The Bulletin of Symbolic Logic
Volume 3, Number 3, Sept. 1997

PROGRAMS, GRAMMARS AND ARGUMENTS: A PERSONAL VIEW OF SOME CONNECTIONS BETWEEN COMPUTATION, LANGUAGE AND LOGIC

J. LAMBEK

Dedicated to the memory of Alonzo Church

As an undergraduate I was taught to multiply two numbers with the help of log tables, using the formula[1]

$$a \times b = 10^{\log a \,+\, \log b}\ .$$

Having graduated to teach calculus to Engineers, I learned that log tables were to be replaced by slide rules. It was then that I made the fateful decision that there was no need for me to learn how to use this tedious device, as I could always rely on the students to perform the necessary computations. In the course of time, slide rules were replaced by pocket calculators and personal computers, but I stuck to my original decision.

My computer phobia did not prevent me from taking an interest in the theoretical question: what is a computation? This question goes back to Hilbert's 10th problem (see Browder [1976]), which asks for an algorithm[2], or computational procedure, to decide whether any given polynomial equation is solvable in integers. It quickly leads to the related question: which numerical functions $f : \mathbb{N}^n \to \mathbb{N}$ are computable?

Received May 10, 1996; revised March 24, 1997

This paper is an expanded version of a talk given at the 1993 LICS meeting in Montreal. It had originally been intended as a contribution to a volume dedicated to Alonzo Church on his 90th birthday. Unfortunately, this volume was never published.

The author acknowledges support from the Natural Sciences and Engineering and the Social Sciences and Humanities Research Councils of Canada. He is grateful to Andreas Blass for his careful reading of the manuscript and his constructive criticism.

[1]This method has been attributed to Noah, who told the animals to go forth and multiply, whereupon one pair objected because they were *adders*.

[2]The word "algorithm" is derived from the name of the Persian mathematician Al Khowarizmi. Opinion is divided whether he was so called after the Turkish tribe of the Khwarizm or for the city of the same name in the alluvial delta of the Oxus river. The modern name of the city is Khiwa.

1079-8986/97/0303-0002/$2.70

While Hilbert's 10th problem was only resolved[3] in 1970, this related question had some earlier answers, of which I shall single out the following three[4] :

(1) f *is recursive* (Gödel, Kleene),
(2) f *is computable on an abstract machine* (Turing, Post),
(3) f *is definable in the untyped λ-calculus* (Church, Kleene).

These tentative answers were shown to be equivalent by Church [1936] and Turing [1936–7].

I shall discuss here some of the more recent developments of these notions of computability and their relevance to linguistics and logic. I hope to be forgiven for dwelling on some of the work I have been involved with personally, with greater emphasis than is justified by its historical significance.

§**1. Recursive functions.** The concept of primitive recursive function, implicit in the work of Dedekind [1888] and Peano [1889, 1891], was developed by Skolem [1923], Gödel [1931], Péter [1934] and Kleene [1952]. For a detailed historical account the reader is referred to Kleene [1981].

In my view [1988, 1992], a *primitive recursive* function $\mathbb{N}^n \to \mathbb{N}$ is an operation on $\mathbb{N}$, viewed as an algebra equipped with certain operations. We shall describe a one-sorted algebraic theory with sort N. Among the operations there are the constant $0 : N^0 \to N$ (*zero*) and the unary operation $S : N \to N$ (*successor*); moreover the set of operations is closed under the *recursion scheme*

$$\frac{g : N^n \to N \qquad h : N^{n+2} \to N}{R_{gh} : N^{n+1} \to N}$$

subject to the following identities, where $\vec{x}$ is short for $x_1 \ldots x_n$:

$$(E_{gh}) \qquad \begin{cases} R_{gh}\vec{x}0 = g\vec{x}, \\ R_{gh}\vec{x}Sy = h\vec{x}yR_{gh}\vec{x}y. \end{cases}$$

Often one also postulates the following implications[5] for an $(n+1)$-ary operation f, where $f\langle 0\rangle$ denotes the n-ary operation defined by $f\langle 0\rangle\vec{x} = f\vec{x}0$:

$$(U_{fh}) \quad f\vec{x}Sy = h\vec{x}yf\vec{x}y \Rightarrow f = R_{f\langle 0\rangle h}.$$

[3]By Yuri Matijasevič, whose achievement was based on earlier work by Martin Davis, Hilary Putnam and Julia Robinson.

[4]In particular, I will ignore the related work by Markov [1961].

[5]The data $0, S, (E_{fg})$ and (U_{fh}) are of course closely related to the well-known Peano postulates. In a cartesian closed category, they boil down to Lawvere's characterization of a natural numbers object: $1 \xrightarrow{0} N \xrightarrow{S} N$ is an initial object in the category of all diagrams $1 \xrightarrow{a} A \xrightarrow{h} A$.

As I have shown elsewhere [1988, 1992], these implications may be replaced by equations. Indeed, let $mxyz = (x + z) \dot{-} y$, where $\dot{-}$ denotes the *naïve difference*. (Actually m could be taken to be any operation satisfying $mxyy = x$ and $myyz = z$. Such operations had first been studied by Maltsev, who showed that their presence in an algebraic theory was necessary and sufficient for any two congruence relations on any algebra to permute.) Let

$$H(m, f, h)\,\vec{x}yz = m(h\vec{x}yz)\,(h\vec{x}yf\vec{x}y)\,(f\vec{x}Sy);$$

then (U_{fh}) is equivalent to[6]

$$(M_{fh}) \quad R_{f\langle 0\rangle H(m,f,h)} = f.$$

Partial and total recursive functions were introduced in lectures by Gödel [1934] and developed by Kleene [1936, 1952]. They are most easily defined as partial or total numerical functions whose graphs are recursively enumerable (by primitive recursive functions, see Kleene [1952, p. 307]).

