1992

R.T.P. Fernando

Provably recursive programs

Computer Science/Department of Software Technology Report CS-R9212 March

CWI is het Centrum voor Wiskunde en Informatica van de Stichting Mathematisch Centrum

CWI is the Centre for Mathematics and Computer Science of the Mathematical Centre Foundation

CWI is the research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a non-profit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands organization for scientific research (NWO).

Provably Recursive Programs

Tim Fernando
fernando@cwi.nl

CWI, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Abstract. The "absoluteness" of the recursion-theoretic notion of computation is contrasted with the "relative" character of formal proofs. Certain cracks in the correspondence, relativized to a (single) formal theory, between proofs and programs (of bounded complexity) are exposed, and linked to (in-)completeness for Π^0_1-sentences.

1991 Mathematics Subject Classification: 03B70.
1991 Computing Reviews Categories: D.3.1.
Key words and phrases: absoluteness, Church's thesis, incompleteness, provably recursive functions, proofs as programs.
Note: The present paper is a revision of (and supersedes) the conference paper Fernando [7]. The author is gratefully indebted to Prof. S. Feferman for supervision, to CWI for refuge, and to the Netherlands Organization for Scientific Research (NWO project NF 102/62-356, 'Structural and Semantic Parallels in Natural Languages and Programming Languages') for funding.

The present paper is, very broadly speaking, about the relationship between logical notions of computation and of deduction. The standard recursion-theoretic account of (finitary) computation (in terms, say, of Turing machines) is, as Gödel [10] remarked in 1946, "absolute" in that it does not depend "on the formalism chosen." By contrast, the notion of a deduction (i.e., proof) takes on a definite meaning only when relativized to a formal theory, at which point the notion then becomes subject to (Gödel) incompleteness. Church's thesis, on the other hand, asserts that the recursion-theoretic notion of computation[1] is stable and robust. So, from a formal logical perspective, computation and deduction differ fundamentally.

Another point of view, common in so-called constructive mathematics, is that proofs and programs come to the same thing. Typically, the programs considered under such a view terminate on all inputs — i.e., are total. This is because the view is often based on some scheme for extracting programs from proofs of Π^0_2-sentences — programs that ever since Kleene's 1945 *realizability* interpretation (Kleene [11]) have been required to be total —, or on a *Curry-Howard-de Bruijn isomorphism* between a certain system of natural deduction and a typed λ-calculus, both of which enjoy normalization (see, for example, Girard, Lafont and Taylor [8]). Inasmuch as

[1] There is some vagueness here as to whether functions or programs are meant. Rigorous statements refer to functions, although a stronger assertion concerning a notion intermediate between that of functions and (a particular system of) programs could be made, if only that notion could be spelled out precisely.

Report CS-R9212
ISSN 0169-118X
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

the basis of such a view can be formalized, it is natural to test the correspondence between proofs and programs relativized to some formal theory T. A minimal requirement on a program captured under such a correspondence is that T prove its totality. Indeed, a standard proof-theoretic measure of the computational content of a formal theory T is the collection of its so-called *provably recursive functions*, viz., the functions computed by programs T proves total. An investigation of the programs behind such functions, however, reveals defects in the correspondence between proofs and programs that can be traced to the conflict between "absoluteness" and "relativeness" mentioned above.

1 Basic definitions and facts

To be precise, fix a standard enumeration $\{\varphi_e\}_{e<\omega}$ of unary partial recursive functions, for example, the Gödel numbers of Turing machines. (The choice of basic computational mechanisms is inessential.) Let

- L be some formal predicate language (given say by $\{+, \times, 0, 1\}$ and/or symbols for all primitive recursive functions) from which formal theories T are drawn[2],
- T be an L-formula expressing (the appropriate analog for the enumeration to) Kleene's T_1-predicate, where recall that the intuition is that

$$\mathrm{T}_1(e, n, m) \text{ iff } m \text{ codes a computation of } e \text{ on input } n \ ,$$

  and
- $Tot(x)$ be $\forall u \exists v T(x, u, v)$.

