# An axiomatic approach to computers

by

MELVIN C. FITTING

(Herbert H. Lehman College of the City University of New York)

## 1. *Introduction*

IN THIS PAPER we outline an axiomatic treatment of idealized computers "from the outside." The computers are idealized in the sense that no consideration is made of storage space or computation time. Our treatment is "from the outside" in the sense that we deal with computers as black boxes, paying attention only to input and output, ignoring what happens on the way through. Our approach thus differs from Turing machines, which are an analysis of the act of computation. Indeed, in our treatment there is nothing available that corresponds to a computation at all. But we think it interesting that many basic qualitative results can be derived from simple, intuitively plausible axioms which don't commit one to a specific model of how computers work inside.

The general plan is as follows. We first establish some very weak conditions for a *computation device*. We organize these into a *computation laboratory*, which must meet certain minimal requirements. We postulate a *universal machine* which can be programmed, and which duplicates the work of all the computation devices in the computation laboratory. Then we postulate that the programs themselves are subject to certain simple manipulations in the computation laboratory. Finally we postulate that our devices receive input and produce output in finite chunks.

After each group of axioms, we state some consequences, and discuss significance. We have chosen the rather unusual policy of omitting almost all proofs. Many are straightforward, those that are not are often modifications of analogs in recursion theory. We believe that what is important here is the organization and arrangement, rather than the details of argument.

In the concluding section we give references for various results and styles of development. We have chosen not to do this as we go along, so as not to break the continuity.

## 2. *Computation devices*

Let $\mathfrak{A}$ be some non-empty set, the *domain*. Informally the members of $\mathfrak{A}$ are the objects our computing devices work with. They may be numbers or words, for example. We make no restrictions on $\mathfrak{A}$ until later.

By $[\mathfrak{A}]^n$ $(n=1, 2, \ldots)$ we mean the collection of all $n$-place relations on $\mathfrak{A}$. That is, $[\mathfrak{A}]^n$ is the power set of $(\underbrace{\mathfrak{A}\times\mathfrak{A}\times\ldots\times\mathfrak{A}}_{n \text{ times}})$.

Let $\Phi$ be a mapping from $[\mathfrak{A}]^n$ to $[\mathfrak{A}]^m$, formally $\Phi$: $[\mathfrak{A}]^n \rightarrow [\mathfrak{A}]^m$. We call $\Phi$ a *computation device* of *order* $\langle n, m\rangle$. If $\Phi(\mathfrak{P})=\mathfrak{R}$, we call $\mathfrak{P}$ *input* for $\Phi$, and $\mathfrak{R}$ *output*. Note that input is a set of $n$-tuples and output is a set of $m$-tuples; in effect our computation devices read and write in columns. Input and output may be finite or infinite. The only restriction, so far, on computation devices is that *order* of input and output is not significant.

*Examples of computation devices.* Let $P^2 : [\mathfrak{A}]^2 \rightarrow [\mathfrak{A}]^1$ be defined by: if $\mathfrak{P}\in[\mathfrak{A}]^2$ then $P^2(\mathfrak{P})=\{x|$ for some $y\in\mathfrak{A}$, $\langle y, x\rangle\in \mathfrak{P}\}$. $P^2$ is a *projection operator* of order $\langle 2, 1\rangle$ and is a "reasonable" computation device in that its operation seems quite "mechanical." It simply takes a set of ordered pairs, $\mathfrak{P}$, as input, deletes all first components, and gives the result as output. For each $n>1$ there is a similar projection operator $P^n$ of order $\langle n, n-1\rangle$ which deletes the first component of a set of $n$-tuples. It is reasonable to have $P^n$ available in any computation laboratory.

As a kind of opposite to this, consider the operator $A^n$, of order $\langle n, n+1\rangle$, that "adds a place". That is, $A^n$ takes a set of $n$-tuples, $\mathfrak{P}$, as input, adjoins an extra first component, which is an arbitrary member of the domain $\mathfrak{A}$, and gives as output the resulting set of $n+1$ tuples. More formally, $A^n(\mathfrak{P})=\{\langle y, x_1, \ldots, x_n\rangle | y\in\mathfrak{A}$ and $\langle x_1, \ldots, x_n\rangle\in \mathfrak{P}\}$. It is plausible that each $A^n$ would be available in any computation laboratory.

As a third basic family of examples, consider the *transposition* *operators*. $T^n_{i,j}$ is the operator of order $\langle n, n\rangle$ that, when given a set of $n$-tuples $\mathcal{D}$ as input, switches around the $i$th and $j$th components. This too seems quite "mechanical", and so one may assume each $T^n_{i,j}$ should be available in any computation laboratory.

Let $\Phi$ and $\Psi$ be two computation devices. There are certain "reasonable" ways $\Phi$ and $\Psi$ might be combined to form a new computation device (by wiring machines together or by hiring somebody to physically scan outputs and collate them appropriately). We single out the following for special attention.