For example, a partial numerical function in one variable may be viewed as a binary relation R on $\mathbb{N}$ such that

$$(yRx \;\wedge\; y'Rx) \Rightarrow y = y'.$$

Using the calculus of relations, we may write this as $RR^{\cup} \subseteq I$, where $R^{\cup}$ is the converse of R and I is the identity relation. To say that a nonempty binary relation R is *recursively enumerable* means that there exist primitive recursive functions f and g such that $R = gf^{\cup}$. A *partial recursive function* may then be defined as a recursively enumerable relation R which is a partial function; thus yRx if and only if

$$y = g\mu_z(fz = x),$$

where μ_x should be read as “the smallest x such that”. This formula is similar to Kleene’s characterization of partial recursive functions. Moreover, it is easily seen that $RR^{\cup} \subseteq I$ if and only if $f^{\cup}f \subseteq g^{\cup}g$. If R is to be universally defined, we also require that $I \subseteq R^{\cup}R$, which is the case if and only if f is surjective.

Primitive recursive binary relations R on $\mathbb{N}$ play a crucial rôle in Gödel’s [1931] proof of his famous incompleteness theorem. He needed to show that, for every such R, there exists a formula $\psi(x, y)$ in the formal language of arithmetic, x and y being variables of type N, such that

(a) *if* mRn *then* $\vdash \psi(S^m0,\; S^n0)$,
(b) *if not* mRn *then* $\vdash \neg\psi(S^m0, S^n0)$.

[6] More precisely,

$$U_{fH(m,f,h)} \Rightarrow M_{fh} \Rightarrow U_{fh}.$$

One must also postulate $(x + y) \dot{-} y = x$; though a consequence of (U_{fh}), this is needed to prove the sufficiency of (M_{fh}).

In the semantic version of his theorem, Gödel produced a closed formula which was not provable, say in type theory, yet *true* in what he believed to be the *real* world. Penrose [1989] and others have concluded from this that the human mind is superior to any computer: it can establish the truth of a formula which a computer cannot prove. They may well be right in this conclusion, though I cannot accept their reasoning. Not only can Gödel's argument be formalized so that a computer can mimic it, but, as I have argued elsewhere [1993c], one need not agree with Gödel about what he considered to be the real world of mathematics. By this he presumably had in mind a model of *classical* type theory, with the further property of being ω^**-complete*:

if $\exists_{x \in N}\ \varphi(x)$ *is true, then so is* $\varphi(S^n 0)$ *for some* $n \in \mathbb{N}$.

It is not obvious that such a model can be shown to exist constructively.

Anyway, intuitionists do not believe that the real world is described by classical type theory. Surprisingly, it turns out that one can construct a model of *intuitionistic* type theory which is ω^*-complete (see, e.g., Lambek and Scott [1986]). This is the so-called *free topos*, an initial object in the category of all toposes and logical morphisms. It may be constructed as the *term model* of intuitionistic type theory; in particular, it identifies truth with provability. In the free topos Gödel's unprovable formula is not true.[7]

Although the set of theorems in mathematics, or in the predicate calculus for that matter, is evidently recursively enumerable, Church [1936] proved that it is not recursive. To the best of my recollection, Chomsky made a similar statement about the set of grammatically well-formed sentences in English: it is recursively enumerable, but probably not recursive.

§2. Theoretical computers. Theoretical machines, ancestors of the modern computer, were invented in the same year by Emil Post and Alan Turing. While the Turing [1936] machine has been more popular and influential (see, e.g., Kleene [1952]), the much simpler Post [1936] machine, though largely ignored, was eloquently defended in the delightful booklet by Uspensky [1983]. It was later realized by various people (e.g., Shepherdson-Sturgis

[7] Actually, Gödel believed the real world to be ω*-complete*:

if $\varphi(S^n 0)$ *is true for all* $n \in \mathbb{N}$, *then* $\forall_{x \in N}\ \varphi(x)$ *is true*.

This property is classically, though not intuitionistically, equivalent to ω^*-completeness. His argument produces a closed formula true in any ω-complete model of *intuitionistic* type theory but not provable. However, some moderate intuitionists don't believe the real world to be ω-complete: they will accept the truth of $\forall_{x \in N}\ \varphi(x)$ only if the truth of all $\varphi(S^n 0)$ can be established in a *uniform* way, which is not the case in the object language. But it must be admitted that more extreme intuitionists think of mathematics as being language-independent; they might accept Gödel's meta-linguistic argument via ω-completeness as being uniform after all.

[1963] and Minsky [1961]) that Turing machines could be replaced by programs on what is best known as a "register machine".

My own [1961] version, which I found convenient as a teaching aid, was based on the prior work by Melzak [1961]. It was called the "abacus" and consisted of a potentially infinite set of distinguishable locations (of unlimited storage capacity). It assumed the availability of a potentially infinite supply of indistinguishable pebbles[8] . A *program* on the abacus consists of a finite set of instructions, labelled 0(= start), $1, \ldots, t-1$, t(= stop), of the following three types:

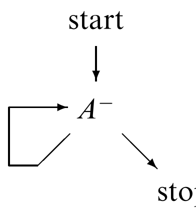
0: go to instruction $\alpha(0)$;
n: put a pebble at location $L(n)$ and go to instruction $\alpha(n)$;
n: if location $L(n)$ is nonempty, take a pebble from $L(n)$ and go to instruction $\alpha(n)$, else go to instruction $\beta(n)$.

Here $0 < n < t$ and $0 < \alpha(0),\ \alpha(n),\ \beta(n) \leq t$.

For example, consider the following program with $t = 2$.

