

On the Computational Complexity of Longley's H Functional

James S. Royer*

30 November 2001

Abstract

Longley [Lon98b] discovered a functional H that, when added to PCF, yields a language that computes exactly SR, the *sequentially realizable functionals* of van Oosten [vO99]. We show that if $P \neq NP$, then the computational complexity of H (and of similar SR-functionals) is inherently infeasible.

§1. Introduction

The *sequentially realizable functionals* (denoted SR) is a class of “sequentially computable” higher-type functionals. This class, which includes the PCF-computable functionals along with elements that fail to be Scott-continuous¹, has quite strong and natural mathematical properties [Lon98a, Lon98b, Lon99, vO99]. Longley [Lon98b] discovered an SR-functional H that, when added to the language PCF, yields a language PCF+ H that computes exactly SR. Longley hesitated to recommend PCF+ H as the basis of a practical programming language because the only known ways of computing H seemed to involve high computational costs. This paper confirms Longley's doubts. We show that if $P \neq NP$, then the computational complexity of H is inherently infeasible. Moreover, we show that any SR-functional that determines the same sort of retract as H also suffers the same sort of inherent infeasibility.

OUTLINE OF THE PAPER. Section 2 sketches some of the details of the SR functionals and the dialog games that define them. The section ends with a statement of Longley's result and a discussion of some of its implications.

*Dept. of Elec. Eng. & Computer Science, Syracuse University, Syracuse, NY 13244, USA. Email: royer@ecs.syr.edu. Research supported in part by NSF grants CCR-9522987 and CCR-0098198. A preliminary version of this paper was presented at the *Second International Workshop on Implicit Computational Complexity*, 29–30 June 2000 at UC/Santa Barbara.

¹For an example, see Ψ^n , defined in Equation (4) below.

Section 3 provides a bit more technical background that leads to a sketch of the core of Longley's construction for H and an illustration of some problematic features of this construction. The basics of the complexity theoretic framework for our hardness results and the statement of these results make up Section 4. Section 5 contains the proofs of these results and some additional complexity-theoretic background. Finally, we state our conclusions and some open problems in Section 6. Excluding Section 5, the paper presumes only a basic understanding of the theory of computation. Section 5 requires some elementary complexity theory, but anyone who has seen an NP-hardness proof before should be able to follow the arguments.

§2. Some Background on SR

Our presentation of the SR functionals is based on the work of van Oosten [vO99] (who was the first to formalize SR along the lines sketched below) and Longley [Lon98b]. The focus in this paper is on the effective version of SR, and to avoid decorating everything in sight with an “*eff*” sub- or superscript, we have skewed van Oosten's and Longley's notation to reflect this focus.

The SR functionals are based on a simple game between two partial functions $r, g: \mathbb{N} \rightarrow \mathbb{N}$. In this game r successively asks g what its value is at particular arguments and the game terminates when, if ever, r decides it has enough information about g and outputs an answer instead of a question. The game can fail to terminate either because r queries g on an argument on which g is undefined, or because r (on receiving a certain response from g) goes undefined, or because r and g go on chatting forever. At each r -move in one of these games, r needs to know the history of the conversation up to the present in order to determine its next action. These histories will be encoded by sequences of natural numbers. We state a few conventions on sequences and their encodings before formalizing these games.

CONVENTIONS ON SEQUENCES. We let α, β , and γ range over $\text{Seq}(\mathbb{N})$, the collection of finite sequences of natural numbers. The sequence x_0, \dots, x_{j-1} is denoted by $[x_0, \dots, x_{j-1}]$; so, $[]$ denotes the empty sequence. For each $\alpha = [x_0, \dots, x_{j-1}]$ and $w \in \mathbb{N}$, let $\alpha;w$ denote $[x_0, \dots, x_{j-1}, w]$. For each $\alpha = [x_0, \dots, x_{j-1}]$ and $i \leq j$, let $\alpha^{<i}$ denote $[x_0, \dots, x_{i-1}]$; so, $\alpha^{<0} = []$. For each α let $\langle \alpha \rangle$ be a polynomial-time encoding of α as an element of \mathbb{N} . (See Section 5 for details.) We write $\langle \rangle$ for $\langle [] \rangle$, $\langle x_0, \dots, x_{j-1} \rangle$ for $\langle [x_0, \dots, x_{j-1}] \rangle$, and $r\langle \alpha \rangle$ for $r(\langle \alpha \rangle)$. Moreover, a defining equation of the form “ $r\langle \alpha \rangle = E(\alpha)$ ” is understood as meaning “ $r(y) = E(\alpha)$, if $y = \langle \alpha \rangle$ for a particular α ; $r(y) = E([])$, otherwise.”

FORMALIZING DIALOG GAMES. For each $x \in \mathbb{N}$, define $?x = 2x$ and $!x = 2x + 1$. We call even numbers *questions* and odd numbers *answers*. We follow

Longley [Lon98b] and define $\text{play}: (\mathbb{N} \rightarrow \mathbb{N}) \times (\mathbb{N} \rightarrow \mathbb{N}) \times \text{Seq}(\mathbb{N}) \rightarrow \mathbb{N}$ by:

$$\text{play}(r, g, \alpha) = \begin{cases} \text{play}(r, g, (\alpha; y)), & \text{if } r\langle\alpha\rangle\downarrow = ?x \text{ and } g(x)\downarrow = y; \\ x, & \text{if } r\langle\alpha\rangle\downarrow = !x; \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

We also define $(\cdot \mid \cdot): (\mathbb{N} \rightarrow \mathbb{N}) \times (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$ and $(\cdot \bullet \cdot): (\mathbb{N} \rightarrow \mathbb{N}) \times (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$ by:

$$\begin{aligned} r \mid g &= \text{play}(r, g, []). \\ r \bullet g &= \lambda x. \text{play}(r, g, [x]). \end{aligned}$$

An intuitive interpretation of $\text{play}(r, g, \alpha)$ is that it is the result of the dialog game between r and g that has reached the point where α consists of the sequence of g 's prior answers to r 's questions and it is r 's turn to make a move. The definition of \bullet bends this interpretation a bit as the initial $[x]$ is used to parameterize the game. In these games, r can be thought of as representing a decision tree based on the values returned by g . There is a very direct connection to the standard notion of type-2 sequential computability from recursion theory: given two total $g_1, g_2: \mathbb{N} \rightarrow \mathbb{N}$, it is straightforward that $g_1 \leq_T g_2$ if and only if, for some partial recursive r , $g_1 = r \bullet g_2$. (See Rogers [Rog67] or Odifreddi [Odi81] for a discussion of Turing-reductions between functions.) Van Oosten [vO99] established that $(\mathbb{N} \rightarrow \mathbb{N}, \bullet)$ is an untyped partial combinatory algebra. Longley [Lon98b] showed the corresponding result for (\mathcal{PR}, \bullet) , where \mathcal{PR} denotes the set of partial recursive functions of type $\mathbb{N} \rightarrow \mathbb{N}$.

We refer to the sequence of interactions between r and g implied by the definition of $r \mid g$ as the *play* of r against g or, alternatively, as the *play* of “ $r \mid g$ ” where the quotes in “ $r \mid g$ ” serve as a pedantic reminder that we are referring to the expression $r \mid g$ and not to the (possible) value denoted by this expression.

