Samsara*

John N. Crossley

Faculty of Information Technology,
Monash University, Clayton, Victoria, Australia 3800
John.Crossley@infotech.monash.edu.au

Abstract. In order to answer the question of what logic may be like in this twenty-first century we examine the history of logic. One thing that we see is a recurrence of ideas but the ideas change form, sometimes quite dramatically. We also see a simple idea getting developed so much that it becomes very complicated, or at least the source of very complicated ideas – and then disappears, to re-emerge, perhaps, as a new idea.

Another aspect is the way the rôle of logic has changed. Logic was at one time the queen of the (mathematical) sciences. Now she is definitely not, but she has certainly become the handmaid of computer science, playing many roles.

To assist our enquiry we address four questions.

1. What logics do we need?
2. What are logical systems and what should they be?
3. What is a proof? and briefly,
4. What foundations do we need?

We take an historical approach rather than a highly technical one.

1 Introduction

Mathematical logic[1] was unknown before about 1850 (but see the remarks on Leibniz below). It was very lively around 1900–1930 and, in the author's opinion, reached its zenith with the 1957 Cornell AMS Summer Institute on Symbolic Logic [74] and Paul Cohen's proof of the independence of the axiom of choice [14, 15] in 1963.

In the second half of the twentieth century, logic may no longer have been the queen of the (mathematical) sciences, but she has certainly become the handmaid of computer science, playing many roles. Logic has become essential in laying a theoretical foundation for computing.

---

* "The endless cycle of death and rebirth to which life in the material world is bound." (OED)

[1] Here the phrase "Mathematical logic" refers to the pure discipline of mathematical logic.

1.1 The structure of the paper

We first give an example, in Section 2, of the way that a part of a discipline, that has been well studied, may go into decline but subsequently be resurrected by new, transforming ideas.

In Section 3 we consider the development of logic, particularly in the last hundred years. Then we go on, in the remaining parts of Section 3, to consider in some detail, how the move to intuitionist logic led to the extraction of programs from proofs. The author does not pretend that this is the most important part of logic. It is, of course, an area with which he is familiar. Very similar remarks could be made about the development of another part of the author's work, that on Recursive Equivalence Types. There one can also see how the simple ideas of Dekker and Myhill [25] were developed dramatically by Anil Nerode [65] and subsequently extended to a whole set of different algebraic structures in [20]. Interest has now substantially waned in this area with only a very few people still publishing in the area according to Mathematical Reviews. The message we are trying to convey is that one should look at how logic has developed in the past in order to determine how it may develop in the future.

Near the end of this section, in Section 3.5 we see how the combination of techniques from computer science and logic can be used together to develop new (kinds of) logics. In Section 3.7 we compare the two standard ways of obtaining "correct" programs.[2]

In Section 4 we briefly consider some other logics, or parts of logic, that have yielded different kinds of systems in which to do mathematics. Next, in Section 5 we consider a few aspects of the notion of proof. Perhaps it is slightly ironical that we should end with foundations. However, the present author finds it hard to accept that the particular foundations of mathematics that we use would make much difference to the logic that we presently do. This is despite his having lived through a period when intuitionist logic was transformed from a rather odd and idiosyncratic logic to one which is fundamental to computer science.

Along the way we note a number of conclusions. These are not ends, but starting points for new discoveries (or inventions) or even of new modes of discovery and invention.

2 An example of a process

Let us first of all not consider logic, but linear algebra (as it is now called).

In the nineteenth century the study of canonical forms of matrices became more and more complicated. Muir's book [63] is regarded as leading to the death of determinants. As Carl C. Cowen put it in [17]:

> [Kenneth O.] May [in the MAA film *Who Killed Determinants?* in the 1960s] documented how determinants flourished in the 19th century with

[2] We use the word "correct" in the sense that the program meets its specification.

> its connections to the study of invariants and how the study of determinants developed into the linear algebra we know today, where determinants are far from central. Linear algebra did not really come to be recognized as a subject until the 1930s. . . . Historian Jean-Luc Dorier [28] regards Paul Halmos' book [37] *Finite Dimensional Vector Spaces*, first published in 1942, as the first book about linear algebra written for undergraduates.

The study of vector spaces did indeed transform the subject. As an undergraduate in the late 1950s, determinants and matrices were familiar to me, but Halmos's book opened up a new world. It transformed the subject from a very complicated, indeed arcane, one into one where what one wrote down was much simpler. With this dramatic increase in simplicity[3] – both typographical and conceptual – the subject became immensely more powerful. How could one have studied Hilbert spaces without vector spaces?

3 What logics do we need?

Subsidiary to the question in the title of this section we shall also ask:

> How do we find/invent these logics?

In order to attack these questions let us ask:

> How has logic developed? How will it develop?

I believe I am following the ancient Greek philosopher Aristotle when I say that logic is the (correct) rearranging of facts to find the information that we want.

Logic has two aspects: formal and informal. In a sense logic belongs to everyone although we often accuse others of being illogical. Informal logic exists whenever we have a language. In particular Indian logic has been known for a very long time (see, for example, [62, 61]).

Formal (often called, "mathematical") logic has its origins in ancient Greece in the West with Aristotle.[4] Mathematical logic has two sides: syntax and semantics. Syntax is how we say things; semantics is what we mean. Later I will suggest that we should perhaps add at least a third element to these two (see Section 3.6 below).

By looking at the way that we behave and the way the world behaves, Aristotle was able to elicit some basic laws. His style of categorizing logic led to the notion of the *syllogism*.

In the seventeenth century Leibniz began the axiomatization of sets (and also of real numbers, in a manuscript in the Niedersächsische Landesbibliothek,

[3] Perhaps it might be better to say "dramatic move to the basics", in the sense of the basic principles of vector space theory being more fundamental.

[4] There is an interesting, and different, account of this history in [2].

now the Gottfried Wilhelm Leibniz Bibliothek). The latter was complicated and became, as far as I can ascertain, essentially lost until the twentieth century.

In the nineteenth century Boole developed his laws of thought (see [6]). It was natural that he should refer to "thought" since, at that time, logic was in the domain of psychology. Then Frege developed his *Begriffsschrift* [32] whence came Russell and Whitehead's *Principia Mathematica* [77]. In tandem Dedekind's axioms for arithmetic (see [24]) were formalized by Peano (see, e.g., [66]).

Frege developed a relatively small set of concepts and notations which were, apparently,[5] adequate to deal with the whole of logic – and mathematics. Russell – initially – built a seemingly simple system for dealing with all of mathematics and logic. But he wanted to reduce mathematics to logic. He did not succeed.

The system that Russell developed became more complicated because of 1) the axiom of infinity[6] and 2) the axiom of reducibility which surely must be a non-logical axiom.[7]

Thus about the year 1900 there was just "one true logic": classical logic. In such a logic one would expect that everything was clear. Certainly, in that logic, any statement was either true or false: there was the law of the excluded middle, $(A \vee \neg A)$. But how do we check an infinite number of instances? What does it mean to say that there is no largest pair of twin primes, that is to say that there is an end to such pairs such as 5 and 7; 11 and 13 or even 202 289 and 202 291?

On the other hand, saying that there are infinitely many pairs of twin primes does have a clear meaning if we can show that, for every pair, there is a larger pair. In this context compare the way that Euclid IX.20 established that there are infinitely many prime numbers (although he did not phrase it like that) [38]. He gave a method for constructing a larger prime from a given (finite) set of prime numbers.

Because of the above style of questioning, led by Brouwer, a Dutch mathematician, indeed a topologist, "constructive logic" or "intuitionist logic" arose.[8]

For Brouwer, the problem of classical logic is that it is not evident that a mathematical proof actually gives you the way of performing the necessary construction. However, that is perhaps the wrong way to look at it. Brouwer was concerned only with constructing mathematical objects that were claimed to exist. He did not like mathematical logic and did not consider it relevant. However, when his approach was formalized, as it was by Heyting in 1930 (see [42]), the

[5] Frege's concepts were adequate but he did not get all the axioms necessary for a complete system of first order logic.

[6] See also below Section 5.1.

[7] The axiom of reducibility was used by Russell to avoid the paradoxes of set theory. Its *ad hoc* nature prevented giving a good philosophical justification for it.

[8] Many people have regarded intuitionist logic as being restrictive because it eschews the use of the law of the excluded middle, but this was not Brouwer's motivation. Further, intuitionist logic actually *includes* constructive logic in that there is a uniform translation (the "negative translation" whereby any formula, A, of classical logic can be translated into a formula, $A^{\neg}$, of intuitionistic logic) which is provable in intuitionist logic if, and only if, A is provable in classical logic. (See Gödel [36].)

details are buried inside the proof. Nowadays one might say that they are buried in the way that algorithms are buried inside computer programs.