The *provably recursive functions of* T form the class

$$\mathsf{Fcn}(T) := \{\varphi_e \mid T \vdash Tot(\dot{e})\}$$

(where $\dot{e}$ is the L-numeral of e), while the *provably recursive programs of* T form the class

$$\mathsf{Pgm}(T) := \{e \mid T \vdash Tot(\dot{e})\} \ .$$

Next, for every program e, let

- c_e be the partial recursive function that returns computations (in the sense of T_1)

$$c_e(n) \simeq \mu m \ \mathrm{T}_1(e, n, m) \ ,$$

  and

[2] The logic of the theories considered below is assumed to be classical, the point being that since all intuitionistic proofs are classical, limitations on classical proofs carry over to intuitionistic proofs. (Furthermore, it is well-known that for various theories the choice of logic is immaterial. See, for starters, Gödel [9], and A.S. Troelstra's accompanying introductory note.)

– t_e be the partial recursive function that gives the running time of e

$$t_e(n) \simeq \text{number of steps before } e \text{ halts on input } n \ .$$

Observe that t_e and c_e are primitive recursively related — i.e., $t_e = h \circ c_e$ and $c_e = g_e \circ t_e$ for primitive recursive functions h and g_e. Define the class E_T of *programs (with computations) bounded by* T as the collection of programs e such that $c_e \in \mathsf{Fcn}(T)$, or equivalently, if T contains primitive recursive arithmetic (PRA), $t_e \in \mathsf{Fcn}(T)$. Note that a program e for a function in $\mathsf{Fcn}(T)$ need *not* belong to E_T since c_e might not be in $\mathsf{Fcn}(T)$. Some basic relationships between the notions are given by

Proposition 1. *Let T be a theory containing PRA.*

1. $\mathsf{Pgm}(T) \subseteq E_T$.
2. *If e computes a total recursive function with $t_e \leq t_{e'}$ a.e. for some $e' \in \mathsf{Pgm}(T)$, then $e \in E_T$.*
3. *If $e \in E_T$ then $\varphi_e \in \mathsf{Fcn}(T)$.*

Proof. 1: For every program e, c_e is computed by the program that on input n looks for the first m such that $\mathrm{T}_1(e, n, m)$. The latter program belongs to $\mathsf{Pgm}(T)$ when $e \in \mathsf{Pgm}(T)$.

2: Observe that there is an (increasing) primitive recursive function g such that for all $\hat{e}$,

$$c_{\hat{e}}(n) \simeq \mu m < g(t_{\hat{e}}(n))\ \mathrm{T}_1(\hat{e}, n, m) \ .$$

Now, let e compute a total function with $t_e(n) \leq t_{e'}(n) + N$, for some $e' \in \mathsf{Pgm}(T)$ and fixed $N < \omega$. Then c_e is computed by the program that on input n finds the first $m < g(t_{e'}(n) + N)$ such that $\mathrm{T}_1(e, n, m)$. This program is provably total in T.

3: If $e \in E_T$, then φ_e is the composition of a function in $\mathsf{Fcn}(T)$ (namely, c_e) with Kleene's output extraction function (also, in $\mathsf{Fcn}(T)$). $\dashv$

It follows that, in accordance with practice, $\mathsf{Fcn}(T)$ forms a (time-)complexity class (when T contains PRA) in the sense that

$$(*)\ \text{whenever } T \vdash Tot(e) \text{ and } t_{e'} < t_e,\ \varphi_{e'} \in \mathsf{Fcn}(T).$$

$(*)$ suggests that perhaps there is a nice match between computation and deduction so long as bounds on computation are imposed. But computation has to do with programs, and the question arises as to whether $(*)$ can be strenghtened to

$$(**)\ \text{whenever } T \vdash Tot(e) \text{ and } t_{e'} < t_e,\ T \vdash Tot(e').$$

Unfortunately, $(**)$ in general fails. An easy counter-example is provided by the following definitions (where $\bot$ is some contradiction such as 0=1) pointed out to the author by S. Feferman and also S. Buss

e' : on input n, check if n codes a T-proof of $\bot$;
return 0 if it does, and loop, otherwise.

e : on input n, check if n codes a T-proof of $\bot$;
in either case, return 0 after calculating $1 + 1$.

