τ-algebra $\mathbf{M}$ and all $\vec{x} \in M, w \in M_{\mathbb{B}}$,*

$$\mathbf{M}, E \vdash \zeta_i(\vec{x}) = w \iff \mathbf{M}, E' \vdash \zeta'_i(\vec{x}) = w.$$

Proof is immediate, by verifying that for each computation

$$\alpha_0 : \beta_0 \to \cdots \to \alpha_n : \beta_n$$

of $\mathcal{T}(E, \mathbf{M})$, the sequence

$$\alpha'_0 : \beta'_0 \to \cdots \to \alpha'_n : \beta'_n$$

produced by replacing each ζ_i by ζ'_i is a computation of $\mathcal{T}(E, \mathbf{M})$. ⊣

This lemma allows "alphabetic changes" in the function variables, and makes it possible to assume (in effect) that any two programs E_1 and E_2 with which we need to deal have no common function variables.

2B.2. **Lemma (Transition locality).** *For every partial computation*

$$\alpha_0 : \beta_0 \to \alpha_1 : \beta_1 \to \cdots \to \alpha_m : \beta_m$$

of the transition system $\mathcal{T}(E, \mathbf{M})$ and any words α^*, β^* such that the sequence

$$\alpha^* \ \alpha_0 : \beta_0 \ \beta^*$$

is a state, the sequence of states

$$\alpha^* \ \alpha_0 : \beta_0 \ \beta^* \to \alpha^* \ \alpha_1 : \beta_1 \ \beta^*, \to \cdots \to \alpha^* \ \alpha_m : \beta_m \ \beta^*$$

is also a partial computation of $\mathcal{T}(E, \mathbf{M})$.

It follows that if

$$\alpha_0 : \beta_0 \to \alpha_1 : \beta_1 \to \cdots$$

is a divergent computation and $\alpha^* \ \alpha_0 : \beta_0 \ \beta^*$ is a state, then the computation

$$\alpha^* \ \alpha_0 : \ \beta_0 \ \beta^* \to \alpha^* \ \alpha_1 : \beta_1 \ \beta^* \to \cdots$$

is also divergent.

PROOF. By induction on $m \geq 0$, with the basis given by the hypothesis. In the induction step, we assume that the sequence

$$\alpha^* \ \alpha_0 : \beta_0 \ \beta^* \ \to \alpha^* \ \alpha_1 : \beta_1 \ \beta^* \ \to \cdots \to \alpha^* \ \alpha_m : \beta_m \ \beta^*$$

is a partial computation, we consider separately the seven cases which justify the transition

$$\alpha_m : \beta_m \to \alpha_{m+1} : \beta_{m+1}$$

and it is obvious in each of them that the same line of Table 2 justifies also the transition

$$\alpha^* \ \alpha_m : \beta_m \ \beta^* \to \alpha^* \ \alpha_{m+1} : \beta_{m+1} \ \beta^*$$

The second claim follows by applying the first to the (initial) partial computations

$$\alpha_0 : \beta_0 \to \alpha_1 : \beta_1 \to \cdots \to \alpha_m : \beta_m \quad (m \in \mathbb{N})$$ ⊣

To formulate the next theorem simply, we extend to all closed M-terms the "logical" notation for computations of 2-23: for each τ-algebra $\mathbf{M}$, each program E, each closed M-term A of $\mathsf{R}(\tau, \zeta_0, \dots, \zeta_k)$, and each $w \in M_{\mathbb{B}}$, set

$$\mathbf{M}, E \vdash A = w \iff A \to s_1 \to \cdots \to : w, \tag{2-28}$$

$$\mathbf{M}, E \vdash A\uparrow \iff \mathrm{comp}_{\mathcal{T}}(A :) \text{ is divergent.} \tag{2-29}$$

(Recall that a computation diverges if it is infinite or stuck.)

2B.3. **Theorem (Computational soundness of $\mathsf{R}(\tau)$).** *Fix a program E in $\mathsf{R}(\tau)$ with function variables $\zeta_0, \dots, \zeta_k$ and a τ-algebra $\mathbf{M}$, and let*

$$\overline{\mathbf{M}} = (\mathbf{M}, \overline{\zeta}_0, \dots, \overline{\zeta}_k) = (M, f_1, \dots, f_L, \overline{\zeta}_0, \dots, \overline{\zeta}_k).$$

Suppose A is any closed M-term of $\mathsf{R}(\tau, \zeta_0, \dots, \zeta_k)$.

(a) *If $\mathrm{den}(\overline{\mathbf{M}}, A)\uparrow$, then the computation $\mathrm{comp}_{\mathcal{T}}(A :)$ of $\mathcal{T}(E, \mathbf{M})$ with initial state $A :$ is infinite or stuck* (and hence divergent), *and*

(b) *if $\mathrm{den}(\overline{\mathbf{M}}, A) = w$, then the computation $\mathrm{comp}_{\mathcal{T}}(A :)$ of $\mathcal{T}(E, \mathbf{M})$ with initial state $A :$ converges with terminal state $: w$.*

Hence for every closed M-term A and $w \in BM$,

$$(\mathbf{M}, \overline{\zeta}_0, \dots, \overline{\zeta}_k) \models A\uparrow \iff \mathbf{M}, E \vdash A\uparrow, \tag{2-30}$$

$$(\mathbf{M}, \overline{\zeta}_0, \dots, \overline{\zeta}_k) \models A = w \iff \mathbf{M}, E \vdash A = w, \tag{2-31}$$

and the partial functions $\overline{\zeta}_0, \dots, \overline{\zeta}_k$ satisfy the program E in $\mathbf{M}$, i.e.,

$$(\mathbf{M}, \overline{\zeta}_0, \dots, \overline{\zeta}_k) \models \zeta_i(\vec{\mathsf{x}}) = E_i, \qquad (i = 1, \dots, K). \tag{2-32}$$

PROOF. We prove (a) and (b) together by induction on the given, closed M-term A, and we consider cases.

(1) If $A \equiv x \in M_{\mathbb{B}}$, then $\mathrm{den}(\mathbf{M}, A) = x$, and the computation

$$\begin{aligned} x :& \qquad \text{(pass)} \\ &: x \end{aligned}$$

computes the correct value.

(2) If $A \equiv f_i(A_1, \dots, A_{n_i})$ for a given partial function f_i of $\mathbf{M}$, then the computation $\mathrm{comp}(A :)$ starts with the transition

$$\begin{aligned} f_i(A_1, \dots, A_{n_i}) :& \quad \text{(comp)} \\ f_i\ A_1\ \dots\ A_{n_i} &: \end{aligned}$$

We consider three cases:

(2a) For some j, $\mathrm{den}(\overline{\mathbf{M}}, A_j)\uparrow$, so that $\mathrm{den}(\overline{\mathbf{M}}, A)\uparrow$. If j is *largest* ($\leq n_i$) with this property, then (by the induction hypothesis) the computation

comp(A :) starts with the steps

$$\begin{array}{rll} f_i(A_1,\dots,A_{n_i}): & & \text{(comp)} \\ f_i\ A_1\ \dots\ A_{n_i}: & & \text{(hyp.)} \\ \vdots & & \\ f_i\ A_1\ \dots\ A_{n_i-1}: & w_{n_i} & \text{(ind. hyp.)} \\ \vdots & & \\ f_i\ A_1\ \dots\ A_j: & w_{j+1}\ \cdots\ w_{n_i} & \end{array}$$

By the induction hypothesis again, the computation

$$\text{comp}(A_j :) = A_j :\!\rightarrow \alpha_1 : \beta_1 \rightarrow \cdots$$

diverges, since $\text{den}(\overline{\mathbf{M}}, A_j)\uparrow$, and by Lemma 2B.2, the computation

$$f_i\ A_1\ \dots\ A_{j-1}\ A_j :\!\rightarrow f_i\ A_1\ \dots\ A_{j-1}\ \alpha_1 : \beta_1 \rightarrow \cdots$$

must also diverge—so that comp(A :) diverges.

(2b) There are points $w_1,\dots,w_{n_i}$ in $M_{\mathbb{B}}$ such that $\text{den}(\overline{\mathbf{M}}, A_j) = w_j$ for $j = 1,\dots,n_i$, but $f_i(w_1,\dots,w_{n_i})\uparrow$. In this case, by the induction hypothesis and another appeal of Lemma 2B.2, the computation comp(A :) starts with the steps

$$\begin{array}{rll} f_i(A_1,\dots,A_{n_i}): & & \text{(comp)} \\ f_i\ A_1\ \dots\ A_{n_i}: & & \text{(ind. hyp.)} \\ \vdots & & \\ f_i\ A_1\ \dots\ A_{n_i-1}: & w_{n_i} & \text{(ind. hyp.)} \\ \vdots & & \\ f_i: & w_1\ w_2\ \cdots\ w_{n_i} & \end{array}$$

and it gets stuck at this point because $f_i(w_1,\dots,w_{n_i})\uparrow$.

(2c) $\text{den}(\overline{\mathbf{M}}, f_i(A_1,\dots,A_{n_i})) = w$, so that there are points $w_1,\dots,w_{n_i}$ in $M_{\mathbb{B}}$ with $\text{den}(\overline{\mathbf{M}}, A_j) = w_j$ for $j = 1,\dots,n_i$ and $f_i(w_1,\dots,w_{n_i}) = w$. By the induction hypothesis and Lemma 2B.2, the computation comp(A :)

now looks as follows:

$$\begin{array}{rll} f_i(A_1,\dots,A_{n_i}): & & \text{(comp)} \\ f_i\ A_1\ \dots\ A_{n_i}: & & \text{(ind. hyp.)} \\ \vdots & & \\ f_i\ A_1\ \dots\ A_{n_i-1}: & w_{n_i} & \text{(ind. hyp.)} \\ \vdots & & \\ f_i: & w_1\ w_2\ \cdots\ w_{n_i} & \\ : & f_i(w_1,\dots,w_{n_i}) & \end{array}$$

which is what we needed to prove.

(3) If $A \equiv \zeta_i(A_1,\dots,A_n)$ for some n-ary function variable ζ_i of E, then the computation comp$(A:)$ starts with the transition

$$\begin{array}{rl} \zeta_i(A_1,\dots,A_n): & \text{(comp)} \\ \zeta_i\ A_1\ \dots\ A_n: & \end{array}$$

We consider three cases, as in case (2):

(3a) For some j, $\mathrm{den}(\overline{\mathbf{M}}, A_j)\!\uparrow$, in which case $\mathrm{den}(\overline{\mathbf{M}}, A)\!\uparrow$.

(3b) There are $w_1,\dots,w_n$ in M such that $\mathrm{den}(\overline{\mathbf{M}}, A_j) = w_j$ for $j = 1,\dots,n$, but $\overline{\zeta}_i(w_1,\dots,w_n)\!\uparrow$.

(3c) $\mathrm{den}(\overline{\mathbf{M}}, \zeta_i(A_1,\dots,A_n)) = w$,which means that there are $w_1,\dots,w_n$ in M such that $\mathrm{den}(\overline{\mathbf{M}}, A_j) = w_j$ for $j = 1,\dots,n$ and $\overline{\zeta}_i(w_1,\dots,w_n) = w$.

For case (3a), the argument is exactly the same as in the corresponding case (2a), and for (3b) and (3c), the proofs are small variations of the arguments in (2b) and (2c) which use the definition of $\overline{\zeta}_i$. For (3c), for example, the induction hypothesis and Lemma 2B.2 guarantee the existence of $w_1,\dots,w_{n_i}$ in M such that the computation looks like this:

$$\begin{array}{rll} \zeta_i(A_1,\dots,A_n): & & \text{(comp)} \\ \zeta_i\ A_1\ \dots\ A_n: & & \text{(ind. hyp.)} \\ \vdots & & \\ \zeta_i\ A_1\ \dots\ A_{n-1}: & w_n & \text{(ind. hyp.)} \\ \vdots & & \\ \zeta_i: & w_1\ w_2\ \cdots\ w_n & \text{(def. of } \overline{\zeta}_i) \\ \vdots & & \\ : & \overline{\zeta}_i(w_1,\dots,w_n) & \end{array}$$

which is what we needed to prove.

To verify (2-30), we need to show the converse of (a), i.e., that

$$\text{if } \mathrm{comp}_{\mathcal{T}}(A:) \text{ is divergent, then } \overline{\mathbf{M}} \models A\uparrow;$$

this holds because if $\overline{\mathbf{M}} \models A = w$ for some $w \in M$, then $\mathbf{M}, E \vdash A = w$ by (b), which contradicts the hypothesis. Similarly, for the converse of (b) which is needed to complete the proof of (2-31): if $\mathbf{M}, E \vdash A = w$ and $\overline{\mathbf{M}} \not\models A = w$, then either $\overline{\mathbf{M}} \models A\uparrow$, in which case, by (a), $\mathbf{M}, E \vdash A\uparrow$, contradicting the hypothesis; or $\overline{\mathbf{M}} \models A = v$ for some $v \neq w$, in which case, by (b), $\mathbf{M}, E \vdash A = v$ which contradicts the fact that the transition system $\mathcal{T}(E, \mathbf{M})$ is deterministic.

Finally, for the last claim (2-32), we compute, for any $\vec{x}, w \in M_{\mathbb{B}}$:

$$\begin{aligned}(\mathbf{M}, \overline{\zeta}_0, \dots, \overline{\zeta}_k) \models \zeta_i(\vec{x}) = w &\iff \mathbf{M}, E \vdash \zeta_i(\vec{x}) = w \\ &\iff \mathbf{M}, E \vdash E_i\{\vec{\mathsf{x}} :\equiv \vec{x}\} = w \\ &\iff (\mathbf{M}, \overline{\zeta}_0, \dots, \overline{\zeta}_k) \models E_i\{\vec{\mathsf{x}} :\equiv \vec{x}\} = w,\end{aligned}$$

where the first and last of these equivalences follow from (2-31) and the middle one is immediate from the transition table of $\mathcal{T}(E, \mathbf{M})$. ⊣

2B.4. **Corollary (Closure properties of $\mathbf{rec}_0(\mathbf{M})$).** *The set of* $\mathbf{M}$*-recursive partial functions contains the givens* $f_1, \dots, f_L$ *of* $\mathbf{M}$*, the projections*

$$P_i^n(x_1, \dots, x_n) = x_i \quad (i = 1, \dots, n),$$

the constants $C_0(\vec{x}) = \mathrm{tt}$ *and* $C_1(\vec{x}) = \mathrm{ff}$*, and it is closed for composition and branching, i.e.,:*

(a) *If* $h, g_1, \dots, g_m$ *are* $\mathbf{M}$*-recursive and*

$$f(\vec{x}) = h(g_1(\vec{x}), \dots, g_m(\vec{x})), \tag{2-33}$$

then f *is* $\mathbf{M}$*-recursive.*

(b) *If* c, g *and* h *are* $\mathbf{M}$*-recursive and*

$$f(\vec{x}) = \text{if } c(\vec{x}) \text{ then } g(\vec{x}) \text{ else } h(\vec{x}), \tag{2-34}$$

then f *is* $\mathbf{M}$*-recursive.*

PROOF. Each given f_i is computed by the program

$$\zeta(x_1, \dots, x_{n_i}) = f_i(x_1, \dots, x_{n_i}) \tag{E_{f_i}}$$

for which, obviously, $\overline{\zeta}(\vec{x}_i) = f(\vec{x}_i)$, and the constants tt, ff and the projections are also computed by trivial programs.

For branching, the hypothesis gives us programs E_c, E_g, E_h and specific function variables c, g and h in these programs, and we must construct a new program E which defines some "fresh" function variable ζ so that

$$\overline{\zeta}_E(\vec{x}) = \text{if } \overline{c}_{E_c}(\vec{x}) \text{ then } \overline{g}_{E_g}(\vec{x}) \text{ else } \overline{h}_{E_h}(\vec{x}),$$

where the subscripts indicate the programs which compute $\overline{c}_{E_c}, \overline{g}_{E_g}$ and $\overline{h}_{E_h}$. We may assume by Lemma 2B.1 that there are no common function variables in the given programs E_c, E_g, E_h. Set

$$E = E_c + E_g + E_h + \{\zeta(\vec{x}) = \text{if } c(\vec{x}) \text{ then } g(\vec{x}) \text{ else } h(\vec{x})\},$$

where by "+" we simply mean the "union" of programs, conceived as sets of recursive definitions. Now E is a program, since each function variable in it is defined exactly once. Also

$$\overline{c}_E(\vec{x}) = \overline{c}_{E_c}(\vec{x}),$$

simply because each computation

$$\text{comp}_{E_c}(c : \vec{x}) = c : \vec{x} \to \alpha_1 : \beta_1 \to \cdots$$

of E_c is also a computation of E and hence the only computation of E which starts with $c : \vec{x}$ (because the recursive machines are deterministic), in other words

$$\text{comp}_{E_c}(c : \vec{x}) = \text{comp}_E(c : \vec{x});$$

thus $\overline{c}_E = \overline{c}_{E_c}$, and the same holds for the function variables g, h. Finally, Theorem 2B.3 implies that $\overline{\zeta}_E$ satisfies the equation which defines it, i.e.,

$$\begin{aligned}\overline{\zeta}(\vec{x}) &= \text{if } \overline{c}_E(\vec{x}) \text{ then } \overline{g}_E(\vec{x}) \text{ else } \overline{h}_E(\vec{x}) \\ &= \text{if } \overline{c}_{E_c}(\vec{x}) \text{ then } \overline{g}_{E_g}(\vec{x}) \text{ else } \overline{h}_{E_h}(\vec{x}).\end{aligned}$$

The proof for closure under composition is the same, Problem x2B.1. ⊣

2B.5. **Corollary.** (a) (**Closure properties of sec(M)**). *The set of* **M***-recursive relations is closed under negation* ($\neg$), *conjunction* (&), *disjunction* ($\vee$) *and total* **M***-recursive substitutions, i.e., if* $P(y_1, \ldots, y_m)$ *is* **M***-recursive,* $g_1(\vec{x}), \ldots, g_m(\vec{x})$ *are total,* **M***-recursive functions and*

$$R(\vec{x}) \iff P(g_1(\vec{x}), \ldots, g_m(\vec{x})), \tag{2-35}$$

then $R(\vec{x})$ *is* **M***-recursive.*

(b) (**Closure properties of env(M)**). *The class of* **M***-semirecursive relations is closed under conjunction* (&) *and* **M***-recursive partial substitutions, i.e., the scheme of definition* (2-35) *where* $g_1, \ldots, g_m$ *are* (not necessarily total) **M***-recursive partial functions.*

PROOF is easy, Problem x2B.2. ⊣

The envelope of **M** is not in general closed under disjunction by Problem *x2B.3, whose proof, however, is not entirely trivial, and which does not answer all that we would like to know about the question, see Problem *x2B.4. The closure of **env**(**M**) under disjunction when **M** is a total algebra is a basic result of the subject which we will postpone until Chapter 5, see Problem *x2B.5.

2B.6. **Corollary (Transitivity property).** *For each partial algebra* $\mathbf{M}$ *and any* $g : M^n \rightharpoonup M_{\mathbb{B}}$, $f : M^m \rightharpoonup M_{\mathbb{B}}$,

$$\text{if } g \in \mathbf{rec}_0(\mathbf{M}) \text{ and } f \in \mathbf{rec}_0(\mathbf{M}, g), \text{ then } f \in \mathbf{rec}_0(\mathbf{M}),$$

where $(\mathbf{M}, g) = (M, f_1, \dots, f_L, g)$ *is the expansion of* $\mathbf{M}$ *by* g.

This is easy to prove, just like Corollary 2B.4, and so we will leave it for Problem x2B.7, but it is an extremely useful result: it says, in effect, that in proofs of recursiveness we may assume that every recursive partial function is among the givens, and we will appeal to it constantly (and without explicit mention).

The next result characterizes the **canonical solutions** of a program E, those computed by the recursive machine.

2B.7. **Theorem (Least Fixed Points).** *For any program* E *of* $\mathsf{R}(\tau)$ *with function variables* $\zeta_0, \dots, \zeta_k$ *and any* τ*-algebra* $\mathbf{M}$, *the partial functions* $\overline{\zeta}_0, \dots, \overline{\zeta}_k$ *computed by* $\mathcal{T}(E, \mathbf{M})$ *are the* $\sqsubseteq$*-least partial functions which satisfy in* $\mathbf{M}$ *the equations of* E.

PROOF. The partial functions $\overline{\zeta}_0, \dots, \overline{\zeta}_k$ satisfy E by Theorem 2B.3, so it is enough to show that *if* $\zeta'_0, \dots, \zeta'_k$ *satisfy the equations of* E, *in* $\mathbf{M}$, *then*

$$\overline{\zeta}_i(\vec{x}) = w \Longrightarrow \zeta'_i(\vec{x}) = w \quad (i = 0, \dots, k).$$

Suppose then that $\zeta'_0, \dots, \zeta'_k$ satisfy E and consider the partial algebras

$$\overline{\mathbf{M}} = (\mathbf{M}, \overline{\zeta}_0, \dots, \overline{\zeta}_k), \qquad \mathbf{M}' = (\mathbf{M}, \zeta'_0, \dots, \zeta'_k).$$

By Theorem 2B.3, for every closed M-term A of $\mathsf{R}(\tau, \zeta_0, \dots, \zeta_k)$,

$$\text{if } \overline{\mathbf{M}} \models A = w, \text{then } \mathbf{M}, E \vdash A = w.$$

We will show by induction on m, that for each closed M-term A and every $w \in M_{\mathbb{B}}$,

$$\text{(2-36)} \quad \text{if } A :\rightarrow \alpha_1 : \beta_1 \rightarrow \cdots \alpha_{m-1} : \beta_{m-1} \rightarrow : w, \quad \text{then } \mathrm{den}(\mathbf{M}', A) = w.$$

In the special case $A \equiv \zeta_i(x_1, \dots, x_n)$, this gives

$$\overline{\zeta}_i(\vec{x}) = w \Longrightarrow \mathrm{den}(\mathbf{M}', \zeta_i(\vec{x})) = w \Longrightarrow \zeta'_i(\vec{x}) = w,$$

which is what we needed to show.

For the proof of (2-36) we consider the form of A, and the argument is trivial (as in the proof of 2B.3) in all cases except when

$$A \equiv \zeta_i(A_1, \dots, A_n),$$

for which the computation takes the form

$$\begin{aligned}
\zeta_i(A_1,\dots,A_n) &: \\
\zeta_i\ A_1\ \cdots\ A_n &: \\
&\vdots \\
\zeta_i\ A_1\ \cdots\ A_{n-1} &: w_n \\
&\vdots \\
\zeta_i &: w_1\ \cdots\ w_n \\
E_i\{\vec{\mathsf{x}} :\equiv \vec{w}\} &: \\
&\vdots \\
&: w
\end{aligned}$$

Now the induction hypothesis guarantees that

$$\mathrm{den}(\mathbf{M}', A_1) = w_1, \dots, \mathrm{den}(\mathbf{A}', A_n) = w_n, \mathrm{den}(\mathbf{M}', E_i\{\vec{\mathsf{x}} :\equiv \vec{w}\}) = w$$

because the computations which produce these values are shorter. Hence

$$\begin{aligned}
\mathrm{den}(\mathbf{M}', \zeta_i(A_1,\dots,A_n)) &= \zeta_i'(\mathrm{den}(\mathbf{M}', A_1),\dots,\mathrm{den}(\mathbf{M}', A_n)) \\
&= \zeta_i'(w_1,\dots,w_n) \\
&= \mathrm{den}(\mathbf{M}', E_i\{\mathsf{x}) :\vec{\equiv}\ \vec{w}\} = w,
\end{aligned}$$

and the last equation expresses exactly the required conclusion

$$\mathbf{M}' \models \zeta_i(\vec{\mathsf{x}}) = E_i. \qquad \dashv$$

This characterization makes it possible to prove that many partial functions are recursive, especially when the partial algebra $\mathbf{M}$ is rich, e.g., if it contains a copy of the natural numbers in the following sense:

2B.8. **Definition.** A **copy of $\mathbf{N}_0$** is any partial algebra

$$\mathbf{N}_0' = (\mathbb{N}', 0', S', Pd', =_0')$$

which is isomorphic with the basic structure of arithmetic $\mathbf{N}_0$. A partial algebra $\mathbf{M}$ **contains** $\mathbf{N}_0$ ($\mathbf{N}_0 \subseteq \mathbf{M}$) if there is a copy of $\mathbf{N}_0$ with $\mathbb{N}' \subseteq M$, and **imbeds** $\mathbf{N}_0$ ($\mathbf{N}_0 \hookrightarrow \mathbf{M}$) if, in addition, the following conditions hold:

(a) $\mathbb{N}'$ is an $\mathbf{M}$-semirecursive subset of M.

(b) The constant (nullary) function $0'$ and the unary partial functions $S', Pd', =_{0',\mathbf{M}}: M \rightharpoonup M$ are $\mathbf{M}$-recursive with (common) domain of convergence $\mathbb{N}'$.

Typically we will skip the primes in the notation, in effect "identifying" $\mathbf{N}_0$ with any copy of it contained or imbedded in a partial algebra $\mathbf{M}$.

In the simplest examples of this, $\mathbf{N}_0 \hookrightarrow \mathbf{N}_0$, but also $\mathbf{N}_0 \hookrightarrow \mathbf{N}^2$, Problem x2B.9, and in general, we can always add a copy of $\mathbf{N}_0$ to $\mathbf{M}$ in the extended (two-sorted) partial algebra

$$\mathbf{M} \uplus \mathbf{N}_0 = (M, \mathbb{N}, f_1, \ldots, f_L, 0_{\mathbf{N}_0}, \chi_{0_{\mathbf{N}_0}}, S, Pd, =_0). \tag{2-37}$$

Notice that in this structure, $\mathbb{N}$ is a recursive (not just semirecursive) set.

Problems for Section 2B

x2B.1. Prove that the composition (2-33) of $\mathbf{M}$-recursive partial functions is $\mathbf{M}$-recursive.

x2B.2. Prove the closure properties of $\mathbf{sec}(\mathbf{M})$ and $\mathbf{env}(\mathbf{M})$ listed in Corollary 2B.5.

*x2B.3. Give an example of an expansion $\mathbf{M} = (\mathbf{N}_0, f, g)$ of $\mathbf{N}_0$ in which some set $A \subseteq \mathbb{N}$ is $\mathbf{M}$-semirecursive and and has $\mathbf{M}$-semirecursive complement, but A is not $\mathbf{M}$-recursive.

*x2B.4. Give an example of a partial algebra $\mathbf{M}$ in which the disjuntion of two $\mathbf{M}$-semirecursive relations is not $\mathbf{M}$-semirecursive.

*x2B.5 (**Open**). Is there an expansion $(\mathbf{M} = \mathbf{N}_0, f_1, \ldots, f_m)$ of $\mathbf{N}_0$ in which the disjuntion of two $\mathbf{M}$-semirecursive relations is not always $\mathbf{M}$-semirecursive?

*x2B.6 (**Open**). Suppose $\mathbf{M}$ is a total algebra; prove that the disjunction of two $\mathbf{M}$-semirecursive relations is $\mathbf{M}$-semirecursive, and if both $R(\vec{x})$ and $\neg R(\vec{x})$ are both $\mathbf{M}$-semirecursive, then $R(\vec{x})$ is $\mathbf{M}$-recursive.

Remark. These claims are true, Corollary **??**, but the proof we will give for them is quite difficult and depends on notions that we have not yet introduced; so the challenge is to find a fairly simple, or at least elementary proof of these basic facts.

x2B.7. Prove Theorem 2B.6: for each partial algebra $\mathbf{M}$ and any partial function $g : M^n \rightharpoonup M$, $f : M^m \rightharpoonup M$,

$$[g \in \mathbf{rec}_0(\mathbf{M}) \;\&\; f \in \mathbf{rec}_0(\mathbf{M}, g)] \Longrightarrow f \in \mathbf{rec}_0(\mathbf{M}),$$

where $(\mathbf{M}, g) = (M, f_1, \ldots, f_L, g)$ is the expansion of $\mathbf{M}$ by g.

x2B.8. Show that if $g, h : M \rightharpoonup M_{\mathbb{B}}$ are $\mathbf{M}$-recursive partial functions, then so is the partial function

$$f(x) = g^m(x) \text{ where } m = (\mu n \geq 1)[h^n(x) = \mathtt{tt}].$$

(This is trivial if $\mathbf{N}_0 \hookrightarrow \mathbf{M}$, so the challenge is to prove it without this assumption.)

x2B.9. Prove that $\mathbf{N}_0 \hookrightarrow \mathbf{N}^2$.

x2B.10. Suppose $\mathbf{N}_0 \hookrightarrow \mathbf{M}$.

(a) Show that if $g(\vec{x})$ and $h(w, y, \vec{x})$ are $\mathbf{M}$-recursive, and $f(y, \vec{x})$ is defined from them by the **primitive recursion**

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

then f is $\mathbf{M}$-recursive.

(b) Show that if $g(t, \vec{x})$ is $\mathbf{M}$-recursive and

$$f(\vec{x}) = (\mu t)[g(t, \vec{x}) = 0]$$

is the minimalization of g defined in (1-60), then f is also $\mathbf{M}$-recursive.

(c) (McCarthy). Show that the $\mathbf{N}_0$-recursive partial functions on $\mathbb{N}$ are exactly the Turing-computable partial functions.

Note. For (c) you can use the classical identification of Turing computability with μ-recursiveness. For (a) and (b), however, it is not necessary that the domains of convergence of g and h or their range be $\mathbb{N}$, and the result is often useful for partial functions which take values outside $\mathbb{N}$.

CHAPTER 3

RECURSIVE FUNCTIONALS

In the first two sections of this chapter, we will extend the results of Chapter 2 to *M-functionals*, which may take partial functions (in addition to points in M) as arguments, and in Section 3C we will establish suitable (very strong) versions of the *Enumeration* and S_n^m Theorems of classical recursion for all partial algebras which imbed $\mathbf{N}_0$.

3A. Explicit functionals and simple fixed points

For some purposes, it is useful to derive a direct, "mathematical" characterization of the **M**-recursive partial functions which bypasses the formal language $\mathsf{R}(\tau)$. We will start with this here, and then we will establish a representation of the **M**-recursive partial functions in terms of the **simple fixed points** of **M**, an interesting and not well understood subclass of $\mathbf{rec}_0(\mathbf{M})$. These results are mostly technical, but they help considerably the exposition of the more basic theorems in the next two sections.

3A.1. **Definition.** A **functional** on a set M is any partial function

$$\alpha : M^n \times (M^{k_1} \rightharpoonup M_{\mathbb{B}}) \times \cdots \times (M^{k_m} \rightharpoonup M_{\mathbb{B}}) \rightharpoonup M_{\mathbb{B}} \tag{3-1}$$

with arguments in M and in the partial function spaces over M. The tuple

$$\text{arity}(\alpha) = (n, k_1, \ldots, k_m),$$

codes the input set of α, but we will often refer to "$\alpha(\vec{x}, \vec{p})$" or "$\alpha(\vec{x}, p_1, \ldots, p_m)$" when $\text{arity}(\alpha)$ is clear from the context or irrelevant.

Every partial function $\alpha : M^n \rightharpoonup M_{\mathbb{B}}$ is a functional, since we allow $m = 0$ in (3-1), and if the nullary functions a, b are among the givens, then

$$\alpha(p, q) = p(a, q(b))$$

is also a functional, with $n = 0, m = 2, k_1 = 2, k_2 = 1$. The most basic "real" functionals are the **evaluations**, one for each n,

$$\text{ev}_n(\vec{x}, p) = p(\vec{x}) \quad (\vec{x} \in M^n, p : M^n \rightharpoonup M_{\mathbb{B}}). \tag{3-2}$$

A functional as in (3-1) is **explicitly defined** in a τ-algebra $\mathbf{M}$ (or just **M-explicit**), if there is a term A of $\mathsf{R}(\tau, \zeta_1, \dots, \zeta_m)$ whose individual variables are all in the list $\vec{\mathsf{x}} \equiv (\mathsf{x}_1, \dots, \mathsf{x}_n)$ such that for all $\vec{x}, w, p_1, \dots, p_m$,

$$\alpha(\vec{x}, p_1, \dots, p_m) = w \iff (\mathbf{M}, p_1, \dots, p_m) \models A\{\vec{\mathsf{x}} :\equiv \vec{x}\} = w. \tag{3-3}$$

We set

$$\mathbf{exp}(\mathbf{M}) = \text{the set of all } \mathbf{M}\text{-explicit functionals.}$$

Since the explicit functionals of a τ-algebra $\mathbf{M}$ are exactly those definable by terms of $\mathsf{R}(\tau)$, we have immediately the following

3A.2. **Proposition.** *Suppose* $\mathbf{M}$ *is a partial algebra.*

(1) *If* $\alpha_1, \dots, \alpha_k$ *is a tuple of* $\mathbf{M}$*-explicit functionals with arities such that the system of equations*

$$\begin{aligned} p_1(\vec{x}_1) &= \alpha_1(\vec{x}_1, p_1, \dots, p_k) \\ &\vdots \\ p_k(\vec{x}_k) &= \alpha_k(\vec{x}_k, p_1, \dots, p_k) \end{aligned} \tag{3-4}$$

makes sense, then it has least $\sqsubseteq$*-solutions* $\overline{p}_1, \dots, \overline{p}_k$.

(2) *A partial function* $f : M^n \rightharpoonup M$ *is* $\mathbf{M}$*-recursive, if and only if* $f = \overline{p}_i$ *for a system of recursive equations with* $\mathbf{M}$*-explicit functionals as in* (1) *and some* i. ⊣

The next definition and result gives a language-independent characterization of $\mathbf{exp}(\mathbf{M})$.

3A.3. **Definition.** A set $\mathcal{F}$ of functionals on the universe M of a partial algebra $\mathbf{M}$ is **explicitly closed** (over $\mathbf{M}$) if it satisfies the following conditions.

(1) $\mathcal{F}$ contains the givens $f_1, \dots, f_L$ of $\mathbf{M}$, the (nullary) constant functions tt, ff, the projection functions $\vec{x} \mapsto x_i$, for every n and every i, $1 \leq i \leq n$, and the evaluation functionals in (3-2), for every n.

(2) $\mathcal{F}$ is closed under substitution (or composition)

$$\alpha(\vec{x}, \vec{y}, \vec{p}, \vec{q}) = \beta(\gamma(\vec{x}, \vec{p}), \vec{y}, \vec{q}) \tag{3-5}$$

(3) $\mathcal{F}$ is closed under branching,

$$\alpha(\vec{x}, \vec{p}) = \text{if } c(\vec{x}, \vec{p}) \text{ then } \beta(\vec{x}, \vec{p}) \text{ else } \gamma(\vec{x}, \vec{p}). \tag{3-6}$$

(4) $\mathcal{F}$ is closed under addition, identification and permutation of variables,

$$\begin{aligned} \alpha(x_1, \dots, x_n, p_1, \dots, p_l) \\ = \beta(x_{\pi(1)}, \dots, x_{\pi(m)}, p_{\rho(1)}, p_{\rho(2)}, \dots, p_{\rho(s)}), \end{aligned} \tag{3-7}$$

where $\pi : \{1, \dots, m\} \to \{1, \dots, n\}$ and $\rho : \{1, \dots, s\} \to \{1, \dots, l\}$.

In the sequel, we will refer collectively to addition, identification and permutation of variables as **variable shuffling**, a useful construct which justifies explicit definitions of the form

$$\alpha(x, y, p_1, p_2, p_3) = \beta(x, y, x, x, p_1, p_3, p_1, p_1),$$

and (in combination with (2)) very general substitution operations.[8]

3A.4. **Exercise.** Prove that if

$$\alpha(\vec{x}, \vec{p}) = \gamma(\delta(\vec{x}, \vec{p}), \vec{x}, \vec{p}),$$
$$\beta(\vec{x}, \vec{p}) = \gamma(\delta_1(\vec{x}, \vec{p}), \delta_2(\vec{x}, \vec{p}), \vec{x}, \vec{p})$$

and $\gamma, \delta, \delta_1, \delta_2 \in \mathbf{exp}(\mathbf{M})$, then $\alpha, \beta \in \mathbf{exp}(\mathbf{M})$.

It is also useful to introduce notation for a special (degenerate) case of branching which gives us a "negation" of sorts:

(3-8) $$\dot{\neg}\alpha(\vec{x}, \vec{p}) = \text{if } \alpha(\vec{x}, \vec{p}) \text{ then ff else tt},$$

(3-9) $$\ddot{\neg}\alpha(\vec{x}, \vec{p}) = \dot{\neg}\dot{\neg}\alpha(\vec{x}, \vec{p}).$$

This allows us to define explicitly the restriction of a functional to the domain of convergence of another:

3A.5. **Exercise.** Prove that for all functionals $\alpha(\vec{x}, \vec{p})$,

$$\alpha(\vec{x}, \vec{p})\downarrow \iff \ddot{\neg}\alpha(\vec{x}, \vec{p}) = \text{tt},$$

and hence

$$\text{if } \alpha(\vec{x}, \vec{p})\downarrow \text{ then } \beta(\vec{x}, \vec{p}) \text{ else } \perp = \text{if } \ddot{\neg}\alpha(\vec{x}, \vec{p}) \text{ then } \beta(\vec{x}, \vec{p}) \text{ else } \alpha(\vec{x}, \vec{p}).$$

3A.6. **Proposition.** *For each partial algebra* $\mathbf{M}$*, the set of* $\mathbf{M}$*-explicit functionals is the smallest set of functionals on* M *which is explicitly closed over* $\mathbf{M}$.

PROOF. We need to show that

(a) *the set* $\mathbf{exp}(\mathbf{M})$ *is explicitly closed*, and

(b) *if* F *is explicitly closed, then* $\mathbf{exp}(\mathbf{M}) \subseteq F$.

---

[8] We allow $m = 0$ in (3-7), in which case (by notational convention) $\{1, \dots, m\} = \emptyset$ and there is exactly one $\pi : \emptyset \to \{1, \dots, n\}$, the empty function; now (3-7) justifies the introduction of individual variables,

$$\alpha(x_1, \dots, x_n, p_1, \dots, p_l) = \beta(p_{\rho(1)}, p_{\rho(2)}, \dots, p_{\rho(s)}).$$

The same remark applies to the case $l = 0$, which justifies the introduction of function variables to a partial function.

For (a), we simply produce the terms which verify each of the required properties: for example, for substitution, if

$$\beta(s,\vec{y},\vec{q}) = w \iff (\mathbf{M},\vec{q}) \models A_\beta\{\mathsf{s} :\equiv s, \vec{\mathsf{y}} :\equiv \vec{y}\} = w,$$
$$\gamma(\vec{x},\vec{p}) = u \iff (\mathbf{M},\vec{p}) \models A_\gamma\{\vec{\mathsf{x}} :\equiv \vec{x}\} = u,$$

then

$$\begin{aligned}\beta(\gamma(\vec{x},\vec{p}),\vec{y},\vec{q}) = w &\iff (\mathbf{M},\vec{p},\vec{q}) \models A_\beta\{\mathsf{s} :\equiv A_\gamma\{\vec{\mathsf{x}} :\equiv \vec{x}\}, \vec{\mathsf{y}} :\equiv \vec{y}\} = w \\ &\iff (\mathbf{M},\vec{p},\vec{q}) \models \Big(A_\beta\{\mathsf{s} :\equiv A_\gamma\}\Big)\{\vec{\mathsf{x}} :\equiv \vec{x}, \vec{\mathsf{y}} :\equiv \vec{y}\} = w.\end{aligned}$$

The rest is (unfortunately) just as messy, but equally routine.

