A Foundational Delineation of Computational Feasibility

Daniel Leivant*

Abstract

A function over $\{0,1\}^*$ is in P-Time iff it is computed by a program which can be proved correct in second-order logic with set-existence (comprehension) restricted to positive quantifier-free formulas. This set-existence principle captures formally the view of infinite totalities as evolving, not completed, entities.

1 Introduction

1.1 Feasibility and P-Time

Feasible computing has been identified for long with computability within deterministic polynomial time, primarily on practical and circumstantial grounds: P-Time functions are easily defined and computed, and are closed under many natural operations; and most known worst-case lower-bounds are either bounded by polynomials of small degrees, which are clearly feasible, or are at least exponential, and clearly non-feasible. The central importance of P-Time has been contested as of late, notably because feasible probabilistic classes might subsume P-Time in their practical significance, and because bounds such as $n^{\log\log n}$ are more feasible in practice than say n^{1000}. At the same time, the fundamental nature of P-Time has been reaffirmed repeatedly by various characterizations and stability results. For example, relations computable in P-Time over enumerated finite structures are the same as the ones computable by recursion equations [Saz80,Gur83] or by pure uninterpreted logic programs [Pap85], or by alternating multi-head automata [CKS81,Gur87]; they are also the same as the relations defined by positive first-order fixpoints [Var82,Imm86], or by first-order inflationary fixpoints [GS86,Lei90a], or by alternating transitive-closure [Imm87]. The P-Time functions over $\mathbf{N}$ have, among others, characterizations in terms of a subrecursive schema [Cob65], provability in a weak system for arithmetic [Bus86], and typability in a bounded version of linear logic [GSS89].

These characterizations testify to the significance of P-Time, but they all seem to lack a principle directly pertinent to feasibility, one that would justify the identification of P-Time with feasible computing. Our aim here is to propose such a principle.

1.2 The ontology of numeric terms

Computational feasibility is closely related to the ontology of numeric terms. As soon as non-feasible functions are named, they take a life of their own, and ontologically problematic natural numbers become easily nameable, such as $3 \uparrow\uparrow 5 =_{df} 3^{3^{3^3}}$. In particular, once exponentiation is admitted, then very short terms exist whose numeric values exceeds not only human imagination, but also possible realization in the physical world: $3 \uparrow\uparrow 5$ could not be spelled out as a decimal numeral even by quark-size computers filling up the observable universe and working concurrently since the big bang at a speed that exceeds the limitations of quantum mechanics.

The abyss between the value and the notation-size of such terms has been addressed by a number of mathematicians and philosophers, including Bernays [Ber35], van Danzig [Dan56], Yessenin-Volpin [Yes70], Isles [Isl91], and Nelson [Nel86]. Gandy [Gan89] concludes that "very large numbers are abstract not concrete (not potentially concrete) objects: they are more akin to infinite sets than to concretely presented numbers."

1.3 Predicativity and potential infinities

Basic infinite sets, such as the set $\mathbf{N}$ of the natural numbers or the first inaccessible cardinal, are conceptualized as being generated by a process. To be admitted as legitimate, we must assume that some "universe" exists within which that process can be applied "indefinitely". Similarly, our belief that $3 \uparrow\uparrow 5$ denotes a natural numbers is based on the conviction that the calculation of that term will be completed eventually. Of course, we can support that conviction with a proof

*Author's current address: SCS, Carnegie Mellon University, Pittsburgh, PA 15213. Address effective Fall 1991: Computer Science Department, Indiana University, Bloomington, IN 47405.

by induction, but, as we shall see, for that proof to make sense we must admit that *infinite sets exist as complete totalities.*

Less than a century ago, the legitimacy of infinite sets as completed totalities was not as universally taken for granted as it is today, under the influence of Cantorian set theory. Hilbert had hoped to shelter Mathematics from the potential dangers of actual infinities by reducing it to its finitistic fragment. An important aspect of Brouwer's intuitionistic foundations is the insistence that infinite totalities are only unbounded constructions: "The natural numbers, though treated, constitute only a potential totality in constructive mathematics" [Kre61].

Recall that a definition of a set X is *impredicative* if it refers to a collection of which X is an element. Uncontrolled impredicativity leads to contradictions, as in Russell's Paradox. However, impredicative definitions abound in Mathematical Analysis, where real numbers (i.e. subsets of $\mathbf{N}$ or functions over $\mathbf{N}$, depending on the representation) are defined in terms of quantification over all reals. We normally expect no contradiction to arise, because we implicitly assume that the power set of $\mathbf{N}$, $\mathcal{P}\mathbf{N}$, is given as a completed totality prior to the definition of any particular members thereof.[1] In predicative systems of analysis[2] one refrains from assuming the power set of $\mathbf{N}$ as given, albeit $\mathbf{N}$ is assumed as a completed totality. This implies that a subset of $\mathbf{N}$ cannot be defined in terms of quantification over $\mathcal{P}\mathbf{N}$, and circular definitions are thereby excluded.

An argument raised by Nelson [1986] is that the definitions of $\mathbf{N}$ are also circular: the generative (inductive) definition, as a set constructed by repeated application of the successor function, presupposes an understanding of $\mathbf{N}$ itself (specifically when Induction is proposed as the formal justification of the process). The definition of $\mathbf{N}$ as the intersection of all sets containing 0 and closed under successor presupposes that such a set exists, and moreover uses a blatantly impredicative quantification over sets.[3]

[1] Impredicative definitions of this form are captured by the Subset (Separation) axiom schema of Zermelo's Set Theory.