Note that $T \vdash Tot(e)$, but that if T is consistent, $T \nvdash Tot(e')$ (by Gödel's second incompleteness theorem), even though $t_{e'} < t_e$.

As pathological as the above counter-example may seem, let us try to understand what is going on here. Toward this end, call a theory T *downward closed (d.c.)* if for every e it proves total, every e' computing φ_e with $t_{e'} < t_e$ is also proved total.

Theorem 2. *For consistent extensions T of PRA, the following are equivalent:*

1. $\mathsf{Pgm}(T) = E_T$.
2. *T is d.c.*
3. *T proves every true Π^0_1-sentence.*

Proof. 1 $\Rightarrow$ 2: An immediate consequence of Proposition 1.

2 $\Rightarrow$ 3: The idea is contained in the counter-example above: given a decidable predicate $\chi(x)$ for which $\forall x \chi(x)$ is true, build padded programs for the constant function 0 that on input n checks if $\chi(n)$ holds

e' : on input n, check if $\chi(n)$ holds;
return 0 if it does, and loop, otherwise.
e : on input n, check if $\chi(n)$ holds;
in either case, return 0 after calculating $1 + 1$.

(A somewhat more elaborate argument that brings out the notion of a "d.c. reduction" is given in Fernando [7].)

3 $\Rightarrow$ 1: For $\mathsf{Pgm}(T)$ to contain E_T, it suffices that T prove the true Π^0_1-sentences

$$\forall u \forall v \quad \mathcal{T}(\dot{e'}, u, v) \wedge \forall w < v \neg \mathcal{T}(\dot{e'}, u, w) \supset \mathcal{T}(\dot{e}, u, \mathcal{U}(v)) \,,$$

(where $\mathcal{U}$ expresses Kleene's output extraction function) for all pairs (e, e') such that $e \in E_T$, $e' \in \mathsf{Pgm}(T)$, and for every $n < \omega$, $\mathrm{T}_1(e, n, \varphi_{e'}(n))$. $\dashv$

Theorem 2 establishes a tight connection between bounded computation and Π^0_1-sentences, showing, in particular, that no r.e. extension of *PRA* can be d.c.

Since not all primitive recursive functions may be deemed "feasible", it would be natural to try to push down the assumed strength of T in Theorem 2 below primitive recursive arithmetic (say to a system whose provably recursive functions are the Kalmar elementary functions adequate for elementary recursion-theoretic results such as the Kleene normal form theorem). Alternatively, the notion of downward closure can be refined to focus, for instance, on programs of a certain complexity.

Theorem 3. *Assume T is a consistent theory that proves total some program $\hat{e}$ with quadratic running time. Suppose further that for every $e \in \mathsf{Pgm}(T)$ with quadratic running time, if e' computes φ_e with running time a.e. less than that of e, then $e' \in \mathsf{Pgm}(T)$. Then T is Π^0_1-hard.*

Proof. Let f be the total recursive function that on input e returns the code of a program that on input n does the following:

run program e on input e for at most (length of) n steps; if this computation (of $\varphi_e(e)$) terminates within (length of) n steps, loop forever; otherwise, run $\hat{e}$ on input n.

Now, it is easy to construct a "padded" copy of $\hat{e}$ in $\mathsf{Pgm}(T)$ with quadratic running time greater (a.e.) than that of every program $f(e)$ that is total. Hence, it follows that for every $e < \omega$,

$$\varphi_e(e)\uparrow \Leftrightarrow T \vdash \dot{Tot}(\dot{f}(\dot{e}))\ ,$$

as required. $\dashv$

Theorem 3 (the proof of which is essentially that of the Rice-Shapiro theorem) dramatizes the futility of introducing fast growing functions to obtain an r.e. theory whose provably recursive programs include all programs of a modest complexity. The argument behind it and $2 \Rightarrow 3$ in Theorem 2 depend crucially on "reduction through padding" (explained in detail in Fernando [7]). That is, an appeal is made to programs that are easily recognized to be non-optimal. It is natural to ask whether such programs can be abstracted away or whether padding is an intrinsic (unavoidable) feature of computation. This question is taken up in the next section.

