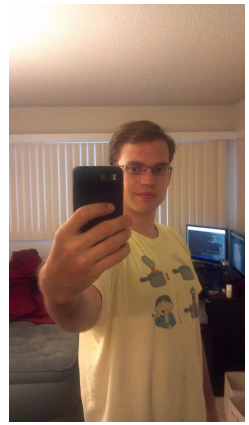
Non-Obfuscated Yet Unprovable Programs

John Case Michael Ralston



Computer and Information Sciences Department
University of Delaware
Newark, DE 19716
USA
Email: {case, ralston}@udel.edu

Asian Logic Conference 2011
Wellington, NZ

For Your Speed Reading Pleasure & Quick Impression (☺)

Background

- The International Obfuscated C Code Contest was a programming contest for the most creatively obfuscated C code, held annually between 1984 and 1996, and thereafter in 1998, 2000, 2001, 2004, and 2006.
- In many cases, the winning programmer did something simple in such an obscure but succinct way that it was hard for other (human) programmers to see how his/her code actually worked.
- By contrast, the interest herein is in programs which are, in a sense, easily seen to be correct, but which cannot be proved correct in pre-assigned, computably axiomatized, powerful, true theories **T**.
- Our programs will be linearly relatively succinct and fast — relative to what, we'll see. We'll employ a few terms for the needed complexity-theoretic computability theory.

Background

- The International Obfuscated C Code Contest was a programming contest for the most creatively obfuscated C code, held annually between 1984 and 1996, and thereafter in 1998, 2000, 2001, 2004, and 2006.
- In many cases, the winning programmer did something simple in such an obscure but succinct way that it was hard for other (human) programmers to see how his/her code actually worked.
- By contrast, the interest herein is in programs which are, in a sense, easily seen to be correct, but which cannot be proved correct in pre-assigned, computably axiomatized, powerful, true theories **T**.
- Our programs will be linearly relatively succinct and fast — relative to what, we'll see. We'll employ a few frames for the needed complexity-theoretic computability theory.

Background

- The International Obfuscated C Code Contest was a programming contest for the most creatively obfuscated C code, held annually between 1984 and 1996, and thereafter in 1998, 2000, 2001, 2004, and 2006.
- In many cases, the winning programmer did something simple in such an obscure but succinct way that it was hard for other (human) programmers to see how his/her code actually worked.
- By contrast, the interest herein is in programs which are, in a sense, easily seen to be correct, but which cannot be proved correct in pre-assigned, computably axiomatized, powerful, true theories **T**.
- Our programs will be linearly relatively succinct and fast — relative to what, we'll see. We'll employ a few frames for the needed complexity-theoretic computability theory.

Background

- The International Obfuscated C Code Contest was a programming contest for the most creatively obfuscated C code, held annually between 1984 and 1996, and thereafter in 1998, 2000, 2001, 2004, and 2006.
- In many cases, the winning programmer did something simple in such an obscure but succinct way that it was hard for other (human) programmers to see how his/her code actually worked.
- By contrast, the interest herein is in programs which are, in a sense, easily seen to be correct, but which cannot be proved correct in pre-assigned, computably axiomatized, powerful, true theories **T**.
- Our programs will be linearly relatively succinct and fast — relative to what, we'll see. We'll employ a few frames for the needed complexity-theoretic computability theory.

Background

- The International Obfuscated C Code Contest was a programming contest for the most creatively obfuscated C code, held annually between 1984 and 1996, and thereafter in 1998, 2000, 2001, 2004, and 2006.
- In many cases, the winning programmer did something simple in such an obscure but succinct way that it was hard for other (human) programmers to see how his/her code actually worked.
- By contrast, the interest herein is in programs which are, in a sense, easily seen to be correct, but which cannot be proved correct in pre-assigned, computably axiomatized, powerful, true theories **T**.
- Our programs will be linearly relatively succinct and fast — relative to what, we'll see. We'll employ a few frames for the needed complexity-theoretic computability theory.

Mathematical Preliminaries I

- Let φ^{TM} be the efficiently laid out and Gödel-numbered acceptable programming system (numbering) from [RC94] and which is based on deterministic multi-tape Turing Machines (with base 2 I/O). Its programs are named by numbers in $\mathbb{N} \stackrel{\mathrm{def}}{=} \{0, 1, 2, \ldots\}$.
- Let Φ^{TM} be the corresponding step-counting Blum Complexity Measure.
- For p, a numerically named program in φ^{TM}, let $|p|$ = the length of p written in binary, which $= (\lceil \log_2(p+1) \rceil)_+$. $(\,)_+$ turns 0 into 1, else, leaves unchanged.
- The numerical naming mentioned above does not feature prime powers and factorization, but, instead, is a linear-time computable and invertible coding.