[2] Predicative Analysis goes back to Borel and the semi-intuitionists of the turn of the century, and has been revived by Kreisel, Feferman, Wang, Schüttee, and many others.

[3] Shoenfield and Wang (in conversation with Kreisel, reported in [Kre61, fn. 1]) have made the interesting dual observation that if the generative justification of $\mathbf{N}$ were to be taken as "predicative", then one should also accept as predicative the set $\boldsymbol{W}$ of all well-founded countably-branching trees, which is complete-Π^1_1 and not "predicative" in the sense of being hyperarithmetical.

Nelson point is, then, that the culprit in generating ontologically dubious terms is the impredicative justification of the set $\mathbf{N}$, and therefore the impredicativity of proof by Induction. Nelson observes that induction presupposes that $\mathbf{N}$ is given as a completed totality, and so using induction to justify that the values of certain terms are in $\mathbf{N}$ is an impredicative argument. He goes on to develop a system of Predicative Arithmetic, in which exponentiation is not provably correct. A problem with Nelson's development is that no clear cut rationale is given for admitting addition and multiplication, but not exponentiation, as primitives. Isles [1991] brings out the impredicative nature of the proof that the exponentiation function is well defined, but he too does not provide a foundational delination of feasibility.

1.4 Strictly Predicative Comprehension

Levels of impredicativity can be precisely calibrated by comprehension (set existence) principles, i.e. the admittance as legitimate of sets $\{x \in \mathbf{N} \mid \varphi\}$ for certain formulas φ. Much progress has been made in the last decade in calibrating the strength of formalisms for second-order arithmetic with weak forms of comprehension (notably by H. Friedman, Mints, Sieg, Simpson, and Smith). However, all formalisms considered are built on top of Primitive Recursive Arithmetic, so these studies are of no help in delineating the impredicativity involved in the primitive recursive (PR) functions, let alone in smaller classes.

A framework for calibrating the impredicativity of sub-PR functions was proposed in [Lei83, Lei90], with second-order *logic* used in place of second-order arithmetic. Contrary to weak systems for second-order arithmetic, the set of natural numbers is here *not* assumed as a completed totality. The method does not depend on any choice of basic numeric functions (such as addition and multiplication) or of axioms for them, and is therefore suitable for calibrating the logical nature of "small" functions. Moreover, it applies as easily to any term algebra as to $\mathbf{N}$.

Consider now the question of what instances of comprehension can be justified on strictly predicative grounds. Since the existence of infinite sets as completed totality cannot be so justified, we must stipulate that relational variables range over finite or potentially-infinite sets, i.e. sets that are "coming into being". Over a given structure we use comprehension to delineate new sets that are finite or potentially infinite, from the structure functions and relations, and from relational variables which denote already-defined finite or potentially infinite sets. Specifically,

if R is a relational variable, and $\vec{\mathbf{t}}$ are terms (where $arity(\vec{\mathbf{t}}) = arity(R)$), we admit $\{x \mid R(\vec{\mathbf{t}})\}$. We must also admit finite unions and intersections of admitted sets. However, we can *not* admit the complement of an admitted set S, since this is tantamount to accepting S as an actual infinity, for which non-members can be identified. Also, the use of quantifiers is suspect, because they refer to exhaustive inspection of the structure universe. [4] We are thus led to accept, on strictly predicative grounds, comprehension over exactly the positive quantifier-free formulas (i.e. without negation or implication).

The main result of this paper states that the computable functions justified on the basis of positive quantifier-free comprehension are precisely the functions computable in deterministic polynomial time. This shows that the class P-Time arises naturally from a foundational analysis of feasibility, and that terms using exponentiation can be justified as meaningful only under the admission of infinite sets as completed totalities. Specific terms, such as $3 \Uparrow 5$, have their own complete computation as direct justification, but since no such computation can ever be exhibited, such terms can be feasibly justified only via the general justification of exponentiation, i.e. via implicit reference to completed infinite sets.

2 Functional programs

2.1 Herbrand-Gödel programs

Our canonical computation model is functional programs, in the Herbrand-Gödel style (See [Kle52] or [Lei90] App. 1 for expositions). The original Herbrand-Gödel definition is for $\mathbf{N}$, the free term algebra generated from a constant 0 and a unary function $\mathbf{s}$. We use such programs over arbitrary free algebras, in particular the term algebra generated from a constant ϵ and unary functions 0 and 1, i.e. simply the set $W = \{0,1\}^*$ (e.g. the word 011 is identified with the term $011\epsilon = 0(1(1(\epsilon)))$). We can assume, without loss of generality, that functional programs are coherent, i.e. that they define a partial function, and not a multiple-valued function.[5]

For example, the following program (over W) computes the function $\odot$, which on input v, w returns $w^n = w \cdots w$ (n factors in concatenation) where $n = length(v)$. We use $\mathbf{c}$ to range over $\{0,1\}$.

$$\begin{array}{ll} \epsilon \oplus w = w & (\mathbf{c}v) \oplus w = \mathbf{c}(v \oplus w) \\ w \odot \epsilon = \epsilon & w \odot (\mathbf{c}v) = w \oplus (w \odot v) \end{array}$$

2.2 Convergence

To formally state the convergence of a functional program for some or for all input one needs to refer to potentially non-terminating computations. An approach common in Proof Theory, and due to Kleene [Kle52, Kle69], is to explicitly describe operational convergence, in a formalism sufficiently rich to code (Gödelize) the operational machinery. In logics of programs one expresses convergence using modal operators (as in Dynamic Logic, see e.g. [Pra80]) or using potentially non-denoting terms (see e.g. [Gol82]).