2 Optimizability and speed-ups

Contrasting $(*)$ with Theorems 2 and 3, an obvious moral to draw is that care must be exercised when passing from a program e to the function φ_e. Complexity is a property of programs, and is only indirectly defined for functions. A fundamental result in complexity theory, Blum's speed-up theorem [1], states that there are functions, every program for which can be sped up. By contrast, a diagonalization argument yields an "anti-speed-up theorem" for programs extracted from specifications.

2.1 A limit on provable speed-ups relative to specifications

Fix Gödel numberings $\#$ of L-formulas and finite sequences of L-formulas, writing $d(n)$ and $D(n)$ respectively for the L-formula and the finite sequence of L-formulas coded by n. Assuming this is carried out reasonably (in say polynomial time), the complexity of membership in

$$A := \{(e,s,p,t) \mid D(p) \text{ is a proof from } d(t) \text{ of } d(s)(\dot{e})\}$$

can be bounded by some primitive recursive function g (e.g., a polynomial); more precisely, for all $e,s,p,t < \omega$, the question "$(e,s,p,t) \in A$?" can be decided within time $g(max\{e,s,p,t\})$.

Theorem 4. *There is a total recursive function $r(k)$, greater than 0 on all k, such that for every consistent, finite L-theory T, and every e such that φ_e is total, there is a program $\hat{e}$ computing φ_e such that for every e' for which $T \vdash$ "$\varphi_e = \varphi_{e'}$",*

$$t_{\hat{e}}(k) < r(k) \times t_{e'}(k) \quad \text{a.e.}$$

Proof. Given a finite L-theory T, and an e for which φ_e is total, define $\hat{e}$ as follows: on input k,

calculate $E_k := \{e' < k \mid \exists p < k\ (e', \#\chi, p, \#\bigwedge T) \in A\}$ and then interleave simulations of e and every $e' \in E_k$ on input k, halting as soon as one of the computations stops.

Observe that for every e' for which $T \vdash$ "$\varphi_{e'} = \varphi_e$",

$$t_{\hat{e}}(k) < k^2 \times g(k) \times t_e(k) \quad \text{a.e.}$$

$\dashv$

Since all r.e. theories can be finitely axiomatized (if necessary by expanding the language), the theorem easily generalizes from finite T's to r.e. T's.

It should be emphasized that Theorem 4 is a result about specifications — the assumption that $T \vdash$ "$\varphi_{e'} = \varphi_e$" is crucial (although natural when studying the mechanical extraction of programs). Without this restriction, one can argue as follows. Given a program e and a function $f \in \mathsf{Fcn}(T)$, build the program $e_f \in \mathsf{Pgm}(T)$ that on input n simulates $f(n)$ steps of e on input n. (In terms of the Kleene normal form theorem, the idea can be depicted roughly as

$$f \in \mathsf{Fcn}(T) \quad : \quad e \mapsto U(\mu m < f(\cdot)\ \mathrm{T}_1(e, \cdot, m)) \in \mathsf{Pgm}(T)\ .)$$

J. Mitchell has described this as a "positive result", noting that a simulation of a program suggests that "essentially the same algorithm" is implemented. It is unlikely, however, that a programmer would equate, on the one hand, writing some program e with, on the other hand, writing (a description of) e *and* a program that simulates e some pre-determined number of times. It is not even possible to determine that the same function is computed without the knowledge that enough steps of simulation are allowed — which is just the information needed when extracting a program from a proof of a Π^0_2 specification!