Mathematical Preliminaries I

- Let φ^{TM} be the efficiently laid out and Gödel-numbered acceptable programming system (numbering) from [RC94] and which is based on deterministic multi-tape Turing Machines (with base 2 I/O). Its programs are named by numbers in $\mathbb{N} \stackrel{\mathrm{def}}{=} \{0, 1, 2, \ldots\}$.
- Let Φ^{TM} be the corresponding step-counting Blum Complexity Measure.
- For p, a numerically named program in φ^{TM}, let $|p|$ = the length of p written in binary, which $= (\lceil \log_2(p+1) \rceil)_+$. $(\cdot)_+$ turns 0 into 1; else, leaves unchanged.
- The numerical naming mentioned above does not feature prime powers and factorization, but, instead, is a linear-time computable and invertible coding.

Mathematical Preliminaries I

- Let φ^{TM} be the efficiently laid out and Gödel-numbered acceptable programming system (numbering) from [RC94] and which is based on deterministic multi-tape Turing Machines (with base 2 I/O). Its programs are named by numbers in $\mathbb{N} \stackrel{\mathrm{def}}{=} \{0, 1, 2, \ldots\}$.
- Let Φ^{TM} be the corresponding step-counting Blum Complexity Measure.
- For p, a numerically named program in φ^{TM}, let $|p|$ = the length of p written in binary, which $= (\lceil \log_2(p+1) \rceil)_+$. $(\cdot)_+$ turns 0 into 1; else, leaves unchanged.
- The numerical naming mentioned above does not feature prime powers and factorization, but, instead, is a linear-time computable and invertible coding.

Mathematical Preliminaries I

- Let φ^{TM} be the efficiently laid out and Gödel-numbered acceptable programming system (numbering) from [RC94] and which is based on deterministic multi-tape Turing Machines (with base 2 I/O). Its programs are named by numbers in $\mathbb{N} \stackrel{\mathrm{def}}{=} \{0, 1, 2, \ldots\}$.
- Let Φ^{TM} be the corresponding step-counting Blum Complexity Measure.
- For p, a numerically named program in φ^{TM}, let $|p|$ = the length of p written in binary, which $= (\lceil \log_2(p+1) \rceil)_+$. $(\cdot)_+$ turns 0 into 1; else, leaves unchanged.
- The numerical naming mentioned above does not feature prime powers and factorization, but, instead, is a linear-time computable and invertible coding.

Mathematical Preliminaries I

- Let φ^{TM} be the efficiently laid out and Gödel-numbered acceptable programming system (numbering) from [RC94] and which is based on deterministic multi-tape Turing Machines (with base 2 I/O). Its programs are named by numbers in $\mathbb{N} \stackrel{\mathrm{def}}{=} \{0, 1, 2, \ldots\}$.
- Let Φ^{TM} be the corresponding step-counting Blum Complexity Measure.
- For p, a numerically named program in φ^{TM}, let $|p|$ = the length of p written in binary, which $= (\lceil \log_2(p+1) \rceil)_+$. $(\cdot)_+$ turns 0 into 1; else, leaves unchanged.
- The numerical naming mentioned above does not feature prime powers and factorization, but, instead, is a linear-time computable and invertible coding.

Mathematical Preliminaries II

From [RC94, Lemma 3.14]:
There are small positive $a \in \mathbb{N}$ and function if-then-else $\in$ **LinearTime** st, for all $p_0, p_1, p_2, x \in \mathbb{N}$:

$$\varphi^{\mathrm{TM}}_{\text{if-then-else}(p_0,p_1,p_2)}(x) = \begin{cases} \varphi^{\mathrm{TM}}_{p_1}(x), & \text{if } \varphi^{\mathrm{TM}}_{p_0}(x)\downarrow \neq 0; \\ \varphi^{\mathrm{TM}}_{p_2}(x), & \text{if } \varphi^{\mathrm{TM}}_{p_0}(x)\downarrow = 0; \\ \uparrow, & \text{otherwise;} \end{cases}$$

$$\Phi^{\mathrm{TM}}_{\text{if-then-else}(p_0,p_1,p_2)}(x) \leq$$

$$\begin{cases} a \cdot (\Phi^{\mathrm{TM}}_{p_0}(x) + \Phi^{\mathrm{TM}}_{p_1}(x))_+, & \text{if } \varphi^{\mathrm{TM}}_{p_0}(x)\downarrow \neq 0; \\ a \cdot (\Phi^{\mathrm{TM}}_{p_0}(x) + \Phi^{\mathrm{TM}}_{p_2}(x))_+, & \textit{if } \varphi^{\mathrm{TM}}_{p_0}(x)\downarrow = 0; \\ \uparrow, & \text{otherwise.} \end{cases}$$