We continue here the alternative approach of [Lei83, Lei90], where programs are considered not as definitions of partial functions over the term-algebra A in hand, but as definitions of total functions over any structure whose vocabulary contains the generators of A. The key connection between such structures and convergence of programs over the intended term-algebra is given by the following observation [Lei83, Lei90]. Fix a term algebra A. For a functional program P (over A) let $[P]$ be the conjunction of the universal closures of the equations in P.

Theorem 2.1 *Let P be a functional program with principal function identifier $\mathbf{f}$. The following condistions are equivalent: (1) P converges (over A) for input $\vec{\mathbf{t}} \in A$; (2) for every model $\mathcal{S}$ of $[P]$, there is some $\mathbf{r} \in A$ such that $\mathcal{S} \models \mathbf{f}(\vec{\mathbf{t}}) = \mathbf{r}$; (3) there is some $\mathbf{r} \in A$ such that for every model $\mathcal{S}$ of $[P]$ $\mathcal{S} \models \mathbf{f}(\vec{\mathbf{t}}) = \mathbf{r}$.*

The entailment relation $\models$ refers here to all structures of the appropriate vocabulary.

2.3 Second-order statement of convergence

We consider a second-order extension of first-order logic with new variables ranging over relations, and quantification over such variables. Let A be a free term algebra. Writing A also for the predicate "is $\in A$", we have

$$A(x) \quad \equiv_{\mathrm{df}} \quad \forall Q \; Cl_A[Q] \rightarrow Q(x)$$

[4] We comment on this in the list of research directions below.

[5] Kleene [1952] showed this for numeric functions. A proof for the general case can be obtained either by generalizing Kleene's proof for a computation model with fixpoint, or by generalizing the simulation used in Lemma 3.2 below for Turing machine computbility. Details will be given elsewhere.

where $Cl_A[Q]$ is a formula stating that Q is closed under the generators of A. For instance,

$$Cl_W[Q] \equiv_{\text{df}} Q(\epsilon) \wedge \forall u\,(Q(u) \rightarrow (Q(0u) \wedge Q(1u)))$$

From Theorem 2.1 we then conclude:

Theorem 2.2 *Let P be a functional program with principal function identifier* $\mathbf{f}$. *P converges (over A) for all input iff*

$$[P] \models A(\vec{x}) \rightarrow A(\mathbf{f}(\vec{x}))$$

Here $arity(\vec{x}) = arity(\mathbf{f})$, $A(x_1 \ldots x_k)$ abbreviates $A(x_1) \wedge \cdots \wedge A(x_k)$, and the relational quantifiers have their standard interpretation.

2.4 Provable convergence

By Theorem 2.2 there is a natural, axiom-independent, way of formulating in formalisms for second-order logic the provable convergence of functions.

Let L be a formalism for second (or higher) order logic. We say that a function f over A is **provable in** L iff it is computed by some functional program P (with principal function identifier $\mathbf{f}$) such that

$$[P] \vdash_L A(\vec{x}) \rightarrow A(\mathbf{f}(\vec{x}))$$

Given a collection C of formulas, let $L_2(C)$ be a formalism for second-order logic with comprehension for formulas in C (for example, the natural deduction formalism of [Pra65]). The interpretation in [Pra65] of second-order arithmetic in second-order logic implies that the provable functions (over $\mathbf{N}$) of L_2(all second-order formulas) are precisely the provably-recursive functions of second-order arithmetic.[6] In particular, from $N(x)$ one gets induction with respect to x for all formulas.

To obtain from $N(x)$ induction for a first-order arithmetic formula φ we need comprehension for the interpretation φ' of φ, which in general is *not* first-order, because quantifiers in φ are interpreted in φ' as quantifiers relativized to N. In [Lei90b, Lei91] it is shown that the provably recursive functions of first-order arithmetic are precisely the provably recursive functions of L_2(strict-Π^1_1), and that the primitive-recursive functions are precisely the provably recursive functions of L_2(strict-Π^1_1without relational parameters). [7]

[6] A simple method for dealing with Peano's third and fourth axioms is given in [Lei90].

[7] A formula is strict-Π^1_1 if it is of the form $\forall \vec{Q} \exists \vec{x} \psi$, with ψ quantifier-free. In [Lei91] we gave an overview of the concept's significance.

2.5 S-provable convergence

We shall refer here to a notion of provable convergence formally weaker than the one defined above. Let $\mathcal{S}$ be a structure in the vocabulary $V_A = \{\mathbf{f}_0 \ldots \mathbf{f}_k\}$ of A, where $arity(\mathbf{f}_i) = r_i \geqslant 0$. We say that $\mathcal{S}$ is *surjective* if its universe $|\mathcal{S}|$ is covered by the range of the structure functions and constants, i.e. if

$$\mathcal{S} \models Surj_A$$

where

$$Surj_A \equiv_{\text{df}} \forall u \bigvee_{i=0\ldots k} \exists v_1 \ldots v_{r_i}\; u = \mathbf{f}_i(v_1 \ldots v_{r_i}).$$

For example

$$Surj_W \equiv \forall u\,(u = \epsilon \vee \exists v\,(u = 0v) \vee \exists v\,(u = 1v))$$

The surjective structures include not only the free algebra A itself, but also most natural examples of non-standard models for the theory of A. For example, the flat A domain is surjective because $\perp = \mathbf{f}(\perp, \ldots)$ for any non 0-ary $\mathbf{f} \in V_A$ (we assume that A is non-trivial).