For (b), we show by induction on the construction of terms, that if F is explicitly closed and (3-3) holds, then $\alpha \in F$. For example, if

$$A \equiv r(B,C),$$

the free variables of B and C are in the lists $\vec{\mathsf{x}}, r, \vec{p}$, and

$$\beta(\vec{x},r,\vec{p}) = s \iff (\mathbf{M},r,\vec{p}) \models B\{\vec{\mathsf{x}} :\equiv \vec{x}\} = s,$$
$$\gamma(\vec{x},r,\vec{p}) = t \iff (\mathbf{M},r,\vec{p}) \models C\{\vec{\mathsf{x}} :\equiv \vec{x}\} = t$$

for suitable functionals $\beta, \gamma \in F$, we set

$$\alpha(\vec{x},r,\vec{p}) = r(\beta(\vec{x},r,\vec{p}),\gamma(\vec{x},r,\vec{p})),$$

and verify easily (using shuffling) that $\alpha \in F$ and

$$\alpha(\vec{x},r,\vec{p}) = w \iff (\mathbf{M},r,\vec{p}) \models A\{\vec{\mathsf{x}} :\equiv \vec{x}\} = w.$$

We will omit the details. ⊣

On some occasions, we will also need the following simple characterization of explicit functionals with at most one partial function argument:

3A.7. **Proposition.** *The class of all* $\mathbf{M}$*-explicit functionals* $\alpha(\vec{x},p)$ *with at most one* m*-ary partial function variable* p *is the smallest class* $\mathcal{F}$ *of such functionals satisfying the following:*

(1) $\mathcal{F}$ *contains the givens* $f_1,\dots,f_L$ *of* $\mathbf{M}$*, the* (nullary) *constant functions* $\mathtt{tt}, \mathtt{ff}$*, the projection functions* $\vec{x} \mapsto x_i$*, for every* n *and every* i*,* $1 \le i \le n$*, and the evaluation functional* $\mathrm{ev}_m(\vec{y},p) = p(\vec{y})$*.*

(2) $\mathcal{F}$ *is closed under the following scheme of composition:*

$$\alpha(\vec{x},\vec{y},p) = \beta(\gamma(\vec{x},p),\vec{y},p) \tag{3-10}$$

(3) $\mathcal{F}$ *is closed under branching,*

$$\alpha(\vec{x},p) = \text{if } c(\vec{x},p) \text{ then } \beta(\vec{x},p) \text{ else } \gamma(\vec{x},p).$$

(4) *$\mathcal{F}$ is closed under addition, identification and permutation of individual variables,*

$$\alpha(x_1,\dots,x_n,p) = \beta(x_{\pi(1)},\dots,x_{\pi(m)},p),$$

where $\pi : \{1,\dots,m\} \to \{1,\dots,n\}$. ⊣

We will leave the proof for Problem x3A.1.

These simple characterizations makes it possible to replace "syntactical" proofs by induction on terms by set theoretic arguments which, in some cases, are easier. Consider the following two properties of functionals, where for two tuples

$$\vec{p} = (p_1,\dots,p_m), \quad \vec{q} = (q_1,\dots,q_m)$$

of partial functions of the same length and such that $\text{arity}(p_j) = \text{arity}(q_j)$ $(j = 1,\dots,m)$, we set

$$\vec{p} \sqsubseteq \vec{q} \iff p_1 \sqsubseteq q_1 \,\&\, \cdots \,\&\, p_m \sqsubseteq q_m.$$

3A.8. **Definition.** Let $\alpha(\vec{x},\vec{p})$ be a functional on M.

(1) α is **monotone**, if for all $\vec{x}$, $\vec{p}$, $\vec{q}$ and w:

(3-11) $$\text{if } \alpha(\vec{x},\vec{p}) = w \,\&\, \vec{p} \sqsubseteq \vec{q}, \text{ then } \alpha(\vec{x},\vec{q}) = w.$$

(2) α is **continuous**, if for all $\vec{x}$, $\vec{p}$ and w:

(3-12) $$\text{if } \alpha(\vec{x},\vec{p}) = w, \text{ then there exists a tuple } \vec{q} \text{ such that } \vec{q} \text{ is finite } \,\&\, \vec{q} \sqsubseteq \vec{p} \,\&\, \alpha(\vec{x},\vec{q}) = w,$$

where $\vec{q} = (q_1,\dots,q_m)$ is **finite** if each q_j has finite domain of convergence.

An example of a monotone, discontinuous functional is the natural representation of the existential quantifier on M:

(3-13) $$E^{\#}_M(p) = \begin{cases} \text{tt}, & \text{if } (\exists x)[\ddot{\neg}p(x) = \text{tt}], \\ \text{ff}, & \text{if } (\forall x)[\ddot{\neg}p(x) = \text{ff}], \\ \bot, & \text{otherwise.} \end{cases}$$

If χ_R is the characteristic function of a unary relation $R \subseteq M$, then

$$E^{\#}_M(\chi_R)\!\downarrow \,\&\, E^{\#}_M(\chi_R) = \text{tt} \iff (\exists x)R(x).$$

3A.9. **Exercise.** Prove that $E^{\#}_M(p)$ is monotone, and that it is discontinuous when M is infinite.

3A.10. **Proposition.** *For each partial algebra* $\mathbf{M}$*, every* $\mathbf{M}$*-explicit functional on M is monotone and continuous.*

PROOF. For the monotonicity first, it is enough to show that the set F_m of all monotone functionals on M satisfies (1) – (4) in Definition 3A.3.

(1) Since every partial function is automatically monotone (as a functional), we only need check the monotonicity of the evaluation functionals, which is trivial.

(2) Substitution. Suppose that

$$\alpha(\vec{x},\vec{y},\vec{p},\vec{q}) = \beta(\gamma(\vec{x},\vec{p}),\vec{y},\vec{q}) = w, \tag{3-14}$$

where $\beta(u,\vec{y},\vec{q})$ and $\gamma(\vec{x},\vec{p})$ are given monotone functionals, so that there is a $u \in M$ such that

$$\gamma(\vec{x},\vec{p}) = u \text{ and } \beta(u,\vec{y},\vec{q}) = w. \tag{3-15}$$

If $\vec{p} \sqsubseteq \vec{p}', \vec{q} \sqsubseteq \vec{q}'$, then $\gamma(\vec{x},\vec{p}') = u$ and $\beta(u,\vec{y},\vec{q}') = w$ by the hypothesis, so $\alpha(\vec{x},\vec{y},\vec{p}',\vec{q}') = w$ as required.

The argument is similar for closure under branching (3) and trivial for closure variable shuffling.

To complete the proof of the Proposition, it suffices to show that the set F of all functionals on M which are both monotone and continuous is explicitly closed.

For (1) of Definition 3A.3, it is again enough to check that the evaluation functionals are continuous, and this is obvious: because if $p(\vec{x}) = w$ and $q = p\restriction\{\vec{x}\}$, then q is defined on just one tuple and $q(\vec{x}) = p(\vec{x}) = w$.

For substitution, suppose again that (3-14) and (3-15) hold, with monotone and continuous β and γ this time. So there are finite $\vec{p}_0 \sqsubseteq \vec{p}$, $\vec{q}_0 \sqsubseteq \vec{q}$ such that $\gamma(\vec{x},\vec{p}_0) = u$ and $\beta(u,\vec{y},\vec{q}_0) = w$, from which we derive again the required $\alpha(\vec{x},\vec{y},\vec{p}_0,\vec{q}_0) = w$.

The argument is similar for branching and variable shuffling. ⊣

The next result illustrates a somewhat more complex application of this method of proof.

3A.11. **Proposition.** *The set* $\mathbf{exp}(\mathbf{M})$ *is closed under the following scheme of* λ**-substitution***:*

$$\alpha(\vec{x},\vec{y},\vec{p},\vec{q}) = \beta(\vec{x},\lambda(\vec{t})\gamma(\vec{t},\vec{y},\vec{p}),\vec{q}). \tag{3-16}$$

PROOF. Fix an $\mathbf{M}$-explicitly closed set of functionals F, and let

$$F' = \{\beta \in F \mid \text{if } \alpha \text{ is defined from } \beta \text{ and any } \gamma \in F \text{ by (3-16), then } \alpha \in F\}.$$

It will be enough to show that F' is $\mathbf{M}$-explicitly closed, because the application of this to $\mathbf{exp}(\mathbf{M})$ will give $\mathbf{exp}(\mathbf{M})' = \mathbf{exp}(\mathbf{M})$, so that $\mathbf{exp}(\mathbf{M})$ is closed under (3-16). This argument is quite routine, if a bit messy.

First, F' contains all the partial functions in F, which have no function arguments so that (3-16) cannot be applied, and it contains the evaluation functionals (the crucial case), since

$$\mathrm{ev}_n(\vec{x}, \lambda(\vec{t})\gamma(\vec{t}, \vec{y}, \vec{p})) = \gamma(\vec{x}, \vec{y}, \vec{p}).$$

To show closure of F' under substitution, suppose

$$\beta(\vec{x}, r, \vec{q}) = \delta_1(\delta_2(\vec{x}_1, r, \vec{q}_1), \vec{x}_2, \vec{q}_2)$$

with $\vec{x} = \vec{x}_1, \vec{x}_2, \vec{q} = \vec{q}_1, \vec{q}_2$ and $\delta_1, \delta_2 \in F'$, and set

$$\delta^*(\vec{x}_1, \vec{y}, \vec{p}, \vec{q}_1) = \delta_2(\vec{x}_1, \lambda(\vec{t})\gamma(\vec{t}, \vec{y}, \vec{p}), \vec{q}_1);$$

now $\delta^* \in F'$ by the induction hypothesis,

$$\alpha(\vec{x}, \vec{y}, \vec{p}, \vec{q}) = \beta(\vec{x}, \lambda(\vec{t})\gamma(\vec{t}, \vec{y}, \vec{p}), \vec{q}) = \delta_1(\delta^*(\vec{x}_1, \vec{y}, \vec{p}, \vec{q}_1), \vec{x}_2, \vec{q}_2),$$

and then $\alpha \in F'$ by the closure of F' under substitution (and shuffling).

The closure of F' under branching is a bit simpler than this, and its closure under variable shuffling is trivial. ⊣

The operation of λ-substitution is usually applied in the simpler form of the next Exercise, which is easily reduced to (3-16) using shuffling:

3A.12. **Exercise.** Prove that if β and γ are **M**-explicit, then so is

$$\alpha(\vec{x}, \vec{p}) = \beta(\vec{x}, \lambda(\vec{t})\gamma(\vec{t}, \vec{x}, \vec{p}), \vec{p}).$$

It is not yet clear why we defined the **M**-recursive partial functions using systems rather than single recursive equations. Actually, the least fixed points of systems with just one recursive equation are important, but, as we will see, they do not (as a rule) exhaust the **M**-recursive partial functions.

3A.13. **Definition.** A functional $\alpha(\vec{x}, p)$ of one partial function is **operative** if its arity is (n, n), so that the recursive equation

$$p(\vec{x}) = \alpha(\vec{x}, p) \tag{3-17}$$

makes sense; and a partial function $g : M^n \rightharpoonup M_{\mathbb{B}}$ is a **simple fixed point** of the partial algebra **M**, if it is the least fixed point of an **M**-explicit, operative functional, i.e., $g = \overline{p}$ in (3-17) with an **M**-explicit α. We set

$$\mathbf{fix}(\mathbf{M}) = \{g : M^n \rightharpoonup M_{\mathbb{B}} \mid g \text{ is a simple fixed point of } \mathbf{M}\}. \tag{3-18}$$

The **M**-recursive partial functions can be "reduced" to the fixed points of **M** if **M** has two "distinguished points", as follows.

3A.14. **Definition.** A point $a \in M$ is **distinguished** in a partial algebra **M** if the nullary constant a and the characteristic function

$$\chi_a(t) = \chi_{\{a\}}(t) = \text{if } (t = a) \text{ then tt else ff}$$

of the singleton $\{a\}$ are both **M**-recursive.

For example, every number n is distinguished in the algebras $\mathbf{N}_0$ and $\mathbf{N}^2$, and the identity e is distinguished in every group $(G, e, \cdot, ^{-1}, =)$.

3A.15. **Proposition.** *If* $\mathbf{M}$ *has two distinguished points* $0, 1$*, then a partial function* $f : M^n \rightharpoonup M$ *is* $\mathbf{M}$*-recursive if and only if there is a simple fixed point* $g : M^{k+n} \rightharpoonup M$ *of* $\mathbf{M}$ *such that for all* $\vec{x}$*,*

$$f(\vec{x}) = g(\vec{0}^k, \vec{x}) \qquad (\vec{0}^k = \underbrace{0, \dots, 0}_{k}). \tag{3-19}$$

PROOF. It is enough to show that every $\mathbf{M}$-recursive $f(\vec{x})$ satisfies (3-19) with a simple fixed point $g(\vec{u}, \vec{x})$. To keep the notation simple, suppose that $f = \overline{p}_3$ for the system of three equations

$$\begin{aligned} p_1(\vec{x}) &= \alpha_1(\vec{x}, p_1, p_2, p_3), \\ p_2(\vec{y}) &= \alpha_2(\vec{y}, p_1, p_2, p_3), \\ p_3(\vec{z}) &= \alpha_3(\vec{z}, p_1, p_2, p_3) \end{aligned} \tag{3-20}$$

with explicit functionals $\alpha_1, \alpha_2, \alpha_3$, and

$$\vec{x} = (x_1, \dots, x_n), \quad \vec{y} = (y_1, \dots, y_m), \quad \vec{z} = (z_1, \dots, z_l).$$

Put

$$\alpha(s, t, \vec{x}, \vec{y}, \vec{z}, r) = \begin{cases} \alpha_1(\vec{x}, \lambda(\vec{x}')r(0, 1, \vec{x}', \vec{0}^m, \vec{0}^l), \lambda(\vec{y}')r(1, 0, \vec{0}^n, \vec{y}', \vec{0}^l), \\ \qquad \lambda(\vec{z}')r(0, 0, \vec{0}^n, \vec{0}^m, \vec{z}')) & \text{if } s = 0 \ \& \ t \neq 0, \\ \alpha_2(\vec{y}, \lambda(\vec{x}')r(0, 1, \vec{x}', \vec{0}^m, \vec{0}^l), \lambda(\vec{y}')r(1, 0, \vec{0}^n, \vec{y}', \vec{0}^l), \\ \qquad \lambda(\vec{z}')r(0, 0, \vec{0}^n, \vec{0}^m, \vec{z}')) & \text{if } s \neq 0 \ \& \ t = 0, \\ \alpha_3(\vec{z}, \lambda(\vec{x}')r(0, 1, \vec{x}', \vec{0}^m, \vec{0}^l), \quad \lambda(\vec{y}')r(1, 0, \vec{0}^n, \vec{y}', \vec{0}^l), \\ \qquad \lambda(\vec{z}')r(0, 0, \vec{0}^n, \vec{0}^m, \vec{z}')) & \text{otherwise,} \end{cases}$$

where $r : M^{2+n+m+l} \rightharpoonup M$ and $\vec{0}^n, \vec{0}^m, \vec{0}^l$ are defined as above, so that the expressions make sense. Let $\overline{r}$ be the least solution of the equation

$$r(s, t, \vec{x}, \vec{y}, \vec{z}) = \alpha(s, t, \vec{x}, \vec{y}, \vec{z}, r); \tag{3-21}$$

it then suffices to prove that for all $\vec{z}$,

$$f(\vec{z}) = \overline{p}_3(\vec{z}) = \overline{r}(0, 0, \vec{0}^n, \vec{0}^m, \vec{z}). \tag{3-22}$$

Let first

$$\begin{aligned} p'_1(\vec{x}) &= \overline{r}(0, 1, \vec{x}, \vec{0}^m, \vec{0}^l), \\ p'_2(\vec{y}) &= \overline{r}(1, 0, \vec{0}^n, \vec{y}, \vec{0}^l), \\ p'_3(\vec{z}) &= \overline{r}(0, 0, \vec{0}^n, \vec{0}^m, \vec{z}); \end{aligned}$$

it is quite easy to verify (by just plugging in) that (p'_1, p'_2, p'_3) satisfy the system (3-20), and hence

$$\overline{p}_1 \sqsubseteq p'_1, \quad \overline{p}_2 \sqsubseteq p'_2, \quad \overline{p}_3 \sqsubseteq p'_3. \tag{3-23}$$

Next let

$$r'(s,t,\vec{x},\vec{y},\vec{z}) = \begin{cases} \overline{p}_1(\vec{x}), & \text{if } s = 0 \ \& \ t \neq 0, \\ \overline{p}_2(\vec{y}), & \text{if } s \neq 0 \ \& \ t = 0, \\ \overline{p}_3(\vec{z}), & \text{otherwise;} \end{cases}$$

by straight computation again, it is clear that r' satisfies (3-21), so that

$$\overline{r} \sqsubseteq r'. \tag{3-24}$$

Now (3-23) and (3-24) imply together that

$$\overline{r}(s,t,\vec{x},\vec{y},\vec{z}) = \begin{cases} \overline{r}_1(\vec{x}), & \text{if } s = 0 \ \& \ t \neq 0, \\ \overline{r}_2(\vec{y}), & \text{if } s \neq 0 \ \& \ t = 0, \\ \overline{r}_3(\vec{z}), & \text{otherwise,} \end{cases}$$

which gives the required (3-22). ⊣

Thus this simple reduction of mutual to single recursion works for $\mathbf{N}_0$ and $\mathbf{N}^2$—but in effect it works for all partial algebras, because of the next definition and result.

3A.16. **Definition.** Suppose $\mathbf{M}_1$ and $\mathbf{M}_2$ are partial algebras of possibly different characteristics, but such that $M_1 \subseteq M_2$. We say that $\mathbf{M}_2$ is an **inessential extension** of $\mathbf{M}_1$, if for every $f : M_1 \rightharpoonup M_{\mathbb{B}}$,

$$f \in \mathbf{rec}_0(\mathbf{M}_1) \iff f = g \restriction M_1^n \text{ for some } g \in \mathbf{rec}_0(\mathbf{M}_2).$$

3A.17. **Proposition.** *Each partial algebra* $\mathbf{M}$ *has an inessential extension with two distinguished points.*

PROOF. We choose some $a, b \notin M$ and set

$$\mathbf{M}[a,b] = (M \cup \{a,b\}, f_1, \dots, f_L, a, b, \chi_a, \chi_b),$$

where $f_1, \dots, f_L$ are the givens of $\mathbf{M}$, extended to $M \cup \{a,b\}$ so that if $x_i \in \{a,b\}$ for some i, then $f_i(x_1, \dots, x_{n_i}) = \bot$.

If $\mathbf{M}$ is a τ-algebra with $\tau = \{\mathsf{f}_1, \dots, \mathsf{f}_L\}$, then $\mathbf{M}[a,b]$ is a $\tau[a,b]$-algebra with the extended vocabulary

$$\tau[a,b] = \tau \cup \{a, \chi_a, b, \chi_b\},$$

and every term or program of $\mathsf{R}(\tau)$ is also a term or program of $\mathsf{R}(\tau[a,b])$. Moreover, if E is any $\mathsf{R}(\tau)$-program with principal symbol ζ and $\vec{x} \in M^n$, then every computation

$$\zeta : \vec{x} \to \dots \to : w$$

of $\mathcal{T}(E, \mathbf{M})$ is also (easily) a computation of $\mathcal{T}(E, \mathbf{M}[a,b])$—and hence the only computation of $\mathcal{T}(E, \mathbf{M}[a,b])$ which starts with $\zeta : \vec{x}$, since $\mathcal{T}(E, \mathbf{M}[a,b])$

is a deterministic transition system. In other words, for $\vec{x} \in M^n$ and $w \in M[a,b]_{\mathbb{B}}$,

$$\mathbf{M}, E \vdash \zeta(\vec{x}) = w \iff \mathbf{M}[a,b], E \vdash \zeta(\vec{x}) = w,$$

so that the partial function $\overline{\zeta} : M^n \rightharpoonup M_{\mathbb{B}}$ computed by E in $\mathbf{M}$ is also computed by E in $\mathbf{M}[a,b]$. Thus every $\mathbf{M}$-recursive partial function is also $\mathbf{M}[a,b]$-recursive.

The converse requires a little more work since $\mathbf{M}[a,b]$ has a larger universe than $\mathbf{M}$ (in addition to the four, new givens) and "provides more room" for its recursive definitions. The idea is to code each partial function $p : M[a,b]^n \rightharpoonup M[a,b]_{\mathbb{B}}$ using $2 \cdot 3^n$ partial functions on M, as follows.

For each n, let (for this proof)

$$F_n = (\{1, \ldots, n\} \to \{0,1,2\})$$

and for each $\pi \in F_n$ and each $\vec{x} \in M^n$, set

$$x_i^{\pi} = \begin{cases} x_i, & \text{if } \pi(i) = 0, \\ a, & \text{if } \pi(i) = 1, \\ b, & \text{if } \pi(i) = 2, \end{cases}$$

so that, for example, if $\rho_0(i) = 0$, $\rho_1(i) = 1$ and $\rho_2(i) = 2$ for all i, then

$$\vec{x}^{\rho_0} = (x_1, \ldots, x_n), \quad \vec{x}^{\rho_1} = (a, \ldots, a), \quad \vec{x}^{\rho_2} = (b, \ldots, b).$$

It is clear that for each n, the operation $(\vec{x}, \pi) \mapsto \vec{x}^{\pi}$ codes $M[a,b]^n$, although some tuples (like (a,a,a)) have many codes.

For each $p : M[a,b]^n \rightharpoonup M[a,b]_{\mathbb{B}}$ and each $\pi \in F_n$, let

$$m = m(\pi) = |\{i \mid \pi(i) = 0\}|,$$

let $\pi^*(1), \ldots, \pi^*(m)$ enumerate in increasing order the set $\{i \mid \pi(i) = 0\}$ (which may be empty), and define $p^{\pi} : M^m \rightharpoonup M_{\mathbb{B}}$ and $p^{\pi}_{\delta} : M^m \rightharpoonup \{\mathrm{tt}, \mathrm{ff}\}$ by

$$p^{\pi}(x_{\pi^*(1)}, \ldots, x_{\pi^*(m)}) = \begin{cases} p(\vec{x}^{\pi}), & \text{if } p(\vec{x}^{\pi}) \in M_{\mathbb{B}}, \\ \mathrm{tt}, & \text{if } p(\vec{x}^{\pi}) = a, \\ \mathrm{ff}, & \text{if } p(\vec{x}^{\pi}) = b, \\ \bot, & \text{otherwise}, \end{cases}$$

$$p^{\pi}_{\delta}(x_{\pi^*(1)}, \ldots, x_{\pi^*(m)}) = \begin{cases} \mathrm{tt}, & \text{if } p(\vec{x}^{\pi}) \in M_{\mathbb{B}}, \\ \mathrm{ff}, & \text{if } p(\vec{x}^{\pi}) \in \{a,b\}, \\ \bot, & \text{otherwise}, \end{cases}$$

with the convention that p^{π} and p^{π}_{δ} are nullary if $m(\pi) = 0$. For example, if $\chi_a(t)$ is the (unary) characteristic function of the singleton $\{a\}$, then there

are just three functions on $\{1\}$ to $\{0,1,2\}$,

$$\rho_0(1) = 0, \rho_1(1) = 1, \text{ and } \rho_2(1) = 2, \tag{3-25}$$

and

$$\begin{aligned} (\chi_a)^{\rho_0}(t) = \chi_a(t) = \mathrm{ff}, \quad (\chi_a)^{\rho_0}_\delta(t) = \mathrm{tt}, \\ (\chi_a)^{\rho_1} = \chi_a(a) = \mathrm{tt}, \quad (\chi_a)^{\rho_1}_\delta = \mathrm{tt}, \\ (\chi_a)^{\rho_2} = \chi_a(b) = \mathrm{ff}, \quad (\chi_a)^{\rho_2}_\delta = \mathrm{tt}. \end{aligned}$$

where the last four of these coding functions on M are nullary.

It is clear that every $p : M[a,b]^n \rightharpoonup M[a,b]_{\mathbb{B}}$ is coded (determined) by the 3^n pairs of partial functions p^π, p^π_δ as π varies over F_n, cf. Problem x3A.2.

To simplify (somewhat) the notation, let us associate with each k-ary function variable p and each $\pi : \{1,\dots,k\} \to \{0,1,2\}$, two variables

$$\mathsf{p}^\pi, \quad \mathsf{p}^\pi_\delta$$

which denote p^π and p^π_δ when $\mathsf{p} := p$, and let

$$\vec{\mathsf{p}} = \mathsf{p}^{\pi_1}, \dots, \mathsf{p}^{\pi_l}, \mathsf{p}^{\pi_1}_\delta, \dots, \mathsf{p}^{\pi_l}_\delta$$

be some fixed enumeration of the $l = 3^k$ such pairs of variables.

Lemma. For each $\mathbf{M}[a,b]$*-explicit functional* $\alpha(\vec{x}, \mathsf{p})$ *of* n *individual variables and one* k*-ary function variable, and for each* $\pi \in F_n$*, there are two* $\mathbf{M}$*-explicit functionals*

$$\alpha^\pi(\vec{x}, \vec{\mathsf{p}}), \quad \alpha^\pi_\delta(\vec{x}, \vec{\mathsf{p}})$$

in the indicated variables with the following property: for any partial function $p : M[a,b]^k \rightharpoonup M[a,b]_{\mathbb{B}}$*, if* $q(\vec{x}) = \alpha(\vec{x}, p)$ *and* $\vec{p}$ *are the partial functions on* M *to* $M_{\mathbb{B}}$ *which code* p*, then for every* $\pi \in F_n$*,*

$$q^\pi(x_{\pi^*(1)}, \dots, x_{\pi^*(m)}) = \alpha^\pi(\vec{x}, \vec{p}), \qquad q^\pi_\delta(x_{\pi^*(1)}, \dots, x_{\pi^*(m)}) = \alpha^\pi_\delta(\vec{x}, \vec{p}).$$

Proof. It is enough to show that the class G of functionals on $M[a,b]$ with at most one partial function variable which satisfy the Lemma satisfies the four conditions listed in Proposition 3A.7.

(1) is basically trivial for the given partial functions of $\mathbf{M}[a,b]$: for each f_i and each π we set

$$f_i^\pi(x_{\pi^*(1)}, \dots, x_{\pi^*(m)}) = \begin{cases} f_i(\vec{x}), & \text{if } \pi(i) = 0 \text{ for all } i = 1, \dots, n_i, \\ \bot, & \text{otherwise,} \end{cases}$$

$$f_{i,\delta}^\pi(x_{\pi^*(1)}, \dots, x_{\pi^*(m)}) = \begin{cases} \mathrm{tt}, & \text{if } \pi(i) = 0 \ \&\ \text{for all } i = 1, \dots, n_i \ \&\ f_i(\vec{x})\!\downarrow, \\ \bot, & \text{otherwise,} \end{cases}$$

and for the nullary, new givens a, b,

$$a^\emptyset = \mathrm{tt}, \qquad a^\emptyset_\delta = \mathrm{ff}, \quad b^\emptyset = \mathrm{ff}, \qquad b^\emptyset_\delta = \mathrm{ff}.$$

For the other, unary new given χ_a we defined the required function(al)s above, and the definitions are similar for χ_b.

For the evaluation functional

$$\alpha(\vec{x}, p) = p(\vec{x})$$

with $\vec{x} = (x_1, \dots, x_k)$,

$$\alpha^\pi(\vec{x}, \vec{p}) = p^\pi(\vec{x}), \quad \alpha^\pi_\delta(\vec{x}, \vec{p}) = p^\pi_\delta(\vec{x}).$$

(2) To show the closure of the class G under the composition in a notationally simple case (skipping the $\vec{y}$), suppose

$$q(\vec{x}) = \alpha(\vec{x}, p) = \beta(\gamma(\vec{x}, p), p),$$

for a fixed p, and set

$$q_\gamma(\vec{x}) = \gamma(\vec{x}, p), \quad q_\beta(u) = \beta(u, p),$$

so that

$$q(\vec{x}) = \beta(q_\gamma(\vec{x}), p) = q_\beta(q_\gamma(\vec{x})),$$

and for any $\pi \in F_n$,

$$q^\pi(x_{\pi^*(1)}, \dots, x_{\pi^*(m)}) = q(\vec{x}^\pi) = q_\beta(q_\gamma(\vec{x}^\pi)) = q_\beta(\gamma^\pi(\vec{x}, p)).$$

We now use β^{ρ_i} for the three $\rho \in F_1$ defined in (3-25) to compute

$$\alpha^\pi(x_{\pi^*(1)}, \dots, x_{\pi^*(m)}, p) = q^\pi(x_{\pi^*(1)}, \dots, x_{\pi^*(m)})$$
$$= \begin{cases} \beta^{\rho_0}(\gamma^\pi(x_{\pi^*(1)}, \dots, x_{\pi^*(m)}, p), p), \\ \quad \text{if } \gamma^\pi_\delta(x_{\pi^*(1)}, \dots, x_{\pi^*(m)}) = \text{tt}, \\ \beta^{\rho_1}(p) \\ \quad \text{if } \gamma^\pi_\delta(x_{\pi^*(1)}, \dots, x_{\pi^*(m)}) = \text{ff} \ \& \ \gamma^\pi(x_{\pi^*(1)}, \dots, x_{\pi^*(m)}) = \text{tt}, \\ \beta^{\rho_2}(p) \\ \quad \text{if } \gamma^\pi_\delta(x_{\pi^*(1)}, \dots, x_{\pi^*(m)}) = \text{ff} \ \& \ \gamma^\pi(x_{\pi^*(1)}, \dots, x_{\pi^*(m)}) = \text{ff}, \\ \bot, \ \text{otherwise} \end{cases}$$

The definition of $\alpha^\pi_\delta(x_{\pi^*(1)}, \dots, x_{\pi^*(m)}, p)$ is equally simple in concept and messy in execution, and the arguments for branching and individual variable shuffling are a bit easier. $\dashv$ (Lemma)

To prove the Proposition now, we associate with each recursive equation

(3-26) $$p(\vec{x}) = \alpha(\vec{x}, p)$$

in $\mathbf{M}[a,b]$ the system of $2l = 2\cdot 3^n$ equations

$$\begin{aligned}
p^{\pi_1}(x_{\pi_1^*(1)},\dots,x_{\pi_1^*(m_1)}) &= \alpha^{\pi_1}(x_{\pi_1^*(1)},\dots,x_{\pi_1^*(m_1)},\vec{p})\\
p_\delta^{\pi_1}(x_{\pi_1^*(1)},\dots,x_{\pi_1^*(m_1)}) &= \alpha_\delta^{\pi_1}(x_{\pi_1^*(1)},\dots,x_{\pi_1^*(m_1)},\vec{p})\\
&\vdots\\
p^{\pi_l}(x_{\pi_l^*(1)},\dots,x_{\pi_l^*(m_l)}) &= \alpha^{\pi_l}(x_{\pi_l^*(1)},\dots,x_{\pi_l^*(m_l)},\vec{p})\\
p_\delta^{\pi_l}(x_{\pi_l^*(1)},\dots,x_{\pi_l^*(m_l)}) &= \alpha_\delta^{\pi_l}(x_{\pi_l^*(1)},\dots,x_{\pi_l^*(m_l)},\vec{p})
\end{aligned} \tag{3-27}$$

in $\mathbf{M}$.

If $\overline{p} : M[a,b] \rightharpoonup BM[a,b]$ is the least solution of (3-26) in $\mathbf{M}[a,b]$ and $\tilde{p}^\pi, \tilde{p}_\delta^\pi$ (with $p : \{1,\dots,n\} \to \{0,1,2\}$) are the least solutions of (3-27) in $\mathbf{M}$, then (by a simple fixed point argument),

$$\overline{p}^\pi = \tilde{p}^\pi, \quad \overline{p}_\delta^\pi = \tilde{p}_\delta^\pi$$

for every $\pi : \{1,\dots,\} \to \{0,1,2\}$, so that $\overline{p}^\pi$ *and* $\overline{p}_\delta^\pi$ *are* $\mathbf{M}$*-recursive, for every simple fixed point* $\overline{p}$ *of* $\mathbf{M}[a,b]$.

Finally, if $g : M[a,b] \rightharpoonup M[a,b]_{\mathbb{B}}$ is $\mathbf{M}[a,b]$-recursive, then (by 3A.15) there is a simple fixed point

$$\overline{p} : M[a,b]^{k+n} \rightharpoonup M[a,b]_{\mathbb{B}}$$

and a sequence $\vec{a}^k$ of a'a such that

$$g(\vec{x}) = \overline{p}(\vec{a}^k, \vec{x});$$

and if we let $\pi(1) = \cdots = \pi(k) = 1$, $\pi(k+1) = \cdots = \pi(k+n) = 0$, then

$$g^\pi(\vec{x}) = \overline{p}(\vec{a}^k, \vec{x}) \qquad (\vec{x} \in M^n),$$

and

$$g \restriction M(\vec{x}) = \begin{cases} g^\pi(\vec{x}), & \text{if } g_\delta^\pi(\vec{x}) = \mathrm{tt},\\ \perp, & \text{otherwise,}\end{cases}$$

so that $g \restriction M$ is $\mathbf{M}$-recursive, which completes the proof. ⊣

Thus, for any $\mathbf{M}$, the $\mathbf{M}$-recursive partial functions are (essentially) just the "distinguished sections" of $\mathbf{M}$-simple fixed points, a very useful fact which simplifies many arguments. It also suggests that perhaps all $\mathbf{M}$-recursive partial functions are $\mathbf{M}$-fixed points, which, however, is far from the truth, cf. Problems x3A.3 – *x3A.8. The structure and extent of $\mathbf{fix}(\mathbf{M})$ is not well understood even for the most familiar algebras, like $\mathbf{N}_0$ or $\mathbf{N}$, and we will return to the question in the sequel.

One of the nice features of simple fixed points is that they can be defined very simply from the functionals which define them, by the following familiar construction.

3A.18. **Theorem (Continuous Least Fixed Points).** *For each monotone and continuous, operative functional $\alpha(\vec{x}, p)$, define its* **iterates** *by the recursion*

$$\overline{p}^0(\vec{x}) = \alpha(\vec{x}, \emptyset), \qquad \overline{p}^{m+1}(\vec{x}) = \alpha(\vec{x}, \overline{p}^m). \tag{3-28}$$

(1) *For all m, $\overline{p}^m \sqsubseteq \overline{p}^{m+1}$, so that $\overline{p} = \bigcup_m \overline{p}^m : M^n \rightharpoonup M$.*

(2) *For all $\vec{x}$, $\overline{p}(\vec{x}) = \alpha(\vec{x}, \overline{p})$.*

(3) *For every partial function $q : M^n \rightharpoonup M$,*

$$\text{if } (\forall \vec{x}, w)[\alpha(\vec{x}, q) = w \Longrightarrow q(\vec{x}) = w], \text{ then } \overline{p} \sqsubseteq q. \tag{3-29}$$

PROOF. (1) is proved by induction on m, the basis being trivial by the monotonicity hypothesis since $\emptyset \sqsubseteq \overline{p}^0$. In the induction step:

$$\begin{aligned} \overline{p}^{m+1}(\vec{x}) = w &\Longrightarrow \alpha(\vec{x}, \overline{p}^m) = w \\ &\Longrightarrow \alpha(\vec{x}, \overline{p}^{m+1}) = w \quad \text{(ind. hyp. and mon.)} \\ &\Longrightarrow \overline{p}^{m+2}(\vec{x}) = w. \end{aligned}$$

(2) In one direction,

$$\begin{aligned} \overline{p}(\vec{x}) = w &\Longrightarrow \overline{p}^{m+1}(\vec{x}) = w \quad \text{(for some } m) \\ &\Longrightarrow \alpha(\vec{x}, \overline{p}^m) = w \\ &\Longrightarrow \alpha(\vec{x}, \overline{p}) = w \qquad \text{(mon.)}, \end{aligned}$$

and so $\overline{p}(\vec{x}) = w \Longrightarrow \alpha(\vec{x}, \overline{p}) = w$. For the converse, suppose that

$$\alpha(\vec{x}, \overline{p}) = w.$$

By the continuity of α, there is a finite partial function $q \sqsubseteq \overline{p}$ such that $\alpha(\vec{x}), q) = w$. Thus for each $\vec{x}$ such that $q(\vec{x})\downarrow$, there is some $m = m(\vec{x})$ such that $\overline{p}^m(\vec{x}) = q(\vec{x})$; and if

$$m = \max\{m(\vec{x}) \mid q(\vec{x})\downarrow\},$$

then $q \sqsubseteq \overline{p}^m$, so that by monotonicity again, $\overline{p}^{m+1}(\vec{x}) = \alpha(\vec{x}, \overline{p}^m) = w$, which is what we needed to show.

(3) Assume that "α presses q down", i.e.,

$$(\forall \vec{x}, w)[\alpha(\vec{x}, q) = w \Longrightarrow q(\vec{x}) = w].$$

To infer $\overline{p} \sqsubseteq q$ from this, we check by induction that $\overline{p}^m \sqsubseteq q$, the basis $\emptyset \sqsubseteq q$ being trivial. Finally, in the induction step:

$$\begin{aligned} \overline{p}^{m+1}(\vec{x}) = w &\Longrightarrow \alpha(\vec{x}, \overline{p}^m) = w \\ &\Longrightarrow \alpha(\vec{x}, q) = w \qquad \text{(ind. hyp. and mon.)} \\ &\Longrightarrow q(\vec{x}) = w \qquad \text{(hyp.).} \end{aligned}$$

⊣

3A.19. **Definition (Stages).** Suppose $\alpha(\vec{x}, p)$ is a monotone, continuous and operative functional on M with least fixed point $\overline{p}$. We set:

$$|\vec{x}|_\alpha = \begin{cases} \text{the least } m \text{ such that } \overline{p}^m(\vec{x})\downarrow, & \text{if } \overline{p}(\vec{x})\downarrow, \\ \infty, & \text{otherwise.} \end{cases} \tag{3-30}$$

This natural assignment of a **stage** to each point of convergence of $\overline{p}$ is characteristic of recursive definitions, and the most important tool for their study. We will study it in detail later in these notes, but it is already very useful for some of the problems of this section.

Problems for Section 3A

x3A.1. Prove Proposition 3A.7.

x3A.2. With the notation of the proof of Proposition 3A.17, suppose $p : M \cup \{a, b\} \rightharpoonup M \cup \{a, b, \mathtt{tt}, \mathtt{ff}\}$. Derive formulas which compute $p(x, y), p(x, a)$ and $p(b, y)$ from the associated coding partial functions p^π, p^π_δ for suitable π's.

3A.20. **Definition.** The **iterates** ϕ^k of a (total) function $\phi : M \to M$ are defined by the recursion,

$$\phi^0(s) = s, \qquad \phi^{k+1}(s) = \phi(\phi^k(s)). \tag{3-31}$$

A (total) function $\phi : \mathbb{N} \to \mathbb{N}$ is **increasing** if

$$s < t \Longrightarrow \phi(s) < \phi(t),$$

in which case all its iterates ϕ^k are (easily) also increasing.

We use increasing functions to measure the growth rates of partial functions and functionals on $\mathbb{N}$: set

$$||\vec{x}|| = \max\{x_1, \ldots, x_n\},$$

and call $p : \mathbb{N}^n \rightharpoonup \mathbb{N}$ is ϕ^k-**bounded** with $k > 0$, if

$$(\forall \vec{x})[p(\vec{x})\downarrow \Longrightarrow p(\vec{x}) \leq \phi^k(||\vec{x}||)].$$

A functional $\alpha(\vec{x}, \vec{p})$ is ϕ^l-**bounded** with $l > 0$, if, for all k,

if $p_1, \ldots, p_m$ are all ϕ^k-bounded,

$$\text{then } (\forall \vec{x})[\alpha(\vec{x}, \vec{p})\downarrow \Longrightarrow \alpha(\vec{x}, \vec{p}) \leq \phi^{lk}(||\vec{x}||).$$

This simply means that if, for any $\vec{p}$, we set

$$f(\vec{x}) = \alpha(\vec{x}, \vec{p})$$

and $p_1, \ldots, p_m$ are all ϕ^k-bounded, then f is ϕ^{lk}-bounded.

x3A.3. Prove that if $\phi : \mathbb{N} \to \mathbb{N}$ is increasing, then for all k, m,

$$k < m \Longrightarrow (\forall s)[\phi^k(s) \leq \phi^l(s)].$$

Infer that if p is ϕ^k-bounded and $0 < k < m$, then p is also ϕ^m-bounded, and the same for functionals.

x3A.4. Suppose $\mathbf{M} = (\mathbb{N}, f_1, \ldots, f_L)$ is a partial algebra with universe $\mathbb{N}$, ϕ is an increasing function on $\mathbb{N}$, and every given f_i is ϕ-bounded. Show that every $\mathbf{M}$-explicit functional $\alpha(\vec{x}, \vec{p})$ is ϕ^l-bounded for some l.

x3A.5. Suppose $\alpha(\vec{x}, p)$ is a monotone and continuous operative functional on M and $|\vec{x}|_\alpha = m$. Prove that for each $i < m$, there is some tuple $\vec{x}_i$ such that $|\vec{x}_i|_\alpha = i$.