*Intersection and union*. Let $\Phi$ and $\Psi$ be computation devices of the same order, say $\langle n, m\rangle$. We define two other computation devices, also of order $\langle n, m\rangle$ by, for $\mathcal{D}\in[\mathfrak{A}]^n$,

$$(\Phi\cap\Psi)(\mathcal{D})=\Phi(\mathcal{D})\cap\Psi(\mathcal{D})$$
$$(\Phi\cup\Psi)(\mathcal{D})=\Phi(\mathcal{D})\cup\Psi(\mathcal{D}).$$

*Product*. Let $\Phi$ be a computation device of order $\langle n, m\rangle$, and $\Psi$ be a computation device of order $\langle n, m'\rangle$. We define a computation device of order $\langle n, m+m'\rangle$ by, for $\mathcal{D}\in[\mathfrak{A}]^n$,

$$(\Phi\times\Psi)(\mathcal{D})=\Phi(\mathcal{D})\times\Psi(\mathcal{D}).$$

*Composition*. Let $\Psi$ be a computation device of order $\langle n, m\rangle$, and $\Phi$ be a computation device of order $\langle m, p\rangle$. We define a computation device of order $\langle n, p\rangle$ by, for $\mathcal{D}\in[\mathfrak{A}]^n$,

$$(\Phi\Psi)(\mathcal{D})=\Phi(\Psi(\mathcal{D})).$$

One can make use of machines designed to generate output, without using input, machines to print trigonometry tables, for example. Rather than create a whole new category in our axiomatic development, we informally identify machines that don't take input with machines for which input is not relevant. Formally, let $\Phi$ be a computation device of order $\langle n, m\rangle$. We say $\Phi$ is *constant* with output $\mathfrak{R}$ if $\Phi(\mathcal{D})=\mathfrak{R}$ for every $\mathcal{D}\in[\mathfrak{A}]^n$. If $A$ is a collection of computation devices, we say $\mathfrak{R}$ is *generated in $A$* if $\mathfrak{R}$ is the output of some constant member of $A$.

## 3. *Computation laboratories*

We postulate the existence of a collection (potentially unlimited in size) of computation devices which can be combined in ways which allow for "reasonable" flexibility.

Let $A$ be a collection of computation devices. We call $A$ a *computation laboratory* if it meets the following conditions:

1. the empty set is generated in $A$.
2. for each $a \in \mathfrak{A}$ the set $\{a\}$ is generated in $A$.
3. $=_{\mathfrak{A}}$ is generated in $A$ where $=_{\mathfrak{A}}$ is $\{\langle x, y\rangle \mid x \in \mathfrak{A}, y \in \mathfrak{A} \text{ and } x = y\}$.
4. $P^n \in A$ (where $P^n$ is a projection operator).
5. $A^n \in A$ (where $A^n$ is a place-adding operator).
6. $T^n_{i,j} \in A$ (where $T^n_{i,j}$ is a transposition operator).
7. $A$ is closed under composition.
8. $A$ is closed under $\cap$ and $\cup$.
9. $A$ is closed under $\times$.

We now give some consequences of these assumptions, grouped into topics.

*Functions*. There are two ways we might think of a function $f: \mathfrak{A}^n \to \mathfrak{A}^m$ being "computed" in a computation laboratory $A$.

1. There is a constant computation device $\Phi$ in $A$ with (the graph of) $f$ as output. (For example, a machine designed to print a sine table.) In this case, we say $f$ is *generated* in $A$, thus identifying $f$ with its graph.

2. There is a computation device $\Phi$ in $A$ which, when given $\{\langle x_1, \ldots, x_n\rangle\}$ as input, gives $\{f(x_1, \ldots, x_n)\}$ as output. (For example, a pocket calculator which, when given $\theta$, displays $\sin\theta$ on command.) In this case, we say $f$ is *computed pointwise* in $A$.

Similar remarks apply to *partial functions*, that is, functions whose domains are subsets of $\mathfrak{A}^n$.

One consequence of our assumptions is: if a (partial) function is generated in a computation laboratory $A$, then it is also computed pointwise in $A$. To get some feeling for how our assumptions are used, we present a proof of this in a representative special case.

Suppose $f: \mathfrak{A} \to \mathfrak{A}$ is generated in the computation laboratory $A$. That is, suppose there is a computation device $\Phi \in A$ of order $\langle 1, 2\rangle$, such that, for every input $\mathcal{D}$, $\Phi(\mathcal{D})$ is the graph of $f$. That is,

$$\Phi(\mathcal{P})=\{\langle x,y\rangle \mid f(x)=y\}.$$

Let $\Psi=P^2(\Phi\cap T^2_{1,2}A^1)$. Since $\Psi$ is built up from operators known to be in $A$, using composition and intersection, $\Psi$ is in $A$. And we claim, for each $x\in\mathfrak{A}$, $\Psi(\{x\})=\{f(x)\}$, and thus $f$ is computed pointwise in $A$.