The actual mechanism of providing proofs in the Russell and Whitehead system was cumbrous, although the techniques could be learned. For example, if one wants to prove the formula $(p \supset p)$ (read "p implies p") then a simple analysis of the axioms reveals that there are very limited possibilities for producing a proof and these are quickly exhausted. (Of course, ingenuity and practice make the task of finding such proofs easier, but Russell, in his autobiography [72], revealed that it had taken him a very great deal of time and effort to work out the formal proofs.)

So proving theorems in the way dictated by formal logic, that is to say, writing out strings of symbols and applying formal (mechanical) rules to obtain new formulae, became very tedious.

Here we already see Russell's system mimicking Aristotle: for the latter always had two premisses in his syllogisms. From these premisses he derived the conclusion.

What happened next? People began to say "It can be done," rather than going to the trouble of writing out the formal proof. This was nicely expressed, though not for our specific context, by Hoare [43], p.578:

> Once a powerful set of supplementary rules has been developed, a "formal proof" reduces to little more than an informal indication of how a formal proof could be constructed.

Indeed, the attitude among mathematicians, the present author included, was often that the approach, of simply showing that a proof existed, was accepted as sufficient. (But see below Section 5.1.)

However the way was already open (though it was not known immediately) for Gödel's incompleteness theorem (see [35, 58]). This theorem showed that the apparatus of formal first order logic was not sufficient to do all that mathematicians would wish to do.

The first major consequence of Gödel's incompleteness theorem is that Peano's axioms, when formalized in *first* order logic, are not sufficient to characterize the natural numbers.[9] But second order logic was unacceptable and has only now begun to resurface (see below section 4.1).

Other methods for establishing that a proof exists were developed, based on semantics. Most striking amongst these are those arising from model theory (see [3, 44, 9]).

Thus the first concern of mathematical logic became the relation between syntax and semantics.[10] Syntax is how we say things; semantics is what we mean.

---

[9] Of course, in *second* order logic, they are categorical, that is to say, there is only one model up to isomorphism.

[10] It could perhaps be argued that this had always been the first concern of logic.

The ultimate result in the positive direction is Gödel's completeness theorem(see [11] or Gödel's original [34]) which was obtained long before model theory was invented (by Alfred Tarski).[11]

Nowadays the formulation of the completeness theorem involves models:

Theorem 1 (Gödel, Henkin). *A set of formulae (of first order logic) is consistent if, and only if, it has a model.*

So one needs the notion of model. Henkin's proof (see [39]) constructs such a model. This involves providing a (syntactic style) "witness" for the x whenever one wants a formula of the form $\exists x A(x)$ to be true in a (formal) model that one is constructing.

Henkin originally came upon this idea of giving names to such witnesses when he was looking at set theory (see his [41]) and he also applied it to the theory of types [40], see below Section 4.1.

All of this work on the semantics of logic had been predicated on the assumption that there was just one kind of logic: "one true logic". The great success of the Frege/Russell initiative had been that it seemed to cover everything.

Brouwer's abrupt departure from the classical world was ultimately to lead to many other kinds of logics. But there were other influences, coming from philosophy: modal logic has a very long history. The modalities of *necessity* and *possibility* also come from Aristotle and were introduced into formal logic by C.I.Lewis (see [55]) in 1932.

However it was not until the work of Saul Kripke [51] that connexions were established between models of modal logics and models of first order logic. Surprisingly, taking many Henkin models, with suitable relations between them, was sufficient to cover the modal case. They even covered the logic of Brouwer's thought [52], although Brouwer did not approve of the formalization of his logic.

Nowadays there are many different logics that all have their value and application. These logics include various kinds of modal logics.

Many of these logics also have completeness theorems analogous to Theorem 1. The proofs of these have similarities with the proof of the completeness of intuitionist logic. Indeed, Kripke proved completeness results for modal logics first [51] and only subsequently used his ideas there to prove the completeness of intuitionist logic. Details of such theorems may be found in [18].

Recently category theory and its connexions with computer science have given a profitable way of generating useful modal systems through the connexion between co-algebras[12] and modal logic (see e.g. [47, 53]).

[11] Tarski's great contribution was to formalize what seems obvious to most students – until they are asked to write it down formally: he gave a formal definition of "model" (see, e.g. [58]).

[12] *Algebras* are characterized by having a domain and functions on that domain. Taking the category-theoretic approach one reverses the arrows to get a co-structure, in this case a co-algebra. In particular this means that co-algebras can be used to characterize the behaviour of machines that change state. The arrows in the co-algebra are the state transitions.

One original aim of Leibniz was to reduce argumentation to calculating with symbols, witness his famous remark: "Let us calculate" in *Ars Inveniendi* (*The Art of Discovery*) of 1685, reprinted in [16]. Many people would say that we are well on the way to that. An extensive discussion of this took place at the recent Royal Society Discussion Meeting on 18-19 October 2004 (see `http://www.royalsoc.ac.uk/event.asp?id=1334`). At this meeting some people seemed to feel that Leibniz's Golden Age had arrived, others that it was unattainable! We shall take up some of the issues below in Section 5

However, there are certainly new styles of logic that have developed in the last fifty years. For example, description logics. As Nardi and Brachman put it in [64]:

> Description Logics in part arose from a need to respond to the inadequacy – the lack of a formal semantic basis – of early semantic networks and frame systems.

Description Logics (see [1]) may be regarded as an adaptation of (first order) logic to databases; databases themselves coming (on the logical side) from the notion of relational system or model.

I believe this is typical of the way that mathematical logic and its practitioners have responded to the needs of computing and computer science. It then follows that

Conclusion 1 *We should expect to see many more logics developing in the twenty-first century.*

3.1 Extracting constructions from proofs

What do we want from the logics we create? If we look at the work of Frege and Russell, I believe it is fair to say that they wished to clarify how mathematics worked or at least to make a fool-proof system for mathematics. For Aristotle, as I said above, logic is the (correct) rearranging of facts to find the information that we want. In present day computer science we are often trying to understand the logic of machines.

The first interpretation we shall consider is the logic of how computers can reliably get results. In this approach one imposes the logic first of all. One proves, for example, that for every x there is a y such that $A(x, y)$. Then one can extract a program from the proof, provided the proof is written in a suitable logic.

When Brouwer's proofs were formalized the information about the constructions he was giving became embedded in the proofs. Therefore, Brouwer's logic: intuitionist logic, is a suitable logic. We give the rules for this logic in Fig. 1. However, if we want to recover the construction we have to do some work.

It was Gentzen [75] in the 1940s who was the first to produce a formal system of logic where it was readily possible to see the information being moved around and, as it turned out, to make it possible to recover information on constructions.

3.2 The Lambda Calculus and the Curry-Howard correspondence

How do we extract the information from a proof in mathematical logic? Curry [23] started, and Bill Howard [46] developed, the basic idea which exploited Gentzen's achievements.

We use the lambda calculus. This was established by Church [10]. In ordinary mathematics if we apply the function $\lambda x.f$ to a then we get $f[a/x]$, which is read "f with a for x". In the lambda calculus however this is *not* the same as the (application) term $\lambda x.fa$, i.e. $\lambda x.f$ applied to a. In particular they are *syntactically* different. We therefore have to introduce the notion of *β-reduction*[13]

$$\lambda x.fa \rhd f[a/x]$$

(Here $\rhd$ is read "reduces to".)

Now note the similarities between $\rightarrow$-introduction, the rule ($\rightarrow$-I), and $\rightarrow$-elimination ($\rightarrow$-E) (in Fig. 1) on the one hand, and λ-introduction and λ-elimination on the other, the β-rule.

Next consider a proof of B from A from which we get a proof[14] of $(A \rightarrow B)$ (by the rule ($\rightarrow$-I)):

$$\frac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{(A \rightarrow B)} \tag{1}$$

and lambda abstraction (which abstracts a function from the process where $a \in A$ gives us $f(a) \in B$): that is $\lambda x.f$. Consider the figure:

$$\frac{\begin{array}{c}a\\ \vdots \\ f[a/x]\end{array}}{\lambda x.f}$$

What is the connexion?

The most obvious thing, I hope, is that the *shapes* are the same! If this is difficulty to see then replace the bottom line by $x \rightarrow f(x)$.

Conclusion 2 *Patterns will arise in formal studies that reflect each other.*

3.3 Proofs as types

Russell, in his Appendix B to [70] introduced the theory of types in an informal setting, and later in a formal setting in [71]. The theory of types was invented

[13] α-reduction refers to the simple renaming of one variable by another (without clashes).