2.2 Non-optimality of normalization

An example of a problem that can be sped-up infinitely often is normalization of a type system rich enough under "propositions as types" to prove Blum's speed-up theorem. More precisely, in the specific case of system F (e.g., see Girard, Lafont, Taylor [8]) and second-order Peano arithmetic PA_2,

Proposition 5. *For every program e that reduces a lambda term typable in* F *to its normal form, and for every (say increasing) $g \in \mathsf{Fcn}(PA_2)$, there is a program e' computing the same function as e with $g(t_{e'}(A)) < t_e(A)$ for infinitely many lambda terms A typable in* F.

Proof. Recall that $\mathsf{Fcn}(PA_2)$ consists exactly of the functions representable in F via Church numerals. Now, because Blum's speed-up theorem can be proved in PA_2, there is for any $g \in \mathsf{Fcn}(PA_2)$, some function f also provably recursive in PA_2 admitting g-speed-up. The "infinitely many lambda terms" mentioned above can be obtained by fixing some lambda term representing f and applying that to Church numerals. $\dashv$

In retrospect, Proposition 5 is hardly surprising, given the difficulty in optimizing a simulator for a sufficiently rich collection of programs. The result does suggest, however, that some notion of "metaprogramming" may be useful. Equating a simulator (for certain programs) with the inference engine of a (fixed) formal system, the proposition displays an advantage in working with different formal systems (over a single formal system, or program for normalization). The "infinitely often" improvement roughly parallels certain classical results (going back to Gödel) on "abbreviating proofs by adding new axioms" (Ehrenfeucht and Mycielski [3]).

3 Discussion

There is something appealing about identifying (or in some way unifying) notions of computation, deduction and truth that is difficult to resist, whatever obstacles may stand in the way of such a unification. And so, in intuitionistic logic, truth is equated with the existence of a proof, and, under the "propositions as types" paradigm, proofs are, in turn, identified with programs. This viewpoint has led to various schemes for extracting programs from proofs. On the one hand, it has become clear that extracting "feasible" programs from proofs is a non-trivial task (suggesting that the notion of a proof is richer than that of a program). On the other hand, proof-theorists talk about the bounds on the complexity of computations extracted from proofs in formal theories wherein such schemes can be formalized. The point emphasized in the present paper, however, is that Gödel incompleteness stands in sharp contrast to the "completeness" of the recursion-theoretic analysis of computation (although here again a careful distinction must be made between interpreting Church's thesis as a statement narrowly about computable functions or more broadly about "algorithms"). The question arises as to whether the very idea of extracting a program from a proof makes sense, or whether, in fact, programs precede proofs (and types, introduced to reason about a pre-existing collection of programs).

A number of "semantic" investigations into programming languages have moved away from recursion theory (and its untyped character), replacing talk of Turing machines by (extensional) functions subject to all manners of type structure. Whether or not the programmer finds such abstractions helpful, he or she must eventually face the reality of compilation to machine code. Inasmuch as the (untyped) recursion-theoretic picture of computation is an adequate abstraction of machine code, and inasmuch as Church's thesis is accepted, it is proper to base one's semantic conception of programming languages ultimately on this "absolute" reality described by recursion theory. There is, under Church's thesis, no way (in practice) to escape this core, but rather the question is how fully does a particular structured (higher-level) programming language cover this core. What are the costs of the discipline that it imposes? The present paper exposes holes left by programs of bounded complexity that are excluded in a programming language that guarantees termination. Can these holes be filled in a systematic fashion? In other words, is there a "natural" way to bridge the gap between $\mathsf{Pgm}(T)$ and E_T for an r.e. theory T?