0: go to instruction 1;
1: if possible, take a pebble from location A and then go to 1, else go to stop.

This program is illustrated by the *flow diagram*:

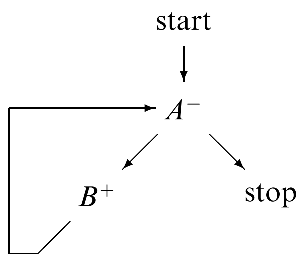


Evidently, the effect of this program is to *empty* A.

Another program with $t = 3$ is the following.

0: go to 1;
1: if possible, take a pebble from A and then go to 2, else go to stop;
2: put a pebble at B, then go to 1.

It is illustrated by the flow diagram:

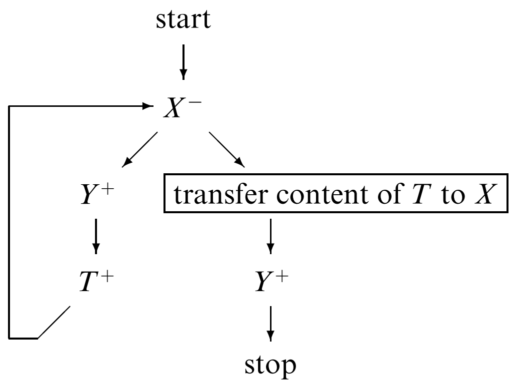


[8] The Latin word for "pebble" is "calculus", which gives rise to our word "calculate".

and its effect is to *transfer the content of A to B*.

To *calculate* a numerical function, say $z = fxy$, we begin with x pebbles at location X, y pebbles at Y and no pebbles anywhere else. We expect the calculation to end with x pebbles at X, y pebbles at Y and fxy pebbles at Z. It is often convenient to require that the other locations end up being empty.

The following diagram illustrates the program for calculating the successor function $y = Sx$. It contains our second program as a *subroutine*.



T here occurs as a *temporary storage location*; it is empty at the beginning and at the end of the calculation.

To show that all primitive recursive functions can be calculated on an abacus, we present a program for calculating the function fxy defined by recursion from functions gx and $hxyz$, which are already known to be calculable:

$$fx0 = gx,$$
$$fxSy = hxyfxy.$$

The flow diagram is in Figure 1 on the next page.

To be able to calculate all total or partial recursive functions on the abacus we need only consider one other scheme, the *minimization scheme*:

$$fx = \mu_y(gxy = 0).$$

The flow diagram for this is in Figure 2 on the next page. Of course, f is well-defined only if $\forall_{x\in N}\ \exists_{y\in N}\ gxy = 0$; otherwise f is a partial function. (In this and in the previous scheme, we have written x in place of $\vec{x}$.)

Conversely, every function calculable on the abacus is recursive. This follows from a standard argument, see, e.g., Shoenfield [1967, pp. 120 and 121].

In the course of my linguistic investigations [e.g., 1989], I found it convenient to introduce yet another device equivalent to a Turing machine, which

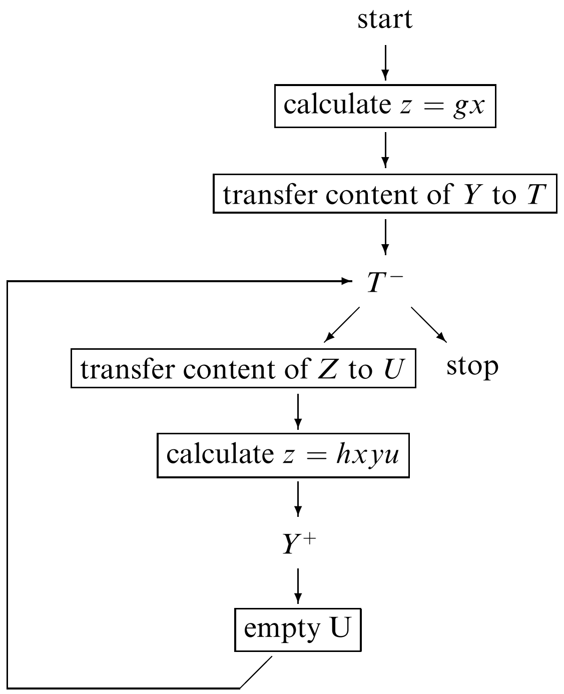


FIGURE 1.

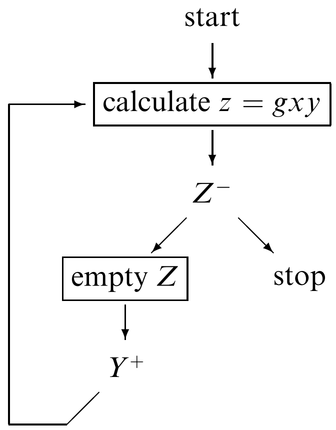


FIGURE 2.

is intended to simulate the way the human brain generates and recognizes sentences (as being well-formed).

It is evident upon reflection that even the simplest English sentence, such "*he likes her*", is generated by certain grammatical rules. These live below the conscious level, as far as the speaker is concerned, but they may be conjecturally reconstructed by a persistent grammarian. According to Chomsky [1957, 1959], they take the form of "rewrite rules" or "productions", as I [1989] prefer to call them (following Rosenbloom [1950] in his exposition of Post's [1943] canonical languages, to which even Chomsky [1959] refers).