[14] The square brackets indicate that A can be discharged, i.e. is not needed for the proof of B, though it is for the proof of B, of course.

Assume that x, y are individual variables, and that t and t' are individual terms.

$$\frac{}{A \vdash A}\ \text{(Ass-I)}$$

$$\frac{\Delta, A \vdash B}{\Delta \vdash (A \to B)}\ (\to\text{-I}) \qquad \frac{\Delta \vdash A \quad \Delta' \vdash (A \to B)}{\Delta, \Delta' \vdash B}\ (\to\text{-E})$$

$$\frac{\Delta \vdash A}{\Delta \vdash \forall x.A}\ (\forall\text{-I}) \qquad \frac{\Delta \vdash \forall x.A}{\Delta \vdash A[t/x]}\ (\forall\text{-E})$$

x is free in A, not free in Δ

$$\frac{\Delta \vdash A[t'/y]}{\Delta \vdash \exists y.A}\ (\exists\text{-I}) \qquad \frac{\Delta_1 \vdash \exists y.A \quad \Delta_2, A[x/y] \vdash C}{\Delta_1, \Delta_2 \vdash C}\ (\exists\text{-E})$$

where x is not free in C

$$\frac{\Delta \vdash A \quad \Delta' \vdash B}{\Delta, \Delta' \vdash (A \wedge B)}\ (\wedge\text{-I})$$

$$\frac{\Delta \vdash (A_1 \wedge A_2)}{\Delta \vdash A_1}\ (\wedge\text{-E}_1) \qquad \frac{\Delta \vdash (A_1 \wedge A_2)}{\Delta \vdash A_2}\ (\wedge\text{-E}_2)$$

$$\frac{\Delta \vdash A_1}{\Delta \vdash (A_1 \vee A_2)}\ (\vee\text{-I}_1) \qquad \frac{\Delta \vdash A_2}{\Delta \vdash (A_1 \vee A_2)}\ (\vee\text{-I}_2)$$

$$\frac{\Delta \vdash A \vee B \quad \Delta_1, A \vdash C \quad \Delta_2, B \vdash C}{\Delta_1, \Delta_2, \Delta \vdash C}\ (\vee\text{-E})$$

$$\frac{\Delta \vdash \bot}{\Delta \vdash A}\ (\bot\text{-E})$$

$A[a/x]$ is read "A with a for x" and denotes the formula A with a substituted for x.

Fig. 1. The basic rules of intuitionistic logic.

by Russell (see [77]) to resolve the difficulties causes by Russell's paradox.[15] The *typed* lambda calculus that we shall consider, that is to say lambda calculus with each term having a *type* assigned to it, can be regarded as the amalgam of two systems: logic, or more precisely, systems of predicate calculus, and the lambda calculus.

Originally types were built up from basic types by one simple operation. The original idea was that they formed a classification of sets. Sets at a "higher" type contained (in a sense), or reflected, sets of lower types. In Howard's system the types were identified with formulae of (propositional) logic.

A special kind of typed lambda calculus involves taking formulae of logic as the types. Now this is a strange idea to accept but it is easier to work with if one thinks of a type (i.e. formula) as the *set of proofs* of that formula. Instead,

[15] The set of sets which are not members of themselves yields a contradiction.

therefore, of variables, we use typed variables of the form $a : A$ where A is the type. Later one can simply treat the types as labels (see footnote 21 below).

The rule of *modus ponens* ($\rightarrow$-I) of Fig. 1 then becomes:

$$\frac{a : A \quad g : (A \rightarrow B)}{(ga) : B} \tag{2}$$

where we have changed f to g to avoid confusion in what follows.

If we had a proof of B from A then we would get an expression $\lambda x : A.f : B$ by the rule of ($\rightarrow$-I) and this has type $(A \rightarrow B)$. If the g in the expression (2) is actually of the form $(\lambda x : A.f : B) : (A \rightarrow B)$, then we get

$$\frac{a : A \qquad (\lambda x : a.f : B) : (A \rightarrow B)}{((\lambda x : A.f : B) : (A \rightarrow B))a : A) : B}$$

which is somewhat hard to read. However the bottom line has the formula B as its type, and the expression reduces to

$$f : B[a : A/x : A] \tag{3}$$

where the substitution of $a : A$ for $x : A$ takes place throughout the term $f : B$.

If we translate this back into proofs it means that the corresponding proofs look as follows. On the one hand we have the complicated proof:

$$\dfrac{\begin{array}{cc} & [A] \\ & \vdots \\ \vdots & B \\ A & \overline{(A \rightarrow B)} \end{array}}{B} \tag{4}$$

and on the other hand, by putting the proof of A from the left on top of the proof of B (from the hypothesis A), and *not* introducing the $\rightarrow$, we no longer need the hypothesis $[A]$ in the proof on the right in order to get a proof of B.

That is to say, we *reduce* the proof in (4) to a simple proof of B of the form

$$\begin{array}{c} \vdots \\ A \\ \vdots \\ B \end{array}$$

This corresponds in the lambda calculus to the reduction[16] that resulted in (3). So we have a direct correspondence between proofs and terms of our typed lambda calculus. This is called the *Curry-Howard correspondence*.[17]

[16] This process of reduction is also called *cut elimination*.

[17] Some people use the term *isomorphism* but there are technical difficulties involved in making the correspondence one to one, so I prefer the weaker terminology.

3.4 Strong Normalization and Program Extraction

Now it is obvious that a long and complicated formal proof has an even longer typed lambda calculus expression associated with it. If, however, all the possible reductions are carried out it may become considerably simpler. Indeed, in the cases with which we are concerned we can usually omit all the types. (They will have served their purpose of ensuring that we get a result of the correct type when the proof is complete. This is related to the use of types in computer programming languages.)

The maximum benefit is when we have a *Strong Normalization Theorem* for the system. Such a theorem says that, whatever the order of the reductions – and there may be many possible different reductions for a long lambda term – the process always stops. (One reason the process might be expected *not* to stop is clear when you look at substituting $x + x$ for x: the number of xs goes up at each substitution and the expression gets longer!)

The Curry-Howard correspondence can be extended to the other logical connectives by modifying the lambda calculus. Surprisingly, in addition to the above operations involving lambdas, we only need the formation of ordered pairs and the projections onto the first and second elements of those pairs in order to capture all first order logic.[18] We give only a few examples; the full details can be found in [22]. The Curry-Howard term for a conjunction $(A_1 \wedge A_2)$ obtained by the rule of $\wedge$)-introduction is the ordered pair $(p : A_1, q : A_2)$ of type $(A_1 \wedge A_2)$where $p : A_1$ is the Curry-Howard term for the proof of A_1, and similarly $q : A_2$ is the Curry-Howard term for the proof of A_2. Conversely we use the projections fst and snd for the rules ($\wedge$-E$_1$) and ($\wedge$-E$_2$). For the rule ($\exists$-I) we get the term $(t', p : A[t'/y])$ where the premise has the Curry-Howard term $p : A[t'/y]$. Thus the Curry-Howard term contains the term t' that had earlier been proved to exist.

The major consequence of the Strong Normalization theorem is then that, if we prove a formula of the form $\exists x A(x)$, we can actually extract, from the normalized proof (i.e. the lambda, or Curry-Howard, term in which no more reductions are possible), an x such that $A(x)$. Further, if we can prove $\forall x \exists y A(x, y)$ then we can actually get a program such that, given an x, it will compute a corresponding y. Moreover, we have a proof of $A(x, y)$ for this x and y so the program is "correct" in the sense that it meets its specification.[19]

Curry-Howard terms are, in general, a generalization of the idea known variously as formulae-as-types or, better, as proofs-as-types: the terms code up a whole proof by successively encoding the applications of the logical rules in a proof.

Not surprisingly, not all rules of logic allow us to prove a strong normalization theorem. One major obstacle is the law of double negation: From $\neg\neg A$ infer A. If we had a rule that would allow us to prove $\exists x A(x)$ from $\neg\neg\exists x A(x)$, how do

[18] The process can also be extended to higher order logic.

[19] Intuitively speaking, the specification is the statement about the result of the program. See also below Section 3.5.1.

we obtain such an x? There is no clear way. So we generally restrict ourselves to constructive logic and all is well.

Changing to other systems, e.g. arithmetic, may bring in other axioms. Here the most dramatic is the rule of induction. Fortunately the induction axiom

$$\frac{A(0) \quad \forall x(A(x) \to A(x+1))}{\forall x A(x)}$$

gives rise to a reduction *exactly* corresponding to the recursion

$$f(\overline{a}, 0) = g((\overline{a}) \tag{5}$$
$$f(\overline{a}, x+1) = h(\overline{a}, x, f(\overline{a}, x)) \tag{6}$$

Happily we can prove a strong normalization theorem for arithmetic (see [22]). We can therefore extract programs from these proofs.

