RAMIFIED RECURSION AND INTUITIONISM

Edward Nelson
Department of Mathematics
Princeton University

Both Jean-Louis Callot and Georges Reeb had a strong interest in the relationship between computers and nonstandard analysis, and Reeb was a dedicated intuitionist. These are the themes of this talk given to honor their memory. First I shall give a brief introduction to recent work on ramified recursion, expressed in language familiar to nonstandard analysts, and then I shall sketch an approach to a constructive logic of the feasible.

I. Ramified recursion

The 1930s witnessed a flowering of mathematical logic. One of the achievements of this period was the creation by Turing, Gödel, and Church of the theory of recursive functions, work that laid the theoretical foundations for the study of computers. With the advent of high-speed computers, attention focused on *feasible* computations as opposed to computations possible in principle.

The usual model for a feasible computation is that of *polynomial time functions*, functions that can be evaluated in time that is bounded by a polynomial in the lengths of their arguments. This notion is robust under changes in the model of computation (Turing machine, register machine, etc.). If a class of functions is characterized by time of evaluation, if it is closed under composition (and this is necessary to have a sensible theory), and if it contains one function requiring time $|x|^{1+\varepsilon}$ (where $|x|$ is the length of the argument x and $\varepsilon > 0$), then it contains all polynomial time functions. On the other hand, a function that is not polynomial time cannot be regarded as feasible for all arguments. But is must be emphasized that the notion is an abstract one, not to be confounded with the pragmatic notion of a function that is computable in practice.

Let me observe that I am using the familiar word "function" when "construction of a function" would be more precise. We are dealing not

with Platonic functions, two of which are the same if they have the same values (whatever that may mean), but with *programs*.

It is well known that there is no syntactical description of all (total) recursive functions. If there were a list of them all, then Cantor's diagonal argument would yield another. Gödel cut this Gordian knot by giving a Platonic definition of recursive function. A recursive function is a program that terminates for all inputs, and Turing showed that the halting problem (to decide whether a program terminates for all inputs) is undecidable.

But Cantor's diagonal argument does not apply to the class of polynomial time functions and this raises the question as to whether a syntactical characterization of the class is possible. The answer is yes, as was proved in a fundamental paper [1] by Stephen Bellantoni and Stephen Cook. A complete account, together with related developments, is in Bellantoni's thesis [2]. Here I shall follow the beautiful treatment by Daniel Leivant [3], with some changes in terminology derived from nonstandard analysis.

Let $\mathcal{W}$ be the *free word algebra* with the constant ϵ (denoting the empty string) and the unary functions 0 and 1. Thus $\mathcal{W}$ consists of all strings of zeroes and ones. We call certain strings *standard*, with the assumptions that ϵ is standard and if x is standard, so are $0x$ and $1x$.

Consider the smallest class of functions mapping $\mathcal{W}^n$, for some n depending on the function, to $\mathcal{W}$, containing ϵ, 0, and 1, closed under composition, and closed under *string recursion*; i.e., such that if g_ϵ, g_0, and g_1 are in the class then so is f defined by

$$\begin{aligned} f(\epsilon, \vec{x}) &= g_\epsilon(\vec{x}), \\ f(0y, \vec{x}) &= g_0\big(y, \vec{x}, f(y, \vec{x})\big), \\ f(1y, \vec{x}) &= g_1\big(y, \vec{x}, f(y, \vec{x})\big). \end{aligned}$$

This is the class of *primitive recursive functions* on $\mathcal{W}$, and it contains many infeasible functions.

Let us make a slight change. Consider functions f some of whose arguments are required to be standard; we write them before a semicolon, so that in $f(\vec{z}; \vec{w})$ the $\vec{z}$ are standard and the $\vec{w}$ are arbitrary. Such functions are called *sorted*. Then we modify recursion by requiring that f be defined only for standard y, but with the recursion arguments of g_0 and g_1 unrestricted. That is, *ramified recursion* is constructed as follows:

$$\begin{aligned} f(\epsilon, \vec{z}; \vec{w}) &= g_\epsilon(\vec{z}; \vec{w}), \\ f(0y, \vec{z}; \vec{w}) &= g_0\big(y, \vec{z}; \vec{w}, f(y, \vec{z}; \vec{w})\big), \\ f(1y, \vec{z}; \vec{w}) &= g_1\big(y, \vec{z}; \vec{w}, f(y, \vec{z}; \vec{w})\big). \end{aligned}$$

Let $\mathcal{P}$ be the smallest class of functions containing ϵ, 0, and 1 and closed under composition and ramified recursion; then the main theorem is that the functions in $\mathcal{P}$ all of whose arguments are standard is precisely the class of polynomial time functions.

