11-30-1994

Semantics vs. Syntax vs. Computations Machine Models for Type-2 Polynomial-Time Bounded Functionals (Preliminary Draft)

James S. Royer
Syracuse University

SU-CIS-94-10

Semantics vs. Syntax vs. Computations

Machine Models for Type-2 Polynomial-Time Bounded Functionals

James S. Royer

November 30, 1994

School of Computer and Information Science
Syracuse University
Suite 4-116, Center for Science and Technology
Syracuse, NY 13244-4100

Semantics vs. Syntax vs. Computations

Machine Models for Type-2 Polynomial-Time Bounded Functionals

(Preliminary Draft)

James S. Royer

School of Computer and Information Science
Syracuse University
Syracuse, NY 13244 USA

Email: royer@top.cis.syr.edu

Abstract

This paper investigates analogs of the Kreisel-Lacombe-Shoenfield Theorem in the context of the type-2 basic feasible functionals, a.k.a. the Mehlhorn-Cook class of type-2 polynomial-time functionals.

We develop a direct, polynomial-time analog of effective operation, where the time bound on computations is modeled after Kapron and Cook's scheme for their basic polynomial-time functionals. We show that (i) if P = NP, these polynomial-time effective operations are strictly more powerful on $\mathcal{R}$ (the class of recursive functions) than the basic feasible functions, and (ii) there is an oracle relative to which these polynomial-time effective operations and the basic feasible functionals have the same power on $\mathcal{R}$.

We also consider a weaker notion of polynomial-time effective operation where the machines computing these functionals have access to the computations of their "functional" parameter, but not to its program text. For this version of polynomial-time effective operation, the analog of the Kreisel-Lacombe-Shoenfield is shown to hold—their power matches that of the basic feasible functionals on $\mathcal{R}$.

November 30, 1994

1. The Problem and Its Classical Solutions in Recursion Theory

In programming languages, a *higher-order procedure* is a procedure that takes as arguments, or produces as results, other procedures. Higher-order procedures are powerful programming tools and are a stock feature of many contemporary programming languages, e.g., Scheme, ML, and Haskell.

One view of a procedure is as a syntactic object. Thus, one way of reading the above definition is as follows. A higher-order procedure is a procedure that takes syntactic objects as inputs, some of which it treats as procedures and others as non-procedural data items. The value produced by a call to such a procedure depends functionally on just the "meaning" (i.e., i/o behavior) of each procedural parameter together with the value of each data parameter.[1] We say that a procedure is *extensional* in a particular (syntactic) argument if and only if the procedure uses this argument only for its i/o behavior. Here are two examples: Let ω denote the natural numbers, let $\langle\varphi_p\rangle_{p\in\omega}$ be an acceptable numbering of the partial recursive functions over ω, let $apply = \lambda p, x \in \omega.\varphi_p(x)$ (i.e., $apply(p, x)$ simply returns the result of running program p on input x), and, finally, let $g = \lambda p, x \in \omega.(\varphi_p(x) + p)$. Clearly, *apply* is extensional in its first argument, but g is not.[2] We call this view of higher-order procedures the *glass-box approach.*

Most programming language texts tell you *not* to think of procedural parameters as syntactic objects. Their view is that a procedural parameter is considered as being contained in a *black box* that one can query for its i/o behavior, but one cannot see the code inside. Moreover, Scheme, ML, Haskell, etc., work to enforce this black-box view of procedures. A key rationale for this is that it is *very* difficult to tell whether a procedure over syntactic objects is extensional in a certain argument. Putting the procedural parameters into black boxes, on the other hand, guarantees that procedural parameters are used *only* for their i/o behavior.

There is an important question as to whether this black-box approach limits one's computing power. By looking at the syntax of a procedure, one can conceivably learn more about its i/o behavior than by simply querying a black box containing the procedure. So, does every glass-box style higher-order procedure correspond to a black-box style higher-order procedure? For type-2 there are two beautiful affirmative answers from recursion theory, given as Theorems 2 and 3 below. Before stating these theorems, we introduce some formal terminology.

Terminology: $A \rightharpoonup B$ (respectively, $A \to B$) denotes the collection of partial (respectively, total) functions from A to B. $\mathcal{PR}$ (respectively, $\mathcal{R}$) denotes the collection of partial (respectively, total) recursive functions over ω. Let $\langle\varphi_p\rangle_{p\in\omega}$ be as before.

General Convention: In this paper we restrict our attention to functionals that take one function argument and one numeric argument and return a numeric result. Generalizing the definitions and results below to functionals that take more that one function argument and multiple numeric arguments is straightforward, but adds nothing except notation to the discussion here.

Definition 1.

(a) Suppose $\mathcal{C} \subseteq \mathcal{PR}$. $\Gamma: \mathcal{C} \times \omega \rightharpoonup \omega$ is an *effective operation* on $\mathcal{C}$ if and only if, for some partial recursive $\alpha: \omega \rightharpoonup \omega$, we have that, for all p with $\varphi_p \in \mathcal{C}$ and all a, $\Gamma(\varphi_p, a) = \alpha(p, a)$; α is said to *determine* Γ.[3] When $\mathcal{C} = \mathcal{PR}$, Γ is called, simply, an *effective operation.* Γ is a *total effective operation on* $\mathcal{C}$ if and only if Γ is an effective operation on $\mathcal{C}$ with the additional property that, for all $\psi \in \mathcal{C}$ and all $a \in \omega$, $\Gamma(\psi, a)$ is defined. *Convention:* Whenever we speak of a *computation of an effective operation* we shall mean the computation of a partial recursive function that determines the effective operation.

[1] In this paper we will not worry about the value itself possibly being a procedure.

[2] Where the "meaning" of p is taken to be φ_p.

[3] Note that α is extensional in its first argument with respect to p with $\varphi_p \in \mathcal{C}$.

(b) $\Gamma\colon (\omega \rightharpoonup \omega) \times \omega \rightharpoonup \omega$ is a *partial recursive functional* if and only if there is an oracle Turing machine $\mathbf{M}$ (with a function oracle) such that, for all $\alpha\colon \omega \rightharpoonup \omega$ and all $a \in \omega$, $\Gamma(\alpha, a) = \mathbf{M}(\alpha, a)$.[4]

(c) Let $\langle \sigma_i \rangle_{i \in \omega}$ be a canonical indexing of finite functions. $\Gamma\colon (\omega \rightharpoonup \omega) \times \omega \rightharpoonup \omega$ is an *effective continuous functional* if and only if there is an r.e. set A such that, for all $\alpha\colon \omega \rightharpoonup \omega$ and all $a, z \in \omega$, $\Gamma(\alpha, a)\downarrow = z \iff (\exists i)[\, \sigma_i \subseteq \alpha \;\&\; \langle i, a, z \rangle \in A\,]$.[5]

(d) Two functionals Γ and Γ' *correspond* on $\mathcal{C}$ if and only if, for all $\psi \in \mathcal{C}$ and all a, $\Gamma(\psi, a) = \Gamma'(\psi, a)$.

(e) Two classes of functionals $\mathbf{F}_0$ and $\mathbf{F}_1$ are said to *correspond on* $\mathcal{C}$ if and only if each Γ_0 in $\mathbf{F}_0$ corresponds on $\mathcal{C}$ to some $\Gamma_1 \in \mathbf{F}_1$ and each Γ_1 in $\mathbf{F}_1$ corresponds on $\mathcal{C}$ to some $\Gamma_0 \in \mathbf{F}_0$. ◇

Clearly, each effectively continuous functional corresponds on $\mathcal{PR}$ to some effective operation. The following two theorems concern converses to this.

Theorem 2 (The Myhill-Shepherdson Theorem [MS55]). *The effective operations correspond on $\mathcal{PR}$ with the effective continuous functionals.*

Theorem 3 (The Kreisel-Lacombe-Shoenfield Theorem [KLS57]). *The total effective operations on $\mathcal{R}$ correspond on $\mathcal{R}$ with the effective continuous functionals.*[6]

Theorem 2 is part of the foundations of programming language semantics (see, for example, [Sco75, pp. 190–193]). The two theorems say that of the power of effective operations (a syntactic/glass-box notion) is no greater than the power effective continuous functionals (a semantic/black-box notion) in two settings considered in the theorems. Thus, treating procedural parameters as black boxes does not lose one computing power.

But what about efficiency? It is quite conceivable that in computing an effective operation, where one has access to the syntax of the argument procedures, one could gain an advantage in efficiency over any corresponding black-box-style, higher-order procedure. Theorems 2 and 3 and their standard proofs are uninformative on this. This paper makes a start at examining this question by considering whether analogues of Theorems 2 and 3 hold for the Mehlhorn-Cook class of type-2 feasible functionals, a.k.a. the basic polynomial-time functionals. After establishing some general conventions in Section 2, Section 3 discusses the basic polynomial-time functionals, and Sections 4 and 5 then describe two different approaches to addressing this question.

[4] If, in the course of its computation, such an $\mathbf{M}$ queries its oracle α on an x for which α is undefined, then at that point, $\mathbf{M}$ goes undefined.

[5] Rogers [Rog67] calls these *recursive functionals*. Intuitively, an effective continuous functional can concurrently query its function argument α about multiple values and when the functional has enough information about α, it can produce an answer without having to wait for all of the answers to its queries to come in. A partial recursive functional, in contrast, must query its function argument *sequentially*. Effective continuous functionals are strictly more general than partial recursive functionals. For example,

$$\mathbf{OR}_{\|} \;=\; \lambda\alpha, x.\begin{cases} 1, & \text{if } \alpha(x)\downarrow \neq 0 \text{ or } \alpha(x+1)\downarrow \neq 0; \\ 0, & \text{if } \alpha(x)\downarrow = \alpha(x+1)\downarrow = 0; \\ \uparrow, & \text{otherwise;} \end{cases} \tag{1}$$

is an effective continuous, but not a partial recursive, functional. See [Rog67, Odi89] for detailed discussions of these notions.

[6] The totality assumption on the effective operation is necessary. Friedberg [Fri58, Rog67] has an example of a (nontotal) effective operation on $\mathcal{R}$ that fails to correspond to any effective continuous functional.