Conclusion 3 *Analogies and similarities, especially geometric similarities, can lead to new discoveries and unexpected parallels.*[20]

3.5 Beyond traditional logic in program extraction

3.5.1 Algebraic Specifications

We now turn to an application of the above ideas to software engineering. Producing programs that satisfy their specifications is a primary goal of software engineering. We start with an algebraic specification and then construct a program. What is an algebraic specification? It is a description in formal logic of a structure, for example, the natural numbers.

As an example we use the *Common Algebraic Specification Language* (*CASL*, see [13]) but the technique could be employed in other specification languages, indeed originally we ourselves used a different language.

Structured specifications in *CASL* are built from *basic* (or *flat*) specifications by means of *translation (or renaming)*, written **with**, taking *unions* of specifications, written **and**, *hiding* signatures, written **hide** and the *extension of specifications*, written **then**. A typical example of a flat specification, this one is for natural numbers, is given in Fig. 2.

When we change a specification, then what is true changes – even if simply because we use new names, e.g. "car" instead of"'auto", "boot" instead of "trunk", etc. but we may also add new predicates (relations). We have developed logical systems to reflect the interaction between such changes and the logic statements.

Originally Martin Wirsing studied a logical calculus for structured specifications (see [78]). This was subsequently extended by Wirsing and his student

[20] On other occasions they will be at best suggestive, but possibly even misleading, as analogies have been. See [45] for examples of the overuse of analogy.

spec NAT =
sorts
Nat
ops $0 : Nat; s : Nat \to Nat; + : Nat \times Nat \to Nat$
preds
$\geq : Nat \times Nat$
axioms

$\forall x : Nat \bullet x + 0 = x$	$\%(Nat\_1)\%$
$\forall x; y : Nat \bullet x + s(y) = s(x + y)$	$\%(Nat\_2)\%$
$\forall x : Nat \bullet x \geq 0$	$\%(Nat\_3)\%$
$\forall x; y : Nat \bullet x + y = y + x$	$\%(Nat\_4)\%$
$\forall x : Nat \bullet s(x) \geq x$	$\%(Nat\_5)\%$
$\forall x; y; v; w : Nat \bullet x \geq v \wedge y \geq w \to x + y \geq v + w$	$\%(Nat\_6)\%$

end

Fig. 2. The specification NAT.

Peterreins. The system at that stage was quite complicated. However, by reflecting on the way that logical rules are developed Wirsing and the present author were able to reformulate the rules in a way that looked almost traditional in [79]. Next Wirsing and the present author extended the idea to algebraic specifications, and then we went further with Iman Poernomo, to include even the parametrized specifications of the language *CASL*.

Abstractly speaking we have an annotated or labelled deductive system.[21] The basic form of a rule in such a logic can be written in the form

$$\frac{p : A \quad q : B}{s(p, q) : \sigma(A, B)}$$

It is convenient to use "contexts" also. That is to say, the actual hypotheses with which we are working. These will be written in the standard logical style using the "turnstile" symbol $\vdash$. Thus one writes $\Gamma \vdash A$ to indicate that A is provable in the context Γ (or equivalently, from the hypotheses Γ).

The annotations we use also involve Curry-Howard terms, specification names and the logical connectives. We have two kinds of rules: those for the logical connectives, *logical rules*; and those for the structural changes in the specifications, *structural rules*. Even with the purely logical rules, the specification of the conclusion depends on those in the premises. For the structural rules, the change in the structure is reflected in the specification of the conclusion.

The logical rules for our system *Structured Specification Logic* are very similar to the standard rules of intuitionist logic. The complete set of rules, including the structural rules, that we have for *CASL*, with their Curry-Howard terms, may be found in [21] or [67].

[21] The logical system that we then have is therefore related to the *labelled deduction systems* of Gabbay [33].

When we wish to extract programs from proofs which are derived from algebraic specifications the Curry-Howard terms that we use are now more complicated for two reasons. In addition to the information from, for example, the logical rule being used, the Curry-Howard term also has to "remember" the specification. We have a similar situation for the structural rules. However, the message is as before: the Curry-Howard term carries *all* the information as to how we have constructed the proof so far.

In this situation we are again able to prove strong normalization. From this strong normalization theorem we are then able to give an *extraction map*, that is to say, we give a formal process which, given a Curry-Howard term for a proof of $\forall x \exists y A(x, y)$ from a given specification, the extraction map returns a suitable y for a given x. Indeed it gives a program in the programming language *Standard ML*. The extraction map works recursively and, in particular, the cases for $\rightarrow$-introduction and elimination correspond directly to the procedures we have outlined above.

3.5.2 Imperative programming

My recent PhD student, Iman Poernomo, has developed a protocol for integrating ordinary computer programs into the kind of deductive system we have been discussing. This protocol he calls the *Curry-Howard* protocol. The logical system for such a situation includes the state of the system (i.e. the contents of registers in the machine) and accounts for the changes that take place when a program is run. Despite the complications this produces it is still possible to produce a constructive version of a Hoare logic (*cf.* [43]), for reasoning about imperative programs, to which the Curry-Howard isomorphism may be adapted.

However we are also concerned to use programs already in the programming language that we regard as "reliable". We do not use the word "correct" here, reserving that word for programs that have been formally proved to meet their specifications. Here we simply mean that we have programs that we are satisfied will give the correct answers. Such programs include very simple ones such as programs for the multiplication of natural numbers. This achieves a significant saving in the length of the programs extracted. Otherwise we would have to prove a formula in formal arithmetic that allows us to extract a program, for example, for the multiplication function. The proof would be inordinately long, involving several applications of induction and its corresponding program would then involve the same number of recursions. This is obviously very uneconomical because we know it is possible to write a relatively simple program for multiplication (if one is not built into the computer already).

Imperative computer programs have *side-effects*: they change the *state* of the machine and, in particular, the values in various registers. The presence of side-effects is a principal feature that distinguishes the imperative programming paradigm from the functional one. However, side-effect-free functions are also important in imperative programs because they enable access to data, obtaining views of state and producing return values. Imperative programs involve both

side-effects and side-effect-free return values. Consider, for instance, a program that triples the number in the register `s` and returns a value that is twice the value in `s`. In *Standard ML* the program is

$$\texttt{s} := !\texttt{s} * 3; !\texttt{s} * 2$$

It has a side-effect producing assignment statement, $\texttt{s} := !\texttt{s} * 3$, followed by the return value $!\texttt{s} * 2$. In many popular imperative languages such as *Standard ML* or *LISP*) such return values are potentially complex, involving higher order functional aspects that are difficult to program correctly.

Our goal is to specify, reason about and synthesize both aspects of imperative programs – side-effects and functional return values. Our approach is as follows. We use a version of Hoare logic to synthesize the side-effect producing aspect of a program, specified in terms of pre- and post-conditions. Hoare logic [43] involves considering triples of the form

$$\{\textit{pre-condition}\}\texttt{program step}\{\textit{post-condition}\}$$

The *pre-condition* is true before the program step commences and the *post-condition* is true after the step.
The formula

$$s_f > s_i$$

specifies a side-effect where the final value of state `s`, denoted by s_f, is greater than the initial value, denoted by s_i. We can use Hoare logic to synthesize a *Standard ML* program that satisfies this specification, by producing, for example, a theorem of the form

$$\vdash \texttt{s} := !\texttt{s} * 3 \bullet s_f > s_i$$

where the left-hand-side of the $\bullet$ symbol is the required *Standard ML* program (written in teletype font), and the right-hand-side is a true statement about the program.

To specify and synthesize return values of a program we adapt *realizability* and the extraction of programs from proofs. We have already treated the latter, so now we consider realizability.

When we extract a program we wish to demonstrate that it is "correct". This requires the notion of *realizing*. This is a different way of verifying proofs in intuitionistic logic by means of computable functions. It was first developed by Kleene. (See the last chapter of [49].) The basic idea is that we produce a program for a (partial) recursive function that is a *witness* to the proof of an assertion. Such witnesses can be produced recursively by going down through the proof. Such a program can be regarded as a number (for example, the binary string that encodes the program). For example, if we have partial recursive functions with programs p, q realizing A, B, respectively, then we take (p, q) as the realizer of $(A \wedge B)$. The full details, which may be found in Kleene [49] for the basic system of intuitionist logic and in our book [67] for the systems we discuss here.