*x3A.6. Suppose $\alpha(\vec{x}, p)$ is a monotone and continuous, operative functional on $\mathbb{N}$, with $\vec{x} = (x_1, \ldots, x_n)$ ranging over n-tuples. Let $\overline{p} : \mathbb{N}^n \rightharpoonup \mathbb{N}$ be its least fixed point and $|\vec{x}| = |\vec{x}|_\alpha$ the stage of each $\vec{x}$, and suppose that the domain of convergence of $\overline{p}$ is infinite. Prove that

$$\text{for infinitely many } \vec{x}, |\vec{x}| \leq (||\vec{x}|| + 1)^n. \tag{3-32}$$

HINT: : Do the problem first for $n = 1$, where it is easier to see what goes on; the precise (somewhat better) result in that case is that

$$\text{for infinitely many } x, |x| \leq x. \tag{3-33}$$

*x3A.7. (1) Suppose $\mathbf{M} = (\mathbb{N}, f_1, \ldots, f_L)$ is a total algebra with universe $\mathbb{N}$, and suppose that $\phi : \mathbb{N} \to \mathbb{N}$ is a (total) function with the following properties:

(a) ϕ is strictly increasing, i.e., $s < t \Longrightarrow \phi(s) < \phi(t)$.
(b) Each of the givens of $\mathbf{M}$ is bounded by ϕ, i.e.,

$$(\forall \vec{x})[f_i(\vec{x}) \leq \phi(||\vec{x}||)]. \tag{3-34}$$

Prove that for every unary simple fixed point of $\mathbf{M}$ there is some l such that

$$\text{for infinitely many } x, \quad \overline{p}(x) \leq \phi^{l^{x+1}}(x).$$

(2) Infer that there is no tuple $f_1, \ldots, f_L$ of total, recursive functions on $\mathbb{N}$ such that every recursive, total unary function on $\mathbb{N}$ is a simple fixed point of $(\mathbb{N}, f_1, \ldots, f_L)$.

*x3A.8 (**Open**). Prove that for every tuple $f_1, \ldots, f_L$ of total, recursive functions on $\mathbb{N}$, there is a recursive total function $f : \mathbb{N}^n \to \{0, 1\}$ which is not a simple fixed point of the expansion $(\mathbf{N}_0, f_1, \ldots, f_L)$.

*x3A.9 (McColm). (1) Show that for each $\mathbf{N}_0$-explicit functional $\alpha(\vec{x}, p)$ with one partial function variable, there is some l such that for all p and k,

$$\text{if for all } \vec{y}, p(\vec{y}) \leq ||\vec{y}|| + k, \text{then for all } \vec{x}, \alpha(\vec{x}, p) \leq ||\vec{x}|| + k + l,$$

(where we assume that $\perp < 0$ to cover the instances of non-convergence).

(2) Prove that if $\overline{p}(\vec{x})$ is a simple, n-ary fixed point of $\mathbf{N}_0$ with infinite domain of convergence, then for some K and infinitely many $\vec{x}$,

$$\overline{p}(\vec{x}) \leq K(||\vec{x}|| + 1)^n.$$

In particular, all unary simple fixed points of $\mathbf{N}_0$ are bounded by $K(x+1)$ for some K and infinitely many x's, and so x^2 is not a simple fixed point of $\mathbf{N}_0$.

McColm has also shown that multiplication $x \cdot y$ is not a simple fixed point of $\mathbf{N}_0$, and he has produced examples of bounded, total, recursive functions which are not simple points of $\mathbf{N}_0$.

3B. Recursive functionals

A **global n-ary** (partial) **function** f on a class $\mathcal{F}$ of τ-algebras is an operation

$$\mathbf{M} \mapsto f_{\mathbf{M}} : M^n \rightharpoonup M_{\mathbb{B}}$$

which assigns to each $\mathbf{M} \in \mathcal{F}$ an n-ary partial function on the universe of $\mathbf{M}$; f is **uniformly recursive** on $\mathcal{F}$, if there is a single program E of $\mathsf{R}(\tau)$ and a function variable ζ of E such that

$$f_{\mathbf{M}}(\vec{x}) = w \iff \mathbf{M}, E \vdash \zeta(\vec{x}) = w \qquad (\mathbf{M} \in \mathcal{F}, \vec{x} \in M^n). \tag{3-35}$$

These are sometimes called **n-ary queries** by computer scientists, at least when $f_M : M^n \to M$. A **Boolean query** is a global, nullary partial function

$$\mathbf{M} \mapsto f_{\mathbf{M}} \in \{\mathtt{tt}, \mathtt{ff}, \perp\}$$

such that for each $\mathbf{M} \in \mathcal{F}$, $f_{\mathbf{M}} \in \{\mathtt{tt}, \mathtt{ff}, \perp\}$, for example the function

$$\mathbf{Conn}_{(H,\to)} = \mathtt{tt} \iff (H, \to) \text{ is a connected graph} \tag{3-36}$$

on the class of all graphs (or just the finite ones).

For each term E of $\mathsf{R}(\tau)$ with free variables in the list $\vec{\mathsf{v}}$, the explicitly defined partial function

$$f_{\mathbf{M}}(\vec{x}) = \mathrm{den}(\mathbf{M}, E, \{\vec{\mathsf{x}} := \vec{x}\})$$

is uniformly recursive on all τ-algebras, because it is defined in all of them by the trivial program

$$\zeta(\vec{\mathsf{v}}) = E.$$

In the same way, the less trivial

$$f_{\mathbf{M}}(x) = g^m(x) \text{ where } m = (\mu n \geq 1)[h^n(x) = \mathtt{tt}]$$

is uniformly recursive in the class of all (g,h)-algebras by Problem x2B.8—the only point being that in solving this problem, we produced a single program E which computes $f_{\mathbf{M}}$ in all (g,h)-algebras. Here we will consider only the following, special but very useful case of this notion.

3B.1. **Definition.** A functional $\alpha(\vec{x}, p_1, \dots, p_m)$ is **M-recursive** if it is uniformly recursive in the class of all expansions $(\mathbf{M}, p_1, \dots, p_m)$ of $\mathbf{M}$ by partial functions of the appropriate arities, i.e., if there is a program E of $\mathsf{R}(\tau, \mathsf{p}_1, \dots, \mathsf{p}_m)$ and a function variable ζ of E such that for all $\vec{x} \in M^n$ and all $p_1, \dots, p_m$,

$$\alpha(\vec{x}, p_1, \dots, p_m) = w \iff (\mathbf{M}, p_1, \dots, p_m), E \vdash \zeta(\vec{x}) = w. \tag{3-37}$$

An obvious move here is to use function variables $\xi_1, \dots, \xi_m$ to name $p_1, \dots, p_m$ in E, so that we can think of E as a program of $\mathsf{R}(\tau)$ in which the function variables $\xi_1, \dots, \xi_m$ are **free**—not defined by the program but given externally, and α is defined by (3-37) with the **function valuation** $\vec{\xi} := \vec{p}$. We let

$$\mathbf{rec}_1(\mathbf{M}) = \text{the class of all } \mathbf{M}\text{-recursive functionals.} \tag{3-38}$$

Since (basically) all the results about the $\mathbf{M}$-recursive partial functions were proved "uniformly", the same proofs establish the basic closure properties of $\mathbf{M}$-recursive functionals, which we list here without additional arguments.

3B.2. **Theorem.** *For each partial algebra* $\mathbf{M}$*:*

(1) *The class* $\mathbf{rec}_1(\mathbf{M})$ *of* $\mathbf{M}$*-recursive functionals includes all* $\mathbf{M}$*-recursive partial functions* $g : M^n \rightharpoonup M_{\mathbb{B}}$ *and all* $\mathbf{M}$*-explicit functionals on* M*.*

(2) $\mathbf{rec}_1(\mathbf{M})$ *is explicitly closed over* $\mathbf{M}$*, and in particular, it is closed under functional composition* (3-5)*, branching* (3-6) *and variable shuffling* (3-7)*.*

(3) $\mathbf{rec}_1(\mathbf{M})$ *is closed under* λ*-substitution* (3-16)*.*

(4) *If* $\mathbf{M}$ *has two distinguished points* $0, 1$*, then a functional* $\alpha(\vec{x}, \vec{p})$ *is* $\mathbf{M}$*-recursive if and only if there is an* $\mathbf{M}$*-explicit functional* $\alpha_1(\vec{t}, \vec{x}, \vec{p}, r)$*, such that*

$$\alpha(\vec{x}, \vec{p}) = \overline{r}_{\vec{p}}(\vec{0}^k, \vec{x}), \tag{3-39}$$

where for each $\vec{p}$*,* $\overline{r}_{\vec{p}}$ *is the least fixed point of the recursive equation*

$$r(\vec{t}, \vec{x}) = \alpha_1(\vec{t}, \vec{x}, \vec{p}, r). \qquad \dashv$$

To help (a little) with the notation, let us write, for any monotone functional β and any partial function q,

$$\{q(\vec{x}) = \beta(\vec{x}, q)\} \iff q \text{ is the least partial function such that } (\forall \vec{x})[q(\vec{x}) = \beta(\vec{x}, q)]; \tag{3-40}$$

now for all q, q',

$$\text{(3-41)}\quad \Big[\{q(\vec{x}) = \beta(\vec{x}, q)\} \;\&\; (\forall \vec{x}, w)[\beta(\vec{x}, q') = w] \Longrightarrow q'(\vec{x}) = w]\Big] \Longrightarrow q \sqsubseteq q',$$

and (4) above can be written in the relatively simple form

$$\text{(3-42)}\qquad \alpha(\vec{x}, \vec{p}) = \overline{r}_{\vec{0},\vec{p}}(\vec{x}) \text{ where } \{\overline{r}_{\vec{p}}(\vec{t}, \vec{x}) = \alpha_1(\vec{t}, \vec{x}, \vec{p}, \overline{r}_{\vec{p}})\}.$$

A functional $\alpha(\vec{x}, \vec{p})$ is a **simple fixed point** of $\mathbf{M}$ if (3-42) holds with no 0's, i.e.,

$$\text{(3-43)}\qquad \alpha(\vec{x}, \vec{p}) = \overline{r}_{\vec{p}}(\vec{x}) \text{ where } \{\overline{r}_{\vec{p}}(\vec{x}) = \alpha_1(\vec{x}, \vec{p}, \overline{r}_{\vec{p}})\}$$

with an $\mathbf{M}$-explicit α_1.

3B.3. **Proposition.** *Every* $\mathbf{M}$*-recursive functional is monotone and continuous.*

PROOF. For monotonicity first, suppose that

$$\alpha(\vec{x}, \vec{p}) = \overline{r}_{\vec{p}}(\vec{x}) \text{ where } \{\overline{r}_{\vec{p}}(\vec{x}) = \alpha_1(\vec{x}, \vec{p}, \overline{r}_{\vec{p}})\}$$

and $\vec{p} \sqsubseteq \vec{q}$. Now

$$\begin{aligned} \alpha_1(\vec{x}, \vec{p}, \overline{r}_{\vec{q}}) = w &\Longrightarrow \alpha_1(\vec{x}, \vec{q}, \overline{r}_{\vec{q}}) = w \quad \text{(mon. of } \alpha_1) \\ &\Longrightarrow \overline{r}_{\vec{q}}(\vec{x}) = w \qquad \text{(def. of } \overline{r}_{\vec{q}}), \end{aligned}$$

and so by (3-41), $\overline{r}_{\vec{p}} \sqsubseteq \overline{r}_{\vec{q}}$, which means that

$$\alpha(\vec{x}, \vec{p}) = \overline{r}_{\vec{p}}(\vec{x}) = w \Longrightarrow \alpha(\vec{x}, \vec{q}) = \overline{r}_{\vec{q}}(\vec{x}) = w.$$

Thus the functionals which are simple fixed points of $\mathbf{M}$ are monotone, and so all $\mathbf{M}$-recursive functionals are monotone by (4) of Theorem 3B.2.

For the continuity, with $\alpha(\vec{x}, \vec{p})$ defined as above again, set

$$\beta_n(\vec{x}, \vec{p}) = \overline{r}^n_{\vec{p}}(\vec{x})$$

where $\overline{r}_{\vec{p}} = \bigcup_n \overline{r}^n_{\vec{p}}$ by the construction in Theorem 3A.18. By an easy induction on n (using the continuity of α_1), each β_n is continuous. But,

$$\text{if } \alpha(\vec{x}, \vec{p}) = \overline{r}_{\vec{p}}(\vec{x}) = w, \text{ then } w = \overline{r}^n_{\vec{p}}(\vec{x}) = \beta_n(\vec{x}, \vec{p}) = w$$

for some n, and so $\beta_n(\vec{x}, \vec{p}') = w$ for some finite $\vec{p}' \sqsubseteq \vec{p}$, which implies that $\alpha(\vec{x}, \vec{p}') = w$. ⊣

Recursive functionals have another interesting property:

3B.4. **Definition.** A monotone functional $\alpha(\vec{x}, \vec{p})$ on M is **deterministic** if for all $\vec{x}, \vec{p}$ and $w \in M$,

$$\text{if } \alpha(\vec{x}, \vec{p}) = w, \text{ then there is a } \sqsubseteq\text{-least } \vec{q} \sqsubseteq \vec{p} \text{ such that } \alpha(\vec{x}, \vec{q}) = w.$$

Perhaps the simplest example of a non-deterministic (monotone and continuous) functional is (full, symmetric) **disjunction**

$$p \vee q = \begin{cases} \mathrm{tt}, & \text{if } p = \mathrm{tt} \text{ or } q = \mathrm{tt}, \\ \bot, & \text{otherwise,} \end{cases} \tag{3-44}$$

where p and q range over nullary partial functions; this is because $\mathrm{tt} \vee \mathrm{tt} = \mathrm{tt}$, but there is no least pair (p, q) of nullary partial functions below $(\mathrm{tt}, \mathrm{tt})$ such that $p \vee q = \mathrm{tt}$, since the only pairs below $(\mathrm{tt}, \mathrm{tt})$ on which $\vee$ converges are $(\mathrm{tt}, \mathrm{ff})$ and $(\mathrm{ff}, \mathrm{tt})$, and these are $\sqsubseteq$-incomparable. Additional examples include the existential quantifier $E^{\#}_M(p)$ defined in (3-13), and the **search** functional (or **half existential quantifier**)

$$E^{\frac{1}{2}}_M(p) = \begin{cases} \mathrm{tt}, & \text{if } (\exists t \in M)[p(t) = \mathrm{tt}], \\ \bot, & \text{otherwise;} \end{cases} \tag{3-45}$$

these are non-deterministic (if M has at least two points) by the same argument.

3B.5. **Proposition.** *For each partial algebra* $\mathbf{M}$*, every* $\mathbf{M}$*-recursive functional is deterministic.*

PROOF. Every partial function $f : M^n \rightharpoonup M_{\mathbb{B}}$ is (trivially) deterministic, and the evaluation functionals also are, since if $p(\vec{x}) = w$, then $q(\vec{x}) = w$ with $q = p \restriction \{\vec{x}\}$.

If α is defined by substitution (3-5) from deterministic functionals, then it is also deterministic: because if

$$\alpha(\vec{x}, \vec{y}, \vec{p}, \vec{q}) = \beta(\gamma(\vec{x}, \vec{p}), \vec{y}, \vec{q}) = w$$

with deterministic β, γ, then there is some u such that

$$\gamma(\vec{x}, \vec{p}) = u \text{ and } \beta(u, \vec{y}, \vec{q}) = w;$$

so by the hypothesis, there exist least $\vec{p}_1 \sqsubseteq \vec{p}$, $\vec{q}_1 \sqsubseteq \vec{q}$ such that $\gamma(\vec{x}, \vec{p}_1) = u$, $\beta(u, \vec{y}, \vec{q}_1) = w$; and then (easily), $\vec{p}_1, \vec{q}_1$ is the $\sqsubseteq$-least tuple of partial functions below $\vec{p}, \vec{q}$ such that $\alpha(\vec{x}, \vec{y}, \vec{p}_1, \vec{q}_1)\downarrow$.

Similar, simple arguments show that the class of deterministic functionals is closed under branching and variable shuffling, so that *every* $\mathbf{M}$*-explicit functional is deterministic.*

Suppose now that $\alpha(\vec{x}, \vec{p})$ is a simple fixed point of $\mathbf{M}$, so that

$$\alpha(\vec{x}, \vec{p}) = \overline{r}_{\vec{p}}(\vec{x}) \text{ where } \{\overline{r}_{\vec{p}}(\vec{x}) = \alpha_1(\vec{x}, \vec{p}, r)\}$$

with α_1 explicit, and hence deterministic. If $\alpha(\vec{x}, \vec{p}) = w$, then

$$\overline{r}_{\vec{p}}(\vec{x}) = \alpha_1(\vec{x}, \vec{p}, \overline{r}_{\vec{p}}) = w,$$

and so there exist a $\sqsubseteq$-least pair $(\vec{q}, r^*)$ such that

$$\vec{q} \sqsubseteq \vec{p}, \quad r^* \sqsubseteq \overline{r}_{\vec{p}} \text{ and } \alpha_1(\vec{x}, \vec{q}, r^*) = w.$$

We claim that $\vec{q}$ is least below $\vec{p}$ such that $\alpha(\vec{x},\vec{q}) = w$. To see this, notice first that

$$\vec{q}' \sqsubseteq \vec{p} \Longrightarrow \overline{r}_{\vec{q}'} \sqsubseteq \overline{r}_{\vec{p}},$$

by the monotonicity of α:

$$\overline{r}_{\vec{q}'}(\vec{x}) = w \Longrightarrow \alpha(\vec{x},\vec{q}') = w \Longrightarrow \alpha(\vec{x},\vec{p}) = w \Longrightarrow \overline{r}_{\vec{p}}(\vec{x}) = w.$$

Thus, for any $\vec{q}'$,

$$\alpha(\vec{x},\vec{q}') = w \Longrightarrow \overline{r}_{\vec{q}'}(\vec{x}) = w \Longrightarrow \alpha_1(\vec{x},\vec{q}',\overline{r}_{\vec{q}'}) = w$$

and the minimality of $(\vec{q}, r^*)$ implies that $\vec{q} \sqsubseteq \vec{q}'$.

Finally, every **M**-recursive functional is deterministic by (3-39). ⊣

3B.6. **Corollary.** *For any* **M**, *the non-deterministic functional* $p \vee q$ *defined in* (3-44) *and the half-quantifier* (3-45) *are not* **M**-*recursive.*

3B.7. **Theorem (The First Recursion Theorem).** *If* $\alpha(\vec{x},p)$ *is* **M**-*recursive, and operative, then its least-fixed-point* $\overline{p}$ *is* **M**-*recursive.*

PROOF. By (4) of Theorem 3B.2,

$$\alpha(\vec{x},p) = \overline{r}_p(\vec{0},\vec{x}) \text{ where } \{\overline{r}_p(\vec{t},\vec{x}) = \alpha_1(\vec{t},\vec{x},p,\overline{r}_p)\} \tag{3-46}$$

with an **M**-explicit $\alpha_1(\vec{t},\vec{x},p,r)$, and, in this notation,

$$\{\overline{p}(\vec{x}) = \alpha(\vec{x},\overline{p})\}, \tag{3-47}$$

where $\overline{p}$ exists because α is monotone and continuous. Consider the system of recursive equations

$$\begin{aligned} r(\vec{t},\vec{x}) &= \alpha_1(\vec{t},\vec{x},p,r), \\ p(\vec{x}) &= r(\vec{0},\vec{x}) \end{aligned}$$

with (mutual) least fixed points $\tilde{r},\tilde{p}$. In the obvious extension of (3-40) to systems of two equations, we can express this simply by

$$\left\{ \begin{aligned} \tilde{r}(\vec{t},\vec{x}) &= \alpha_1(\vec{t},\vec{x},\tilde{p},\tilde{r}) \\ \tilde{p}(\vec{x}) &= \tilde{r}(\vec{0},\vec{x}) \end{aligned} \right\}. \tag{3-48}$$

and the indicated mutual least fixed points are **M** recursive.

(1) $\overline{r}_{\tilde{p}} \sqsubseteq \tilde{r}$. This holds because for all $\vec{t},\vec{x}$,

$$\alpha_1(\vec{t},\vec{x},\tilde{p},\tilde{r}) = \tilde{r}(\vec{t},\vec{x})$$

by (3-48), and so $\overline{r}_{\tilde{p}} \sqsubseteq \tilde{r}$ by (3-41) applied to the system in (3-46).

(2) $\overline{p} \sqsubseteq \tilde{p}$. This holds because for any $\vec{x}$,

$$\begin{aligned} \alpha(\vec{x},\tilde{p}) = w &\Longrightarrow \overline{r}_{\tilde{p}}(\vec{0},\vec{x}) = w \\ &\Longrightarrow \tilde{r}(\vec{0},\vec{x}) = w \ \text{(by (1))} \\ &\Longrightarrow \tilde{p}(\vec{x}) = w \quad \text{(by (3-48)} \end{aligned}$$

(3) *The pair* $(\overline{r}_{\overline{p}}, \overline{p})$ *satisfies the system* (3-48). This is because

$$\alpha_1(\vec{t}, \vec{x}, \overline{p}, \overline{r}_{\overline{p}}) = \overline{r}_{\overline{p}}(\vec{t}, \vec{x})$$

by (3-46), and $\overline{r}_{\overline{p}}(\vec{0}, \vec{x}) = \alpha(\vec{x}, \overline{p}) = \overline{p}(\vec{x})$, directly from the definitions. It follows that

$$\tilde{r} \sqsubseteq \overline{r}_{\overline{p}}, \quad \tilde{p} \sqsubseteq \overline{p},$$

which together with (2) gives $\tilde{p} = \overline{p}$, which is **M**-recursive. ⊣

Problems for Section 3B

x3B.1. Prove that (as a Boolean query defined on the class of all finite graphs, (3-36)), connectedness is not recursive.

x3B.2. Prove (3) and (4) of Theorem 3B.2.

x3B.3. Prove that the class of deterministic functionals on M is closed under variable shuffling.

x3B.4. Prove that every system of equations as in (3-4) with **M**-recursive functionals $\alpha_1, \ldots, \alpha_k$ has a $\sqsubseteq$-least tuple of solutions $\overline{p}_1, \ldots, \overline{p}_k$, and these are **M**-recursive.

3C. Computation theories and Kleene master recursions

We will establish in this section a very strong version of the *Enumeration Theorem* of classical recursion for partial algebras **M** such that $\mathbf{N}_0 \hookrightarrow \mathbf{M}$. The proof we will give is based on Kleene's notion of a *master recursion*, which Kleene used in **?**[**?**] to define his *recursive functionals of higher type*, but which (as he noted) also provides a new characterization of the (Turing) computable partial functions on $\mathbb{N}$. It also provides a novel approach to recursion on partial algebras, which is the way that we will present this work here.

3C.1. **Definition.** Suppose M is set and $\mathbf{N}_0 = (\mathbb{N}, 0, S, Pd, =_0)$ is an isomorphic copy of the basic structure of arithmetic with $\mathbb{N} \subseteq M$. A **computation theory** over $(M, \mathbf{N}_0)$ is a pair $\mathbf{K} = (K, |\ |)$ where

$$K : \mathbb{N} \times M^* \rightharpoonup M_{\mathbb{B}}$$

is a partial function with domain a set of non-empty tuples from M with first member in $\mathbb{N}$;

$$|\ | : \text{Domain}(K) \to \text{Ordinals}$$

is a function which assigns an ordinal number $|e, \vec{x}|$ to each tuple such that $K(e, \vec{x})\downarrow$; and Axioms **(KL1)** – **(KL7)** below are satisfied.

We use the traditional Kleene notation

$$\{z\}(\vec{x}) = \{z\}_{\mathbf{K}}(\vec{x}) = K(z, \vec{x})$$

for the basic operation of a computation theory, and we interpret the stage $|z, \vec{x}|$ as "the length" of some (unspecified) computation of $\{z\}(\vec{x})$. These stages will be finite in the constructions of this Chapter, but it will cause no problems to allow infinite ordinals as stages, and we will find use for those later on.

A partial function $f : M^n \rightharpoonup M$ is **K-recursive** if there exists a $z \in \mathbb{N}$ (a **K-code** of f) such that

$$f(\vec{x}) = \{z\}(\vec{x}) \qquad (\vec{x} \in M^n).$$

To simplify the formulation of the axioms, we also set

$$|e, \vec{x}| >^* |m, \vec{y}|$$
$$\iff \text{either } \{e\}(\vec{x})\uparrow \text{ or } [\{e\}(\vec{x})\downarrow \ \&\ \{m\}(\vec{y})\downarrow \ \&\ |e, \vec{x}| > |m, \vec{y}|].$$

(KL1): Basic functions. The identity $\mathrm{id}(t) = t$ $(t \in M)$, the nullary functions $0, \mathrm{tt}, \mathrm{ff}$, and the unary successor and predecessor functions S and Pd (which converge only on $\mathbb{N}$) are all **K**-computable. The (characteristic function of) equality with 0 is also **K**-computable,

$$=_0 (t) = \begin{cases} \mathrm{tt}, & \text{if } t = 0, \\ \mathrm{ff}, & \text{otherwise,} \end{cases} \qquad (t \in M).$$

(KL2): Substitution. There is a ternary primitive recursive function $\mathrm{subst} : \mathbb{N}^3 \to \mathbb{N}$, such that for all $e, m, n \in \mathbb{N}$ and $\vec{x} \in M^n$, $\vec{y} \in M^*$,

$$\{\mathrm{subst}(e, m, n)\}(\vec{x}, \vec{y}) = \{e\}(\{m\}(\vec{x}), \vec{y}),$$
$$|\mathrm{subst}(e, m, n), \vec{x}, \vec{y}| >^* |m, \vec{x}|, \quad |\mathrm{subst}(e, m, n), \vec{x}, \vec{y}| >^* |\{m\}(\vec{x}), \vec{y}|.$$

In particular, the class of **K**-computable partial functions on M is closed under the substitution scheme

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

(KL3): Branching. There is a ternary primitive recursive function $\mathrm{br} : \mathbb{N}^3 \to \mathbb{N}$, such that for all $e, m_1, m_2 \in \mathbb{N}, \vec{x} \in M^*$, if $z = \mathrm{br}(e, m_1, m_2)$, then

$$\{z\}(\vec{x}) = \text{if } \{e\}(\vec{x}) \text{ then } \{m_1\}(\vec{x}) \text{ else } \{m_2\}(\vec{x}),$$
$$\text{if } \{e\}(\vec{x}) = \mathrm{tt}, \text{then } |z, t, \vec{x}| >^* |e, \vec{x}|,\ |z, t, \vec{x}| >^* |m_1, \vec{x}|,$$
$$\text{if } \{e\}(\vec{x}) = w \neq \mathrm{tt}, \text{then } |z, t, \vec{x}| >^* |e, \vec{x}|,\ |z, t, \vec{x}| >^* |m_2, \vec{x}|.$$

In particular, the class of **K**-computable partial functions on M is closed under branching.

(KL4): Primitive recursion. There is a binary primitive recursive function $\mathrm{pr} : \mathbb{N}^2 \to \mathbb{N}$, such that for all $e, m, y \in \mathbb{N}, \vec{x} \in M^*$,

$$\{\mathrm{pr}(e,m)\}(0,\vec{x}) = \{e\}(\vec{x}), \qquad |\mathrm{pr}(e,m),0,\vec{x}| >^* |e,\vec{x}|,$$
$$\{\mathrm{pr}(e,m)\}(y+1,\vec{x}) = \{m\}(\{\mathrm{pr}(e,m)\}(y,\vec{x}),y,\vec{x}),$$
$$|\mathrm{pr}(e,m),y+1,\vec{x}| >^* |\mathrm{pr}(e,m),y,\vec{x}|,$$
$$|\mathrm{pr}(e,m),y+1,\vec{x}| >^* |m,\{\mathrm{pr}(e,m)\}(y,\vec{x}),y,\vec{x}|.$$

In particular, the class of **K**-computable partial functions on M is closed under primitive recursion.

(KL5): Variable shuffling. There is a ternary primitive recursive function $\mathrm{sh} : \mathbb{N}^3 \to \mathbb{N}$, such that for all $e, f, m \in \mathbb{N}$ and $\vec{x} \in M^n$, if $1 \leq (f)_i \leq n$ for $i = 1, \ldots, m$, then

$$\{\mathrm{sh}(e,f,m)\}(x_1,\ldots,x_n) = \{e\}(x_{(f)_1},\ldots,x_{(f)_m})$$
$$|\mathrm{sh}(e,f,m),x_1,\ldots,x_n| >^* |e,x_{(f)_1},\ldots,x_{(f)_m}|.$$

In particular, the class of **K**-computable partial functions is closed under the variable shuffling scheme[9]

$$f(x_1,\ldots,x_n) = g(x_{\pi(1)},\ldots,x_{\pi(m)}) \quad (\pi : \{1,\ldots,m\} \to \{1,\ldots,n\}).$$

The conditions so far on $\{z\}(\vec{x})$ and $|z,\vec{x}|$ are natural, and they can be read almost as definitions: it appears that we have been simply assuming a natural assignment of number codes and a measure of computational complexity to all the partial functions defined from the (trivial) basics by primitive recursion, shuffling, composition and branching. In particular, they imply that every primitive recursive function $f : \mathbb{N}^n \to \mathbb{N}$ is **K**-computable.

The next axiom is a bit surprising, as it appears to "postulate" the Enumeration Theorem:

(KL6): Enumeration. There is a number S9 such that for all $e \in \mathbb{N}$, $\vec{x} \in M^*$,

$$\{\mathrm{S9}\}(e,\vec{x}) = \{e\}(\vec{x}), \quad |\mathrm{S9},e,\vec{x}| >^* |e,\vec{x}|.$$

This is the characteristic scheme in Kleene's approach, and we have given it the traditional name S9, which is what he happened to call it in **?**[**?**].

In the final axiom we also postulate the so-called S^m_n-Theorem, but (for convenience) uniformly in n:

---

[9] The axiom applies to the case $m = 0$ and yields

$$\{\mathrm{sh}(e,f,0)\}(x_1,\ldots,x_n) = \{e\}(), \qquad |\mathrm{sh}(e,f,0),x_1,\ldots,x_n| >^* |e|.$$

See Footnote 8.

(KL7): The S^m-Theorem. For each $m \in \mathbb{N}$, there is a primitive recursive function $S^m : \mathbb{N}^{m+1} \to \mathbb{N}$ such that if $\vec{y} = (y_1, \ldots, y_m) \in \mathbb{N}^m$ and $\vec{x} \in M^*$, then

$$\{S^m(e, \vec{y})\}(\vec{x}) = \{e\}(\vec{y}, \vec{x}) \quad |S^m(e, \vec{y}), \vec{x}| >^* |e, \vec{y}, \vec{x}|.$$

The condition $\vec{y} \in \mathbb{N}^m$ is important here, because $\{z\}(\vec{w})$ is assumed defined only when $z \in \mathbb{N}$.

3C.2. **Exercise.** Every primitive recursive function $f : \mathbb{N}^n \to \mathbb{N}$ is **K**-computable, for any computation theory **K**.

The first, basic result about computation theories provides a powerful tool for defining **K**-computable functions:

3C.3. **The Second Recursion Theorem**. *If* **K** *is a computation theory and*

$$f(e, \vec{x}) = \{\tilde{f}\}(e, \vec{x}) \qquad (e \in M, \vec{x} \in M^n)$$

is **K***-computable with code* $\tilde{f}$*, then there exists some* $\tilde{e} \in \mathbb{N}$ *such that for all* $\vec{x} \in M^n$*,*

$$\{\tilde{e}\}(\vec{x}) = \{\tilde{f}\}(\tilde{e}, \vec{x}) \tag{3-49}$$

$$\text{if } \{\tilde{e}\}(\vec{x})\downarrow, \text{ then } |\tilde{e}, \vec{x}| > |\tilde{f}, \tilde{e}, \vec{x}|. \tag{3-50}$$

PROOF. Following the usual (mystical) proof of the Second Recursion Theorem in ordinary recursion theory, we set

$$g(m, \vec{x}) = f(S^1(m, m), \vec{x}) \qquad (m \in \mathbb{N}, \vec{x} \in M^n),$$

we observe that this is **K**-computable, and we choose a code $\tilde{g}$ of it; it follows that for all $m, \vec{x}$,

$$f(S^1(m, m), \vec{x}) = \{\tilde{g}\}(m, \vec{x}) = \{S^1(\tilde{g}, m)\}(\vec{x}),$$

and if we set $m := \tilde{g}$ and $\tilde{e} = S^1(\tilde{g}, \tilde{g})$ in this equation, we get the required

$$\{\tilde{e}\}(\vec{x}) = f(\tilde{e}, \vec{x}).$$

To derive the crucial stage-increasing property (3-50), we need to compute the code $\tilde{g}$ using the axioms for **K**, so that we can appeal to their stage-increasing assumptions. So choose first some $a \in \mathbb{N}$ such that

$$\{a\}(m) = S^1(m, m),$$

and infer from **(KL2)** that if $\tilde{g} = \text{subst}(\tilde{f}, a, 1)$, then

$$\{\tilde{g}\}(m, \vec{x}) = \{\tilde{f}\}(S^1(m, m), \vec{x}), \quad |\tilde{g}, m, \vec{x}| >^* |\tilde{f}, S^1(m, m), \vec{x}|.$$

By **(KL7)** now,

$$\{S^1(\tilde{g}, m)\}(\vec{x}) = \{\tilde{g}\}(m, \vec{x}), \quad |S^1(\tilde{g}, m), \vec{x}| >^* |\tilde{g}, m, \vec{x}|,$$

which gives us the required stage-increasing property when we apply this to $m = \tilde{g}$ and set $\tilde{e} := S^1(\tilde{g}, \tilde{g})$. ⊣

3C.4. **Corollary (Minimalization).** *If $g(t, \vec{x})$ is* **K***-computable and*

$$f(\vec{x}) = (\mu t \in \mathbb{N})[g(t, \vec{x}) = 0],$$

then $f(\vec{x})$ is also **K***-computable.*

PROOF. By the closure properties of the **K**-computable partial functions and Theorem 3C.3, there is a **K**-computable partial function $h(t, \vec{x})$ such that

$$h(t, \vec{x}) = \text{if } (g(t, \vec{x}) = 0) \text{ then } 0 \text{ else } S(h(S(t), \vec{x})), \quad (t \in \mathbb{N}, \vec{x} \in M^n).$$

It is enough to show that

$$h(t, \vec{x}) = (\mu s \in \mathbb{N})[g(t + s, \vec{x}) = 0], \tag{3-51}$$

from which the Corollary follows by setting

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

Note that since the successor function $S(t)$ converges only if $t \in \mathbb{N}$,

$$\text{if } h(t, \vec{x})\downarrow, \text{ then } h(t, \vec{x}) \in \mathbb{N}.$$

We will use this in the verification of (3-51), for which we consider three cases.

Case 1. *For all $s \in \mathbb{N}$, $g(t + s, \vec{x})\downarrow$ & $g(t + s, \vec{x}) \neq 0$.* In this case, the defining equation of $h(t, \vec{x})$ gives

$$h(t, \vec{x}) = s + h(t + s, \vec{x}) \qquad (s \in \mathbb{N}),$$

and this is impossible, unless $h(t, \vec{x})\uparrow$.

Case 2. *There exists some $s \in \mathbb{N}$ such that*

$$(\forall i < s)[g(t + i, \vec{x})\downarrow \ \& \ g(t + i, \vec{x}) \neq 0] \ \& \ g(t + s, \vec{x})\uparrow.$$

Now the right-hand-side of (3-51) diverges, and also

$$\begin{aligned} h(t, \vec{x}) &= 1 + h(t + 1, \vec{x}) = \cdots = s + h(t + s, \vec{x}) \\ &= s + \text{if } (g(t + s, \vec{x}) = 0) \text{ then } 0 \text{ else } S(h(S(t + s), \vec{x})) = \perp. \end{aligned}$$

Case 3. *There exists some $s \in \mathbb{N}$ such that*

$$(\forall i < s)[g(t + i, \vec{x})\downarrow \ \& \ g(t + i, \vec{x}) \neq 0] \ \& \ g(t + s, \vec{x}) = 0.$$

The same computation as in *Case* 2 now gives

$$\begin{aligned} h(t, \vec{x}) &= 1 + h(t + 1, \vec{x}) = \cdots = s + h(t + s, \vec{x}) \\ &= s + \text{if } (g(t + s, \vec{x}) = 0) \text{ then } 0 \text{ else } S(h(S(t + s), \vec{x})) = s, \end{aligned}$$

as required. ⊣

3C.5. **Corollary.** *Every recursive partial function $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ is* **K***-computable, for any computation theory* **K**.

Proof. This follows immediately from the identification of (general) recursion with μ-recursion on $\mathbb{N}$. ⊣

The next result is one of those whose proof is more important than its statement—i.e., we will later use and establish additional properties of the specific computation theory constructed in it.

3C.6. **Theorem (The Master Recursion).** *Suppose* $\mathbf{M} = (f_1, \dots, f_L)$ *is a partial algebra and* $\mathbf{N}_0 = (\mathbb{N}, 0, S, Pd, =_0)$ *is a copy of the natural numbers with* $\mathbb{N} \subseteq M$.

There is a monotone and continuous functional

$$\Phi : \mathbb{N} \times M^* \times (\mathbb{N} \times M^* \rightharpoonup M) \rightharpoonup M_{\mathbb{B}}, \tag{3-52}$$

such that if $\overline{p}$ *is the least solution of the recursive equation*

$$p(z, \vec{x}) = \Phi(z, \vec{x}, p) \quad (z \in \mathbb{N}, \vec{x} \in M^*)$$

and we set

$$K(z, \vec{x}) = \{z\}(\vec{x}) = \overline{p}(z, \vec{x}), \tag{3-53}$$

$$|z, \vec{x}| = |z, \vec{x}|_\Phi, \quad (\{z\}(\vec{x})\downarrow), \tag{3-54}$$

then the pair

$$\mathbf{K}[\mathbf{M}, \mathbf{N}_0] = (K, |\ |) \tag{3-55}$$

is a computation theory over $(M, \mathbf{N}_0)$ *such that* $f_1, \dots, f_L$ *are* **K***-computable.*

Proof. We let $\vec{w} = (w_1, \dots, w_l)$ vary over sequences from M of (arbitrary) length $l = l(\vec{w})$ and set

$$\Phi(z, \vec{w}, p) = \text{if } (z \in \mathbb{N}) \text{ then } \Psi(z, \vec{w}, p) \text{ else } \perp,$$

so that $\Phi(z, \vec{w}, p)\uparrow$ unless $z \in \mathbb{N}$. The functional $\Psi(z, \vec{w}, p)$ is defined by seven cases which correspond to the axioms **(KL1)** – **(KL7)** coded in the first component $(z)_0$ of z, i.e.,

$$\begin{aligned}
\Psi(z, \vec{w}, p) = \text{if } ((z)_0 = 1) \text{ then } & \Phi_1(z, \vec{w}, p) \\
\text{else if } ((z)_0 = 2) \text{ then } & \Phi_2(z, \vec{w}, p) \\
\vdots & \\
\text{else if } ((z)_0 = 7) \text{ then } & \Phi_7(z, \vec{w}, p) \\
\text{else } & \perp.
\end{aligned}$$

Finally, the functionals $\Phi_i(z, \vec{w}, p)$ are also defined by cases on the form of the code z and the sequence $\vec{w}$ (especially its length $l = l(\vec{w})$), so that, in the end, the fixed point $\overline{p}(z, \vec{w})$ and the stage assignment $|\ |_\Phi$ satisfy the axioms of a computation theory.

In the construction to follow, we will depend heavily on the classical, primitive recursive coding of tuples of natural numbers, using the following (standard) notations, where p_i is the i'th prime number and the values of the indicated primitive recursive functions on arguments other than those indicated are irrelevant:

$$\begin{aligned}\langle x_0,\dots,x_{n-1}\rangle &= 2^{x_0+1}\cdots p_{n-1}^{x_{n-1}+1}, \quad (\text{with } \langle\,\rangle = 1),\\ (\langle x_0,\dots,x_{n-1}\rangle)_i &= x_i, \quad (i=0,\dots,n-1),\\ \mathrm{lh}(\langle x_0,\dots,x_{n-1}\rangle) &= n,\\ \mathrm{last}(\langle x_0,\dots,x_{n-1}\rangle) &= x_{n-1}.\end{aligned}$$

The relation

$$\mathrm{Seq}(u) \iff (\exists x_0,\dots,x_{n-1})[u = \langle x_0,\dots,x_{n-1}\rangle]$$

is also primitive recursive.

We will incorporate the main steps of the proof of the theorem in the definition of Φ.