Consider, for example, the following productions[9] to account for a tiny fragment of English, as in my [1989] and [1993d].

$$\begin{array}{ll}
S \;\to \text{Subj Pred} & P_3\ T_1 \to C_{13} \\
\text{Subj} \to NP_k\ P_k (k = 1, 2, 3) & C_{13}\ V \to V + s \\
NP_3 \to \ \textit{he, she}, \ldots & \textit{like} + s \to \textit{likes}, \ldots \\
\text{Pred} \;\to T_1\ \text{Inf} & \text{Acc } \textit{he} \to \textit{him} \\
\text{Inf} \to V\ \text{Obj} & \text{Acc } \textit{she} \to \textit{her} \\
\text{Obj} \to \text{Acc } NP_k & \ldots\ldots,
\end{array}$$

where V is any transitive verb, e.g., $V = \textit{like}$.

We note that the first six of the above productions are *context-free*, which means that a single term appears on the left of the arrow. However, the last five productions listed are not context-free. A so-called "pushdown automaton" can be used to construct sentences generated by context-free productions alone. Another pushdown automaton can be used to recognize such sentences, using a different set of moves. On the other hand, a single machine, which may be thought of as combining two pushdown automata, can be used both to generate and to recognize English sentences, by essentially the same set of moves, provided all productions are of the form $(A)B \to C(D)$, as are the above. Here and later in this section, letters A, B, C and D stand for vocabulary items, that is, English words and abbreviated grammatical terms such as S, Pred, Acc etc.

---

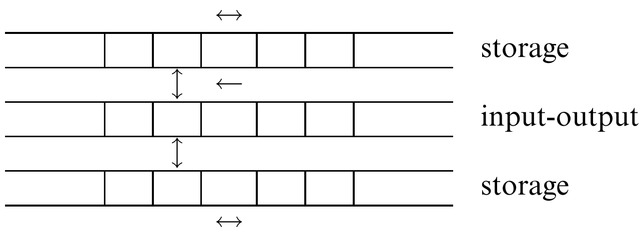
[9]Here is how these productions would generate our sample sentence:

$$\begin{array}{l}
S \to \text{Subj Pred} \to NP_3\ P_3\ \text{Pred} \to \textit{he}\ P_3\ \text{Pred} \to \textit{he}\ P_3\ T_1\ \text{Inf} \\
\to \textit{he}\ C_{13}\ \text{Inf} \to \textit{he}\ C_{13}\ \textit{like}\ \text{Obj} \to \textit{he like} + s\ \text{Obj} \\
\to \textit{he likes}\ \text{Obj} \to \textit{he likes}\ \text{Acc}\ NP_3 \to \textit{he likes}\ \text{Acc}\ \textit{she} \to \textit{he likes her}.
\end{array}$$

To recognize the same string as a sentence, we use the dual productions, reversing all the arrows, and proceed as follows:

$$\begin{array}{l}
\textit{he likes her} \to NP_3\ \textit{likes her} \to NP_3\ \textit{like} + s\ \textit{her} \to NP_3\ C_{13}\ \textit{like her} \\
\to NP_3\ P_3\ T_1\ \textit{like her} \to \text{Subj}\ T_1\ \textit{like her} \\
\to \text{Subj}\ T_1\ \textit{like}\ \text{Acc}\ \textit{she} \to \text{Subj}\ T_1\ \textit{like}\ \text{Acc}\ NP_3 \to \text{Subj}\ T_1\ \textit{like}\ \text{Obj} \\
\to \text{Subj}\ T_1\ \text{Inf} \to \text{Subj Pred} \to S.
\end{array}$$

This machine consists of three potentially infinite tapes, subdivided into squares, two storage tapes and one input-output tape. Only one square on each tape is *scanned* at any given time. The two storage tapes can move in either direction; they are pushdown storages: the left half of one tape and the right half of the other are always empty. The input-output tape is placed between the two storage tapes so that the three scanned squares are aligned; its right half is reserved for input and remains empty during sentence generation; its left half is reserved for output and remains empty during sentence recognition.



We require seven moves to generate or recognize any English sentence, assuming that all productions have the form $(A)B \to C(D)$. First, we list four "book-keeping" moves:

$$\begin{matrix} C \\ O \\ (A) \end{matrix} \xrightarrow{1} \begin{matrix} O \\ C \\ \text{stay} \end{matrix}\ , \quad \begin{matrix} O \\ B \\ O \end{matrix} \xrightarrow{2} \begin{matrix} \text{left} \\ O \\ B \end{matrix}\ , \quad \begin{matrix} C \\ B \\ (A) \end{matrix} \xrightarrow{3} \begin{matrix} \text{right} \\ \text{stay} \\ \text{stay} \end{matrix}\ , \quad \begin{matrix} O \\ B \\ A \end{matrix} \xrightarrow{4} \begin{matrix} \text{stay} \\ \text{stay} \\ \text{left} \end{matrix}\ .$$

Here "O" indicates that the scanned square is empty and "left", "right" and "stay" mean that the relevant tapes move one square to the left, to the right or remain stationary, respectively. The important moves, which have preference over the book-keeping moves, are the following.

$$\begin{matrix} O \\ B \\ (A) \end{matrix} \xrightarrow{5} \begin{matrix} (D) \\ C \\ O\ \text{right} \end{matrix}\ , \text{ if } (A)B \to C(D) \text{ is a production;}$$

$$\begin{matrix} O \\ D \\ O \end{matrix} \xrightarrow{6} \begin{matrix} \text{left} \\ \text{left} \\ \text{stay} \end{matrix}\ , \text{ if } D \text{ is an English word (during generation only);}$$

$$\begin{matrix} O \\ O \\ (A) \end{matrix} \xrightarrow{7} \begin{matrix} \text{stay} \\ \text{left} \\ \text{stay} \end{matrix}\ , \text{ during recognition only.}$$

Using the productions listed above, the machine can generate "*he likes her*" in about 30 moves, starting with S written on the scanned square of the top storage tape, all other squares being empty, and ending up with the three English words written on the output tape. Using the *dual* productions, with all arrows reversed, the machine can recognize "*he likes her*", when written on the input tape immediately to the right of the scanned square, as a sentence also in about 30 moves, so that ultimately S is written on the scanned square of the bottom storage tape. The machine does not stop, but the input tape keeps on moving to the left, waiting for further input. It may be necessary to perform a number of parallel computations. For more details, see my [1993d].