BUILDING FUNCTIONALS FROM GAMES. The SR functionals of type $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$ are those of the form “ $g \mapsto r \mid g$ ” where for the full type structure we let r and g be any partial functions over \mathbb{N} and for the effective version we restrict the functions to \mathcal{PR} . The r in “ $g \mapsto r \mid g$ ” is called a *realizer* for the functional defined. To construct functionals of other types at type-level 2 and above, we can parameterize (as in the definition of \bullet) or impose extensionality constraints (see the definition of $\llbracket \bar{3} \rrbracket$ below).

IRREDUNDANCY AND SEMI-IRREDUNDANCY. In studying the H functional, it is helpful to consider two restricted classes of realizers for the type-2 functionals. The first of these two notions is:

DEFINITION 1. An $r \in \mathcal{PR}$ is *irredundant* if and only if, for all $\alpha = [x_0, \dots, x_{k-1}]$ such that $r\langle\alpha\rangle\downarrow$,

- (i) $r\langle\alpha^{<i}\rangle\downarrow$ for each $i < k$, and
- (ii) in the sequence $r\langle\alpha^{<0}\rangle, \dots, r\langle\alpha^{<k}\rangle$, a particular question can appear only once and an answer can appear only as the last element. \diamond

Intuitively, an irredundant r is a function that is precisely tailored for the role of a realizer of a type $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$ functional. This is because: (i) in a play of “ $r \mid g$ ” a repeated question adds nothing to the information r is gathering about g , and (ii) an irredundant r is undefined on all arguments that cannot be reached in the course of the play of “ $r \mid g$ ” for some g . We will see in Lemmas 8 and 9 below how irredundant realizers can be used to provide “finite bases” for convergent computations for type-3 SR functionals.

It turns out that any $F: \mathcal{PR} \rightarrow \mathbb{N}$ realized by some $r \in \mathcal{PR}$ is also realized by some irredundant $r' \in \mathcal{PR}$. In fact, there is a “semantics preserving” retraction of \mathcal{PR} onto the set of irredundant elements of \mathcal{PR} in the following sense.

LEMMA 2 (LONGLEY [LON98B]). *There is an $\mathbf{irr} \in \mathcal{PR}$ such that, for all $r \in \mathcal{PR}$,*

- (a) for each $g \in \mathcal{PR}$, $(\mathbf{irr} \bullet r) \mid g = r \mid g$ (i.e., $\mathbf{irr} \bullet r$ and r realize the same $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$ functional),
- (b) $\mathbf{irr} \bullet r$ is an irredundant element of \mathcal{PR} , and
- (c) $\mathbf{irr} \bullet r = r$ when r is irredundant.

REMARK 3 (RETRACTIONS). The old-fashioned definition of a retraction is that it is a map of a space onto a subspace that leaves everything in the subspace fixed. By parts (b) and (c) of the above lemma, $\lambda r.(\mathbf{irr} \bullet r)$ is a retraction (in this sense) of \mathcal{PR} onto the collection of irredundant elements of \mathcal{PR} . The modern definition of a retraction is that it is simply the left inverse of an arrow. If ι is the inclusion map of the class of irredundant elements of \mathcal{PR} into \mathcal{PR} , then (b) and (c) imply that $\lambda r.(\mathbf{irr} \bullet r)$ is the left inverse of ι . \diamond

Longley also introduced a slightly broader class of realizers than the irredundant ones.

DEFINITION 4. An $r \in \mathcal{PR}$ is *semi-irredundant* if and only if, for all $\alpha = [x_0, \dots, x_{k-1}]$ such that $r\langle\alpha\rangle\downarrow$,

- (i) $r\langle\alpha^{<i}\rangle\downarrow$ for each $i < k$, and
- (ii) in the sequence $r\langle\alpha^{<0}\rangle, \dots, r\langle\alpha^{<k}\rangle$, a particular question can appear only once. \diamond

Let $\mathcal{PR}_{\text{s.i.}}$ denote the collection of semi-irredundant elements of \mathcal{PR} .

Semi-irredundant realizers are not the least bit interesting *except* that in [Lon98b] they help to solve a technical problem in establishing some properties of H ; they thus figure prominently in Longley's setup for defining H . The aforementioned technical problem will not concern us in this paper, but we shall follow Longley in the use of semi-irredundant realizers for type- $\bar{2}$ functionals (discussed next). To simplify matters a bit we shall, unlike Longley, require that realizers for type- $\bar{3}$ functionals be semi-irredundant as well.

INTERPRETING THE TYPES. The *pure* types $(\bar{0}, \bar{1}, \bar{2}, \dots)$ over a particular base type \mathbf{b} are a subset of the simple types over \mathbf{b} , where $\bar{0}$ is \mathbf{b} and $\overline{n+1}$ is $\bar{n} \rightarrow \mathbf{b}$. We are concerned with interpreting the pure types $\bar{0}$, $\bar{1}$, $\bar{2}$, and $\bar{3}$ for the effective semantics of the SR functionals. We take $\llbracket \bar{0} \rrbracket \cong \mathbb{N}$ and $\llbracket \bar{1} \rrbracket \cong \mathcal{PR}$. Each $g \in \llbracket \bar{1} \rrbracket$ is its own (unique) realizer. Following Longley, we take $\llbracket \bar{2} \rrbracket$ to be the collection of all $F: \llbracket \bar{1} \rrbracket \rightarrow \mathbb{N}$ such that, for some $r \in \mathcal{PR}_{\text{s.i.}}$,

$$F(g) = r \mid g, \quad \text{for each } g \in \llbracket \bar{1} \rrbracket \quad (= \mathcal{PR}). \quad (1)$$

For each such F , we define the set of realizers of F (written $\|F\|_{\bar{2}}$) to be the collection of all $r \in \mathcal{PR}_{\text{s.i.}}$ such that (1) holds. We write $r \sim_2 r'$ if and only if $r, r' \in \|F\|_{\bar{2}}$ for some F . Clearly, \sim_2 is an equivalence relation on $\mathcal{PR}_{\text{s.i.}}$. Finally, we take $\llbracket \bar{3} \rrbracket$ to be the collection of all $\Psi: \llbracket \bar{2} \rrbracket \rightarrow \mathbb{N}$ such that, for some $r \in \mathcal{PR}_{\text{s.i.}}$,

$$\Psi(F) = r \mid g_F, \quad \text{for each } F \in \llbracket \bar{2} \rrbracket \text{ and each } g_F \in \|F\|_{\bar{2}}. \quad (2)$$

For each such Ψ , we define the set of realizers of Ψ (written $\|\Psi\|_{\bar{3}}$) to be the collection of all $r \in \mathcal{PR}_{\text{s.i.}}$ such that (2) holds. We note that $r \in \|\Psi\|_{\bar{3}}$ for some Ψ if and only if $r \in \mathcal{PR}_{\text{s.i.}}$ and is \sim_2 -*extensional*: for all g, g' with $g \sim_2 g'$, we have $r \mid g = r \mid g'$. We write $r \sim_3 r'$ if and only if $r, r' \in \|\Psi\|_{\bar{3}}$ for some Ψ . Clearly, \sim_3 is a partial equivalence relation on $\mathcal{PR}_{\text{s.i.}}$.