Case **(KL1)**, the basic functions and the givens $f_1,\dots,f_L$. We set:

$$\Phi_1(z,\vec{w},p) = \begin{cases} w_1, & \text{if } (z)_1 = 0 \;\&\; l = 1,\\ 0, & \text{if } (z)_1 = 1 \;\&\; l = 0,\\ \mathrm{tt}, & \text{if } (z)_1 = 2 \;\&\; l = 0,\\ \mathrm{ff}, & \text{if } (z)_1 = 3 \;\&\; l = 0,\\ S(w_1), & \text{if } (z)_1 = 4 \;\&\; l = 1 \;\&\; w_1 \in \mathbb{N},\\ Pd(w_1), & \text{if } (z)_1 = 5 \;\&\; l = 1 \;\&\; w_1 \in \mathbb{N},\\ =_0(w_1), & \text{if } (z)_1 = 6 \;\&\; l = 1,\\ f_1(w_1,\dots,w_{n_1}) & \text{if } (z)_1 = 7 \;\&\; (z)_2 = 1 \;\&\; l = n_1,\\ \vdots & \\ f_L(w_1,\dots,w_{n_L}) & \text{if } (z)_1 = 7 \;\&\; (z)_2 = L \;\&\; l = n_L,\\ \perp, & \text{otherwise.}\end{cases}$$

It is clear from this clause that the partial function $\{z\}(\vec{x})$ defined by (3-53), will satisfy **(KL1)**, and all the givens will be **K**-computable, e.g.,

$$\{\langle 1,0\rangle\}(t) = t, \quad \{\langle 1,7,i\rangle\}(x_1,\dots,x_{n_i}) = f_i(x_1\dots,x_{n_i}).$$

Notice also that if $(z)_0 = 1$ and $\{z\}(\vec{x})\downarrow$, then $|z,\vec{x}| = |z,\vec{x}|_\Phi = 0$.

Case **(KL2)**, substitution. If $(z)_0 = 2$, we let $n = (z)_3$ and we set

$$\Phi_2(z,\vec{w},p) = \begin{cases} p((z)_1, p((z)_2, w_1,\dots,w_n), w_{n+1},\dots,w_l), & \text{if } n \le l,\\ \perp, & \text{otherwise,}\end{cases}$$

$$\mathrm{subst}(e,m,n) = \langle 2,e,m,n\rangle.$$

Now definitions (3-53), (3-54) give

$$\{\text{subst}(e,m,n)\}(\vec{x},\vec{y}) = \overline{p}(e,\overline{p}(m,\vec{x}),\vec{y}) = \{e\}(\{m\}(\vec{x}),\vec{y})$$

with $\vec{x} = (x_1,\dots,x_n)$, and

$$|\text{subst}(e,m,n),\vec{x},\vec{y}|_\Phi = \max\{|m,\vec{x}|_\Phi, |e,\{m\}(\vec{x}),\vec{y}|_\Phi\} + 1,$$

so that the required stage-increasing property also holds.

Case **(KL3)**, branching. If $(z)_0 = 3$, we set

$$\Phi_3(z,\vec{w},p) = \text{if } p((z)_1,\vec{w}) \text{ then } p((z)_2,\vec{w}) \text{ else } p((z)_3,\vec{w}),$$
$$\text{br}(e,m_1,m_2) = \langle 3,e,m_1,m_2\rangle.$$

The argument that **(KL3)** holds with these definitions is similar to that in *Case* **(KL2)**.

Case **(KL4)**, primitive recursion. If $(z)_0 = 4$, we set

$$\Phi_4(z,\vec{w},p) = \begin{cases} p((z)_1,w_2,\dots,w_l), & \text{if } l>0 \ \&\ w_1 = 0,\\ p((z)_2,p(z,Pd(w_1),w_2,\dots,w_l),w_2,\dots,w_l), & \\ & \text{if } l>0 \ \&\ w_1>0,\\ \perp, & \text{otherwise.}\end{cases}$$

$$\text{pr}(e,m) = \langle 4,e,m\rangle.$$

Case **(KL5)**, variable shuffling. If $(z)_0 = 5$, we let

$$e = (z)_1, \quad f = (z)_2, \quad m = (z)_3,$$

and we set

$$\Phi_5(z,\vec{w},p) = \begin{cases} p(e,w_{(f)_1},\dots,w_{(f)_m}), & \text{if } 1\le(f)_i\le l \text{ for } i=1,\dots,m,\\ \perp, & \text{otherwise.}\end{cases}$$

$$\text{sh}(e,f,m) = \langle 5,e,f,m\rangle.$$

Case **(KL6)**, enumeration. If $(z)_0 = 6$, we simply set

$$\Phi_6(z,\vec{w},p) = \begin{cases} p(\vec{w}) & \text{if } z = \langle 6\rangle \ \&\ l>0,\\ \perp, & \text{otherwise.}\end{cases}$$
$$\text{S9} = \langle 6\rangle.$$

With the definitions (3-53), (3-54), we now have

$$\{\text{S9}\}(e,\vec{x}) = \overline{p}(\langle 6\rangle,e,\vec{x}) = \Phi_6(\langle 6\rangle,e,\vec{x},\overline{p}) = \overline{p}(e,\vec{x}) = \{e\}(\vec{x}),$$

and $|\text{S9},e,\vec{x}|_\Phi = |e,\vec{x}|_\Phi + 1$, which implies the required stage-increasing property.

Case **(KL7)**, the S^m-theorem. If $(z)_0 = 7$, we let $m = (z)_3$ and we set

$$\Phi_7(z,\vec{w},p) = p((z)_1,(z)_{2,1},\dots,(z)_{2,m},\vec{w}),$$
$$S^m(e,y_1,\dots,y_m) = \langle 7,e,\langle y_1,\dots,y_m\rangle,m\rangle.$$

It follows that

$$\begin{aligned}\{S^m(e,y_1,\dots,y_m)\}(\vec{x}) &= \overline{p}(\langle 7,e,\langle y_1,\dots,y_m\rangle,m\rangle,\vec{x})\\ &= \Phi_7(\langle 7,e,\langle y_1,\dots,y_m\rangle,m\rangle,\vec{x},\overline{p}) = \overline{p}(e,y_1,\dots,y_m,\vec{x})\\ &= \{e\}(y_1,\dots,y_m,\vec{x}),\end{aligned}$$

as required. And for the stages, clearly,

$$|S^m(e,y_1,\dots,y_m),\vec{x}|_\Phi = |e,y_1,\dots,y_m,\vec{x}|_\Phi + 1. \qquad \dashv$$

This canonical computation theory $\mathbf{K}[\mathbf{M},\mathbf{N}_0]$ constructed from $\mathbf{M}$ (and $\mathbf{N}_0$) is called the (Kleene) **master recursion** over $(\mathbf{M},\mathbf{N}_0)$.

Next we show that if $\mathbf{N}_0 \hookrightarrow \mathbf{M}$, then $\mathbf{M}$-recursion coincides with $\mathbf{K}[\mathbf{M},\mathbf{N}_0]$-computability. Key to the proof of one direction is the following, carefully formulated notion.

3C.7. **Definition.** Suppose $\mathbf{K}$ is a computation theory over $(\mathbf{M},\mathbb{N})$. A monotone functional $\alpha(\vec{x},p,q)$ on M with p unary and q binary is **K-effective**, if the partial function

$$f(e,m,\vec{x},\vec{y},\vec{z}) = \alpha(\vec{x},\lambda(s)\{e\}(s,\vec{y}),\lambda(u,v)\{m\}(u,v,\vec{z}))$$

is $\mathbf{K}$-computable; it is **strongly K-effective** if, in addition, there exists a code $\tilde{f}$ of f such that for all $e,m,\vec{x},\vec{y},\vec{z}$,

$$\begin{aligned}(3\text{-}56)\quad &\alpha(\vec{x},\lambda(s)\{e\}(s,\vec{y}),\lambda(u,v)\{m\}(u,v,\vec{z})) = w\\ \Longrightarrow\ &\{\tilde{f}\}(e,m,\vec{x},\vec{y},\vec{z}) = w\\ &\ \&\ (\exists p,q)[p\sqsubseteq\lambda(s)\{e\}(s,\vec{y})\ \&\ q\sqsubseteq\lambda(u,v)\{m\}(u,v,\vec{z})\\ &\qquad\ \&\ (\forall s)[p(s)\!\downarrow\Longrightarrow|e,s,\vec{y}|<|\tilde{f},e,m,\vec{x},\vec{y},\vec{z}|]\\ &\qquad\ \&\ (\forall u,v)[q(u,v)\!\downarrow\Longrightarrow|m,u,v,\vec{z}|<|\tilde{f},e,m,\vec{x},\vec{y},\vec{z}|].\end{aligned}$$

These notions have obvious extensions to functionals with any number of partial function arguments, of any arity.

Notice that the condition (3-56) depends only on the values of $\alpha(\vec{x},p,q)$ on partial functions p and q which are $\mathbf{K}$-computable *from parameters in* M, namely

$$p(s) = \{e\}(s,\vec{y}),\quad q(u,v) = \{m\}(u,v,\vec{z})$$

in the specific example treated, where $\vec{y},\vec{z}$ are arbitrary tuples of members of M. It is important here that the values of α on arbitrary partial functions are irrelevant, but also that we need to allow parameters from M in the partial functions we must consider; this is, of course, because (in general), not every constant in M is $\mathbf{K}$-computable.

3C.8. **Lemma.** *If* $\mathbf{M}$ *is a partial algebra and* $\mathbf{N}_0 \hookrightarrow \mathbf{M}$*, then every* $\mathbf{M}$*-explicit functional is strongly* $\mathbf{K}$*-effective, for every computation theory* $\mathbf{K}$ *over* $(\mathbf{M}, \mathbb{N})$.

Proof. It suffices to show that the set of strongly effective functionals is explicitly closed over $\mathbf{M}$, and this is very easy: the stage-increasing properties of the master recursion have been postulated precisely for this reason. ⊣

3C.9. **Lemma.** *If* $\mathbf{N}_0 \hookrightarrow \mathbf{M}$*, then every* $\mathbf{M}$*-recursive partial function is* $\mathbf{K}$*-computable, for every computation theory* $\mathbf{K}$ *over* $(\mathbf{M}, \mathbb{N})$.

Proof. It is enough to show that every simple fixed point of $\mathbf{M}$ is $\mathbf{K}$-computable, since the class of partial functions which are $\mathbf{K}$-computable is closed under composition with the constant 0.

Suppose then that $\overline{p}$ is the least solution of the equation

$$p(\vec{x}) = \alpha(\vec{x}, p)$$

with $\mathbf{M}$-explicit α, and choose $\tilde{f}$ such that $\alpha(\vec{x}, \{e\}) = \{\tilde{f}\}(e, \vec{x})$ and

$$\begin{aligned}\alpha(\vec{x}, \{e\}) = w \Longrightarrow \{\tilde{f}\}(e, \vec{x}) = w \\ \& \ (\exists q)[\alpha(\vec{x}, q) = w \ \& \ (\forall \vec{t})[q(\vec{t})\downarrow \Longrightarrow |e, \vec{x}| < |\tilde{f}, e, \vec{x}|]].\end{aligned}$$

By the Second Recursion Theorem 3C.3, choose $\tilde{e}$ such that

$$(\forall \vec{x})[\{\tilde{e}\}(\vec{x}) = \{\tilde{f}\}(\tilde{e}, \vec{x})] \ \& \ (\forall \vec{x})[\{\tilde{e}\}(\vec{x})\downarrow \Longrightarrow |\tilde{f}, \tilde{e}, \vec{x}| < |\tilde{e}, \vec{x}|].$$

The first of these facts implies that, for all $\vec{x}$,

$$\alpha(\vec{x}, \{\tilde{e}\}) = \{\tilde{f}\}(\tilde{e}, \vec{x}) = \{\tilde{e}\}(\vec{x}),$$

so that $\{\tilde{e}\}$ is a fixed point of α and hence $\overline{p} \sqsubseteq \{\tilde{e}\}$. To show that $\overline{p} = \{\tilde{e}\}$ (and hence $\mathbf{K}$-computable), we show by (complete) induction on k, that

$$\text{if } \{\tilde{e}\}(\vec{x}) = w \ \& \ |\tilde{e}, \vec{x}| = k, \text{ then } \overline{p}(\vec{x}) = w.$$

Computing:

$$\begin{aligned}\{\tilde{e}\}(\vec{x}) = w &\Longrightarrow \{\tilde{f}\}(\tilde{e}, \vec{x}) = w \ \& \ |\tilde{e}, \vec{x}| > |\tilde{f}, \tilde{e}, \vec{x}| \\ &\Longrightarrow (\exists q \sqsubseteq \{\tilde{e}\})[\alpha(\vec{x}, q) = w \ \& \ (\forall \vec{s})[q(\vec{s})\downarrow \Longrightarrow |\tilde{f}, \tilde{e}, \vec{x}| > |\tilde{e}, \vec{s}|]] \\ &\Longrightarrow (\exists q \sqsubseteq \{\tilde{e}\})[\alpha(\vec{x}, q) = w \ \& \ (\forall \vec{s})[q(\vec{s})\downarrow \Longrightarrow |\tilde{e}, \vec{s}| < |\tilde{e}, \vec{x}|]] \\ &\Longrightarrow (\exists q \sqsubseteq \{\tilde{e}\})[\alpha(\vec{x}, q) = w \ \& \ (\forall \vec{s})[q(\vec{s})\downarrow \Longrightarrow \overline{p}(\vec{s}) = q(\vec{s})]] \\ &\Longrightarrow \overline{p}(\vec{x}) = \alpha(\vec{x}, p) = w,\end{aligned}$$

where the induction hypothesis was used in the next-to-the-last inference, and the last one follows by the monotonicity of α ⊣

In particular, every $\mathbf{M}$-recursive partial function is $\mathbf{K}[\mathbf{M}, \mathbf{N}_0]$-computable.

The first idea for proving the converse is to assume a coding of finite sequences

$$c : M^* \rightarrowtail M$$

which is recursive in $\mathbf{M}$, and using it to replace the functional Φ in (3-52) by some

$$\Phi' : \mathbb{N} \times M \times (\mathbb{N} \times M \rightharpoonup M) \rightharpoonup M$$

which will be $\mathbf{M}$-recursive, so that its least fixed point (and hence $\overline{p}_\Phi$) will also be $\mathbf{M}$-recursive. This is easy to do, if we assume an $\mathbf{M}$-recursive coding of M^*, which is not an unreasonable assumption but is not necessary for the result and it does not always hold, cf. Problem *x3C.2. What we do instead is to use the fact that for any $z, \vec{x}$, the computation of $\{z\}(\vec{x})$ in the master recursion takes place in the subset $[\vec{x}]$ of M *generated* from $\mathbb{N} \cup \{\vec{x}\}$ by the givens of $\mathbf{M}$; this set can be effectively coded in $\mathbb{N}$, and so we can use the classical coding of tuples in $\mathbb{N}$ in the argument suggested.

As in Problem x2A.8 (but including $\mathbb{N}$ this time), we let

$$\begin{aligned} [\vec{x}]_0 &= \mathbb{N} \cup \{x_1, \dots, x_n\}, \\ [\vec{x}]_{m+1} &= [\vec{x}]_m \\ &\quad \bigcup_{i=1}^{K} \{f_i(t_1, \dots, t_{n_i}) \mid t_1, \dots t_{n_i} \in [\vec{x}]_m, f_i(t_1, \dots, t_{n_i})\downarrow\}, \\ [\vec{x}] &= \bigcup_m [\vec{x}]_m. \end{aligned} \tag{3-57}$$

3C.10. **Lemma.** *For any $\vec{x} \in M^n$ and $z \in \mathbb{N}, s_1, \dots, s_k, w \in M$,*

$$\text{if } s_1, \dots, s_k \in [\vec{x}] \ \&\ \{z\}(\vec{s}) = w, \text{ then } w \in [\vec{x}] \cup \{\mathrm{tt}, \mathrm{ff}\},$$

where $\{z\}(\vec{s})$ is the basic operation of the master recursion $\mathbf{K}[\mathbf{M}, \mathbf{N}_0]$.

PROOF is by a routine induction on $|z, \vec{s}|_\Phi$. ⊣

For each n, let C_n be the closure in $\mathbb{N}$ of the set

$$\{\underline{\mathrm{x}}_1, \dots, \underline{\mathrm{x}}_n\} \cup \{\underline{\mathrm{j}} \mid j \in \mathbb{N}\}$$

under the L functions

$$\underline{\mathrm{f}}_i(t_1, \dots, t_{n_i}) = \langle i, t_1, \dots, t_{n_i} \rangle, \qquad i = 1, \dots, i = L,$$

where

$$\underline{\mathrm{x}}_i = \langle 1, i \rangle \quad (i = 1, \dots, n) \text{ and } \underline{\mathrm{j}} = \langle 2, j \rangle \quad (j \in \mathbb{N}).$$

It is clear that C_n is a recursively enumerable set of natural numbers, and so $\mathbf{M}$-semirecursive. We use C_n to code the members of $[\vec{x}]$, and the specific values of these codes are irrelevant. What matters is the next simple Lemma, where $\mathrm{d}(t, \vec{x})$ is interpreted as the member of $[\vec{x}]$ coded by t, when we assume that $\underline{\mathrm{x}}_i$ codes x_i for $i = 1, \dots, n$.

3C.11. **Lemma.** *If* $\mathbf{N}_0 \hookrightarrow \mathbf{M}$*, then there is an* $\mathbf{M}$*-recursive partial function* $\mathrm{d}(t, \vec{x})$ *of* $n+1$ *arguments with the following properties:*

(1) $\mathrm{d}(\underline{\mathrm{j}}, \vec{x}) = j, \quad \mathrm{d}(\underline{\mathrm{x}}_i, \vec{x}) = x_i,$

$$\mathrm{d}(\underline{\mathrm{f}}_i(t_1, \dots, t_{n_i}), \vec{x}) = f_i(\mathrm{d}(t_1, \vec{x}), \dots, \mathrm{d}(t_{n_i}, \vec{x})).$$

(2) *For each* $w \in [\vec{x}]$*, there is* (at least) *one* $t \in C_n$ *such that* $\mathrm{d}(t, \vec{x}) = w$.

(3) *If* $\{z\}(\vec{x})$ *is the basic operation of* $\mathbf{K}[\mathbf{M}, \mathbf{N}_0]$*, then the partial function*

$$\varphi(z, u, \vec{x}) = \begin{cases} \{z\}(\mathrm{d}((u)_0, \vec{x}), \dots, \mathrm{d}((u)_{\mathrm{last}(u)}, \vec{x})), & \text{if } \mathrm{Seq}(u), \\ \perp, & \text{otherwise} \end{cases} \tag{3-58}$$

is $\mathbf{M}$*-recursive.*

PROOF. (1) is proved easily by interpreting the conditions it sets on $\mathrm{d}(t, \vec{x})$ as a recursive definition of it in $\mathbf{M}$, and (2) is proved by induction on the definition of $[\vec{x}]$.

The heart of the Lemma is (3), which is proved by identifying φ as the least solution $\overline{p}$ of a recursive equation

$$p(z, u, \vec{x}) = \Phi'(z, u, \vec{x}, p)$$

where the functional $\Phi'(z, u, \vec{x}, p)$ is $\mathbf{M}$-recursive. This functional is a coded version of the basic functional Φ of Theorem 3C.6, and it is defined by copying (and coding) the definition of that Φ in the proof of Theorem 3C.6. We give only the structure of the argument and some of the cases.

First we set

$$\Phi'(z, u, \vec{x}, p) = \text{if } (z \in \mathbb{N} \ \&\ \mathrm{Seq}(u) \ \&\ (\forall i < \mathrm{lh}(u))(u)_i \in C_n) \text{ then } \Psi'(z, u, \vec{x}, p) \text{ else } \perp.$$

Next $\Psi'(z, u, \vec{x}, p)$ is defined by seven cases (and an "otherwise") from functionals $\Phi'_i(z, u, \vec{x}, p)$, $i = 1, \dots, 7$ which are coded versions of the functionals $\Phi_i(z, \vec{w}, p)$. For example, with $l = \mathrm{lh}(u)$:

$$\Phi'_1(z, u, \vec{x}, p) = \begin{cases} \mathrm{d}((u)_0, \vec{x}), & \text{if } (z)_1 = 0 \ \&\ l = 1, \\ \underline{0}, & \text{if } (z)_1 = 1 \ \&\ l = 0, \\ \mathrm{tt}, & \text{if } (z)_1 = 2 \ \&\ l = 0, \\ \mathrm{ff}, & \text{if } (z)_1 = 3 \ \&\ l = 0, \\ S(\mathrm{d}((u)_0, \vec{x}), & \text{if } (z)_1 = 4 \ \&\ l = 1 \ \&\ w_1 \in \mathbb{N}, \\ Pd(\mathrm{d}((u)_0, \vec{x}), & \text{if } (z)_1 = 5 \ \&\ l = 1 \ \&\ w_1 \in \mathbb{N}, \\ =_0 (\mathrm{d}((u)_0, \vec{x}), & \text{if } (z)_1 = 6 \ \&\ l = 1, \\ f_1(\mathrm{d}((u)_0, \vec{x}), \dots, \mathrm{d}((u)_{n_1 - 1}, \vec{x}) & \text{if } (z)_1 = 7 \ \&\ (z)_2 = 1 \ \&\ l = n_1, \\ \vdots & \\ f_L(\mathrm{d}((u)_0, \vec{x}), \dots, \mathrm{d}((u)_{n_L - 1}, \vec{x}) & \text{if } (z)_1 = 7 \ \&\ (z)_2 = 1 \ \&\ l = n_L, \\ \perp, & \text{otherwise.} \end{cases}$$

This clause will insure in the end that if $z = \langle 1, n_i, i\rangle$, then

$$\overline{p}(z, u, \vec{x}) = f_i(\mathrm{d}((u)_0, \vec{x}), \dots, \mathrm{d}((u)_{n_i-1}, \vec{x})),$$

the relevant case of (3-58).

(KL3), Substitution ($z = \langle \mathrm{sub}, \mathrm{lh}(u), e, e_1, \dots, e_m\rangle$):

$$\Phi'_4(z, u, \vec{x}, p) = p((z)_2, p((z)_3, u, \vec{x}), p((z)_4, u, \vec{x}), \dots, p((z)_{\mathrm{lh}(z) \dot{-} 1}, u, \vec{x})).$$

This will insure in the end that for such z,

$$\begin{aligned}\overline{\Phi'}(z, u, \vec{x}) &= \overline{\Phi'}((z)_2, \overline{\Phi'}((z)_3, u, \vec{x}), \overline{\Phi'}((z)_4, u, \vec{x}), \dots, \overline{\Phi'}((z)_{\mathrm{lh}(z) \dot{-} 1}, u, \vec{x})\\ &= \{(z)_2\}(\{(z)_3\}(\mathrm{d}((u)_0, \vec{x}), \dots, \mathrm{d}((u)_{n_1-1}, \vec{x})),\\ &\quad \dots, \{(z)_{\mathrm{lh}(z) \dot{-} 1}\}(\mathrm{d}((u)_0, \vec{x}), \dots, \mathrm{d}((u)_{n_1-1}, \vec{x})))\\ &= \{z\}(\mathrm{d}((u)_0, \vec{x}), \dots, \mathrm{d}((u)_{n_1-1}, \vec{x}))),\end{aligned}$$

the relevant case of (3-58).

The definitions and the arguments are similar in all the cases and they are easy (in principle) to put down. It is important to keep in mind that $\vec{x}$ is here carried along as a parameter, and it does not enter the recursive definition which takes place on the level of the codes; $\vec{x}$ is only used at end of the recursion, so to speak (the basis cases), to apply the givens. ⊣

Problems for Section 3C

x3C.1. Suppose $\mathbf{K}$ is a computation theory and $g(t, \vec{x})$ is $\mathbf{K}$-computable. Prove that there is some $\tilde{h} \in \mathbb{N}$ such that

$$\begin{aligned}\{\tilde{h}\}(t, \vec{x}) = \text{if } g(t, \vec{x}) = 0 \text{ then } t \text{ else } \{\tilde{h}\}(S(t), \vec{x}),\\ \text{if } \{\tilde{h}\}(t, \vec{x})\downarrow \ \&\ \{\tilde{h}\}(t, \vec{x}) \neq 0, \text{ then } |\tilde{h}, t, \vec{x}| > |\tilde{h}, t+1, \vec{x}|.\end{aligned}$$

Show also that for this $\tilde{h}$,

$$\{\tilde{h}\}(t, \vec{x}) = (\mu s \geq t)[g(s, \vec{x}) = 0].$$

x3C.2. Prove Lemma 3C.8.

*x3C.3. Give an example of a total algebra $\mathbf{M}$ which imbeds $\mathbf{N}_0$ but for which there is no $\mathbf{M}$-recursive injection $\pi : M^2 \rightarrowtail M$. HINT: Take $\mathbf{M} \uplus \mathbf{N}_0$ as in (2-37), where $\mathbf{M} = (M, =)$ with an uncountable M. (Or, you may think up a more natural example!)

The next result is about Kleene's approach to abstract computability and does not require the assumption that $\mathbf{M}$ imbeds $\mathbf{N}_0$:

*x3C.4 (Characterization of the master recursion). Let $\mathbf{K} = \mathbf{K}[\mathbf{M}, \mathbf{N}_0]$ be the master recursion over a partial algebra $\mathbf{M}$ (with $\mathbb{N} \subseteq M$) defined in Theorem 3C.6, and suppose $\mathbf{K}'$ is any computation theory over $M, \mathbf{N}_0$ such that $f_1, \dots, f_L$ are $\mathbf{K}'$-recursive. Prove that there is a recursive function $u : \mathbb{N} \to \mathbb{N}$ such that for all $z \in \mathbb{N}$ and $\vec{x} \in M^*$,

$$\{z\}_{\mathbf{K}}(\vec{x}) = \{u(z)\}_{\mathbf{K}'}(\vec{x}), \quad \{z\}_{\mathbf{K}}(\vec{x})\downarrow \Longrightarrow |z, \vec{x}|_{\mathbf{K}} \leq |u(z), \vec{x}|_{\mathbf{K}'}. \tag{3-59}$$

In particular, every $\mathbf{K}$-computable partial function is $\mathbf{K}'$-recursive. Hint: Use the Second Recursion Theorem on (classical) recursion on $\mathbb{N}$.

CHAPTER 4

LEAST FIXED POINTS

In this Chapter we will establish the basic facts about least-fixed-point recursion on *complete partially ordered sets*, the most general, simplest, and most widely applicable part of recursion theory. For easy reference (and to make this Chapter essentially self-contained), we have included in Section 4A a brief review of the basic definitions and facts about partially ordered sets and we have repeated in Section 4B some of the definitions in Chapters 2 and 3.

4A. Posets

4A.1. A **partially ordered set** or **poset** is a pair $(D, \leq)$ of a set[10] D and a binary relation $\leq$ on D which is a *partial ordering*, i.e., for all $x, y \in D$,

$$x \leq x,\ x \leq y \,\&\, y \leq z \Longrightarrow x \leq z,\ x \leq y \,\&\, y \leq x \Longrightarrow x = y.$$

We will typically refer to *a poset* D with *points* $x, y, \ldots \in D$ when the relation $\leq$ is clear from the context, and we will abbreviate

$$x < y \iff_{\text{df}} x \leq y \,\&\, x \neq y.$$

Two points $x, y \in D$ are *comparable* if either $x \leq y$ or $y \leq x$, and *compatible* if $x \leq z, y \leq z$ for some $z \in D$.

Most often we will be studying posets with a least element which we will denote by the same symbol $\bot$ (read "bottom") for all D,

$$\bot = \bot_D =_{\text{df}} \inf\{x \mid x \in D\}.$$

[10] We work in the theory **ZFDC**, which is the classical Zermelo-Fraenkel set theory weakened by omitting the Axiom of Foundation and replacing the full Axiom of Choice by the weaker Axiom of Depended Choices. Results whose proofs appeal to the full Axiom of Choice are marked by the symbol (**AC**), and if we ever need the Axiom of Foundation we will assume it explicitly.

Every subset $E \subseteq D$ of a poset is also a poset with the induced partial ordering,

$$x \leq_E y \iff_{\rm df} x, y \in E \ \& \ x \leq_D y;$$

we call E a *subposet* of D if in addition $\bot_D \in E$, or $\bot_D$ does not exist. This is only a convention, but a useful one.

4A.2. **Mappings**. A mapping (function) $f : D \to E$ from one poset to another is **monotone** if it preserves the ordering,

$$x \leq_D y \Longrightarrow f(x) \leq_E f(y);$$

strict if it preserves $\bot$ (when it exists),

$$f(\bot_D) = \bot_E;$$

and a **poset homomorphism** if it respects the ordering (in both directions),

$$x \leq_D y \iff f(x) \leq_E f(y).$$

A bijective homomorphism is called an **isomorphism** or **similarity** of D with E, and it is automatically strict. We will denote the classes of these mappings by embellishing the standard notation $(D \to E)$ for the most general function space in obvious ways,

$$\mathrm{Mon}(D \to E), \ \mathrm{Strict}(D \to E),$$

etc. These classes of mappings are all closed under composition.

Two posets are **isomorphic** (or similar) if there is an isomorphism from one onto the other, in symbols

$$D \cong E \iff_{\rm df} (\exists \pi : D \rightarrowtail\!\!\!\!\!\twoheadrightarrow E)[\pi \text{ is an isomorphism}].$$

Isomorphic posets have the same order-theoretic properties.

4A.3. Every set A can be viewed as a (trivial) **discrete poset**, partially ordered by the equality relation,

$$x \leq_{d,A} y \iff_{\rm df} x = y \quad (x, y \in A).$$

With each poset D we associate its **bottom lifting** $D_\bot$, where $\bot$ is a new point put below every point of D:

$$\begin{array}{rcl} D_\bot & =_{\rm df} & \{\bot\} \cup D, \\ x \leq_{D_\bot} y & \iff_{\rm df} & x = \bot \vee x \leq_D y. \end{array}$$

If A is a discrete poset—a set—then $A_\bot$ is the **flat poset** associated with A: its members are all the (mutually incomparable) members of A, and $\bot$ put below all of them.

4A.4. Especially useful is the three-member flat Boolean poset

$$\mathbb{B}_\perp =_{\rm df} \{\perp, \mathtt{ff}, \mathtt{tt}\},$$

where $\mathbb{B} = \{\mathtt{ff}, \mathtt{tt}\}$ is the standard Boolean set. It is quite common to identify a relation $R \subseteq A$ with its **characteristic function** $\chi_R : A \to \mathbb{B}$,

$$\chi_R(x) = \begin{cases} \mathtt{tt}, & \text{if } R(x), \\ \mathtt{ff}, & \text{if } \neg R(x). \end{cases}$$

A **partial relation** on a poset D is a monotone function $R : D \to \mathbb{B}_\perp$. Quite often we will give definitions using **conditional expressions** of the form

$$\begin{aligned} f(x) = {} & \text{if } R_1(x) \text{ then } g_1(x) \\ & \text{else if } R_2(x) \text{ then } g_2(x) \\ & \text{else } g_3(x), \end{aligned}$$

where R_1, R_2 are partial relations on D and $g_1, g_2, g_3 : D \to E$ are monotone. This form of definition abbreviates the obvious:

$$f(x) = \begin{cases} g_1(x), & \text{if } R_1(x) = \mathtt{tt}, \\ g_2(x), & \text{if } R_1(x) = \mathtt{ff} \ \&\ R_2(x) = \mathtt{tt}, \\ g_3(x), & \text{if } R_1(x) = R_2(x) = \mathtt{ff}, \\ \perp_E, & \text{otherwise}, \end{cases}$$

so that e.g., $f(x) = \perp$ if $R_1(x) = \perp$. Notice that such definitions yield monotone functions.

4A.5. **Sequences and streams**. The set $\mathrm{Seqs}(A)$ of all (finite and infinite) sequences from a set A is naturally partially ordered by the **initial segment** partial ordering,

$$u \sqsubseteq v \iff |u| \leq |v| \ \&\ (\forall i < |u|)[u_i = v_i],$$

the set of strings (finite sequences) A^* is a subposet of it and $\mathrm{Streams}(A)$ (defined in 1.14) is a subposet of $\mathrm{Seqs}(A \cup \{\mathbf{t}\})$.

4A.6. **Direct products.** Suppose $\{D_i \mid i \in I\}$ is a family of (non-empty) posets indexed by the set I. The (direct) **product** of $\{D_i \mid i \in I\}$ is the Cartesian product

$$\textstyle\prod_{i\in I} D_i =_{\rm df} \{u : I \to \bigcup_{i\in I} D_i \mid (\forall i \in I)[u(i) \in D_i]\}$$

partially ordered **pointwise** by the relation

$$u \leq v \iff_{\rm df} (\forall i \in I)[u_i \leq_i v_i].$$

We often deal with finite families of posets, in which case we can take $I = \{0, \ldots, n-1\}$, we write $D_0 \times D_1 \times \cdots \times D_{n-1}$ for the product, and the functions $u : \{i \in \mathbb{N} \mid i < n\} \to \bigcup_{i<n} D_i$ are just the sequences of length n.

Still more special is the case $I = \emptyset$, which yields the **empty product**

$$\mathbf{I} =_{\mathrm{df}} \prod_{i\in\emptyset} D_i = \{\emptyset\}, \tag{4-1}$$

clearly independent of any D_i. Here $\emptyset$ is the (unique) function with empty domain by the usual set-theoretic conventions. A function $p : \mathbf{I} \to W$ is determined by its single value $p(\emptyset)$, and we will think of such functions as **nullary**—of no arguments—and write synonymously

$$p(\emptyset) = p(\,) = p \quad (p : \mathbf{I} \to W). \tag{4-2}$$

In accordance with this convention, we will also write

$$\lambda(\,)w =_{\mathrm{df}} \lambda(i \in \mathbf{I})w : \mathbf{I} \to W \quad (w \in W).$$

With each function

$$f : A \to \prod_{i\in I} D_i$$

into a product poset, we associate its **component functions**

$$f_i : A \to D_i, \quad f_i(x) = f(x)_i,$$

one for each index $i \in I$. We can recover the function f from these by the λ-operator,

$$f(x) = \lambda(i \in I)f_i(x).$$

This is only notation, but very useful notation.

4A.7. **Exercise.** Show that if D and each E_i are posets, then a mapping $f : D \to \prod_{i\in I} E_i$ is monotone if and only if each component function $f_i : D \to E_i$ is monotone.

4A.8. **Exercise.** Show that if $\mathbf{I}$ is the singleton poset (4-1), then $\mathbf{I} \times D$ and $D \times \mathbf{I}$ are both isomorphic with D.

Sometimes we formulate results about arbitrary products $D \times E$, with the simpler case about D following by taking $E = \mathbf{I}$.

4A.9. **Function posets.** Another important special case of the direct product is when all the D_i are the same, in which case the product is the function poset

$$\prod_{i\in I} D = (I \to D) = \{u \mid u : I \to D\}.$$

If I is itself a poset, then the function posets $\mathrm{Mon}(I \to D)$ and $\mathrm{Strict}(I \to D)$ of monotone and strict mappings are subposets of $(I \to D)$.

4A.10. **Exercise.** Show that for all posets D, E and W,

$$(\mathbf{I} \to D) \cong D,$$
$$(D \to (E \to W)) \cong (D \times E \to W).$$

4A.11. **Partial functions.** Still more special is the case $D = B_\perp$ of the flat poset over the set B, when

$$(I \to B_\perp) = (I \rightharpoonup B)$$

is the poset of partial functions already defined in 1.6. It is easy to verify that the partial ordering on $(I \rightharpoonup B)$ defined in 1.6 is exactly the pointwise partial ordering on $(I \to B_\perp)$.

4A.12. The **disjoint sum** of an indexed family of posets $\{D_i \mid i \in I\}$ is the set

$$\oplus_{i\in I}\ D_i =_{\rm df} \{(i,d) \mid i \in I, d \in D_i\}$$

partially ordered by the relation

$$(i,d) \leq (j,e) \iff_{\rm df} i = j\ \&\ d \leq_i e;$$

and the **coalesced sum** of an indexed family of posets $\{D_i \mid i \in I\}$ is the poset

$$\textstyle\sum_{i\in I} D_i =_{\rm df} (\oplus_{i\in I}\ (D_i \setminus \{\perp_i\})_\perp = \{\perp\} \bigcup \oplus_{i\in I}\ (D_i \setminus \{\perp_i\}),$$

where each $\perp_i$ is the least element of D_i. We use infix notation for the disjoint or coalesced sums of finite families, $D \oplus E$ or $D + E$. In the more useful coalesced sum construction, we place the non-$\perp$ parts of the given posets side-by-side and add a common bottom below them; or (alternatively) we take their disjoint sums and then identify all the bottom elements into one.

4A.13. A point w in a set $X \subseteq D$ is **minimum** (least) in X if for every $x \in X$, $w \leq x$; it is **minimal** in X if there is no $x \in X$ such that $x < w$. **Maximum** (greatest) and **maximal** are defined in the same way.

A point $w \in D$ is an **upper bound** of a set $X \subseteq D$ if for every $x \in X$, $x \leq w$, and a **least upper bound** or **supremum** of X if it is least among such. It is clear that X can have at most one supremum which we will denote by $\sup X$ when it exists,

$$\sup X =_{\rm df} \inf \{x \in D \mid (\forall y \in X)[y \leq x]\};$$

and if $\sup X \in X$, then $\sup X$ is the maximum of X. If $x_0 \leq x_1 \leq \ldots$ is a non-decreasing sequence with a supremum, we will also use a limit notation,

$$\lim_n x_n =_{\rm df} \sup \{x_0, x_1, \ldots\}.$$

4A.14. **Exercise.** (1) Consider the powerset

$$\mathcal{P}(A) =_{\rm df} \{B \mid B \subseteq A\}$$

of a set A, partially ordered by the subset relation,

$$X \leq_{\mathcal{P}(A)} Y \iff_{\rm df} X \subseteq Y \quad (X, Y \subseteq A).$$

Verify that this is a poset in which every set $X \subseteq \mathcal{P}(A)$ has a supremum, namely its union $\cup X$.

(2) Show that in a flat poset $A_\perp$, the only sets which have suprema are subsets of doubletons $\{\perp, a\}$.

The situation in Part (2) of the Exercise is more typical, i.e., most sets in an arbitrary poset do not have suprema. The next, basic definition singles out a condition on a set $X \subseteq D$ which makes possible—and in many cases insures—the existence of sup X.

4A.15. A subset $X \subseteq D$ of a poset is a **chain** if every two members of X are comparable, i.e., if for all $x, y \in X$, either $x \leq y$ or $y \leq x$; and it is **directed**, if every two members of X have an upper bound in X, i.e.,

$$x, y \in X \Longrightarrow (\exists z \in X)[x \leq z \ \& \ y \leq z].$$

Every chain is directed, since $\max\{x, y\}$ is an upper bound of x and y, when x and y are comparable.

4A.16. **Exercise.** Show that if X is a directed set, then every finite set $x_1, \ldots, x_n$ of members of X has an upper bound $x \in X$. (Note that this fails for infinite sets, e.g., $\mathbb{N}$ has no upper bound as a subset of itself.)

4A.17. **Exercise.** If $f : D \to E$ is monotone and $X \subseteq D$ is directed, then the image

$$f[X] = \{f(x) \mid x \in X\}$$

is directed in E.

4A.18. **Completeness.** A poset D is **complete** (inductive, directed complete, a dcpo) if every directed set in D has a least upper bound.

Every complete poset has a minimum, because $\emptyset$ is a chain, and every non-decreasing sequence $x_0 \leq x_1 \leq \cdots$ in a complete poset has a limit, because $\{x_n \mid n \in \mathbb{N}\}$ is a chain.

For most of the applications, we only need the seemingly weaker assumption of **chain completeness**, that *every chain in D has a supremum.* In fact chain complete posets are directed complete, see Problem *x4A.6. We will work from the start with the seemingly stronger hypothesis, because some important results depend on it; because in practice, it is just as easy to establish full completeness as chain completeness; and because the proof of the equivalence of the two notions is a non-trivial exercise in set theory which invokes the Axiom of Choice and has little to do with recursion.

4A.19. **Exercise.** Show that each flat poset $B_\perp$ is complete, and that if W is complete, then so is its bottom lifting $W_\perp$.

4A.20. **Exercise.** Show that the posets $\mathrm{Seqs}(A)$ and $\mathrm{Streams}(A)_M$ of sequences and streams are complete.

4A.21. **Exercise.** Show that if each D_i is complete, then so is the coalesced sum $\sum_{i \in I} D_i$.

4A.22. **Proposition.** *If each D_i is a complete poset, then the product $\prod_{i\in I} D_i$ is also complete. In particular, if W is complete, then so is the function poset $(I \to W)$, for any I.*

Proof. Suppose $X \subseteq \prod_{i\in I} D_i$ is a directed set, and for each $i \in I$ let

$$X_i = \{u_i \in D_i \mid u \in X\}$$

be the *projection* of X to D_i. Each X_i is directed in D_i: because if $x = u_i, y = v_i \in X_i$, then there is some $z \in X$ such that $u, v \leq z$, and so $v_i, u_i \leq z_i \in X_i$. If we set $w_i =_{\rm df} \sup(X_i)$, then $w \in \prod_{i\in I} D_i$ and easily, $w = \sup X$. ⊣

4A.23. **Continuity.** A mapping $f : D \to E$ of one poset into another is **continuous** if for every directed $X \subseteq D$ and every $w \in D$,

$$[\emptyset \neq X \;\&\; w = \sup X] \Longrightarrow f(w) = \sup f[X], \tag{4-3}$$

i.e., if f preserves the suprema of non-empty directed sets whenever these suprema exist.