2. Notation, Conventions, and Such

Turing Machines M (with and without decorations) varies over multi-tape, deterministic Turing machines (TMs). $\mathbf{M}$ and M (with and without decorations) vary over oracle, multi-tape, deterministic Turing machines (OTMs) where the oracles are partial functions. We shall play a bit loose with the TM and OTM models: we'll speak of Turing machines as having subroutines, counters, arrays, etc. All of these can be realized in the standard model in straightforward ways with polynomial overhead, which suffices for the purposes of this paper. In our one bit of fussiness, we follow Kapron and Cook's [KC91] conventions for assigning costs to OTM operations. Under these conventions, the cost of an oracle query $\alpha(x)$=? is $|\alpha(x)|$, if $\alpha(x)\downarrow$, and ∞, if $\alpha(x)\uparrow$. Every other OTM operation has unit cost.

The Standard Acceptable Programming System and Complexity Measure We assume that our standard acceptable programming system, $\langle\varphi_p\rangle_{p\in\omega}$, is based on an indexing of Turing machines; $\langle\Phi_p\rangle_{p\in\omega}$ denotes the standard run-time measure on Turing machines, our standard complexity measure associated with $\langle\varphi_p\rangle_{p\in\omega}$. Thus, for all p and x, $\Phi_p(x) \geq \max(|x|, |\varphi_p(x)|)$ where the max is ∞ when $\varphi_p(x)\uparrow$. For each p and x, define $\Phi^*_p(x) = \max(|p|, \Phi_p(x))$. Intuitively, under Φ^*, the costs of loading the program, reading the input, and writing the output are all part of the cost of running program p on input x. Also, for each p and n, define $\overline{\Phi}_p(n) = \max\{\, \Phi_p(x) : |x| \leq n \,\}$ and $\overline{\Phi^*_p}(n) = \max\{\, \Phi^*_p(x) : |x| \leq n \,\}$.

Et Cetera We let $\langle\cdot,\cdot\rangle$ denote a standard, polynomial-time pairing function. Following Kapron and Cook [KC91] we define the length of a function $\alpha\colon \omega \rightharpoonup \omega$ (denoted $|\alpha|$) to be the function $\lambda n.\, \max\{\, |\alpha(x)| : |x| \leq n \,\}$.

3. Basic Polynomial-Time Functionals

Mehlhorn [Meh76] introduced a class of type-2 functionals to generalize the notion of Cook reducibility from reductions between sets to reductions between functions. His definition was based on a careful relativization of Cobham's [Cob65] syntactic definition of polynomial-time. Some years later Cook and Urquhart [CU89] defined an equivalent class of type-2 functionals (as well as analogous functionals of type greater than 2). This class of functionals, which they called the (type-2) *basic feasible functionals*, were developed by Cook and co-workers in a series of papers, see, for example, [Coo91, CK90, KC91]. Kapron and Cook's 1991 paper [KC91] is of particular importance here, as it introduced the first natural machine characterization of the type-2 basic feasible functionals, stated as Theorem 6 below.

Definition 4 (Kapron and Cook [KC91]). A *second-order polynomial* over type-2 variables $f_0, \ldots, f_m$ and type-1 variables $x_0, \ldots, x_n$ is an expression of one of the following five forms:

1. a 2. x_i 3. $\mathbf{q}_1 + \mathbf{q}_2$ 4. $\mathbf{q}_1 \cdot \mathbf{q}_2$ 5. $f_j(\mathbf{q}_1)$

where $a \in \omega$, $i \leq n$, $j \leq m$, and $\mathbf{q}_1$ and $\mathbf{q}_2$ are second-order polynomials over $\vec{f}$ and $\vec{x}$. The value of a second-order polynomial as above on $g_0, \ldots, g_m\colon \omega \to \omega$ and $a_0, \ldots, a_n \in \omega$ is the obvious thing. ◇

Definition 5 (Kapron and Cook [KC91]). $\Gamma\colon (\omega \to \omega) \times \omega \to \omega$ is a *basic polynomial-time functional* if and only if there is an OTM $\mathbf{M}$ and a second-order polynomial $\mathbf{q}$ such that, for all $g\colon \omega \to \omega$ and $a \in \omega$:

1. $\Gamma(g, a) = \mathbf{M}(g, a)$.
2. On input (g, a), $\mathbf{M}$ runs within $\mathbf{q}(|g|, |a|)$ time. ◇

Theorem 6 (Kapron and Cook [KC91]). *The class of type-2 basic feasible functionals corresponds on $\omega \to \omega$ to the class of basic polynomial-time functionals.*

This is a lovely and important result. However, there are two difficulties with Definition 5. First, the bound $\mathbf{q}(|\alpha|, |a|)$ is nonsensical when α is not total. Second, even for a total function g, the bound $\mathbf{q}(|g|, |a|)$ seems problematic. This is because $|g|(n)$ is defined by a max over $2^{n+1} - 1$ many values, hence the bound $\mathbf{q}(|g|, |a|)$ is not, in general, feasibly computable even when g is polynomial-time.

Seth [Set92, Set94] resolved both of these problems by formalizing the clocking notion (Definition 7) and showing the characterization (Theorem 8) below. (As Seth notes, these ideas were implicit in [KC91].) The idea behind the clocking scheme is that, in running a machine $\mathbf{M}$ clocked by a second-order polynomial $\mathbf{q}$, one keeps a running lower approximation to $\mathbf{q}(|g|, |a|)$ based on the information on g gathered from $\mathbf{M}$'s queries during the computation. Under the clocking scheme, $\mathbf{M}$'s computation runs to completion if and only if, for each step of $\mathbf{M}$, $\mathbf{M}$'s run time up to this step is less than the current approximation to $\mathbf{q}(|g|, |a|)$.

Notation: Suppose q is an ordinary polynomial over two variables. For each $i \in \omega$, define $q^{[i]}$ to be the second-order polynomial over f and x as follows: $q^{[0]}(f, x) = q(0, x)$. $q^{[i+1]}(f, x) = q(f(q^{[i]}(f, x)), x)$. A straightforward argument shows that, for each second-order polynomial $\mathbf{q}$ over f and x, there is a polynomial q and an $m \in \omega$ such that, for all g and a, $\mathbf{q}(|g|, |a|) \leq q^{[m]}(|g|, |a|)$.

Definition 7.

(a) Suppose $\mathbf{M}$ is an OTM, q is an ordinary polynomial over two variables, and $m \in \omega$. Let $\mathbf{M}_{q,m}$ be the OTM that, on input (α, a), operates as follows. $\mathbf{M}_{q,m}$ maintains a counter, clock, and two arrays $\mathsf{x}[0..m]$ and $\mathsf{y}[0..m-1]$. $\mathbf{M}_{q,m}$ maintains the following invariants:

$$\mathsf{x}[0] = q(0, |a|). \qquad \mathsf{x}[i+1] = q(\mathsf{y}[i], |a|),\ i < m.$$

$$\mathsf{y}[i] = \max\left(\left\{ |\alpha(x)| : \begin{array}{l} |x| \leq \mathsf{x}[i] \text{ and the query} \\ \alpha(x) =? \text{ has been made} \end{array} \right\}\right),\ i < m.$$

(For each $i \leq m$, $\mathsf{x}[i]$ is our lower approximation to $q^{[i]}(|\alpha|, |a|)$ and, for each $i < m$, $\mathsf{y}[i]$ is our lower approximation to $|\alpha|(q^{[i]}(|\alpha|, |a|))$.) On start up, $\mathbf{M}_{q,m}$ sets clock and each $\mathsf{y}[i]$ to 0 and sets each $\mathsf{x}[i]$ to $q(0, |a|)$. Then, $\mathbf{M}_{q,m}$ simulates $\mathbf{M}$, step by step, on input (α, a). For each step of $\mathbf{M}$ simulated:

- If the step of $\mathbf{M}$ just simulated is the query $\alpha(x)$=?, then:
  - if $\alpha(x)\uparrow$, then $\mathbf{M}_{q,m}(\alpha, a)$ is undefined, and
  - if $\alpha(x)\downarrow$, then, if necessary, $\mathbf{M}_{q,m}$ recomputes the $\mathsf{x}[i]$'s and $\mathsf{y}[i]$'s to re-establish the invariants.
- If, in the step of $\mathbf{M}$ just simulated, $\mathbf{M}$ halts with output y, then $\mathbf{M}_{q,m}$ outputs y and halts.
- If $\mathbf{M}$ did not halt in the step just simulated, then the cost of the step is added to clock and, if clock $< \mathsf{x}[m]$, the simulation continues; otherwise, $\mathbf{M}_{q,m}$ outputs 0 and halts.

(b) We say that Γ: $(\omega \rightharpoonup \omega) \times \omega \rightharpoonup \omega$ is a *basic polynomial-time clocked functional* if and only if Γ is computed by one of the $\mathbf{M}_{q,m}$'s as defined above. ◇

Note that a basic polynomial-time clocked functional has domain $(\omega \rightharpoonup \omega) \times \omega$, whereas a basic polynomial-time functional has just $(\omega \to \omega) \times \omega$ as its domain. Seth's notion is very conservative and constructive as compared to Definition 5, which on the surface seems to allow for machines that nonconstructively obey their time bounds. The following characterization is thus a little surprising and very pleasing.

Theorem 8 (Seth [Set92]). *The class of basic polynomial-time clocked functionals correspond on $(\omega \to \omega)$ to the class of basic polynomial-time functionals.*

Therefore, the classes of type-2 basic feasible functionals, basic polynomial-time functionals, and basic polynomial-time clocked functionals all correspond on $\omega \to \omega$, which is fairly good evidence that this class of functionals is robust.

4. Polynomial-Time Effective Operations

4.1. Definitions

We now consider how to define a sensible polynomial-time analog of an effective operation. To start, let us reconsider $apply: \omega^2 \rightharpoonup \omega$ from page 1. For most straightforward implementations we would have that the cost of computing $apply(p, x)$ is at least $\Phi_p(x)$ and is bounded above by $(\Phi_p(x))^{O(1)}$. It seems reasonable, then, that an account of the cost of computing an effective operation would include some dependence on the costs of running the program argument on various values during the course of the computation. Proposition 18 of the appendix shows that this dependence is, in fact, necessary to obtain a nontrivial notion.

We thus introduce the following definition, which is along the lines of Definition 5. Recall that, for all p, x, and n, $\Phi^*_p(x) \geq \max(|p|, |x|, |\varphi_p(x)|)$ and $\overline{\Phi}^*_p(n) = \max\{\, \Phi^*_p(x) : |x| \leq n \,\}$.