To verify our claim, we need only compute $\Psi(\{a\})$ for an arbitrary $a\in\mathfrak{A}$. And we do this by computing both $\Phi$ and $T^2_{1,2}A^1$ on $\{a\}$, taking the intersection of the results, and then taking $P^2$ of that. Well,

$$\Phi(\{a\})=\{\langle x,y\rangle \mid f(x)=y\}$$

and

$$\begin{aligned}&T^2_{1,2}A^1(\{a\})=T^2_{1,2}(A^1(\{a\}))=\\&=T^2_{1,2}(\{\langle y,a\rangle \mid y\in\mathfrak{A}\})=\\&=\{\langle a,y\rangle \mid y\in\mathfrak{A}\}.\end{aligned}$$

Intersecting these,

$$\begin{aligned}&\{\langle x,y\rangle \mid f(x)=y\}\cap\{\langle a,y\rangle \mid y\in\mathfrak{A}\}=\\&=\{\langle a,f(a)\rangle\}.\end{aligned}$$

And finally, $P^2$ of this is simply $\{f(a)\}$.

Thus, for functions, generated implies computed pointwise. Later, using some additional assumptions about $A$, we will also have the converse. We do not yet, so we take the notion of being generated as basic.

*Relative generation.* Let $\mathcal{P}$ and $\mathfrak{R}$ be relations on $\mathfrak{A}$. We write $\mathcal{P}\leq\mathfrak{R}$ if $\mathcal{P}=\Phi(\mathfrak{R})$ for some $\Phi\in A$ (thus $\mathcal{P}$ is generated in $A$ by some computation device that uses $\mathfrak{R}$ as input.

For this notion the following items hold in all computation laboratories.

1. $\leq$ is transitive and reflexive.
2. $\mathcal{P}$ is generated in $A$ iff $\mathcal{P}\leq\mathfrak{R}$ for every $\mathfrak{R}$.
3. for $\mathcal{P},\mathfrak{R}\neq\varnothing$, $\mathcal{P}\leq\mathcal{P}\times\mathfrak{R}$ and $\mathfrak{R}\leq\mathcal{P}\times\mathfrak{R}$.
4. if $\mathcal{P}\leq S$ and $\mathfrak{R}\leq S$, then $\mathcal{P}\times\mathfrak{R}\leq S$.

*Characteristic functions.* Let $\mathcal{P} \in [\mathfrak{A}]^n$. The characteristic function of $\mathcal{P}$ is

$$c_{\mathcal{P}}(v_1, \ldots, v_n) = \begin{cases} 1 \text{ if } \langle v_1, \ldots, v_n \rangle \in \mathcal{P} \\ 0 \text{ if } \langle v_1, \ldots, v_n \rangle \notin \mathcal{P} \end{cases}$$

(Here we must assume $\mathfrak{A}$ has at least two members. We use 0 and 1 without any numerical significance.)

Also, for $\mathcal{P} \in [\mathfrak{A}]^n$, we write $\bar{\mathcal{P}}$ for $\mathfrak{A}^n - \mathcal{P}$.

The following hold in all computation laboratories.

1. $\mathcal{P} \times \bar{\mathcal{P}} \leq c_{\mathcal{P}}$ and $c_{\mathcal{P}} \leq \mathcal{P} \times \bar{\mathcal{P}}$
2. $c_{\mathcal{P}} \leq c_{\mathcal{R}}$ iff $\mathcal{P} \times \bar{\mathcal{P}} \leq \mathfrak{R} \times \bar{\mathfrak{R}}$

*Deciding about membership.* Let $\mathcal{P} \in [\mathfrak{A}]^n$. What meaning might be attached to: computation laboratory $A$ provides means for deciding whether or not $\langle x_1, \ldots, x_n \rangle \in \mathcal{P}$ for arbitrary $x_1, \ldots, x_n \in \mathfrak{A}$? We might take it to mean one of the following three items:

a) Both $\mathcal{P}$ and $\bar{\mathcal{P}}$ are generated in $A$. (Then to decide if $\langle x_1, \ldots, x_n \rangle \in \mathcal{P}$, turn on a device to generate $\mathcal{P}$, and one to generate $\bar{\mathcal{P}}$, and see which turns out $\langle x_1, \ldots, x_n \rangle$.)

b) $\mathcal{P} \times \bar{\mathcal{P}}$ is generated in $A$. (Then to decide if $\langle x_1, \ldots, x_n \rangle \in \mathcal{P}$, turn on a device to generate $\mathcal{P} \times \bar{\mathcal{P}}$ and see if $\langle x_1, \ldots, x_n \rangle$ turns up among the first or the last $n$ components.)

c) $c_{\mathcal{P}}$ is generated in $A$. (Then to decide if $\langle x_1, \ldots, x_n \rangle \in \mathcal{P}$, turn on a device to generate $c_{\mathcal{P}}$ and let it work until it tells you whether $\langle x_1, \ldots, x_n \rangle$ maps to 1 or 0.)

A consequence of our assumptions is that these three are equivalent in all computation laboratories.