Continuous mappings are monotone, since if $x \leq y$, then $\sup\{x, y\} = y$, so (4-3) gives $f(y) = \sup\{f(x), f(y)\} \geq f(x)$. If D and E are complete, then (4-3) takes the simpler form

$$\emptyset \neq X \Longrightarrow f(\sup[X]) = \sup f[X],$$

and in particular, for every $x_0 \leq x_1 \leq \cdots$,

$$f(\lim_n x_n) = \lim_n f(x_n).$$

4A.24. **Exercise.** Show that every isomorphism $f : D \rightarrowtail\!\!\!\!\twoheadrightarrow E$ is continuous.

4A.25. **Exercise.** Show that if $g : D \to E$ and $f : E \to W$ are both continuous, then so is their composition $f \circ g : D \to W$.

An example of a discontinuous mapping is

$$f(x) = \begin{cases} a, & \text{if } x = a^n \neq a^\omega, \\ a^\omega & \text{if } x = a^\omega, \end{cases}$$

on $\mathrm{Seqs}(\{a\})$.

4A.26. **Exercise.** Show that if D is a discrete poset, then every mapping $f : D \to E$ is continuous, and if $D = A_\perp$ is flat, then a mapping $f : D \to E$ is continuous if and only if f is monotone.

4A.27. **Proposition.** *A mapping $f : D \to \prod_{i\in I} E_i$ is continuous if and only if each of its component mappings $f_i(x) = (f(x))_i, (i \in I)$ is continuous; in particular, $f : D \to (I \to E)$ is continuous, if and only if, for each $i \in I$, the mapping*

$$f_i(x) = f(x)(i)$$

is continuous.

PROOF. The continuity of each component follows trivially from the continuity of f. For the converse, suppose all components are continuous, $\emptyset \neq X \subseteq D$ is directed and $w = \sup X$. By hypothesis then,

$$f_i(w) = \sup \{f_i(x) \mid x \in X\} = \sup \{f(x)_i \mid x \in X\} \quad (i \in I),$$

which means exactly that $f(w) = \sup \{f(x) \mid x \in X\}$. ⊣

After these basic definitions, we now establish three simple but fundamental theorems about monotone and continuous mappings.

4A.28. **Separate continuity.** For each function $f : D \times E \to W$ and each $d \in D$, $e \in E$, let

$$f_d(y) =_{\text{df}} f(d, y) \quad (y \in E, f_d : E \to W),$$
$$f^e(x) =_{\text{df}} f(x, e) \quad (x \in D, f^e : D \to W).$$

We say that f is **monotone in the first variable** if each f^e is monotone, and **monotone in the second variable** if each f_d is monotone, and similarly for **continuous** in the first or the second variable. Less formally, we will also say that "$f(x, y)$ is monotone in x", "$g(x, y, z)$ is continuous in z", etc., meaning that "$\lambda(x)f(x, y)$ is monotone for each y", "$\lambda(z)g(x, y, z)$ is continuous for all x, y," etc.

Notice that with this terminology, if $f(x, y)$ is continuous in x and

$$\phi(x) = \lambda(y)f(x, y),$$

then $\phi : D \to (E \to W)$ is continuous, by 4A.27; this is the standard way of introducing continuous mappings into function posets.

4A.29. **Theorem.** *A function $f : D \times E \to W$ is monotone* (or continuous) *if and only if it is separately monotone* (or continuous) *in each of its variables.*

PROOF. If $f : D \times E \to W$ is monotone and $e \in E$, then for all $x, y \in D$,

$$\begin{aligned} x \leq y &\Longrightarrow (x, e) \leq (y, e) \\ &\Longrightarrow f(x, e) \leq f(y, e) \text{ because } f \text{ is monotone} \\ &\Longrightarrow f^e(x) \leq f^e(y), \end{aligned}$$

so that each f^e is monotone, and the same argument works for each f_d.

Conversely, if all mappings f^y and f_x are monotone, then

$$\begin{aligned} (x_1, y_1) \leq (x_2, y_2) &\Longrightarrow x_1 \leq x_2, y_1 \leq y_2 \\ &\Longrightarrow f(x_1, y_1) \leq f(x_2, y_1), f(x_2, y_1) \leq f(x_2, y_2) \\ &\Longrightarrow f(x_1, y_1) \leq f(x_2, y_2). \end{aligned}$$

For the more interesting direction of the part about continuity, suppose that all f_x, f^y are continuous, $X \subseteq D \times E$ is directed, non-empty, and $(a,b) = \sup X$. Now

$$\sup\{f(x,y) \mid (x,y) \in X\} \leq f(a,b),$$

immediately, using the monotonicity of f which follows from the first part of the Theorem, so it is enough to show that

$$f(a,b) \leq \sup\{f(x,y) \mid (x,y) \in X\}. \tag{4-4}$$

It is easy to verify (as in the proof of 4A.22) that the sets

$$X_1 = \{x \in D \mid (\exists y)(x,y) \in X\}, \quad X_2 = \{y \in E \mid (\exists x)(x,y) \in X\}$$

are both directed, and $a = \sup X_1$, $b = \sup X_2$. From the assumed continuity of each f_x, we have

$$\sup\{f(x,y) \mid y \in X_2\} = f(x,b) \quad (x \in X_1);$$

and from the continuity of f^b and this, we then get

$$\sup\{\sup\{f(x,y) \mid y \in X_2\} \mid x \in X_1\} = f(a,b), \tag{4-5}$$

or in somewhat different notation,

$$\sup_{x \in X_1} \sup_{y \in X_2} f(x,y) = f(a,b).$$

Now, if $x \in X_1$ and $y \in X_2$, there are y' and x' such that $(x,y') \in X$, $(x',y) \in X$; and then there is some $(x'',y'') \in X$ which is above both these pairs, since X is directed; and so, by the monotonicity of f, we have

$$x \in X_1, y \in X_2 \Longrightarrow f(x,y) \leq f(x'',y'') \leq \sup\{f(x,y) \mid (x,y) \in X\},$$

from which, taking sups over y and x successively and using (4-5) we get (4-4).

The other direction of the part about continuity is simpler and we skip it. ⊣

The second theorem gives a very simple characterization of continuity, for a case which appears to be very special but actually covers a large class of applications.

4A.30. **Theorem.** *For all sets A, B, C, a mapping*

$$f : (A \rightharpoonup B) \to C_\perp$$

is continuous if and only if it is monotone, and for every partial function $p : A \rightharpoonup B$,

$$f(p) = w \in C \Longrightarrow (\exists p^*)[p^* \leq p \;\&\; f(p^*) = w \;\&\; p^* \text{ is finite}]. \tag{4-6}$$

PROOF. Recall from 1.6 that a partial function p is finite if $\{x \mid p(x)\downarrow\}$ is finite, suppose first that $f : (A \rightharpoonup B) \to C_\perp$ is continuous, $f(p) = w$, and let

$$X_p = \{p^* \in (A \rightharpoonup B) \mid p^* \leq p \ \& \ p^* \text{ is finite}\}.$$

Lemma A. The set X_p is directed in $(A \rightharpoonup B)$, and $\sup X_p = p$.

PROOF. If p_1 and p_2 are both finite partial functions below p, then

$$q(x) = \begin{cases} p(x) & \text{if } p_1(x)\downarrow \text{ or } p_2(x)\downarrow, \\ \perp & \text{otherwise} \end{cases}$$

is also finite, above p_1 and p_2 and below p, and so X_p is directed. It is also clear that p is an upper bound of X_p. If q is another upper bound, then for every x such that $p(x)\downarrow$, let

$$p_x(t) = \text{if } (t = x) \text{ then } p(x) \text{ else } \perp;$$

now $p_x \in X_p$, and so $p_x \leq q$, so that $q(x) = p(x)$; and since x was arbitrary such that $p(x)\downarrow$, this shows that $p \leq q$. ⊣ (Lemma A)

Now the continuity of f implies that

$$f(p) = w \in C \Longrightarrow w = \sup \{f(p^*) \mid p^* \in X_p\},$$

which is exactly what (4-6) says.

For the other direction, suppose f satisfies (4-6), $X \subseteq (A \rightharpoonup B)$ is directed, non-empty, $p = \sup X$, and $f(p) = w \in C$. We need to show that there exists some $q \in X$ such that $f(q) = w$.

By (4-6), there is a finite $p^* \leq p$ such that $f(p^*) = w$. Let

$$A^* = \{x_0, \ldots, x_{n-1}\} = \{x \in A \mid p^*(x)\downarrow\},$$

and for each $i < n$, let

$$p_i(t) = \text{if } (t = x_i) \text{ then } p(x_i) \text{ else } \perp.$$

Lemma B. For each $i < n$, there is some $q_i \in X$ such that $p_i \leq q_i$.

PROOF. Suppose this fails for some i and let

$$q(t) = \text{if } (t \neq x_i) \text{ then } p(t) \text{ else } \perp.$$

Now if $r \in X$, then for every $t \neq x_i$, $r(t) \leq p(t) = q(t)$, and it must be the case that $r(x_i) = \perp$, otherwise $p_i \leq r$; so q is an upper bound of X and $p \leq q$, contradicting the fact that $p(x_i)\downarrow$ while $q(x_i)\uparrow$. ⊣ (Lemma B)

Since X is directed, we can find some $q \in X$ above $q_0, \ldots, q_{n-1}$, and then $p^* \leq q$, so that $f(q) \geq f(p^*) = w$, by the monotonicity of f, which completes the proof. ⊣

The theorem makes it clear that in most of the examples and the problems of Chapter 1 we looked for fixed points of continuous mappings. In the case of the Euclidean algorithm (Problem x1.3) for example, the relevant mapping was

$$f(p) = \lambda(m,n)\big[\text{if } n \mid m \text{ then } n \quad (m \geq n \geq 1)$$
$$\text{else } p(n, \text{rm}(m,n))\big],$$

which is continuous by 4A.30 and 4A.27. As we will see in Chapter **??**, these "functionals" on partial function posets are the only mappings needed in many parts of *classical abstract recursion theory*, and so their continuity (when it holds) will always be trivial, verified by direct inspection.

The third theorem in this group enriches substantially our stock of complete posets.

4A.31. **Theorem.** *Suppose D is a poset, E is a complete poset and F is a non-empty, directed subset of the complete poset $(D \to E)$.*

(1) *If every mapping in F is monotone, then* $\sup F : D \to E$ *is monotone.*

(2) *If every mapping in F is continuous, then* $\sup F : D \to E$ *is continuous.*

It follows that the posets

$$\text{Mon}(D \to E) =_{\text{df}} \{f : D \to E \mid f \text{ is monotone}\}$$
$$\text{Cont}(D \to E) =_{\text{df}} \{f : D \to E \mid f \text{ is continuous}\}$$

of the monotone and continuous mappings on D to E are complete subposets of $(D \to E)$.

Proof. Let $\overline{f} = \sup F$, so that, by definition,

$$\overline{f}(x) = \sup \{f(x) \mid f \in F\}.$$

If every $f \in F$ is monotone, then

$$\begin{aligned} x \leq y &\Longrightarrow f(x) \leq f(y) \quad (f \in F) \\ &\Longrightarrow f(x) \leq \overline{f}(y) \\ &\Longrightarrow \overline{f}(x) \leq \overline{f}(y), \end{aligned}$$

taking sups first on the right and then on the left. This proves (1).

For (2), suppose that each $f \in F$ is continuous, that $X \subseteq D$ is nonempty, directed and that $\overline{x} = \sup X$; we must show that

$$\overline{f}(\overline{x}) = \sup \{\overline{f}(x) \mid x \in X\}. \tag{4-7}$$

Notice first that for all $x \in X$ and $f \in F$, $f(x) \leq f(\overline{x})$, because f is monotone; and $f(\overline{x}) \leq \overline{f}(\overline{x})$ by the definition of $\overline{f}$; and hence

$$f(x) \leq \overline{f}(\overline{x}) \quad (x \in X, f \in F).$$

Taking the supremum over $f \in F$ on the left, we get

$$\overline{f}(x) \leq \overline{f}(\overline{x}) \quad (x \in X),$$

so that

$$\sup\{\overline{f}(x) \mid x \in X\} \leq \overline{f}(\overline{x}).$$

Suppose now that w is any upper bound of $\{\overline{f}(x) \mid x \in X\}$, so that

$$\overline{f}(x) \leq w \quad (x \in X).$$

Since $\overline{f}$ is the pointwise supremum of F, this gives

$$f(x) \leq w \quad (x \in X, f \in F),$$

and taking the supremum over X on the left, by the continuity of each $f \in F$,

$$f(\overline{x}) \leq w \quad (f \in F);$$

by the definition of $\overline{f}$, finally, this gives

$$\overline{f}(\overline{x}) \leq w,$$

so that $\overline{f}(\overline{x})$ is below every upper bound of $\{\overline{f}(x) \mid x \in X\}$, which completes the proof of (4-7). ⊣

4A.32. **Exercise.** Suppose D is a poset, Y is a set and E is a complete poset; show that

$$W = \{p : D \times Y \to E \mid p \text{ is continuous in its first variable}\}$$

is a complete poset. (In detail, the membership condition for W is that for every $y \in Y$ and every directed, non-empty $X \subseteq D$, if $w = \sup X$, then $p(w, y) = \sup\{p(x, y) \mid x \in X\}$.)

One might suspect from the terminology that posets can be viewed as topological spaces and the continuous mappings are exactly those which are topologically continuous. Indeed this is true, but we will not need these facts and we have left them for the problems. We include here only the basic definition.

4A.33. **Theorem (The Scott topology).** *A set $G \subseteq D$ is* **Scott open** *in a complete poset D if it is upward closed, i.e.,*

$$[x \in G \ \& \ x \leq y] \Longrightarrow y \in G,$$

and for every directed $X \subseteq D$,

$$[X \neq \emptyset \ \& \sup X \in G] \Longrightarrow (\exists x \in X)[x \in G].$$

A set $F \subseteq D$ is **Scott closed** *if its complement $D \setminus F$ is Scott open, i.e., F is downward closed and for every directed set $X \subseteq D$,*

$$X \neq \emptyset \ \& \ X \subseteq F \Longrightarrow \sup X \in F.$$

Notice that if G is Scott open and $\perp \in G$, then $G = D$.

The Scott open subsets of a complete poset D form a topology, Problem x4A.9, but not a very nice one from the geometrical point of view—it is not *Hausdorff*, since the only open set which contains $\perp$ is the whole space D. On the other hand and for the same trivial reason, the Scott topology is *compact*: because if $\mathcal{U}$ is any family of open sets such that $D \subseteq \bigcup \mathcal{U}$ (an *open cover*), then $D \in \mathcal{U}$ since D is the only open set which contains $\perp$, so that the singleton $\{D\} \subseteq \mathcal{U}$ is a finite subcover of $\mathcal{U}$.

Problems for Section 4A

x4A.1. Show that a finite poset is complete if and only if it has a least element.

x4A.2. If $\{x_{n,m}\}_{n,m}$ is a doubly-indexed sequence of points in a complete poset D which is non-decreasing in each of its arguments, then the supremum of its image $\sup\{x_{n,m} \mid n, m \in \mathbb{N}\}$ exists and

$$\sup\{x_{n,m} \mid n, m \in \mathbb{N}\} = \lim_n \lim_m x_{n,m} = \lim_m \lim_n x_{n,m}.$$

x4A.3. Prove that the liftup $p \mapsto \tilde{p}$ of a partial function to its strict extension (1-18) is an isomorphism of $(A \rightharpoonup B)$ with $\text{Strict}(A_\perp \to B_\perp)$.

*x4A.4. A **linearization** of a poset $(D, \leq_D)$ is any linearly ordered poset $(D, \leq)$ on the same field which extends D, i.e.,

$$x \leq_D y \Longrightarrow x \leq y.$$

(a) Prove that every finite poset admits a linearization, in fact

$$x \leq_D y \iff (\forall \leq \subseteq D \times D)[\leq \text{ a linearization} \Longrightarrow x \leq y]. \tag{4-8}$$

(b) (**AC**) Show that (4-8) holds for every poset D, so in particular, every poset admits a linearization. HINT: Show the result first for a countable D, without using any choice principles. For the general case, start with a wellordering of D and use definition by transfinite recursion to define for any two incomparable elements x, y in D a linearization $\leq$ in which $x < y$.

x4A.5. Prove that every countable, directed set X in a poset D contains a cofinal chain, i.e., a chain $C \subseteq X$ such that

$$x \in X \Longrightarrow (\exists z \in C)[x \leq z].$$

Infer that if every countable chain in D has a supremum, then every countable, directed set in D has a supremum; in particular, every countable, chain-complete poset is complete.

The next problem gives a generalization of this fact to arbitrary posets, which is the key to the proof that chain complete posets are complete. Notice that by "chain" in a poset D we mean any subset $C \subseteq D$ which

is linearly ordered by the poset partial ordering $\leq_D$; C is a *well ordered chain* if in addition, the restriction of $\leq_D$ to C is a wellordering. When we say that "X is well orderable" for some $X \subseteq D$, we mean that X *admits some wellordering* $\leq$, which may be (and typically is) totally unrelated to the given partial ordering $\leq_D$ of D.

*x4A.6. If every well ordered chain in a poset $(D, \leq_D)$ has a least upper bound, then for every well orderable, directed subset X of D there exists a well ordered chain C with the following two properties.

(1) X is bounded by C, i.e., for each $x \in X$ there exists some $y \in C$ such that $x \leq_D y$.

(2) For each $y \in C$, there exists a directed subset $C_y \subseteq X$ such that $|C_y| < |X|$ and $y = \sup_D C_y$.

Notice that C may satisfy these conditions without being a subset of X. HINT: (W. Allen.) Towards a contradiction, let X be a well orderable, directed counterexample to the conclusion which is least in cardinality among such, so that it is uncountable by x4A.5, and let $\leq$ be a **best wellordering** of X, i.e., a wellordering with least length, the cardinality of X. Define a function $f : X \times X \to X$ so that $x, y \in X \Longrightarrow x, y \leq_D f(x, y)$, and for every $x \in X$, let C_x be the least subset of X which contains $\{y \in X \mid y < x\}$ and is closed under f. Show that this is directed, that $\sup_D C_x$ exists for each $x \in X$, and that

$$C =_{\text{df}} \{\sup C_x \mid x \in X\}$$

is a well ordered chain in X which has properties (1) and (2) for X.

*x4A.7. (**AC**) The following three conditions are equivalent, for every poset D:

1. Every directed set in D has a least upper bound.
2. Every chain in D has a least upper bound.
3. Every well ordered chain in D has a least upper bound.

In particular: (**AC**) *A poset is complete if and only if it is chain complete.*

*x4A.8. (**AC**) Show that a monotone mapping $\pi : D \to E$ on one complete poset to another satisfies the identity

$$\pi(\sup X) = \sup \pi[X] \tag{4-9}$$

for every non-empty chain $X \subseteq D$, if and only if it satisfies (4-9) for every non-empty directed $X \subseteq D$.

x4A.9. Prove that the collection $\mathcal{S}(D)$ of all Scott open subsets of a complete poset D is a topology on D, i.e., (1) $\emptyset$ and D are Scott open; (2) if G, H are Scott open, then so is their intersection $G \cap H$; and (3) if $\mathcal{U}$ is a family of Scott open subsets of D, then their union $\cup\mathcal{U}$ is also Scott open.

Recall that a function $f : S \to T$ on one topological space to another is *topologically continuous* if for every open set $G \subseteq T$, the inverse image $f^{-1}[G]$ is open in S; equivalently, if for every closed $F \subseteq T$, $f^{-1}[F]$ is closed in S.

x4A.10. Prove that a mapping $f : D \to E$ on one complete poset to another is topologically continuous relative to the Scott topologies if and only if it is continuous in the sense of Definition 4A.23. HINT: Show that the inverse image of each closed set is closed.

*x4A.11. (**AC**) Prove that a set $F \subseteq D$ is Scott closed in a complete poset D if and only if for every non-empty chain $X \subseteq D$, if $X \subseteq F$, then $\sup X \in F$.

The uses of posets and complete posets in theoretical computer science have led to the development of a rich and elegant theory for these structures, generally called **domain theory**. We will not go into domain theory here, but it is worth including just one problem which gives something of its flavor, sometimes called "Scottery".

*x4A.12. A poset W has the **Scott property** if every pair of compatible points in W has a sup,

$$(\exists z)[x \leq z \ \& \ y \leq z] \Longrightarrow \sup\{x, y\} \text{ exists.}$$

(For example, discrete and flat posets have the Scott property.) Suppose W is a complete poset with the Scott property and prove the following:

(a) For every set A, the function poset $(A \to W)$ has the Scott property.

(b) For every poset D, there exists a monotone function

$$\pi : (D \to W) \to \text{Mon}(D \to W)$$

which is the identity on $\text{Mon}(D \to W)$ (a **projection**)

(c) $\text{Mon}(D \to W)$ has the Scott property.

(d) For every poset D, there exists a monotone function

$$\pi : (D \to W) \to \text{Cont}(D \to W)$$

which is the identity on $\text{Cont}(D \to W)$.

(e) $\text{Cont}(D \to W)$ has the Scott property.

Give an example where the projection π in (d) is not continuous.

Note. Several of the results in the problems are marked as depending on the Axiom of Choice, typically because their easiest (most natural) proofs appeal to Problem *x4A.6 and its Corollaries. I do not know whether these results (including *x4A.6) can be proved without **AC**, or whether (more likely) they are equivalent with some Choice Principles weaker than **AC**—although I imagine that at least some of the relevant independence results are easy or well-known.

4B. The Fixed Point Theorem

Despite its simplicity, the next basic result provides the mathematical foundation for the so-called *fixed point theory of programs.*

4B.1. **Theorem (The Continuous Fixed point Theorem).** (1) *Suppose $f : D \to D$ is a continuous mapping on a complete poset D and define by recursion on n the* **orbit** *of f,*

$$\overline{x}^0 = \overline{x}^0_f = f(\bot), \quad \overline{x}^{n+1} = \overline{x}^{n+1}_f = f(\overline{x}^n);$$

this is a non-decreasing sequence,

$$\overline{x}^0 \leq \overline{x}^1 \leq \ldots \ \overline{x}^n \leq \overline{x}^{n+1} \ldots,$$

and its least upper bound $\overline{x} = \lim_n \overline{x}^n$ satisfies the conditions:

$$\overline{x} \ = \ f(\overline{x}), \tag{4-10}$$

$$f(y) \leq y \implies \overline{x} \leq y. \tag{4-11}$$

In particular, every continuous mapping $f : D \to D$ on a complete poset has a least fixed point $\overline{x}$ which is uniquely determined by (4-10), (4-11).

(2) *If Y is a poset, D a complete poset and $f : D \times Y \to D$ a continuous mapping, then the function $\overline{x} : Y \to D$ defined by*

$$\overline{x}(y) = (\mu x \in D)[x = f(x, y)]$$

is continuous.

PROOF. (1) Skipping the subscripts and starting with the trivial $\bot \leq \overline{x}^0$, use the monotonicity of f to get immediately by induction that for all n, $\overline{x}^n \leq \overline{x}^{n+1}$; thus the limit $\overline{x}$ exists, and by the continuity of f,

$$f(\lim_n \overline{x}^n) = \lim_n f(\overline{x}^n) = \lim_n \overline{x}^{n+1} = \lim_n \overline{x}^n,$$

so that $\overline{x}$ is a fixed point of f. If $f(y) \leq y$, then by another easy induction, for each n, $\overline{x}^n \leq y$, so that $\overline{x} = \lim_n \overline{x}^n \leq y$. Finally, if w also satisfies (4-10), (4-11), then $\overline{x} \leq w$ taking $y = w$ in (4-11) and $w \leq \overline{x}$ taking $y = \overline{x}$ in the version of (4-11) for w.

(2) By the construction in Part (1),

$$\overline{x}(y) = \lim_n \overline{x}^n(y),$$

where $\overline{x}^0(y) = f(x, y)$ and $\overline{x}^{n+1}(y) = f(\overline{x}^n(y), y)$. Now each $\overline{x}^n : Y \to D$ is continuous, by induction, since composition preserves continuity; and then the limit function $\overline{x} = \lim_n \overline{x}^n$ is also continuous by Theorem 4A.31. ⊣

We use several notations for the least fixed point of a function, including

$$\overline{x}_f = \mathbf{fix}(f) = (\mu x \in D)[x = f(x)] =_{\rm df} \lim \overline{x}^n_f. \tag{4-12}$$

In practice, we often apply this basic fact to mappings on function posets:

4B.2. **Corollary.** *If $f : A \times (A \to W) \times X \to W$ is continuous, W is complete, and $x \in X$, then the recursive equation*

$$p(t) = f(t, p, x) \quad (t \in A, p : A \to W, x \in X)$$

has a least solution $\overline{p}_x : A \to W$ characterized by the conditions

$$\overline{p}_x(t) = f(t, \overline{p}_x, x) \quad (t \in A, x \in X),$$
$$\Big[x \in X, q : A \to W, (\forall t \in A)[f(t, q, x) \leq q(t)]\Big] \Longrightarrow \overline{p}_x \leq q.$$

In addition, the function $\overline{p} : A \times X \to W$ defined by

$$\overline{p}(t, x) = \overline{p}_x(t)$$

is continuous.

Proof. The mapping

$$\phi(p, x) = \lambda(t \in A) f(t, p, x)$$

on $\mathrm{Cont}(A \to W) \times X$ to $\mathrm{Cont}(A \to W)$ is continuous by Proposition 4A.27, the poset $\mathrm{Cont}(A \to W)$ is complete by 4A.31, and so 4B.1 gives us a continuous

$$\overline{\phi} : X \to \mathrm{Cont}(A \to W)$$

such that

$$\overline{\phi}(x) = (\mu p)[p = \lambda(t) f(y, p, x)].$$

If we set

$$\overline{p}(t, x) = \overline{p}_x(t) = \overline{\phi}(x)(t),$$

then this mapping is separately continuous, and hence continuous by 4A.27, and it (easily) has all the required properties. ⊣

4B.3. **Varying the parameter.** To make precise the counting of comparisons needed by the mergesort and the insert-sort algorithms in Chapter 1, we now introduce explicitly the dependence of $\mathrm{sort}(u)$ on the given ordering $\leq$ of the basic set L, or, more precisely, on parts of it. Let $r : L \times L \to \mathbb{B}_\perp$ vary over partial, binary relations on L, and consider first the recursive equation (in Problem x1.9) which defines the function $\mathrm{insert}(x, u)$, when r is the characteristic function of a total ordering of L:

$$\begin{aligned}(4\text{-}13)\quad \mathrm{insert}(x, u) = {} & \text{if } (|u| = 0) \text{ then } \langle x \rangle \\ & \text{else if } (r(x, u_0) = \mathrm{tt}) \text{ then } x \frown u \\ & \text{else if } (r(x, u_0) = \mathrm{ff}) \text{ then } u_0 \frown \mathrm{insert}(x, \mathrm{tail}(u)) \\ & \text{else } \perp\end{aligned}$$

The mapping

$$\begin{aligned}\phi(x,u,\mathrm{insert},r) = \text{if } (|u|=0) \text{ then } \langle x\rangle\\ \text{else if } (r(x,u_0)=\mathrm{tt}) \text{ then } x\frown u\\ \text{else if } (r(x,u_0)=\mathrm{ff}) \text{ then } u_0\frown \mathrm{insert}(x,\mathrm{tail}(u))\\ \text{else } \bot\end{aligned}$$

(with $\mathrm{insert}: L\times L^*\rightharpoonup L^*$ a variable—replace it by p if it looks confusing!) is evidently continuous by 4A.30 and 4A.29, and so Corollary 4B.2 yields a continuous mapping $\mathrm{insert}(x,u,r)$, such that for each r, the component

$$\mathrm{insert}_r(x,u) = \mathrm{insert}(x,u,r)$$

is the least solution of (4-13). Next consider the equation which expresses the insert-sort algorithm, with r again as a parameter and this $\mathrm{insert}(x,u,r)$,

$$\begin{aligned}\mathrm{sort}(u) = \text{if } (|u|\leq 1) \text{ then } u\\ \text{else } \mathrm{insert}(u_0,\mathrm{sort}(\mathrm{tail}(u),r);\end{aligned} \tag{4-14}$$

the mapping on the right is again continuous, and so we have a continuous mapping $\mathrm{isort}(u,r)$, such that for each r, the component

$$\mathrm{isort}_r(u) = \mathrm{isort}(u,r)$$

is the least solution of (4-14). We can now repeat the argument in Problem x1.9 (word-for-word) to establish the following property of $\mathrm{isort}(u,r)$:

4B.4. **Proposition.** (1) *If* $r: L\times L\to \mathbb{B}_\bot$ *is total and the relation*

$$\leq_r =_{\mathrm{df}} \{(x,y)\mid r(x,y)=\mathrm{tt}\}$$

is a total ordering of L, *then* $\mathrm{isort}(u,r)=\mathrm{sort}(u)$, *for this ordering* $\leq_r$.

(2) *For each* $u\in L^*$ *and each partial relation* $r: L\times L\to\mathbb{B}$, *if* $\mathrm{isort}(u,r)=v$, *then there exists some* $r^*\leq r$, *such that* $\mathrm{isort}(u,r^*)=v$ *and* $|r^*|\leq \frac{1}{2}|u|(|u|-1)$, meaning that the domain of r^* has no more than $\frac{1}{2}|u|(|u|-1)$ elements.

A similar analysis of the recursive equations which express the merge-sort algorithm yields a continuous mapping $\mathrm{msort}(u,r)$, such that *when* r *is the characteristic function of a total ordering* $\leq_r$, *then* $\mathrm{msort}(u,r)=\mathrm{sort}(u)$ *for that ordering*; and *when* $\mathrm{msort}(u,r)=v$, *then* $\mathrm{msort}(u,r^*)=v$, *for some* $r^*\leq r$ *of size no more than* $|u|\log_2|u|$. We will leave the details of this for Problem x4B.14.

4B.5. **Algorithms as continuous mappings.** The distinct, continuous mappings $\mathrm{isort}(u,r)$ and $\mathrm{msort}(u,r)$ refine the function $\mathrm{sort}(u)$, and differentiate between the insert-sort and the merge-sort algorithms by capturing in their modulus of continuity the separate requirements of these

algorithms for "resources" (the ordering). We might suspect from the example that algorithms can be "faithfully modeled" by such continuous mappings in general, and indeed, the classical, denotational semantics of programs assume (at least implicitly) a modeling of this kind. In the end we will adopt a refined modeling of algorithms which captures a great deal more than their continuous dependence on resources. Nevertheless, the method of variation of the parameter is often very useful in distinguishing between algorithms, by very simple arguments as in these examples.

The Continuous Fixed Point Theorem 4B.1 suffices for most of the applications of recursion theory to computer science, but its extension to monotone mappings is needed in some crucial places—and, in any case, it is indispensable for the development and applications of abstract recursion theory. Note that its proof is essentially that of 4B.1, with definition by ordinal recursion and proof by induction on the ordinals replacing the corresponding notions for the natural numbers.

4B.6. **Theorem (The Fixed Point Theorem).** (1) *For each monotone mapping $f : D \to D$ on a complete poset, there is a unique sequence $\{\overline{x}^\xi \mid \xi < \kappa\}$ indexed by the ordinals below some κ, so that the following hold.*

(a) $\eta < \xi < \kappa \Longrightarrow \overline{x}^\eta < \overline{x}^\xi$.
(b) $\xi < \kappa \Longrightarrow \overline{x}^\xi = f(\sup\{\overline{x}^\eta \mid \eta < \xi\})$.
(c) *The point* $\overline{x} = \sup\{\overline{x}^\xi \mid \xi < \kappa\}$ *is a fixed point of* f, $f(\overline{x}) = \overline{x}$.
(d) *For every* $y \in D$, $f(y) \leq y \Longrightarrow \overline{x} \leq y$.

In particular, every monotone mapping $f : D \to D$ on a complete poset has a least fixed point $\mathbf{fix}(f) = \overline{x} = (\mu x \in D)[x = f(x)]$ *which is uniquely determined by the conditions*

$$\begin{aligned} \overline{x} &= f(\overline{x}), \\ f(y) \leq y &\Longrightarrow \overline{x} \leq y. \end{aligned}$$

(2) *If D is complete and $f : D \times Y \to D$ is monotone, then the function*

$$\overline{x}(y) = (\mu x \in D)[x = f(x, y)]$$

is also monotone.

Proof. (1) By recursion on the ordinals, set first

$$\overline{x}^\xi = \begin{cases} f(\sup\{\overline{x}^\eta \mid \eta < \xi\}), & \text{if the set } \{\overline{x}^\eta \mid \eta < \xi\} \text{ is a chain,} \\ \bot, & \text{otherwise,} \end{cases}$$

and then prove by transfinite recursion on ξ that

$$\eta < \xi \Longrightarrow \overline{x}^\eta \leq \overline{x}^\xi,$$

so that, in particular, the second case in the definition never comes up and for all ordinals ξ,

$$\overline{x}^{\xi} = f(\sup\{\overline{x}^{\eta} \mid \eta < \xi\}).$$

Now, it cannot be the case that for all κ,

$$\sup\{\overline{x}^{\xi} \mid \xi < \kappa\} < \overline{x}^{\kappa};$$

because then the operation $\kappa \mapsto \overline{x}^{\kappa}$ would map the (proper) class of ordinals one-to-one into the set D, which is absurd. So there is a least

$$\kappa =_{\rm df} ||f||, \tag{4-15}$$

for which

$$f(\sup\{\overline{x}^{\xi} \mid \xi < \kappa\}) = \overline{x}^{\kappa} = \sup\{\overline{x}^{\xi} \mid \xi < \kappa\}, \tag{4-16}$$

which proves (a) – (c) of the theorem. The strong minimality of $\overline{x}$ is proved exactly as in the continuous case, using transfinite induction instead of induction on the natural numbers, and the fact that (a) – (c) characterize the sequence $\{\overline{x}^{\xi} \mid \xi < \kappa\}$ is also easy, by another transfinite induction.

Proof of (2) is similar to the proof of (2) in 4B.1, replacing again the ordinary induction on $\mathbb{N}$ by a simple transfinite induction and appealing to (the easier part) of 4A.31. ⊣

The **closure ordinal** $||f||$ of f defined in (4-15) is an important invariant associated with the recursive equation $x = f(x)$, a measure of its complexity.

Sometimes we can establish properties of $\mathbf{fix}(f)$ directly from its characterization as the least solution of $x = f(x)$, as we did in the proof of 1.9, while in other cases we must analyze the iterates of f which define it, e.g., in the proof of Part (2) of 4B.1. Somewhere in-between these two methods are proofs by the following principle.

4B.7. **Proposition (Scott induction).** *A property $P \subseteq D$ of points in a complete poset D is continuous if for every chain $X \subseteq D$,*

$$(\forall x \in X)P(x) \Longrightarrow P(\sup X),$$

so in particular $P(\bot)$. If P is continuous and $f : D \to D$ is monotone, then

$$(\forall x)[P(x) \Longrightarrow P(f(x))] \Longrightarrow P(\mathbf{fix}(f)).$$

PROOF. In the notation of the Fixed Point Theorem 4B.6, check by transfinite induction that for each ordinal ξ, $P(\overline{x}^{\xi})$. (If f is continuous we only need ordinary induction on $\mathbb{N}$ for this argument.) ⊣

4B.8. **Dumb search.** In Example 3 of the Introduction, we showed directly that for each $A \subseteq \mathbb{N}$, the partial function

$$q(m) = (\mu n \geq m)[n \in A] \tag{4-17}$$

is the least solution of the recursive equation

$$p(m) = \text{if } m \in A \text{ then } m \text{ else } p(m+1). \tag{4-18}$$

For another proof of this, verify (easily) that the property

$$P(p) \iff_{\text{df}} (\forall m)[p(m)\downarrow \Longrightarrow p(m) = (\mu n \geq m)[n \in A]]$$

is continuous on $(\mathbb{N} \rightharpoonup \mathbb{N})$, and

$$P(p) \Longrightarrow P(\lambda(m)[\text{if } m \in A \text{ then } m \text{ else } p(m+1)]).$$

This gives immediately by Scott Induction that if $\overline{p}$ is the least solution of (4-18), then

$$\overline{p}(m)\downarrow \Longrightarrow \overline{p}(m) = (\mu n \geq m)[n \in A],$$

and then we can finish off the proof by observing, as we did in 1.9, that the q defined in (4-17) satisfies (1-15), and so $\overline{p} = q$. Notice that this proof is "backwards" from what we did in 1.9: here we know that (4-18) has a least solution and we only need to identify it with q, while in 1.9 we started with the obvious solution and then proved that it is least.

Proofs by Scott Induction are often very elegant, but it is a rare case when the relevant continuous properties can be discovered without the kind of detailed analysis of iterates or computations illustrated by the proofs of 1.9 or 4B.1.

Some of the problems of this section ask for *the solution* of a recursive equation, not a precisely defined term. This should be understood as similar demands *to solve* or *to simplify* are understood in High School algebra: one is expected to describe the solution in explicit terms which give a better understanding of it than its identification as "the least solution of the given equation." The correct answer to *solve the recursive equation*

$$p(n) = \text{if } (n = 0) \text{ then } 0 \text{ else } p(n-1) + 1 \quad (n \in \mathbb{N}),$$

in $(\mathbb{N} \rightharpoonup \mathbb{N})$ is $\overline{p}(n) = n$.

Problems for Section 4B

x4B.1. A mapping $f : D \to D$ is **expansive** if for each $x \in D$, $x \leq f(x)$. Give an example of a mapping $f : D \to D$ which is monotone but not expansive and another one which is expansive but not monotone.

x4B.2. (The Fixed Point Theorem for expansive mappings.) Prove that if $f : D \to D$ is expansive and D is complete, then the recursive equation $x = f(x)$ has a solution in D; give an example where there is no least solution.

x4B.3. Suppose D is complete, $f : D \to D$ is monotone and let

$$E = \{x \in D \mid x \leq f(x) \ \& \ (\forall y)[f(y) \leq y \Longrightarrow x \leq y]\}.$$

Show that E is complete, $f[E] \subseteq E$, f is expansive on E, and there is only one solution $\overline{x}$ of $x = f(x)$ on E, which is the least solution of $x = f(x)$ on D. (This gives an alternative proof of the Fixed Point Theorem 4B.6, as a Corollary of the corresponding result for expansive mappings, Problem x4B.2.)

x4B.4. Show that for each monotone mapping $f : D \to D$ on a complete poset D, $(\mu x)[x = f(x)] = (\mu x)[x = f(f(x))]$.

x4B.5. Suppose $f, g : D \to D$ are continuous on the complete poset D, $f(\bot) = g(\bot)$ and f commutes with g, i.e., for all x, $f(g(x)) = g(f(x))$. Prove that $\mathbf{fix}(f) = \mathbf{fix}(g)$. HINT: Show by induction on n, that $\overline{x}_f^n = \overline{x}_g^n$.

*x4B.6. (G. Whitney.) Show that the hypothesis of continuity is necessary in the preceding Problem x4B.5.

*x4B.7. (Plotkin ?[?].) Suppose $f, g : D \to D$ are continuous on the complete poset D and such that

$$f(\bot) = g(\bot), \ gf = f^{(2)}g,$$

i.e., for all x, $g(f(x)) = f(f(g(x)))$. Show that $\mathbf{fix}(f) = \mathbf{fix}(g)$. HINT: Set $f(\bot) = g(\bot) = a$ and show that with a suitable, recursively defined (exponential) function $h : \mathbb{N} \to \mathbb{N}$ and all $n \geq 1$,

$$\overline{x}_g^n = f^{(h(n))}(a) = \overline{x}_f^{h(n)+1}.$$

(There is also a proof by Scott Induction, but it is not much simpler.)