Definition 9. $\Gamma: \mathcal{R} \times \omega \to \omega$ is a *polynomial-time effective operation* if and only if there exist a partial recursive $\alpha: \omega^2 \rightharpoonup \omega$ and a second-order polynomial $\mathbf{q}$ such that α determines Γ (as an effective operation on $\mathcal{R}$) and, for all p with φ_p total and all $a \in \omega$, $\alpha(p, a)$ is computable within time $\mathbf{q}(\overline{\Phi}^*_p, |a|)$.[7] ◇

Clearly, each basic polynomial-time functional corresponds on $\mathcal{R}$ to some polynomial-time effective operation. However, Definition 9 isn't terribly satisfactory. It has the same problems noted of Definition 5—only worse, as it will turn out. To address these problems we introduce a clocked version of polynomial-time effective operation analogous to Seth's notion of Definition 7. But there is a difficulty in the way of this. In the computation of an effective operation, there are neither oracle calls nor reliable ways of telling when, for particular p_0 and x_0, $\varphi_{p_0}(x_0)$ is evaluated. Hence, it is a puzzle how a clocking mechanism is to gather appropriate information to approximate $\mathbf{q}(\overline{\Phi}^*_p, |a|)$. Our solution is an appeal to bureaucracy—we make clocked Turing machines computing effective operations fill out standardized forms to justify their expenses. That is, we equip the machines computing effective operations with UNIV, a standard subroutine that computes $\lambda p, x.\varphi_p(x)$, and when UNIV is called on arguments (p, x_0), we use the number of steps UNIV simulates of φ-program p on input x_0 as data for our lower approximation of $\mathbf{q}(\overline{\Phi}^*_p, |a|)$. Thus, one of these clocked machines has, at each point of each computation, an observable, verifiable justification for the amount of time it has consumed. These machines for computing effective operations are perfectly free to use means other than UNIV to evaluate $\varphi_p(x_0)$ for various x_0, but UNIV is the only means for justifying big run times to the clocking mechanism. Here are the details.

[7] Note that this notion (and the notions of Definitions 1(a) and 10) is implicitly parameterized by our choices of φ and Φ. Also note that by rights these functionals should be called the '*basic* polynomial-time effective operations' to indicate a bit of reservation about this class being the "correct" polynomial-time analogs of effective operations. While this reservation is quite reasonable, the terminology is already too long-winded. Thus, I've avoided using "basic" in this and the other terminology of sections 4 and 5.

Definition 10.

(a) Let UNIV denote a fixed TM subroutine that takes two arguments p_0 and x_0 and step-by-step simulates φ-program p_0 on input x_0 until, if ever, the simulation runs to its conclusion, at which time UNIV writes $\varphi_{p_0}(x_0)$ and $\Phi_{p_0}(x_0)$ on two separate tapes and erases all of its other tapes. We assume UNIV on arguments (p_0, x_0) runs in $(\Phi_{p_0}(x_0))^{O(1)}$ time.

(b) A *special Turing machine* (STM) M is a Turing machine defined as follows. M takes two inputs (p, x). M includes UNIV as a subroutine. M's instructions outside of UNIV do not write on any of UNIV's tapes except UNIV's input tape. When M is running UNIV on arguments (p_0, x_0) and $p_0 = p$, we say that M is making a *normal query*.[8] By convention, whenever an STM exits from a call to UNIV, the next instruction executed cannot be another call to UNIV.

(c) Suppose M is an STM, q is a polynomial over two variables, and $m \in \omega$. Let $M_{q,m}$ be the STM that, on input (p, a), operates as follows. $M_{q,m}$ maintains a counter clock and two arrays x[0..m] and y[0..$m-1$]. $M_{q,m}$ maintains the invariants: x[0] $= q(0, |a|)$; x[$i+1$] $= q(\mathsf{y}[i], |a|)$, $i < m$; and

$$\mathsf{y}[i] \;=\; \max\left(\left\{ |\Phi_p(x_0)| \;:\; \begin{array}{l} |x_0| \le \mathsf{x}[i] \text{ and the normal query} \\ \varphi_p(x_0) =? \text{ has been made} \end{array} \right\}\right), \quad i < m.$$

On start up, $M_{q,m}$ initializes clock, x, and y exactly as $\mathbf{M}_{q,m}$ does. Then, $M_{q,m}$ simulates M, step by step, on input (p, a). For each step of M simulated:

- If the step of M just simulated was part of a normal query, but the next step of M to be simulated is not (i.e., we are returning from a call to UNIV on (p, x_0) for some x_0), then, if necessary, $M_{q,m}$ recomputes the x[i]'s and y[i]'s to re-establish the invariants.
- If, in the step of M just simulated, M halts with output y, then $M_{q,m}$ outputs y and halts.
- If M did not halt in the step just simulated, then the value of clock is increased by 1 and, if clock $<$ x[m] *or if the step of M just simulated was part of a normal query*, then the simulation continues; otherwise, $M_{q,m}$ outputs 0 and halts.

(d) $\Gamma\colon (\omega \rightharpoonup \omega) \times \omega \rightharpoonup \omega$ is a *polynomial-time clocked effective operation* if and only if there exists an $M_{q,m}$ such that, for all p and x, $\Gamma(\varphi_p, x) = M_{q,m}(p, x)$. ◇

Note that a polynomial-time clocked effective operation has domain $\mathcal{PR} \times \omega$, whereas a polynomial-time effective operation has just $\mathcal{R} \times \omega$ as its domain, but a polynomial-time clocked effective operation when restricted to $\mathcal{R} \times \omega$ corresponds to some polynomial-time effective operation.[9]

4.2. Comparisons

Section 3 concluded by stating the correspondence on $\omega \to \omega$ of the classes type-2 basic feasible functionals, basic polynomial-time functionals, and basic polynomial-time clocked functionals. Here, complexity theoretic conundrums preclude having such a neat or conclusive story.

We first show that if P = NP, then the type-2 basic feasible functionals and the polynomial-time clocked effective operations *fail* to correspond on $\mathcal{R}$. We make use of the following

[8] Thus, a normal query to UNIV is like a query to an oracle for the φ-universal function to find the value of $\varphi_p(x_0)$, except we are bereft of divine inspiration and have to work out the answer to the query.

[9] Also note that this clocking scheme is based on *sequential* queries to UNIV. This causes a problem for nontotal function arguments. For example, the functional $\mathbf{OR}_\parallel$ from (1) is intuitively feasibly computable, but it is easy to show that $\mathbf{OR}_\parallel$ is not a polynomial-time clocked effective operation.

functional. For each $\alpha: \omega \rightharpoonup \omega$ and $x \in \omega$, define

$$\Gamma_0(\alpha, x) = \begin{cases} \uparrow, & \text{if (i): for some } y \in \{\mathbf{0}, \mathbf{1}\}^{|x|},\ \alpha(y)\uparrow; \\ 1, & \text{if (ii): } \textit{not} \text{ (i) and } (\exists y \in \{\mathbf{0}, \mathbf{1}\}^{|x|})[\ \alpha(y) \text{ is odd }]; \\ 0, & \text{if (iii): } \textit{not} \text{ (i) and } (\forall y \in \{\mathbf{0}, \mathbf{1}\}^{|x|})[\ \alpha(y) \text{ is even }]. \end{cases}$$

Proposition 11.
(a) The restriction of Γ_0 to $(\omega \to \omega) \times \omega$ is not basic feasible.[10]
(b) If P = NP, *then Γ_0 is a polynomial-time clocked effective operation.*

Thus, if the analog of the Kreisel-Lacombe-Shoenfield Theorem holds for the classes of functionals here under consideration, then P $\neq$ NP. Remark 19 in the appendix notes that one can weaken the P = NP hypothesis of Proposition 11(b). Proving the failure of this polynomial-time analog of the KLS Theorem seems problematic because of the following proposition.

Proposition 12. *There is an oracle relative*[11] *to which every polynomial-time effective operation corresponds on $\mathcal{R}$ to some basic polynomial-time functional.*

Similar difficulties arise in comparing the polynomial-time clocked and unclocked effective operations.

Proposition 13.
(a) If P = NP, *then the polynomial-time effective operations correspond on $\mathcal{R}$ to the polynomial-time clocked effective operations.*
(b) There is an oracle relative to which there is a polynomial-time effective operation that fails to correspond on $\mathcal{R}$ to any polynomial-time clocked effective operation.

The question with which we started was whether, in a polynomial-time setting, effective operations have an efficiency advantage over black-box style functionals. The above results demonstrate that there is little hope of resolving this question with present-day complexity theory. Since we've hit an apparent dead end with the original question, let us change the question a bit and ask instead to what extent can one open up the black boxes and still obtain a *provable* equivalence with the black-box models. The next section investigates one approach to this.

5. Functionals Determined by Computations over Computations

5.1. Definitions

Machines computing black-box style functionals have access only to the i/o behavior of their "procedural" parameters. Here we consider a style of functional where the machines computing them have access only to the *computational* behavior of their "procedural" parameter. Ideally, what we would like is a model where a machine computing a functional has the "text" of its procedural parameter hidden, but in which the machine can run its procedural parameter step-by-step on various arguments and observe the results, i.e., observe traces of computations evolve. In this paper we settle for a simplified/sanitized version of the above model which is still in the same spirit. In the model we use, machines computing a functional are supplied with an oracle

[10] Moreover, there is an honest, exponential-time computable function g such that, for each q and m, there is an n for which $\Gamma_0(g, \mathbf{0}^n) \neq \mathbf{M}_{q,m}(g, \mathbf{0}^n)$.

[11] Note: All the oracle results of this paper involve *full* relativizations.

that corresponds to the functional's procedural parameter as follows. When queried on $(x, \mathbf{0}^k)$, the oracle returns the result of running the procedural parameter on argument x, *provided* the procedural parameter produces an answer within k steps; if this is not the case, then the oracle returns $\star$, indicating "no answer yet." (Think of this model as providing black boxes with cranks attached that you have to turn a requisite number of times to receive an answer.) Below we formalize *shreds* ($\approx$ faint traces), a class of functions corresponding to such oracles, and *computation systems*, the recursion/complexity theoretic inspiration of shreds.

Notation: Define $\omega_\star = \omega \cup \{\star\}$, where $\star \notin \omega$. Let τ denote a copy of ω where the elements of τ are understood to be represented in unary over $\mathbf{0}^*$. Also let t (with and without decorations) range over elements of τ.

Definition 14.

(a) Suppose $\widehat{\varphi}$ is a acceptable programming system and $\widehat{\Phi}$ is a complexity measure associated with $\widehat{\varphi}$. The *computation system* for $\widehat{\varphi}$ and $\widehat{\Phi}$ is the recursive function $\widehat{\chi}\colon \omega^2 \times \tau \to \omega_\star$ defined by

$$\widehat{\chi} \;=\; \lambda p, x, \mathsf{t}.\begin{cases} \widehat{\varphi}_p(x), & \text{if } \widehat{\Phi}_p(x) \leq \mathsf{t}; \\ \star, & \text{otherwise.} \end{cases} \tag{2}$$

We usually write $\widehat{\chi}_p(x, \mathsf{t})$ for $\widehat{\chi}(p, x, \mathsf{t})$. Let χ be the computation system associated with φ and Φ, our standard acceptable programming system and associated complexity measure.