A rigorous proof of the adequacy of the above machine for generating and recognizing sentences is being written up by Riccardo Pucella [1996]. With minor modifications, the machine could also be used for translation, say from French to English.

§**3. Lambda-calculus.** Church [1941] had realized that the natural numbers and the usual arithmetical operations could be defined in terms of the untyped λ-calculus. For example,

$$2 = \lambda_f\ \lambda_x\ (f`(f`x))$$

and

$$m \times n = \lambda_x (m`(n`x)),$$

where $\lambda_x \varphi(x)$ should be read as "the function which assigns $\varphi(x)$ to x" and $f`x$ should be read as "f *of* x". Church and Kleene proved that a numerical function can be defined in the untyped λ-calculus if and only if it is recursive.

The untyped λ-calculus operates on the implausible ontological assumption that *everything is a function that is everywhere defined.* Not surprisingly, models of this calculus are hard to come by. The best known models are due to Dana Scott. However, Church [1937] had proposed a semigroup model; in my book with Phil Scott [1986] similar models were studied under the name of "C-monoids".

A C*-monoid* is a monoid $(M,\ 1,\ \cdot)$ with additional operations

$$(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{e}, {}^{*}, \langle -, - \rangle),$$

the first three being nullary and the last two unary and binary respectively, satisfying the following identities:

$$\begin{aligned}
&\boldsymbol{p}\langle x, y\rangle = x,\\
&\boldsymbol{q}\langle x, y\rangle = y,\\
&z = \langle \boldsymbol{p}x, \boldsymbol{q}z\rangle,\\
&\boldsymbol{e}\langle x^{*}\boldsymbol{p}, \boldsymbol{q}\rangle = x,\\
&y = (\boldsymbol{e}\langle y\boldsymbol{p}, \boldsymbol{q}\rangle)^{*},
\end{aligned}$$

where the multiplication symbol has been suppressed, as usual. Inasmuch as a monoid is a one-object category, a C-monoid is a one-object cartesian closed category, but with the terminal object missing. The *Freyd-Karoubi* envelope of a C-monoid is, however, a genuine cartesian closed category (CCC): its objects are idempotent elements of the monoid and its arrows $f : a \to b$ are elements such that $bfa = f$.

Church [1940] was also aware of the typed λ-calculus; in fact, he used it to present a cleaned up version of Russell and Whitehead's type theory. We now know that models of typed λ-calculus are precisely the cartesian closed categories of Lawvere [1969] and that models of intuitionistic type theory are the elementary toposes of Lawvere [1972] and Tierney [1972]. Computer scientists have accepted CCCs, but not toposes; they would like the object of *truth values*, a Heyting algebra, to be replaced by an object of *proofs*, a CCC. Such "metatoposes" have recently been studied by Duško Pavlović [1995].

Unfortunately, Church's construction of the natural numbers won't work in typed λ-calculus, unless one postulates a type of natural numbers with appropriate properties. But even then one cannot represent all recursive functions, but only the *provably recursive functions* (see Thibault [1982]). Another method, which has recently found favour in computer science, is to allow variable types and quantification over such. One is thus led to the *polymorphic λ-calculus* of Girard [1972, 1986] and Reynolds [1974, 1984]. The natural numbers object may be constructed in a model of polymorphic λ-calculus as a retract of

$$\Pi_X \, (X^X)^{(X^X)} \cong \Pi_X \, X^{(X^{X+1})},$$

where Π_X is a kind of *formal product* in a CCC. A CCC with formal products has been called a $CCCP$ in my recent investigations [1993 a,b].

The best known model of polymorphic λ-calculus was discovered by Moggi (see Longo and Moggi [1989]) and lives inside Hyland's [1982, 1988] effective topos. It is the category PER, which may be constructed from the monoid of all partial recursive functions from $\mathbb{N}$ to $\mathbb{N}$. Curiously, this monoid closely resembles a C-monoid, except that all equations in the definition of the latter are replaced by inequalities. (This will explain why we have written some of these equations in what looks like the wrong way round.)

Let us pause for a moment to see why the set of partial recursive functions (or even the set of recursively enumerable relations) from $\mathbb{N}$ to $\mathbb{N}$ is closed under relative products. Let $R = gf^{\cup}$ and $S = kh^{\cup}$, where f, g, h and k are primitive recursive. Then $h^{\cup}g$ is recursive, hence recursively enumerable, so $h^{\cup}g = qp^{\cup}$ for some primitive recursive functions p and q. It follows that $SR = (kq)\,(fp)^{\cup}$ is also partial recursive.

The typed λ-calculus without natural numbers type may also be viewed as the proof theory of the positive intuistionistic propositional calculus, via the so-called Curry-Howard isomorphism (see Howard [1980]). Influenced by Curry's [1952] work, I had arrived at the same conclusion [1972, 1974, 1980] by realizing that both the λ-calculus and the proof theory are essentially *CCC*s.

Curry had noticed that formulas of the calculus of implication were the same as the types appearing in the λ-calculus, although he preferred to work with "combinators" rather than λ-terms, in order to avoid bound variables. His insight gave rise to the slogan "formulas as types". Curry [1961] also noticed that the same types could be used to denote English word classes. For example, in "*John snores loudly*", *John* has type n, *snores* has type $n \Rightarrow s$ and *loudly* has type $s \Rightarrow s$. In retrospect, Curry's types should be called *semantic types*. His idea was further developed by Montague [1974], who brought the full power of the λ-calculus to bear on the semantics of natural languages.

