Constructivity in the Nineties *

Michael Beeson
Department of Mathematics and Computer Science
San Jose State University
San Jose, California 95192
USA
email: beeson@cats.ucsc.edu

March 9, 1998

1 Introduction

In the last ten years, what Heyting called "the constructive trend in mathematics" has broadened and branched, partly in response to the impact of the computer. This paper gives an overview of these developments. Specifically, we will discuss:

- constructive type theories in computer science
- semantics of constructive systems
- actual implementation of constructive systems
- computer algebra as constructive mathematics
- spread of the computational viewpoint in mathematics
- constructivity and the sciences

2 Historical Origins of the Subject

It is worthwhile to review the original reasons for interest in constructivity. An interesting starting point is the controversy over Zermelo's proof that the set of real numbers can be well-ordered. Zermelo presented this proof to the International Congress of Mathematicians in 1906. It was not well-received,

*Published in the *Proceedings of the International Wittgenstein Symposium on Philosophy of Mathematics*, Kirchberg am Wechsel (near Vienna), Austria, Aug. 16-22, 1992.

primarily because Zermelo could not exhibit the well-ordering he claimed to have proved existed. Zermelo responded, as a mathematician must, by writing his proof up in more detail. This process led to the first explicit formulation of the axiom of choice.

This was not the first such controversy, for there had already been a similar furor over Hilbert's proof of existence of invariants. But Zermelo's proof came just three years after Russell's paradox had shaken the foundations of mathematics, and it seemed to some that the wolf was at the door, because Zermelo's theorem concerned not some abstract set that could be dispensed with if it turned out to be contradictory, but the real numbers themselves.

The climate was ripe for a general philosophical reflection on the meaning of existence. Such investigations were undertaken by Brouwer in the second decade of the century, and in the twenties Hilbert's proof theory was intended to address the same issue. Brouwer and Hilbert are often portrayed as antagonists, but it is important to remember that they were bothered by the same thing–existence proofs that could not be supported by example-giving. Their differences arose only in what they proposed to do about this difficulty. Brouwer wanted to give up all such proofs and any parts of mathematics that required them. Hilbert wanted a systematic way to show that they were unnecessary in principle.

The efforts to make the questions precise led to proof theory, which in turn led to the precise formulation of the *Entscheidungsproblem*, or decision problem for predicate calculus. This problem in turn led to the development of mathematical theories of computation by Turing, Church, Kleene, and Gödel in the thirties, and to the incompleteness theorems of Gödel. The point is that, directly or indirectly, the troublesome business of existence proofs without examples was the stimulus to all these important developments.

In the forties, Gentzen and Kleene led the way back to the original questions, armed with the new technical tools. With their inventions of cut-elimination and recursive realizability, they were able to show how exemplifying information (examples and algorithms for constructive examples) could be extracted from certain kinds of formal proofs. With the help of Gödel's two interpretations (the double-negation and the Dialectica)[1] it could be said that some progress was made in the direction pointed to by Hilbert, using tools invented by those following in the footsteps of Brouwer.

The computer was born about 1950, but had little immediate influence on the subject. A man thirty years ahead of his time, N. de Bruijn, built AUTOMATH in the fifties [14], but not until the eighties did this line of work get carried forward. The fifties were a period of consolidation, in which Kleene's book on intuitionistic analysis, many papers of Kreisel, Heyting's book, and the Amsterdam volume *Constructivity in Mathematics* appeared, applying the tools of proof theory and realizability systematically to various systems. This devel-

[1] The Dialectica interpretation was known in the early forties, although it was not published until 1958.

opment continued into the sixties, and became increasingly technical. These technical developments, while of great interest in their own right, did not cast further light on the philosophical issues raised by Zermelo's proof. Indeed, in the meantime mathematics had marched ahead, and the non-constructive existence proof held sway in the land without dissension, save for a handful of proof-theorists, who were not even considered "real mathematicians".

Bishop's book, *Foundations of Constructive Analysis* [11], was written to show that modern analysis does have a constructive content. Bishop hoped that his work would produce a widespread appreciation for the constructive viewpoint among the mathematical community, and died disappointed with the results. But his work has been very influential, although not perhaps as directly or as quickly as he had hoped. His work stimulated the logicians to develop new theories, capable of formalizing constructive mathematics more naturally. These theories, due to Myhill [50], Friedman, and Feferman [22], and the type theories of Martin-Löf [48] (which were developed independently of Bishop's influence), were formulated and studied during the 1970's and half the 1980's. The results are summarized in [4].

3 Type theory and the Curry-Howard isomorphism

Curry and Howard realized that typed lambda-terms could be thought of as denoting formal proofs, and that types are isomorphic to propositions. A proof of a proposition is a lambda-term of the type of that proposition. The rule of inference

$$\frac{A \qquad A \to B}{B}$$

corresponds to the rule of typed λ-calculus

$$\frac{p : A \qquad q : A \to B}{qp : B}$$

and the rule

$$\frac{p : B(x : A)}{\lambda x : A.p : A \to B}$$

corresponds to the logical rule

$$\frac{\begin{array}{c}[A] \\ \hline B\end{array}}{A \to B}$$

In this way typed lambda-calculus represents the implicational fragment of a natural-deduction logical calculus. Other type-forming operations correspond

to other logical connectives. For example, Cartesian product $A \times B$ corresponds to conjunction. A proof of $A \wedge B$ is just a member of the product $A \times B$.