An old result of Turing's described in Feferman [4] (and to which the reader is referred for background for the present paragraph) is that the "progression" $\{T_\alpha\}_\alpha$ of

theories starting with some effectively presented theory T_0 and continuing according to

$$T_{\alpha+1} := T + Con(T)$$
$$T_\lambda := \bigcup_{\alpha<\lambda} T_\alpha \quad \text{for limit } \lambda$$

(where $Con(T)$ says "T is consistent") proves all true Π^0_1-sentences. Theorem 2 suggests that such a progression provides an "absolute" notion of proof that coincides with bounded computation, but that goes beyond strictly "constructive" (in the precise sense of r.e.) means. (See also Parson's introductory note to Gödel [10].) The α's above, however, are not actually ordinals, but ordinal notations in some Π^1_1-complete subset $\mathcal{O}$ of ω. The structure of $\mathcal{O}$ represents a formidable challenge in proof theory — in particular, the characterization of a "natural path" in $\mathcal{O}$. A fundamental result in this field (due to Feferman and Spector [6]) is that Π^1_1 paths through $\mathcal{O}$ (i.e., naming every recursive ordinal) are incomplete for Π^0_1-sentences, and that such paths exist. (Turing's completeness result mentioned above depends heavily on ordinal notations that encode information extraneous to the ordinal named.) Pursuing the intuition that programs correspond to proofs, programs in E_T not proved total in such a Π^1_1-path might, perhaps, safely be dismissed as pathological. More generally, one might associate the problem of characterizing natural paths in $\mathcal{O}$ with the problem of characterizing "natural programs" in E_T (for instance, by identifying natural programs with those obtained by natural paths). The question is whether such an association would shed any light on either problem.

Another approach to filling in E_T is to allow the introduction of programs outside of E_T. (Kreisel [12] pointed out some time ago that adding true Π^0_1-sentences, such as consistency, does not yield new provably recursive functions.) Proofs of consistency that proceed by normalization arguments which bring in new provably recursive functions are examples of such. To the extent that these are to be preferred over proofs that consist simply (as in the progression above) of asserting the conclusion (i.e., consistency), there is nothing wrong with stepping outside E_T (in an attempt to fill it). In this vein, Colson [2] isolates a natural primitive recursive fragment of Gödel's system T that he shows excludes an optimal algorithm (living in the larger system) for a primitive recursive function. (The necessity in this case, however, of passing to a system whose provably recursive functions exceed the primitive recursive ones is disputed in Feferman [5].)

References

1. Manuel Blum. A machine-independent theory of the complexity of recursive functions. *J. Assoc. Computing Machinery*, 14, 1967.
2. Loic Colson. About primitive recursive algorithms. In G. Ausiello et al, editor, *Proc. ICALP '89*, LNCS 372. Springer-Verlag, Berlin, 1989.
3. Andrzej Ehrenfeucht and Jan Mycielski. Abbreviating proofs by adding new axioms. *Bulletin of the American Mathematical Society*, 77(3), 1971.
4. Solomon Feferman. Turing in the land of 0(z). In R. Herken, editor, *The universal Turing machine.* Kammerer and Unverzagt, 1988.

5. Solomon Feferman. Logics for termination and correctness of functional programs, II. Leeds Proof Theory '90, to appear.
6. Solomon Feferman and Clifford Spector. Incompleteness along paths in progressions of theories. *Journal of Symbolic Logic*, 27, 1962.
7. Tim Fernando. Provably recursive programs and program extraction. In J. Leach Albert et al, editor, *Proc. ICALP '91*, LNCS 510. Springer-Verlag, Berlin, 1991.
8. Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types.* Cambridge University Press, Cambridge, 1989.
9. Kurt Gödel. Zur intuitionistischen arithmetik und zahlentheorie. In S. Feferman et al, editor, *Collected works, volume* I. Oxford University Press, 1986.
10. Kurt Gödel. Remarks before the Princeton bicentennial conference on problems in mathematics. In S. Feferman et al, editor, *Collected works, volume* II. Oxford University Press, 1990.
11. S.C. Kleene. On the interpretation of intuitionistic number theory. *J. Symbolic Logic*, 10, 1945.
12. Georg Kreisel. On the interpretation of non-finitist proofs, I. *Journal of Symbolic Logic*, 16, 1951.

This article was processed using the LaTeX macro package with LLNCS style