Quite independently [1958], I had introduced a calculus of *syntactic types*, which distinguishes two kinds of implication, reflecting the non-commutativity of concatenation. Thus, *he* has type $s \Leftarrow (n \Rightarrow s)$, or $s/(n\backslash s)$, as I preferred to write, using an algebraic notation with antecedents in Ajdukiewicz [1935] and Bar-Hillel [1955]. As a logical system, this calculus differed from positive intuitionistic logic by the absence of Gentzen's three structural rules (see, e.g., Kleene [1952]). On the other hand, Girard's [1987] *linear logic* allows one of Gentzen's rules, the interchange rule, but it also takes into account some additional operations and it generalizes classical logic. Whereas Curry's semantic calculus corresponds to a *CCC*, the syntactic calculus corresponds to a "biclosed monoidal category" and linear logic corresponds to Barr's [1979] "star-antonomous categories" (see Seely [1987]).

My basic idea was to push all the grammar into the dictionary, assigning to each English word one or more types, and to use the calculus to decide whether a string of words is a grammatically well-formed sentence. For example, "*he snores loudly*" is shown to be a sentence by proving that

$$(s/(n\backslash s))\ (n\backslash s)\ (s\backslash s) \to s.$$

Among the theorems of the syntactic calculus there appeared not only contractions such as $n(n\backslash s) \to s$, but also expansions such as $n \to s/(n\backslash s)$. Fortunately, Gentzen's cut elimination theorem could be used to provide a decision procedure; in fact, its proof was much simplified by the absence of the structural rules.

In view of this decision procedure, the class of sentences recognized by such a grammar is recursive, not just recursively enumerable. This observation led Chomsky to conjecture that such a grammar is context-free. After a number of people had attacked this conjecture unsuccessfully, it was finally proved by Mati Pentus [1993]. However, Kanazawa [1992] showed that, if one also admitted intersections of syntactic types, such a grammar could also recognize intersections of context-free languages. It so happens that the best known examples of non-context-free languages are intersections of context-free ones, and it is therefore conceivable that all natural languages can be recognized by such grammars after all. I must confess that, in my own linguistic investigations (e.g., [1989]), I have abandoned these grammars in favour of production grammars, even while there has been a revival of interest in them elsewhere (see, e.g., Buszkowski et al. [1988], Oehrle et al. [1988], Morrill [1988] and Moortgat [1988]). A classical version of the syntactic calculus, or non-commutative linear logic, is now being investigated by Claudia Casadio.

§4. On the unity of algebra, logic and linguistics. This survey was intended as a discussion of some recent developments arising from the equivalent notions of recursiveness, computability and λ-definability implicit in the so-called Church-Turing thesis. With excursions into logic and linguistics, I may have appeared to jump from topic to topic. If pushed to produce a unifying theme, I would suggest the application of concepts of universal algebra to logic and linguistics.

A multisorted algebraic *operation* has the form $A_1 \cdots A_m \to A_{m+1}$, where the A_i are sorts or types. In most familiar algebras there is only a single sort A and one writes $A^m \to A$. As pointed out in Section 1, primitive recursive functions are such operations. It is crucial that all conditions on operations can be written as equations.

In logic, equations between operations are usually ignored and a multisorted operation is just a Gentzen-style *deduction* or "sequent"; the types are now taken to be formulas. Both operations in algebra and deductions in logic are subject to Gentzen's three structural rules: *interchange*, *contraction* and *weakening*, forming operations f^i, f^c and f^w respectively from a given operation f. In the algebraic context, say in case $m = 2$, these are subject to the equations:

$$f^c x = fxx, \quad f^i xy = fyx, \quad f^w xyz = fxy.$$

All three rules are used to form new primitive recursive functions from old ones.

In linguistics, multisorted operations appear as *context-free productions*, although the direction of the arrow must be reversed from that in generative grammar as in Chomsky [1957]. However, in syntax, as opposed to semantics, the three structural rules are absent. They are also absent in non-commutative linear logic and in multilinear algebra; in linear logic the interchange rule alone is admitted.

When looking at classical, rather than intuitionistic logic, Gentzen allowed a more general kind of operation:

$$A_1 \cdots A_m \to B_1 \cdots B_n \ ,$$

where juxtaposition on the left stands for conjunction and juxtaposition on the right for disjunction. In the absence of the structural rules, conjunction should be replaced by the tensor product and disjunction by its dual, which Girard [1987] denotes by an upside-down ampersand. However, in Hopf algebras and in general production grammars the tensor product and its dual coincide.

In addition to operations, also binary relations play an important, though subsidiary role in algebra, logic and linguistics. We have already mentioned the permutability of congruence relations in motivating Maltsev operations and we pointed out that partial recursive functions are most easily defined as binary relations on $\mathbb{N}$. In anthropological linguistics, there is an emphasis on kinship relations, see, e.g., Bhargava and Lambek [1995] for a recent reference. Moreover, binary relations on a set offer a simple model for the syntactic calculus. Thus, given binary relations R and S, one may define

$$x(R/S)y \Leftrightarrow \forall_z (ySz \Rightarrow xRz) \ .$$

REFERENCES

[1935] K. Ajdukiewicz, *Die syntaktische Konnexität*, ***Studia Philosophica***, vol. 1, pp. 1–27.

[1953] Y. Bar-Hillel, *A quasi-arithmetical notation for syntactic description*, ***Language***, vol. 29, pp. 47–58.

[1979] M. Barr, ***∗-autonomous categories***, Springer Lecture Notes in Mathematics, vol. 752, Springer-Verlag, Berlin.