*Equality.* We say a computation laboratory $A$ is one *with equality* if the relation $\neq_{\mathfrak{A}}$ is generated in $A$, where $\neq_{\mathfrak{A}}$ is the relation $\{\langle x, y \rangle \mid x \in \mathfrak{A}, y \in \mathfrak{A} \text{ and } x \neq y\}$. If $A$ is a computation laboratory with equality, the following hold:

1. Let $\mathcal{P}$, $\mathcal{P}' \in [\mathfrak{A}]^n$. If they differ by a finite number of elements, then

$$\mathcal{P} \leq \mathcal{P}'$$
$$\mathcal{P} \times \bar{\mathcal{P}} \leq \mathcal{P}' \times \bar{\mathcal{P}}'$$
$$c_{\mathcal{P}} \leq c_{\mathcal{P}'}$$

2. If $\mathfrak{I}$ is finite, then $\mathfrak{D} \leq \mathfrak{I}$ iff $\mathfrak{D}$ is generated in $A$. (That is, a finite input can always be built into the computation device instead.)
3. If $f$ and $g$ are *total* functions then $f \leq g$ iff $f \times \bar{f} \leq g \times \bar{g}$ iff $c_f \leq c_g$.

## 4. *Universal machines*

We now impose some requirements on the domain $\mathfrak{A}$. We suppose that, to each computation device $\Phi$ of order $\langle n, m\rangle$ in our computation laboratory $A$, there corresponds at least one *program*, so that no program corresponds to more than one computation device, and that program is a member of $\mathfrak{A}$ itself. Intuitively, the program $p$ contains information on how to work in the manner that $\Phi$ does, though how it contains this information does not concern us. What is significant here is that $p$ must be a member of $\mathfrak{A}$, thus programs themselves are things our computation devices can manipulate. We do not, by the way, postulate that a computation device has only one program. Incidentally, our present assumption about programs implies that $\mathfrak{A}$ is infinite.

*Programmable computation devices.* Let $n$ and $m$ be fixed. We now postulate the existence, in our computation laboratory $A$, of one universal machine that can simultaneously do the work of all our computation devices of order $\langle n, m\rangle$. The way it keeps track of what output goes with what computation device is to "tag" outputs with the program according to which they were computed.

More precisely, we suppose there is a computation device $\mathfrak{A}^{\langle n, m\rangle}$ in $A$ of order $\langle n, m+1\rangle$ such that, if $\Phi$ is any computation device in $A$ of order $\langle n, m\rangle$, and if $p$ is any program for $\Phi$, then $\langle x_1, \ldots, x_m\rangle \in \Phi(\mathfrak{D})$ iff $\langle p, x_1, \ldots, x_m\rangle \in \mathfrak{A}^{\langle n, m\rangle}(\mathfrak{D})$ for all inputs $\mathfrak{D} \in [\mathfrak{A}]^n$.

Note that different choices of $n$ and $m$ give different universal machines. We return to this point in section 6.

A consequence of our present assumptions is: there exists a set $\mathfrak{D} \subseteq \mathfrak{A}$ such that $\mathfrak{D}$ is generated in $A$ but $\bar{\mathfrak{D}}$ is not. In other words, there is a set for which our computation laboratory $A$ can not provide a decision procedure. The proof of this is short, and we insert it here; it is a simple diagonal argument.

Let $\mathfrak{D} = \{x \mid \langle x, x\rangle \in \mathfrak{A}^{\langle 1, 1\rangle}(\emptyset)\}$. If $\bar{\mathfrak{D}}$ were generated, there would be a program, say $i$, for a constant computation device with $\bar{\mathcal{P}}$ as output. Then $x \in \bar{\mathcal{P}} \Leftrightarrow \langle i, x\rangle \in \mathfrak{U}^{\langle 1,1\rangle}(\emptyset)$. But also $x \in \bar{\mathcal{P}} \Leftrightarrow x \notin \mathcal{P} \Leftrightarrow$ $\Leftrightarrow \langle x, x\rangle \notin \mathfrak{U}^{\langle 1,1\rangle}(\emptyset)$. Thus $\langle i, x\rangle \in \mathfrak{U}^{\langle 1,1\rangle}(\emptyset) \Leftrightarrow \langle x, x\rangle \notin \mathfrak{U}^{\langle 1,1\rangle}(\emptyset)$. Taking $x$ to be $i$ gives a contradiction.

## 5. *Manipulating programs*

Since programs are in the domain $\mathfrak{A}$, they are objects to be worked with by our computation devices. We will make two assumptions, one concerning outputs, the other, inputs. After each, we list and discuss consequences. First, some terminology.

*Output sections.* Let $\Phi$ be a computation device in $A$ of order $\langle k, q+n\rangle$, and let $x_1, \ldots, x_q \in \mathfrak{A}$. By $\Phi_{x_1,\ldots,x_q}$ we mean the mapping of order $\langle k, n\rangle$ defined by

$$\Phi_{x_1,\ldots,x_q}(\mathcal{P}) = \{\langle y_1,\ldots,y_n\rangle \mid \langle x_1,\ldots,x_q,y_1,\ldots,y_n\rangle \in \Phi(\mathcal{P})\}.$$

It can be shown that, for each choice of $x_1, \ldots, x_q \in \mathfrak{A}$, the map $\Phi_{x_1,\ldots,x_q}$ is also a computation device in $A$. We call it the *output section* of $\Phi$ at $\langle x_1, \ldots, x_q\rangle$.