Mathematical Preliminaries II

From [RC94, Lemma 3.14]:
There are small positive $a \in \mathbb{N}$ and function $\text{if-then-else} \in \textbf{LinearTime}$ st, for all $p_0, p_1, p_2, x \in \mathbb{N}$:

$$\varphi^{\mathrm{TM}}_{\text{if-then-else}(p_0,p_1,p_2)}(x) = \begin{cases} \varphi^{\mathrm{TM}}_{p_1}(x), & \text{if } \varphi^{\mathrm{TM}}_{p_0}(x)\downarrow \neq 0; \\ \varphi^{\mathrm{TM}}_{p_2}(x), & \text{if } \varphi^{\mathrm{TM}}_{p_0}(x)\downarrow = 0; \\ \uparrow, & \text{otherwise;} \end{cases}$$

$$\Phi^{\mathrm{TM}}_{\text{if-then-else}(p_0,p_1,p_2)}(x) \leq$$

$$\begin{cases} a \cdot (\Phi^{\mathrm{TM}}_{p_0}(x) + \Phi^{\mathrm{TM}}_{p_1}(x))_+, & \text{if } \varphi^{\mathrm{TM}}_{p_0}(x)\downarrow \neq 0; \\ a \cdot (\Phi^{\mathrm{TM}}_{p_0}(x) + \Phi^{\mathrm{TM}}_{p_2}(x))_+, & \textit{if } \varphi^{\mathrm{TM}}_{p_0}(x)\downarrow = 0; \\ \uparrow, & \text{otherwise.} \end{cases}$$

Mathematical Preliminaries II

From [RC94, Lemma 3.14]:
There are small positive $a \in \mathbb{N}$ and function $\text{if-then-else} \in \textbf{LinearTime}$ st, for all $p_0, p_1, p_2, x \in \mathbb{N}$:

$$\varphi^{\mathrm{TM}}_{\text{if-then-else}(p_0,p_1,p_2)}(x) = \begin{cases} \varphi^{\mathrm{TM}}_{p_1}(x), & \text{if } \varphi^{\mathrm{TM}}_{p_0}(x)\downarrow \neq 0; \\ \varphi^{\mathrm{TM}}_{p_2}(x), & \text{if } \varphi^{\mathrm{TM}}_{p_0}(x)\downarrow = 0; \\ \uparrow, & \text{otherwise;} \end{cases}$$

$$\Phi^{\mathrm{TM}}_{\text{if-then-else}(p_0,p_1,p_2)}(x) \leq$$

$$\begin{cases} a \cdot (\Phi^{\mathrm{TM}}_{p_0}(x) + \Phi^{\mathrm{TM}}_{p_1}(x))_+, & \text{if } \varphi^{\mathrm{TM}}_{p_0}(x)\downarrow \neq 0; \\ a \cdot (\Phi^{\mathrm{TM}}_{p_0}(x) + \Phi^{\mathrm{TM}}_{p_2}(x))_+, & \textit{if } \varphi^{\mathrm{TM}}_{p_0}(x)\downarrow = 0; \\ \uparrow, & \text{otherwise.} \end{cases}$$

Mathematical Preliminaries III

Essentially from [RC94, Theorem 4.8] we have the following constructive, efficient, and parametrized version of Kleene's 2nd (not Rogers') Recursion Theorem [Rog67, Page 214].
There are small positive $b \in \mathbb{N}$ and function $\mathrm{krt} \in$ **LinearTime** st, for all parameter values p, tasks r, inputs $x \in \mathbb{N}$:

$$\varphi^{\mathrm{TM}}_{\mathrm{krt}(p,r)}(x) = \varphi^{\mathrm{TM}}_{r}(\langle \mathrm{krt}(p,r), p, x\rangle).$$

Above, $\mathrm{krt}(p,r)$ has p, r stored inside, and, on x, makes a self-copy, forms $y = \langle \text{self-copy}, p, x\rangle$, and runs task r on this y.

$$\Phi^{\mathrm{TM}}_{\mathrm{krt}(p,r)}(x) \leq b \cdot (|p| + |r| + |x| + \Phi^{\mathrm{TM}}_{r}(\langle \mathrm{krt}(p,r), p, x\rangle)).$$

Mathematical Preliminaries III

Essentially from [RC94, Theorem 4.8] we have the following constructive, efficient, and parametrized version of Kleene's 2nd (not Rogers') Recursion Theorem [Rog67, Page 214].
There are small positive $b \in \mathbb{N}$ and function $\mathrm{krt} \in$ **LinearTime** st, for all parameter values p, tasks r, inputs $x \in \mathbb{N}$:

$$\varphi^{\mathrm{TM}}_{\mathrm{krt}(p,r)}(x) = \varphi^{\mathrm{TM}}_{r}(\langle \mathrm{krt}(p,r), p, x\rangle).$$

Above, $\mathrm{krt}(p,r)$ has p, r stored inside, and, on x, makes a self-copy, forms $y = \langle \text{self-copy}, p, x\rangle$, and runs task r on this y.