[1995] M. Bhargava and J. Lambek, *A rewrite system of the Western Pacific*, ***Theoretical Linguistics***, vol. 21.

[1976] *Mathematical developments arising from Hilbert problems*, ***Proceedings of the Symposium on Pure Mathematics 28*** (F. E. Browder, editor), American Mathematical Society, Providence, R.I.

[1988] W. Buszkowski, W. Marciszewski, and J. van Benthem, ***Categorial grammar, linguistic & literary studies in Eastern Europe***, vol. 25, John Benjamins Publishing Co., Amsterdam.

[1957] N. Chomsky, ***Syntactic structures***, Mouton and Co., The Hague.

[1959] ———, *On certain formal properties of grammars*, ***Information and Control***, vol. 2, pp. 137–167.

[1936] A. Church, *A note on the Entscheidungsproblem*, ***The Journal of Symbolic Logic***, vol. 1, pp. 40–41, 101–102.

[1937] ———, *Combinatory logic as a semigroup, abstract*, ***Bulletin of the American Mathematical Society***, vol. 43, p. 353.

[1940] ———, *A formulation of the simple theory of types*, ***The Journal of Symbolic Logic***, vol. 5, pp. 56–68.

[1941, 1951] ———, ***The calculi of lambda conversion***, Annals of Mathematical Studies, vol. 6, Princeton University Press.

[1952] H. B. CURRY, ***Leçons de logique algébrique***, Gauthier-Villars, Paris.

[1961] ———, *Some logical aspects of grammatical structure*, ***American Mathematical Society Proceedings of the Symposia on Applied Mathematics 12***, pp. 56–68.

[1888] R. DEDEKIND, ***Was sind und was sollen die Zahlen***, Braunschweig, translated in ***Essays on the theory of numbers***, Dover Publ., New York, N.Y., 1963.

[1972] J.-Y. GIRARD, *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur*, ***Thèse de doctorat d'état***, Paris.

[1986] ———, *The system F of variable types fifteen years later*, ***Theoretical Computer Science***, vol. 45, pp. 159–192.

[1987] ———, *Linear logic*, ***Theoretical Computer Science***, vol. 50, pp. 1–102.

[1931] K. GÖDEL, *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I*, ***Monatshefte für Mathematik. Phys.***, vol. 38, pp. 173–198, translated in J. VAN HEIJENOORT, ***From Frege to Gödel***, Harvard University Press, Cambridge, Massachusetts, 1967, pp. 596–616.

[1934] ———, *On undecidable propositions of formal mathematical systems*, notes by S. C. Kleene and J. B. Rosser on lectures at the Institute for Advanced Study; reprinted in M. DAVIS, ***The undecidable***, Raven Press, Hewlett, N.Y., 1965, pp. 39–74.

[1949] L. A. HENKIN, *The completeness of the first-order functional calculus*, ***The Journal of Symbolic Logic***, vol. 14, pp. 159–166.

[1980] W. A. HOWARD, ***The formulae-as-types notion of construction*** (J. P. Seldin et al., editors), pp. 479–490.

[1982] J. M. E. HYLAND, *The effective topos*, ***The L. E. J. Brouwer Centenary Symposium*** (A. S. Troelstra and D. van Dalen, editors), North-Holland, Amsterdam, pp. 165–216.

[1988] ———, *A small complete category*, ***Annals of Pure and Applied Logic***, vol. 40, pp. 135–165.

[1992] M. KANAZAWA, *The Lambek calculus enriched with additional connectives*, ***Journal of Logic, Language and Information***, vol. 1, pp. 141–171.

[1936] S. C. KLEENE, *General recursive functions of natural numbers*, ***Mathematische Annalen***, vol. 112, pp. 727–742.

[1952] ———, ***Introduction to metamathematics***, Van Nostrand Co., New York.

[1981] ———, *The theory of recursive functions, approaching its centennial*, ***Bulletin of the American Mathematical Society***, vol. 5, pp. 43–60.

[1958] J. LAMBEK, *The mathematics of sentence structure*, ***The American Mathematical Monthly***, vol. 65, pp. 154–170.

[1961] ———, *How to program an (infinite) abacus*, ***Canadian Mathematical Bulletin***, vol. 4, pp. 295–302.

[1972] ———, ***Deductive systems and categories III***, Springer Lecture Notes in Mathematics, vol. 274, Springer-Verlag, pp. 57–82.

[1974] ———, *Functional completeness of Cartesian categories*, ***Annals of Mathematical Logic***, vol. 6, pp. 259–292.

[1980] ———, *From λ-calculus to Cartesian closed categories*, (J. Seldin et al., editors), pp. 375–402.

[1988] ———, *On the unity of algebra and logic*, ***Categorical algebra and its applications*** (F. Borceux, editor), Springer Lecture Notes in Mathematics, vol. 1348, pp. 221–229.

[1989] ———, *Grammar as mathematics*, ***Canadian Mathematics Bulletin***, vol. 31, pp. 1–17.

[1992] ———, *On the ubiquity of Mal'cev operations*, ***Proceedings International Conference on Algebra*** (L. A. Bokut et al., editors), Contemporary Mathematics, vol. 131, pp. 135–146.

[1993a] ———, *Some aspects of categorical logic*, ***Proceedings of the 9th international congress of logic, methodology and philosophy of science, Uppsala 1991*** (D. Prawitz, B. Skyrms, and D. Westerstahl, editors), Elsevier, Amsterdam.

[1993b] ———, *Least fixpoints of endofunctors of cartesian closed categories*, ***Mathematical Stuctures in Computer Science***, vol. 3.

[1993c] ———, *Are the traditional philosophies of mathematics really incompatible?*, ***The Mathematical Intelligencer***, vol. 15.