(b) A function $s\colon \omega \times \tau \to \omega_\star$ is a *shred* if and only if for each x, either (i) for all t, $s(x, \mathsf{t}) = \star$, or else (ii) there are $y \in \omega$ and $\mathsf{t}_0 \in \tau$ such that, for all $\mathsf{t} < \mathsf{t}_0$, $s(x, \mathsf{t}) = \star$ and, for all $\mathsf{t} \geq \mathsf{t}_0$, $s(x, \mathsf{t}) = y$. (Thus, each χ_p is a shred.)

(c) Suppose s is a shred. We define $\kappa s = \lambda x\,.\,(\mu \mathsf{t})[s(x, \mathsf{t}) \neq \star]$ and $\iota s = \lambda x\,.[s(x, (\kappa s)(x))$, if $(\kappa s)(x)\!\downarrow$; undefined, otherwise]. We also define $\overline{\kappa s} = \lambda n\,.\,\max\{\,(\kappa s)(x) : |x| \leq n\,\}$. (Thus, for each p, $\iota\chi_p = \varphi_p$, $\kappa\chi_p = \Phi_p$, and $\overline{\kappa\chi_p} = \overline{\Phi}_p$.)

(d) $\mathcal{S}_{\text{all}}$ denotes the collection of all shreds.

(e) For each $\mathcal{S} \subseteq \mathcal{S}_{\text{all}}$, $\iota\mathcal{S}$ denotes $\{\,\iota s : s \in \mathcal{S}\,\}$ and $\text{tot}(\mathcal{S})$ denotes $\{\,s \in \mathcal{S} : \iota s \text{ is total}\,\}$.

(f) For each computation system $\widehat{\chi}$, $\mathcal{S}_{\widehat{\chi}}$ denotes $\{\,\widehat{\chi}_p : p \in \omega\,\}$. ◇

The right hand side of (2) is a familiar tool from numerous recursion and complexity theoretic arguments. In most of these arguments the right hand side of (2) embodies all the information one needs about the computations of $\widehat{\varphi}$-programs; hence, for such arguments, shreds represent an adequate black box for computations.

Our next goal is to formalize an analog of the notion of effective operation where shreds take over the role played by programs in Definition 1(a).

Notation: $\mathcal{S}$ will range over subsets of $\mathcal{S}_{\text{all}}$. M will range over OTMs whose function oracles range over $\mathcal{S}_{\text{all}}$.

Definition 15. Suppose $\mathcal{S}$ is such that $\mathcal{R} \subseteq \iota\mathcal{S}$ and suppose $\Gamma\colon \mathcal{R} \times \omega \rightharpoonup \omega$.

(a) We say that an OTM M is *extensional* with respect to $\mathcal{S}$ if and only if for all s and $s' \in \mathcal{S}$ and all $a \in \omega$, if $\iota s = \iota s'$ then $\mathsf{M}(s, a) = \mathsf{M}(s', a)$.

(b) We say that Γ is an *effective shred-operation* with respect to $\mathcal{S}$ if and only if there is an OTM M that is extensional with respect to $\mathcal{S}$ such that, for all $s \in \mathcal{S}$ and $a \in \omega$, $\Gamma(\iota s, a) = \mathsf{M}(s, a)$; we say that M *determines* Γ.

(c) Γ is a *polynomial-time effective shred-operation* with respect to $\mathcal{S}$ if and only if there is an extensional (wrt $\mathcal{S}$) OTM M and a second order polynomial $\mathbf{q}$ such that, for all $s \in \mathcal{S}$ and $a \in \omega$:

1. $\Gamma(\iota s, a) = \mathsf{M}(s, a)$.
2. On input (s, a), M runs within $\mathbf{q}(\overline{\kappa s}, |a|)$ time. ◇

For each of the notions just defined, when the collection $\mathcal{S}$ is understood, we usually suppress mention of it.[12]

Definition 15(c) suffers from apparent difficulties analogous to the problems with Definitions 5 and 9—the bound $\mathbf{q}(\overline{\kappa s}, |a|)$ isn't generally feasibly computable and the totality restriction is a nuisance. So, as in sections 3 and 4, here we introduce a clocked version of the primary functional notion. Our clocking scheme is again based on the petty bureaucratic measure of having clocked machines fill out standardized forms to justify their expenses. In the present case this means that we equip OTMs computing our clocked functionals with a subroutine RUN which is as follows. Suppose $s \in \mathcal{S}_{\text{all}}$ is the function oracle of one of these OTM's. When an OTM calls RUN on $x \in \omega$, the result is either (i) $\langle s(x, \mathbf{0}^{2^k}), \mathbf{0}^{2^k}\rangle$ is returned, if there exists a k' such that $s(x, \mathbf{0}^{2^{k'}}) \neq \star$ and k is the least such k'; (ii) the calling OTM goes undefined, if no such k' exists.[13] The $\mathbf{0}^{2^k}$ values returned by calls to RUN are used as data for running our lower approximation of $\mathbf{q}(\overline{\kappa s}, |a|)$ in the same way we used the run times from calls to UNIV in Definition 10 as data for the running of our lower approximation of $\mathbf{q}(\overline{\Phi}_p, |a|)$. We call the class of functionals determined by such (extensional) machines the *polynomial-time effective clocked shred-operations.* Definition 20 in the appendix provides the formal definitions.[14]

5.2. Comparisons

The program behind our formalization of shreds and effective shred-operations was: (i) to see if we could partially open up black boxes in some complexity-theoretically interesting fashion, (ii) to formalize a natural class of functionals based on these partially open black boxes that would be analogous to the polynomial-time effective operations, and (iii) to see if we could *provably* compare this new class of functionals to the basic polynomial-time functionals. Proposition 16 delivers this comparison.

Recall from Definition 14 that χ denotes the computation system associated with φ and Φ, our standard, Turing machine-based acceptable programming system and associated complexity measure. Also recall from that definition that $\mathcal{S}_\chi$ denotes the collection $\{\, \chi_p : p \in \omega \,\}$, which is roughly the collection of all (sanitized) Turing machine traces.

Proposition 16. *The following classes of functionals all correspond on $\mathcal{R}$:*

(a) The polynomial-time effective shred-operations with respect to $\mathcal{S}_\chi$.

(b) The polynomial-time clocked effective shred-operations with respect to $\mathcal{S}_\chi$.

(c) The basic polynomial-time functionals.[15]

[12] Parameterizing these notions with respect to the class $\mathcal{S}$ is a bit irritating, but an analogous parameterization (with respect to the acceptable programming system φ) is implicit in the notion of effective operation.

[13] Note if RUN(x) returns $\langle s(x, \mathbf{0}^{2^k}), \mathbf{0}^{2^k}\rangle$, then $\kappa s(x) \leq 2^k < 2 \cdot \kappa s(x)$, and, hence, $s(x, \mathbf{0}^{2^k}) = \iota s(x)$. Also note that RUN$(x)$ can be computed with only $1 + \log_2 \kappa s(x)$ calls to s. Moreover, assuming that, for all x, $\kappa s(x) \geq |x|$, the total time to compute RUN(x) is $\Theta(\kappa s(x) \log_2 \kappa s(x))$.

[14] Note that a polynomial-time *clocked* effective shred-operation has domain $\mathcal{P}\mathcal{R} \times \omega$, whereas a polynomial-time effective shred-operation has just $\mathcal{R} \times \omega$ as its domain, but a polynomial-time clocked effective shred-operation when restricted to $\mathcal{R} \times \omega$ corresponds to some polynomial-time effective shred-operation.

Also note that this clocking scheme is based on *sequential* calls to RUN, and this causes problems for shred oracles outside of $\text{tot}(\mathcal{S}_{\text{all}})$, e.g., it is easy to show that $\mathbf{OR}'_{\|} = \lambda s, x.\mathbf{OR}_{\|}(\iota s, x)$ (where $\mathbf{OR}_{\|}$ is as in (1)) fails to be a polynomial-time clocked shred functional. We can rectify this problem by generalizing our clocking notion as follows. Replace the subroutine RUN above with a subroutine RACE that takes a nonempty list $\vec{x}$ of elements of ω. A call to RACE on $\vec{x}$ results in: (i) $\langle s(x, \mathbf{0}^{2^k}), x, \mathbf{0}^{2^k}\rangle$, where k is the least number such that, for some x' in $\vec{x}$, $s(x', \mathbf{0}^{2^k}) \neq \star$ and x is the least such x'; or (ii) the calling OTM going undefined if no such k exists. Everything else can go as before. Clearly, $\mathbf{OR}'_{\|}$ can be computed by such clocked machines. We call the class of functionals determined by such (extensional) machines the *polynomial-time parallel-clocked effective shred-operations.*

[15] We can also add: *(d) The polynomial-time parallel-clocked effective shred-operations.*

The correspondence of (a) and (b) is the shred analog of Seth's Theorem 8 and the correspondence of (a) and (c) is the polynomial-time/shred analog of the Kreisel-Lacombe-Shoenfield Theorem (Theorem 3). Thus, one *can* partially open up black boxes and obtain something like the classical recursion theoretic correspondences of Theorems 2 and 3.[16]

6. Further Problems

The results of section 4 indicate that the original question of whether a polynomial-time analog of the Kreisel-Lacombe-Shoenfield Theorem holds is, like P = NP?, yet another technically intractable complexity theoretic problem. How important a problem this is, I can't say. Some of the key problems in contemporary programming languages center around the issue of information hiding, e.g., data structures that hide their implementations. My guess is that some of these programming language problems can be sharpened to the point where they become interesting complexity theoretic questions, and in such a context the polynomial-time KLS problem may play an interesting role.

Section 5 showed that by weakening the notion of effective operation one can obtain a polynomial-time analog of the Kreisel-Lacombe-Shoenfield Theorem. One obvious question left open is whether one can replace shreds with real traces and still obtain the equivalence. My guess is "yes." Computations can be very coy about what they are up to until very late in their course, e.g., they can run lots of unrelated subcomputations and leave until the very end which of these subcomputations are used to produce the final result of the main computation.