*x4B.8. (G. Whitney.) Show that the continuity hypothesis is necessary in the preceding Problem *x4B.6.

x4B.9. Suppose $\overline{p}(t, k) = \overline{p}_k(t)$ where each $\overline{p}_k$ is the least solution of the recursive equation

$$\begin{aligned} p(t) = &\text{ if } t > k \text{ then } 1 \\ &\text{else if } t \notin A \text{ then } 0 \\ &\text{else } p(2t+1) \cdot p(2t+2), \end{aligned}$$

$t, k \in \mathbb{N}$, and $A \subseteq \mathbb{N}$ is a set of integers. Prove that $\overline{p}$ is a total function with values in $\{0, 1\}$, and

$$\overline{p}(0, k) = 1 \iff (\forall t \leq k)[t \in A].$$

x4B.10. Solve the recursive equation

$$p(i) = \text{if } (i = 0) \text{ then } 1 \text{ else } \min(p(i-1), r(i-1)),$$

where $r : \mathbb{N} \rightharpoonup \mathbb{N}, p : \mathbb{N} \rightharpoonup \mathbb{N}$.

x4B.11. Solve the recursive equation

$$p(m) = \text{if } (r(m) = \text{tt}) \text{ then } m \text{ else } p(m+1),$$

where $r : \mathbb{N} \to \mathbb{B}_\perp$, and if $\overline{p}_r$ is the solution for each r, let $\overline{p}(m,r) = \overline{p}_r(m)$. Prove that $\overline{p}(m,r)$ is continuous, and that

$$\overline{p}(m,r)\downarrow \Longrightarrow (\exists r^* \leq r)[\overline{p}(m,r^*)\downarrow \ \& \ |r^*| = 1 + \overline{p}(m,r) - m].$$

x4B.12. Solve the recursive equation[11]

$$\begin{aligned} R(u) = &\text{if Symbol}(u) \text{ then } u \\ &\text{else } R(\text{tail}(u)) \,;\, \langle \text{head}(u), \mathbf{t}\rangle \quad (u \in \text{Streams}(A)) \end{aligned}$$

and prove that the solution is continuous. For what pairs of streams does the equation $R(u\,;v) = R(v)\,;R(u)$ hold?

x4B.13. For each set L, define a continuous mapping

$$\text{stmax} : L^* \times (L \times L \rightharpoonup \mathbb{B}_\perp) \to L$$

with the following two properties, for each sequence u with $|u| = n > 0$:

1. If r is a total function and $\leq_r = \{(x,y) \mid r(x,y) = \text{tt}\}$ is a total ordering of L, then
$$\text{stmax}(u,r) = \max\{u_0, \ldots, u_{n-1}\}.$$
2. If $\text{stmax}(u,r)\downarrow$, then there exists some finite $r^* \leq r$, such that
$$\text{stmax}(u,r^*)\downarrow \text{ and } |r^*| \leq |u| - 1.$$

x4B.14. In the notation of 4B.3, prove that there is a continuous functional $\text{msort}(u,r)$, such that when r is the characteristic function of a total ordering $\leq_r$, then $\text{msort}(u,r) = \text{sort}(u)$ for that ordering; and when $\text{msort}(u,r) = v$, then $\text{msort}(u,r^*) = v$, for some $r^* \leq r$ of size no more than $|u| \log_2 2|u|$.

x4B.15. **Tail recursion.** Suppose $C \subseteq S$ and $\tau : S \to S$, $o : S \to B$ are functions, where S, B are arbitrary sets, and let $\overline{p} : S \rightharpoonup B$ be the least solution of the equation

$$p(s) = \text{if } s \in C \text{ then } o(s) \text{ else } p(\tau(s)). \tag{4-19}$$

Prove that

$$\overline{p}(s) = o(\tau^{(n(s))}(s)),$$

where

$$\tau^{(0)}(s) = s, \ \tau^{(n+1)}(s) = \tau(\tau^{(n)}(s))$$

---

[11] Here, for any $u \in \text{Streams}(A)$,

$$\text{Symbol}(u) \iff \text{lh}(u) = 2 \ \& \ (u)_1 = \mathbf{t} \quad (u \in \text{Streams}(A),$$

so that Symbol(u) holds when $u = (a, \mathbf{t})$ for some $a \in A$.

$\nu:$	1	0	1	1	0	0	0	$\cdots$
$u:$		u_0			u_1	**t**		
$v:$	v_0		v_1	v_2			$\cdots$	
$\nu[u,v]:$	v_0	u_0	v_1	v_2	u_1	v_2	v_3	$\cdots$

FIGURE 1. Action of binary merger on streams.

denotes *function iteration* and

$$n(s) = (\mu n \in \mathbb{N})\tau^{(n)}(s) \in C.$$

Equation (4-19) is called a **tail recursion**, because the "unknown" function p occurs just once on the right, at the tail end. Which of the recursive equations in the Introduction were tail recursions?

In the next problem we consider a different kind of merging of **streams**, which does not depend on a given ordering but on a separately given *merger*.

x4B.16. A (binary, strict) **merger** is any infinite sequence ν of 0's and 1's, and it is **fair** if it contains infinitely many 0's and infinitely many 1's. Figure 1 illustrates the action $\nu[u,v]$ of a merger ν on two streams u and v, which is defined precisely by the recursion:

$$\begin{aligned}(4\text{-}20)\qquad \nu[u,v] = \text{if } (\text{head}(\nu) = 0) \text{ then}&\\ &\text{if Symbol}(u) \text{ then head}(u) \frown v\\ &\text{else head}(u) \frown \text{tail}(\nu)[\text{tail}(u), v]\\ \text{else }\ &\text{if } (\text{head}(\nu) = 1) \text{ then}\\ &\text{if Symbol}(v) \text{ then head}(v) \frown u\\ &\text{else head}(v) \frown \text{tail}(\nu)[u, \text{tail}(v)].\end{aligned}$$

(1) Show that the map

$$(\nu, u, v) \mapsto \nu[u,v] : \text{Streams}(\{0,1\}) \times \text{Streams}(A)^2 \to \text{Streams}(A)$$

defined by the recursion (4-20) is continuous.

(2) Show that if ν is a fair merger and both u and v are infinite, then $\nu[u,v]$ is a **fair merge** of u and v in the following precise sense: there exist disjoint, infinite sets N_0, N_1 such that $N_0 \cup N_1 = \mathbb{N}$ and bijections $\pi_0 : N_0 \rightarrowtail \mathbb{N}$, $\pi_1 : N_1 \rightarrowtail \mathbb{N}$ such that

$$\nu[u,v](i) = \text{if } i \in N_0 \text{ then } u(\pi_0(i)) \text{ else } v(\pi_1(i)).$$

(3) Formulate and prove a similar characterization of $\nu[u,v]$ when ν is a fair merger and each of u and v is either convergent or infinite.

x4B.17. Solve the recursive equation

$$p(n,u,v) = \text{if } (\text{head}(u) = n) \text{ then } n \frown p(n+1,v,v)$$
$$\text{else } p(n,\text{tail}(u),v),$$

where $n \in \mathbb{N}$ and u, v vary over natural number streams, and in particular compute $p(0,u,u)$.

x4B.18. Fix a poset $(A, \leq)$ and solve the recursive equation

$$w(t) = \text{if } (\forall s < t)[w(s) = \text{tt}] \text{ then tt else ff}.$$

(This is a discontinuous equation and a bit of set theory is required to solve it.)

4C. Mutual recursion and the $\overline{\textbf{where}}$ construct

For each family $\{D_i \mid i \in I\}$ of complete posets and monotone mappings $f_i : \prod_{j\in I} D_j \to D_i$, the *system of mutual recursive equations*

$$x_i = f_i(x) \quad (i \in I) \tag{4-21}$$

has least solutions

$$\overline{x}_i = \overline{x}_i^{\kappa},$$

where the *simultaneous iterates* of the system are defined by the (mutual) transfinite recursion

$$\overline{x}_i^{\xi} = f_i(\lambda(j \in I)\, \sup_{\eta<\xi} \overline{x}_j^{\eta}) \quad (i \in I),$$

and κ is least such that

$$\overline{x}_i^{\kappa} = \sup\{\overline{x}_i^{\xi} \mid \xi < \kappa\}) \quad (i \in I).$$

This is seen by applying the Fixed Point Theorem 4B.6 to $f : D \to D$, where $D = \prod_{i\in I} D_i$ and f has components f_i, i.e., $f(x) = \lambda(i \in I) f_i(x)$. If the system is finite and each f_i is continuous, then we need only apply the simpler 4B.1 and the iteration formulas take the form

$$\left.\begin{aligned} \overline{x}_i^0 &=_{\text{df}} f_i(\bot, \ldots, \bot), \\ \overline{x}_i^{n+1} &=_{\text{df}} f_i(\overline{x}_1^n, \ldots, \overline{x}_k^n) \end{aligned}\right\}$$
$$\overline{x}_i = {}_{\text{df}}\lim_n \overline{x}_i^n.$$

The next simple but fundamental result reduces *mutual to iterated recursion* by justifying the solution of a system *one equation at a time*.

4C.1. **Theorem (The Bekič-Scott Lemma).** *Consider the system of two recursive equations*

$$\left.\begin{aligned} x &= f(x,y) \\ y &= g(x,y) \end{aligned}\right\} \tag{4-22}$$

where $f : D \times E \to D$, $g : D \times E \to E$ are monotone and the posets D, E are complete, and set

$$\hat{y}(x) = (\mu y \in E)[y = g(x, y)], \tag{4-23}$$

$$\begin{aligned} \overline{x} &= (\mu x \in D)[x = f(x, \hat{y}(x))], \\ &= (\mu x \in D)[x = f(x, (\mu y \in E)[y = g(x, y)])], \end{aligned} \tag{4-24}$$

$$\overline{y} = \hat{y}(\overline{x}); \tag{4-25}$$

the points $\overline{x}, \overline{y}$ then are the mutual, least solutions of the system (4-22).

PROOF. First, $\overline{x}, \overline{y}$ are solutions of the given system, since

$$f(\overline{x}, \overline{y}) = f(\overline{x}, \hat{y}(\overline{x})) = \overline{x}, \qquad g(\overline{x}, \overline{y}) = g(\overline{x}, \hat{y}(\overline{x})) = \hat{y}(\overline{x}), = \overline{y}.$$

To prove that they are the least solutions, suppose that

$$f(u, v) \leq u, \quad g(u, v) \leq v.$$

From the first of these and the minimality of $\hat{y}$ in (4-23) we get $\hat{y}(u) \leq v$, so that $f(u, \hat{y}(u)) \leq f(u, v) \leq u$; and then the minimality of $\overline{x}$ by (4-24) gives $\overline{x} \leq u$, the first of the inequalities we want to prove. The second follows from it, the definition of $\overline{y}$ and $\hat{y}(u) \leq v$ again,

$$\overline{y} = \hat{y}(\overline{x}) \leq \hat{y}(u) \leq v. \qquad \dashv$$

4C.2. **Exercise.** Suppose $\overline{x}_1, \ldots, \overline{x}_n$ are the least, mutual solutions of the system

$$x_i = f_i(x_1, \ldots, x_n) \quad (i = 1, \ldots, n)$$

where each D_i is complete and each $f_i : D_1 \times \cdots \times D_n \to D_i$ is monotone, let

$$\hat{x}_1 : D_2 \times \cdots \times D_n \to D_1$$

be the solution of

$$x_1 = f_1(x_1, \ldots, x_n),$$

and let $\hat{x}_2, \ldots, \hat{x}_n$ be the solutions of the system

$$x_i = f_i(\hat{x}_1(x_2, \ldots, x_n), x_2, \ldots, x_n) \quad (i = 2, \ldots, n).$$

Prove that $\overline{x}_i = \hat{x}_i$ for $i = 2, \ldots, n$ and $\overline{x}_1 = \hat{x}_1(\hat{x}_2, \ldots, \hat{x}_n)$.

In order to be able to express more easily and to use identities like (4-25), we introduce now a simple notation for mutual recursion.

4C.3. **Definition (The $\overline{\text{where}}$ construct).** For any poset X, complete posets $W, D_1, \ldots, D_n$ and monotone mappings

$$\begin{aligned} h &: X \times D_1 \times \cdots \times D_n \to W, \\ f_i &: X \times D_1 \times \cdots \times D_n \to D_i \quad (i = 1, \ldots, n), \end{aligned}$$

we set

$$h(x,\vec{d}) \overline{\textbf{where}} \{d_1 = f_1(x,\vec{d}),\dots,d_k = f_k(x,\vec{d})\} \tag{4-26}$$
$$=_{\text{df}} h(x,\overline{d}_{1,x},\dots,\overline{d}_{k,x}),$$

where, for each $x \in X$, $\overline{d}_{1,x},\dots,\overline{d}_{k,x}$ are the mutual, least fixed points of the system of equations

$$\begin{aligned} d_1 &= f_1(x,d_1,\dots,d_k)\\ d_2 &= f_2(x,d_1,\dots,d_k)\\ &\vdots\\ d_n &= f_n(x,d_1,\dots,d_k). \end{aligned}$$

This is one of several notations for mutual recursion used in programming languages. Two of the most popular variants of are

$$h(x,\vec{d}) \text{ with } \{d_1 = f_1(x,\vec{d}),\dots,d_k = f_k(x,\vec{d})\}$$

which simply uses "with" instead of "where", and

$$\text{letrec } \{d_1 = f_1(x,\vec{d}),\dots,y_k = f_k(x,\vec{d})\} \text{ in } h(x,\vec{d})$$

which puts the **head** of the recursive definition at the end.

The Bekič-Scott Lemma takes a simple, very general and useful form in this notation:

4C.4. **Theorem (The Bekič-Scott Rule).** *If $f_0,\dots,f_k$, $g_0,\dots,g_l$ are all monotone on complete posets, then*

$$\begin{aligned} f_0(r,\vec{p}) \overline{\textbf{where}} \{r = g_0(r,\vec{p},\vec{q}) \overline{\textbf{where}} \{q_1 = g_1(r,\vec{p},\vec{q}),\dots,q_l = g_l(r,\vec{p},\vec{q})\}&\\ p_1 = f_1(r,\vec{p}),\dots,p_k = f_k(r,\vec{p})\}&\\ = f_0(r,\vec{p}) \overline{\textbf{where}} \{r = g_0(r,\vec{p},\vec{q}), q_1 = g_1(r,\vec{p},\vec{q}),\dots,q_l = g_l(r,\vec{p},\vec{q})&\\ p_1 = f_1(r,\vec{p}),\dots,p_k = f_k(r,\vec{p})\}.& \end{aligned}$$

PROOF. For $l = 1$, $k = 0$ and $f_0(r,\vec{p}) = r$ the claimed identity becomes

$$\begin{aligned} r \overline{\textbf{where}} \{r = g_0(r,q) \overline{\textbf{where}} \{q = g_1(r,q)\}\}&\\ = r \overline{\textbf{where}} \{r = g_0(r,q), q = g_1(r,q)\},& \end{aligned}$$

which is exactly the Bekič-Scott Lemma 4C.1 (with r,q,g_0,g_1 in place of x,y,f,g). The proof of the general case is similar and we'll leave it for Problem x4C.1. ⊣

We used the letters p,q,r in this theorem to agree with the common choice of variables over function posets, in which these results have their most useful applications. For example, directly from the definitions:

4C.5. **Exercise.** Let $\mathbf{M} = (M, f_1, \dots, f_L)$ be a partial algebra. Show that a functional $\alpha(\vec{x}, p)$ on M is $\mathbf{M}$-recursive if it can be defined in the form

$$\begin{aligned}(4\text{-}27)\quad &\alpha(\vec{x}, p) = \alpha_0(\vec{x}, p, \vec{q})\\ &\quad \overline{\textbf{where}}\ \{q_1 = \lambda(\vec{u}_1)\alpha_1(\vec{u}_1, \vec{x}, p, \vec{q}), \dots, q_k = \lambda(\vec{u}_k)\alpha_k(\vec{u}_1, \vec{x}, p, \vec{q})\},\end{aligned}$$

where $\alpha_0, \dots, \alpha_k$ are $\mathbf{M}$-explicit functionals.

Next we collect some simple but useful identities satisfied by the $\overline{\textbf{where}}$ operator.

4C.6. **Theorem (Identities for $\overline{\textbf{where}}$).** *With all functions monotone from posets into complete posets:*

(1) **The head rule***:*

$$\begin{aligned}&\Big(f_0(\vec{p}, \vec{q})\ \overline{\textbf{where}}\ \{p_1 = f_1(\vec{p}, \vec{q}), \dots, p_k = f_k(\vec{p}, \vec{q})\}\Big)\\ &\qquad\qquad \overline{\textbf{where}}\ \{q_1 = g_1(\vec{q}), \dots, q_l = g_l(\vec{q})\}\\ &= f_0(\vec{p}, \vec{q})\\ &\qquad \overline{\textbf{where}}\ \{p_1 = f_1(\vec{p}, \vec{q}), \dots, p_k = f_k(\vec{p}, \vec{q}), q_1 = g_1(\vec{q}), \dots, q_l = g_l(\vec{q})\}\end{aligned}$$

(2) **The recap rules** (reducing application to recursion)*:*

$$\begin{aligned}f(g(x)) &= f(y)\ \overline{\textbf{where}}\ \{y = g(x)\},\\ f(g(x)\ \overline{\textbf{where}}\ \{x = h(x)\}) &= f(g(x))\ \overline{\textbf{where}}\ \{x = h(x)\}\\ &= f(y)\ \overline{\textbf{where}}\ \{y = g(x), x = h(x)\}.\end{aligned}$$

(3) **The λ-rule** (permuting the λ and the $\overline{\textbf{where}}$ operators)*: If x varies over a poset X and for $i = 0, \dots, k$, p_i varies over a complete poset D_i, then*

$$\begin{aligned}&\lambda(x)\Big[f_0(x, \vec{p})\ \overline{\textbf{where}}\ \{p_1 = f_1(x, \vec{p}), \dots, p_k = f_k(x, \vec{p})\}\Big]\\ &\qquad = f'_0(x, \vec{r})\ \overline{\textbf{where}}\ \{r_1 = f'_1(x, \vec{r}), \dots, r_k = f'_k(x, \vec{r})\},\end{aligned}$$

where each r_i varies over the poset $\mathrm{Mon}(X \to D_i)$ *and*

$$f'_i(x, \vec{r}) = \lambda(x) f_i(x, r_1(x), \dots, r_k(x)) \qquad (i = 0, \dots, k).$$

PROOF. We show only (3), and leave the similar arguments for (1) and (2) for the problems.

For each x, let $\overline{p}_{1,x}, \dots, \overline{p}_{k,x}$ be the solutions of the system LHS within the braces $\{\dots\}$ on the left-hand-side of the identity to be shown, and let $\overline{r}_1, \dots, \overline{r}_k$ be the solutions of the corresponding system RHS on the right-hand-side.

(a) *For* $i = 1, \dots, k$, $\lambda(x)\overline{p}_{i,x} \sqsubseteq \overline{r}_i$. It is enough to show that for each x, the tuple

$$\overline{r}_1(x), \dots, \overline{r}_k(x)$$

satisfies the equations of the system LHS, because then $\overline{p}_{i,x} \sqsubseteq \overline{r}_i(x)$, for each x, and hence $\lambda(x)\overline{p}_{i,x} \sqsubseteq \overline{r}_i$; and for this, we compute:

$$\begin{aligned}\overline{r}_i(x) &= f_i'(x, \overline{r}_1, \dots, \overline{r}_k)(x)\\ &= \Big(\lambda(x) f_i(x, \overline{r}_1(x), \dots, \overline{r}_k(x))\Big)(x) = f_i(x, \overline{r}_1(x), \dots, \overline{r}_k(x)).\end{aligned}$$

(b) *For* $i = 1, \dots, k$, $\overline{r}_i \sqsubseteq \lambda(x)\overline{p}_{i,x}$. It is enough to show that the tuple

$$\lambda(x)\overline{p}_{1,x}, \dots, \lambda(x)\overline{p}_{k,x}$$

satisfies the equations of the system RHS, because then $\overline{r}_i \sqsubseteq \lambda(x)\overline{p}_{i,x}$; and for this we compute,

$$\begin{aligned}\lambda(x)\overline{p}_{i,x} &= \lambda(x) f_i(x, \overline{p}_{1,x}, \dots, \overline{p}_{k,x})\\ &= \lambda(x) f_i(x, \Big(\lambda(x)\overline{p}_{1,x}\Big)(x), \dots, \Big(\lambda(x)\overline{p}_{k,x}\Big)(x))\\ &= f_i'(x, \Big(\lambda(x)\overline{p}_{1,x}\Big)(x), \dots, \Big(\lambda(x)\overline{p}_{k,x}\Big)(x)).\end{aligned}$$

Now (a) and (b) imply that $\overline{r}_i(x) = \overline{p}_{i,x}$ for all x and $i = 1, \dots, k$, and hence

$$\begin{aligned}\text{value of LHS} &= \lambda(x) f_0(x, \overline{p}_{1,x}, \dots, \overline{p}_{k,x})\\ &= \lambda(x) f_0(x, \overline{r}_1(x), \dots, \overline{r}_k(x))\\ &= f_0'(x, \overline{r}_1, \dots, \overline{r}_k) = \text{value of RHS},\end{aligned}$$

which completes the proof. ⊣

As an example of how these rules can be used, we give a direct proof of the First Recursion Theorem 3B.7 for partial algebras.

4C.7. **Proposition.** (1) *Suppose that* X *is a poset,* W *is a complete poset,* $\alpha : X \times \mathrm{Mon}(X \to W) \to W$, *and*

$$\alpha(x,p) = \alpha_0(x,p,\vec{q}) \ \overline{\textbf{where}}\ \{q_1 = \alpha_1(x,p,\vec{q}), \dots, q_k = \alpha_k(x,p,\vec{q})\},$$

where $\alpha_0, \dots, \alpha_k$ *are all monotone and take values in complete posets; then, abbreviating* $\vec{r}(x) = (r_1(x), \dots, r_k(x))$,

$$\begin{aligned}&p(x) \ \overline{\textbf{where}}\ \{p = \lambda(x)\alpha(x,p)\}\\ &\quad = p(x) \ \overline{\textbf{where}}\ \{p = \lambda(x)\alpha_0(x,p,\vec{r}(x)),\\ &\qquad\qquad r_1 = \lambda(x)\alpha_1(x,p,\vec{r}(x)), \dots, r_k = \lambda(x)\alpha_k(x,p,\vec{r}(x))\}.\end{aligned}$$

(2) (The First Recursion Theorem for a partial algebra $\mathbf{M}$.) *If $\alpha(\vec{x},p)$ is $\mathbf{M}$-recursive and operative, then its least fixed point*

$$p(\vec{x}) = p(\vec{x}) \ \overline{\textbf{where}} \ \{p(\vec{x}) = \alpha(\vec{x},p)\}$$

is $\mathbf{M}$-recursive.

PROOF. (1) By the λ-rule (3) of Theorem 4C.6,

$$\begin{aligned}&\lambda(x)\alpha(x,p)\\ &\quad = \lambda(x)\alpha_0(x,p,\vec{r}(x))\\ &\qquad \overline{\textbf{where}} \ \{r_1 = \lambda(x)\alpha_1(x,p,\vec{r}(x)),\dots,r_k = \lambda(x)\alpha_k(x,p,\vec{r}(x))\},\end{aligned}$$

and so,

$$\begin{aligned}&p(x) \ \overline{\textbf{where}} \ \{p = \lambda(x)\alpha(x,p)\}\\ &\qquad = p(x) \ \overline{\textbf{where}} \ \{p = \lambda(x)\alpha_0(x,p,\vec{r}(x)),\\ &\quad \overline{\textbf{where}} \ \{r_1 = \lambda(x)\alpha_1(x,p,\vec{r}(x)),\dots,r_k = \lambda(x)\alpha_k(x,p,\vec{r}(x))\}\},\end{aligned}$$

from which the claimed equation follows by the Bekič-Scott rule, Theorem 4C.4.

(2) follows immediately from (1) and Exercise 4C.5, taking $X = M^n$ and $W = M_{\mathbb{B}}$. ⊣

Problems for Section 4C

In the problems, all functions are assumed to be monotone, with arguments on various posets and values in complete posets, so that the exhibited equations make sense. We also use "vector notation" for n-part mutual recursion, i.e., $\vec{x} = (x_1\dots,x_n)$ stands for an n-tuple and $\vec{f}(\vec{x}) = \big(f_1(\vec{x}),\dots,f_n(\vec{x})$:

$$h(\vec{x}) \ \overline{\textbf{where}} \ \{\vec{x} = \vec{f}(\vec{x})\} =_{\text{df}} h(\vec{x}) \ \overline{\textbf{where}} \ \{x_1 = f_1(\vec{x}),\dots,x_n = f_n(\vec{x})\}.$$

x4C.1. Prove the Bekič-Scott Rule, Theorem 4C.4.

x4C.2. Prove the head rule, (1) of Theorem 4C.6.

x4C.3. Prove the recap rules, (2) of Theorem 4C.6.

x4C.4. Show that

$$h(y,\vec{x}) \ \overline{\textbf{where}} \ \{y = y, \vec{x} = \vec{f}(y,\vec{x})\} = h(\bot,\vec{x}) \ \overline{\textbf{where}} \ \{\vec{x} = \vec{f}(\bot,\vec{x})\}.$$

x4C.5. Show that

$$\begin{aligned}&h(y,z,\vec{x}) \ \overline{\textbf{where}} \ \{z = y, y = g(y,z,\vec{x}), \vec{x} = \vec{f}(y,z,\vec{x})\}\\ &\qquad = h(y,y,\vec{x}) \ \overline{\textbf{where}} \ \{y = g(y,y,\vec{x}), \vec{x} = \vec{f}(y,y,\vec{x})\}.\end{aligned}$$

x4C.6. Show that

$$h(x,y) \overline{\textbf{where}} \{x = f(x,y), y = f(x,y)\} = h(x,x) \overline{\textbf{where}} \{x = f(x,x)\}.$$

x4C.7. Show the following general formula which reduces mutual recursion with $n+1$ parts to n-part recursion:

$$h(y,\vec{x}) \overline{\textbf{where}} \left\{ y = g(y,\vec{x}), \vec{x} = \vec{f}(y,\vec{x}) \right\}$$
$$= \Big(h(y,\vec{x}) \overline{\textbf{where}} \left\{ \vec{x} = \vec{f}(y,\vec{x}) \right\} \Big)$$
$$\overline{\textbf{where}} \left\{ y = g(y,\vec{x}) \overline{\textbf{where}} \left\{ \vec{x} = \vec{f}(y,\vec{x}) \right\} \right\}.$$

The classical definition of addition on the natural numbers is by recursion on the second argument, i.e., $x + y = \mathrm{add}(x,y)$ where $\mathrm{add} : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}$ is the least solution of

$$\mathrm{add}(x,y) = \text{if } (y = 0) \text{ then } x \text{ else } \mathrm{add}(x, y-1) + 1. \tag{4-28}$$

It is easy to show by induction on y that every solution of (4-28) is total. Next one wants to know that add is *commutative*,

$$\mathrm{add}(x,y) = \mathrm{add}(y,x),$$

but this is not quite so simple as it requires a proof by "double induction." This problem gives a different proof of the commutativity of addition, which illustrates the Bekič-Scott Theorem and the method of proving properties of mutual recursive solutions by analyzing the mutual iterates.

*x4C.8. Let id, sum be the mutual solutions of the system.

$$\left.\begin{array}{rcl} \mathrm{id}(x) & = & \text{if } (x = 0) \text{ then } 0 \text{ else } \mathrm{id}(x-1) + 1, \\ \mathrm{sum}(x,y) & = & \text{if } (y = 0) \text{ then } \mathrm{id}(x) \text{ else } \mathrm{sum}(x, y-1) + 1. \end{array}\right\} \tag{4-29}$$

(1) Prove that $\mathrm{sum}(x,y) = \mathrm{add}(x,y)$, using only the recursive definition (4-28) of $\mathrm{add}(x,y)$.

(2) If $\mathrm{id}^n, \mathrm{sum}^n$ are the simultaneous iterates of the system (4-29), show that each sum^n is commutative, i.e., $\mathrm{sum}^n(x,y) = \mathrm{sum}^n(y,x)$. Infer that $\mathrm{sum}(x,y) = \mathrm{sum}(y,x)$.

CHAPTER 5

FUNCTIONAL RECURSION

In this chapter we will develop the general theory of recursion on **functional algebras**, i.e., partial algebras of the form

$$\mathbf{M} = (M, f_1, \dots, f_K), \tag{5-1}$$

where, however, the givens $f_1, \dots, f_K$ are **monotone functionals** on M, i.e., in general,

$$f_i : M^{n_i} \times (M^{k_{i,1}} \rightharpoonup M) \times \cdots \times (M^{k_{i,m_i}} \rightharpoonup M) \rightharpoonup M. \tag{5-2}$$

As in Chapter 2, we set

$$\text{arity}(f_i) = (n_i, k_{i,1}, \dots, k_{i,m_i}),$$

and the **characteristic** of the functional algebra encodes all of these,

$$\chi(\mathbf{M}) = (\text{arity}(f_1), \dots, \text{arity}(f_K)).$$

Fortunately, we will never need to actually put down these complex codes, but will be content to indicate informally (or let the context indicate) the arities of the various functionals involved and the algebra characteristic. Since the givens of a functional structure are functionals, we will pay special attention to the class $\mathbf{rec}_1(\mathbf{M})$ of **recursive functionals** of $\mathbf{M}$, which includes all the $\mathbf{M}$-recursive partial functions.

Functional recursion is a generalization of first-order recursion, and all the basic results of Chapter 2 extend quite easily to functional structures. More interesting here are the examples and the applications, and so we will start with a brief description of the most important of them in Section 5A and then use these examples to illustrate the general results in the remaining of this Chapter.

The main result of the Chapter is the *Stage Comparison Theorem* 5C.4 which, among other things, implies that the disjunction of two $\mathbf{M}$-semirecursive relations is $\mathbf{M}$-semirecursive, provided that $\mathbf{M}$ is *normal*; total (first-order) algebras are normal, and so this theorem will finally resolve some of the basic questions we left open in Chapter 2.

5A. The basic examples

These are expansions of (first-order) partial algebras by some specific, simple, monotone functionals which represent (in various ways) quantification. Some of them are "total functionals", in the most natural extension of the notion of "total function" to functionals:

5A.1. **Definition.** A **Kleene object** on M is a functional $f(\vec{x},\vec{p})$ such that

$$f(\vec{x},p_1,\dots,p_m)\downarrow \iff p_1,\dots,p_m \text{ are total functions.}$$

Notice that a partial function $f : M^n \rightharpoonup M_{\mathbb{B}}$ is a Kleene object exactly when it is totally defined, and that for every functional $f(\vec{x},\vec{p})$, its restriction to total arguments is a Kleene object. Kleene objects are, obviously, monotone, deterministic, and if M is infinite, they are not continuous.

5A.2. **The Kleene quantifiers**. The simplest (non-trivial) Kleene object is the Kleene representation of the existential quantifier on M,

$$E_M(p) = \begin{cases} \mathrm{tt}, & \text{if } p \text{ is total and } (\exists t \in M)[\ddot{\neg}p(t) = \mathrm{tt}], \\ \mathrm{ff}, & \text{if } (\forall t \in M)[\ddot{\neg}p(t) = \mathrm{ff}], \end{cases} \tag{5-3}$$

where the partial function $\ddot{\neg}(s)$ is defined in (3-9) and associates with every total function $p : M^n \to M$ the characteristic function of the relation

$$p(\vec{x}) \iff p(\vec{x}) = \mathrm{tt}.$$

5A.3. **The search functional**. This was defined in (3-45), which we repeat here for easy reference:

$$E_M^{\frac{1}{2}}(p) = \begin{cases} \mathrm{tt}, & \text{if } (\exists t \in M)[p(t) = \mathrm{tt}], \\ \bot, & \text{otherwise.} \end{cases} \tag{5-4}$$

It is called the **search functional** or **half existential quantifier** over M, and it is evidently continuous but not deterministic. Recursion in the expansion $(\mathbf{M}, E_M^{\frac{1}{2}})$ of a partial algebra $\mathbf{M}$ is generally referred-to as **search recursion** or **search computability** on $\mathbf{M}$, depending on whether $\mathbf{M}$ is arbitrary or admits a **recursive pair**, as we will make this precise.

Search computability is essentially a first-order kind of computability, and it has many of the properties of Turing computability on $\mathbb{N}$ which do not hold in general for prime recursion. It was introduced and studied in the period 1950 - 1975 in many different ways by various logicians (including Fraisse, Lacombe, Moschovakis and Friedman) and the most basic results about it concern the (basic) equivalence of these diverse definitions.

5A.4. **The non-deterministic connectives**. With p, q varying over $M_\perp$, i.e., the nullary partial functions on M, we set

$$p \vee q = \begin{cases} \text{tt}, & \text{if } \ddot{\neg}p = \text{tt} \text{ or } \ddot{\neg}q = \text{tt}, \\ \text{ff}, & \text{if } \ddot{\neg}p = \text{ff} \text{ and } \ddot{\neg}q = \text{ff}, \end{cases} \tag{5-5}$$

$$p \,\&\, q = \dot{\neg}(\dot{\neg}p \vee \dot{\neg}q) = \begin{cases} \text{tt}, & \text{if } \ddot{\neg}p = \text{tt} \text{ and } \ddot{\neg}q = \text{tt}, \\ \text{ff}, & \text{if } \ddot{\neg}p = \text{ff} \text{ or } \ddot{\neg}q = \text{ff}. \end{cases} \tag{5-6}$$

5A.5. **Quantifiers**. An **n-ary, monotone quantifier** on a set M is any non-trivial, monotone collection of subsets of M^n,

$$\emptyset \subsetneq \mathsf{Q} \subsetneq \mathcal{P}(M^n), \qquad [X \in \mathsf{Q} \,\&\, X \subseteq Y] \Longrightarrow Y \in \mathsf{Q}.$$

For any relation $R(\vec{x}, \vec{y})$ on M we write

$$(\mathsf{Q}\vec{x})R(\vec{x}, \vec{y}) \iff \{\vec{x} \mid R(\vec{x}, \vec{y})\} \in \mathsf{Q},$$

and we define the **dual quantifier** of Q by

$$\breve{\mathsf{Q}} = \{X \subseteq M^n \mid M^n \setminus X \notin \mathsf{Q}\}$$

so that

$$\breve{\mathsf{Q}}(\vec{x}, \vec{y}) \iff \neg(\mathsf{Q}\vec{x})\neg R(\vec{x}, \vec{y}).$$

The dual of a monotone quantifier is (easily) also monotone, Problem x5A.2.

The standard examples from logic are the unary

$$\exists = \{X \subseteq M \mid X \neq \emptyset\}, \quad \forall = \breve{\exists} = \{M\},$$

and the **cardinality quantifiers**, one for each cardinal κ,

$$(\mathsf{Q}_\kappa x)R(x, \vec{y}) \iff |\{x \mid R(x, \vec{y})\}| \geq \kappa.$$

Many more quantifiers arise naturally in definability theory, e.g., the binary **well-foundedness** quantifier,

$$\begin{aligned} (\mathsf{WF}x, y)R(x, y) &\iff (\forall f : \mathbb{N} \to M)(\exists n)R(f(n), f(n+1)) \\ &\iff \text{the negation } \neg R \text{ is well founded}, \end{aligned} \tag{5-7}$$

and its dual

$$\begin{aligned} (\breve{\mathsf{WF}}x, y)R(x, y) &\iff (\exists f : \mathbb{N} \to M)(\forall n)R(f(n), f(n+1)) \\ &\iff \text{there is an infinite descending chain in } R. \end{aligned} \tag{5-8}$$

With each n-ary quantifier we will associate three monotone functionals which represent it in various ways.

First, there is the **half-Q** functional, which we have already introduced for $\mathsf{Q} = \exists$:

$$Q^{\frac{1}{2}}(p) = \begin{cases} \mathrm{tt}, & \text{if } (\mathsf{Q}\vec{x})[\ddot{\neg}p(\vec{x}) = \mathrm{tt}], \\ \bot, & \text{otherwise.} \end{cases} \tag{5-9}$$

The half-$\breve{\mathsf{Q}}$ functional for the dual quantifier is defined in the same way, and then we put these two together in the **sharp functional** for Q:

$$Q^{\#}(p) = \begin{cases} \mathrm{tt}, & \text{if } (\mathsf{Q}\vec{x})[\ddot{\neg}p(\vec{x}) = \mathrm{tt}], \\ \mathrm{ff}, & \text{if } (\breve{\mathsf{Q}}\vec{x})[\ddot{\neg}p(\vec{x}) = \mathrm{ff}], \\ \bot, & \text{otherwise.} \end{cases} \tag{5-10}$$

For the existential quantifier then, the sharp functional representing it is $E^{\#}_M(p)$, defined in (3-13).

The Kleene object associated with a quantifier Q is the restriction of $Q^{\#}$ to total functions:

$$Q(p) = \begin{cases} Q^{\#}(p), & \text{if } p : M^n \to M \text{ is total,} \\ \bot, & \text{otherwise.} \end{cases} \tag{5-11}$$

This is defined in (5-3) for the existential quantifier.

Notice that for total p with values in $\{\mathrm{ff}, \mathrm{tt}\}$,

$$(\mathsf{Q}\vec{x})[p(\vec{x}) = \mathrm{tt}] \iff Q^{\#}(p) = \mathrm{tt} \iff Q(p) = \mathrm{tt},$$
$$(\breve{\mathsf{Q}}\vec{x})[p(\vec{x}) = \mathrm{tt}] \iff Q^{\#}(\dot{\neg}p) = \mathrm{ff} \iff Q(\dot{\neg}p) = \mathrm{ff},$$

where we have used the natural abbreviation

$$\dot{\neg}p = \lambda(\vec{x})[\dot{\neg}p(\vec{x})],$$

skipping the obvious λ-abstraction operation.[12]

The main difference between the two main functionals associated with Q is that $Q^{\#}$ is typically non-deterministic while the Kleene object Q is always deterministic, and this makes, in general, recursion in the first very different from recursion in the second.

Problems for Section 5A

x5A.1. Prove the identity of the two expressions for $p \,\&\, q$ given in (5-6).

x5A.2. Show that the dual $\breve{\mathsf{Q}}$ of a monotone quantifier is a monotone quantifier.

[12] Much as we write $f + g$ for the function $\lambda(x)[f(x) + g(x)]$.

x5A.3. Compute $(\widetilde{\mathsf{Q}}_{\mathbb{N}}x)R(x)$ for $\mathsf{Q} = \mathsf{Q}_\omega$ on $\mathbb{N}$.

x5A.4. Prove that for every monotone quantifier Q, if $p : M^n \to M$ is a total function, then $Q^{\#}(p)\downarrow$.

x5A.5. Find all the n-ary quantifiers Q on a set M such that $Q^{\#}(p)$ is a deterministic functional.

x5A.6. Prove that for all $p, q \in M_\perp$,

$$p \vee q = E^{\#}(\lambda(s)[\text{if } (s = 0) \text{ then } \ddot{\neg}p \text{ else } \ddot{\neg}q]).$$

5B. Explicit and recursive functionals

Recall from Section 2A that (recursively)

$$(M^{n+1} \rightharpoonup M_{\mathbb{B}}) = (M \rightharpoonup (M^n \rightharpoonup M_{\mathbb{B}})),$$

so that if $p : M^{n+1} \rightharpoonup M_{\mathbb{B}}$ and $x \in M$, then $p(x) : M^n \rightharpoonup M_{\mathbb{B}}$. This convention works especially well with the operation of λ-**abstraction**: if, e.g., $p : M^3 \rightharpoonup M_{\mathbb{B}}$, then for any z,

$$\lambda(x)\lambda(y)p(y,z,x) = \lambda(x,y)p(y,z,x) : M^2 \rightharpoonup M_{\mathbb{B}}$$
$$\text{and } \lambda(x)\lambda(y)p(y,z,x)\Big(a,b\Big) = p(b,z,a).$$

5B.1. **The formal language $\mathsf{R}(\tau)$ for functional structures**. To avoid cumbersome notations, we will assume for the basic definitions that the given vocabulary is of the form

$$\tau = \langle \mathsf{f}_1, \dots, \mathsf{f}_K, \mathsf{F} \rangle,$$

where $\mathsf{f}_1, \dots, \mathsf{f}_K$ are constants denoting partial functions and F names a functional $F(x,p)$ with one individual and one binary partial function argument. The terms of $\mathsf{R}(\tau)$ for such a vocabulary are then defined by

$$\text{(5-12)} \quad A :\equiv \mathrm{ff} \mid \mathrm{tt} \mid \mathsf{v}_i \mid \zeta_i^n(A_1, \dots, A_n) \mid \mathsf{f}_i(A_1, \dots, A_{n_i}) \mid \mathsf{F}(A_1, \lambda(\mathsf{v}_i)\lambda(\mathsf{v}_j)B) \mid (\text{if } A_1 \text{ then } A_2 \text{ else } A_3)$$

where the individual variables $\mathsf{v}_i, \mathsf{v}_j$ are distinct in the new case which applies F. All the basic facts about terms (e.g., unique readability) are established as usual, and there is only one additional complication: some of the occurrences of individual variables in terms are now **bound** by the λ-operator. These are specified by a simple recursion on the terms.