Conventions: When F is understood to be in $\llbracket \bar{2} \rrbracket$, we write $\|F\|$ in place of $\|F\|_{\bar{2}}$, and similarly when Ψ is understood to be in $\llbracket \bar{3} \rrbracket$. We define $\|\bar{2}\| = \bigcup \{ \|F\| : F \in \llbracket \bar{2} \rrbracket \}$ and $\|\bar{3}\| = \bigcup \{ \|\Psi\| : \Psi \in \llbracket \bar{3} \rrbracket \}$. Note that $\|\bar{3}\|$ is a proper subset of $\|\bar{2}\|$ and that $\|\bar{2}\|$ is just another name for $\mathcal{PR}_{\text{s.i.}}$.

REMARK 5. In (2), r plays against a realizer for F instead of F itself since in this setting the only way we have of naming F is with its realizers. Being a member of $\|\bar{3}\|$ means meeting some very stringent conditions since each F has infinitely many realizers and an r satisfying (2) must end-up with the same result when playing against each of them. These conditions are so strong that it is not obvious that there is a recursive presentation of realizers for all of $\llbracket \bar{3} \rrbracket$ (and hence, not obvious that one can build a programming language for SR).

One of the interesting things about the H functional is that it provides a way to construct such a recursive presentation. However, the stringent conditions do end up causing a different sort of trouble as we shall see in Section 4. \diamond

LONGLEY'S FUNCTIONAL. The H functional involves a "retraction" of $\llbracket \bar{2} \rrbracket$ onto $\llbracket \bar{3} \rrbracket$ in the following sense.

DEFINITION 6. We say that $t \in \mathcal{PR}$ realizes a retraction if and only if, for each $r \in \llbracket \bar{2} \rrbracket$,

- (i) $(t \bullet r) \in \llbracket \bar{3} \rrbracket$, and
- (ii) $(t \bullet r) \sim_3 r$ when $r \in \llbracket \bar{3} \rrbracket$. \diamond

Thus, t realizes a retraction if and only if $\lambda r.t \bullet r$ is a "retraction modulo \sim_3 " of $\llbracket \bar{2} \rrbracket$ onto $\llbracket \bar{3} \rrbracket$.

THEOREM 7 (LONGLEY [LON98B]). *There is an h that realizes a retraction.*

The h Longley constructs realizes his functional $H: \llbracket \bar{2} \rrbracket \rightarrow \llbracket \bar{3} \rrbracket$, which is a retraction in the proper sense of the term.² Longley's papers [Lon98b, Lon99, Lon98a] have nice treatments of the structural part of the story. Our emphasis here is on the mapping on realizers asserted by the theorem. Part of the computational import of the theorem is clear: From this h one can construct a recursive presentation of an *adequate* collection of realizers for $\llbracket \bar{3} \rrbracket$, where here adequate means that each $\Psi \in \llbracket \bar{3} \rrbracket$ has at least one element of $\llbracket \Psi \rrbracket$ in the recursive presentation. (Actually, the h Longley constructs does much better than "adequate.") Longley shows how, from h , one can uniformly construct analogous retracts onto each other simple type and so one obtains recursive presentations of adequate sets of realizers for each these types. PCF+H determines exactly SR because, roughly, through h one can obtain "names" for all, and only, the elements of SR.

§3. Longley's Construction

To sketch Longley's construction for Theorem 7, we first need to note a few facts about type- $\bar{2}$ SR functionals and their realizers. For $f, g \in \mathbb{N} \rightarrow \mathbb{N}$, we write $f \subset g$ if and only if the graph of f is a proper subset of the graph of g . Let ν and ξ (and decorated versions of them) range over elements of $\mathbb{N} \rightarrow \mathbb{N}$ with finite domains. Given $G \in \llbracket \bar{2} \rrbracket$ and ν , we say that ν is G -minimal if and only if $G(\nu) \downarrow$ and, for each $\nu' \subset \nu$, we have that $G(\nu') \uparrow$. Clearly, for each $g \in \mathcal{PR}$ such that $G(g) \downarrow$, there is a unique G -minimal $\nu \subseteq g$. For each

²Note that $\llbracket \bar{3} \rrbracket \rightarrow \llbracket \bar{2} \rrbracket \cong \llbracket \bar{3} \rrbracket \rightarrow \llbracket \bar{1} \rrbracket \rightarrow \llbracket \bar{0} \rrbracket$. Let $I: \llbracket \bar{3} \rrbracket \rightarrow \llbracket \bar{2} \rrbracket$ be the SR functional such that $I(\Psi)(f) = \Psi(\lambda g.f | g)$. Then it turns out that $H \circ I = \mathbf{id}_{\llbracket \bar{3} \rrbracket}$.

$G \in \llbracket \bar{2} \rrbracket$, the *trace* of G (written $tr(G)$) is $\{(\nu, G(\nu)) \mid \nu \text{ is } G\text{-minimal}\}$. Each $G \in \llbracket \bar{2} \rrbracket$ is completely determined by its trace. We write $G \sqsubseteq_{tr} G'$ when $tr(G) \subseteq tr(G')$. We say that a $G \in \llbracket \bar{2} \rrbracket$ is *finite* if and only if $tr(G)$ is finite. Given (the graph of) a finite ξ , one can effectively decide if $\xi \in \llbracket G \rrbracket$ for some G ; if so, this G is finite and one can construct $tr(G)$ from ξ . For $G \in \llbracket \bar{2} \rrbracket$, let $\llbracket G \rrbracket^{\mathcal{I}}$ be the collection of irredundant realizers for G . The next lemma shows that, for finite G , $\llbracket G \rrbracket^{\mathcal{I}}$ is equivalent to $tr(G)$ and nicely represents all of $\llbracket G \rrbracket$.

LEMMA 8 (LONGLEY [LON98B]). *Suppose $G_0 \in \llbracket \bar{2} \rrbracket$ is finite. Then:*

- (a) $\llbracket G_0 \rrbracket^{\mathcal{I}}$ is finite and each element of $\llbracket G_0 \rrbracket^{\mathcal{I}}$ has a finite domain.
- (b) Each $r \in \llbracket G_0 \rrbracket$ extends exactly one element of $\llbracket G_0 \rrbracket^{\mathcal{I}}$.
- (c) From $tr(G_0)$ one can effectively compute a finite list of finite graphs making up the elements of $\llbracket G_0 \rrbracket^{\mathcal{I}}$.

The finiteness properties of $\llbracket G_0 \rrbracket^{\mathcal{I}}$ in part (a) depend crucially on the properties of irredundancy. The next lemma uses these sets of irredundant realizers of finite G_0 's to show that convergent type- $\bar{3}$ computations have strong finite bases.

LEMMA 9 (LONGLEY [LON98B]). *Suppose $\Psi \in \llbracket \bar{3} \rrbracket$, $G \in \llbracket \bar{2} \rrbracket$, and $z \in \mathbb{N}$. Then, $\Psi(G)\downarrow = z$ if and only if, for each $r \in \llbracket \Psi \rrbracket$, there is a finite G_0 with $G_0 \sqsubseteq_{tr} G$ such that*

- (i) for each $\xi \in \llbracket G_0 \rrbracket^{\mathcal{I}}$, $(r \mid \xi)\downarrow = z$, and
- (ii) each $g \in \llbracket G \rrbracket$ extends exactly one element of $\llbracket G_0 \rrbracket^{\mathcal{I}}$.