In the theory of programming languages, the effectively continuous functionals (Definition 1(c)) and their generalizations play a much greater role than the partial recursive functionals (Definition 1(b)). So, another set of problems concerns the polynomial-time parallel-clocked effective shred-operations of footnote 14. These functionals in some respects resemble the effective continuous functionals. How close is this resemblance? Can one obtain a language characterization of this class along the lines of Cook and Kapron's characterizations of the basic feasible functionals [CK90] or of Plotkin's PCF [Plo77]? Can one define a more general class of "polynomial-time effective shred-operations" on $\mathcal{PR} \times \omega$ (as opposed to just $\mathcal{R} \times \omega$) and compare these to the parallel-clocked ones?

I am curious to see if the ideas and results presented above are useful in extending type-2 complexity beyond (and below) polynomial-time to develop a general, machine-based theory of type-2 computational complexity. Additionally, I am hopeful that shreds, or something like them, will be of help in sorting out useful machine models for computation at above type 2. Functional programming techniques like continuations and monads are naturally set at type 3. It would be great fun to have good type-3 machine models so as to subject algorithms built through such techniques to complexity analyses.[17]

[16] If one replaces $\mathcal{S}_\chi$ with either $\mathcal{S}_{\text{all}}$ or $\mathcal{S}_{\text{comp}} = \{s \in \mathcal{S}_{\text{all}} : s \text{ is computable}\}$ in Proposition 16, the analogous results are true and simpler to prove. However, consider M, an OTM computing a polynomial-time effective shred-operation with respect to $\mathcal{S}_\chi$. M has as its oracle something that reasonably represents the computations of an *actual* TM program. Hence, the polynomial-time effective shred-operations with respect to $\mathcal{S}_\chi$ correspond much more closely to polynomial-time effective operations than the polynomial-time effective shred-operations with respect to either $\mathcal{S}_{\text{all}}$ or $\mathcal{S}_{\text{comp}}$. There are difficulties with the use of $\mathcal{S}_\chi$ in Proposition 16. The current proof of the proposition makes shameless use of special complexity properties of the TM model, and it is not clear how far the proposition generalizes to apply to a broad class of computation systems. Remark 21 in the appendix discusses these problems in more detail.

[17] Recently Seth [Set94] gave an extension of the Kapron and Cook Theorem (Theorem 6 above) to all finite types.

A. Technical Details

A.1. Background Proofs and Results

Here we present proofs of the Kreisel-Lacombe-Shoenfield Theorem and Seth's Clocking Theorem, as ideas from these arguments play important parts in the proofs of our Propositions 12 and 16. The proof of Theorem 3 is based on the one given in Rogers [Rog67]. The proof of Theorem 8 is a considerable simplification of the one given in [Set92]. (Seth's original proof had other goals besides simply establishing Theorem 8.)

Theorem 3 (The Kreisel-Lacombe-Shoenfield Theorem [KLS57]). *Each total effective operation on $\mathcal{R}$ corresponds on $\mathcal{R}$ to an effective continuous functional.*

Proof. *Notation:* For each σ, a function of finite domain, define $\widehat{\sigma} = \lambda x.[\sigma(x)$, if $\sigma(x)\downarrow$; 0, otherwise]. We also define, for each p and s, $\varphi_p^s = \lambda x.[\varphi_p(x)$, if x and $\Phi_p(x) \leq s$; $\uparrow$, otherwise]. We assume a fixed canonical indexing of functions of finite domain; $\sigma \leq s$ means σ's index is $\leq s$.

Suppose p is such that φ_p determines a total effective operation $\mathbf{F}$ on $\mathcal{R}$. We construct an effectively continuous functional $\mathbf{G}$: $(\omega \rightharpoonup \omega) \times \omega \rightharpoonup \omega$ such that $\mathbf{F}$ and $\mathbf{G}$ correspond on $\mathcal{R}$.

By the parametric recursion theorem [Rog67, RC94], there is a recursive function r such that, for all i, $\varphi_{r(i)} = \cup_{s\geq 0} \varphi_{r(i),s}$, where, for all i and s,

$$\varphi_{r(i),s} = \lambda x. \begin{cases} \varphi_i^s(x), & \text{if (i): } \varphi_p^s(i)\uparrow \text{ or } \varphi_p^s(i)\downarrow \neq \varphi_p^s(r(i)); \\ \varphi_i^w(x), & \text{if (ii): } \varphi_p^s(i)\downarrow = \varphi_p^s(r(i))\downarrow; \text{ and,} \\ & \quad \text{for } w = \max(\Phi_p(i), \Phi_p(r(i))) \\ & \quad \text{and, for all } \sigma \leq s \text{ with } \varphi_i^w \subseteq \sigma, \\ & \quad \mathbf{F}(\widehat{\sigma}) = \varphi_p(i); \\ \widehat{\sigma_0}(x), & \text{if (iii): otherwise, where } \sigma_0 \text{ is the} \\ & \quad \text{least } \sigma \supseteq \varphi_i^w \text{ with } \mathbf{F}(\widehat{\sigma}) \neq \varphi_p(i). \end{cases} \tag{3}$$

We observe the following about the construction.

1. For all i and s, $\varphi_{r(i),s} \subseteq \varphi_{r(i),s+1}$.
2. For all i and s, clause (iii) never holds in (3) because otherwise $\varphi_{r(i)} = \widehat{\sigma_0}$, and hence $\mathbf{F}(\varphi_{r(i)}) = \varphi_p(r(i)) = \varphi_p(i) \neq \mathbf{F}(\widehat{\sigma_0})$, a contradiction.
3. If φ_i is total, then for all but finitely many s, clause (ii) holds in (3), because otherwise clause (i) would hold for all but finitely many s, and hence $\varphi_{r(i)} = \varphi_i$, but then $\mathbf{F}(\varphi_i) = \varphi_p(i) \neq \varphi_p(r(i)) = \mathbf{F}(\varphi_{r(i)})$, a contradiction.

Define

$$\mathbf{G} = \lambda\alpha. \begin{cases} \varphi_p(j), & \text{if } \langle i, s\rangle \text{ is the least number such that} \\ & \quad \text{(a) } \Phi_p(i), \Phi_p(r(i)) \leq s \text{ and (b) for} \\ & \quad w = \Phi_p(r(i)), \ \varphi_i^w \subseteq \alpha, \text{ and where} \\ & \quad j \text{ is a } \varphi\text{-program for } \widehat{\varphi_i^w}; \\ \uparrow, & \text{if no such } \langle i, s\rangle \text{ exists.} \end{cases}$$

Using the three observations, a straightforward argument shows that $\mathbf{F}$ and $\mathbf{G}$ coincide on $\mathcal{R}$. □

Theorem 8 (Seth's Clocking Theorem [Set92]). *The class of basic polynomial-time clocked functionals correspond on $(\omega \to \omega)$ to the class basic polynomial-time functionals.*

Proof Sketch. We need to show:

(a) Each $\mathbf{M}_{q,m}$ computes a basic polynomial-time functional.

(b) Each basic polynomial-time functional is computed by some $\mathbf{M}_{q,m}$.

Proof of (a). Clearly, $q^{[m]}(|f|, |a|)$ bounds the number of steps simulated by $\mathbf{M}_{q,m}$ on input (f, a). The overhead of the clocking machinations blows up the run time by no more than a quadratic amount. Hence there exists a constant c such that $c \cdot \left(q^{[m]}(|f|, |a|)\right)^2$ bounds the total run time of $\mathbf{M}_{q,m}$ on input (f, a). Therefore, (a) follows.

Proof of (b). Suppose $\mathbf{M}$ is an OTM that computes Γ with time bound given by $\mathbf{q}$, where $\mathbf{q}$ is a second-order polynomial over g and x. We may assume without loss of generality that $\mathbf{q} = q^{[m]}$ for some first-order polynomial q and $m \in \omega$. We show that, for all $f: \omega \to \omega$ and all $a \in \omega$, $\mathbf{M}(f, a) = \mathbf{M}_{q,m}(f, a)$. Fix f and a. Let t_* be the number of steps taken by $\mathbf{M}$ on input (f, a). By hypothesis, $t_* \leq \mathbf{q}(|f|, |a|)$. For each $t \leq t_*$, define

$$f_t = \lambda x. \begin{cases} f(x), & \text{if } \mathbf{M} \text{ on input } (f, a) \text{ makes the query} \\ & \quad f(x) = ? \text{ within its first } t \text{ steps;} \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

A straightforward induction argument shows that, for each $t \leq t_*$:

(i) After t steps, $\mathbf{M}_{q,m}$ on input (f, a) has $\mathbf{q}(|f_t|, |a|)$ as the contents of $\mathbf{x}[m]$.

(ii) $\mathbf{M}(f_t, a) = \mathbf{M}_{q,m}(f_t, a)$.

Hence $\mathbf{M}(f_{t_*}, a) = \mathbf{M}_{q,m}(f_{t_*}, a)$. But by (4) it follows that $\mathbf{M}(f, a) = \mathbf{M}(f_{t_*}, a)$. Therefore, $\mathbf{M}(f, a) = \mathbf{M}_{q,m}(f, a)$, as claimed. Hence (b) follows. □

We state without proof the following lemma about Turing machines of which we make use.

Lemma 17 (The Patching Lemma). *For each φ-program p and for each finite function σ, there is an other φ-program p_σ such that, for all x,*

$$\varphi_{p_\sigma}(x) = \begin{cases} \sigma(x), & \text{if } x \in \text{domain}(\sigma); \\ \varphi_p(x), & \text{otherwise.} \end{cases} \qquad \Phi_{p_\sigma}(x) = \begin{cases} |x| + |\sigma(x)|, & \text{if } x \in \text{domain}(\sigma); \\ \Phi_p(x), & \text{otherwise.} \end{cases}$$

A.2. Proofs and Proof Sketches for the Results of Sections 4 and 5

The following proposition implies that, for any non-trivial, polynomial-time analog of effective operation, the "polynomial" upper bound of the cost of computing such thing needs to depend, in part, on the costs of running the program argument on various values during the course of the computation.

Proposition 18. *Suppose that φ_i determines a total effective operation on $\mathcal{R}$ and that there is a second-order polynomial $\mathbf{q}$ such that, for all p with φ_p total and all x, $\Phi_i(p, x)$ runs within $\mathbf{q}(|\varphi_p|, \max(|p|, |x|))$ time. Then, there is a polynomial-time computable $f: \omega \to \omega$ such that, for all p with φ_p total and all x, $\varphi_i(p, x) = f(x)$.*

Proof. The argument is a variant of a standard proof of Rice's Theorem. (See Case's proof in either of [DW83, DSW94].) Suppose by way of contradiction that

(5) there are p_0, p_1, and x such that φ_{p_0} and φ_{p_1} are total and $\varphi_i(p_0, x) \neq \varphi_i(p_1, x)$.