$$\Phi^{\mathrm{TM}}_{\mathrm{krt}(p,r)}(x) \leq b \cdot (|p| + |r| + |x| + \Phi^{\mathrm{TM}}_{r}(\langle \mathrm{krt}(p,r), p, x\rangle)).$$

Mathematical Preliminaries III

Essentially from [RC94, Theorem 4.8] we have the following constructive, efficient, and parametrized version of Kleene's 2nd (not Rogers') Recursion Theorem [Rog67, Page 214].
There are small positive $b \in \mathbb{N}$ and function $\mathrm{krt} \in$ **LinearTime** st, for all parameter values p, tasks r, inputs $x \in \mathbb{N}$:

$$\varphi^{\mathrm{TM}}_{\mathrm{krt}(p,r)}(x) = \varphi^{\mathrm{TM}}_{r}(\langle \mathrm{krt}(p,r), p, x\rangle).$$

Above, $\mathrm{krt}(p,r)$ has p, r stored inside, and, on x, makes a self-copy, forms $y = \langle \text{self-copy}, p, x\rangle$, and runs task r on this y.

$$\Phi^{\mathrm{TM}}_{\mathrm{krt}(p,r)}(x) \leq b \cdot (|p| + |r| + |x| + \Phi^{\mathrm{TM}}_{r}(\langle \mathrm{krt}(p,r), p, x\rangle)).$$

Mathematical Preliminaries III

Essentially from [RC94, Theorem 4.8] we have the following constructive, efficient, and parametrized version of Kleene's 2nd (not Rogers') Recursion Theorem [Rog67, Page 214].
There are small positive $b \in \mathbb{N}$ and function $\mathrm{krt} \in$ **LinearTime** st, for all parameter values p, tasks r, inputs $x \in \mathbb{N}$:

$$\varphi^{\mathrm{TM}}_{\mathrm{krt}(p,r)}(x) = \varphi^{\mathrm{TM}}_{r}(\langle \mathrm{krt}(p,r), p, x\rangle).$$

Above, $\mathrm{krt}(p,r)$ has p, r stored inside, and, on x, makes a self-copy, forms $y = \langle \text{self-copy}, p, x\rangle$, and runs task r on this y.

$$\Phi^{\mathrm{TM}}_{\mathrm{krt}(p,r)}(x) \leq b \cdot (|p| + |r| + |x| + \Phi^{\mathrm{TM}}_{r}(\langle \mathrm{krt}(p,r), p, x\rangle)).$$

Mathematical Preliminaries IV

- Let **T** be a computably axiomatized first order (fo) theory extending fo Peano arithmetic — with numerals represented in base 2 to avoid size blow up from unary representation (for how, see [Bus86, Page 29]) — and which proves no false sentences of fo arithmetic. **T** could be, for example: fo Peano arithmetic, the two-sorted fo Peano arithmetic permitting quantifiers over numbers and sets of numbers [Rog67] (a so arithmetic), ZFC + ones favorite large cardinal axiom, etc.
- The theorems of **T** form a c.e. set, so we can/do fix an automatic theorem prover for **T**.
- By [RC94, Theorem 3.20] a carefully crafted, time-bounded universal simulation of any φ^{TM}-program can be unformly slowed down by a log log factor to run in **LinearTime**.
- Let $\mathbf{T} \vdash_x \mathrm{E}$ mean that a so slowed down, linear-time computable, time-bounded version of this automatic theorem prover proves E from **T** within x steps — that's linear-time in $\langle |\mathrm{E}|, x \rangle$.

Mathematical Preliminaries IV

- Let **T** be a computably axiomatized first order (fo) theory extending fo Peano arithmetic — with numerals represented in base 2 to avoid size blow up from unary representation (for how, see [Bus86, Page 29]) — and which proves no false sentences of fo arithmetic. **T** could be, for example: fo Peano arithmetic, the two-sorted fo Peano arithmetic permitting quantifiers over numbers and sets of numbers [Rog67] (a so arithmetic), ZFC + ones favorite large cardinal axiom, etc.
- The theorems of **T** form a c.e. set, so we can/do fix an automatic theorem prover for **T**.
- By [RC94, Theorem 3.20] a carefully crafted, time-bounded universal simulation of any φ^{TM}-program can be unformly slowed down by a log log factor to run in **LinearTime**.
- Let $\mathbf{T} \vdash_x \mathrm{E}$ mean that a so slowed down, linear-time computable, time-bounded version of this automatic theorem prover proves E from **T** within x steps — that's linear-time in $(|\mathrm{E}| + |x|)$.