Now we are in a position to give some details of the construction of h . For our purposes it is enough to consider, for $r, g \in \mathcal{PR}_{s.i.}$, the play of $(h \bullet r)$ against g . This is sketched in Figure 1. (See Longley [Lon98b] for details of the \bullet -play of h against r .) The **For**-loop is the most computationally expensive part of this sketch, and expensive it certainly is. Here is an example that illustrates the problem.

EXAMPLE 10. Fix an $n \in \mathbb{N}$. Let G^n be the type- $\bar{2}$ functional given by:

$$G^n(f) = \begin{cases} 0, & \text{if, for each } x < n, f(x) = 0; \\ \text{undefined,} & \text{otherwise.} \end{cases}$$

For each $i \in \mathbb{N}$, let ζ_i be a length- i sequence of 0's. Let π range over permutations of $\{0, \dots, n-1\}$. For each π let $g_\pi^n: \mathbb{N} \rightarrow \mathbb{N}$ be given by:

$$g_\pi^n\langle \alpha \rangle = \begin{cases} ?\pi(i), & \text{if } \alpha = \zeta_i \text{ for } i < n; \\ !0, & \text{if } \alpha = \zeta_n; \\ \text{undefined,} & \text{otherwise.} \end{cases} \quad (3)$$

Sketch of the computation for the play of $(h \bullet r)$ against g .

Play out “ $r \mid g$ ” until, if ever, a result $z \in \mathbb{N}$ is produced.

Let ν_0 be the finite part of g used in the play of r against g .

If ν_0 realizes some finite member of $\llbracket \bar{2} \rrbracket$,

then let G_0 denote this finite member,

else diverge.

For each $\nu \in \|G_0\|^{\mathcal{I}}$ **do**: (* By Lemma 8, $\|G_0\|^{\mathcal{I}}$ is finite. *)

 Play out “ $r \mid \nu$ ” until, if ever, a result $z' \in \mathbb{N}$ is produced.

If $z \neq z'$ or some $x \in \text{dom}(\nu)$ is not queried in this play,

then diverge.

End for

Output $!z$.

End sketch

Figure 1: The Core of Longley's Construction

Note that $\|G^n\|^{\mathcal{I}}$ consists exactly of the g_π^n 's. Let $r_n \in \mathcal{PR}_{\text{s.i.}}$ be given by:

$$r_n \langle \alpha \rangle = \begin{cases} \langle \zeta_i \rangle, & \text{if } \alpha = [?x_0, \dots, ?x_k] \text{ where } k < n \text{ and} \\ & x_0, \dots, x_k \text{ are } < n \text{ and pairwise distinct;} \\ !0, & \text{if } \alpha = [?x_0, \dots, ?x_{n-1}, !0] \text{ where } x_0, \dots, x_{n-1} \\ & \text{are } < n \text{ and pairwise distinct;} \\ \text{undefined,} & \text{otherwise.} \end{cases}$$

Clearly, r_n is \sim_2 -extensional and realizes $\Psi^n \in \llbracket \bar{3} \rrbracket$ where

$$\Psi^n(F) = \begin{cases} 0, & \text{if } G^n \sqsubseteq_{\text{tr}} F; \\ \text{undefined,} & \text{otherwise.} \end{cases} \quad (4)$$

Thus, $(h \bullet r_n) \sim_3 r_n$. Moreover, for a given π , the plays of both “ $(h \bullet r_n) \mid g_\pi^n$ ” and “ $r_n \mid g_\pi^n$ ” are identical. However, if one reads the above construction straightforwardly, then after the last move of g_π^n and before the last move of $h \bullet r_n$ in the play of “ $(h \bullet r_n) \mid g_\pi^n$,” the computation specified for $(h \bullet r_n)$ internally checks that $(r_n \mid g_\pi^n) \downarrow = 0$ for each of the $n!$ many $g_\pi^n \in \|G^n\|^{\mathcal{I}}$ and only then outputs the answer $!0$. So intuitively, the time to play out “ $(h \bullet r_n) \mid g_\pi^n$ ” should be around $n!$ times longer than the time to play out “ $r_n \mid g_\pi^n$.” \diamond

The job of a faithful implementer of H is thus to try to find a way to be true to the semantics, but to avoid the potentially horrendous computation at the end of the sketch. To see if there is such a way, we first need to formalize a bit about implementations.

§4. Implementations and Their Costs

Our focus now shifts from realizers to *programs* for realizers and the costs associated with these programs. A few standard notions from computability and complexity theory suffice for our rudimentary formalization of these notions.

Conventions: Our basic model of computation will be RAMs that compute over \mathbb{N} and that have the logarithmic cost model [Jon97, Pap94]. For each $n > 0$, let $\langle \varphi_i^n \rangle_{i \in \mathbb{N}}$ be a RAM-based, acceptable indexing [Jon97, Rog67] of the partial recursive functions of type $\mathbb{N}^n \rightarrow \mathbb{N}$ and, for each i and $\vec{x} \in \mathbb{N}^n$, let $\Phi_i^n(\vec{x})$ denote the cost of the computation of the i -th, n -argument RAM on input \vec{x} . (If the i -th, n -argument RAM diverges on \vec{x} , then $\Phi_i^n(\vec{x}) = \infty$.) We write φ_i for φ_i^1 and Φ_i for Φ_i^1 .

Now we formalize our notion of the cost of a play.

DEFINITION 11. For each $i \in \mathbb{N}$ and $g \in \mathcal{PR}$, let

$$C(i, g) = \begin{cases} \sum_{j \leq k} \Phi_i \langle \alpha^{< j} \rangle, & \text{if (i) } (\varphi_i \mid g) \downarrow, \text{ where } \alpha \text{ is the final sequence presented to } \varphi_i \text{ in the play of "}\varphi_i \mid g\text{" and } k \text{ is the length of } \alpha; \\ \infty, & \text{if (ii) the play of "}\varphi_i \mid g\text{" is infinite or else } \varphi_i \text{ diverges at some point in this play;} \\ \text{undefined,} & \text{(iii) otherwise.} \end{cases}$$

We call $C(i, g)$ the *cost* of playing program i against g . ◇

In a play of program i against g , the responses of g are presumably computed by some program, but $C(i, g)$ measures only the work done by program i in this play; g is treated as an oracle. Note that condition (ii) corresponds to an observable divergence (the program does infinitely much work) and condition (iii) corresponds to an unobservable divergence (the divergence occurs offstage in the course of an oracle query).

DEFINITION 12. Suppose t realizes a retraction. An *instantiation* of t is a partial recursive $inst: \mathbb{N} \rightarrow \mathbb{N}$ such that, for all i such that φ_i is semi-irredundant, $inst(i) \downarrow$ and $\varphi_{inst(i)} = t \bullet \varphi_i$. ◇

Suppose $inst_h$ is an instantiation of h . Then $inst_h$ is a mapping over programs and so long as it satisfies $\varphi_{inst_h(i)} = h \bullet \varphi_i$ for each i with $\varphi_i \in \|\bar{2}\|$, it need not follow Longley's construction at all. In particular, it has access to the complete text of the program computing the realizer φ_i and it is apparently free to use nonsequential methods.