If $\varphi_{p_0} = \varphi_{p_1}$, then, clearly, α is not extensional in its first argument, a contradiction. So, suppose $\varphi_{p_0} \neq \varphi_{p_1}$. Without loss of generality, assume that $\mathbf{q}$ is monotone. Let $g = \lambda n.\, \max(|\varphi_{p_0}|(n), |\varphi_{p_1}|(n))$. By the recursion theorem there is a φ-program e such that, for all y,

$$\varphi_e(y) = \begin{cases} 0, & \text{if (i): } \Phi_i(e,x) > \mathbf{q}(g, \max(|e|, |x|)); \\ \varphi_{p_1}(y), & \text{if (ii): } \textit{not} \text{ (i) and } \varphi_i(e,x) = \varphi_i(p_0,x); \\ \varphi_{p_0}(y), & \text{if (iii): otherwise.} \end{cases} \tag{6}$$

Note that the clauses (i), (ii), and (iii) in (6) do not depend on y. Also note that, whichever of clauses (i), (ii), and (iii) hold, φ_e is total, and hence $\varphi_i(e,x)\downarrow$. We consider the following three exhaustive cases.

Case 1: Clause (i) in (6) holds. Then, $\varphi_e = \lambda y.0$. Hence by our hypotheses on i, $\Phi_i(e,x) \leq \mathbf{q}(|\varphi_e|, \max(|e|, |x|) \leq \mathbf{q}(g, \max(|e|, |x|))$, which contradicts clause (i).

Case 2: Clause (ii) in (6) holds. Then, $\varphi_e = \varphi_{p_1}$, hence $\varphi_i(e,x) = \varphi_i(p_1,x)$, by φ_i's extensionality. But, since $\varphi_i(p_1,x) \neq \varphi_i(p_0,x)$, this contradicts clause (ii).

Case 3: Clause (iii) in (6) holds. Then, $\varphi_e = \varphi_{p_0}$, hence $\varphi_i(e,x) = \varphi_i(p_0,x)$, by φ_i's extensionality. But, in this clause, clause (ii) should hold, which contradicts clause (iii).

Thus, since (5) fails, we have that, for all p_0 and p_1 with φ_{p_0} and φ_{p_1} total and all x, $\varphi_i(p_0,x) = \varphi_i(p_1,x)$. Let $p_\star$ be a φ-program for $\lambda x.0$. Then $f = \lambda x.\varphi_i(p_\star, x)$ is as required. □

Recall from Section 4.2 that, for each $\alpha\colon \omega \rightharpoonup \omega$ and $x \in \omega$,

$$\Gamma_0(\alpha,x) =_{\text{def}} \begin{cases} \uparrow, & \text{if (i): for some } y \in \{\mathbf{0},\mathbf{1}\}^{|x|},\ \alpha(y)\uparrow; \\ 1, & \text{if (ii): } \textit{not} \text{ (i) and } (\exists y \in \{\mathbf{0},\mathbf{1}\}^{|x|})[\,\alpha(y) \text{ is odd }]; \\ 0, & \text{if (iii): } \textit{not} \text{ (i) and } (\forall y \in \{\mathbf{0},\mathbf{1}\}^{|x|})[\,\alpha(y) \text{ is even }]. \end{cases}$$

Proposition 11.

(a) The restriction of Γ_0 to $(\omega \to \omega) \times \omega$ is not basic feasible. Moreover, there is an honest, exponential-time computable function g such that, for each q and m, there is an n for which $\Gamma_0(g, \mathbf{0}^n) \neq \mathbf{M}_{q,m}(g, \mathbf{0}^n)$.

(b) If P = NP, *then Γ_0 is a polynomial-time clocked effective operation.*

Proof Sketch of Proposition 11. The proof of part (a) is a standard oracle construction, where in this case, g is the oracle constructed.

For part (b), first consider the predicates:

$$\begin{aligned} P(p, \mathbf{0}^m, \mathbf{0}^n) &\equiv (\exists x \in \{\mathbf{0},\mathbf{1}\}^m)\left[\,\Phi_p(x) > n\,\right]. \\ Q(p, \mathbf{0}^n, x_0, x_1) &\equiv \left[|x_0| = |x_1| \;\&\; x_0 \leq x_1 \;\&\; (\exists x : x_0 \leq x \leq x_1)[\Phi_p(x) > n]\right]. \\ R(p, \mathbf{0}^m, \mathbf{0}^n) &\equiv (\exists x \in \{\mathbf{0},\mathbf{1}\}^m)\left[\,\Phi_p(x) \leq n \text{ and } \varphi_p(x) \text{ is odd}\right]. \end{aligned}$$

Clearly, P, Q, and R both are nondeterministically decidable in time polynomial in the lengths of their arguments. Hence, since P = NP, P, Q, and R are each in polynomial-time. Fix polynomial-time decision procedures for P, Q, and R, and let q_1 be a polynomial such that q_1(the max over the lengths of all the arguments) > the run times of these procedures. Let ψ be the partial recursive function computed by the following informally stated program.

Program for ψ.

Input p, x.

Set $m \leftarrow |x|$ and $n \leftarrow \max(|p \qquad\qquad |, m)$.

While $P(p, \mathbf{0}^m, \mathbf{0}^n)$ **do**

Use Q in a binary search to find an $x_0 \in \{\mathbf{0}, \mathbf{1}\}^m$ such that $\Phi_p(x_0) > n$.

Set $n \leftarrow 2 \cdot \Phi_p(x_0)$. *(Note: if $\Phi_p(x_0)\uparrow$, then the program diverges.)*

End while

If $R(p, \mathbf{0}^m, \mathbf{0}^n)$ then output 1 else output 0.

End program

Clearly, $\psi = \lambda p, x.\Gamma_0(\varphi_p, x)$. We argue that one can insert an appropriate clocking mechanism into the above program so as to make it equivalent to an $M_{q,m}$. Note that throughout the course of execution of the program we have that $n \geq \max(|p|, m)$. Now, evaluating $P(p, \mathbf{0}^m, \mathbf{0}^n)$ in the while test takes $q_1(n)$ time, and using Q in the binary search takes $c \cdot n \cdot q_1(n)$ time for some constant c. Determining $\Phi_p(x_0)$ can be done through a normal query to UNIV, and once we know the value of $\Phi_p(x_0)$, we can bound the cost of the next iteration by $q_2(\Phi_p(x_0))$, where q_2 is an appropriate polynomial such that, for all n, $q_2(n) > c \cdot (2n + 2) \cdot q_1(2n)$. Thus, with appropriate choice of q, it is clear that we can transform the above program into an equivalent $M_{q,1}$. Hence, part (b) follows. □

Remark 19. The only use of the P = NP hypothesis in the argument for Proposition 11(b) is in making the predicates P, Q, and R polynomial-time decidable. One can exploit this to convert the argument into a construction of an oracle relative to which: (i) Γ_0 is again a polynomial-time clocked effective operation, and (ii) P $\neq$ NP. Hence P = NP is not equivalent to the failure of the correspondence on $\mathcal{R}$ of the polynomial-time clocked effective operations the basic polynomial-time functionals. ◇

We delay discussing the proof of Proposition 12 until after we've shown Proposition 16, since the ideas developed in the latter argument are used in the former.

Proposition 13.

(a) If P = NP, *then the polynomial-time effective operations correspond on $\mathcal{R}$ to the polynomial-time clocked effective operations.*

(b) There is an oracle relative to which there is a polynomial-time effective operation that fails to correspond on $\mathcal{R}$ to any polynomial-time clocked effective operation.

Proof Sketch. *Part (a).* Suppose M determines a total polynomial-time effective operation on $\mathcal{R}$ and that $\mathbf{q}$ is a second-order polynomial such that, for all p with φ_p total and all x, $M(p, x)$ runs in time $\mathbf{q}(\overline{\Phi}_p, |x|)$. Without loss of generality, suppose $\mathbf{q} = q^{[k]}$ for some polynomial q and $k \in \omega$. Using the P = NP hypothesis and the technique of the proof of Proposition 11, construct a TM M' that (i) clockably computes $\mathbf{q}(\overline{\Phi}_p, |x|)$ and then (ii) runs M on p and x. A straightforward argument shows that M' corresponds to a $M_{q,m}$.

Part (b). We build an oracle A, relative to which the functional

$$\Gamma^A = \lambda f \in \mathcal{R}, x \in \omega. \max\{ f(y) : |y| = |x| \} \tag{7}$$

is a polynomial-time effective operation, but fails to correspond on $\mathcal{R}$ to any polynomial-time clocked effective operation. This A is built by means of a simple coding and witness-hiding construction.

Let $\langle \varphi_p^B \rangle_{p \in \omega}$ and $\langle \Phi_p^B \rangle_{p \in \omega}$ respectively denote the standard relativizations of $\langle \varphi_p \rangle_{p \in \omega}$ and $\langle \Phi_p \rangle_{p \in \omega}$ to an oracle B. Let $\langle M_i \rangle_{i \in \omega}$ be an effective indexing of the relativized $M_{q,m}$'s (where,

in this case, the relativization is to a set oracle). Also, for each i, let $\mathbf{q}_i = q^{[m]}$ for the appropriate q and m for M_i.

We first determine A on strings of the form $\langle p+1, x, y\rangle$ $(p, x, y \in \omega)$ as follows.

$$A(\langle p+1, x, y\rangle) = 0, \quad \text{if } x \notin \mathbf{0}^* \text{ or } y \notin \mathbf{0}^*.$$

$$A(\langle p+1, \mathbf{0}^m, \mathbf{0}^n\rangle) = \begin{cases} 1, & \text{if either (a) } \overline{\Phi}_p^A(m) > n \text{ or (b) for some } j \geq 0, \\ & \quad n = \overline{\Phi}_p^A(m) + 2j + 1 \text{ and } \mathbf{1} \text{ is the } j^{\text{th}} \text{ bit of the} \\ & \quad \text{binary expansion of } \max\{\varphi_p^A(x) : |x| = m\} \text{ is } \mathbf{1} \text{ or} \\ & \quad \text{(c) for some } j \geq 0,\ n = \overline{\Phi}_p^A(m) + 2j \text{ and} \\ & \quad j \leq |\max\{\varphi_p^A(x) : |x| = m\}|; \\ 0, & \text{otherwise.} \end{cases}$$

Note that, for each p, m, and n, since $|\langle p+1, \mathbf{0}^m, \mathbf{0}^n\rangle| > n$, there is no φ^A-program p that, on input x with $|x| \leq n$, can in its first n steps query A on $\langle p+1, \mathbf{0}^m, \mathbf{0}^n\rangle$. From this definition of A on $\{\langle p+1, x, y\rangle : p, x, y \in \omega\}$, it is a simple argument that, no matter how we determine the rest of A, the functional of (7) is a polynomial-time effective operation.