The natural question was, what type-formation operations are required to extend this isomorphism to include disjunction and quantifiers? Disjunction corresponds to the "disjoint union" type-forming operation, in which a member of $A + B$ is a member of A or of B together with a "label" (for example 0 or 1) telling which.

Quantification on the logical side requires "dependent products" and "dependent sums" on the type-theory side. Dependent products are written

$$(\Pi x : A)B(x)$$

and intuitively their members are "functions" such that for $x : A$, we have $f(x) : B(x)$. Note that the set-theoretic range would be the union of the $B(x)$ for $x : A$, which doesn't exist in the type theory. Dependent sums

$$(\Sigma x : A)B(x)$$

are a kind of generalized Cartesian product; the members are pairs (x, y) with $x : A$ and $y : B(x)$.

These were introduced by Martin-Löf around 1970. His type theories were refined and developed throughout the 1970's; see [48] and [4] for detailed formulations.

Girard's thesis introduced his System F, which has a second-order type formation rule, allowing quantification over types. On the logical side, this corresponds to second-order propositional logic. The rules of System F are fairly simple:

If U and V are types, then $U \to V$ is a type.

If V is a type and X a type variable, then $\Pi X.V$ is a type.

If T is a type then we have variables of type T. In other words, all variables are labelled with a type.

Terms of type T are formed by (first-order) application and λ-abstraction, or by second-order application and Λ-abstraction:

If $v : V$ then $\Lambda X.v : \Pi X.V$, provided X is not free in the type of a free variable of v.

If $t : \Pi X.V$ and U is a type, then $tU : V[U/X]$.

This system was rediscovered by Reynolds [54], who called it second-order lambda calculus. Girard came to System F from proof theory; Reynolds came to it from programming language theory. This shows what a central position it occupies.

Observe that Martin-Löf's type theory extends the Curry-Howard isomorphism to systems including first-order predicate calculus and Peano's arithmetic,

while Girard's System F corresponds to second-order propositional calculus but does not include the dependent-sum and dependent-product type formation operations. System F, with its impredicative second-order type-formation rule, can be used to interpret second-order Peano arithmetic, so it is proof-theoretically quite strong; Martin-Löf's theories are much weaker. Still there doesn't seem to be a natural embedding of Martin-Löf's theories in System F–at least I don't know one.

System F has been extended to system F^ω, which corresponds on the logical side to the theory HA^ω, that is, the theory of the first ω levels of the cumulative hierarchy of sets. This theory is further extended by the Coquand-Huet theory of constructions [18].

Feferman has introduced another variety of type theory, beginning in 1975 [22]. The type theories of Girard and Coquand-Huet are far from first-order logic. Feferman's theories, in essence, reduce type theory to first-order logic again by introducing a predicate "is a type", regarding the relation $x : A$ as a binary predicate, and stating type formation rules simply as ordinary axioms of a first-order theory. The intuitive motivation is that the variables range over "objects", including data objects, algorithms, and representations of types as data objects (names of the types). One can equivalently use a two-sorted predicate logic, with capital letters for types, lower-case letters for objects, and a binary relation $E(a, A)$ to mean a is a name of A. One obtains different variants of the theory according to which type-formation axioms are allowed. One interesting type-formation principle is the *elementary comprehension axiom*, which allows the formation of the type $c_\phi = \{x : \phi(x)\}$ where ϕ is *elementary*, which means in essence that only free type variables are allowed on the right of the colon in ϕ, and the name relation E does not occur in ϕ. Another principle, *join*, allows the formation of dependent types, and consequently the embedding of Martin-Löf style type theories in Feferman's theories. Details can be found in [4].

Bringing this subject at last into the nineties, Feferman showed that adding a second-order comprehension axiom to his theories allows him to embed Girard's System F [24], [23]. The model constructions Feferman used to prove the consistency of these theories are unrelated to those previously used to prove the consistency of System F, and hence cast new light on this result. Moreover, it is helpful to have a single system that unifies the type theories of Martin-Löf and Girard. Further developments of Feferman's type theories may be found in [25], [26], [27].

From the point of view of applying type theory to the actual formalization of constructive mathematics, one problematic point is the notion of *subtype*. This notion is supported only indirectly by Martin-Löf's theories and by System F, via explicit embeddings. Feferman's theories offer a more natural treatment of subtypes. Other researchers, for example Curien [16] and Cardelli [15], are working on ways of adding subtyping to System F.

4 Semantics of Constructive Systems

Work on the semantics of constructive systems goes back to Beth and Kripke, who introduced certain kinds of models for Heyting's predicate calculus in the 1930's (see [63] for an introduction), and to Kleene, who introduced the realizability semantics in the 1940's.

Half a century later, there are dozens or maybe even hundreds of people working on the semantics of various constructive theories. Many different kinds of semantics have been discovered, and at the same time, the various kinds of semantics have been related and unified using the tools of category theory. This explosion began with Scott's construction of models of the untyped λ-calculus in 1969 (see [3] for the details of this construction).

Scott's construction and category theory. In the 1970's an important connection was made between the λ-calculus and category theory. It was realized (by Lambek) that there is a one-one correspondence between models of typed λ-calculus and Cartesian closed categories. (A Cartesian closed category is a category in which one can form from two objects a new object A^B behaving like the space of arrows from A to B.) Previously, models of typed λ-calculus had been constructed by starting with models of untyped λ-calculus, and selecting out the typed elements. Now, thanks to Scott and Lambek, one could reverse the process. Scott constructed Cartesian closed categories in which a certain object U satisied U^U isomorphic to U, so models of untyped λ-calculus could be constructed from typed models.