Now, let $\Phi$ be a fixed computation device in $A$. Is there any systematic way of telling a lab assistant how to carry out the $\Phi_{x_1,\ldots,x_q}$ computations, for various values of $x_1, \ldots, x_q$? One might do this quite simply as follows. Use the device $\Phi$ and, as outputs emerge, look at them and see if they begin with $\langle x_1, \ldots, x_q\rangle$; if they do, keep them, otherwise throw them away; of the outputs left, discard the initial $\langle x_1, \ldots, x_q\rangle$. These are general instructions on how to compute $\Phi_{x_1,\ldots,x_q}$ once $x_1, \ldots, x_q$ are specified. We now make the assumption that what we can explain to a lab assistant, a computation device could "explain" to our universal machine. That is, once the computation device $\Phi$ has been fixed, it should be a "mechanical" process, to program our universal machine to imitate the various $\Phi_{x_1,\ldots,x_q}$.

*Output place-fixing assumption.* Let $\Phi$ be a computation device in $A$ of order $\langle k, q+n\rangle$. We suppose there is a function $f:\mathfrak{A}^q \to \mathfrak{A}$ *which is generated in $A$*, such that for each $x_1, \ldots, x_q$ we have that $f(x_1, \ldots, x_q)$ is a program for $\Phi_{x_1,\ldots,x_q}$.

We now list some consequences of this assumption.

*Manipulation of programs.* Among our basic assumptions about computation laboratories was closure under $\cap$, $\cup$, and $\times$. This closure should be reflected in some kind of routine reprogramming of our universal machines. In fact, we have the following.

1. Pick an order, $\langle n, k\rangle$. Then there exist functions $f$ and $g$, *both of which are generated in $A$* such that, if $\Phi$ and $\Psi$ are computation devices in $A$ of order $\langle n, k\rangle$ and if $p$ is any program for $\Phi$ and $q$ is any program for $\Psi$ then: $f(p, q)$ is a program for $\Phi \cap \Psi$ and $g(p, q)$ is a program for $\Phi \cup \Psi$.

2. Pick two orders $\langle n, k\rangle$ and $\langle n, k'\rangle$. Then there exists a function $h$ *which is generated in $A$* such that, if $\Phi$ is any computation device of order $\langle n, k\rangle$, if $\Psi$ is any computation device of order $\langle n, k'\rangle$, if $p$ is any program for $\Phi$ and if $q$ is any program for $\Psi$, then $h(p, q)$ is a program for $\Phi \times \Psi$.

*Kleene fixed point theorem.* This important consequence has as direct consequences some of the results below.

Let $t: \mathfrak{A} \to \mathfrak{A}$ be any function which is generated in $A$. Pick an order $\langle n, k\rangle$. Then there exists a program $p$ for a computation device of order $\langle n, k\rangle$ such that $t(p)$ is also a program for the same computation device.

*Alternate programs.* We have supposed that to each computation device there corresponds at least one program. Suppose, in addition to these "official" programs, we had a second "alternate" way of assigning programs to our computation devices (analogous to using a different programming language). One might suppose that the passage between "official" and "alternate" programs ought to be a routine matter. In fact, we have the following. Suppose our "alternate" programs meet the same conditions that our "official" ones do; that is, there are universal computation devices in $A$ that use the "alternate" programs, and for which an output place-fixing assumption holds. Then for each order $\langle n, k\rangle$ there are functions $f$, $g: \mathfrak{A} \to \mathfrak{A}$ *both of which are generated in $A$* such that: 1) if $p$ is an "official" program for a computation device of order $\langle n, k\rangle$ then $f(p)$ is an "alternate" program for the same computation device and 2) if $q$ is an "alternate" program for a computation device of order

$\langle n, k\rangle$ then $g(q)$ is an "official" program for the same computation device.

In the sequel, we suppose only the original "official" programs are used.

*Rice's Theorem.* Pick an order $\langle n, k\rangle$. Let $\mathcal{P}$ be a set of programs for computation devices of order $\langle n, k\rangle$. $\mathcal{P}$ is called *closed* if, whenever it contains one program for some computation device, it contains every program for it. Recall, $\mathcal{P}$ is *decidable* if $\mathcal{P}$ and $\bar{\mathcal{P}}$ are both generated in $A$. Then, the only closed, decidable sets are the empty set and the set of all programs for all computation devices of order $\langle n, k\rangle$.

This result has, in its turn, many undecidability results as consequences. For example, let $\Phi$ be some fixed computation device of order $\langle n, k\rangle$ and consider the problem of deciding what is and what is not a program for $\Phi$. Let $\mathcal{P}$ be the set of all programs for $\Phi$. $\mathcal{P}$ is closed, not empty, and not all programs. Hence $\mathcal{P}$ is not decidable. There are many other such examples.

We now turn to the second assumption of this section. Again, some terminology.