Since every term algebra A is surjective, Theorem 2.2 holds trivially when validity is restricted to validity in surjective structures; i.e. P converges over A for all input iff

$$[P],\ Surj_A \models A(\vec{x}) \rightarrow A(\mathbf{f}(\vec{x})).$$

Given a formalism L as above, we say that a function f over A is **s-provable** in L iff it is computed by some functional program P (with principal function identifier $\mathbf{f}$) such that

$$[P],\ Surj_A \vdash_L A(\vec{x}) \rightarrow A(\mathbf{f}(\vec{x})).$$

Every function provable in L is trivially s-provable in L. The next theorem states that the converse holds when L has enough comprehension. Let $\alpha \equiv \alpha[x]$ be a formula with some single free variable x. If φ is a second-order formula, its *relativization to* α, φ^α, is obtained by restricting first-order quantification to elements satisfying α, and restricting second-order quantification to subsets of the the extension of α. I.e., φ^α is defined by recurrence on φ as follows, where, for k-ary Q, $Q \subseteq \alpha$ abbreviates $\forall v_1 \ldots v_k\, Q(\vec{v}) \rightarrow \alpha[v_1] \wedge \cdots \wedge \alpha[v_k]$.

$$\begin{array}{rcl} \varphi^\alpha & \equiv_{\text{df}} & (\varphi \text{ quantifier free}) \\ (\neg\varphi)^\alpha & \equiv_{\text{df}} & \neg(\varphi^\alpha) \\ (\varphi \star \psi)^\alpha & \equiv_{\text{df}} & \varphi^\alpha \star \psi^\alpha \quad (\star \text{ a binary connective}) \end{array}$$

$$
\begin{array}{lll}
(\forall v\,\varphi)^\alpha & \equiv_{\mathrm{df}} & \forall v\,(\,\alpha[v] \to \varphi^\alpha\,) \\
(\exists v\,\varphi)^\alpha & \equiv_{\mathrm{df}} & \exists v\,(\,\alpha[v] \wedge \varphi^\alpha\,) \\
(\forall Q\,\varphi)^\alpha & \equiv_{\mathrm{df}} & \forall Q\,(\,(Q \subseteq \alpha) \to \varphi^\alpha\,) \\
(\exists Q\,\varphi)^\alpha & \equiv_{\mathrm{df}} & \exists Q\,(\,(Q \subseteq \alpha) \wedge \varphi^\alpha\,)
\end{array}
$$

Theorem 2.3 *Suppose L has comprehension over a class of formulas Φ which contains the strict-Π^1_1 formulas and is closed under relativization to strict-Π^1_1 formulas.*[8] *Then every function s-provable in L is provable in L.*

Proof. By a straightforward induction on proofs one proves that if L is a logic as above, and $\vdash_L \varphi$, then $\vdash_L \alpha[\varphi] \to \varphi^\alpha$, where $\alpha[\varphi]$ is the formula stating that every free individual variable of φ satisfies α, and every free relational variable of φ lies within α (in the obvious sense). Suppose

$$[P],\ \forall u\, R_A[u] \vdash_L\ A[\vec{x}] \to A[\mathbf{f}(\vec{x})].$$

Then the previous observation implies

$$[P]^A,\ \forall u^A\, R_A[u]^A \vdash_L\ A[\vec{x}]^A \to A[\mathbf{f}(\vec{x})]^A.$$

Using strict-Π^1_1 comprehension it is fairly easy to prove $\forall u^A\, R_A[u]^A$ and $\forall x\, A[x] \leftrightarrow A^A[x]$. Since $[P] \to [P]^A$ trivially, we obtain

$$[P],\ \forall u\, R_A[u] \vdash_L\ A[\vec{x}] \to A[\mathbf{f}(\vec{x})].$$

□

A formula is **positive** if it contains no negation or implication. Let QF^+ be the collection of positive quantifier-free formulas. The main result of this work is:

Theorem 2.4 *A function over W is computable in deterministic polynomial time iff it is s-provable in $L_2(QF^+)$.*

3 Every P-Time function is s-provable

3.1 Provable correctness of the multiplicative string function

Lemma 3.1 *The function $\odot$ is provable in $L_2(QF^+)$.*

[8] For example, second-order logic with strict-Π^1_1 comprehension (with relational parameters) is such a logic.

Proof. Let $P_\odot$ be the program above for $\odot$. We derive in $L_2(QF^+)$ the formula

$$[P_\odot] \wedge W(x,y) \to W(x \odot y).$$

Assume $[P_\odot]$, $W(x,y)$, and $Cl_W[Q]$. We need to derive $Q(x \odot y)$. First, we show

$$Q(x \odot z) \to Q(x \oplus (x \odot z)). \tag{1}$$

Indeed, from the equations for $\odot$, using $Cl_W[Q]$, we have $\forall u\ Q(u \oplus (x \odot z)) \to Q(0u \oplus (x \odot z)) \wedge Q(1u \oplus (x \odot z))$, and from $Q(x \odot z)$ we get $Q(\epsilon \oplus (x \odot z))$. By $W(x)$, with comprehension for the atomic formula $\lambda u\, Q(u \oplus (x \odot z))$, the latter two formulas imply $Q(x \oplus (x \odot z))$.