Now let p_0 be a relativized program and $c \in \omega$ such that, for all oracles X and $x \in \omega$,

$$\varphi_{p_0}^X(x) = (\mu \mathbf{0}^m)[X(0, x, \mathbf{0}^m) = 0]. \tag{8}$$

$$\Phi_{p_0}^X(x) \leq c \cdot (\max(|x|, |\varphi_{p_0}^X(x)|))^2. \tag{9}$$

We define A on strings of the form $\langle 0, x, y\rangle$ to make $\varphi_{p_0}^A$ total and to guarantee that, for each M_i^A, there is an x such that $M_i^A(p_0, x) \neq \Gamma^A(\varphi_{p_0}, x)$.

We determine A on $\{\langle 0, x, y\rangle : x, y \in \omega\}$ in stages. Define $A_0 : \omega \to \{0, 1, \star\}$ as follows. For each p, x, and y, define $A_0(\langle p+1, x, y\rangle) = A(\langle p+1, x, y\rangle)$ and $A_0(\langle 0, x, y\rangle) = \star$. Also define $n_0 = 0$. At each stage i, determine $A_{i+1} : \omega \to \{0, 1, \star\}$ and $n_{i+1} \in \omega$. For all i, the A_i's and n_i's will satisfy:

1. For all z, if $A_i(z) \neq \star$, then $A_{i+1}(z) = A_i(z)$.
2. $A_i^{-1}(\{0, 1\}) = \{\langle p, x, y\rangle : p > 0 \text{ or } |x| < n_i\}$.
3. There is an x such that $M_i^{A_{i+1}}(p_0, x) \neq \Gamma(\varphi_{p_0}, x)$, where in the computation of $M_i^{A_{i+1}}(p_0, x)$ all the queries to A_{i+1} are in $A_{i+1}^{-1}(\{0, 1\})$.

The construction also arranges that $\lim_{i\to\infty} n_i = \infty$ so that defining $A = \lambda z.\ \lim_{i\to\infty} A_i(z)$ yields an A as required.

Stage i.

Let $A' = \lambda z.[A_i(z)$, if $A_i(z) \neq \star$; 0, otherwise$]$.

Let n be the least number $\geq n_i$ such that

(a) $\overline{\Phi}_{p_0}^{A'}(n) \leq n^3$ and *(b)* $\mathbf{q}_i(\lambda z.z^3, n) < 2^n$. *(By (9) such an n must exist.)*

Let $k = |M_i^{A'}(p_0, \mathbf{0}^n)|$.

Let x' be the least number of length n such that in the computation of $M_i^{A'}(p_0, \mathbf{0}^n)$, no number of the form $\langle 0, x', y\rangle$ is queried. *(By (a) and (b), such an x' must exist.)*

Set $n_{i+1} = 1 + \max(k, \mathbf{q}_i(\lambda z.z^3, n))$ and, for each p, x, and $y \in \omega$, define A_{i+1} as follows:

$$A_{i+1}(\langle p, x, y\rangle) = A(\langle p, x, y\rangle), \quad p > 0.$$

$$A_{i+1}(\langle 0, x, y\rangle) = \begin{cases} A_i(\langle 0, x, y\rangle), & \text{if (i): } A_i(\langle 0, x, y\rangle) \neq \star; \\ 1, & \text{if (ii): not (i) and } x = x' \text{ and } y = \mathbf{0}^j \\ & \quad \text{for some } j \leq k; \\ 0, & \text{if (iii): not (i) and not (ii) and } |x| < n_{i+1}; \\ \star, & \text{otherwise.} \end{cases}$$

End stage i.

A straightforward argument shows that the A_i's and n_i's are as required. □

Definition 20.

(a) Let RUN denote a fixed OTM subroutine such that when an OTM calls RUN on $x \in \omega$, the result is

$$\begin{cases} \langle s(x, \mathbf{0}^{2^k}), \mathbf{0}^{2^k}\rangle \text{ is returned}, & \text{if } k = (\mu k')[s(x, \mathbf{0}^{2^{k'}}) \neq \star]; \text{ and} \\ \text{the calling OTM goes undefined}, & \text{if no such } k \text{ exists.} \end{cases}$$

(b) A *special oracle Turing machine* (SOTM) M is an oracle Turing machine defined as follows. M takes an oracle $s \in \mathcal{S}_{\text{all}}$ and an input $x \in \omega$. M includes RUN as a subroutine and obeys the same constraints M (of Definition 10) does with respect to UNIV.

(c) Suppose M is a SOTM, q is a polynomial over two variables, and $m \in \omega$. Let $\mathsf{M}_{q,m}$ be the SOTM that, on input (s, a), operates as follows. $\mathsf{M}_{q,m}$ maintains a counter clock and two arrays x[0..m] and y[0..$m-1$]. $\mathsf{M}_{q,m}$ maintains the invariants: x[0] $= q(0, |a|)$; x[$i+1$] $= q(\mathsf{y}[i], |a|)$, $i < m$; and

$$\mathsf{y}[i] = \max\left(\left\{ \mathbf{0}^{2^k} : \begin{array}{l} |x| \leq \mathsf{x}[i] \text{ and the call RUN}(x) \text{ was} \\ \text{made and returned } \langle s(x, \mathbf{0}^{2^k}), \mathbf{0}^{2^k}\rangle \end{array} \right\}\right), \quad i < m.$$

On start up $\mathsf{M}_{q,m}$ initializes clock, x, and y exactly as $\mathsf{M}_{q,m}$ does. Then, $\mathsf{M}_{q,m}$ simulates M step by step on input (s, a). For each step of M simulated:

- If the step of M just simulated was the last step of an execution of RUN, then, if necessary, $\mathsf{M}_{q,m}$ recomputes the x[i]'s and y[i]'s to re-establish the invariants.
- If, in the step of M just simulated, M halts with output y, then $\mathsf{M}_{q,m}$ outputs y and halts.
- If M did not halt in the step just simulated, then the value of clock is increased by 1 and, if clock $<$ x[m] *or* if the step of M just simulated was part of an execution of RUN, then the simulation continues; otherwise, $\mathsf{M}_{q,m}$ outputs 0 and halts.

(d) Suppose $\mathcal{S} \subseteq \mathcal{S}_{\text{all}}$ is such that $\mathcal{PR} = \iota\mathcal{S}$. Γ: $\mathcal{PR} \times \omega \rightharpoonup \omega$ is a *polynomial-time clocked effective shred-operation* with respect to $\mathcal{S}$ if and only if there exists an extensional (wrt $\mathcal{S}$) $\mathsf{M}_{q,m}$ that determines Γ as per Definition 15. ◇

Proposition 16. *The following classes of functionals all correspond on* $\mathcal{R}$*:*

(a) The polynomial-time effective shred-operations with respect to $\mathcal{S}_\chi$*.*

(b) The polynomial-time clocked effective shred-operations with respect to $\mathcal{S}_\chi$*.*

(c) The basic polynomial-time functionals.

Proof. *Convention:* All the clocked and unclocked polynomial-time effective shred-operations mentioned in this proof will be with respect to $\mathcal{S}_\chi$. So, to cut the clutter, the "with respect to $\mathcal{S}_\chi$" clause will be dropped below.

Now, when all the functionals concerned are restricted to $\mathcal{R}$ we clearly have that

$$\text{(a)} \supseteq \text{(b)} \supseteq \text{(c)}.$$

Our job is then to show that the two containments reverse. Claims 1 and 2 below correspond to (a) $\subseteq$ (b) and (b) $\subseteq$ (c), respectively. We first consider Claim 1.

Claim 1. *Each polynomial-time effective shred-operation corresponds on $\mathcal{R}$ to some polynomial-time clocked effective shred-operation.*

Our argument for Claim 1 is a modification of the proof given for Theorem 8.

Suppose Γ: $\mathcal{R}\times\omega \to \omega$ is a polynomial-time effective shred-operation. Suppose also that M is an extensional OTM that determines Γ and that $\mathbf{q}$ is a second-order polynomial such that, for all $s \in \mathcal{S}_\chi$ and $a \in \omega$, $\mathsf{M}(s, a)$ runs within $\mathbf{q}(\overline{\kappa s}, |a|)$ time. We sketch two other OTMs, M_0 and M_1, that both run a step-by-step simulation of M on an input a and an oracle to be determined.

Program for M_0

Input $(a, \mathbf{0}^k)$ with oracle s'.

Set $\mathsf{step} \leftarrow 0$.

Go through a step-by-step simulation M on input a. After each step, add one to step. Each M step that is *not* an oracle call is faithfully carried out. Each oracle query, $s(x, \mathbf{0}^k) =?$, is simulated as follows.

Condition 1. $k < |x| + 1$.
Make $\star$ the answer to the query in the simulation of M.

Condition 2. $k \geq \max(|x| + 1, \mathsf{step})$.
Call $\mathsf{RUN}(x)$ to determine $\iota s'(x)$ and give $\iota s'(x)$ as the answer to M's query.

Condition 3. $|x| + 1 \leq k < \mathsf{step}$.
Give 0 as the answer to M's query.

If, in the course of the simulation, M halts with output y, then output y and halt.

End program

The program for M_1 is the same as for M_0 except that M_1 takes only a as input and omits using step (and hence omits Condition 3).

It follows from Lemma 17 that, in any run of M_0 or M_1, the answers fed to the simulation of M are consistent with some actual χ_p. Hence, if $\iota s'$ is total, the simulations of M by M_0 and M_1 eventually halt. Moreover, it also follows that, for each a and s', for all sufficiently large k,

- when M_0 is run on input $(a, \mathbf{0}^k)$ Condition 3 never occurs, and
- the use M makes of the (simulated) oracle is identical in the simulations both M_0 on input $(a, \mathbf{0}^k)$ and oracle s' and M_1 on input a and oracle s'.

Fix some a and $p \in \omega$. We argue that:

(10) $$\mathsf{M}_1(\chi_p, a) = \mathsf{M}(\chi_p, a).$$

(11) M_1 on input a and oracle χ_p can be clocked (as per Definition 20) by a second-order $\mathbf{q}_0$, where $\mathbf{q}_0$ is independent of p and a.