This is a beautiful and astonishing result. The proof is ingenious and not very long. Notice that there is no arithmetic in the hypotheses of the theorem; the polynomial bounds are a consequence of ramified recursion. There are many related results and I urge you to read the references.

II. Intuitionism

The results of Bellantoni, Cook, and Leivant concern functions. It seems worthwhile to construct a full-fledged logic of the feasible, and I shall sketch how this might be done. Such a logic must be constructive, so I begin with a brief review of intuitionism.

Stephen Cole Kleene was the mathematician who saw most deeply into the nature of intuitionism, with his notion of *recursive realizabilility*; see [4] and [5]. I shall discuss realizability for formulas of arithmetic; the modifications necessary to discuss the free word algebra $\mathcal{W}$ are minor; for example, induction for $\mathcal{W}$ should be formulated as

$$\mathrm{A_x}(\epsilon) \quad \& \quad \forall \mathrm{x}[\mathrm{A} \quad \rightarrow \quad \mathrm{A_x}(0\mathrm{x}) \quad \& \quad \mathrm{A_x}(1\mathrm{x})] \quad \rightarrow \quad \mathrm{A}.$$

The function symbols are 0, S (*successor*), $+$, and $\cdot$. The logical operators are

$$\neg \ \& \ \rightarrow \ \forall \ \vee \ \exists$$

We can eliminate $\neg$ by regarding $\neg$A as an abbreviation for $\mathrm{A} \rightarrow \mathrm{S}0 = 0$. The intuitionistic semantics is epistemological, rather than ontological like the classical semantics, and to an intuitionist a formula is an incomplete communication of knowledge.

The notion of realizability is most simply expressed by introducing some *additional function symbols*. Listed with variables to show the placement of arguments, these are

$$\langle x, y\rangle \quad \pi_1 x \quad \pi_2 x \quad \gamma(x, y, z) \quad x\{y\} \quad \rho(x, y) \quad \Lambda xy.$$

A *code* is a term formed with these function symbols and the function symbols of arithmetic such that the first argument of each occurrence of Λ is a variable. I have called Λ a function symbol, but in many respects it is similar to a quantifier symbol. An occurrence of the variable x in the code c is Λ-*free* in case it is not in a part of c of the form

Λxb. We use the abbreviation $\mathrm{c}_{\mathrm{x}}(\mathrm{b})$ for the code obtained by substituting b for all Λ-free occurrences of x in c, with the tacit understanding that b is Λ-*substitutable* for x in c (meaning that there is no Λ-free occurrence of x in any part Λyd of c where y occurs in b), and similarly for $\mathrm{c}_{\mathrm{x}_1 \dots \mathrm{x}_\nu}(\mathrm{b}_1 \dots \mathrm{b}_\nu)$.

We use n to stand for a variable-free term of arithmetic. A code is *reduced* by making the following replacements as long as possible, say in the order listed:

	replace:	by:
(R1)	$\pi_1\langle \mathrm{a}, \mathrm{b}\rangle$	a
(R2)	$\pi_2\langle \mathrm{a}, \mathrm{b}\rangle$	b
(R3)	$\gamma(\mathrm{a}, \mathrm{b}, \langle 1, \mathrm{c}\rangle)$	$\mathrm{a}\{\mathrm{c}\}$
(R4)	$\gamma(\mathrm{a}, \mathrm{b}, \langle 2, \mathrm{c}\rangle)$	$\mathrm{b}\{\mathrm{c}\}$
(R5)	$\rho(\mathrm{n} + 0, \mathrm{c})$	$\rho(\mathrm{n}, \mathrm{c})$
(R6)	$\rho(\mathrm{n} + \mathrm{Sm}, \mathrm{c})$	$\rho(\mathrm{S}(\mathrm{n} + \mathrm{m}), \mathrm{c})$
(R7)	$\rho(\mathrm{n} \cdot 0, \mathrm{c})$	$\rho(0, \mathrm{c})$
(R8)	$\rho(\mathrm{n} \cdot \mathrm{Sm}, \mathrm{c})$	$\rho(\mathrm{n} \cdot \mathrm{m} + \mathrm{n}, \mathrm{c})$
(R9)	$\rho(0, \mathrm{c})$	$\pi_1\mathrm{c}$
(R10)	$\rho(\mathrm{Sn}, \mathrm{c})$	$\big((\pi_2\mathrm{c})\{\mathrm{n}\}\big)\{\rho(\mathrm{n}, \mathrm{c})\}$
(R11)	$(\Lambda \mathrm{xa})\{\mathrm{b}\}$	$\mathrm{a}_{\mathrm{x}}(\mathrm{b})$

Notice that with incorrect code, such as

$$(\Lambda \mathrm{xx}\{\mathrm{x}\})\{\Lambda \mathrm{xx}\{\mathrm{x}\}\},$$

this process may never terminate. We use

$$\mathrm{c} \Rightarrow \mathrm{c}'$$

to indicate that the code c reduces to c′.