DEFINITION 13. Suppose t realizes a retraction and $inst$ is an instantiation of t . We say that $inst$ has *polynomial overhead* if and only if $inst$ is polynomially-time computable and there is a polynomial p such that, for all $i \in \mathbb{N}$ with $\varphi_i \in \|\bar{3}\|$ and all $g \in \mathcal{PR}_{s.i.}$, we have

$$C(inst(i), g) \leq p \left(\max \left\{ C(i, \hat{g}) \mid \begin{array}{l} \hat{g} = g \text{ or } \hat{g} \text{ is irre-} \\ \text{dundant and } \hat{g} \sim_2 g \end{array} \right\} \right).$$

(By Lemma 9, the above maximization is over finitely many things, provided $C(i, g) \downarrow < \infty$.) \diamond

Suppose again that $inst_h$ is an instantiation of h . If $inst_h$ has polynomial overhead, then the intuition is that in $inst_h(i)$'s computations in playing against g , the program is allowed to play out " $\varphi_i \mid g$ " and also " $\varphi_i \mid \hat{g}$ " for "polynomially-many" irredundant $\hat{g} \sim_2 g$, but not more than that. Note that the maximalization on the right-hand side is over g and *all possible* irredundant \hat{g} with $\hat{g} \sim_2 g$, that is, all of the possible candidate \hat{g} 's from which $inst_h(i)$ might choose the "polynomially-many" to play against φ_i .

It is clear that an instantiation of h based directly on Longley's construction would *not* have polynomial overhead. Unfortunately, it is unlikely that there are any polynomial overhead instantiations of h as shown by:

THEOREM 14. *If $P \neq NP$, then h has no polynomial overhead instantiations.*

The proof of this can be found in the next section. The argument makes strong use of the fact that the play of $(h \bullet r)$ against g diverges whenever there is any inconsistency detected in the various plays of r against the variants of g . There are various ways to modify Longley's construction to make it more convergent. However, some analysis shows that one of the less obvious purposes of the construction's **For**-loop is to search for a divergence in the " $r \mid \nu$ " plays. To see what we mean by this, suppose that $(r \mid \nu) \uparrow$ for some $\nu \in \|G_0\|^{\mathcal{I}}$. Then the play of " $(h \bullet r) \mid \nu$ " goes undefined on the first line of the construction's sketch. Hence, since $\nu \sim_2 g$, we have to make sure that if there is such a ν , then $(h \bullet r) \mid g$ is also undefined to preserve \sim_2 -extensionality. An analysis of the proof of Lemma 18 will show that any realizer of a retract has to reflect this search. Based on this we obtain

THEOREM 15. *Suppose t realizes a retraction. If $P \neq NP$, then t has no polynomial overhead instantiations.*

§5. Complexity-Theoretic Details

REPRESENTING NUMBERS AND SEQUENCES. We make use of two representations of the natural numbers: dyadic and unary. Each element of \mathbb{N} is identified with its dyadic representation over $\{\mathbf{0}, \mathbf{1}\}$, i.e., $0 \equiv \epsilon$, $1 \equiv \mathbf{0}$, $2 \equiv \mathbf{1}$, $3 \equiv \mathbf{00}$, etc. We will freely pun between $x \in \mathbb{N}$ as a number and a $\mathbf{0}$ - $\mathbf{1}$ -string. Each element of ω (a copy of \mathbb{N}) is identified with its unary representation over $\{\mathbf{0}\}$, i.e., $0 \equiv \epsilon$, $1 \equiv \mathbf{0}$, $2 \equiv \mathbf{0}^2$, $3 \equiv \mathbf{0}^3$, etc. We treat ω as the subset of \mathbb{N} of numbers with a dyadic representation of the form $\mathbf{0}^x$. We use the elements of \mathbb{N} as natural number (and string) values to be computed over, and we use the elements of ω as tallies to represents lengths, run times, and, generally, anything that corresponds to a size measurement.

For each $x, y \in \mathbb{N}$, let $x \diamond y$ denote the concatenation of (the dyadic representations of) x and y . For each $x \in \mathbb{N}$, let $|x|$ be the length of the dyadic representation of x and let $E(x) = \mathbf{1}^{|x|} \mathbf{0} \diamond x$. (Note that the image of E in $\{\mathbf{0}, \mathbf{1}\}^*$ is a collection of prefix codes [LV97, section 1.4].) We code each α as a distinct element of \mathbb{N} by:

$$\langle \alpha \rangle = \begin{cases} \epsilon, & \text{if } \alpha = []; \\ E(x_0) \diamond \dots \diamond E(x_{k-1}), & \text{if } \alpha = [x_0, \dots, x_{k-1}] \text{ where } k > 0. \end{cases}$$

By convention, any element of \mathbb{N} that is not of the form $\langle x_1, \dots, x_j \rangle$ is considered as coding $[]$. It is clear from our definition of $\langle \cdot \rangle$ that concatenations, projections, and so on, involving coded sequences are all polynomial-time computable.

MORE ON φ^n , Φ^n , AND THE COST OF GAMES. By standard results [Jon97, RC94], for each $m, n \geq 1$, there is a polynomial-time computable $s_{m,n}: \mathbb{N}^{m+1} \rightarrow \mathbb{N}$ and an $(m+1)$ -variable polynomial $p_{m,n}$ such that, for all $i \in \mathbb{N}$, $\vec{x} \in \mathbb{N}^m$, and $\vec{y} \in \mathbb{N}^n$:

$$\varphi_{s_{m,n}(i, \vec{x})}^n(\vec{y}) = \varphi_i^{m+n}(\vec{x}, \vec{y}). \quad (5)$$

$$\Phi_{s_{m,n}(i, \vec{x})}^n(\vec{y}) \leq p_{m,n}(|x_0|, \dots, |x_{m-1}|, \Phi_i^{m+n}(\vec{x}, \vec{y})). \quad (6)$$

It is also standard that, for each n , there are $R^n, S^n, T^n: \mathbb{N}^{n+1} \times \omega \rightarrow \mathbb{N}$ such that, for each $i, t \in \mathbb{N}$ and $\vec{x} \in \mathbb{N}^n$,

$$R^n(i, \vec{x}, \mathbf{0}^t) = \begin{cases} \varphi_i^n(\vec{x}), & \text{if } \Phi_i^n(\vec{x}) \leq t; \\ 0, & \text{otherwise;} \end{cases} \quad (7)$$

$$S^n(i, \vec{x}, \mathbf{0}^t) = \begin{cases} \Phi_i^n(\vec{x}), & \text{if } \Phi_i^n(\vec{x}) \leq t; \\ 0, & \text{otherwise;} \end{cases} \quad (8)$$

$$T^n(i, \vec{x}, \mathbf{0}^t) = \begin{cases} 1, & \text{if } \Phi_i^n(\vec{x}) \leq t; \\ 0, & \text{otherwise;} \end{cases} \quad (9)$$

and that these functions are computable in time polynomial in $|i| + |x_0| + \dots + |x_{n-1}| + |\mathbf{0}^t|$, where $|\mathbf{0}^t| = t$.³

For the cost of plays there is a $T': \mathbb{N} \times (\mathbb{N} \rightarrow \mathbb{N}) \times \omega \rightarrow \{0, 1\}$ that is roughly analogous to the T^n 's above. This T' is given by:

$$T'(i, g, \mathbf{0}^t) = \begin{cases} 1, & \text{if (i) } C(i, g) \leq t; \\ 0, & \text{if (ii) } C(i, g) > t \text{ and } g \text{ was defined} \\ & \text{on all the questions asked it in} \\ & \text{the play of “}\varphi_i|g\text{” up to the point} \\ & \text{where the accumulated cost ex-} \\ & \text{ceeded } t; \\ \text{undefined,} & \text{(iii) otherwise.} \end{cases} \quad (10)$$

The key complexity property of T' used in the proof of Theorem 14 is:

LEMMA 16. *Suppose that $f: \mathbb{N}^{m+1} \rightarrow \mathbb{N}$ is polynomial-time computable. Then the function $T'_f = \lambda i, x_0, \dots, x_{m-1}, \mathbf{0}^t. T'(i, \lambda y. f(x_0, \dots, x_{m-1}, y), \mathbf{0}^t)$ is total and, moreover, polynomial-time computable.*

PROOF SKETCH. Since f is total, so is $\lambda y. f(\vec{x}, y)$ for any choice of \vec{x} . So by Definition 13, for any i and \vec{x} , $C(i, \lambda y. f(\vec{x}, y))$ cannot be undefined—although it might be ∞ . It thus follows that T'_f is total.