Here is an example. Given the theorem

$$\texttt{s} := \texttt{s} * 3 \bullet s_f > s_i \wedge (\exists x : int.Even(x) \wedge x > s_i)$$

we can synthesize a program of the form

$$\mathtt{s} := \mathtt{s} * 3; \mathtt{f}$$

where the function f is a side-effect-free function (such as $!\mathtt{s} * 2$) that realizes the existential statement of the post-condition $(\exists x : int.Even(x) \wedge x > s_i)$, by providing a witness for the x.

When using our program extraction, users will have no need to manually code the return value, instead they can work within the Hoare logic. There they prove a theorem from which the return value is then synthesized.

Conclusion 4 *Techniques developed in one part of a discipline may be applicable in another. One needs to use one's eyes, and one's ingenuity.*

3.6 Proofs from programs [22]

So far we have seen how to obtain programs from proofs in constructive systems of logic. Therefore we could conclude that all proofs are already programs, or at least, that every proof in (constructive) logic contains a program.[23]

What if we were to write the program first? Would we automatically have a proof? The answer is obviously "No!" if we simply write computer programs as many people do. However, a thoughtful computer programmer would wish to know that the program written would do what it was expected to do, that is to say, would meet its specification.[24] Therefore, as part of the task of writing the program, a proof should be produced at the same time.

The approach that we have presented shows how to accomplish both of these tasks at the same time. It does not require a separate investigation to produce a proof that the program will be correct.

From a practical point of view it is sometimes obvious how to write the proof. I studied a program for quicksort.[25] Then I wrote a proof corresponding to the program and extracted a program from it. The resulting program was essentially the quicksort program from which I had started. However I have not yet been able to formalize the procedure that I used in producing the proof from the program. It would appear that one needs to know the *algorithm*, rather than the program, in order to construct the proof. This in itself indicates that one also needs to know that the program is a *correct* implementation of the algorithm. This is indeed work for the future.

In addition, perhaps we ought to add to syntax and semantics, as major concerns of logic, implementation. Consider the process of writing a computer program, even when a formal specification is given.

[22] Alternatively this subsection might be labelled **Changing direction**.

[23] The restriction to constructive systems of logic is essential for us.

[24] This is a very serious issue when it comes to the control of powerful systems, in particular, the control of nuclear weapons.

[25] This was inspired by looking at work of Helmut Schwichtenberg on program extraction in [4].

We have to work out the algorithm that will fulfil the specification. Then we have to implement that algorithm in a computer language[26] So if one is going to use computer proofs, then besides the syntax of our formal language, perhaps we ought also to consider the implementation of our logic in the machine and the effects that that will have.

3.7 Programs then proofs

The second interpretation of the statement at the beginning of Section 3.1 revolves around the behaviour of computing machines. In this case, instead of starting with a proof in logic we start with a computer program.

We consider a specification. This may be an informal one but it seems inevitable that at some stage it will have been turned into a formal one. The specification is then built up into a program and, accompanying that, or subsequent to it, a verification is built.

We may contrast the two methods: the first of extracting programs from proofs, and the second of providing a proof for (or in the process of writing) a program, that is to say, *verifying* a program, in the following table.

Extraction	Verification
Specification	Specification
Proof	Program
Program extraction	Program verification

This second is the method invented by Tony Hoare [43] mentioned above in Section 3.5.2. The problem with this method is that it gives a necessary condition for correctness and not a sufficient one. Nevertheless the method is widely used and very valuable.

Conclusion 5 *A logical method is only clearly useful if it is used.*

4 What are logical systems and what should they be? [27]

We started out with the simple systems of Aristotle. Now we have progressed to systems where the form of the rules is (essentially) the same: two premisses and a conclusion. However, there is a great deal more baggage accompanying the traditional logic. There are labels which may represent a specification of a system, or a state of a machine.

Thus the techniques we have presented here are based on a variant of Gabbay's labelled deductive systems [33]. Our logical rules are of the form

$$\frac{\text{Logical context, State, Curry-Howard term} \vdash \text{Formula}}{\text{New Logical context, New State, New Curry-Howard term} \vdash \text{New Formula}}$$

[26] And some may add, for a particular implementation. On this problem Hoare's original paper [43] is still very valuable.

[27] This section heading is inspired by Dedekind [24].

although the actual order may vary. Further, each of the items on the lower line may depend on, that is to say, be functions of, any or all of those on the top line, and of course there may be two or more sequences on the top line.

The semantics of these rules will depend on the structures that we are using. Also the interpretation of the informal terms: Logical context, State, etc. will also vary.

What seems to be most important is that we have extended the notion of logic in two ways. First of all we now have programs or other constructions (for example, specifications) interacting with the standard logical connectives. Secondly, the context of the logic may change in the course of a proof. This certainly happens in the context of algebraic specifications. In the simplest case we may just be changing the language, say, from English to American. Thirdly, we are now discussing logics (plural) and we arrive at such a logic by an analysis of a technical setting. This seems to me to be following Aristotle's approach of looking at the real world, or a small part of it, and then abstracting the logical principles that work in that arena. But we have come a long way from the "one true logic" that I mentioned in Section 3, near the beginning!

But this has only been in a very limited number of contexts, in particular, algebraic specifications and imperative programming. The idea of building proofs and programs together has surely much more potential.

Conclusion 6 *The rules that we have in traditional logic represent the rules of rational thought. There is no reason why we should not look at the logic of other procedures or constructions. Logics can reflect the logic of systems other than human thought.*

4.1 Higher order logic

Let us now turn to more extensions of logic. It should not be surprising that the logician Dana Scott should have taken up the question of the semantics of computer programs and programming languages when he came in contact with Christopher Strachey in the 1960s. (See [73].) However, in this context it is very startling that the (untyped) lambda calculus, which sits at the base of the Scott-Strachey semantics, should not admit set-theoretic models (see [68]).

The models require equating (in some sense) a set A with the set of all functions from A to A, that is, $A \to A$ or A^A which is immediately an uncountable set if A is infinite.

An equally disturbing situation arose when Henkin was establishing his completeness proofs for the theory of types [40]. In the this theory (even as first devised by Russell [70, 71]), types are built up from basic types by means of juxtaposition. Given two types σ and τ one forms the type $(\sigma\tau)$ which can be regarded as the collection of all functions from type σ to type τ. In his thesis, Leon Henkin produced two completeness proofs (see [39, 40]). The one for first order logic referred to in Theorem 1 above and one for the theory of types. The models Henkin constructed in his proof for first order logic gave names to all the elements of the model. In the theory of types (and the same applies to higher

order logic) there are uncountably many objects and therefore one cannot give names to all of them (from a countable alphabet/language). Thus there arose models that were not standard.[28] In such models, the set of all subsets of a given set, for example, was modelled by a set which did not have the "correct" cardinality: its members were only ones that had names.

Perhaps because of this second order logic (and other higher order logic) fell into disuse. It has recently been resurrected. This is partly because of a renewed interest in the theory of types and its applications in computer science (for example, for type-checking) and partly because it is possible to simulate second (and higher) order objects by using different *sorts* of first order ones. Thus one may distinguish points and lines in a graph by having predicates $P(x)$ for "x is a point" and $L(x)$ for "x is a line" when dealing with graph theory. To ensure these work together appropriately one requires extra axioms. An illustration may be found in [48]. Feferman [30] also notes this point.

Conclusion 7 *We use an alphabet constructed from a finite number of symbols to talk about infinite things. Countability is not an issue for the analysts, why should it be one for logicians?*

Another drive for looking at higher order logic has come from logic programming. Originally logic programming(see e.g. [12]) essentially dealt with only a quantifier-free fragment of first order logic. Over recent years, however, it has been developed into higher order logic and has embraced the lambda calculus. The work of Dale Miller has been central in this. (See [60] and more recently [59] and the links there.) As Miller says: "This programming language incorporates large amounts of logic." (Downloaded from `http://www.lix.polytechnique.fr/Labo/Dale.Miller/lProlog/cse360/syllabus.html`)

Conclusion 8 *Logic can be used not only to analyze but also to synthesize.*

4.2 A note on set theory

There was a dichotomy between set theory and logic existing from around 1900. Set theory had entirely different origins from logic, although the elucidations of foundations[29] did invigorate both. Cantor [8] did not reach his paradise from logic but from analysis. Problems abut Fourier series gave rise to sequences of operations longer than ω, the first infinite ordinal.

In the middle of last century logic reached a zenith, in the author's opinion, with Paul Cohen's work on the independence of the axiom of choice [14, 15]. Since that time the area of set theory has become more and more complicated.[30] The

[28] These should be distinguished from the non-standard models of Abraham Robinson [69].