The reduction rules (R1)–(R11) express the intended meaning of the additional function symbols. The ordered pair $\langle \mathrm{a}, \mathrm{b}\rangle$ has first and second projections π_1 and π_2. The value of the function a on the argument b is $\mathrm{a}\{\mathrm{b}\}$. These are not necessarily numerical-valued functions, but code-valued functions where the code may itself be a function. Since the reduction rules are given explicitly, these are recursive functions, but since the process may not terminate, they are partial functions. We use γ to choose which of two functions a and b to apply to the argument c. We use ρ for recursion: the rules (R5)–(R8) convert n to a numeral, (R9) gives the beginning of the recursion, and (R10) gives the recursion step. By (R11), Λxa is the partial function taking b to what (if anything) $\mathrm{a}_{\mathrm{x}}(\mathrm{b})$ reduces to.

Now we can define "c realizes C", where c is a Λ-closed code and C is a closed formula:

(C1) if C is atomic, c realizes C in case C is true;

(C2) if C is A & B, c realizes C in case π_1c realizes A and π_2c realizes B;

(C3) if C is A → B, c realizes C in case for all a, if a realizes A, then c{a} realizes B;

(C4) if C is ∀xA, c realizes C in case for all n, c{n} realizes $\mathrm{A_x(n)}$;

(C5) if C is A ∨ B, c realizes C in case c ⇒ ⟨1, a⟩ (for some a) and a realizes A, or c ⇒ ⟨2, b⟩ (for some b) and b realizes B;

(C6) if C is ∃yA, c realizes C in case c ⇒ ⟨n, a⟩ (for some n and a) and a realizes $\mathrm{A_y(n)}$.

For a formula D whose free variables are $\mathrm{x_1}, \ldots, \mathrm{x}_\nu$, d realizes D in case $\Lambda \mathrm{x_1} \ldots \Lambda \mathrm{x}_\nu \mathrm{d}$ realizes the closure of D.

Kleene formulated recursive realizability so as to ensure that the theorems of intuitionistic arithmetic would all be realizable. David Nelson undertook to establish that this is the case, and did so in [6]. Here are the realization codes for the axioms (nonlogical and logical) of intuitionistic arithmetic. (Note that → is associated from right to left.)

1.	$\neg\ \mathrm{S}x = 0$	0
2.	$\mathrm{S}x = \mathrm{S}y \to x = y$	0
3.	$x + 0 = x$	0
4.	$x + \mathrm{S}y = \mathrm{S}(x + y)$	0
5.	$x \cdot 0 = 0$	0
6.	$x \cdot \mathrm{S}y = x \cdot y + x$	0
7.	$\mathrm{A_x}(0)\ \&\ \forall \mathrm{x}[\mathrm{A} \to \mathrm{A_x(Sx)}] \to \mathrm{A}$	$\Lambda b \rho(\mathrm{x}, b)$
8.	$x = x$	0
9.	$x = y \to \mathrm{S}x = \mathrm{S}y$	0
10.	$x_1 = y_1 \to x_2 = y_2 \to x_1 + x_2 = y_1 + y_2$	0
11.	$x_1 = y_1 \to x_2 = y_2 \to x_1 \cdot x_2 = y_1 \cdot y_2$	0
12.	$x_1 = y_1 \to x_2 = y_2 \to x_1 = x_2 \to y_1 = y_2$	0
13.	$\mathrm{A} \to \mathrm{B} \to \mathrm{A}$	$\Lambda a \Lambda b a$
14.	$[\mathrm{A} \to \mathrm{B}] \to [\mathrm{A} \to \mathrm{B} \to \mathrm{C}] \to \mathrm{A} \to \mathrm{C}$	$\Lambda p \Lambda q \Lambda a q\{a\}\{p\{a\}\}$
15.	$\mathrm{A} \to \mathrm{B} \to \mathrm{A}\ \&\ \mathrm{B}$	$\Lambda a \Lambda b \langle a, b\rangle$
16.	$\mathrm{A}\ \&\ \mathrm{B} \to \mathrm{A}$	$\Lambda c \pi_1 c$
17.	$\mathrm{A}\ \&\ \mathrm{B} \to \mathrm{B}$	$\Lambda c \pi_2 c$
18.	$\mathrm{A} \to \mathrm{A} \vee \mathrm{B}$	$\Lambda a \langle 1, a\rangle$
19.	$\mathrm{B} \to \mathrm{A} \vee \mathrm{B}$	$\Lambda b \langle 2, b\rangle$
20.	$[\mathrm{A} \to \mathrm{C}] \to [\mathrm{B} \to \mathrm{C}] \to \mathrm{A} \vee \mathrm{B} \to \mathrm{C}$	$\Lambda p \Lambda q \Lambda r \gamma(p, q, r)$