One way to show the moreover clause is through the theory of the basic feasible functionals (BFFs) [CU93, IKR01]. First, it is straightforward to show that T' a type-2 BFF when the type-1 argument is restricted to be total. Then, since the 1-section of BFF is exactly the class of (type-1) polynomial-time computable functions, the moreover clause follows immediately.

Here is a sketch an alternative argument for the moreover clause that avoids a digression into the theory of the BFFs. We shall establish a polynomial upper bound on the cost of running the play of “ $\varphi_i|\lambda y. f(\vec{x}, y)$ ” up to the point where either the play finishes or else the accumulated cost exceeds t . This possibly truncated play we shall call a *play* by abuse of terminology. One important fact to keep in mind in the following is that, under the uniform cost model for RAMs, reading or writing down a string of length k has cost k .

For the moment fix i, x_0, \dots, x_{n-1} , and t .

We first consider the cost of running the $\lambda y. f(\vec{x}, y)$ -side of the play. Suppose $p_f: \mathbb{N}^{m+1} \rightarrow \mathbb{N}$ is a polynomial (or more properly, the function named by

³It is easily shown that that the time required for testing $\Phi_i^n(x_0, \dots, x_{n-1}) \leq t$ has a lower bound that is polynomial in $|x_0|, \dots, |x_{n-1}|$ and *exponential* in $|t|$. Hence, representing the “time parameter” t as a tally string is essential in having R^n , S^n , and T^n computable within time polynomial in the lengths of their inputs.

a polynomial) that bounds the run time of some program for f . So by our choice of p_f and Definition 13 it follows that in the course of the play: (i) $\lambda y.f(\vec{x}, y)$ cannot be asked more than t many queries, (ii) the length of any one of these queries must be no greater than t , and (iii) the cost of computing the reply to one of these queries is no greater than $p_f(|x_0|, \dots, |x_{n-1}|, t)$. Thus, the cost of computing the replies to all of these queries is no more than $t \cdot p_f(|x_0|, \dots, |x_{n-1}|, t)$. Moreover, if α is the final sequence of these replies in the play, then it follows by the definition of the coding of sequences that $|\langle \alpha \rangle| \leq t \cdot (2 \cdot p_f(|x_0|, \dots, |x_{n-1}|, t) + 3)$.

We now consider the cost of running the φ_i -side of the play. Let $p_1: \mathbb{N}^3 \rightarrow \mathbb{N}$ be a polynomial that bounds the run times of particular programs for R^1 , S^1 , and T^1 (from (7), (8), and (9)). It follows from the definitions of play, C , and T' that there is a polynomial $p: \mathbb{N}^2 \rightarrow \mathbb{N}$ (independent of i , x_0, \dots, x_{n-1} , and t) such that if the play goes on for m rounds and the length of the coded sequence of replies is never greater than n , then the cost of running the φ_i -side of the play is no more than $p(m, p_1(|i|, n, t))$. It follows from the bounds of the previous paragraph that $m \leq t$ and $n \leq t \cdot (2 \cdot p_f(|x_0|, \dots, |x_{n-1}|, t) + 3)$. Thus, we have established our polynomial bound. \square

VERTEX COVERS. The *vertex cover* problem is:

GIVEN an undirected graph $G = (V, E)$ and a $k \leq \mathbf{Card}(V)$,
 DECIDE whether there is a $V' \subseteq V$ with $\mathbf{Card}(V') \leq k$ such that,
 for each $\{u, v\} \in E$, at least one of u and v belongs to V' .

Karp [Kar72, GJ79] showed that the vertex cover problem is NP-complete.

PROOF OF THEOREM 14. We prove the contrapositive. Suppose that $inst_h$ is a polynomial overhead instantiation of h , and p_h is the associated polynomial. We show how to use $inst_h$ to build a deterministic, polynomial-time vertex-cover tester.

Our first order of business is to establish a convenient way of encoding instances of the vertex-cover problem into strings. We define a *coded instance* of vertex cover to be a triple $(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle)$, where $n \geq k$ and β is of the form $[\langle y_0, z_0 \rangle, \dots, \langle y_m, z_m \rangle]$ where for each i , $y_i < z_i < n$ and where no $\langle y_i, z_i \rangle$ is repeated in the list. Thus, the instance coded is (G, k) where $G = (\{0, \dots, n-1\}, E)$ and E consists of the pairs in β .⁴ A *solved coded instance* consists of

⁴A standard measure of the size of a graph $G = (V, E)$ is $|G| = \mathbf{Card}(V) + \mathbf{Card}(E)$. Suppose $(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle)$ is a coded instance (G, k) where $G = (\{0, \dots, n-1\}, E)$. Then some calculation shows that $|G| + k \leq |\mathbf{0}^n| + |\mathbf{0}^k| + |\langle \beta \rangle| \leq n + k + \mathbf{Card}(E) \cdot (8 \log_2 n + 13) \leq 8 \cdot (|G| + 2)^2 + k$. Hence, our representation of the problem and the standard representation are polynomially close in size. This would not have been the case if we had chosen to represent n in dyadic (or binary) notation instead of unary.

a 4-tuple $(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle, \langle \gamma \rangle)$, where $(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle)$ is a coded instance, γ is of the form $[?x_0, \dots, ?x_{n-1}, !0]$, and each coded pair in β has at least one member in $\{x_0, \dots, x_{k-1}\}$.⁵ Let \mathcal{S} be the collection of all solved coded instances. It is straightforward that membership in \mathcal{S} is polynomial-time decidable.