Topos theory as semantics. The seventies also saw the development of topos-theoretic semantics of intuitionistic systems. A topos is a Cartesian closed category with an object Ω to play the role of the type of propositions, and a "subobject classifier", which essentially means that arrows into Ω behave like characteristic functions, that is, there is an object in the category behaving like $\{x \in A : \phi(x) = t\}$, when ϕ is an arrow from A to Ω. For an introduction to topos theory, one should read [34]. The connection between topos theory (which was independently developed) and intuitionistic systems grew out of the realization that if a complete Heyting algebra H (the intuitionistic analogue of a Boolean algebra) is given, then the H-sets (defined inductively as functions from H-sets to H) form a topos.

In connection with category theory, it is natural to consider "intuitionistic type theory" HA^ω, which differs from the type theories considered above in that there is a "power type" operation constructing the power type $P(A)$ of a type A. This type theory is more like the simple type theory of traditional set theory. The term model of HA^ω is the "free topos". Category-theoretic techniques due to Peter Freyd were used to show that the free topos T satisfies the existence and disjunction properties: if $p \vee q$ holds in T, then p holds in T or q holds in T, and if $(\exists x : A)B$ holds in T then there is an arrow $a : 1 \to A$ such that $B[a/x]$ holds in T. These techniques turned out to be equivalent to known variants of realizability on the proof-theoretic side, but the connection

to topos theory helped to systematize these constructions, and give what had been purely syntactic translations a semantic appearance. For the details of these concepts and results, one should read [46].

Realizability Topos. Martin Hyland [40] invented the "realizability topos", which gave realizability a semantic appearance similar to H-sets, and made it easier to extend realizability to complicated formal theories. This is well-explained in [64], p. 724 *ff.*, as well as in Hyland's paper. Even though the realizability topos is now more than ten years old, there are still open questions about it, for example, whether internal realizability is the same as external realizability.[2]

Recent work of Phoa [52] represents the latest word from this direction of research: Phoa constructs an "effective topos" over a term model of the λ-calculus in which closed terms are identified if they have the same Böhm tree.

Metric Semantics. Scott's model construction was based the category of complete partial orders. A natural question is whether there are other interesting categories that permit the construction of models of type theory in which "domain equations" such as $A \to A = A$ can be solved. One interesting answer is *metric semantics*, invented by de Bakker and Zucker (in connection with a study of "process algebras" in computer science). In metric semantics, types are metric spaces with diameter at most 1. $A \to B$ gets the sup metric. Morphisms in this category are maps that do not increase distance, that is $d(fx, fy) \leq d(x, y)$. Direct limits in this category C generally don't exist, but as America and Rutten [1] showed, the sequences arising in Scott's construction *do* have direct limits, *provided* we insert contractions by some $\epsilon < 1$ at each stage, so Scott's construction can be transposed to this category, almost! The result is that if we consider equivalence classes of metric spaces (two being equivalent if they have the same shape but possibly different size) then in this category we can solve domain equations.

Wagner [65] noticed that the category C is not Cartesian closed, because the evaluation map can increase distances, and so is not a morphism in the category.[3] A metric space is called *ultra-metric* if $d(x, y) \leq \max\{d(x, z), d(z, y)\}$. Wagner proved that the subcategory W of C formed by ultra-metric spaces is Cartesian closed. Moreover, if a sequence of ultra-metric spaces has a direct limit in C, that direct limit is in W too.

Up to this point the ultrametric semantics looks like a formal footnote to metric semantics, but Wagner showed that it is very natural. Every ultrametric space in category C can be associated with an Ω-set based on the complete Heyting algebra $[0, 1]$. (Let $[\![x = y]\!] = d(x, y)$, so distance zero means true, distance 1 means false.) Now the condition that morphisms be distance-non-increasing becomes the internal version of an equality axiom $x = y \to fx = fy$.

[2] In arithmetic, $\exists e(e\mathbf{r}A)$ is equivalent to its realizability interpretation, because the formula $e\mathbf{r}A$ belongs to the syntactic class of almost-negative formulae. This argument doesn't extend immediately to higher types.

[3] $d(fx, gy)$ can be greater than $d((f, x), (g, y)) = \sup\{d(x, y), d(f, g)\}$

Transitivity of equality is directly verified using the strong triangle inequality. The internal equality between elements of the power set of an Ω-set turns out to be the Hausdorff distance between subsets of a metric space, which has long been an important mathematical tool.[4]

Fractal models of lambda-calculus. Fractals are direct limits of subsets of a metric space (usually the plane) in the Hausdorff metric, so their complements are inverse limits. In adapting Scott's construction of a model of lambda calculus to the metric semantics, we use inverse limits of metric spaces A_n (defined by $A_0 = \mathbf{N}$ and $A_{n+1} = A_n \to A_n$) to construct a model D which has copies of each A_n as subsets and is equal to the union of these copies. D can thus be considered a generalized fractal. Generalized, because we don't see how to give these models a geometrical form; they involve objects of higher type which don't lend themselves to computer graphics.

Coherence-space semantics. In 1989, Girard's book [33] introduced the coherence-space semantics, the study of which led to linear logic. The effort to understand and apply linear logic is nowadays absorbing the efforts of many researchers around the world. Still the coherence-space semantics is of independent interest for intuitionistic systems.