*Input sections.* Let $\Phi$ be a computation device in $A$ of order $\langle q+k, n\rangle$, and let $x_1, \ldots, x_q \in \mathfrak{A}$. By $\Phi^{x_1, \ldots, x_q}$ we mean the mapping of order $\langle k, n\rangle$ defined by

$$\Phi^{x_1, \ldots, x_q}(\mathcal{P}) = \Phi(\{\langle x_1, \ldots, q_q\rangle\} \times \mathcal{P}).$$

It can be shown that, for each $x_1, \ldots, x_q \in \mathfrak{A}$, the map $\Phi^{x_1, \ldots, x_q}$ is also a computation device in $A$. We call it the *input section* of $\Phi$ at $\langle x_1, \ldots, x_q\rangle$.

Let $\Phi$ be fixed, and consider how one might instruct a lab assistant to carry out the $\Phi^{x_1 \cdots x_q}$ computations for various values of $x_1, \ldots, x_q$. One might say: take your input $\mathcal{P}$, tack $\langle x_1, \ldots, x_q\rangle$ on to the beginning of all members of $\mathcal{P}$, give the result to $\Phi$ and collect the output. We now assume that comparable instructions can be "mechanically" given to our universal machines. The following is in addition to the other assumptions thus far.

*Input place-fixing assumption.* Let $\Phi$ be a computation device in $A$ of order $\langle q+k, n\rangle$. We suppose there is a function $f:\mathfrak{A}^q \to \mathfrak{A}$ *which is generated in $A$*, such that for each $x_1, \ldots, x_q$ we have that $f(x_1, \ldots, x_q)$ is a program for $\Phi^{x_1,\ldots,x_q}$.

We now continue listing consequences.

*Functions.* A (partial) function $f$ is generated in $A$ iff $f$ is computed pointwise in $A$.

*Manipulation of programs (cont.).*

1. One can "mechanically" produce programs for constants. That is, for each $n$ there is a function $f:\mathfrak{A}^n \to \mathfrak{A}$ *which is generated in $A$* such that, for each $x_1, \ldots, x_n$ we have that $f(x_1, \ldots, x_n)$ is a program for a constant computation device with output $\{\langle x_1, \ldots, x_n\rangle\}$.

2. Composition is effective. That is, pick two orders $\langle n, k\rangle$ and $\langle k, q\rangle$; then there is a function $f$, *which is generated in $A$*, such that if $\Phi$ is any computation device of order $\langle k, q\rangle$, if $\Psi$ is any computation device of order $\langle n, k\rangle$, if $p$ is any program for $\Phi$ and if $q$ is any program for $\Psi$, then $f(p, q)$ is a program for $\Phi\Psi$.

3. Now all the basic notions of computation laboratories have been shown to have counterparts in terms of "mechanical" reprogramming of our universal machines. It can easily be shown that composition of generated functions produces a generated function. Thus the correspondence extends to operators built up from the basic ones as well.

## 6. *Pairing functions*

This business of having different universal machines for different orders can be annoying. If there were some way of combining the information of an $n$-tuple into a single item, the problem could be avoided. Now, $\mathfrak{A}$ is infinite, so $\mathfrak{A}\times\mathfrak{A}\times \ldots \times\mathfrak{A}$ and $\mathfrak{A}$ have a 1-1 correspondence (standard result of set theory). If there were such a 1-1 correspondence which was generated in $A$, it would take care of the problem completely. As a matter of fact, there will be such for every value of $n$ if there is one for $n=2$. Such generated 1-1 correspondences between $\mathfrak{A}\times\mathfrak{A}$ and $\mathfrak{A}$ are usually called *effective pairing func-*

*tions*, and they exist in many models for our axioms thus far. Actually, a somewhat weaker condition suffices for most purposes.

*From now on we assume.* There is a function $f$: $\mathfrak{A} \times \mathfrak{A} \to \mathfrak{A}$ which is 1-1, not necessarily onto, and which is generated in $A$.

We may think of $f$ as a device to collapse pairs of information into singletons. As indicated above, there are then similar generated 1-1 functions to collapse triples, quadruples, etc.

If we agree to identify a pair $\langle x, y \rangle$ with its image $f(x, y)$ under $f$, then the multiplicity of orders disappears, and a universal machine of order $\langle 1, 1 \rangle$ can be shown to do the work of all the others.

Our assumption about pairing functions will not be used explicitly below, but it is involved in the form of some of our further assumptions.

## 7. *Finitary behavior*

We have said nothing so far about the processes by which our computation devices operate. We are about to impose some mild restrictions to the effect that, though input may be infinite, it is used in finite chunks and, though output may be infinite, it is generated in finite chunks according to a regular pattern. We will make this more precise below, but first we must introduce some means of getting our computation devices to recognize finite sets. The problem is, as things stand, we have no mechanism for telling a computation device that the finite input we may have given thus far is all it is ever going to get. The device can't tell the difference between finite and incomplete. Neither, for that matter, can we when outputs are concerned. It ought to be possible for our computation devices to deal with finite sets as single, complete objects. For these purposes we make use of *finite codes*.