By the equations for $\odot$, (1) implies

$$Q(x \odot z) \to Q(x \odot 0z) \wedge Q(x \odot 1z). \tag{2}$$

Since $x \odot \epsilon = \epsilon$, and $Q(\epsilon)$ is given (since $Cl_W[Q]$ is assumed), we also have

$$Q(x \odot \epsilon). \tag{3}$$

By $W(y)$, with comprehension for the atomic formula $\lambda u\, Q(x \odot u)$, (2) and (3) imply $Q(x \odot y)$, concluding the proof. □

3.2 Simulation of P-Time Turing machines

It is well-known that virtually all usual models of computation are polynomial-time reducible to each other. We exhibit a reduction from Turing machines to functional programs that yields provably converging programs from deterministic TM's.

We use a variant of TM's where transitions are represented by strings of the following forms:

- $sq_0q_1q_B$ (in state s, if scanning 0, 1, or B, then move to state q_0, q_1, or q_B, respectively)
- scs' (in state s, replace the scanned character by $\mathbf{c}$ and move to state s')
- sRs' (in state s, step head to the right and move to state s')
- sLs' (in state s, step head to the left, if possible, and move to state s')

We also assume, without loss of generality, that M has exactly one transition for every configuration (including accepting and abortive configuration, which can

be made to cycle), and that it never reaches a configuration with B to the left of the scanning head. We let the states of M be represented by strings in W.

Given a deterministic TM M, let $r(w,t) \equiv r_M(w,t) =_{\mathrm{df}}$ the string from the scanned character of M and on to the right, after $|t|$ steps of M's run on input w. We define a functional program $P(M)$ for r_M, as follows. $P(M)$ computes r simultaneously with functions ℓ and h, where $\ell(w,t)$ is the string to the left of M's head, and $h(w,t)$ is the (binary-coded) state of M, both after $|t|$ steps of M's run on input w.

The equations in $P(M)$ are:

- $r(w,0) = w$, $\ell(w,0) = \epsilon$, $h(w,0) =$ the initial state of M;
- $r(w,0t) = r(w,1t) = r'(h(w,t),\ell(w,t),r(w,t))$,
  $\ell(w,0t) = \ell(w,1t) = \ell'(h(w,t),\ell(w,t),r(w,t))$,
  $h(w,0t) = h(w,1t) = h'(h(w,t),\ell(w,t),r(w,t))$;
- for each transition $sq_0q_1q_B$ of M the equations
  $$r'(s,u,v) = v, \qquad \ell'(s,u,v) = u,$$
  $$h'(s,u,0v) = q_0, \qquad h'(s,u,1v) = q_1,$$
  $$h'(s,u,\epsilon) = q_B;$$
- for each transition scs' the equations
  $$r'(s,u,0v) = r'(s,u,1v) = cv, \quad r'(s,u,\epsilon) = c\epsilon,$$
  $$\ell'(s,u,v) = u, \qquad h'(s,u,v) = s';$$
- for each transition sRs' the equations
  $$r'(s,u,cv) = v, \qquad \ell'(s,u,cv) = cu,$$
  $$r'(s,u,\epsilon) = \epsilon, \qquad \ell'(s,u,\epsilon) = \epsilon;$$
  $$h'(s,u,v) = s',$$
- for each transition sLs' the equations
  $$r'(s,cu,v) = cv, \qquad \ell'(s,cu,v) = u,$$
  $$r'(s,\epsilon,v) = v, \qquad \ell'(s,\epsilon,v) = \epsilon.$$
  $$h'(s,u,v) = s',$$

Lemma 3.2 *For every* deterministic *TM M as above, $P(M)$ is a coherent program, and M has a computation of length n with input w and output w' iff $P(M)$ has a computation whose last equation is $r(w,0^{[n]}\epsilon) = w'$.*

3.3 Provability of P-Time functions

Lemma 3.3 *Let M and $P(M)$ be as above. Then*

$$[P(M)],\ Surj_W \vdash_{L_2(QF^+)}$$
$$\forall t\ W(t) \wedge Cl_W[Q] \wedge Q(w)$$
$$\rightarrow Q(r_M(w,t)) \wedge Q(\ell(w,t)) \wedge \bigvee_{s \text{ a state of } M} h(w,t) = s$$

Proof sketch. The assumption $W(t)$ permits to deduce the conclusion of the implication above by induction (with respect to u) for the positive quantifier-free formula

$$Q(r_M(w,u)) \wedge Q(\ell(w,u)) \wedge \bigvee_{s \text{ a state of } M} h(w,u) = s.$$

The basis and induction steps are straightforward, using $[P(M)]$, $Q(w)$ and $Surj_W$. □

Theorem 3.4 *Every function over W computable in deterministic polynomial time is s-provable in $L_2(QF^+)$.*

Proof. Let M be a deterministic TM computing a function f in time bounded by the k'th power of the input's size. Let r_M be defined as above. From Lemma 3.3 it follows that r_M is s-provable in $L_2(QF^+)$. By Lemma 3.1 the function $\exp_k(w) =_{\mathrm{df}} w \odot \cdots \odot w$ (k factors) is provable in $L_2(QF^+)$. Since $f(w) = r_M(w, \exp_k(w))$ is a composition of a provable function and a s-provable function, it is s-provable. □