[29] Whatever "foundations" may mean!, cf. Feferman's book [30] and Bostock's logistical [7].

[30] The same can be said of parts of model theory, thanks to the spectacular work of Shelah.

latest moves in this area, however, seem to be of a different nature. Woodin's recent work on the Continuum Hypothesis [80], with his new idea of *Strong Logics* takes us to a different approach to set theory. Whether it will lead to a renaissance by simplifying the arena remains to be seen.

4.3 Computation and proof

Algorithms are now today's lifeblood as functions were in the nineteenth and early twentieth centuries. Algorithms make us think of computations but, as we know to our cost as computer users, algorithms, implemented in the software that we use everyday, do not always produce the answer or behaviour that they should.[31]

Along with means of computation one also needs means of proof and we have shown two approaches to supplying such proofs above. So computation and proof should be developed together. Otherwise, how does anyone see the need for a proof that the computation actually works?

5 The nature of proof

There is also the question of why proofs are needed in mathematics in general. This is sometimes harder to see.

Frank Harary gave a very nice exposition on this topic many years ago in Malaysia.[32] I heard it, and I still remember and heed his instructions. The essence was as follows. Suppose you want to present to an audience a theorem that A happens under the hypothesis H.

1. Give examples where A occurs.
2. Give counterexamples where it does not occur.
3. Prove the theorem under the hypothesis H.
4. Give examples where the hypothesis H does not hold, and A does not occur.

(The last item may be strengthened. Harary was at pains to give theorems where the hypothesis was as weak as possible.)

Such a procedure also leads to better understanding, and surely understanding should accompany doing (in particular, calculating or computing).

Proofs give understanding, or at least they should; computations give results. But what is a proof?

In the preceding sections we have restricted ourselves almost entirely to proofs in formal logical systems, though we have seen how these systems have been developed far beyond the original ones of, say, Frege.

We have also noted that mathematicians have often replaced giving a proof by instead showing that a proof exists.

[31] This is because there are two kinds of problems here. The first is the obvious one of human error. The second is that the specification for the computation, while being correct as a specification, may not be a specification for what was really desired to be computed.

[32] This was at a seminar at the University of Malaya associated with the first bi-annual meeting of the South-East Asian Mathematical Society in 1974.

5.1 The question of scale and the rôle of technology

At this point in our history we can use mechanical or, more frequently, electronic devices to perform repetitious tasks or to shortcut them. This has been the case since slide rules were introduced – some might even say, since the abacus was introduced. What is different now is that a multitude of tasks, identical or different, can be performed at, so to say, the touch of a button and very quickly. This change in quantity brings with it a change in quality.[33]

It is easy to deal well with single computations such as finding an n-th root. This is because we can give a clear and explicit process for finding such a root *and* a proof that it is correct. On the other hand we do not seem to be able to deal as well or as adequately with long sequences of computations or of proofs. The very complexity of some computations is too much for us to grasp. Further mechanical aids may give wrong results. For example, try taking the square root of a number over and over again on (different) hand-held calculators. Usually twenty or so times will suffice to illustrate the problem.

We do not seem to be able to handle such long sequences in a way that is satisfactory enough from a formal point of view (*cf.* Devlin's article [26]).[34]

Let us also be thoughtful about when it is appropriate to use computers or calculators. We use them when we do repetitious work. When we need to do a calculation a thousand times, or even weekly or daily, it is foolish to use pen and paper: a computer or calculator is more reliable and less stressful. This is an issue involving scale, here meaning the number of times we repeat a calculation.

Note. I have not touched on the subject of complexity although I believe that it will become very much involved, especially in the context of computer proofs (see above Section 5.1). This is partly because I find the notion of complexity ill defined. Recent work on parametrized (or as they call it "parameterized") complexity by Downey and Fellows, see e.g. [29], has helped to make some improvement so that one can get a more realistic, that is to say, useful, approach to the concept.

One of the great virtues of mathematics is that, since mathematics gives us the logic of the world, we can use it for any scale of activity. Here "scale" does not only refer to physical size. It also includes complexity. Thus in training people to use and understand mathematics we should show them how we can shift from macrocosm to microcosm or anywhere in between and still be able to use our mathematical techniques.[35] Likewise we can look at processes and at pieces of software in the same mathematical way.

To what extent can we formalize this? To what extent should we formalize proofs? One overriding advantage of formal logic as practised by Russell was that

[33] Compare the music of Philip Glass where repetition gives a different, distinctive, quality to his music.

[34] The way that we (attempt to) cope with this is to take a more macroscopic view and to look at the structure of the computation in a more coarse-grained way. Then we may be able to see our way through the various levels of the computation.

[35] Of course there are some physical situations, for example in atomic physics, where the scale means we have to use *different* mathematics, but we still use mathematics.

the proofs obtained were tangible and could be mechanically checked. However they differ remarkably even from proofs in the mid-twentieth century. Consider, for example, a standard modern algebra text [5]. The treatment starts off with integral domains (very much modelled on the natural numbers and then the integers). Induction is introduced but within a few pages the idea of induction has gone from its use *within* the domain to induction about the domain. To be precise: induction is used to prove, for example, the distributive law

$$x * (y + z) = x * y + x * z$$

yet a few pages later it is used to prove the general associative law

$$(a_1 * a_2 * \ldots * a_{n-1}) * a_n = a_1 * (a_2 * \ldots * a_n)$$

In this setting we have a proof in the metalanguage. This is not remarked on by the authors but is very striking to a logician. This is by no means an isolated example. So one becomes accustomed to "layering" as I have referred to it. (See my paper [19].) However, apart from acknowledging that one has moved to a metalanguage, there seems to be a dearth of formal systems allowing one to make such moves.

Creating such a system would allow the contracting of proofs in a very formal way. Nevertheless, the problems of sheer size (as noted in Barendregt's [2]) have to be dealt with in some way and, as Alan Robinson points out (quoted by Donald MacKenzie in his paper at the same Royal Society Discussion Meeting on 18–19 October 2004, see [57])

> You've got to prove the theorem-proving correct. You're in a regression, aren't you?

That is to say, if we have a mechanical means of proving a theorem, we then need a proof that this prover is correct. If we have a mechanical way of proving that, then we again need a proof of correctness for this latest device, and so on. So the question of a final authority emerges.

Despite Lewis Carroll's salutary paper [27] of 1895, around 1900 there was a general faith that there *was* such a final authority and people were at pains to provide one. Russell's system of logic was intended to do that, and as we have noted, it failed. Nowadays I believe it would be foolish even to search for such a final authority. Nevertheless there is a remarkable robustness about mathematical proofs as John Shepherdson point out to me many years ago.[36] And this is despite the assaults of Imre Lakatos on the notion of proof and his beautiful illustration of the evolution of a specific proof and the mathematical definitions surrounding it in his [54].

Frank Ramsey's dictum,[37] presumably from the 1920s, remains a serious question:

[36] In private conversation, Melbourne 1992.
[37] Quoted in [56].

> Suppose a contradiction were to be found in the axioms of set theory.
> Do you seriously believe that a bridge would fall down?

We shall not pursue the nature of mechanical proof further here but do commend the papers at the Royal Society Discussion Meeting on 18–19 October 2004 which we hope will all eventually appear in print. We simply draw attention to other kinds of proof.

In a footnote on p. 101 in his [50], Kreisel pointed out that if we had, and accepted, a classical proof that there were indeed infinitely many twin primes, then we should immediately have an algorithm: just test pairs of primes bigger than the given pair until we find the next pair!

The question then arises: what is the appropriate logic?

If one takes intuitionism seriously, as did Brouwer, and espouses its philosophy, then surely only proofs constructed in an intuitionist fashion are acceptable. In particular, a proof that a theorem can be proved intuitionistically should be an intuitionistic proof. For many of us this is too much to ask. We would rather accept the approach of Kreisel's footnote. Obviously one can repeat this argument at any level.

Finally just in case the notion of proof seems purely logical, consider what I regard as the most astounding proof in all the mathematics I have ever seen or even heard of. Near the end of Dedekind's [24] there is a (claimed) proof that infinite sets exist.[38] It begins: "Consider the realm of my thoughts ...". It then goes on to consider not just thoughts but thoughts of thoughts. In this way an infinite set is constructed. How psychological and unmathematical this is! It is so far away from the studious way that Dedekind has built up the characterization of the natural numbers. The proof is not very well known. It should be better known to serve as an object lesson.

Conclusion 9 *We should consider not only the logic in which we do a (logical) proof, but also the logic of the system in which we do the proof.*

5.2 Foundations

Which, of all the logics we have discussed in this paper, or perhaps rather, which combinations of these logics should we use, seems quite problematic.