PER semantics. Moggi and Hyland [41] showed how to use partial equivalence relations (PER) to define a semantics. The definition of PER is inspired by realizability: what kinds of binary relations can interpret equality? This semantics led to the construction of models for polymorphic types, in particular Girard's System F. Prior to the PER semantics, only term models constructed by Tait's method were known for polymorphic systems.

5 Actual implementation of constructive formal systems

One of the most exciting strands of constructivity in the nineties is the use of constructive formal systems in the "computerization of mathematics". People around the world are trying to construct systems in which mathematics can be done on a computer. That is, a proof will be represented by an explicit internal data object. Such proofs may be constructed interactively by a human user, or automatically by a theorem-prover, or, in the future, interactlvely by a user with some assistance from the computer. I will describe some of the projects that are working in this direction.[5]

The most extensive implementation of a constructive system is Nuprl, developed under the direction of R. Constable at Cornell [17] Nuprl is based on

[4]The Hausdorff distance is a measure of the size of the set-theoretic difference of two sets A and B, namely the supremum over $x \in A$ of the infimum over $y \in B$ of $d(x, y)$.

[5]I have already mentioned the pioneering effort AUTOMATH. I was able to experiment with AUTOMATH in 1983, but I believe AUTOMATH is history now, in the sense that it is no longer running on any computer.

a constructive type theory similar to those of Martin-Löf, but including some facilities for subtyping. The PRL stands for Proof Refinement Logic. This essentially means that proofs are constructed from the goal backwards. In the traditional way of writing proofs, that would be from the bottom up. (In Nuprl, goals are written at the top and the derivations on which they depend are written below and indented.) Nuprl includes an interactive proof editor. This means that the user constructs a proof interactively, getting one line at a time accepted by Nuprl. The user can define new rules of inference in the ML programming language; these are called "tactics".

Nuprl has been used by many graduate students and researchers at Cornell, and is now installed at several other sites around the world. Numerous case studies have been made of formalizing proofs in Nuprl. One may point to D. Howe's formalization of the proof of Girard's paradox (the fact that you can't have a type of all types in Martin-Löf's theories).

A few years ago I wrote a review of Nuprl in the JSL [9], in which I wrote that the acid test of such systems is whether they are used by people other than the creators and their students. Nuprl is passing this test nowadays: it is being used for practical purposes, in hardware verification. Electrical engineers both in Cornell and in Leeds are using Nuprl to construct formal proofs of the correctness of circuit designs for chips.

Feferman's systems formed the basis for Hayashi's program PX [36]. PX, which stands for proof extraction, permits one to construct a formal proof, and then automatically extract an algorithm from it, by realizability. The algorithm comes out in LISP and can be executed. To make the algorithms come out in LISP, Hayashi modified Feferman's theories so that they are based directly on LISP, rather than on λ-calculus. The main example given in [36] is a decision procedure for propositional calculus.

In Sendai, Japan, M. Sato has developed a system called RPT [56], which is based on Aczel's idea of Frege structures, or more precisely on a formal theory of Frege structures similar to that set out in the last chapter of [4]. The advantage of this approach is that proofs are "first class objects" which can be directly manipulated within the theory. With the help of Kameyama, RPT has been implemented on the computer, and Kameyama has used the implementation to check a proof of the Church-Rosser theorem for RPT's reduction rules.

ICOT developed a theorem-prover MGTP, and associated with it a proof-extractor Papyrus. In September, 1991, I saw a demonstration in which MGTP checked a constructive proof of the Chinese remainder theorem, and Papyrus extracted an executable algorithm from the proof.

Rod Burstall and Randy Pollack have built a proof-checker they call Lego, at Edinburgh. In Lego, they have defined real numbers via Cauchy sequences, and they have proved that metric spaces have a completion, thus making a good start on the computerization of the first chapters of Bishop's book.

Isabelle [51] is a program which allows its user to specify a type theory, and then to check proofs in it.

Dale Miller [49] has designed a programming language called λ-Prolog, which is based on intuitionistic type theory. Just as Prolog works with the Horn fragment of first-order logic, λ-Prolog works with the Horn fragment (actually a somewhat more general fragment) of higher-order logic. The key to this enterprise is the use of Huet's λ-unification algorithm [39] in place of Robinson's first-order unification. λ-Prolog has been used by Amy Felty [29] as a means of implementing theorem-provers and proof-checkers for other constructive logics.

6 Computer Algebra as Constructive Mathematics

Since about 1970, the subject of Computer Algebra, which includes computer calculus, has grown by leaps and bounds. One wishing an introduction to the subject should begin with [42] and [19]. The project being undertaken is nothing less than the constructivization of algebra. Algorithms were provided for the fundamental operations such as polynomial greatest common divisor, factoring polynomials, and integrating elementary functions. Algorithms for dealing with algebraic numbers have also been studied. Berlekamp's factoring algorithm for polynomials introduced the use of p-adic methods based on Hensel's lemma, which were soon applied to other algebraic problems, for example fast polynomial gcd computations.