$\mathfrak{A}$ is infinite, so the collection of finite subsets of $\mathfrak{A}$ has a 1-1 correspondence with $\mathfrak{A}$ itself (standard result of set theory). If there were such a 1-1 correspondence $f$ which we could somehow make use of in $A$, it would provide a way around the difficulty. For a finite subset $F$ of $\mathfrak{A}$ we could take the member of $\mathfrak{A}$ which corresponds to $F$ under $f$ as a "code" for $F$ on which our computation devices could work. It would, in fact, be a single object in $\mathfrak{A}$ as required.

As a matter of fact, a somewhat weaker setup suffices for our needs.

*From now on we assume the following:*

1. To each finite subset $F$ of $\mathfrak{A}$ there corresponds one or more members of $\mathfrak{A}$, called *finite codes* for $F$.

2. Different finite subsets have distinct finite codes. (Thus we suppose a finite coding which is neither unique nor which requires that everything in $\mathfrak{A}$ be a code. We next add assumptions which make this coding useful in $A$.)

We write $D_a$ for the finite subset of $\mathfrak{A}$ with code $a$. If $a$ is not a finite code, $D_a$ has no meaning.

3. The relation: $y$ is a finite code and $x \in D_y$, is generated in $A$.

4. The relation: $y$ is a finite code and $x \notin D_y$, is generated in $A$.

5. There is a computation device $S$ in $A$ of order $\langle 1, 1\rangle$ such that $S(\mathcal{D}) = \{a \mid D_a \subseteq \mathcal{D}\}$.

We remark that many models exist that satisfy all our assumptions, including these. Indeed, given any infinite set $\mathfrak{A}$, and given any finite list of relations $\mathfrak{R}_1, \ldots, \mathfrak{R}_k$ on $\mathfrak{A}$, there is a model $A$ in which all our assumptions hold, with $\mathfrak{A}$ as its domain, and with each of $\mathfrak{R}_1, \ldots, \mathfrak{R}_k$ being a generated relation. We have more to say about this in section 8 below. Now we return to listing consequences.

*Elementary consequences.* There are computation devices, $U$, $V$ and $W$ in $A$ of order $\langle 1, 1\rangle$ such that, if $y$ is a finite code,

$$U(\{y\}) = D_y$$
$$V(\{y\}) = \bar{D}_y$$
$$W(\{y\}) = \{x \mid D_x = D_y\}$$

We are not yet assuming our computation devices work on inputs in finite chunks, but it is now possible to produce some that do.

*Compact, monotone computation devices.* Let $R$ be some two-place relation which is generated in $A$. Define a map $\Phi$ of order $\langle 1, 1\rangle$ by: $\Phi(\mathcal{D}) = \{x \mid \text{ for some } D_y \subseteq \mathcal{D},\ R(y, x)\}$. It is a consequence of our assumptions that $\Phi$ is a computation device in $A$. Further, $\Phi$ is easily seen to be monotone ($\mathcal{P} \subseteq \mathcal{R}$ implies that $\Phi(\mathcal{P}) \subseteq \Phi(\mathcal{R})$ and *compact* ($x \in \Phi(\mathcal{P})$ implies $x \in \Phi(F)$ for some finite $F \subseteq \mathcal{P}$).

Saying that $\Phi$ is monotone is essentially saying that an output won't be recalled on the basis of additional input. Saying that $\Phi$ is compact is essentially saying that it makes use of input in finite chunks; no single output needs more than a finite part of an infinite input.

We cannot prove that all our computation devices must be monotone and compact. In fact, there are models for the above axioms in which there are non-compact operators. We are, however, able to prove that all compact, monotone operators have the form which appears above.

*Compact, monotone computation devices* (*cont*). Let $\Phi$ be a computation device in $A$ of order $\langle 1, 1\rangle$ which is monotone and compact. It is a consequence of our assumptions that there is a relation $R$, which is generated in $A$, such that $\Phi(\mathcal{P}) = \{x \mid \text{for some } D_y \subseteq \mathcal{P}, R(y, x)\}$.

This is significant in the following sense. If we want to study a monotone, compact computation device $\Phi$ in $A$, we can get at it through the relation $R$ associated with it by the above. But $R$ is generated in $A$, that is, $R$ is the output of a constant computation device, one for which input is not relevant. Thus we can study monotone, compact computation devices in $A$ indirectly by studying computation devices in $A$ which don't use input. So we turn our attention to them now. We want to postulate that constant computation devices generate output in finite chunks, according to a regular pattern. A little more precisely, we want to postulate that if a set can be generated in $A$ at all, it can be generated in finite chunks, and we can say what those chunks are by giving (having a machine give) codes for them.

*From now on we assume the following*. Let $\mathcal{P}$ be a set which is generated in $A$. There is a chain $\mathsf{C}$ of finite sets such that

1. $\cup \mathsf{C} = \mathcal{P}$
2. If $\mathcal{D}$ is a proper initial segment of $\mathsf{C}$ then $\cup \mathcal{D}$ is finite
3. the set of finite codes for members of $\mathsf{C}$ is generated in $A$.

In order to properly present some consequences of the above, we should say something about infinite inputs. Since we are finite beings, the only way we can give an infinite input $\mathcal{D}$ to a computation device is by having some machine (which presumably works forever) generate $\mathcal{D}$. That is, the only infinite inputs we are practically concerned with are those which are generated in $A$.