It also brings up the question of foundations. The, to the author's mind, unsatisfactory debates of the twentieth century on logicism, formalism and intuitionism seem to have left out the human dimension and the fact that Mathematics (and with it Logic) is a human activity. Work such as MacKenzie's [57] must surely be taken into account.

Added to this we shall also need to consider the time that it takes to get a proof. Even if we are producing a mechanical proof the question will ultimately arise as to deciding at what stage the computation is satisfactory, *cf.* Alan Robinson's remark above.

[38] The assertion of the theorem should be compared with Russell's axiom of infinity.

Finally, there is the question of how deeply we can, or should, examine what a logical system is. The work of Yesenin-Volpin, though much neglected, has been mentioned recently in my hearing. There are at least two aspects worthy of consideration. The first is whether the (logical) universe is finite, and in particular, whether there is only a finite number of natural numbers. This is documented in [81]. The second item is his analysis, not recorded in that work but presented in lectures at SUNY, Buffalo about 1972, which I heard. This concerns his detailed analysis of even the symbols used in the logical expressions and their repetitions: – if they are indeed repetitions; in what sense are the symbols "the same". At the present time this seems too far-fetched to warrant further investigation, but what has seemed obvious in earlier centuries has sometimes later turned out to be quite problematical. The most outstanding example of this is perhaps that of infinitesimals which were happily used by Leibniz, then came into disrepute in the nineteenth century, and finally were resurrected and justified by Abraham Robinson in the 1960s, see [69]. However, I do not believe that Abraham Robinson's infinitesimals are in any way the same as those of centuries earlier. There is moreover a question as to whether Robinson actually defined a unique set of infinitesimals (even in the context of the real numbers).[39]

In a different setting, while the proof systems look very similar (one only has to give up the law of the excluded middle to go from classical to intuitionist logic) nevertheless the actual working of the systems of intuitionist logic and classical logic are dramatically different. In this case as is often the case:

Conclusion 10 *We often need to start again each time we revisit an idea. There is no guarantee that the previously used techniques will work.*

6 Final remarks

The ten conclusions that we have drawn encompass a number of different aspects of the development of logic. First there is the variety of logics that now exist and can be expected to proliferate (Conclusion 1). Then there is the "transfer of technology": the use of the same pattern, same idea, or an analogous one in another context (Conclusions 2,3,4). Prejudices should be broken down: we should look beyond our own narrow horizons (Conclusions 6,7,9). I believe we also have a duty to our fellows: a logical method is only clearly useful if it is used (Conclusion 5). We should think of our fellows in developing logic. Then there is the fact that logic can be used not only to analyze but also to synthesize (Conclusion 8). But each time we may need to start again, almost from scratch at times, but informed by the past (Conclusion 10).

So our conclusions are not endings, but new beginnings.

Samsara.

[39] He gives a procedure for getting a model of the real numbers, including infinitesimals, but any one of a range of models will give the required results.

References

1. Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
2. Hendrik Pieter (Henk) Barendregt. The challenge of computer mathematics. Submitted to the Royal Society, `http://www.cs.ru.nl/~freek/notes/RSpaper.pdf`, accessed 8.03.05.
3. John Lane Bell and Alan B. Slomson. *Models and ultraproducts: an introduction*. North Holland, Amsterdam, 1969.
4. Ulrich Berger and Helmut Schwichtenberg. Program extraction from classical proofs. In D. Leivant, editor, *Logic and Computational Complexity, International Workshop LCC '94, Indiapolis, IN, USA, October 1994*, pages 77–97, 1995.
5. Garrett Birkhoff and Saunders Maclane. *A Survey of Modern Algebra*. Macmillan, New York, 1st edition, 1941.
6. George Boole. *An investigation of the laws of thought on which are founded the mathematical theories of logic and probabilities*. Dover, New York, 1958. Originally published in 1854.
7. David Bostock. *Logic and arithmetic*. Clarendon Press, Oxford, 1974–79.
8. Georg Cantor. Über die Ausdehnung eines Satzes aus der Theorie der trigonometrischen Reihen. *Mathematische Annalen*, 5:123–132, 1872.
9. Chen-Chung Chang and H. Jerome Keisler. *Model theory*. North-Holland Pub. Co., 1973. 3rd ed. 1990.
10. Alonzo Church. An unsolvable problem of elementary number theory. *Proc. Nat. Acad. Sci., USA*, 20:584–590, 1934.
11. Alonzo Church. *Introduction to Mathematical Logic, Volume I*. Princeton University Press, Princeton, NJ, 1956.
12. William F. Clocksin and Chrisopher S. Mellish. *Programming in Prolog*. Springer Verlag, New York, 3rd edition, 1987.
13. CoFI Language Design Task Group on Language Design. *CASL, The Common Algebraic Specification Language, Summary, 25 March 2001*, March 2001. Available at `http://www.brics.dk/Projects/CoFI/Documents/CASL/Summary/` (accessed 3.01.2005).
14. Paul Joseph Cohen. The independence of the continuum hypothesis. *Proc. Nat. Acad. Sci. U.S.A.*, 50:1143–1148, 1963.
15. Paul Joseph Cohen. The independence of the continuum hypothesis. II. *Proc. Nat. Acad. Sci. U.S.A.*, 51:105–110, 1964.
16. Louis Couturat. *Opuscules et fragments inédits de Leibniz*. Feliz Alcan, Paris, 1903.
17. Carl C. Cowen. On the centrality of Linear Algebra in the curriculum, remarks on receiving the Deborah and Franklin Tepper Haimo Award for Distinguished College or University Teaching of Mathematics, San Diego California, January 1997. `http://www.maa.org/features/cowen.html`.
18. Maxwell John Cresswell and George Edward Hughes. *A New Introduction to Modal Logic*. Taylor & Francis Inc., 1996.
19. John Newsome Crossley. Structures with features. Presented at the International Conference in Honor of Fr B.F. Nebres, Manila, February 2001, and submitted for publication in *Matimyás Matematika*. Available at `http://www.csse.monash.edu.au/~jnc/features.pdf` or `../features.ps`.

20. John Newsome Crossley and Anil Nerode. *Combinatorial functors*, volume 81 of *Ergebnisse der Mathematik und ihrer Grenzgebiete*. Springer, New York, 1979.
21. John Newsome Crossley, Iman Poernomo, and Martin Wirsing. Extraction of structured programs from specification proofs. In Didier Bert, Christine Choppy, and Peter Mosses, editors, *Workshop on Algebraic Development Techniques*, volume 1827 of *Lecture Notes in Computer Science*, pages 419–437, 1999.
22. John Newsome Crossley and John Cedric Shepherdson. Extracting programs from proofs by an extension of the Curry-Howard process. In John Newsome Crossley, Jeffrey B. Remmel, Richard A. Shore, and Moss Eisenberg Sweedler, editors, *Logical Methods: In honor of Anil Nerode's Sixtieth Birthday*, pages 222–288. Birkhäuser, Boston, MA, 1993.
23. Haskell Brooks Curry. Functionality in combinatory logic. *American Journal of Mathematics*, 58:345–363, 1936.
24. Richard Dedekind. The nature and meaning of numbers. In *Essays on theory of numbers*. Dover, 1963. Originally published in 1901. Translation of *Was sind und was sollen die Zahlen?*
25. Jacob C. E. Dekker and John Myhill. Recursive equivalence types. *University of California Publications in Mathematics (NS)*, 3:67–214, 1960.
26. Keith Devlin. Devlin's Angle: When is a proof? In *MAA Online*. Mathematical Association of America, June 2003. `http://www.maa.org/devlin/devlin_06_03.html`, accessed 8.03.05.
27. Charles Lutwidge Dodgson. What the Tortoise said to Achilles. *Mind*, 4(14):278–280, 1895.
28. Jean-Luc Dorier. A general outline of the genesis of vector space theory. *Historia Mathematica*, 22:227–261, 1995.
29. Rod Downey. Parameterized complexity for the skeptic. In *18th IEEE Annual Conference on Computational Complexity, 7–10 July 2003*, pages 147–168, Los Alamitos, California, 2003. IEEE.
30. Solomon Feferman. *In the light of logic*. Oxford University Press, 1998.
31. Solomon Feferman, John W. Dawson, Jr, Stephen Cole Kleene, Gregory H. Moore, Robert M. Solovay, and Jean van Heijenoort, editors. *Kurt Gödel: Collected works, Volume I, Publications 1929–1936*. Oxford University Press, 1986.
32. Gottlob Frege. *Begriffsschrift und andere Aufsätze*. Olms, Hildesheim, 2nd edition, 1995. Notes by E.Husserl and H.Scholz, edited by I.Angelelli.
33. Dov Gabbay. *Labelled deductive systems*. Oxford University Press, 1996.
34. Kurt Gödel. Die Vollständigkeit der Axiome des logischen Funktionenkalküls. *Monatshefte für Mathematik und Physik*, 37:349–360, 1930.
35. Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931. Reprinted in [31], pages 144–194.
36. Kurt Gödel. Zur intuitionistischen Arithmetik und Zahlentheorie. *Ergebnisse eines mathematischen Kolloquiums*, 4:34–38, 1933. Reprinted in [31], pages 286–295.
37. Paul Richard Halmos. *Finite Dimensional Vector Spaces*. Princeton University Press, Princeton, NJ, 1942.
38. Thomas L. Heath, editor. *The Thirteen Books of Euclid's Elements*. Cambridge University Press, Cambridge, 2nd edition, 1925. Reprinted, Dover Books, New York, 1956.
39. Leon Henkin. The completeness of the first-order functional calculus. *Journal of Symbolic Logic*, 14:159–166, 1949.
40. Leon Henkin. Completeness in the Theory of Types. *Journal of Symbolic Logic*, 15:81–91, 1950.