The simplification of expressions involving radicals led to difficult problems, which were solved recently (in theory at least) by S. Landau [47]. Landau's methods have have not yet been implemented, so it is still easy to find expressions with radicals that are not properly simplified by popular symbolic computation programs. Similarly, the implementation of the Risch integration algorithm led to difficult algebraic questions. I believe the full Risch algorithm has been implemented for the first time in *Mathematica* version 2.0. The Gröbner basis algorithm discovered by Buchberger [12], [13] has been fundamental for dealing with polynomials of several variables. Under the direction of Buchberger, the group at RISC-Linz has been studying other algorithms for solving several polynomial equations as well, notably the method of characteristic sets and the method of polynomial remainder sequences.

It is interesting and important to compare the concerns of the computer algebraists with those of the followers of Bishop. There is a great deal of overlap: both want theorems to have computational meaning. But the focus of the computer algebraists is on the algorithms themselves. The Bishop school's focus is on the theorems and proofs. The algorithms are left implicit in the proofs. The Bishop school is not concerned with the efficiency of the algorithms. Kronecker's factorization algorithm for polynomials is good enough for them, because it shows that in principle polynomials can be factored. At about the same time as Davenport's book on computer algebra appeared, Richman,

Ray, and Mines published a Bishop-style treatment of algebra. The main efforts of this book are to deal with the dependence of the algebraic structure on parameters; there are many counterexamples of the form $R(a)$, where a is a number (parameter) which may or may not have zero imaginary part; if it does, the field is C, if not it is R. Computer algebra, so far, deals only with specific (concrete) algebraic structures; in that respect it is not yet "modern algebra". Richman, Ray, and Mines, on the other hand, face and solve the difficulties of a constructive formulation of modern algebra, but without concern for the implementability or usability of the algorithms.

The computer algebra systems, such as *Mathematica*, Maple, and MACSYMA, are so far calculators only, rather than implementations of coherent formal systems for mathematics. Indeed, it is easy to derive contradictions in these systems. For example, in MACSYMA, if we set $a = 0$, and then divide both sides of the equation by a, using the command $\%/a$, we will get $1 = 0$, because MACSYMA thinks $a/a = 1$ and $0/a = 0$. Further examples are given in [5] and [62]. These are not simply "bugs" that could be easily removed. The underlying difficulty is that symbolic operations often have pre-conditions, or side conditions, that must be met before they can be applied. Keeping track of these side conditions requires a full-blown logical apparatus. In essence each line of computation is the succedent of a Gentzen sequent, and the antecedent lists the conditions or assumptions on which it depends. Present-day symbolic computation systems track only the succedent, leaving the logic and the judgement of the correctness of the result to the user.

The next step is the construction of computerized systems incorporating serious mathematical algorithms, like MACSYMA, but built on a solid logical foundation, like Nuprl. This is one of the tasks of the next decade. I have made a start in this direction by building a system adequate for instruction in algebra, trigonometry, and calculus [5], [6], [7]. It is limited in both its logical and computational features to what is required in those courses, but still, in integrating computational and logical features, it goes beyond any other system.

7 Spread of influence of computational viewpoint in mathematics

Everywhere today one can see the influence of the computer on mathematics. From the computer-assisted proof of the four-color theorem, to the ubiquituous graphic images of fractals, one can see that the feeling is becoming more and more widespread that a problem isn't solved until you can compute the answer. The shift in philosophy that Bishop hoped his book would bring about, is occurring today due to the influence of the computer. The proceedings of the *Computers and Mathematics* conferences show that the computer is being used

in all branches of mathematics. There is, for example, a program *SnapPea* that is used by topologists to help them understand 3-manifolds. Some mathematicians are turning their attention to issues of constructivity and computability. For example, Smale, Blum and Shub have worked on definitions of computability of real and complex functions [59], and Smale has also taken up complexity issues in linear programming [60], and the speed and domain of convergence of Newton's method [61]. Many, perhaps most, mathematics departments have at least one person conversant with some computer algebra system. Perhaps it is safe to say that nobody today publishes an existence theorem without at least thinking about how to compute the solution.

We still have a long way to go, though. I asked quite a few mathematicians, including a couple of algebraic geometers, for a good algorithm to graph a polynomial relation $f(x, y) = 0$ over a specified plane rectangle. Apparently the accumulated knowledge of the centuries does not offer us one bit of help with this problem.[6]

8 Constructivity and the Sciences

A recurrent question is the relationship of constructivity to science, or philosophy of science. I put forward the view that the interesting questions in this area have thus far not been answered. For example, one of the most fundamental questions was first formulated by Kreisel more than thirty years ago. Given the initial positions x and velocities v of three Newtonian bodies of unit mass (say spheres of radius 1, or idealized point particles if you prefer; and for simplicity suppose the initial conditions to be given by rational numbers). Define $f(x, v)$ to be 1 if the bodies eventually collide, and 0 if not, when the system evolves under the laws of Newtonian gravity and mechanics. Is f computable?

Pour-El and Richards [53] have shown that the wave equation has non-computable solutions for some computable initial data. This result depends on the fact that the wave equation converts wiggles in the initial data's derivative to wiggles in the function itself at a later time. To get noncomputable data at time 1, you must start with noncomputable derivatives at time 0. (See [4], pp. 80-81 for details; see also Kreisel's review of Pour-El and Richards in [45]. Those who would attach philosophical signifance to the results must then argue that computable data with non-computable derivatives are physically realizable, but non-computable data are not physically realizable. Contemplation of the issues raised by this question, particularly the question of what it means for a function to be physically realizable, leads on the one hand to the philosophical morass of discreteness versus continuity, and on the other hand to the mysteries of quantum mechanics and measurement.