Our vertex-cover tester will be based on r_n of Example 10. For each j and n , let G^n , g_π^n , and ζ_j be as in Example 10. Before we consider our modification of r_n , let us discuss some key properties of r_n itself. From our definition of r_n it is clear that there is an $i_n \in \mathbb{N}$ and a polynomial p_n such that $\varphi_{i_n} = r_n$ and, for all α with $r_n \langle \alpha \rangle \downarrow$, we have $\Phi_{i_n}(\langle \alpha \rangle) \leq p_n(|\langle \alpha \rangle|)$. It is straightforward to show that for all the α 's for which $r_n \langle \alpha \rangle \downarrow$, we have $|\langle \alpha \rangle| \leq 2(n+2)^2$. Hence, a little calculation shows that for each g_π^n we have that $C(i_n, g_\pi^n) \leq (n+1) \cdot p_n(2(n+2)^2)$, and thus, by our assumptions on $inst_h$ and p_h , for each π we have

$$C(inst_h(i_n), g_\pi^n) \leq p_h((n+1) \cdot p_n(2(n+2)^2)).$$

Now suppose that i'_n is a possible variant of i_n such that on certain of the g_π^n 's we may (or may not) have $(\varphi_{i'_n} | g_\pi^n) \uparrow$ and for all the other g_π^n 's, the computations of i'_n in the play of " $\varphi_{i'_n} | g_\pi^n$ " are identical to those of i_n in the play of " $\varphi_{i_n} | g_\pi^n$." So, if $(\varphi_{i'_n} | g_\pi^n) \downarrow$ for *all* the g_π^n , then for all π ,

$$C(inst_h(i'_n), g_\pi^n) \leq p_h((n+1) \cdot p_n(2(n+1))). \quad (11)$$

On the other hand, if $(\varphi_{i'_n} | g_\pi^n) \uparrow$ for at least one of the g_π^n , then for *all* π ,

$$C(inst_h(i'_n), g_\pi^n) = \infty.$$

Hence, we can test if $(\varphi_{i'_n} | g_\pi^n) \uparrow$ on any of the π 's by simply picking an arbitrary π and checking if (11) fails for this particular π . **N.B.** This is the key idea for performing the search in our vertex-cover tester. To apply this idea, we have to integrate it with an appropriate parameterization of coded instances and check that all the details work out.

Here then is our modification of r_n . Let $f: \omega^2 \times \mathbb{N}^2 \rightarrow \mathbb{N}$ be such that:

$$f(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle, \langle \alpha \rangle) = \begin{cases} ?\langle \zeta_j \rangle, & \text{if } \alpha = [?x_0, \dots, ?x_m] \text{ where } m < n \text{ and} \\ & x_0, \dots, x_m \text{ are } < n \text{ and pairwise distinct;} \\ !0, & \text{if } \alpha = [?x_0, \dots, ?x_{n-1}, !0] \text{ where } x_0, \dots, \\ & x_{n-1} \text{ are } < n \text{ and pairwise distinct, and} \\ & (\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle, \langle \alpha \rangle) \notin \mathcal{S}; \\ \text{undefined,} & \text{otherwise.} \end{cases} \quad (12)$$

⁵Coding solutions this way is completely *ad hoc*, but quite convenient as we shall see.

Note: If $(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle, \langle \gamma \rangle) \in \mathcal{S}$, then $f(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle, \langle \gamma \rangle) \uparrow$.

We need to analyze the complexity of this f . Since our coding of sequences is polynomial time, it follows that there is an i_f such that $f = \varphi_{i_f}^4$ and also there is a polynomial p_f such that, for all n, k, β , and α such that $f(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle, \langle \alpha \rangle) \downarrow$, we have

$$\Phi_{i_f}^4(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle, \langle \alpha \rangle) \leq p_f(n, k, |\langle \beta \rangle|, |\langle \alpha \rangle|).$$

It is straightforward to show that for each α such that $f(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle, \langle \alpha \rangle) \downarrow$, we have that $|\langle \alpha \rangle| \leq 2(n+2)^2$. Hence, we can replace the above inequality with:

$$\Phi_{i_f}^4(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle, \langle \alpha \rangle) \leq p_f(n, k, |\langle \beta \rangle|, 2(n+2)^2). \quad (13)$$

To make use of this f with $inst_h$, we need a ‘‘partially evaluated’’ version of i_f . So suppose $s: \omega^2 \times \mathbb{N} \rightarrow \mathbb{N}$ is given by: $s(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle) = s_{3,1}(i_f, \mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle)$. Then s is polynomial-time computable, and for all n, k, β , and α ,

$$\varphi_{s(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle)} \langle \alpha \rangle = f(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle, \langle \alpha \rangle). \quad (14)$$

Moreover, if $f(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle, \langle \alpha \rangle) \downarrow$, then by (6) and (13) we have

$$\Phi_{s(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle)} \langle \alpha \rangle \leq p_s(n, k, |\langle \beta \rangle|), \quad (15)$$

where p_s is the polynomial such that

$$p_s(n, k, m) = p_{3,1} \left(n, k, m, p_f(n, k, m, 2(n+2)^2) \right).$$

For this paragraph fix $(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle)$, a coded instance of vertex cover. Let (G, k) be the instance of vertex cover so coded, let π_{id} be the identity permutation on $\{0, \dots, n-1\}$, let π_0 be an arbitrary permutation on $\{0, \dots, n-1\}$, and let $\gamma = [?\pi_0(0), \dots, ?\pi_0(n-1), !0]$. Thus,

$$\begin{aligned} & (\varphi_{s(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle)} \mid g_{\pi_0}^n) \\ &= ((\lambda y. f(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle, y)) \mid g_{\pi_0}^n) && \text{(by (14))} \\ &= f(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle, \langle \gamma \rangle) && \text{(by (12), (3), and} \\ & && \text{the definition of play).} \end{aligned}$$

By our definition of f , we thus have:

- If $(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle, \langle \gamma \rangle) \in \mathcal{S}$, then $(\varphi_{s(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle)} \mid g_{\pi_0}^n) \uparrow$.
- If $(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle, \langle \gamma \rangle) \notin \mathcal{S}$, then $(\varphi_{s(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle)} \mid g_{\pi_0}^n) \downarrow = 0$.

Therefore, we have the following two cases.

CASE 1: G has a vertex cover of size $\leq k$. Then, for at least one π , we have $(\varphi_{s(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle)} \mid g_{\pi}^n) \uparrow$. Hence by h 's construction, $(\varphi_{inst_h(s(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle))} \mid g_{\pi_{id}}^n) \uparrow$ and

$$C(inst_h(s(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle)), g_{\pi_{id}}^n) = \infty.$$

CASE 2: G fails to have a vertex cover of size $\leq k$. Then, for *all* π , we have $(\varphi_{s(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle)} \mid g_{\pi}^n) \downarrow = 0$. Hence by h 's construction, $(\varphi_{inst_h(s(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle))} \mid g_{\pi_{id}}^n) \downarrow = 0$. In fact, by (15) and our hypotheses on $inst_h$ and p_h , we have

$$C(inst_h(s(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle)), g_{\pi_{id}}^n) \leq p_h((n+1) \cdot p_s(n, k, |\langle \beta \rangle|)). \quad (16)$$

Finally, let $VC: \omega^2 \times \mathbb{N} \rightarrow \{\text{true}, \text{false}\}$ be given by:

$$VC(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle) \equiv [(16) \text{ fails to hold}].$$

We claim that this is our polynomial-time, vertex-cover tester. First let us check correctness. Suppose $(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle)$ is a coded instance of the vertex-cover problem and (G, k) is the instance so coded. We know from the case analysis of the previous paragraph that G has a vertex cover of size k or less if and only if (16) is false. Hence, correctness follows. Now let us check that VC is polynomial-time computable. First note that there is a polynomial-time computable $g: \omega \times \mathbb{N} \rightarrow \mathbb{N}$ such that

$$g(\mathbf{0}^n, x) = \begin{cases} g_{\pi_{id}}^n(x), & \text{if } g_{\pi_{id}}^n(x) \downarrow; \\ 0, & \text{otherwise.} \end{cases}$$