4 S-provable functions are P-Time

4.1 The structure of normal convergence proofs

We use an analysis of normal derivations in the natural deduction formalism for L_2, as defined e.g. in [Pra65]. Most methods of extracting an algorithm from normal proofs follow the following pattern: suppose Π is a normal proof of a formula φ that states the convergence of a function f for all input. Given a closed canonical term $\mathbf{t}$ (i.e. a numeral for the case of f over $\mathbb{N}$), Π can be easily specialized to a proof $\Pi_{\mathbf{t}}$ that f converges on input $\mathbf{t}$. [9] By normalizing $\Pi_{\mathbf{t}}$ one obtains a proof from which the value of $f(\mathbf{t})$ can be easily found. Thus the function f is reducible, by an easy computation, to the normalization procedure for the formalism in hand. [10]

This method cannot be used here, because the normalization is not in P-Time, though each branch of the normal proof can be computed independently in P-Time. The procedure described below uses subderivations of the derivation Π above directly, to obtain a value for $f(\mathbf{t})$. The method is equivalent to computing the branch of the normal form of $\Pi_{\mathbf{t}}$ from which that value would be extracted.

[9] "Easy" means, in all such applications, "on line".

[10] This method, for our formalisms $L_2(C)$, is used in [Lei90,Lei91].

Let P be a functional program, with principal function letter $\mathbf{f}$, which we assume unary (without loss of generality). Suppose Π is a normal $L_2(QF^+)$-derivation of $W(\mathbf{f}(x))$ from $W(x)$, $Surj_W$ and $[P]$. We assume, w.l.o.g., that Π is without redundant parameters (see [Pra65]). By normality, Π has a subderivation Π', deriving $Q(\mathbf{f}(x))$ from $[P]$, $Surj_W$, $Cl_W[Q]$, and formulas $Cl_W[\chi_i] \rightarrow \chi_i[x]$ for some $\chi_1 \ldots \chi_k \in QF^+$. The normality of Π' implies that each assumption $Cl_W[\chi_i] \rightarrow \chi_i[x]$ is the premise of an elimination, and so Π' factors into a normal derivation Π_0, deriving $Q(\mathbf{f}(x))$ from $[P]$, $Surj_W$, $Cl_W[Q]$, and $\chi_1[x] \ldots \chi_k[x]$, and normal derivations $\Sigma_1 \ldots \Sigma_k$, where Σ_i derives $Cl[\chi_i]$ from $[P]$, $Surj_W$, $Cl_W[Q]$, and some $\chi_{j_{i,1}} \cdots \chi_{j_{i,n_i}}$, $j_{i,l} \neq i$.

4.2 Linear-time algorithm for a simple case

To fix ideas, let us consider the simplest non-trivial example, i.e. with $k = 1$. Then Σ_1 has the following normal subderivations:

- a derivation $\Sigma_1^{\epsilon}[x]$ of $\chi_1[\epsilon/y, x]$ from $[P]$, $Surj_W$, and $Cl_W[Q]$
- a derivation $\Sigma_1^0[u, x]$ of $\chi_1[0u/y]$ from $\chi_1[u/y]$, $[P]$, $Surj_W$, and $Cl_W[Q]$
- a derivation $\Sigma_1^1[u, x]$ of $\chi_1[1u/y]$ from $\chi_1[u/y]$, $[P]$, $Surj_W$, and $Cl_W[Q]$

We now sketch a linear-time algorithm for the function f computed by program P. Given input $w \in W$, say $w = 001\epsilon$, we consider the root $Q(f(w))$ of $\Pi_0[w/x]$, and decree $f(w)$ as the first "labeled term." We go up in the derivation $\Pi_0[w/x]$, branching from a formula φ with labeled term $\mathbf{t}$ to a premise ψ that is *not* an equation, and that contains a term $\mathbf{s}$ whose value would determine the value of $\mathbf{t}$, with the term $\mathbf{s}$ then becoming our new labeled term. It is not difficult to see that this is always possible, and that the values of successive labeled terms have lengths that differ by at most 1. (The complete argument uses a tedious enumeration of cases.) This search either terminates with the subformula $Q(\epsilon)$ of $Cl_W[Q]$, or with $\chi_1[w/x]$. In the first case we have obtained a value for $f(w)$ (by reconstructing the values of successive labeled terms). In the second case, given that $w = 001\epsilon$, we repeat the search process for the derivation $\Sigma_1^0[01\epsilon/u, w/x]$, starting with the derived formula $\chi_1[001\epsilon/y]$. Again, we may either terminate with $Q(\epsilon)$, or with $\chi_1[01\epsilon/y]$. Repeating again a search through $\Sigma_1^0[1\epsilon/x]$ we either terminate or hit $\chi_1[1\epsilon]$. Next we perform a search through $\Sigma_1^1[\epsilon/u]$, and finally a search through Σ_1^{ϵ}. In all, we have performed $1 + |w| + 1$ searches, each of a fixed length independent of w.