[6] The difficulty is that we do not know how many connected components the graph will have. Although we can find a zero and trace out the curve containing that zero, how do we know whether or when we have found all the connected components?

There does seem to be an open technical question in this area: do the Maxwell equations behave similarly to the wave equation in this respect?

In the foundations of quantum mechanics, there is a formal theory of operators on Hilbert spaces. Some of the fundamental theorems of this theory are non-constructive on the face of the matter, although they do have constructive versions. Again, it is difficult to decide the philosophical significance of these observations.

9 A Philosophical Thesis

The mathematician Truesdell predicted that the computer will influence mathematics as much as the telescope influenced astronomy or the microscope influence biology. I believe that the philosophy of constructive mathematics must also respond. A philosophy resting on the concept of computational meaning cannot ignore the changes in our ability to perform computations. If these changes were not so dramatic, one could argue that the *concept* of computational meaning is not affected by them. But the changes in our ability to compute *have* been dramatic, so much so that our concepts about computation have also been affected.

Brouwer, Heyting, and Bishop were united in their feeling that the efficiency of computation wasn't the point. What mattered was whether a solution could be computed in principle. That may have been an appropriate viewpoint, when we were limited to pencil-and-paper computations. Now that we have computers, it has to be rethought. It does matter, when we want to prove that polynomials can be factored, whether we take Kronecker's proof with its unusably inefficient algorithm, or Berlekamp's proof that actually enables us to factor polynomials of interest. Now, one may say that this is a matter of practical, but not philosophical, interest. But I will make the matter of efficiency into a philosophical point. I put forward the following thesis:

> *Constructivity is an attribute of proofs, like elegance or beauty, that can be possessed in greater or lesser degree–it is not an all-or-nothing quantity like correctness.*

For example, Berlekamp's proof is more constructive than Kronecker's, because the algorithm involved is more efficient. Even between two proofs, neither one of which is constructive in the traditional sense, one may be able to say that one is "more constructive" than the other. For example, König's lemma for binary trees is more constructive than Zermelo's proof that the reals can be well-ordered.

In more technical terms, one can classify proofs according to the complexity of the algorithms which implicit in the proofs. (Various technical notions such as realizability could be used to make this precise.) Traditionally, constructivity has drawn the line at Turing computability, considering everything on one side

constructive and everything on the other side beyond the pale. Until the advent of the computer, only a handful of specialists cared about this distinction. Now, practicing mathematicians are interested, but they are drawing the line lower: they are interested in proofs that can be programmed and run.

Feferman has long proposed a technical study on these lines: one should go through Bishop's book and see how much of it can be done "feasibly", e.g. with polynomial-time algorithms. Steps in this direction were already taken by Ko and Friedman [43], [30], [44]. There is still more work to be done.

I will now make a direct argument for the thesis. "Constructivity" as used in the title of this paper refers to this: when one has solved a problem constructively, one can exhibit the solution; or if it depends on some data, one can show how to compute the solution from the data. The traditional notion of constructivity arose by identifying computability with Turing computability. But everybody who has played with computers knows this is unrealistic; many Turing computable functions can't actually be run. Morever, we are able to understand notions of relative computability, ordinal computability, etc., that go beyond Turing computability. Once we admit that these distinctions are possible and meaningful, the general principle that *meaningful distinctions should be made* implies that the notion of constructivity must fragment into many related notions.

Support for the above philosophical thesis can be drawn indirectly from technical research in proof theory conducted over the last forty years. Proof-theorists have constructed a hierarchy of formal systems, corresponding more or less to increasing non-constructivity. In the sixties Feferman [20], [21] formulated systems corresponding to the philosophical idea of *predicativity*, in which (roughly speaking) sets defined using quantifiers over power sets not considered legitimate. The main results of Feferman's research imply that this restriction on set definition principles translates to a restriction on the complexity of functions definable; that is, if you can solve a problem predicatively, you can "exhibit" the solution as a function of the data, where the function is defined by ordinal recursion up to a certain ordinal Γ_0. This program of research quite possibly inspired Friedman to develop his idea of "reverse mathematics", which was further developed by Simpson. (See [57], or, when it becomes available, [58]). In reverse mathematics, basic theorems such as Bolzano-Weierstrass are shown to be equivalent to axiom schemata considered in proof theory. Thus mathematics as a whole is classified into strata according to the degree of constructivity involved. In the last decade, these ideas have been extended in the other direction, to create a sub-constructive hierarchy of subtheories of Heyting's arithmetic. These theories have been defined and studied by Buss and Takeuchi, and shown to have natural connections with the hierarchies of functions considered in complexity theory. Thus Heyting's arithmetic HA, and with it the traditional notion of constructivity, is just one point on a spectrum. Of course, one may take the view that the other points have only technical significance; but the view advanced here is rather the opposite, that while HA is certainly an

interesting point on this spectrum, the others also have philosophical meaning.

10 Postscript

At the moment, there is too little communication between the different groups discussed in this paper. The computer algebraists have not read Bishop. The Bishop followers have not read Davenport. Neither one knows much about Nuprl. Systems like Nuprl do not incorporate algorithms from computer algebra, which would greatly increase their power to formalize real mathematics. Computer algebra systems work entirely without any logical formalism, which means that they are subject to error, as they can't check the preconditions of algebraic operations. If this paper helps some of these people to get interested in what some of the others are doing, I will have achieved my aim in writing it.