Also note that in the play of “ $(\lambda y. f(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle, y)) \mid g_{\pi_0}^n$,” the function $g_{\pi_0}^n$ is never queried on any value on which it is undefined. So (16) is equivalent to:

$$C(inst_h(s(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle)), \lambda x. g(\mathbf{0}^n, x)) \leq p_h((n+1) \cdot p_s(n, k, |\langle \beta \rangle|)).$$

This, in turn is equivalent to

$$T'(inst_h(s(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle)), \lambda x. g(\mathbf{0}^n, x), \mathbf{0}^m) = 1, \quad (17)$$

where $m = p_h((n+1) \cdot p_s(n, k, |\langle \beta \rangle|))$ and T' is as in (10). It is straightforward that from $(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle)$ one can compute $\mathbf{0}^m$ in polynomial time. Also, since $inst_h$, s , and g are polynomial-time functions, it follows from Lemma 16 that within polynomial time we can test (17), and hence, also (16). Therefore, VC is polynomial-time computable. \square

DEFINITION 17. Suppose $r, g \in \mathcal{PR}$. We say that r queries g in tree order if and only if, in the play of “ $r | g$ ”, r 's first question is $?\langle \rangle$ (i.e., r asks g what its first question to its oracle is) and every subsequent question r asks is of the form $?\langle \alpha; x \rangle$ where $?\langle \alpha \rangle$ is a question r asked previously in this game and $g\langle \alpha \rangle$ is not an answer. \diamond

This is a rather technical definition. The idea is that if r queries g in tree order, then r must systematically explore down branches of g 's decision tree. For example, r cannot ask $?\langle 6, 7, 98 \rangle$ without having previously asked $?\langle \rangle$, $?\langle 6 \rangle$, and $?\langle 6, 7 \rangle$ —in that order. Also note that querying in tree order allows for games in which, at a given point in the play, multiple branches of a decision tree that are only partially explored.

LEMMA 18. Suppose t realizes a retraction and r queries g in tree order where r is semi-irredundant and g is irredundant.

(a) The play of $(t \bullet r)$ against g is identical to the play of r against g . Thus in particular, if $(r | g)\uparrow$ then $((t \bullet r) | g)\uparrow$.

(b) Suppose that \hat{g} is irredundant, $\hat{g} \sim_2 g$, and $(r | g)\uparrow$. Then $((t \bullet r) | \hat{g})\uparrow$.

PROOF SKETCH. *Part (a).* Suppose by way of contradiction that the plays of “ $(t \bullet r) | g$ ” and “ $r | g$ ” fail to match. We consider the case in which the first place that the two plays do not match $(t \bullet r)$ asks a question, say $?z$. (The other cases are similar.) Let ν and ξ be the finite parts of r and g , respectively, used in the play of the two games up to this point. Then it is straightforward to extend ν and ξ to r_0 and g_0 , respectively, so that r_0 is \sim_2 -extensional, $(r_0 | g_0)\downarrow$, and $g_0(z)\uparrow$. So, since in the play of “ $(t \bullet r_0) | g_0$ ”, $(t \bullet r_0)$ will ask the question $?z$, we have that $(t \bullet r_0) | g_0\uparrow$. Hence, r_0 is \sim_2 -extensional and $r_0 | g_0 \neq (t \bullet r_0) | g_0$. Thus t fails to realize a retraction, a contradiction.

Part (b). Since $t \bullet r$ must be \sim_2 -extensional, this part follows from part (a). \square

PROOF SKETCH OF THEOREM 15. Note that in the proof of Theorem 14, for each n, k, β and π , $\lambda y.f(\mathbf{0}^n, \mathbf{0}^k, \langle \beta \rangle, y)$ queries g_π^n in tree order. Hence it follows from Lemma 18 that the argument for h works just as well for t . \square

§6. Conclusions and Problems

We have confirmed Longley's suspicions [Lon98b] as to the infeasibility of H . Our results are thus not too surprising. What may be a bit surprising is that these results were obtained by very simple (if ham-fisted) means. We would like to think that this indicates that simple standard tools and techniques

suffice to start exploring some of the complexity theoretic issues around the recent work on dialog games and realizability.

SR is a very natural and strong class of sequentially computable functionals. A key question is whether one can construct a practical functional programming language based on SR. The results of this paper show that one set of approaches to constructing such a language is infeasible because of complexity concerns. We suspect any *functional* language based on SR will run into similar problems. Formalizing and establishing (or refuting) this suspicion are open problems. (The situation for non-functional languages based on SR [Lon00] looks more hopeful.) Moreover, the proofs of Theorems 14 and 15 suggest that there are serious complexity theoretic difficulties that result from mixing strong forms of sequentiality and extensionality. Exploring the extent of these difficulties could be quite interesting.

ACKNOWLEDGMENTS. The author wishes to thank John Longley, Sue Older, and the anonymous referees for numerous comments that helped to greatly improve the paper. Thanks, as always, to Elaine Weinman for her careful comments on the text.

References

- [CU93] S. Cook and A. Urquhart, *Functional interpretations of feasibly constructive arithmetic*, Annals of Pure and Applied Logic **63** (1993), 103–200.
- [GJ79] M. Garey and D. Johnson, *Computers and intractability*, Freeman, 1979.
- [IKR01] R. Irwin, B. Kapron, and J. Royer, *On characterizations of the basic feasible functional, Part I*, Journal of Functional Programming **11** (2001), 117–153.
- [Jon97] N. Jones, *Computability and complexity from a programming perspective*, MIT Press, 1997.
- [Kar72] R. Karp, *Reducibility among combinatorial problems*, Complexity of Computer Computations (R.E. Miller and J.W. Thatcher, eds.), Plenum Press, 1972, pp. 85–103.
- [Lon98a] J. Longley, *Realizability models for sequential computation*, draft of a paper based on a talk given at the APPSEM '98 Workshop in Pisa, available as <http://www.dcs.ed.ac.uk/home/jrl/pisa.ps.gz>, 1998.

-
- [Lon98b] J. Longley, *The sequentially realizable functionals*, Tech. Report ECS-LFCS-98-402, Dept. of Computer Science, University of Edinburgh, 1998, to appear in *Annals of Pure and Applied Logic*.
- [Lon99] J. Longley, *When is a functional program not a functional program?*, Proceedings of Fourth ACM SIGPLAN International Conference on Functional Programming, 1999.
- [Lon00] John Longley, *Matching typed and untyped realizability*, Electronic Notes in Theoretical Computer Science (Lars Birkedal, Jaap van Oosten, Giuseppe Rosolini, and Dana S. Scott, eds.), vol. 23, Elsevier Science Publishers, 2000.
- [LV97] M. Li and P. Vitányi, *An introduction to Kolmogorov complexity and its applications, second edition*, Springer-Verlag, 1997.
- [Odi81] P. Odifreddi, *Strong reducibilities*, Bulletin of the American Mathematical Society **4** (1981), 37–86.
- [Pap94] C. Papadimitriou, *Computational complexity*, Addison-Wesley, 1994.
- [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.
- [vO99] J. van Oosten, *A combinatory algebra for sequential functionals of finite type*, Models and Computability (Leeds, 1997) (S. Cooper and J. Truss, eds.), Cambridge University Press, 1999, pp. 389–405.