A **valuation** of the variables into a set M is any (total) function π which assigns to each v_i some $\pi(\mathsf{v}_i) \in M$, and to each n-ary ζ_i^n some partial

function $\pi(\zeta_i^n) : M^n \rightharpoonup M_{\mathbb{B}}$. The denotation $\mathrm{den}(A, \pi)$ of each term in a functional algebra

$$\mathbf{M} = (M, f_1, \dots, f_K, F)$$

is defined by adding just one clause for F to the definition for partial algebras, which we repeat for easy reference:

$$\mathrm{den}(\mathsf{ff}, \pi) = \mathsf{ff}; \quad \mathrm{den}(\mathsf{tt}, \pi) = \mathsf{tt}; \quad \mathrm{den}(\mathsf{v}_i, \pi) = \pi(\mathsf{v}_i)$$
$$\mathrm{den}(\mathsf{f}_i(A_1, \dots, A_{n_i}), \pi) = f_i(\mathrm{den}(A_1, \pi), \dots, \mathrm{den}(A_{n_i}, \pi))$$
$$\mathrm{den}(\boldsymbol{\zeta}_i^n(A_1, \dots, A_n), \pi) = \pi(\zeta_i^n)(\mathrm{den}(A, \pi), \dots, \mathrm{den}(A_n, \pi))$$

$$\mathrm{den}(\mathsf{F}(A_1, \lambda(\mathsf{v}_i)\lambda(\mathsf{v}_j)B), \pi)$$
$$= F(\mathrm{den}(A_1, \pi), \lambda(x)\lambda(y)\mathrm{den}(B, \pi\{\mathsf{v}_i := x, \mathsf{v}_j := y\}))$$

$$\mathrm{den}(\text{if } A_1 \text{ then } A_2 \text{ else } A_3, \pi)$$
$$= \begin{cases} \mathrm{den}(A_2, \pi), & \text{if } \mathrm{den}(A_1, \pi) = \mathsf{tt}, \\ \mathrm{den}(A_3, \pi), & \text{if } \mathrm{den}(A_1, \pi)\downarrow \ \&\neq \mathsf{tt}, \\ \bot, & \text{otherwise, i.e., if } \mathrm{den}(A_1, \pi)\uparrow. \end{cases}$$

A functional $f(\vec{x}, \vec{p})$ on M is **M-explicit** if there is a term A whose free variables are all in a fixed list $\vec{\mathsf{x}}, \vec{\boldsymbol{\zeta}}$, such that for all $\vec{x} \in M^n$ and partial functions $\vec{p}$ of matching arities,

$$f(\vec{x}, \vec{p}) = \mathrm{den}(A, \{\vec{\mathsf{v}} := \vec{x}, \vec{\boldsymbol{\zeta}} := \vec{p}\}).$$

The class of **M**-explicit functionals can be characterized exactly as for the case of partial algebras in Proposition 3A.6, with the addition of closure under λ-substitution into the givens:

5B.2. **Definition.** A set $\mathcal{F}$ of functionals on the universe M of a functional algebra **M** is **explicitly closed** (over **M**) if it satisfies conditions (1) – (4) of Definition 3A.3 and also

(5) $\mathcal{F}$ is closed under λ-substitution into the givens, i.e., the definition scheme

$$f(y, \vec{x}, \vec{p}) = F(y, \lambda(s)\lambda(t)h(s, t, \vec{x}, \vec{p})), \tag{5-13}$$

if $F(x, p)$ is the only given, with p binary.

5B.3. **Proposition.** *Let* **M** *be a functional algebra.*

(1) *The set* $\mathbf{exp}_1(\mathbf{M})$ *of* **M***-explicit functionals is the smallest set of functionals on* M *which is explicitly closed over* **M**.

(2) *Every* **M***-explicit functional is monotone.*

(3) *If every given of* **M** *is deterministic, then every* **M***-explicit functional is deterministic.*

PROOF of (1) is exactly like that of Proposition 3A.6, with an additional, simple argument to handle (5). (2) and (3) follow easily, by checking that the sets of monotone and monotone deterministic functionals on M satisfies (1) – (5). We outline briefly the proof of (3) for monotone deterministic functionals, in a notationally simple case.

Lemma. If $g(\vec{x}, r, q)$ and $h(t, \vec{y}, p)$ are monotone and deterministic functionals and

$$f(\vec{x}, \vec{y}, p, q) = g(\vec{x}, \lambda(t)h(t, \vec{y}, p), q),$$

then $f(\vec{x}, \vec{y}, p, q)$ is deterministic.

Fix $\vec{x}, \vec{y}, p, q$ and suppose that

$$f(\vec{x}, \vec{y}, p, q) = g(\vec{x}, \lambda(t)h(t, \vec{y}, p), q) = w.$$

By the determinism of g, there are $\sqsubseteq$-least partial functions $r, \hat{q}$ such that

$$r \sqsubseteq \lambda(t)h(t, \vec{y}, p), \ \hat{q} \sqsubseteq q, \text{ and } g(\vec{x}, r, \hat{q}) = w;$$

and by the determinism of h, for each t for which $r(t)\downarrow$, there is a least p_t such that

$$p_t \sqsubseteq p \text{ and } r(t) = h(t, \vec{y}, p_t);$$

it follows that

$$\hat{p} = \bigcup \{p_t \mid r(t)\downarrow\} \sqsubseteq p.$$

Claim 1. $f(\vec{x}, \vec{y}, \hat{p}, \hat{q}) = w$. This is because

$$\text{if } r(t)\downarrow, \text{ then } r(t) = h(t, \vec{y}, p_t) = h(t, \vec{y}, \hat{p})$$

by monotonicity, which means that

$$r \sqsubseteq \lambda(t)h(t, \vec{y}, \hat{p});$$

and since also $g(\vec{x}, r, \hat{q}) = w$, by monotonicity again,

$$w = g(\vec{x}, r, \hat{q}) = g(\vec{x}, \lambda(t)h(t, \vec{y}, \hat{p}), \hat{q}).$$

Claim 2. If $p' \sqsubseteq p$, $q' \sqsubseteq q$ and $f(\vec{x}, \vec{y}, p', q') = w$, then $\hat{p} \sqsubseteq p'$ and $\hat{q} \sqsubseteq q'$. This is because (with the notation set above), the hypothesis $g(\vec{x}, \lambda(t)h(t, \vec{y}, p'), q') = w$ gives

$$r \sqsubseteq \lambda(t)h(t, \vec{x}, p'), \quad \hat{q} \sqsubseteq q',$$

and the second of these inequalities is one of the facts we needed to verify. For the other we appeal to the first inequality: for each t, if $r(t)\downarrow$, then $h(t, \vec{x}, p') = r(t)$, and so $p_t \sqsubseteq p'$, and so

$$\hat{p} = \bigcup \{p_t \mid r(t)\downarrow\} \sqsubseteq p',$$

as required. ⊣

Finally, the closure of $\mathbf{exp}(\mathbf{M})$ under full λ-substitution is also proved like the corresponding Proposition 3A.11, with just one additional, simple case to handle λ-substitution into the givens:

5B.4. **Proposition.** *The set* $\mathbf{exp}(\mathbf{M})$ *is closed under the following scheme of* λ**-substitution***:*

$$f(\vec{x},\vec{y},\vec{p},\vec{q}) = h(\vec{x},\lambda(\vec{t})h(\vec{t},\vec{y},\vec{p}),\vec{q}). \tag{5-14}$$

We leave the proof for Problem x5B.1.

5B.5. **Definition.** A (monotone) **operator** on M is any monotone mapping

$$\alpha : (M^{k_{i,1}} \rightharpoonup M) \times \cdots \times (M^{k_{i,m_i}} \rightharpoonup M) \to (M^n \rightharpoonup M), \tag{5-15}$$

and the **associated functional** of α is defined by evaluation,

$$\alpha^*(\vec{x},\vec{p}) = \alpha(\vec{p})(\vec{x}). \tag{5-16}$$

Conversely, for each functional $f(\vec{x},\vec{p})$, we can define by λ-abstraction the associated operator

$$\alpha_f(\vec{p}) = \lambda(\vec{x})f(\vec{x},\vec{p}), \tag{5-17}$$

so that $\alpha_f^* = f$. It is useful to view operators of this form as just another way of considering M-functionals, which in some cases simplifies the notation.

The **target space** of α is $(M^n \rightharpoonup M)$ in (5-15), including the case $(M^0 \rightharpoonup M) = M \cup \{\perp\}$ when $n = 0$.

The **M**-recursive partial functions are the least solutions of systems of recursive equations with **M**-explicit parts, and we can define them exactly as we did for partial algebras in Proposition 3A.2. We will, however, take advantage of the $\overline{\textbf{where}}$ construct introduced in Section 4C, which makes it possible to define directly the **M**-recursive functionals and to establish very easily their basic properties.

5B.6. **Definition (M-recursion).** A functional $f(\vec{x},\vec{p})$ is **recursive** in a functional algebra **M**, if

$$\begin{aligned} f(\vec{x},\vec{p}) &= \alpha_0(\vec{x},\vec{p},q_1,\ldots,q_m) \\ &\quad \overline{\textbf{where}}\ \{q_1 = \alpha_1(\vec{x},\vec{p},q_1,\ldots,q_m),\ldots,q_m = \alpha_m(\vec{x},\vec{p},q_1,\ldots,q_m)\} \end{aligned} \tag{5-18}$$

where α_0 is an **M**-explicit functional, $\alpha_1,\ldots,\alpha_m$ are **M**-explicit operators, and each function variable q_i varies over the target space of α_i (so that the definition makes sense). Unless the detail is needed, we will typically write (5-18) in vector form,

$$f(\vec{x},\vec{p}) = \alpha_0(\vec{x},\vec{p},\vec{q})\ \overline{\textbf{where}}\ \{\vec{q} = \vec{\alpha}(\vec{x},\vec{p},\vec{q})\}. \tag{5-19}$$

As with partial algebras, $\mathbf{rec}_1(\mathbf{M})$ is the collection of $\mathbf{M}$-recursive functionals.

A functional $\alpha(\vec{x},\vec{p})$ is a **simple fixed point** of $\mathbf{M}$ if it is the least (uniform in $\vec{p}$) fixed point of an $\mathbf{M}$-explicit operator, i.e.,

$$f(\vec{x},\vec{p}) = q(\vec{x}) \ \overline{\textbf{where}} \ \{q = \alpha_1(\vec{p},q)\}; \tag{5-20}$$

for partial functions this means simply that $f(\vec{x})$ is the least solution of the recursive equation

$$q(\vec{x}) = \alpha_1^*(\vec{x},q),$$

with an $\mathbf{M}$-explicit functional α_1^*—the functional associated with the operator α_1.

We say that f **is recursive in** $g_1,\dots,g_m$ **over** $\mathbf{M}$ if f is recursive in the expansion $(\mathbf{M},g_1,\dots,g_m)$, and we skip the reference to $\mathbf{M}$ if the claim holds for all functional algebras on the set M.

Distinguished points are defined for functional algebras as for partial algebras by Definition x5B.1, and the natural extension of Proposition 3A.15 is established by exactly the same proof:

5B.7. **Proposition.** *If a functional algebra* $\mathbf{M}$ *has two distinguished points* $0,1$*, then a functional* $f(\vec{x},\vec{p})$ *on* M *is* $\mathbf{M}$*-recursive if and only if there is a simple fixed point* $g(\vec{u},\vec{x},\vec{p})$ *of* $\mathbf{M}$ *such that for some* k *and all* $\vec{x},\vec{p}$*,*

$$f(\vec{x},\vec{p}) = g(\vec{0}^k,\vec{x},\vec{p}).$$

We leave for Problems x5B.6 and *x5B.6 the proof of this and the corresponding extension of Proposition 3A.17, which (in effect) allow us to assume that every functional algebra has two distinguished points.

5B.8. **Theorem.** *The set* $\mathbf{rec}_1(\mathbf{M})$ *of recursive functionals on a functional algebra* $\mathbf{M}$ *is explicitly closed, and also closed under the full* λ*-substitution scheme* (5B.4)*.*

PROOF is easy, using the rules satisfied by the $\overline{\textbf{where}}$ construct. We consider a few of the cases using small numbers of parts in the recursions to simplify notation.

(1) *The givens of* $\mathbf{M}$ *are* $\mathbf{M}$*-recursive*, because

$$f(\vec{x},\vec{p}) = f(\vec{x},\vec{p}) \ \overline{\textbf{where}} \ \{\ \}.$$

(Or, if the empty body in a recursion offends you, just put in some body which does not define any of the variables in $\vec{p}$.) In fact, all explicit functionals are recursive by similar, "dummy" recursions.

(2) *Closure under substitution*, $h(\vec{x},\vec{y},\vec{p},\vec{q}) = f(g(\vec{x},\vec{p}),\vec{y},\vec{q})$. We compute:

$$\begin{aligned} f(g(\vec{x},\vec{p}),\vec{y},\vec{q}) &= f(a,\vec{y},\vec{q})\ \overline{\textbf{where}}\ \{a = g(\vec{x},\vec{p})\} \quad \text{(by recap, (2) of 4C.6)}\\ &= f(a,\vec{y},\vec{q})\ \overline{\textbf{where}}\ \{a = \alpha_0(\vec{x},\vec{p},\vec{r})\ \overline{\textbf{where}}\ \{\vec{r} = \vec{\alpha}(\vec{x},\vec{p},\vec{r})\}\}\\ &= f(a,\vec{y},\vec{q})\ \overline{\textbf{where}}\ \{a = \alpha_0(\vec{x},\vec{p},\vec{r}), \vec{r} = \vec{\alpha}(\vec{x},\vec{p},\vec{r})\}, \end{aligned}$$

where the last transformation is justified by by the Bekič-Scott rule, Theorem 4C.4. Now by the hypothesis on f,

$$f(a,\vec{y},\vec{q}) = \beta_0(a,\vec{y},\vec{q},\vec{r}')\ \overline{\textbf{where}}\ \{\vec{r}' = \vec{\beta}(a,\vec{q},\vec{r}')\}$$

with **M**-explicit $\beta_0, \vec{\beta}$, and if we replace this expression in the last formula and appeal to the head rule (1) of Theorem 4C.6, we get

$$\begin{aligned} h(\vec{x},\vec{y},\vec{p},\vec{q}) = \beta_0(a,\vec{y},\vec{q},\vec{r}')\ &\overline{\textbf{where}}\\ &\{\vec{r}' = \vec{\beta}(a,\vec{y},\vec{q},\vec{r}'), a = \alpha_0(\vec{x},\vec{p},\vec{r}), \vec{r} = \vec{\alpha}(\vec{x},\vec{p},\vec{r})\} \end{aligned}$$

which witnesses the fact that $h(\vec{x},\vec{y},\vec{p},\vec{q})$ is **M**-recursive.

(3) and (4) (closure under branching and shuffling) are proved similarly and we will skip them, see Problem x5B.8.

(5) *Closure under λ-substitution into the givens*,

$$f(y,\vec{x},\vec{p}) = F(y,\lambda(s)\lambda(t)h(s,t,\vec{x},\vec{p}))$$

in the notationally simple cases we have been considering. We compute again, using now the λ-rule (3) of Theorem 4C.6, and skipping the $\vec{x},\vec{p}$ (which do not enter the computation) after the first line:

$$\begin{aligned} f(y,\vec{x},\vec{p}) &= F(y,\lambda(s)\lambda(t)h(s,t,\vec{x},\vec{p}))\\ &= F(y,r')\ \overline{\textbf{where}}\ \{r' = \lambda(s)\lambda(t)h(s,t)\}\\ &= F(y,r')\ \overline{\textbf{where}}\ \{r' = \lambda(s,t)[\gamma_0(s,t,\vec{q})\ \overline{\textbf{where}}\ \{\vec{q} = \vec{\gamma}(s,t,\vec{q})\}]\}\\ &= F(y,r')\ \overline{\textbf{where}}\ \{r' = \lambda(s,t)\gamma_0(s,t,\vec{r}(s,t))\\ &\qquad\qquad \overline{\textbf{where}}\ \{\vec{r} = \lambda(s,t)\vec{\gamma}(s,t,\vec{r}(s,t))\}\}\\ &= F(y,r')\\ &\qquad \overline{\textbf{where}}\ \{r' = \lambda(s,t)\gamma_0(s,t,\vec{r}(s,t)), \vec{r} = \lambda(s,t)\vec{\gamma}(s,t,\vec{r}(s,t))\} \end{aligned}$$

If we put in again the "side terms" $y,\vec{x},\vec{p}$, the conclusion is that

$$\begin{aligned} f(y,\vec{x},\vec{p}) &= F(y,r')\\ &\overline{\textbf{where}}\ \{r' = \lambda(s,t)\gamma_0(s,t,\vec{x},\vec{p},\vec{r}(s,t)), \vec{r} = \lambda(s,t)\vec{\gamma}(s,t,\vec{x},\vec{p},\vec{r}(s,t))\} \end{aligned}$$

which witnesses that $f(y,\vec{x},\vec{p})$ is **M**-recursive. ⊣

5B.9. **Theorem (The First Recursion Theorem).** *If $f(\vec{x},p,\vec{q})$ is* **M***-recursive where $\vec{x} = (x_1,\ldots,x_n)$ and p varies over $(M^n \rightharpoonup M)$, and if*

$$g(\vec{x},\vec{q}) = p(\vec{x})\ \overline{\textbf{where}}\ \{p = \lambda(\vec{x})f(\vec{x},p,\vec{q})\}$$

is the simple fixed point defined by $f(\vec{x}, p, \vec{q})$, then $g(\vec{x}, \vec{q})$ is **M***-recursive.*

PROOF. The argument for (2) of Proposition 4C.7 does not depend on the continuity of the given functionals—only their monotonicity—and so it gives this result too. ⊣

5B.10. **Corollary (Transitivity Theorem).** *For each* **M***-recursive functional $f(\vec{x}, \vec{p})$,*

$$\mathbf{rec}_1(\mathbf{M}, f) = \mathbf{rec}_1(\mathbf{M}).$$

PROOF. Every explicit functional of the expansion $(\mathbf{M}, f)$ is recursive in $\mathbf{M}$, by the closure properties of $\mathbf{rec}_1(\mathbf{M})$, and so every simple fixed point of the expansion is recursive in $\mathbf{M}$ by the Recursion Theorem; and then every $(\mathbf{M}, f)$-recursive functional is $\mathbf{M}$-recursive, since it is the section of a simple fixed point of $\mathbf{M}$ by Proposition 5B.7. ⊣

Problems for Section 5B

x5B.1. Prove Proposition 5B.4.

x5B.2. Prove that E is recursive in $E^{\#}$.

x5B.3. Prove that the functionals $p \vee q$ and $p \;\&\; q$ defined in (5-5) and (5-6) are recursive in $E^{\#}$.

x5B.4. Prove that for every set M, Q_M is recursive in $E_M, Q_M^{\#}$.

x5B.5 (**open**). Is Q_M always recursive in $Q_M^{\#}$?

x5B.6. Prove Proposition 5B.7.

5B.11. **Definition.** Suppose $\mathbf{M}_1$ and $\mathbf{M}_2$ are functional algebras of possibly different characteristics, but such that $M_1 \subseteq M_2$. We say that $\mathbf{M}_2$ is an **inessential extension** of $\mathbf{M}_1$, if a functional $f(\vec{x}, \vec{p})$ on M_1 is $\mathbf{M}_1$-recursive if and only if there exists some $\mathbf{M}_2$-recursive functional $g(\vec{x}, \vec{p})$ such that if $\vec{x} \in M_1^n$ and $p_i : M_1^{k_i} \rightharpoonup M_1 \cup \{\mathrm{tt}, \mathrm{ff}\}$ for $i = 1, \ldots, m$, then

$$f(\vec{x}, \vec{p}) = g(\vec{x}, \vec{p}).$$

(The equation makes sense, because if $p : M_1^k \rightharpoonup M_1 \cup \{\mathrm{tt}, \mathrm{ff}\}$, then also $p : M_2^k \rightharpoonup M_2 \cup \{\mathrm{tt}, \mathrm{ff}\}$.)

*x5B.7. Prove that every functional structure $\mathbf{M}$ has an inessential extension $\mathbf{M}[0,1]$ with two distinguished points.

x5B.8. Prove that if g, h_1, h_2 are $\mathbf{M}$-recursive and

$$f(\vec{x}, \vec{p}) = \text{if } g(\vec{x}, \vec{p}) \text{ then } h_1(\vec{x}, \vec{p}) \text{ else } h_2(\vec{x}, \vec{p}),$$

then $f(\vec{x}, \vec{p})$ is also $\mathbf{M}$-recursive.

5C. Stage comparison

We will establish here one of the basic results about functional recursion, which makes it possible to *combine in parallel* (*interweave*) recursive definitions, and to define *selection operators*. One of its elementary corollaries is that in *normal functional algebras* (which include all total, partial algebras), the class of **M**-semirecursive relations is closed under disjunction.

5C.1. **Definition (the stage comparison function).** Suppose $f(\vec{x}, p)$ is a monotone, operative functional on M, and let

$$\alpha_f(p) = \lambda(\vec{x}) f(\vec{x}, p)$$

be the associated operator. By the Fixed Point Theorem 4B.6, there is a sequence of partial functions

$$\{\overline{f}^{\xi} : M^n \rightharpoonup M_{\mathbb{B}} \mid \xi \in \text{Ordinals}\}$$

indexed by the ordinal numbers, such that

$$\eta \leq \xi \Longrightarrow \overline{f}^{\eta} \sqsubseteq \overline{f}^{\xi},$$

and for some κ, the partial function

$$\overline{f} = \bigcup_{\xi} \overline{f}^{\xi} = \bigcup_{\xi<\kappa} \overline{f}^{\xi}$$

is the least fixed point of α_f. As in Definition 3A.19 for the continuous case (but using f rather than α_f in the notation), we set

$$|\vec{x}|_f = \begin{cases} \text{the least } \xi \text{ such that } \overline{f}^{\xi}(\vec{x})\downarrow, & \text{if } \overline{f}(\vec{x})\downarrow, \\ \infty, & \text{otherwise.} \end{cases} \tag{5-21}$$

The ordinal $|\vec{x}|_f$ is a (crude) measure of the complexity of computing $\overline{f}(\vec{x})$ using the given recursive definition.

If $f(\vec{x}, p)$ and $g(\vec{y}, q)$ are two monotone, operative functionals on M, we set

$$\boldsymbol{\sigma}_{f,g}(\vec{x}, \vec{y}) = \begin{cases} \mathrm{tt}, & \text{if } \overline{f}(\vec{x})\downarrow \ \& \ |\vec{x}|_f \leq |\vec{y}|_g, \\ \mathrm{ff}, & \text{if } \overline{g}(\vec{y})\downarrow \ \& \ |\vec{y}|_g < |\vec{x}|_f, \\ \bot, & \text{otherwise.} \end{cases} \tag{5-22}$$

This is the **stage comparison** (partial) **function** of $f(\vec{x}, p)$ and $g(\vec{y}, q)$.

To see the significance of this function, suppose

$$P(\vec{x}) \iff \overline{f}(\vec{0}^k, \vec{x})\downarrow, \quad Q(\vec{x}) \iff \overline{g}(\vec{0}^l, \vec{x})\downarrow$$

are two typical **M**-semirecursive relations, with $\overline{f}$ and $\overline{g}$ **M**-simple fixed points of a functional algebra which admits two distinguished points; now

$$P(\vec{x}) \vee Q(\vec{x}) \iff \boldsymbol{\sigma}_{f,g}(\vec{0}^k, \vec{x}, \vec{0}^l, \vec{x}) = \mathrm{tt},$$

and so if $\boldsymbol{\sigma}_{f,g}$ is **M**-recursive, then the disjunction $P(\vec{x}) \vee Q(\vec{x})$ is **M**-semirecursive. Thus it is useful to investigate conditions under which the stage comparison function of any two **M**-explicit functionals is **M**-recursive. Key to this problem is the following notion:

5C.2. **Definition (Normality).** Suppose $f(\vec{x}, p, q)$ is a functional on M. A **normalizing functional** for f is any $\Delta(\vec{x}, p, \delta, q, \varepsilon)$ with δ, ε ranging over partial functions with arities (respectively) those of p and q, such that the following hold, with

$$Z_\delta = \{\vec{t} \mid \delta(\vec{t}) = \mathrm{tt}\}, \quad Z_\varepsilon = \{\vec{s} \mid \varepsilon(\vec{s}) = \mathrm{tt}\},$$

the *truthsets* of δ and ε:

(N1) If $f(\vec{x}, p \restriction Z_\delta, q \restriction Z_\varepsilon)\downarrow$, then $\Delta(\vec{x}, p, \delta, q, \varepsilon) = \mathrm{tt}$.

(N2) If δ and ε are total functions, $p(\vec{t})\downarrow$ for every $\vec{t} \in Z_\delta$, $q(\vec{s})\downarrow$ for every $\vec{s} \in Z_\varepsilon$, and $f(\vec{x}, p \restriction Z_\delta, q \restriction Z_\varepsilon)\uparrow$, then $\Delta(\vec{x}, p, \delta, q, \varepsilon) = \mathrm{ff}$.

The definition extends naturally to functionals $f(\vec{x}, \vec{p})$ with any number of partial function arguments, with Δ having twice as many partial function arguments as f, and it gives us a way to decide whether $f(\vec{x}, p)$ converges or not in certain (circumscribed) situations. At the other end, the only normalizing functional for a partial function $f : M^n \rightharpoonup M_{\mathbb{B}}$ is the characteristic function of the domain of convergence of f,

$$\Delta(\vec{x}) = \begin{cases} \mathrm{tt}, & \text{if } f(\vec{x})\downarrow, \\ \mathrm{ff}, & \text{otherwise.} \end{cases}$$

A functional $f(\vec{x}, \vec{p})$ is **normal in M** if it admits an **M**-recursive normalizing functional, and a functional algebra **M** is **normal** if all the givens of **M** are **M**-normal.

Thus a partial algebra **M** is normal if the domains of convergence of all the givens are **M**-recursive—including the trivial case when all the givens are total functions. We can think of normal functionals as the analog of partial functions which have a recursive domain of convergence, just as Kleene objects correspond to total functions.

For the simplest example of a normal honest functional, consider an evaluation

$$\mathrm{ev}(\vec{x}, p) = p(\vec{x}).$$

It is quite clear that there is no way to decide in general whether $\mathrm{ev}(\vec{x}, p)\downarrow$, but it is normalized by the trivial

$$\Delta(\vec{x}, p, \delta) = \dot{\neg}\dot{\neg}\delta(\vec{x}) = \begin{cases} \mathrm{tt}, & \text{if } \delta(\vec{x}) = \mathrm{tt}, \\ \mathrm{ff}, & \text{if } \delta(\vec{x})\downarrow \ \&\neq \mathrm{tt}; \end{cases}$$

because if $\mathrm{ev}(\vec{x}, p\restriction Z_\delta) = (p\restriction Z_\delta)(\vec{x})\downarrow$, then $\delta(\vec{x}) = \mathrm{tt}$, and if δ is total and the domain of convergence of p includes Z_δ, then

$$(p\restriction Z_\delta)(\vec{x})\uparrow \Longrightarrow \delta(\vec{x}) \neq \mathrm{tt} \Longrightarrow \Delta(\vec{x}, p, \delta) = \ddot{\neg}\delta(\vec{x}) = \mathrm{ff}.$$

5C.3. **Theorem.** *If* $\mathbf{M}$ *is a normal functional algebra, then every* $\mathbf{M}$*-explicit functional is* $\mathbf{M}$*-normal.*

PROOF. It is enough to show that the set $\mathcal{F}$ of all $\mathbf{M}$-explicit, $\mathbf{M}$-normal functionals on M is explicitly closed over $\mathbf{M}$, and we have already shown that $\mathcal{F}$ contains all total functions on M and the evaluation functionals. Thus $\mathcal{F}$ satisfies (1) of Definition 3A.3

(2), *Substitution.* In a notationally simple example, suppose that

$$f(\vec{x}, \vec{y}, p, q) = g(h(\vec{x}, p), \vec{y}, q)$$

and $\Delta_h(\vec{x}, p, \delta), \Delta_g(s, \vec{y}, q, \varepsilon)$ are recursive normalizing functionals for g and h. Let

$$\begin{aligned}(5\text{-}23)\quad &\Delta_f(\vec{x}, \vec{y}, p, \delta, q, \varepsilon)\\ &\quad = \text{if } \ddot{\neg}\Delta_h(\vec{x}, p, \delta) \text{ then ff else } \Delta_g(h(\vec{x}, p\restriction Z_\delta), \vec{y}, q, \varepsilon).\end{aligned}$$

This is clearly recursive. To prove that it normalizes $f(\vec{x}, \vec{y}, p, q)$, we verify the two required properties (N1) and (N2) in Definition 5C.2.

(N1) *If* $f(\vec{x}, \vec{y}, p\restriction Z_\delta, q\restriction Z_\varepsilon)\downarrow$*, then* $\Delta_f(\vec{x}, \vec{y}, p, \delta, q, \varepsilon) = \mathrm{tt}$.

Proof. From the hypothesis, there is some $s \in M$ such that

$$h(\vec{x}, p\restriction Z_\delta) = s \text{ and } g(s, \vec{y}, q\restriction Z_\varepsilon)\downarrow,$$

and so,

$$\Delta_h(\vec{x}, p, \delta) = \mathrm{tt}, \quad \Delta_g(s, \vec{y}, q, \varepsilon) = \mathrm{tt},$$

and (5-23) yields immediately the required $\Delta_f(\vec{x}, \vec{y}, p, \delta, q, \varepsilon) = \mathrm{tt}$.

(N2) *If* δ *and* ε *are total functions,* $p(\vec{s})\downarrow$ *for every* $\vec{s} \in Z_\delta$, $q(\vec{t})\downarrow$ *for every* $\vec{t} \in Z_\varepsilon$*, and* $f(\vec{x}, \vec{y}, p\restriction Z_\delta, q\restriction Z_\varepsilon)\uparrow$*, then* $\Delta_f(\vec{x}, \vec{y}, p, \delta, q, \varepsilon) = \mathrm{ff}$.

Proof. We assume the hypothesis for some specific $\vec{x}, \vec{y}, p, \delta, q, \varepsilon$, and we take cases on the two ways in which $f(\vec{x}, \vec{y}, p\restriction Z_\delta, q\restriction Z_\varepsilon)$ may diverge.

(N2a) $h(\vec{x}, p\restriction Z_\delta)\uparrow$. Now $\Delta_h(\vec{x}, p, \delta) = \mathrm{ff}$, and so $\Delta_f(\vec{x}, \vec{y}, p, \delta) = \mathrm{ff}$.

(N2b) $h(\vec{x}, p\restriction Z_\delta) = s$ for some s, but $g(s, \vec{y}, q\restriction Z_\varepsilon)\uparrow$. Now the hypothesis on h gives $\Delta_h(\vec{x}, p, \delta) = \mathrm{tt}$, and that on g gives $\Delta_g(s, q, \varepsilon) = \mathrm{ff}$, so that by the definition of Δ_f,

$$\Delta_f(\vec{x}, \vec{y}, p, \delta, q, \varepsilon) = \Delta_g(s, q, \varepsilon) = \mathrm{ff}.$$

The arguments for branching (3) and shuffling (4) are similar and we skip them (see Problem x5C.1).

(5), *λ-substitution in the givens.* We show that (in general) the scheme of λ-substitution (3-16) preserves normality. So suppose (in a notationally simple case) that

$$f(\vec{x},\vec{y},p,q) = g(\vec{x},\lambda(t)h(t,\vec{y},q),p),$$

and assume that we are given recursive, normalizing functionals

$$\Delta_h(t,\vec{y},q,\varepsilon) \text{ and } \Delta_g(\vec{x},r,\zeta,p,\delta)$$

for h and g. We set

(5-24) $\Delta_f(\vec{x},\vec{y},p,\delta,q,\varepsilon) = \Delta_g(\vec{x},\lambda(t)h(t,\vec{y},q\restriction Z_\varepsilon),\lambda(t)\Delta_h(t,\vec{y},q,\varepsilon),p,\delta).$

For fixed $\vec{x},\vec{y},p,\delta,q,\varepsilon$, let

$$r = \lambda(t)h(t,\vec{y},q\restriction Z_\varepsilon), \quad \zeta = \lambda(t)\Delta_h(t,\vec{y},q,\varepsilon)$$

and notice that by the hypothesis on Δ_h, for any t,

$$r(t)\downarrow \Longrightarrow \zeta(t) = \mathrm{tt}, \text{ and so } r = r\restriction Z_\zeta.$$

Hence,

(5-25) $$\begin{aligned} f(\vec{x},\vec{y},p\restriction Z_\delta,q\restriction Z_\varepsilon) &= g(\vec{x},\lambda(t)h(t,y,q\restriction Z_\varepsilon),p\restriction Z_\delta) \\ &= g(\vec{x},r,p\restriction Z_\delta) = g(\vec{x},r\restriction Z_\zeta,p\restriction Z_\delta) \end{aligned}$$

and

(5-26) $$\Delta_f(\vec{x},\vec{y},p,\delta,q,\varepsilon) = \Delta_g(\vec{x},r,\zeta,p,\delta).$$

(a) Suppose $f(\vec{x},\vec{y},p\restriction Z_\delta,q\restriction Z_\varepsilon)\downarrow$; now (5-25) and (5-26) yield immediately that $\Delta_f(\vec{x},\vec{y},p,\delta,q,\varepsilon) = \mathrm{tt}$.

(b) Suppose $f(\vec{x},\vec{y},p\restriction Z_\delta,q\restriction Z_\varepsilon)\uparrow$, δ and ε are total functions, $p(t)\downarrow$ for each $t\in Z_\delta$ and $q(s)\downarrow$ for each $s\in Z_\varepsilon$. By (5-25) and (5-26),

$$g(\vec{x},r\restriction Z_\zeta,p\restriction Z_\delta)\uparrow, \quad \Delta_f(\vec{x},\vec{y},p,\delta,q,\varepsilon) = \Delta_g(\vec{x},r,\zeta,p,\delta),$$

and so to infer that $\Delta_f(\vec{x},\vec{y},p,\delta,q,\varepsilon) = \mathrm{ff}$ it is enough to verify from the assumptions that ζ is total and $r(t)\downarrow$ for every $t\in Z_\zeta$—but these conditions follow immediately from the definitions of r and ζ. ⊣

5C.4. **Theorem (Stage Comparison Theorem).** *If* **M** *is a normal functional structure, then the stage comparison function* $\boldsymbol{\sigma}_{f,g}(\vec{x},\vec{y}$ *of any two* **M***-explicit operative functionals* $f(\vec{x},p)$ *and* $g(\vec{y},q)$ *is* **M***-recursive.*

PROOF. We fix two explicit, operative functionals $f(\vec{x},p), g(\vec{y},q)$ and recursive functionals $\Delta_f(\vec{x},p,\delta), \Delta_g(\vec{y},q,\varepsilon)$ which normalize them, and we let

$$\boldsymbol{\sigma}(\vec{x},\vec{y}) = \boldsymbol{\sigma}_{f,g}(\vec{x},\vec{y})$$

be the stage comparison function for f and g defined in (5-22). We will also skip the subscripts in the notation for stages

$$|\vec{x}| = |\vec{x}|_f, \quad |\vec{y}| = |\vec{y}|_g,$$

letting the specific letters $\vec{x}, \vec{y}$ specify the relevant functionals. Finally, set

$$\overline{f}^{<\xi}(\vec{x}) = w \iff (\exists \eta < \xi)[\overline{f}^{\eta}(\vec{x}) = w], \tag{5-27}$$

and similarly for $\overline{g}^{<\xi}(\vec{y})$. We also understand (5-27) for $\xi = \infty$, in which case

$$\overline{f}^{<\infty}(\vec{x}) = \overline{f}^{\infty}(\vec{x}) = \overline{f}(\vec{x}).$$

To discover the recursive equation which should be satisfied by $\boldsymbol{\sigma}(\vec{x}, \vec{y})$, assume $\overline{f}(\vec{x})\downarrow \vee \overline{g}(\vec{y})\downarrow$ and compute:

$$\begin{aligned} \boldsymbol{\sigma}(\vec{x}, \vec{y}) = \mathrm{tt} &\iff \overline{f}^{|\vec{y}|}(\vec{x})\downarrow \\ &\iff f(\vec{x}, \overline{f}^{<|\vec{y}|})\downarrow \\ &\iff f(\vec{x}, \overline{f}\restriction\{\vec{x}' \mid |\vec{x}'| < |\vec{y}|\})\downarrow. \end{aligned}$$

Now,

$$|\vec{x}'| < |\vec{y}| \iff |\vec{x}'| < \infty \ \&\ \overline{g}^{|\vec{x}'|}(\vec{y})\uparrow \iff |\vec{x}'| < \infty \ \&\ g(\vec{y}, \overline{g}^{<|\vec{x}'|})\uparrow;$$

this implies that

$$\overline{f}\restriction\{\vec{x}' \mid |\vec{x}'| < |\vec{y}|\})\downarrow = \overline{f}\restriction\{\vec{x}' \mid g(\vec{y}, \overline{g}^{<|\vec{x}'|})\uparrow\},$$

because we only need consider the values of these two partial functions in the domain of $\overline{f}$, i.e., for $\vec{x}'$ such that $|\vec{x}'| < \infty$. Assuming that the correct conditions can be verified, so that the normalizing functionals compute correctly convergence and non-convergence, this yields

$$\begin{aligned} \boldsymbol{\sigma}(\vec{x}, \vec{y}) = \mathrm{tt} &\iff f(\vec{x}, \overline{f}\restriction\{\vec{x}' \mid g(\vec{y}, \overline{g}\restriction\{\vec{y}' \mid |\vec{y}'| < |\vec{x}'|\})\uparrow\})\downarrow \\ &\iff \Delta_f(\vec{x}, \overline{f}, \lambda(\vec{x}')\dot{\neg}\Delta_g(\vec{y}, \overline{g}, \lambda(\vec{y}')\dot{\neg}\boldsymbol{\sigma}(\vec{x}', \vec{y}'))) = \mathrm{tt}, \end{aligned}$$

and a similar equivalence for $\boldsymbol{\sigma}(\vec{x}, \vec{y}) = \mathrm{ff}$. Thus, assuming again that the correct conditions for the application of the normalizing functinals can be verified,

$$\boldsymbol{\sigma}(\vec{x}, \vec{y})\downarrow \Longrightarrow \boldsymbol{\sigma}(\vec{x}, \vec{y}) = \Delta_f(\vec{x}, \overline{f}, \lambda(\vec{x}')\dot{\neg}\Delta_g(\vec{y}, \overline{g}, \lambda(\vec{y}')\dot{\neg}\boldsymbol{\sigma}(\vec{x}', \vec{y}'))),$$

so that $\boldsymbol{\sigma} \sqsubseteq \overline{r}$ with $\overline{r}$ the least solution of the recursive equation

$$r(\vec{x}, \vec{y}) = \Delta_f(\vec{x}, \overline{f}, \lambda(\vec{x}')\dot{\neg}\Delta_g(\vec{y}, \overline{g}, \lambda(\vec{y}')\dot{\neg}r(\vec{x}', \vec{y}'))). \tag{5-28}$$

This is an explicit equation in the expansion $(\mathbf{M}, \overline{f}, \overline{g}, \Delta_f, \Delta_g)$, so that $\overline{r}$ is $\mathbf{M}$-recursive by the transitivity property, Corollary 2B.6.

We now prove by ordinal induction on ξ, that if $\overline{r}(\vec{x},\vec{y})$ is the least solution of (5-28), then

(5-29) $$\text{if } \min(|\vec{x}|,|\vec{y}|)=\xi, \text{ then } \overline{r}(\vec{x},\vec{y})=\boldsymbol{\sigma}(\vec{x},\vec{y}).$$

This suffices to show that $\boldsymbol{\sigma}(\vec{x},\vec{y})$ is **M**-recursive, since it implies that

$$\begin{aligned}\boldsymbol{\sigma}(\vec{x},\vec{y}) = \text{if } \overline{r}(\vec{x},\vec{y}) \text{ then}&\\ &\text{if } \overline{f}(\vec{x}) \text{ then tt else tt}\\ &\text{else if } \overline{g}(\vec{y}) \text{ then ff else ff.}\end{aligned}$$

Lemma A. If $|\vec{x}|=\xi\le|\vec{y}|$, *then* $\overline{r}(\vec{x},\vec{y})=\text{tt}$.