*Compact, monotone computation devices* (*cont. again*). It is a consequence of our assumptions that every computation device in $A$ agrees with some monotone, compact computation device on inputs which are generated in $A$.

If we are interested in inputs which are generated in $A$, we have not only the inputs themselves to work with, but also programs for computation devices which generated them.

*Generated Inputs.* Let $\Phi$ be a computation device in $A$ of order $\langle 1, 1\rangle$. There is a function $f$, which is generated in $A$, such that if $\mathcal{D}$ is any generated set and if $p$ is any program for $\mathcal{D}$, then $f(p)$ is a program for a constant computation device which generates $\Phi(\mathcal{D})$.

This can be proved without using our above assumption about outputs. But using it, we also have the converse.

*Generated inputs* (*cont.*). Let $f$ be any function which is generated in $A$. There is a computation device $\Phi$ in $A$ such that, if $\mathcal{D}$ is any generated set, and if $p$ is any program for generating $\mathcal{D}$, then $f(p)$ is a program for generating $\Phi(\mathcal{D})$.

This carries our development far enough. From this point on, much depends on just what additional assumptions about the finite sets we may wish to add. As a matter of fact, the assumptions we made above hold in various models, interpreting "finite" to mean something other than truly finite. A discussion of this is beyond the scope of the present paper.

## 8. *Conclusion*

The preceding is really a version of axiomatic recursion theory. Besides (what is sometimes called) ordinary recursion theory, there have been developed several other theories bearing a strong resemblance to ordinary recursion theory. Axiomatic recursion theory aims, in part, at abstracting the structure common to these theories. The axiom system above is actually a simplified version of a more general system we have developed (called a *production system*) in which operators need not be everywhere defined, and in which "finite" need not mean actually finite. The full axiom system applies to hyperarithmetic theory and to $\alpha$-recursion theory as well as to ordinary recursion theory. Here we have kept only so much as applies to computers. The axioms are developed in full generality in [2].

Rather than choosing the recursive functions of ordinary recursion theory as the basic items to be abstractly developed, we have chosen things called *enumeration operators* which, we feel, are a better formal counterpart to the actual operation of computers than are recursive functions. See [6 starting on p. 146] for a treatment of them in ordinary recursion theory.

In [2] we define a notion of "recursion theory" for an arbitrary structure, and investigate the conditions under which the assumptions of this paper will hold for it. Though we proceed along entirely different lines, our approach is equivalent to using Rogers' definition of enumeration operator from ordinary recursion theory [6], but applying it in the framework of *search computability* [3], which is meaningful for an arbitrary structure.

There are several "families" of axiomatic recursion theories in the literature. Ours is in that one originating in [1], and our *computation laboratories* in section 3 are certain *admissible subcategories of* $R(X^*)$ in their terminology. The axioms after these initial ones are of our choosing. The results in section 3 are all basic in ordinary recursion theory. Establishing them on the basis of our axioms should present no great difficulties.

In section 4, the existence of a universal machine derives from Turing's seminal work [8].

In section 5, our output place-fixing assumption is our formulation for operators of a result about recursive functions, due to Kleene and known as the Iteration Theorem. The results on Manipulation of Programs, the Kleene fixed point theorem, and alternate programs follow by proofs in the style of [7, pp. 67—73], but translated to operators. Rice's theorem comes from [5]. It can be derived from the Kleene theorem.

The Input place-fixing assumption seems to be new. It is easily shown to hold in ordinary recursion theory, and the results we list after it are simple consequences.

Finite codes in ordinary recursion theory are developed in [6, pp. 69—71] where they are called *canonical indices*. Our choice of axioms seems to be new, as is our derivation in section 7, of what Rogers takes as the definition of enumeration operators.

The results on generated inputs are due to [4] in ordinary recursion theory.

## *References*

[1] EILENBERG, S. & ELGOT, C. *Recursiveness*, Academic Press, New York (1970).

[2] FITTING, M. *Fundamentals of generalized recursion theory*, to be published, North-Holland Publishing Co., Amsterdam.

[3] MOSCHOVAKIS, Y. "Abstract first order computability I, II" *Transactions of the American mathematical society*, Vol. 138 (1969), pp. 427—504.

[4] MYHILL, J. & SHEPHERDSON, J., "Effective operations on partial recursive functions". *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, vol. 1 (1955), pp. 310—317.

[5] RICE, H. "Classes of recursively enumerable sets and their decision problems", *Transactions of the American mathematical society*, vol. 74 (1953), pp. 358—366.

[6] ROGERS, H. *Theory of recursive functions and effective computability*, McGraw-Hill Book Co., New York (1967).

[7] SMULLYAN, R. *Theory of formal systems*, Princeton University Press, Princeton (1961).

[8] TURING, A. "On computable numbers with an application to the Entscheidungsproblem", *Proceedings of the London mathematical society*, ser. 2, vol. 42 (1936), pp. 230—265, vol. 43 (1937), pp. 544—546.

*Received on January 30, 1979.*