References

[1] P. America, J. Rutten, Solving reflexive domain equations in a category of complete metric spaces, Centre for Mathematics and Computer Science, Report CS-R8079, February 1987.

[2] J. W. de Bakker, J. Zucker, Processes and the denotational semantics of concurrency, *Information and Control* **54** (1082) 70-120

[3] H. Barendregt, *The Lambda Calculus: Its Syntax and Semantics*, North-Holland (1981).

[4] M. Beeson, *Foundations of Constructive Mathematics*, Springer-Verlag (1985).

[5] M. Beeson, Logic and computation in *Mathpert*: an expert system for learning mathematics, in: Kaltofen, E., and Watt, S. M., *Computers and Mathematics*, pp. 202-214, Springer-Verlag (1989).

[6] M. Beeson, *Mathpert*: a computerized environment for learning algebra, trig, and calculus, *J. Artificial Intelligence and Education* **2** (1990), pp. 1-11.

[7] M. Beeson, *Mathpert*: Computer Support for Learning Algebra, Trig, and Calculus, in the proceedings of the conference on *Logic Programming and Automated Reasoning, St. Petersburg, Russia, July, 1992*, to appear in Springer Lecture Notes in Computer Science.

[8] M. Beeson, Towards a computation system based on set theory, *Theoretical Computer Science* **60**: 297–340 (1988).

[9] M. Beeson, Review of [17], *J. Symbolic Logic* **55** (1990) p. 1299.

[10] M. Beeson, Some applications of Gentzen's proof theory in automated deduction, in: Schroeder-Heister, P., *Extension of Logic Programming*, Springer Lecture Notes in Computer Science **475**, pp. 101-156, Springer-Verlag (1991)

[11] E. Bishop, *Foundations of Constructive Analysis*, McGraw-Hill, New York (1967). The original edition is out of print. There is a a revised and extended second edition, E. Bishop and D. Bridges, *Constructive Analysis*, Springer-Verlag (1985).

[12] Buchberger, B., History and basic features of the critical-pair completion procedure, *J. Symbolic Computation* **3**: 3-38 (1987)

[13] Buchberger, B., Gröbner bases: an algorithmic method in polynomial ideal theory, pp. 184-232 in: Bose (ed.), *Multidimensional Systems Theory*, Reidel, Dordrecht (1985).

[14] N. De Bruijn, A survey of the project AUTOMATH, in: J. R. Hindley and J. P. Seldin (eds.), *To H. B. Curry: Combinatory Logic, Lambda Calculus, and Formalism*, Academic Press (1980).

[15] L. Cardelli, S. Martini, J. Mitchell, A. Scedrov, An extension of system F with subtyping, in: Ito, T., and Meyer, A.R, (eds), *Theoretical Aspects of Computer Software*, pp. 750-770, Springer Lecture Notes in Computer Science **526** (1991).

[16] P. Curien and G. Ghelli, Subtyping + extensionality: confluence of $\beta\eta$ top reduction in $F_{\leq}$, in: T. Ito and A. R. Meyer (eds), *Theoretical Aspects of Computer Software*, pp. 731-749, Springer Lecture Notes in Computer Science **526** (1991).

[17] R. Constable, *et. al.*, *Implementing Mathematics with the Nuprl Proof Development System*, Prentice-Hall, Englewood Cliffs, New Jersey (1986).

[18] T. Coquand and G. Huet, The calculus of constructions, *Information and Computation* **76**: 95-120 (1988).

[19] J. Davenport, Y. Siret, and E. Tournier, *Computer Algebra*, Academic Press, London/ San Diego (1988).

[20] S. Feferman, Systems of predicative analysis, *Journal of Symbolic Logic* **29** (1964) 1-30.

[21] S. Feferman, Systems of predicative analysis II: representations of ordinals, *Journal of Symbolic Logic* **33** (1968) 193-220.

[22] S. Feferman, Constructive theories of functions and classes, pp. 159-224 in: M. Boffa, D. van Dalen, and K. McAloon (eds.), *Logic Colloquium '78: Proceedings of the Logic Colloquium at Mons*, North-Holland, Amsterdam (1979).

[23] S. Feferman, Logics for termination and correctness of functional programs, in: Y. Moschovakis (ed.), *Logic and Computer Science: Proc. of the '89 MSRI Conference*, Springer (1989).

[24] S. Feferman, Polymorphic type lambda-calculi in a type-free axiomatic framework, manuscript, May, 1988.

[25] S. Feferman, Logics for termination and correctness of functional programs, II: Logics of strength PRA, to appear in Proceedings of Leeds Proof Theory '90 Conference

[26] S. Feferman, A new approach to abstract data types,I. Informal development, Math. Structures in Computer Science 2 (1992) 193-229.

[27] S. Feferman, A new approach to abstract data types, II. Computation on ADTs as ordinary computation, in: E. Boerger *et. al.* (eds.) *Computer Science Logic*, pp. 79-95. Springer Lecture Notes in Computer Science **626** (1992).

[28] J. Farmer *et. al.*, MITRE Technical Reports.

[29] A. Felty and D. Miller, Specifying theorem provers in a higher-order programming language, in: *Proc. of the Ninth International Conference on Automated Deduction, Argonne, Ill., May 1988*, pp. 61-80, Lecture Notes in Computer Science **310**, Springer-Verlag, Berlin/ Heidelberg/ New York (1988).