[1993d] ———, *Production grammars revisited*, ***Linguistic Analysis***, vol. 23.

[1986] J. LAMBEK and P. J. SCOTT, ***Introduction to higher order categorical logic***, Cambridge University Press, Cambridge.

[1972] *Introduction*, ***Toposes, algebraic geometry and logic*** (F. W. Lawvere, editor), Springer Lecture Notes in Mathematics, vol. 274, Springer-Verlag, pp. 1–12.

[1969] ———, *Adjointness in foundations*, ***Dialectica***, vol. 23, pp. 281–296.

[1989] G. LONGO and E. MOGGI, *Constructive natural deduction and its "modest" interpretation*, ***Semantics of natural and computer languages*** (J. Meseguer et al., editors), M.I.T. Press, Cambridge, Massachusetts.

[1961] A. A. MARKOV, ***Theory of algorithms***, National Science Foundation, Washington, D.C.

[1961] Z. A. MELZAK, *An informal arithmetical introduction to computability and computation*, ***Canadian Mathematical Bulletin***, vol. 4, pp. 279–293.

[1961] M. MINSKY, *Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines*, ***Annals of Mathematics***, vol. 74, pp. 437–454.

[1974] R. MONTAGUE, ***Formal philosophy; selected papers of Richard Montague*** (R. H. Thomason, editor), Yale University Press, New Haven, Connecticut.

[1988] M. MOORTGAT, ***Categorial investigations***, Groningen-Amsterdam studies in semantics, no. 9, Foris Publications, Dordrecht.

[1988] G. MORRILL, ***Type logical grammar***, Kluwer Academic Publishers, Dordrecht.

[1988] R. T. Oehrle, E. Bach, and D. Wheeler (editors), ***Categorial grammars and natural language structures***, Reidel Publ. Co., Dordrecht.

[1995] D. PAVLOVIĆ, *On completeness and cocompleteness*, ***Annals of Pure and Applied Logic***, vol. 74, pp. 121–152.

[1889] G. PEANO, ***Arithmetices principia, nova methodo exposita***[10], Bocca, Turin.

[1891] ———, *Sul concetto di numero*, ***Rivista di Mathematica***, vol. 1, pp. 87–102, 256–267.

[1989] R. PENROSE, ***The emperor's new mind***, Oxford University Press, New York.

[1993] M. PENTUS, *Lambek grammars are context free*, ***Proceedings of the 8th LICS Conference, Montreal***, pp. 429–433.

[1934] R. PÉTER, *Über den Zusammenhang der verschiedenen Begriffe der rekursiven Funktion*, ***Mathematische Annalen***, vol. 110, pp. 612–632.

[1936] E. POST, *Finite combinatory processes—formulation I*, ***The Journal of Symbolic Logic***, vol. 1, pp. 103–105.

[1943] ———, *Formal reductions of the general decision problem*, ***American Journal of Mathematics***, vol. 65, pp. 197–215.

[1996] R. R. PUCELLA, *Production grammars, machines and syntactic translation*, ***Technical Report SOC5-96.6***, McGill School of Computer Science.

[10]This book was written in language invented by Peano, which I believe is surviving today as *Interlingua*.

[1974] J. C. REYNOLDS, *Towards a theory of type structure*, ***Programming Symposium '74*** (B. Robinet, editor), Springer Lecture Notes in Computer Science, vol. 19, Springer-Verlag, pp. 408–425.

[1984] ———, *Polymorphism is not set-theoretic*, ***Symposium on semantics of data types*** (Kahn et al., editors), Springer Lecture Notes in Computer Science, vol. 173, Springer-Verlag, pp. 145–156.

[1950] P. C. ROSENBLOOM, ***The elements of mathematical logic***, Dover Publications, New York, N.Y.

[1987] R. A. G. SEELY, *Linear logic and coalgebras*, ***Categories in computer science and logic*** (J. W. Gray and A. Scedrov, editors), Contemporary Mathematics, vol. 92.

[1980] J. P. Seldin and J. R. Hindley (editors), ***To H. B. Curry: Essays on combinatory logic, lambda calculus and foundations***, Academic Press, London.

[1963] J. C. SHEPHERDSON and R. E. STURGIS, *Computability of recursive functions*, ***Journal of the Association for Computing Machinery***, vol. 10, pp. 217–255.

[1967] J. R. SHOENFIELD, ***Mathematical logic***, Addison-Wesley Publishing Co., Reading Massachusetts.

[1923] T. SKOLEM, *Begründung der elementaren Arithmetik durch die rekurrierende Denkweise*, English translation in J. VAN HEIJENOORT, ***From Frege to Gödel***, Harvard University Press, Cambridge, Massachusetts, 1967, pp. 302–333.

[1982] M.-F. THIBAULT, *Prerecursive categories*, ***Journal of Pure and Applied Algebra***, vol. 24, pp. 79–93.

[1972] M. TIERNEY, *Sheaf theory and the continuum hypothesis*, ***Toposes, algebraic geometry and logic*** (F. W. Lawvere, editor), Springer Lecture Notes in Mathematics, vol. 274, pp. 13–42.

[1936] A. M. TURING, *On computable numbers, with an application to the Entscheidungsproblem*, ***Proceedings of the London Mathematical Society***, vol. 42, pp. 230–265.

[1937] ———, *On computable numbers, with an application to the Entscheidungsproblem*, ***Proceedings of the London Mathematical Society***, vol. 43, pp. 544–546.

[1983] V. A. USPENSKY, ***Post's machine***, Mir Publishers, Moscow.

DEPT. OF MATHEMATICS
MCGILL UNIVERSITY
MONTREAL, QUEBEC, CANADA, H3A 2K6