Mathematical Preliminaries IV

- Let **T** be a computably axiomatized first order (fo) theory extending fo Peano arithmetic — with numerals represented in base 2 to avoid size blow up from unary representation (for how, see [Bus86, Page 29]) — and which proves no false sentences of fo arithmetic. **T** could be, for example: fo Peano arithmetic, the two-sorted fo Peano arithmetic permitting quantifiers over numbers and sets of numbers [Rog67] (a so arithmetic), ZFC + ones favorite large cardinal axiom, etc.
- The theorems of **T** form a c.e. set, so we can/do fix an automatic theorem prover for **T**.
- By [RC94, Theorem 3.20] a carefully crafted, time-bounded universal simulation of any φ^{TM}-program can be unformly slowed down by a log log factor to run in **LinearTime**.
- Let $\mathbf{T} \vdash_x \mathrm{E}$ mean that a so slowed down, linear-time computable, time-bounded version of this automatic theorem prover proves E from **T** within x steps — that's linear-time in $(|\mathrm{E}| + |x|)$.

Mathematical Preliminaries IV

- Let **T** be a computably axiomatized first order (fo) theory extending fo Peano arithmetic — with numerals represented in base 2 to avoid size blow up from unary representation (for how, see [Bus86, Page 29]) — and which proves no false sentences of fo arithmetic. **T** could be, for example: fo Peano arithmetic, the two-sorted fo Peano arithmetic permitting quantifiers over numbers and sets of numbers [Rog67] (a so arithmetic), ZFC + ones favorite large cardinal axiom, etc.
- The theorems of **T** form a c.e. set, so we can/do fix an automatic theorem prover for **T**.
- By [RC94, Theorem 3.20] a carefully crafted, time-bounded universal simulation of any φ^{TM}-program can be unformly slowed down by a log log factor to run in **LinearTime**.
- Let $\mathbf{T} \vdash_x \mathrm{E}$ mean that a so slowed down, linear-time computable, time-bounded version of this automatic theorem prover proves E from **T** within x steps — that's linear-time in $(|\mathrm{E}| + |x|)$.

Mathematical Preliminaries IV

- Let **T** be a computably axiomatized first order (fo) theory extending fo Peano arithmetic — with numerals represented in base 2 to avoid size blow up from unary representation (for how, see [Bus86, Page 29]) — and which proves no false sentences of fo arithmetic. **T** could be, for example: fo Peano arithmetic, the two-sorted fo Peano arithmetic permitting quantifiers over numbers and sets of numbers [Rog67] (a so arithmetic), ZFC + ones favorite large cardinal axiom, etc.
- The theorems of **T** form a c.e. set, so we can/do fix an automatic theorem prover for **T**.
- By [RC94, Theorem 3.20] a carefully crafted, time-bounded universal simulation of any φ^{TM}-program can be unformly slowed down by a log log factor to run in **LinearTime**.
- Let $\mathbf{T} \vdash_x \mathtt{E}$ mean that a so slowed down, linear-time computable, time-bounded version of this automatic theorem prover proves $\mathtt{E}$ from **T** within x steps — that's linear-time in $(|\mathtt{E}| + |x|)$.

Main Result

Theorem

There exists computable g and small positive $c, d \in \mathbb{N}$ such that, for any p, $|D_{g(p)}| = 2$ and there is a $q \in D_{g(p)}$ for which: $\varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}}$; for all $x \in \mathbb{N}$, $\Phi_q^{\mathrm{TM}}(x) \leq c \cdot (|p| + |x| + \Phi_p^{\mathrm{TM}}(x))$; $|q| \leq d \cdot |p|$; yet $\mathbf{T} \not\vdash \ll \varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}} \gg$.

The proof on the next frame, a rubber wall argument, makes it easily transparent that $\varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}}$. Hence, q is not obfuscated, yet its correctness (at computing φ_p^{TM}) is unprovable in $\mathbf{T}$.

From the time and program size complexity content of the above theorem, q is nicely only slightly more complex than p.

Main Result

Theorem

There exists computable g and small positive $c, d \in \mathbb{N}$ such that, for any p, $|D_{g(p)}| = 2$ and there is a $q \in D_{g(p)}$ for which: $\varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}}$; for all $x \in \mathbb{N}$, $\Phi_q^{\mathrm{TM}}(x) \leq c \cdot (|p| + |x| + \Phi_p^{\mathrm{TM}}(x))$; $|q| \leq d \cdot |p|$; yet $\mathbf{T} \not\vdash \ll \varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}} \gg$.

The proof on the next frame, a rubber wall argument, makes it easily transparent that $\varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}}$. Hence, q is not obfuscated, yet its correctness (at computing φ_p^{TM}) is unprovable in $\mathbf{T}$.

From the time and program size complexity content of the above theorem, q is nicely only slightly more complex than p.

Main Result

Theorem

There exists computable g and small positive $c, d \in \mathbb{N}$ such that, for any p, $|D_{g(p)}| = 2$ and there is a $q \in D_{g(p)}$ for which: $\varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}}$; for all $x \in \mathbb{N}$, $\Phi_q^{\mathrm{TM}}(x) \leq c \cdot (|p| + |x| + \Phi_p^{\mathrm{TM}}(x))$; $|q| \leq d \cdot |p|$; yet $\mathbf{T} \not\vdash \ll \varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}} \gg$.