Let σ be the finite function such that

$$\sigma = \left\{ (x, \varphi_p(x)) : \begin{array}{l}\text{when } \mathsf{M}_1 \text{ is run on input } a \text{ and oracle } \chi_p, \text{ the simu-} \\ \text{lation of } \mathsf{M} \text{ queries } s(x, \mathbf{0}^k) \text{ for some } k \geq |x| + 1\end{array} \right\}.$$

By Lemma 17 there is a φ-program p_σ such that:

$$\varphi_{p_\sigma}(x) = \begin{cases} \sigma(x), & \text{if } x \in \text{domain}(\sigma); \\ \varphi_p(x), & \text{otherwise.} \end{cases} \qquad \Phi_{p_\sigma}(x) = \begin{cases} |x| + |\sigma(x)|, & \text{if } x \in \text{domain}(\sigma); \\ \Phi_p(x), & \text{otherwise.} \end{cases}$$

We observe that M's use of the oracle is identical in both M_1's simulation M and M on input a and oracle χ_{p_σ}. It follows from this observation that $\mathsf{M}_1(\chi_p, a) = \mathsf{M}(\chi_{p_\sigma}, a)$. Since $\iota\chi_p = \varphi_p = \varphi_{p_\sigma} = \iota\chi_{p_\sigma}$, we have by M's extensionality that $\mathsf{M}(\chi_{p_\sigma}, a) = \mathsf{M}(\chi_p, a)$. Therefore, (10) follows.

Now, for each k, let σ_k be the finite function

$$\sigma_k = \left\{ (x, \varphi_p(x)) : \begin{array}{l}\text{when } \mathsf{M}_0 \text{ is run on input } (a, \mathbf{0}^k) \text{ and oracle } \chi_p, \text{ the sim-} \\ \text{ulation of } \mathsf{M} \text{ queries } s(x, \mathbf{0}^k) \text{ for some } k \geq |x| + 1\end{array} \right\}.$$

By Lemma 17, for each k, there is a φ-program z_k such that:

$$\varphi_{z_k}(x) = \begin{cases} \sigma_k(x), & \text{if } x \in \text{domain}(\sigma_k); \\ 0, & \text{otherwise.} \end{cases} \qquad \Phi_{z_k}(x) = \begin{cases} |x| + |\sigma_k(x)|, & \text{if } x \in \text{domain}(\sigma_k); \\ |x| + 1, & \text{otherwise.} \end{cases}$$

It follows that, for each k, the run time of M on input a and oracle χ_{z_k} is bounded by $\mathbf{q}(\overline{\kappa\chi_{z_k}}, |a|)$, and the run time of M_0 on input $(a, \mathbf{0}^k)$ and oracle χ_p is bounded above by

$$c \cdot \left((\max\{\kappa\chi_p(x) : x \in \text{domain}(\sigma_k)\}) \cdot \mathbf{q}(\overline{\kappa\chi_{z_k}}, |a|)\right)^m, \tag{12}$$

where c and m are numbers independent of a, k, and p. By a little second-order algebra, there is a second-order polynomial $\mathbf{q}_0$, independent of a and p, such that, $\mathbf{q}_0(\overline{\kappa\chi_{p_k}}, |a|)$ is an upper bound on (12). Hence, for each k, the time used by M_0 (on input $(a, \mathbf{0}^k)$ and oracle χ_p) in simulating the first k steps of M is bounded above by $\mathbf{q}_0(\overline{\kappa\chi_{p_k}}, |a|)$. By our observations it follows that, for each k, the state of the simulation of M after k steps is identical in both M_0 on input $(a, \mathbf{0}^k)$ and oracle χ_{z_k} and M_1 on input a and oracle χ_p. Therefore, we have that, for each k, the time used by M_1 (on input a and oracle χ_p) in simulating the first k steps of M is also bounded above by $\mathbf{q}_0(\overline{\kappa\chi_{p_k}}, |a|)$. Thus, by this observation, the definitions of M_0, M_1, and the σ_k's, and the clocking scheme of Definition 20, (11) follows. Claim 1 thus follows.

Claim 2. *Each polynomial-time clocked effective shred-operation corresponds on $\mathcal{R}$ to some basic polynomial-time functional.*

Suppose $\Gamma: \mathcal{R} \times \omega \to \omega$ is a polynomial-time clocked effective shred-operation. Suppose also that $\mathsf{M}_{q,m}$ is an extensional OTM that determines Γ. By the argument for Claim 1, we may assume without loss of generality that the only way that $\mathsf{M}_{q,m}$ queries its oracle is through calls to the RUN subroutine. Consider the OTM $\mathbf{M}$ whose program is sketched below.

Program for M

Input a with oracle g.

Go through a step-by-step simulation $\mathsf{M}_{q,m}$ on input a. Each $\mathsf{M}_{q,m}$ step that is *not* an oracle call is faithfully carried out. Each oracle query, $s(x, \mathbf{0}^k) =?$, is simulated as follows.

Condition 1. $k < |x| + |g(x)|$.
Make $\star$ the answer to the query in the simulation of $\mathsf{M}_{q,m}$.
Condition 2. $k \geq |x| + |g(x)|$.
Give $g(x)$ as the answer to $\mathsf{M}_{q,m}$'s query.

If, in the course of the simulation, $\mathsf{M}_{q,m}$ halts with output y, then output y and halt.

End program

By similar argument to that given in Claim 1, we have that, for each $a \in \omega$ and each $p \in \omega$ with $\varphi_p \in \mathcal{R}$:

(13) $\mathbf{M}(\varphi_p, a) = \mathsf{M}(\chi_p, a)$.

(14) There is a second-order polynomial $\mathbf{q}$, independent of a and p, such that the run time of $\mathbf{M}$ on input a and oracle φ_p is bounded above by $\mathbf{q}(|\varphi_p|, |a|)$.

Claim 2 thus follows. □ **Proposition 16**

Remark 21. The strong dependence on Lemma 17 in the above argument is very unsatisfying, but it is indicative of deeper problems. Consider an acceptable programming system φ' and associated complexity measure Φ' which are polynomially related to our standard, Turing machine-based φ and Φ, but which are such that, for each p such that $\varphi_p(0)\downarrow$, one can somehow reconstruct p from $\Phi_p(0)$. Let χ' be the computation system associated with the φ' and Φ'. Any attempt to prove the analog of Proposition 16 will run into the difficulties of Section 4. What is probably needed for the analog of Proposition 16 to be true for a given computation system χ'' is some strong, complexity theoretic version of Rice's Theorem to hold for the φ'' and Φ'' with which χ'' is associated. ◇

Finally we discuss the proof of

Proposition 12. *There is an oracle relative to which every polynomial-time effective operation corresponds on $\mathcal{R}$ to some polynomial-time functional.*

In this version of the paper we omit the proof of this theorem. The proof, which is more involved than any of the above, is an amalgam of the proofs of Theorem 3 and Proposition 16. The oracle enters to help swarms of programs perform diagonalizations like those of (3) and to hide information about certain programs from polynomial-time effective operations.

B. Acknowledgements

Thanks to Jin Yi Cai, John Case, Robert Irwin, Bruce Kapron, Stuart Kurtz, Ken Regan, and Alan Selman for discussing this work at various stages of its development. Special thanks to Neil Jones and the TOPPS group at DIKU for letting me spend a week at DIKU to discuss my ideas and to Peter O'Hearn for many, many discussions on these and related topics.

References

[CK90] S. Cook and B. Kapron, *Characterizations of the basic feasible functions of finite type*, Feasible Mathematics: A Mathematical Sciences Institute Workshop, (S. Buss and P. Scott, eds.), Birkhäuser, 1990, pp. 71–95.

[Cob65] A. Cobham, *The intrinsic computational difficulty of functions*, Proc. Int. Conf. Logic, Methodology and Philosophy (Y. Bar Hillel, ed.), North-Holland, 1965, pp. 24–30.

[Coo91] S. Cook, *Computability and complexity of higher type functions*, Logic from Computer Science (Y.N. Moschovakis, ed.), Springer-Verlag, 1991, pp. 51–72.

[CU89] S. Cook and A. Urquhart, *Functional interpretations of feasibly constructive arithmetic*, Proc. of the 21st Ann. ACM Symp. on Theory of Computing, 1989, pp. 107–112.

[DSW94] M. Davis, R. Sigal, and E. Weyuker, *Computability, complexity, and languages*, Academic Press, 1994, second edition of [DW83].

[DW83] M. Davis and E. Weyuker, *Computability, complexity, and languages*, Academic Press, 1983.

[Fri58] R. Friedberg, *Un contre-example relatif aux fonctionnelles récursives*, Comptes Rendus Hebdomadaires des séances de l'Académie des Sciences **247** (1958), 852–854.

[KC91] B. Kapron and S. Cook, *A new characterization of Mehlhorn's polynomial time functionals*, Proc. of the 32nd Ann. IEEE Symp. Found. of Comp. Sci., 1991, pp. 342–347.

[KLS57] G. Kreisel, D. Lacombe, and J Shoenfield, *Partial recursive functionals and effective operations*, Constructivity in Mathematics: Proceedings of the Colloquium held at Amsterdam (A. Heyting, ed.), North-Holland, 1957, pp. 195–207.

[Meh76] K. Mehlhorn, *Polynomial and abstract subrecursive classes*, Journal of Computer and System Science **12** (1976), 147–178.

[MS55] J. Myhill and J. Shepherdson, *Effective operations on partial recursive functions*, Zeitschrift für Mathematische Logik und Grundlagen der Mathematik **1** (1955), 310–317.

[Odi89] P. Odifreddi, *Classical recursion theory*, North-Holland, 1989.

[Plo77] G. Plotkin, *Lcf considered as a programming language*, Theoretical Computer Science **5** (1977), 223–255.

[RC94] J. Royer and J. Case, *Subrecursive programming systems: Complexity & succinctness*, Birkhäuser, 1994.

[Rog67] H. Rogers, *Theory of recursive functions and effective computability*, McGraw-Hill, 1967, reprinted, MIT Press, 1987.

[Sco75] D. Scott, *Lambda calculus and recursion theory*, Proceedings of the Third Scandinavian Logic Symposium (S. Kanger, ed.), North-Holland, 1975, pp. 154–193.

[Set92] A. Seth, *There is no recursive axiomatization for feasible functionals of type 2*, Seventh Annual IEEE Symposium on Logic in Computer Science, 1992, pp. 286–295.

[Set94] A. Seth, *Complexity theory of higher type functionals*, Ph.D. thesis, University of Bombay, 1994.