41. Leon Henkin. The discovery of my completeness proofs. *Bulletin of Symbolic Logic*, 2:127–158, 1996.
42. Arend Heyting. Die formalen Regeln der intuitionistischenLogik. *Sitzungsberichte der Preussischen Academie der Wissenschaften, Physikalisch-Matematische Klasse*, 9:42–56, 1930.
43. Charles Anthony Richard Hoare. An axiomatic basis for computer programming. *Communications of the Association for Computing Machinery*, 12(10):576–80, 1969.
44. Wilfrid Hodges. *Model theory*. Encyclopedia of mathematics and its applications. Cambridge University Press, Cambridge, 1992.
45. Douglas R. Hofstadter. *Gödel, Escher, Bach: an eternal golden braid*. Vintage Books, New York, 1999.
46. William Howard. The formulae-as-types notion of construction. In John Roger Hindley and Jonathan Seldin, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, 1969.
47. Bart Jacobs and Jan Rutten. A tutorial on (Co)Algebras and (Co)Induction. *Bulletin of the European Association for Theoretical Computer Science*, 62:222–259, 1997.
48. John S. Jeavons, Iman Poernomo, Bolis Basit, and John Newsome Crossley. A layered approach to extracting programs from proofs with an application in Graph Theory. In Rod Downey, Ding Decheng, Tung Shih Ping, Qiu Yu Hui, and Mariko Yasugi, editors, *Proceedings of the 7th and 8th Asian Logic Conferences, Chongqing, China, 29 August - 2 September 2002*, pages 193–222, Singapore, 2003. Singapore University Press and World Scientific.
49. Stephen Cole Kleene. *Introduction to Metamathematics*. North-Holland, Amsterdam, 1952.
50. Georg Kreisel. Interpretation of analysis by means of constructive functionals of finite types. In Arend Heyting, editor, *Constructivity in Mathematics, Proceedings of the Colloquim held at Amsterdam in 1957*, pages 101–128. North-Holland, Amsterdam, 1959.
51. Saul Kripke. A completeness theorem in modal logic. *Journal of Symbolic Logic*, 24:1–14, 1959.
52. Saul Kripke. Semantical analysis of intuitionistic logic I. In John Newsome Crossley and Michael Anthony Eardley Dummett, editors, *Formal Systems and Recurive Functions*. North-Holland, Amsterdam, 1965.
53. Alexander Kurz and Alessandra Palmigiano. Coalgebras and modal expansions of logics. *Electronic Notes in Theoretical Computer Science*, 106:243–259, 2004.
54. Imre Lakatos. *Proofs and refutations: the logic of mathematical discovery:* Edited by John Worrall and Elie Zahar. Cambridge University Press, Cambridge, New York, 1976.
55. Clarence Irving Lewis and Cooper Harold Langford. *Symbolic Logic*. Dover, New York, 2nd edition, 1959. Originally published in 1932.
56. Des MacHale. *Comic sections: The book of mathematical jokes, humour, wit, and wisdom*. Boole Press, Dublin, 1993.
57. Donald MacKenzie. Computing and the cultures of proving. Presented at the the Royal Society Discussion Meeting on 18–19 October 2004, `http://www.sps.ed.ac.uk/staff/Royal%20Society%20paper.pdf`, accessed 8.03.05.
58. Elliott Mendelson. *Introduction to mathematical logic*. Chapman & Hall, 4th edition, 1997.
59. Dale Miller. λprolog. `http://www.lix.polytechnique.fr/Labo/Dale.Miller/lProlog/index.html`, accessed 8.03.05.

60. Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic Programming*, 1(4):497–536, 1991.
61. Bimal Krishna Motilal. *Logic, language, and reality: an introduction to Indian philosophical studies*. Motilal Banarsidass, Delhi, 1985.
62. Bimal Krishna Motilal. *The Character of Logic in India*. Oxford University Press, Oxford, 2000. Edited by Jonardon Ganeri and Heeraman Tiwari.
63. Sir Thomas Muir. *The theory of determinants in the historical order of development*. Macmillan, London, 1906.
64. Daniele Nardi and Ronald J. Brachman. An introduction to Description Logics. `http://www.inf.unibz.it/~franconi/dl/course/dlhb/dlhb-01.pdf`, accessed 2.03.05.
65. Anil Nerode. Extensions to isols. *Annals of Mathematics*, 73:362–403, 1961.
66. Giuseppe Peano. *Formulario Mathematico Editio V (Tomo V de Formulario completo*. Turin, 1908.
67. Iman Hafiz Poernomo, John Newsome Crossley, and Martin Wirsing. *Adapting proofs-as-programs*. Springer, New York, 2005 (forthcoming).
68. John C. Reynolds. Polymorphism is not set-theoretic. In *Semantics of data types (Valbonne, 1984)*, pages 145–156. Springer, Berlin, 1984.
69. Abraham Robinson. *Non-standard analysis*. North-Holland Pub. Co., 1966.
70. Bertrand Arthur William Russell. *Principles of Mathematics*. Cambridge University Press, Cambridge, 1903.
71. Bertrand Arthur William Russell. Mathematical logic as based on the Theory of Types. *American Journal of Mathematics*, 30:222–262, 1908. Reprinted in [76], 152–182.
72. Bertrand Arthur William Russell. *The autobiography of Bertrand Russell*. Allen and Unwin, London, 1967–69. 3 volumes.
73. Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1977.
74. *Summaries of talks, Summer Institute for Symbolic Logic, Cornell University*. Institute for Defense Analyses, Princeton, 2nd edition, 1960. First edition, 3 vols, cyclostyled, 1957.
75. Michael E. Szabo, editor. *The collected papers of Gerhard Gentzen*. North-Holland Pub. Co., Amsterdam, 1969.
76. Jean van Heijenoort, editor. *From Frege to Gödel*. Harvard University Press, 1967.
77. Alfred North Whitehead and Bertrand Arthur William Russell. *Introduction to Mathematical Philosophy*. Cambridge University Press, Cambridge, 1925–27. First published in 1910–13.
78. Martin Wirsing. Structured specifications: Syntax, semantics and proof calculus. In Manfred Broy, editor, *Informatik und Mathematik, Festschrift für F. L. Bauer*, pages 269–283. Springer, Berlin, 1991.
79. Martin Wirsing, John Newsome Crossley, and Hannes Peterreins. Proof normalization of structured algebraic specifications is convergent. In J. Fiaderio, editor, *Workshop on Algebraic Development Techniques, Proceedings of the Twelfth International Workshop on Recent Trends in Algebraic Development Techniques*, volume 1589 of *Lecture Notes in Computer Science*, pages 326–340, Berlin, 1998. Springer.
80. Hugh Woodin. The Continuum Hypothesis, 2000. Lectures at the Paris Logic Colloquium, `http://math.berkeley.edu/~woodin/talks/lc2000.1.pdf` – `math.berkeley.edu/~woodin/talks/lc2000.3.pdf`, three files.

81. Alexander Sergeevich Ésénine-Volpine [Yesenin-Volpin]. Le programme ultra-intuitionniste des fondements des mathématiques. In *Infinitistic methods, Proceedings of the Symposium on Foundations of Mathematics, Warsaw, 2–9 September 1959*, pages 201–223. Pergammon Press, Oxford; Państwowe Wydawnictwo Naukowe, Warszawa, 1960.