PROOF. Fix $\vec{x}'$ with $|\vec{x}'|<\xi$. By the induction hypothesis, for all $\vec{y}'$,

$$\overline{r}(\vec{x}',\vec{y}')=\begin{cases}\text{tt}, & \text{if } |\vec{x}'|\le|\vec{y}'|,\\ \text{ff}, & \text{otherwise, i.e., if } |\vec{y}'|<|\vec{x}'|.\end{cases}$$

In particular,

$$\delta=\lambda(\vec{y}')\dot{\neg}\overline{r}(\vec{x}',\vec{y}')$$

is a total function, and

$$\vec{y}'\in Z_\delta\Longrightarrow|\vec{y}'|<|\vec{x}'|<\infty\Longrightarrow\overline{g}(\vec{y}')\downarrow;$$

but $g(\vec{y},\overline{g}\restriction\{\vec{y}'\mid|\vec{y}'|<|\vec{x}'|\})\uparrow$, since the opposite implies that $|\vec{y}|\le|\vec{x}'|<\xi$ contradicting the hypothesis of the Lemma. It follows that

$$\Delta_g(\vec{y},\overline{g},\lambda(\vec{y}')\dot{\neg}\overline{r}(\vec{x}',\vec{y}'))=\text{ff},$$

and so (taking "complements"), we have shown that

$$|\vec{x}'|<\xi\Longrightarrow\dot{\neg}\Delta_g(\vec{y},\overline{g},\lambda(\vec{y}')\dot{\neg}\overline{r}(\vec{x}',\vec{y}'))=\text{tt}.$$

Now the hypothesis $|\vec{x}|=\xi$ of the Lemma also implies that

$$f(\vec{x},\overline{f}\restriction\{\vec{x}'\mid|\vec{x}'|<\xi\})\downarrow,$$

and so by monotonicity,

$$f(\vec{x},\overline{f}\restriction\{\vec{x}'\mid\dot{\neg}\Delta_g(\vec{y},\overline{g},\lambda(\vec{y}')\dot{\neg}\overline{r}(\vec{x}',\vec{y}'))=\text{tt}\})\downarrow,$$

and then by the normalizing property (N1) for Δ_f,

$$\overline{r}(\vec{x},\vec{y})=\Delta_f(\vec{x},\overline{f},\lambda(\vec{x}')\dot{\neg}\Delta_g(\vec{y},\overline{g},\lambda(\vec{y}')\dot{\neg}\overline{r}(\vec{x}',\vec{y}')))=\text{tt},$$

which completes the proof. ⊣ (Lemma A)

Lemma B. If $|\vec{y}|=\xi<|\vec{x}|$, *then the partial function*

(5-30) $$\delta=\lambda(\vec{x}')\dot{\neg}\Delta_g(\vec{y},\overline{g},\lambda(\vec{y}')\dot{\neg}\overline{r}(\vec{x}',\vec{y}'))$$

it total.

PROOF. To show that $\delta(\vec{x}')\downarrow$ for all $\vec{x}'$, we consider two cases.

Case B1. $|\vec{x}'| < \xi$. By the induction hypothesis, for all $\vec{y}'$,

$$\overline{r}(\vec{x}', \vec{y}') = \begin{cases} \mathrm{tt}, & \text{if } |\vec{x}'| \leq |\vec{y}'|, \\ \mathrm{ff}, & \text{otherwise, i.e., if } |\vec{y}'| < |\vec{x}'|. \end{cases}$$

Thus $\overline{g}(\vec{y}')\downarrow$ when $\dot{\neg}\overline{r}(\vec{x}', \vec{y}') = \mathrm{tt}$, since for those $\vec{y}'$'s, $|\vec{y}'| < |\vec{x}'| < \xi$. It follows that

$$g(\vec{y}, \overline{g}, \lambda(\vec{y}')\dot{\neg}\overline{r}(\vec{x}', \vec{y}'))\uparrow,$$

since the opposite (with monotonicity) would give $|\vec{y}| \leq |\vec{x}'| < \xi$, and so

$$\delta(\vec{x}') = \Delta_g(\vec{y}, \overline{g}, \lambda(\vec{y}')\dot{\neg}\overline{r}(\vec{x}', \vec{y}')) = \mathrm{ff},$$

and, in particular, $\delta(\vec{x}')\downarrow$.

Let us also record for later the specific value here,

$$|\vec{x}'| < \xi \Longrightarrow \dot{\neg}\Delta_g(\vec{y}, \overline{g}, \lambda(\vec{y}')\dot{\neg}\overline{r}(\vec{x}', \vec{y}')) = \mathrm{tt}. \tag{5-31}$$

Case B2. $\xi \leq |\vec{x}'|$, including the possibility that $|\vec{x}'| = \infty$. Now $\overline{r}(\vec{x}', \vec{y}')\downarrow$ for all $\vec{y}'$ with $|\vec{y}'| < \xi$, by the induction hypothesis, and it gives the correct value, so that

$$|\vec{y}'| < \xi \Longrightarrow \dot{\neg}\overline{r}(\vec{x}', \vec{y}') = \mathrm{tt}.$$

Since $|\vec{y}| = \xi$ and so $g(\vec{y}, \overline{g}\restriction\{\vec{y}' \mid |\vec{y}'| < \xi\})\downarrow$ it follows that

$$\delta(\vec{x}') = \Delta_g(\vec{y}, \overline{g}, \lambda(\vec{y}')\dot{\neg}\overline{r}(\vec{x}', \vec{y}')) = \mathrm{tt},$$

and, in particular, $\delta(\vec{x}')\downarrow$. ⊣ (Lemma B)

Lemma C. If $\delta(\vec{x}') = \mathrm{tt}$ with δ defined in (5-30)*, then $\overline{f}(\vec{x}')\downarrow$.*

PROOF. In fact we will show the stronger implication,

$$\dot{\neg}\Delta_g(\vec{y}, \overline{g}, \lambda(\vec{y}')\dot{\neg}\overline{r}(\vec{x}', \vec{y}')) = \mathrm{tt} \Longrightarrow |\vec{x}'| < \xi. \tag{5-32}$$

So assume that

$$\Delta_g(\vec{y}, \overline{g}, \lambda(\vec{y}')\dot{\neg}\overline{r}(\vec{x}', \vec{y}')) = \mathrm{ff},$$

and suppose, towards a contradiction, that $\xi \leq |\vec{x}'|$, allowing for $|\vec{x}'| = \infty$. Now, for each $\vec{y}'$ with $|\vec{y}'| < \xi$, by the induction hypothesis

$$\overline{r}(\vec{x}', \vec{y}') = \mathrm{ff}, \text{ and hence } \dot{\neg}\overline{r}(\vec{x}', \vec{y}') = \mathrm{tt},$$

$\overline{g}(\vec{y}')\downarrow$, and $g(\vec{y}, \overline{g}\restriction\{\vec{y}' \mid \dot{\neg}\overline{r}(\vec{x}', \vec{y}') = \mathrm{tt}\})\downarrow$; but this implies that

$$\Delta_g(\vec{y}, \overline{g}, \lambda(\vec{y}')\dot{\neg}\overline{r}(\vec{x}', \vec{y}')) = \mathrm{tt},$$

which contradicts the hypothesis of (5-32) and establishes it. ⊣ (Lemma C)

To complete now the proof of (5-29) (and the theorem), we note that

$$f(\vec{x}, \overline{f} \restriction \{\vec{x}' \mid \delta(\vec{x}') = \mathrm{tt}\})\uparrow$$

(because $|\vec{x}| > \xi$), and since the necessary conditions on δ and $\overline{f}$ have been established by Lemmas B and C,

$$\overline{r}(\vec{x}, \vec{y}) = \Delta_f(\vec{x}, \overline{f}, \lambda(\vec{x}')\dot{\neg}\Delta_g(\vec{y}, \overline{g}, \lambda(\vec{y}')\dot{\neg}\overline{r}(\vec{x}', \vec{y}')) = \mathrm{ff}$$

as required. ⊣

In the rest of the section we will prove a sequence of Propositions which insure that this basic result applies in a strong way to all the examples listed in Section 5A—except for search computability.

5C.5. **Proposition.** *If the Kleene existential quantifier $E = E_M$ defined in* (5-3) *is* **M***-recursive, then every Kleene object on M is normal in* **M**.

PROOF. Assuming for simplicity a Kleene object $F(p)$ which takes just one, unary partial function argument, let

$$\Delta(p, \delta) = \dot{\neg}E(\lambda(t)[\text{if } \delta(t) \text{ then test}(\mathrm{ff}, p, t) \text{ else tt}]),$$

where the (explicit) functional

$$\text{test}(x, t, p) = \text{if } p(t) \text{ then } x \text{ else } x$$

converges exactly when $p(t)\downarrow$ and then it gives x as output. For given p and δ, let

$$r = \lambda(t)[\text{if } \delta(t) \text{ then test}(\mathrm{ff}, p, t) \text{ else tt}]).$$

If $F(p \restriction Z_\delta)\downarrow$, then $p \restriction Z_\delta$ is total, so that $\delta(t) = \mathrm{tt}$ and $p(t)\downarrow$ for every t, hence $r(t) = \mathrm{ff}$ for every t, and hence

$$\Delta(p, \delta) = \dot{\neg}E(r) = \dot{\neg}\mathrm{ff} = \mathrm{tt}.$$

If δ is total and $p(t)\downarrow$ for every $t \in Z_\delta$, then r is a total function and so $E(r)\downarrow$; now $F(p \restriction Z_\delta)\uparrow$ can only happen when there is some t such that $\ddot{\neg}\delta(t) = \mathrm{ff}$, which gives $r(t) = \mathrm{tt}$ for that t, and hence

$$\Delta(p, \delta) = \dot{\neg}E(r) = \dot{\neg}\mathrm{tt} = \mathrm{ff}.$$ ⊣

5C.6. **Proposition.** *For every quantifier* Q *on M, if the sharp object $Q^\#$ is recursive in* **M**, *then $Q^\#$ is normal in* **M**.

PROOF. We set !

$$\Delta_Q(p, \delta) = \begin{cases} \mathrm{tt}, & \text{if } Q^\#(\lambda(\vec{x})[\text{if } \delta(\vec{x}) \text{ then } p(\vec{x}) \text{ else tt}]) = \mathrm{tt}, \\ \mathrm{tt}, & \text{ow., if } Q^\#(\lambda(\vec{x})[\text{if } \delta(\vec{x}) \text{ then } p(\vec{x}) \text{ else ff}]) = \mathrm{ff}, \\ \mathrm{ff}, & \text{otherwise.} \end{cases}$$

This is recursive in **M**.

Lemma A. *If $Q^\#(p \restriction Z_\delta) = \mathrm{tt}$, then $\Delta_Q(p, \delta) = \mathrm{tt}$.*

Proof. The hypothesis means that $(\mathsf{Q}\vec{x})[\ddot{\neg}(p\restriction Z_\delta)(\vec{x}) = \mathtt{tt}]$, so that

$$Q^{\#}(\lambda(\vec{x})[\text{if } \delta(\vec{x}) \text{ then } \ddot{\neg}p(\vec{x}) \text{ else } \bot]) = \mathtt{tt};$$

but

$$\lambda(\vec{x})[\text{if } \delta(\vec{x}) \text{ then } \ddot{\neg}p(\vec{x}) \text{ else } \bot] \sqsubseteq \lambda(\vec{x})[\text{if } \delta(\vec{x}) \text{ then } \ddot{\neg}p(\vec{x}) \text{ else } \mathtt{tt}],$$

and so (by monotonicity) $Q^{\#}(\lambda(\vec{x})[\text{if } \delta(\vec{x}) \text{ then } \ddot{\neg}p(\vec{x}) \text{ else } \mathtt{tt}]) = \mathtt{tt}$ and by the definition, $\Delta_Q(p,\delta) = \mathtt{tt}$. ⊣ (Lemma A)

Lemma B. *If* $Q^{\#}(p\restriction Z_\delta) = \mathtt{ff}$, *then* $\Delta_Q(p,\delta) = \mathtt{tt}$.

Proof. The hypothesis means that $(\breve{\mathsf{Q}}\vec{x})[\ddot{\neg}(p\restriction Z_\delta)(\vec{x}) = \mathtt{ff}]$, so that

$$Q^{\#}(\lambda(\vec{x})[\text{if } \delta(\vec{x}) \text{ then } \ddot{\neg}p(\vec{x}) \text{ else } \bot]) = \mathtt{ff};$$

but

$$\lambda(\vec{x})[\text{if } \delta(\vec{x}) \text{ then } \ddot{\neg}p(\vec{x}) \text{ else } \bot] \sqsubseteq \lambda(\vec{x})[\text{if } \delta(\vec{x}) \text{ then } \ddot{\neg}p(\vec{x}) \text{ else } \mathtt{tt}],$$

and so (by monotonicity) $Q^{\#}(\lambda(\vec{x})[\text{if } \delta(\vec{x}) \text{ then } \ddot{\neg}p(\vec{x}) \text{ else } \mathtt{tt}]) = \mathtt{ff}$ and the definition of $\Delta_Q(p,\delta)$ falls into the second case. Also,

$$\lambda(\vec{x})[\text{if } \delta(\vec{x}) \text{ then } \ddot{\neg}p(\vec{x}) \text{ else } \bot] \sqsubseteq \lambda(\vec{x})[\text{if } \delta(\vec{x}) \text{ then } \ddot{\neg}p(\vec{x}) \text{ else } \mathtt{ff}],$$

and so (by monotonicity) $Q^{\#}(\lambda(\vec{x})[\text{if } \delta(\vec{x}) \text{ then } \ddot{\neg}p(\vec{x}) \text{ else } \mathtt{ff}]) = \mathtt{ff}$ and by the definition in the second case, $\Delta_Q(p,\delta) = \mathtt{tt}$. ⊣ (Lemma B)

Lemma C. *If* δ *is total, and* $\delta(\vec{x}) = \mathtt{tt} \Longrightarrow p(\vec{x})\downarrow$, *and* $Q^{\#}(p\restriction Z_\delta)\uparrow$, *then* $\Delta_Q(p,\delta) = \mathtt{ff}$.

Proof. In this case, $Q^{\#}$ is applied to total functions in the definition of $\Delta_Q(p,\delta)$, and so it converges on these values and $\Delta_Q(p,\delta)\downarrow$; thus it is enough to show that neither of the first two cases in its definition is satisfied, so that $\Delta_Q(p,\delta) = \mathtt{ff}$.

First, we cannot have $Q^{\#}(\lambda(\vec{x})[\text{if } \delta(\vec{x}) \text{ then } p(\vec{x}) \text{ else } \mathtt{tt}]) = \mathtt{tt}$, because that implies that $(\mathsf{Q}\vec{x})[\delta(\vec{x}) = \mathtt{tt} \;\&\; p(\vec{x}) = \mathtt{tt}]$, so that $Q^{\#}(p\restriction Z_\delta) = \mathtt{tt}$, contrary to hypothesis.

Similarly, we cannot have $Q^{\#}(\lambda(\vec{x})[\text{if } \delta(\vec{x}) \text{ then } p(\vec{x}) \text{ else } \mathtt{ff}]) = \mathtt{ff}$, because this implies that $(\breve{\mathsf{Q}}(\vec{x})[\delta(\vec{x}) = \mathtt{tt} \;\&\; \ddot{\neg}p(\vec{x}) = \mathtt{ff}]$, so that $Q^{\#}(p\restriction Z_\delta) = \mathtt{ff}$, again contrary to hypothesis. ⊣ (Lemma C)

Now the Proposition follows immediately from these Lemmas. ⊣

Problems for Section 5C

x5C.1. Prove that of g, h_1, h_2 are normal in $\mathbf{M}$, then so is the functional

$$f(\vec{x},\vec{p}) = \text{if } g(\vec{x},\vec{p}) \text{ then } h_1(\vec{x},\vec{p}) \text{ else } h_2(\vec{x},\vec{p}).$$

*x5C.2. Prove that if $\mathbf{M}$ is normal, then the inessential extension $\mathbf{M}[0,1]$ by two distinguished points is also normal.

x5C.3. Prove that if $\mathbf{M}$ is normal, then the disjunction of any two $\mathbf{M}$-semirecursive relations is $\mathbf{M}$-recursive, and a relation $R(\vec{x})$ is $\mathbf{M}$-recursive if and only if it is semirecursive with semirecursive complement.

5D. The master recursion for functional algebras

The Kleene Master Recursion for functional algebras which imbed $\mathbf{N}_0$ is defined by a simple extension of the construction in Section 3C, and the proof of that *every* $\mathbf{M}$*-recursive partial function is Kleene-recursive* is a very minor modification of the proof given for partial algebras in Section 3C. The converse, however, (apparently) cannot be established in general for functional algebras which imbed $\mathbf{N}_0$, because the computation of a recursive function $f(\vec{x})$ does not necessarily take place in the subalgebra generated by the arguments $\vec{x}$: for that direction, we will need to assume that $\mathbf{M}$ *admits a recursive pair*.

Recall the definition 3C.1 of a **computation theory K** over a pair $(M, \mathbf{N}_0)$ with $\mathbb{N} \subseteq M$, and that of **strongly K-effective functional**, 3C.7.

We state in full the extended version of Theorem 3C.6 that we need for functional algebras.

5D.1. **Theorem (The Master Recursion for functional algebras).** *Suppose* $\mathbf{M} = (f_1, \dots, f_L)$ *is a functional algebra and* $\mathbf{N}_0 = (\mathbb{N}, 0, S, Pd, =_0)$ *is a copy of the natural numbers with* $\mathbb{N} \subseteq M$.

There is a monotone functional !

$$\Phi : \mathbb{N} \times M^* \times (\mathbb{N} \times M^* \rightharpoonup M_{\mathbb{B}}) \rightharpoonup M_{\mathbb{B}}, \tag{5-33}$$

such that if $\overline{p}$ *is the least solution of the recursive equation*

$$p(z, \vec{x}) = \Phi(z, \vec{x}, p) \quad (z \in \mathbb{N}, \vec{x} \in M^*)$$

and we set

$$K(z, \vec{x}) = \{z\}(\vec{x}) = \overline{p}(z, \vec{x}), \tag{5-34}$$

$$|z, \vec{x}| = |z, \vec{x}|_{\Phi}, \quad (\{z\}(\vec{x})\downarrow), \tag{5-35}$$

then the pair

$$\mathbf{K}[\mathbf{M}, \mathbf{N}_0] = (K, |\ |) \tag{5-36}$$

is a computation theory over $(M, \mathbf{N}_0)$ *such that the given functionals* $f_1, \dots, f_L$ *are strongly* $\mathbf{K}$*-effective.*

By definition 3C.7, trivially, a partial function $f : M^n \rightharpoonup M_{\mathbb{B}}$ is strongly **K**-effective if and only if it is **K**-computable, and so this theorem is a direct generalization of Theorem 3C.6.

PROOF is an extension of the proof of Theorem 3C.6, with a small change in *Case* (**KL1**) which subtracts the introduction of the givens ($(z)_1 = 7$) and the addition of one more case in which the givens are introduced, as follows.

Case **(KL8)**, introduction of the givens. If $(z)_0 = 8$, $(z)_1 = i$ $(1 \leq i \leq L)$ and $f_i(y, r)$ is one of the givens with (for example) r ranging over binary partial function variables, then

$$\Phi_8(z, \vec{w}, p) = f_i(w_2, \lambda(s)\lambda(t)p(w_1, s, t, w_3, \dots, w_l)),$$

provided that the length l of $\vec{w} = (w_1, \dots, w_l)$ is at least 2. (The definition is adjusted in the obvious way for given functionals f_i or different arities.)

The proof that $\mathbf{K}[\mathbf{M}, \mathbf{N}_0]$ is a computation theory is exactly as before, and so we only need verify that the new *Case* **(KL8)** insures the strong **K**-effectiveness of all the givens. First, because $\{z\}(\vec{w}) = \overline{p}(z, \vec{w})$ is a fixed point of Φ, if $f_i(y, r)$ is a given with r ranging over binary partial functions, we have the equation

$$\{\langle 8, i\rangle\}(e, y, \vec{x}) = f_i(y, \lambda(s)\lambda(t)\{e\}(s, t, \vec{x})),$$

and so $f_i(y, r)$ is **K**-effective. Moreover, taking $\tilde{f} = \langle 8, i\rangle$ in Definition 3C.7 and fixing $e, y, \vec{x}$, suppose that

$$\{\langle 8, i\rangle\}(e, y, \vec{x}) = w \text{ and } |\langle 8, i\rangle, e, y, \vec{x}| = \xi.$$

Let $\overline{r}^{\xi} : M^2 \rightharpoonup M_{\mathbb{B}}$ be the part of $\lambda(s)\lambda(t)\{e\}(s, t, \vec{x})$ computed before stage ξ, i.e.,

$$\overline{r}^{\xi}(s, t) = \begin{cases} \{e\}(s, t, \vec{x}), & \text{if } |e, s, t, \vec{x}| < \xi, \\ \perp, & \text{otherwise.} \end{cases}$$

It follows from the assignment of stages to the least fixed point of a monotone functional that

$$\{\langle 8, i\rangle\}(e, y, \vec{x}) = f_i(y, \overline{r}^{\xi}),$$

which is exactly the condition required of $\langle 8, i\rangle$ to verify that $f_i(y, r)$ is strongly **K**-effective. ⊣

5D.2. **Lemma.** *If* $\mathbf{M}$ *is a functional algebra,* $\mathbf{N}_0 \hookrightarrow \mathbf{M}$*, and* $\mathbf{K}$ *is a computation theory over* $(M, \mathbf{N}_0)$ *such that every given of* $\mathbf{M}$ *is strongly* **K***-effective, then every* $\mathbf{M}$*-explicit functional is strongly* **K***-effective.*

PROOF. This extends Lemma 3C.8, whose proof we omitted because it is very simple. Here too, we will leave for Problem x5D.1 the new, required case, which is also basically simple. ⊣

At this point, the proof of Lemma 3C.9 goes through word-for-word in this extended case and gives us half of what we want to prove in this section.

5D.3. **Lemma.** *If* $\mathbf{M}$ *is a functional algebra and* $\mathbf{N}_0 \hookrightarrow \mathbf{M}$*, then every* $\mathbf{M}$*-recursive partial function is* $\mathbf{K}[\mathbf{M}, \mathbf{N}_0]$*-computable.*

Next we turn to the converse, which requires stronger hypotheses.

5D.4. **Definition.** A **recursive pair** in a functional algebra $\mathbf{M}$ is any triple $\boldsymbol{\pi} = (\pi, \pi_1, \pi_2)$ such that:

(1) $\pi : M \times M \rightarrowtail M$ is an $\mathbf{M}$-recursive injection—a coding of pairs.

(2) π_1 and π_2 are $\mathbf{M}$-recursive inverses of π, i.e.,

$$\pi_1\pi(u, v) = u, \ \pi_2\pi(u, v) = v \quad (u, v \in M).$$

(3) The image of π,

$$\text{Pair}(w) \iff w \in \pi[M \times M] = \{\pi(u, v) \mid u, v \in M\}$$

is $\mathbf{M}$-recursive.

The last condition (3) is automatically true if $\pi : M \times M \rightarrowtail\!\!\!\!\rightarrow M$ is a bijection, or if the equality relation $=_M$ is $\mathbf{M}$-recursive, since by (1) and (2)

$$\text{Pair}(w) \iff w = \pi(\pi_1(w), \pi_2(w)).$$

An algebra $\mathbf{M}$ is **paired** if it has two distinguished points $(0, 1)$ and admits an $\mathbf{M}$-recursive pair.

5D.5. **Lemma.** *If* $\mathbf{M}$ *is paired, then* $\mathbf{N}_0 \hookrightarrow \mathbf{M}$.

PROOF. Suppose first that for all v, $\pi(1, v) \neq 0$ (we will take care of that possibility later), and set

$$\begin{aligned} 0_{\mathbf{M}} &= 0, \\ S(u) &= \pi(1, u), \\ Pd(u) &= \text{if } (u = 0) \text{ then } 0 \text{ else } \pi_1(u), \\ \mathbb{N} &= \bigcap\{X \subseteq M \mid 0 \in X \ \&\ S[X] \subseteq X\}. \end{aligned}$$

The function S is an injection and never $= 0$ by the extra hypothesis on π, and the algebra $(\mathbb{N}, 0, S \restriction \mathbb{N}, Pd \restriction \mathbb{N}, =_0)$ clearly satisfies the Peano axioms and so is isomorphic with $\mathbf{N}_0$. To see that $\mathbb{N}$ is semirecursive, suppose $\overline{r}$ is the least solution of the recursion equation

$$\begin{aligned} r(u) = \text{if } (u = 0) \text{ then } \mathrm{tt} \\ \text{else if } (\text{Pair}(u) \ \&\ \pi_1(u) = 1) \text{ then } r(Pd(u)) \\ \text{else } \perp. \end{aligned} \tag{5-37}$$

By an easy induction on n, $\overline{r}(n) = \mathrm{tt}$ for every $n \in \mathbb{N}$. For the converse, it is enough to show that the partial function

$$s\chi_{\mathbb{N}}(u) = \text{if } (u \in \mathbb{N}) \text{ then } \mathrm{tt} \text{ else } \perp$$

satisfies (5-37), and this is obvious for $u \in \mathbb{N}$. If for some $u \notin \mathbb{N}$ the right-and-side of (5-37) were defined, then $u \in \pi[M \times M]$, $\pi_1(u) = 1$ and $\pi_2(u) = Pd(u) \in \mathbb{N}$, which means that $u = \pi(1, Pd(u)) = S(Pd(u)) \in \mathbb{N}$ contradicting the hypothesis.

Finally, if $0 = \pi(1, v)$ for some v, replace π by

$$\pi'(u, v) = \pi(0, \pi(u, v))$$

which never takes on the value 0 since $\pi_1\pi'(u, v) = 0 \neq 1 = \pi_1(0)$. The projection functions for this pair and the image $\pi'[M \times M]$ are (easily) **M**-recursive. ⊣

5D.6. **Proposition (Sequence coding).** *If* **M** *is paired, then there exists an injection*

$$\langle\ \rangle^M : M^* \rightarrowtail M,$$

a **tuple coding**, *so that the following conditions hold.*

(1) *For each* $n = 0, 1, \dots$, *the* n*-ary function*

$$f_n(u_0, \dots, u_{n-1}) = \langle u_0, \dots, u_{n-1}\rangle^M$$

is recursive.

(2) *The relations*

$$\begin{aligned} \text{Null}(u) &\iff u = \langle\emptyset\rangle^M, \\ \text{Symbol}(u) &\iff (\exists s)[u = \langle s\rangle^M] \end{aligned}$$

are recursive.

(3) *There exist* (total) *recursive functions* $\text{head}(u)$, $\text{tail}(u)$ *and* $\text{append}(s, u)$, *such that*

$$\begin{aligned} \text{head}(\langle u_0, \dots, u_{n-1}\rangle^M) &= u_0 \quad \textit{if } (n > 0) \\ \text{tail}(\langle u_0, \dots, u_{n-1}\rangle^M) &= \langle u_1, \dots, u_{n-1}\rangle^M \\ \text{append}(s, \langle u_0, \dots, u_{n-1}\rangle^M) &= \langle s, u_0, \dots, u_{n-1}\rangle^M. \end{aligned}$$

PROOF. First we define $\langle u_0, \dots, u_{n-1}\rangle^M$ by the following recursion on the length n of the tuple:

$$\begin{aligned} \langle\emptyset\rangle^M &= \pi(0, 1), \\ \langle u_0, \dots, u_n\rangle^M &= \pi(1, \pi(u_0, \langle u_1, \dots, u_n\rangle^M)). \end{aligned}$$

The restriction of $\langle\ \rangle^M$ to tuples of any fixed length n is obviously recursive,

$$\begin{aligned} \text{Null}(u) &\iff \text{Pair}(u)\ \&\ \pi_1(u) = 0\ \&\ \pi_2(u) = 1, \\ \text{Symbol}(u) &\iff \text{Pair}(u)\ \&\ \pi_1(u) = 1, \end{aligned}$$

and (3) follows easily by setting

$$\begin{aligned} \mathrm{head}(u) &=_{\mathrm{df}} \pi_1\pi_2(u), \\ \mathrm{tail}(u) &=_{\mathrm{df}} \pi_2\pi_2(u), \\ \mathrm{append}(s,u) &=_{\mathrm{df}} \pi(1,\pi(s,u)). \end{aligned}$$ ⊣

Using these basic sequence-coding functions, we can define many more, useful recursive functions which allow us to manipulate sequence codes.

5D.7. **Proposition.** *If a coding of sequences* $\langle\ \rangle^M : M^* \rightarrowtail M$ *in a functional algebra* **M** *satisfies* (1) – (3) *of Proposition* 5D.6, *then the following hold, where*

$$\mathrm{Seq}^M(u) \iff (\exists u_0,\dots,u_{n-1})[u = \langle u_0,\dots,u_{n-1}\rangle^M] \tag{5-38}$$

and $\mathbb{N}$ *is the copy of the natural numbers in* M.

(4) *There is a recursive partial function* $\mathrm{lh}^M(u) : M \rightharpoonup \mathbb{N}$, *such that*

$$\begin{aligned} \mathrm{Seq}^M(u) &\iff \mathrm{lh}^M(u)\!\downarrow, \\ \mathrm{lh}^M(\langle u_0,\dots,u_{n-1}\rangle^M) &= n \quad (u_0,\dots,u_{n-1} \in M), \end{aligned}$$

so that, in particular, the relation $\mathrm{Seq}^M(u)$ *is* **M**-*semirecursive.*

(5) *There is a binary,* **M**-*recursive partial function* $(u)_i^M$, *such that*

$$\begin{aligned} (u)_i^M\!\downarrow &\iff i \in \mathbb{N} \quad (u \in M), \\ (\langle u_0,\dots,u_{n-1}\rangle^M)_i^M &= u_i \quad (i < n), \end{aligned}$$

In using iterations of these partial functions, we will write

$$(u)_{i,j}^M = ((u)_i^M)_j^M, \quad (u)_{i,j,k}^M = (((u)_i^M)_j^M)_k^M, \ \textit{etc.} \tag{5-39}$$

(6) *There is a binary, recursive partial function* $u * s$, *which codes concatenation on sequence codes,*

$$\langle u_0,\dots,u_{n-1}\rangle^M * \langle v_0,\dots,v_{m-1}\rangle^M = \langle u_0,\dots,u_{n-1},v_0,\dots,v_{m-1}\rangle^M,$$

and such that

$$\mathrm{Seq}^M(u) \implies (\forall s)(u * s\!\downarrow), \tag{5-40}$$

$$\mathrm{Seq}^M(u)\ \&\ \mathrm{Seq}^M(v) \implies (\forall s)[(u * v) * s) = u * (v * s)]. \tag{5-41}$$

(7) *There is a recursive, partial function* $\mathrm{rev}(u)$, *such that*

$$\mathrm{rev}(\langle u_0,\dots,u_{n-1}\rangle^M) = \langle u_{n-1},u_{n-2},\dots,u_0\rangle^M.$$

PROOF. For (4), set

$$\mathrm{lh}^M(u) = p(u) \tag{5-42}$$
$$\overline{\textbf{where}}\ \{p(u) = \text{if } (u = \langle\emptyset\rangle^M) \text{ then } \Uparrow \text{ else } p(\mathrm{tail}(u)) + 1\}.$$

Clearly $\mathrm{lh}^M(\langle u_0, \dots, u_{n-1}\rangle^M) = n$, by induction in n; and the length function (defined only on sequence codes) satisfies the recursion equation (5-42), and so the least solution cannot have a larger domain.

For the remaining functions, we set:

$$(u)_i^M = p(u) \overline{\textbf{where}}\ \{p(u) = \text{if } (i = 0) \text{ then head}(u) \text{ else } p(\text{tail}(u))_{i-1}\}$$

$$u * s = p(u, s) \overline{\textbf{where}}\ \{p(u, s) = \text{if } (u = \langle\emptyset\rangle^M) \text{ then } s \text{ else if Symbol}(u) \text{ then append}(\text{head}(u), s) \text{ else append}(\text{head}(u), p(\text{tail}(u), s))\}$$

$$\mathrm{rev}(u) = p(u) \overline{\textbf{where}}\ \{\text{if } u = \langle\emptyset\rangle^M \text{ then } u \text{ else } (p(\text{tail}(u)) * \langle\text{head}(u)\rangle^M)\}.$$

The crucial properties (5-40), (5-41) of the concatenation function are proved by induction on $\mathrm{lh}^M(u)$, and the remaining claims are trivial. ⊣

5D.8. **Exercise.** Show that if $\mathbf{M}$ is paired, then there are recursive, partial functions $u \restriction j$, $u \upharpoonright j$, $f(j, u)$ such that

$$\langle u_0, \dots, u_{n-1}\rangle^M \restriction j = \langle u_0, \dots, u_{j-1}\rangle^M \quad (\text{tt} \le j \le n),$$
$$\langle u_0, \dots, u_{n-1}\rangle^M \upharpoonright j = \langle u_j, \dots, u_{n-1}\rangle^M \quad (\text{tt} \le j \le n),$$
$$f(j, \langle u_1, \dots, u_{n-1}\rangle^M) = \langle u_j, u_1, \dots, u_{j-1}, u_{j+\text{ff}}, \dots, u_n\rangle^M.$$

HINT: Use primitive recursion for $u \restriction j$ and $u \upharpoonright j$ and define $f(j, u)$ using these functions.

We have used the annoying superscript ^{M} to decorate some of the functions and relation associated with a coding of M^*, because we will be using them in the next theorem along with the corresponding functions of classical (primitive recursive) coding of tuples of $\mathbb{N}$ and we must avoid confusion. It goes without saying that when there is no need of confusion, we will just skip the superscript—just as we often say "recursive" when we mean "$\mathbf{M}$-recursive".

5D.9. **Theorem.** *For a paired functional algebra* $\mathbf{M}$*, a partial function* $f : M^n \rightharpoonup M$ *is* $\mathbf{M}$*-recursive if and only if it is* $\mathbf{K}[\mathbf{M}, \mathbf{N}_0]$*-recursive.*

PROOF. We only need to show the "if" part of the theorem, which amounts to proving that the partial function

$$\phi(z, x) = \{z\}((x)_0^M, (x)_1^M, \dots, (x)^M_{\mathrm{lh}^M(z) \dot{-} 1})$$

is $\mathbf{M}$-recursive. In fact, it is the fixed point of some $\mathbf{M}$-recursive functional whose definition is routine—if somewhat messy. ⊣

This basic result applies to most functional algebras which arise naturally in computability theory, if for no other reason because we always can (and we often want to) add a distinguished points and a "free" pair to algebra $\mathbf{M}$, cf. Problem *x5D.2. And when it holds, then $\mathbf{M}$-recursion has all the nice

properties whose proof requires the Enumeration Theorem—most often via the Second Recursion Theorem. A most beautiful example of this is the following theorem of Gandy.

5D.10. **Theorem (Gandy's Selection Theorem).** *Suppose* $\mathbf{M}$ *is a paired and normal functional algebra and* $f : M^{n+1} \rightharpoonup M$ *is* $\mathbf{M}$*-recursive: then there exists an* $\mathbf{M}$*-recursive partial function* $g : M \rightharpoonup \mathbb{N}$ *such that*

$$\text{if } (\exists n \in \mathbb{N}) f(n, \vec{x})\downarrow, \text{ then } g(\vec{x})\downarrow \ \&\ f(g(\vec{x}), \vec{x})\downarrow.$$

PROOF. Let $\boldsymbol{\sigma}(z, \vec{u}, e, \vec{v})$ be the stage comparison function for the enumerating partial function $\{z\}(\vec{u})$ "against itself", so that

$$\boldsymbol{\sigma}(z, \vec{u}, e, \vec{v}) = \begin{cases} \text{tt}, & \text{if } \{z\}(\vec{u})\downarrow \ \&\ |z, \vec{u}| \leq |e, \vec{v}|, \\ \text{ff}, & \text{if } \{e\}(\vec{v})\downarrow \ \&\ |e, \vec{v}| < |z, \vec{u}|, \end{cases}$$

where $\vec{u}$ and $\vec{v}$ range over tuples of the same length, say $n+1$. Suppose

$$f(n, \vec{x}) = \{\tilde{f}\}(n, \vec{x}),$$

and choose by the Second Recursion Theorem a number e such that

$$\{e\}(t, \vec{x}) = \begin{cases} 0, & \text{if } t \in \mathbb{N} \ \&\ \boldsymbol{\sigma}(\tilde{f}, t, \vec{x}, e, t+1, \vec{x}) = \text{tt}, \\ \{e\}(t+1, \vec{x}) + 1, & \text{otherwise.} \end{cases}$$

(1) *If* $f(t, \vec{x})\downarrow$, *then* $\{e\}(t, \vec{x})\downarrow$. This is because if $f(t, \vec{x})\downarrow$, then

$$w = \boldsymbol{\sigma}(\tilde{f}, t, \vec{x}, e, t+1, \vec{x})\downarrow.$$

If $w = \text{tt}$, then $\{e\}(t, \vec{x}) = 0$, by the definition; and if $w = \text{ff}$, then the definition gives

$$\{e\}(t, \vec{x}) = \{e\}(t+1, \vec{x}) + 1,$$

but $|e, t+1, \vec{x}| < |\tilde{f}, t, \vec{x}| < \infty$, so $\{e\}(t+1, \vec{x})\downarrow$.

By (essentially) the same argument,

(2) *If* $\{e\}(t+1, \vec{x})\downarrow$, *then* $\{e\}(t, \vec{x})\downarrow$;

and then directly from (1) and (2),

(3) *If* $f(n, \vec{x})\downarrow$ *for some* n, *then* $\{e\}(0, \vec{x})\downarrow$.

(4) *If* $\{e\}(t, \vec{x})\downarrow$, *then either* $\{e\}(t, \vec{x}) = 0$, *or* $\{e\}(t, \vec{x}) = \{e\}(t+1, \vec{x}) + 1$.

Let

$$s = \text{the least } t \text{ such that } \{e\}(t, \vec{x}) = 0;$$

this exists, otherwise by (4)

$$\{e\}(0, \vec{x}) = \{e\}(1, \vec{x}) + 1 = \{e\}(2)(2, \vec{x}) + 2 = \cdots$$

so that $\{e\}(0, \vec{x}) \geq t$ for all t, which is absurd. For this s then,

$$f(s, \vec{x})\downarrow, \quad \{e\}(s, \vec{x}) = 0,$$

and then, by the choice of s, successively,

$$i < s \Longrightarrow \{e\}(s-i) = i,$$

so that $\{e\}(0,\vec{x}) = s$ and the conclusion of the theorem is satisfied if we set $g(\vec{x}) = \{e\}(0,\vec{x})$. $\dashv$

Problems for Section 5D

x5D.1. Suppose $\mathbf{K}$ is a computation theory over $(M, \mathbf{N}_0)$, $g(\vec{x}, r, p)$ and $h(s,t,\vec{y},q)$ are strongly $\mathbf{K}$-effective functionals, and

$$f(\vec{x},\vec{y},q,p) = g(\vec{x},\lambda(s)\lambda(t)h(s,t,\vec{y},q),p).$$

Prove that $f(\vec{x},\vec{y},q,p)$ is strongly $\mathbf{K}$-effective.

*x5D.2. Prove that for every functional algebra $\mathbf{M}$ such that $\mathbf{N}_0 \hookrightarrow \mathbf{M}$, every $\mathbf{M}$-recursive functional is strongly $\mathbf{K}[\mathbf{M},\mathbf{N}_0]$-effective.

*x5D.3. Suppose $\mathbf{M} = (M, f_1, \dots, f_L)$ is a functional algebra, $0, 1$ are points not in the universe M, set $M[0,1] = M \cup \{0,1\}$. Show first that there exists an injection

$$\pi : M[0,1] \times M[0,1] \rightarrowtail M[0,1] \rightarrowtail U \setminus M[0,1]$$

into some set U which never takes on a value in $M[0,1]$. Let M^+ be the closure of $M[0,1]$ under such an injection π, and set

$$\mathbf{M}^+ = (M^+, f_1^+, \dots, f_L^+, 0, 1, \chi_0, \chi_1, \pi, \pi_1, \pi_2)$$

where π_1, π_2 are the inverses of π (set $= 0$ on $M[0,1]$) and $f_1^+, \dots, f_L^+$ are the minimal extensions of $f_1, \dots, f_L$ to monotone functionals on M^+, e.g., with p binary,

$$f_i^+(\vec{x},p) = f_i(\vec{x}, p \restriction M^2).$$

(1) Prove that every $\mathbf{M}$-recursive partial function is $\mathbf{M}^+$-recursive.

(2) Prove that if $\mathbf{M}$ is paired and $f : (M^+)^n \rightharpoonup {M^+}_{\mathbb{B}}$ is $\mathbf{M}^+$-recursive, then the restriction $f \restriction M^n : M^n \rightharpoonup M_{\mathbb{B}}$ is $\mathbf{M}$-recursive.

*x5D.4 (**open**). Prove the Gandy Selection Theorem for normal functional algebras $\mathbf{M}$ such that $\mathbf{N}_0 \hookrightarrow \mathbf{M}$. (This is likely to be false, but there is no obvious counterexample.)

REFERENCES

J. McCarthy [1963], *A basis for a mathematical theory of computation*, **Computer programming and formal systems** (P. Braffort and D Herschberg, editors), North-Holland, pp. 33–70.