The proof on the next frame, a rubber wall argument, makes it easily transparent that $\varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}}$. Hence, q is not obfuscated, yet its correctness (at computing φ_p^{TM}) is unprovable in $\mathbf{T}$.
From the time and program size complexity content of the above theorem, q is nicely only slightly more complex than p.

Proof of Main Result

Proof.

By two applications of linear-time krt, if-then-else & $\lambda \mathtt{E}, x\,.\,\mathbf{T} \vdash_x \mathtt{E}$, from any φ-program p, one can find in linear-time (in $|p|$), programs $q_{1,p}$ (for domain(φ_p^{TM}) ∞) & $q_{2,p}$ (for domain(φ_p^{TM}) finite) behaving as follows.

$$\varphi_{q_{1,p}}(x) = \begin{cases} 1 + \varphi_p(x), & \text{if } \mathbf{T} \vdash_x \ll \varphi_{q_{1,p}} = \varphi_p \gg; \\ \varphi_p(x), & \text{otherwise.} \end{cases} \tag{1}$$

$$\varphi_{q_{2,p}}(x) = \begin{cases} 0, & \text{if } \mathbf{T} \vdash_x \ll \varphi_{q_{2,p}} = \varphi_p \gg; \\ \varphi_p(x), & \text{otherwise.} \end{cases} \tag{2}$$

Let $g(p)$ be the canonical index of the set $\{q_{1,p}, q_{2,p}\}$. □

A canonical indexing $\notin$ [Rog67] & just for cardinality 2 yields $g \in \mathbf{LinearTime}$.

[CK09] codes any size finite sets in cubic time & decodes in linear time.

Proof of Main Result

Proof.

By two applications of linear-time krt, if-then-else & $\lambda \mathtt{E}, x\,.\,\mathbf{T} \vdash_x \mathtt{E}$, from any φ-program p, one can find in linear-time (in $|p|$), programs $q_{1,p}$ (for domain$(\varphi_p^{\mathrm{TM}})$ ∞) & $q_{2,p}$ (for domain$(\varphi_p^{\mathrm{TM}})$ finite) behaving as follows.

$$\varphi_{q_{1,p}}(x) = \begin{cases} 1 + \varphi_p(x), & \text{if } \mathbf{T} \vdash_x \ll \varphi_{q_{1,p}} = \varphi_p \gg; \\ \varphi_p(x), & \text{otherwise.} \end{cases} \tag{1}$$

$$\varphi_{q_{2,p}}(x) = \begin{cases} 0, & \text{if } \mathbf{T} \vdash_x \ll \varphi_{q_{2,p}} = \varphi_p \gg; \\ \varphi_p(x), & \text{otherwise.} \end{cases} \tag{2}$$

Let $g(p)$ be the canonical index of the set $\{q_{1,p}, q_{2,p}\}$. □

A canonical indexing $\notin$ [Rog67] & just for cardinality 2 yields $g \in \mathbf{LinearTime}$.

[CK09] codes any size finite sets in cubic time & decodes in linear time.

Proof of Main Result

Proof.

By two applications of linear-time krt, if-then-else & $\lambda \mathtt{E}, x\,.\,\mathbf{T} \vdash_x \mathtt{E}$, from any φ-program p, one can find in linear-time (in $|p|$), programs $q_{1,p}$ (for domain$(\varphi_p^{\mathrm{TM}})$ ∞) & $q_{2,p}$ (for domain$(\varphi_p^{\mathrm{TM}})$ finite) behaving as follows.

$$\varphi_{q_{1,p}}(x) = \begin{cases} 1 + \varphi_p(x), & \text{if } \mathbf{T} \vdash_x \ll \varphi_{q_{1,p}} = \varphi_p \gg; \\ \varphi_p(x), & \text{otherwise.} \end{cases} \tag{1}$$

$$\varphi_{q_{2,p}}(x) = \begin{cases} 0, & \text{if } \mathbf{T} \vdash_x \ll \varphi_{q_{2,p}} = \varphi_p \gg; \\ \varphi_p(x), & \text{otherwise.} \end{cases} \tag{2}$$

Let $g(p)$ be the canonical index of the set $\{q_{1,p}, q_{2,p}\}$. □

A canonical indexing $\notin$ [Rog67] & just for cardinality 2 yields $g \in \mathbf{LinearTime}$.

[CK09] codes any size finite sets in cubic time & decodes in linear-time.

Proof of Main Result

Proof.