4.3 Polynomial-time algorithms for s-provable functions

For a slightly more general example, consider the case with $k = 3$, where Σ_1 has χ_2 and χ_3 as assumptions, and Σ_2, Σ_3 have χ_1 as assumption. The process above can then be amended to yield a quadratic-time algorithm for f, as follows. A search through $\Sigma_1^0[01\epsilon/u, 001\epsilon/x]$ might terminate with $\chi_2[001\epsilon/v, 01\epsilon/u]$. After three successive passes through $\Sigma_2^0[01\epsilon/v, 01\epsilon/u]$, $\Sigma_2^0[1\epsilon/v, 01\epsilon/u]$, and $\Sigma_2^{\epsilon}[01\epsilon/u]$ we might reach the assumption $\chi_1[01\epsilon/u]$ (i.e. considering χ_1 again!). We then have to restart a search through $\Sigma_1^0[1\epsilon/u]$.

In general, if we have nesting up to depth p of the k derivations $\Sigma_1 \ldots \Sigma_k$ (so $p \leqslant k$), the procedure described above will terminate within time

$$(|w|+2)^p \times \langle \text{ maximal height of } \Pi_0, \Sigma_1 \ldots \Sigma_k \rangle \times \langle \text{ the constant time needed for search-steps } \rangle$$

(Details in the full paper.) We conclude:

Theorem 4.1 *Every s-provable function (over W) of $L_2(QF^+)$ is computable in deterministic polynomial time.*

Hence, by Theorem 3.4, a function over W is in P-Time iff it is s-provable in $L_2(QF^+)$.

This concludes the proof outline for our main Theorem 2.4.

5 Directions for future research

5.1 Refinements and delineations of the main result

1. The exclusion of quantifiers from strictly-predicative comprehension is not quite convincing, since in the abstract setting considered here nothing is assumed about the universe of discourse, so that the meaning of (non-relativized) quantifiers may be viewed as being in fact independent of the scope of the universe. [11] This dilemma, of whether or not to accept as

[11] The formula-as-type homomorphism of [Lei83,Lei90] also shows that non-relativized quantifiers have no computational content.

strictly predicative comprehension over positive *first-order* comprehension, would have a most satisfactory resolution if we could show that it does not matter, i.e. that the s-provable functions of the resulting formalism are all in P. In fact, it seems that a refinement of the search algorithm above would yield this result.

2. Are there P-Time functions that are not provable (as opposed to s-provable) in $L_2(QF^+)$? We conjecture a positive answer. The numeric functions provable in L(atomic) are exactly the extended polynomials[12]: under the Curry-Howard isomorphism, amended as in [Lei83,Lei90], a natual deduction proof of L(atomic) of a function's totality is mapped to a representation of that function in the simply typed λ-calculus, with input and output of type 0. These functions are the extended polynomials [Schw76,Sta79]. Conversely, the extended polynomials are all provable in L(atomic) [Lei83]. We in fact conjecture that the numeric functions provable in L(atomic) are again just the extended polynomials.

3. Provable convergence can be conveyed in L_2 in other forms, and referring to other models of computation (e.g. PROLOG programs). We conjecture that the result of this paper is robust with respect to most such variations.

4. Establish direct connections between the approach presented here and proof theoretic characterizations of complexity classes that are based on weak forms of induction, initiated by Sam Buss [Bus86].

5.2 Characterizations of other complexity classes by set-existence

1. The methods of this paper seem to establish that the s-provable functions of L(first-order) are precisely the Kalmar-elementary ones. We conjecture, however, that the subtraction function is not provable in L(first-order), and that consequently provablity is still weaker than s-provability for first-order comprehension.[13]

2. What are the (s-) provable functions of with positive strict-Π^1_1 comprehension? with positive Π^1_1 comprehension? We conjecture that both are exactly the PR functions (using a detailed inspection of the proof in [Lei91]).

3. What are the provable functions of stratified second-order logic with positive comprehension?

4. What are the (s-) provable functions of full type theory with positive comprehension?

5.3 Feasible mathematics

1. The main result here suggests that $L(QF^+)$ is a natural formalism in which to develope Feasible Mathematics.

2. The methods of this paper can be lifted to higher types, with potential applications to feasibility of functionals of higher type (compare [CK90,Tow90]). In particular, what is the class of type-2 functionals over W, computed by a systen P of recurrence equations (with principal identifier $\mathbf{F}$), for which

$$[P],\ Surj_W \vdash_{L(QF^+)} \forall f\ (\forall x(W(x) \to W(f(x)))\ ) \to W(\mathbf{F}(f)),$$

and how does this class compare with other definitions of feasibility for type-2 functionals?

[12] These are the functions defined by composition from addition, multiplication, and defined by cases

[13] In addition to Theorem 2.3, we also know that the difference between provability and s-provability disappears for finitely stratified second-order logic, by the methods of [Lei91a].

References

Ber35 *Sur le Platonisme dans les Mathématiques,* **L'Enseignement Mathématique 34** (1935). English translation: *On platonism in Mathematics,* in P. Benacerraf and H. Putnam (eds.), **Philosophy of Mathematics** (second edition), Cambridge University Press, 1983, 258–271.

Bus86 Samuel Buss, **Bounded Arithmetic**, Bibliopolis, Naples, 1986.

CK90 Stephen Cook and Bruce Kapron, *Characterizations of the basic feasible functionals of finite type,* in Samuel Buss & Philip Scott (eds.), **Feasible Mathematics**, *Perspectives in Computer Science*, Birkhauser-Boston, New York (1990) 71–95.