21.	$[\mathrm{A} \to \mathrm{B}] \to [\mathrm{A} \to \neg \mathrm{B}] \to \neg \mathrm{A}$	$\Lambda p \Lambda q \Lambda a q\{a\}\{p\{a\}\}$
22.	$\neg \mathrm{A} \to \mathrm{A} \to \mathrm{B}$	0
23.	$\mathrm{A}_{\mathrm{x}}(\mathrm{a}) \to \exists \mathrm{x} \mathrm{A}$	$\Lambda y \langle \mathrm{a}, y \rangle$
24.	$\forall \mathrm{x} \mathrm{A} \to \mathrm{A}_{\mathrm{x}}(\mathrm{a})$	$\Lambda y y\{\mathrm{a}\}$

As an example, let us verify that the code given for the propositional axiom scheme (14) does realize it. (This axiom scheme justifies the use of deduction in mathematical reasoning: one introduces a hypothesis A, proves B and B $\to$ C and thereby C on the basis of it, and then discharges the hypothesis by concluding A $\to$ C.) Let D be

$$[\mathrm{A} \to \mathrm{B}] \to [\mathrm{A} \to \mathrm{B} \to \mathrm{C}] \to \mathrm{A} \to \mathrm{C}$$

and let d be

$$\Lambda p \Lambda q \Lambda a q\{a\}\{p\{a\}\}.$$

Let d_1 realize A $\to$ B. By (C3), to show that d realizes D we must show that $\mathrm{d}\{\mathrm{d}_1\}$ realizes $[\mathrm{B} \to \mathrm{C}] \to \mathrm{A} \vee \mathrm{B} \to \mathrm{C}$. But by (R11),

$$\mathrm{d}\{\mathrm{d}_1\} \Rightarrow \Lambda \mathrm{q} \Lambda \mathrm{a q}\{\mathrm{a}\}\{\mathrm{d}_1\{\mathrm{a}\}\}.$$

Let d_2 realize A $\to$ B $\to$ C. We need to show that

$$\big(\Lambda q \Lambda a q\{a\}\{\mathrm{d}_1\{a\}\}\big)\{\mathrm{d}_2\},$$

which reduces to $\Lambda a \mathrm{d}_2\{a\}\{\mathrm{d}_1\{a\}\}$, realizes A $\to$ C. Let d_3 realize A. We need to show that

$$\big(\Lambda a \mathrm{d}_2\{a\}\{\mathrm{d}_1\{a\}\}\big)\{\mathrm{d}_3\},$$

which reduces to $\mathrm{d}_2\{\mathrm{d}_3\}\{\mathrm{d}_1\{\mathrm{d}_3\}\}$, realizes C. But since d_3 realizes A and d_2 realizes A $\to$ B $\to$ C, $\mathrm{d}_2\{\mathrm{d}_3\}$ realizes B $\to$ C, and since d_1 realizes A $\to$ B, $\mathrm{d}_1\{\mathrm{d}_3\}$ realizes B; consequently, $\mathrm{d}_2\{\mathrm{d}_3\}\{\mathrm{d}_1\{\mathrm{d}_3\}\}$ does realize C.

We also have the following three *rules of inference*. (For rule (25), let $\mathrm{y}_1, \ldots \mathrm{y}_\mu$ be the variables occurring free in A but not in B; for rules (26) and (27), let w be a variable that does not occur in A or B):

25. If a is code for A and c is code for A $\to$ B, then the code for B is $(\mathrm{c}\{\mathrm{a}\})_{\mathrm{y}_1 \ldots \mathrm{y}_\mu}(0 \ldots 0)$.

26. If c is code for A $\to$ B and x is not free in B, then the code for $\exists$xA $\to$ B is $\Lambda \mathrm{w}((\Lambda \mathrm{x c})\{\pi_1 \mathrm{w}\})\{\pi_2 \mathrm{w}\}$.

27. If c is code for A $\to$ B and x is not free in A, then the code for A $\to \forall$xB is $\Lambda \mathrm{w} \Lambda \mathrm{x c}\{\mathrm{w}\}$.