By two applications of linear-time krt, if-then-else & $\lambda \mathtt{E}, x\,.\,\mathbf{T} \vdash_x \mathtt{E}$, from any φ-program p, one can find in linear-time (in $|p|$), programs $q_{1,p}$ (for domain$(\varphi_p^{\mathrm{TM}})$ ∞) & $q_{2,p}$ (for domain$(\varphi_p^{\mathrm{TM}})$ finite) behaving as follows.

$$\varphi_{q_{1,p}}(x) = \begin{cases} 1 + \varphi_p(x), & \text{if } \mathbf{T} \vdash_x \ll \varphi_{q_{1,p}} = \varphi_p \gg; \\ \varphi_p(x), & \text{otherwise.} \end{cases} \tag{1}$$

$$\varphi_{q_{2,p}}(x) = \begin{cases} 0, & \text{if } \mathbf{T} \vdash_x \ll \varphi_{q_{2,p}} = \varphi_p \gg; \\ \varphi_p(x), & \text{otherwise.} \end{cases} \tag{2}$$

Let $g(p)$ be the canonical index of the set $\{q_{1,p}, q_{2,p}\}$. □

A canonical indexing ∉ [Rog67] & just for cardinality 2 yields $g \in \mathbf{LinearTime}$.

[CK09] codes any size finite sets in cubic time & decodes in linear-time.

Proof of Main Result

Proof.

By two applications of linear-time krt, if-then-else & $\lambda \mathtt{E}, x\,.\,\mathbf{T} \vdash_x \mathtt{E}$, from any φ-program p, one can find in linear-time (in $|p|$), programs $q_{1,p}$ (for domain$(\varphi_p^{\mathrm{TM}})$ ∞) & $q_{2,p}$ (for domain$(\varphi_p^{\mathrm{TM}})$ finite) behaving as follows.

$$\varphi_{q_{1,p}}(x) = \begin{cases} 1 + \varphi_p(x), & \text{if } \mathbf{T} \vdash_x \ll \varphi_{q_{1,p}} = \varphi_p \gg; \\ \varphi_p(x), & \text{otherwise.} \end{cases} \tag{1}$$

$$\varphi_{q_{2,p}}(x) = \begin{cases} 0, & \text{if } \mathbf{T} \vdash_x \ll \varphi_{q_{2,p}} = \varphi_p \gg; \\ \varphi_p(x), & \text{otherwise.} \end{cases} \tag{2}$$

Let $g(p)$ be the canonical index of the set $\{q_{1,p}, q_{2,p}\}$. □

A canonical indexing $\notin$ [Rog67] & just for cardinality 2 yields $g \in \mathbf{LinearTime}$.

[CK09] codes any size finite sets in cubic time & decodes in linear-time.

Proof of Main Result

Proof.

By two applications of linear-time krt, if-then-else & $\lambda \mathtt{E}, x\,.\,\mathbf{T} \vdash_x \mathtt{E}$, from any φ-program p, one can find in linear-time (in $|p|$), programs $q_{1,p}$ (for domain$(\varphi_p^{\mathrm{TM}})$ ∞) & $q_{2,p}$ (for domain$(\varphi_p^{\mathrm{TM}})$ finite) behaving as follows.

$$\varphi_{q_{1,p}}(x) = \begin{cases} 1 + \varphi_p(x), & \text{if } \mathbf{T} \vdash_x \ll \varphi_{q_{1,p}} = \varphi_p \gg; \\ \varphi_p(x), & \text{otherwise.} \end{cases} \tag{1}$$

$$\varphi_{q_{2,p}}(x) = \begin{cases} 0, & \text{if } \mathbf{T} \vdash_x \ll \varphi_{q_{2,p}} = \varphi_p \gg; \\ \varphi_p(x), & \text{otherwise.} \end{cases} \tag{2}$$

Let $g(p)$ be the canonical index of the set $\{q_{1,p}, q_{2,p}\}$. □

A canonical indexing $\notin$ [Rog67] & just for cardinality 2 yields $g \in \mathbf{LinearTime}$.

[CK09] codes any size finite sets in cubic time & decodes in linear-time.

Next Results

It's interesting to ask: can $|D_{g(p)}| = 2$ above be instead $|D_{g(p)}| = 1$?

For the natural acceptable programming system φ^{TM}, it cannot:

Theorem

It is not the case that there exists computable g such that, for any p, for $q = g(p)$: $\varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}}$, yet $\mathbf{T} \not\vdash \ll \varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}} \gg$.

Proof.

Idea: for φ^{TM} one can pretty much provably in **PA** fix point any computable g. N.B. One need not (in general, cannot) prove in **PA** arbitrary computable g's totality. □

Subtlety: we can fully constructively provide and prove correct a pathological representation of φ^{TM} for fo Peano Arithmetic (**PA**), so that the theorem just above fails; we make not much about program equivalence provable in $\mathbf{T}$ re this representation.

Next Results

It's interesting to ask: can $|D_{g(p)}| = 2$ above be instead $|D_{g(p)}| = 1$?
For the natural acceptable programming system φ^{TM}, it cannot:

Theorem

It is not the case that there exists computable g such that, for any p, for $q = g(p)$: $\varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}}$, yet $\mathbf{T} \not\vdash \ll \varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}} \gg$.