CKS81 Ashok Chandra, Dexter Kozen and Larry Stockmeyer, *Alternation*, **Journal of the ACM 28** (1981), 114–133.

Cob65 A. Cobham, *The intrinsic computational difficulty of functions,* in Y. Bar-Hillel (ed.), **Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science**, North-Holland, Amsterdam (1962) 24–30.

Dan56 D. van Danzig, *Is 10^(10^10) a finite number?,* **Dialectica 19** (1956) 273–277.

Gan89 Robin Gandy, *Totally finite interpretations of first-order arithmetic* (second draft), Manuscript, June 1989.

Gol82 Robert Goldblatt, **Axiomatising the Logic of Computer Programming**, Springer-Verlag (LNCS # 130), Berlin (1982).

GS86 Yuri Gurevich and Saharon Shelah, *Fixed-point extensions of first-order logic,* **Annals of Pure and Applied Logic 32** (1986) 265–280.

GSS89 Jean-Yves Girard, Andre Scedrov and Philip Scott, *Bounded Linear Logic: A modular approach to polynomial time computability* (Extended Abstract), in Samuel Buss & Philip Scott (eds.), **Feasible Mathematics**, *Perspectives in Computer Science*, Birkhauser-Boston, New York (1990) 195–207.

Gur83 Yuri Gurevich, *Algebras of feasible functions,* **Twenty Fourth Symposium on Foundations of Computer Science**, IEEE Computer Society Press, 1983, 210–214.

Gur87 Yuri Gurevich, *Logic and the challenge of Computer Science,* in **Current Trends in Theoretical Computer Science,** (Egon Börger, editor), Computer Science Press, 1987.

Isl91 David Isles, *What evidence is there that 2^65536 is a natural number?,* submitted for publication, January 1991.

Imm86 Neil Immerman, *Relational queries computable in polynomial time,* **Information and Control 68** (1986) 86–104. Preliminary report in **Fourteenth ACM Symposium on Theory of Computing** (1982) 147–152.

Imm87 Neil Immerman, *Languages which capture complexity classes,* **SIAM Journal of Computing 16** (1987) 760–778.

Kle52 S.C. Kleene, **Introduction to Metamathematics**, Wolters-Noordhof, Groningen, 1952.

Kle69 S.C. Kleene, *Formalized Recursive Functions and Formalized Realizability,* **Memoirs of the AMS 89**, American Mathematical Society, Providence, 1969.

Kre61 Georg Kreisel, *Set theoretic problems suggested by the notion of potential totality,* in **Infinitistic Methods** (Proceedings of the Symposium on Foundations of Mathematics, Warsaw, 1959), Pergamon Press, New York, pp. 103–140.

Kre62 Georg Kreisel, *Foundations of intuitionistic logic,* in Nagel, Suppes & Tarski (eds.), **Logic, Methodology, and the Philosophy of Science**, Stanford University Press, Stanford (1962) 198–210.

Lei83 Daniel Leivant, *Reasoning about functional programs and complexity classes associated with type disciplines,* **Twenty-fourth Annual Symposium on Foundations of Computer Science** (1983) 460-469.

Lei90 Daniel Leivant, *Contracting proofs to programs,* in P. Odifreddi (editor) **Logic and Computer Science**, Academic Press, London, 1990, 279–327.

Lei90a Daniel Leivant, *Inductive definitions over finite structures,* **Information and Computation 89** (1990) 95–108.

Lei90b Daniel Leivant, *Computationally based set existence principles*, in W. Sieg (ed.), **Logic and Computation,** Contemporary Mathematics, volume 106, American Mathematical Society, Providence, R.I., 1990, pp. 197–211.

Lei91 Daniel Leivant, *Semantic characterization of number theories,* in Yiannis Moschovakis (ed.), **Logic from Computer Science**, Springer-Verlag, Berlin, 1991 (expected).

Lei91a Daniel Leivant, *Finitely Stratified polymorphism,* to appear in **Information and Computation**, 1991.

Nel86 Edward Nelson, **Predicative Arithmetic,** Princeton University Press, Princeton, 1986.

Pap85 Christos Papadimitriou, *A note on the expressive power of PROLOG,* **Bull. EATCS 26** (June 1985) 21–23.

Pra65 Dag Prawitz, **Natural Deduction**, Almqvist and Wiskel, Uppsala, 1965.

Saz80 Vladimir Sazonov, *Polynomial computability and recursivity in finite domains,* **Electronische Informationsverarbeitung und Kybernetik 7** (1980) 319–323.

Schw76 Helmut Schwichtenberg, *Definierbare Funktionen im Lambda-Kalkul mit Typen,* **Archiv Logik Grundlagenforsch. 17** (1976) 113–114.

Sta79 Richard Statman, *The typed λ-calculus is not elementary recursive,* **Theoretical Computer Science 9** (1979) 73–81.

Tow90 Michael Towsend, *Complexity of type 2 relations,* **Notre Dame Journal of Formal logic 31** (1990) 241–262.

Var82 Moshe Vardi, *Complexity and relational query languages,* **Fourteenth ACM Symposium on Theory of Computing** (1982) 137–146.

Yes70 A.S. Yessenin-Volpin, *The ultra-intuitionistic criticism and the anti-traditional program for foundations of mathematics,* in Kino, Myhill &Vesley (eds.), **Intuitionism and Proof Theory**, North-Holland, Amsterdam (1970) 3–46.