Then for any proof of a theorem in intuitionistic arithmetic, we have an algorithm (a polynomial time algorithm!) for producing a realization code for it, and Nelson proved that this code realizes the theorem.

Since $\mathrm{S0} = 0$ is not realizable, a consequence of this argument is that $\mathrm{S0} = 0$ is not a theorem of intuitionistic arithmetic; that is, the theory is consistent. The reason that conflict with Gödel's second theorem is avoided is that the notion of realizability cannot itself be expressed within arithmetic.

III. Logic of the feasible

How should this be modified for a logic of polynomial time computation? For intuitionistic arithmetic, the realization codes are recursive partial functions; for the feasible arithmetic of $\mathcal{W}$, they should be polynomial time functions. But we have a choice: to consider ordinary functions or sorted functions. Let us choose the latter since this gives a richer theory. Then in addition to (C4) we have

(C4′) if C is $\forall^{\mathrm{st}}\mathrm{xA}$, c realizes C in case for all standard n, $\mathrm{c}\{\mathrm{n}\}$ realizes $\mathrm{A_x(n)}$.

More interestingly, in addition to (C3) we have

(C3′) if C is $\mathrm{A} \stackrel{\mathrm{st}}{\to} \mathrm{B}$, c realizes C in case for all standard a, if a realizes A then $\mathrm{c}\{\mathrm{a}\}$ realizes B.

Notice that $\forall^{\mathrm{st}}\mathrm{xA}$ is weaker than $\forall\mathrm{xA}$ but $\mathrm{A} \stackrel{\mathrm{st}}{\to} \mathrm{B}$ is stronger than $\mathrm{A} \to \mathrm{B}$.

Now we can realize induction for $\mathcal{W}$ in the form

$$\mathrm{A_x}(\epsilon) \quad \& \quad \forall\mathrm{x}[\mathrm{A} \quad \to \quad \mathrm{A_x}(0\mathrm{x}) \quad \& \quad \mathrm{A_x}(1\mathrm{x})] \quad \to \quad \forall^{\mathrm{st}}\mathrm{xA}.$$

But there is one axiom that cannot be feasibly realized, and the problem comes in a surprising place: the logical axiom scheme (14) of the propositional calculus. (Notice that (21) can be regarded as a special case of (14) in which C is $\mathrm{S0} = 0$.) This is because in the realization code $\Lambda p \Lambda q \Lambda a q\{a\}\{p\{a\}\}$ the a appears in two places. When we replace the variables p and q by realization codes d_1 and d_2, the function

$$a \mapsto \mathrm{d}_2\{a\}\{\mathrm{d}_1\{a\}\}$$

is not polynomial time. The $\mathrm{d}_2\{a\}$ may reduce in polynomial time, but there is no polynomial bound on the time it takes to evaluate a code $\mathrm{d}_2\{a\}$ of a given length on an argument $\mathrm{d}_1\{a\}$.

The problem of constructing a propositional calculus suitable for feasible reasoning demands investigation.

IV. A personal note

One of the most treasured experiences of my life is my friendship with Georges Reeb. We had many strong discussions together, intuitionist versus formalist. What he created was unique in my experience. His rare spirit, gentle but fiercely demanding of the highest standards, inspired a group of younger mathematicians with an unmatched ethos of collegiality. And their discoveries are extraordinary.

Reeb found, and led others to find, not only knowledge and beauty in mathematics, but also virtue. His insights into the nature of mathematics will point the way towards the mathematics of the future.

References

[1] Stephen Bellantoni and Stephen Cook, "A new recursion-theoretic characterization of the poly-time functions", *Computational Complexity*, 2:97–110, 1992.

[2] Stephen Bellantoni, *Predicative Recursion and Computational Complexity*, Ph.D. Thesis, University of Toronto, 1992. Available online, `ftp://ftp.cs.toronto.edu/pub/reports/theory/cs-92-264.ps.Z`

[3] Daniel Leivant, *Ramified recurrence and computational complexity I: Word recurrence and poly-time*, in Peter Cole and Jeffrey Remmel, editors, *Feasible Mathematics II*, Perspectives in Computer Science, pages 320-343, Birkhauser-Boston, New York, 1994.

[4] Stephen Cole Kleene, "On the interpretation of intuitionistic number theory", *Journal of Symbolic Logic* 10, 109–124, 1945.

[5] Stephen Cole Kleene, *Introduction to Metamathematics*, North-Holland, Amsterdam, 1980 (originally published in 1952),

[6] David Nelson, "Recursive functions and intuitionistic number theory", *Transactions of the American Mathematical Society* 61, 307–368, 1947.

`nelson@math.princeton.edu`