Proof.

Idea: for φ^{TM} one can pretty much provably in **PA** fix point any computable g. N.B. One need not (in general, cannot) prove in **PA** arbitrary computable g's totality. □

Subtlety: we can fully constructively provide and prove correct a pathological representation of φ^{TM} for fo Peano Arithmetic (**PA**), so that the theorem just above fails; we make not much about program equivalence provable in $\mathbf{T}$ re this representation.

Next Results

It's interesting to ask: can $|D_{g(p)}| = 2$ above be instead $|D_{g(p)}| = 1$?
For the natural acceptable programming system φ^{TM}, it cannot:

Theorem

It is not the case that there exists computable g such that, for any p, for $q = g(p)$: $\varphi^{\mathrm{TM}}_q = \varphi^{\mathrm{TM}}_p$, yet $\mathbf{T} \not\vdash \ll \varphi^{\mathrm{TM}}_q = \varphi^{\mathrm{TM}}_p \gg$.

Proof.

Idea: for φ^{TM} one can pretty much provably in **PA** fix point any computable g. N.B. One need not (in general, cannot) prove in **PA** arbitrary computable g's totality. □

Subtlety: we can fully constructively provide and prove correct a pathological representation of φ^{TM} for fo Peano Arithmetic (**PA**), so that the theorem just above fails; we make not much about program equivalence provable in $\mathbf{T}$ re this representation.

Next Results

It's interesting to ask: can $|D_{g(p)}| = 2$ above be instead $|D_{g(p)}| = 1$?
For the natural acceptable programming system φ^{TM}, it cannot:

Theorem

It is not the case that there exists computable g such that, for any p, for $q = g(p)$: $\varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}}$, yet $\mathbf{T} \not\vdash \ll \varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}} \gg$.

Proof.

Idea: for φ^{TM} one can pretty much provably in **PA** fix point any computable g. N.B. One need not (in general, cannot) prove in **PA** arbitrary computable g's totality. □

Subtlety: we can fully constructively provide and prove correct a pathological representation of φ^{TM} for fo Peano Arithmetic (**PA**), so that the theorem just above fails; we make not much about program equivalence provable in **T** re this representation.

Next Results

It's interesting to ask: can $|D_{g(p)}| = 2$ above be instead $|D_{g(p)}| = 1$?
For the natural acceptable programming system φ^{TM}, it cannot:

Theorem

It is not the case that there exists computable g such that, for any p, for $q = g(p)$: $\varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}}$, yet $\mathbf{T} \not\vdash \ll \varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}} \gg$.

Proof.

Idea: for φ^{TM} one can pretty much provably in **PA** fix point any computable g. N.B. One need not (in general, cannot) prove in **PA** arbitrary computable g's totality. □

Subtlety: we can fully constructively provide and prove correct a pathological representation of φ^{TM} for fo Peano Arithmetic (**PA**), so that the theorem just above fails; we make not much about program equivalence provable in **T** re this representation.

Next Results

It's interesting to ask: can $|D_{g(p)}| = 2$ above be instead $|D_{g(p)}| = 1$?
For the natural acceptable programming system φ^{TM}, it cannot:

Theorem

It is not the case that there exists computable g such that, for any p, for $q = g(p)$: $\varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}}$, yet $\mathbf{T} \not\vdash \ll \varphi_q^{\mathrm{TM}} = \varphi_p^{\mathrm{TM}} \gg$.

Proof.

Idea: for φ^{TM} one can pretty much provably in **PA** fix point any computable g. N.B. One need not (in general, cannot) prove in **PA** arbitrary computable g's totality. □

Subtlety: we can fully constructively provide and prove correct a pathological representation of φ^{TM} for fo Peano Arithmetic (**PA**), so that the theorem just above fails; we make not much about program equivalence provable in **T** re this representation.

References

S. Buss.

Bounded Arithmetic.
Bibliopolis, Naples, 1986.
Revision of 1985 Ph.D. Thesis: `http://www.math.ucsd.edu/~sbuss/ResearchWeb/BAthesis/` (Department of Mathematics, Princeton University).

J. Case and T. Kötzing.

Difficulties in forcing fairness of polynomial time inductive inference.
In *20th International Conference on Algorithmic Learning Theory (ALT'09)*, volume 5809 of *Lecture Notes in Artificial Intelligence*, pages 263–277, 2009.

J. Royer and J. Case.

Subrecursive Programming Systems: Complexity and Succinctness.
Research monograph in *Progress in Theoretical Computer Science*. Birkhäuser Boston, 1994.

H. Rogers.

Theory of Recursive Functions and Effective Computability.
McGraw Hill, New York, 1967.
Reprinted, MIT Press, 1987.