[30] H. Friedman, The computational complexity of maximization and integration, *Advances in Mathematics* **53** (1984) 80-98.

[31] M. Fourman and D. Scott, Sheaves and Logic, in Fourman, Mulvey, and Scott (eds.), *Applications of Sheaves, Proceedings, Durham 1977*, Lecture Notes in Mathematics **753**, Springer-Verlag.

[32] M. Fujita, R. Hasegawa and Y. Shirai, Program Synthesis by A Model Generation Theorem Prover, ICOT-TR-629 (1991).

[33] J. Girard, Y. Lafont, and P. Taylor, *Proofs and Types*, Cambridge University Press (1989).

[34] R. Goldblatt, *Topoi: The Categorial Analysis of Logic*, North-Holland, Amsterdam (1984).

[35] R. Hasegawa, H. Fujita and M. Fujita, A Parallel Theorem Prover in KL1 and Its Application to Program Synthesis, Italy-Japan-Sweden Workshop, ICOT-TR-588 (1990).

[36] S. Hayashi and H. Nakano, *PX: A Computational Logic*, MIT Press, Cambridge, Mass. (1988).

[37] D. Howe, *Automated Deduction in Constructive Type Theory*, Birkhäuser-Boston, New York (1991).

[38] B. Kutzler, On the application of Buchberger's algorithm to automated geometry theorem proving, *J. Symbolic Computation* **2** (1986) 389-397.

[39] G. Huet, A unification algorithm for type lambda-calculus, *Theoretical Computer Science* **1**: 27-57 (1975).

[40] M. Hyland, The effective topos, in: *The L. E. J. Brouwer Centenary Symposium*, pp. 165-216, North-Holland, Amsterdam (1982).

[41] M. Hyland, E.P. Robinson, and G. Rosolini, The discrete objects in the effective topos, *Proc. Lond. Math. Soc.* (3) **60** (1990) 1-36.

[42] D. Knuth, *Seminumerical Algorithms, Art of Computer Programming* **2**, Addison-Wesley.

[43] K. Ko and H. Friedman, Computational complexity of real functions, *Theoretical Computer Science* **20** (1982) 323-352.

[44] K. Ko, *Feasible Analysis*, Birkauser, Boston (1991).

[45] G. Kreisel, Review of [53], *J. Symbolic Logic* **47** (1982) 900-902.

[46] J. Lambek and P. Scott, *Introduction to higher order categorical logic*, Cambridge University Press (1986).

[47] S. Landau, Simplifying nested radicals, *Journal of Symbolic Computation*, January 1992.

[48] P. Martin-Löf, *Intuitionistic Type Theory*, Biblipolis, Naples (1984).

[49] D. Miller, and G. Nadathur, An overview of λ Prolog, pp. 810-827 in: Kowalski, R., and Bowen, K. (eds.), *Fifth International Conference Symposium on Logic Programming*, MIT Press, Cambridge, Mass. (1988)

[50] Myhill, J., Constructive set theory, *J. Symbolic Logic* **40** (1975) 347-382.

[51] L. Paulson, Isabelle: The next 700 theorem provers, in: P. Odifreddi (ed.), *Logic and Computer Science*, pp. 361-385, Academic Press (1990).

[52] W. Phoa, From term models to domains, in: Ito, T., and Meyer, A.R, (eds), *Theoretical Aspects of Computer Software*, pp. 88-111, Springer Lecture Notes in Computer Science **526** (1991).

[53] M. Pour-El, and I. Richards, The wave equation with computable initial data such that its unique solution is not computable, *Advances in Mathematics* **39** (1981) 215-239.

[54] J. C. Reynolds, Towards a theory of type structure, in: *Paris Colloquium on Programming*, Springer Lecture Notes in Computer Science **19**, pp. 408-425 (1974).

[55] N. Shankar, Proof Search in the Intuitionistic Sequent Calculus, in: D. Kapur, *Proceedings of CADE-11*, Lecture Notes in Computer Science (1992).

[56] M. Sato, Adding proof objects and inductive definition mechanisms to Frege structures, in: Ito, T., and Meyer, A.R, (eds), *Theoretical Aspects of Computer Software*, pp. 53-87, Springer Lecture Notes in Computer Science **526** (1991).

[57] S. Simpson, Appendix to the second edition of *Proof Theory*, by G. Takeuti, North-Holland, Amsterdam (1987).

[58] S. Simpson, *Subsystems of Second Order Arithmetic*, to appear.

[59] S. Smale, L. Blum, and M. Shub, On a theory of computation and complexity over the real numbers: NP completeness, recursive functions, and universal machines, *Bulletin of the A.M.S.* **21** (1989) 1-46.

[60] S. Smale, On the average number of steps of the simplex method of linear programming, *Math. Programming* **27** (1983) 241-262.

[61] S. Smale, Algorithms for solving equations, *Proc. International Congress of Mathematicians, Berkeley, 1986*, A.M.S., Providence, R.I. (1987).

[62] R. Stoutemeyer, Crimes and misdemeanors in the computer algebra trade, *Notices of the A.M.S.* **38**(7) 779-785, September 1991.

[63] A. S. Troelstra, *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, Springer Lecture Notes in Mathematics **754** (1973).

[64] A. S. Troelstra and D. van Dalen, *Constructivism in Mathematics: An Introduction*, two volumes, North-Holland, Amsterdam (1988).

[65] K. Wagner, work in progress at RISC-Linz, Austria.