<u>Logicism and the Development of Computer Science</u>*

By Donald Gillies, King's College London

A Paper for Bob Kowalski's 60th Birthday. Published in Antonis C. Kakas and Fariba Sadri (eds,) *Computational Logic: Logic Programming and Beyond*, Part II, Springer, 2002, pp. 588-604.

Abstract

This paper argues for the thesis that ideas originating in the philosophy of mathematics have proved very helpful for the development of computer science. In particular, logicism, the view that mathematics can be reduced to logic, was developed by Frege and Russell, long before computers were invented, and yet many of the ideas of logicism have been central to computer science. The paper attempts to explain how this serendipity came about. It also applies Wittgenstein's later theory of meaning to human-computer interaction, and draws the conclusion that computers do understand the meaning of the symbols they process. The formal language of logic is suitable for humans trying to communicate with computers.

Contents

1. Introduction

Philosophy is often thought of as an activity, which may have considerable theoretical interest, but which is of little practical importance. Such a view of philosophy is, in my opinion, profoundly mistaken. On the contrary, I would claim that philosophical ideas and some kind of philosophical orientation are necessary for many quite practical activities. Bob Kowalski's researches are an excellent example of this thesis, since they have been characterised by an explicit and productive use of philosophical ideas. His work, therefore, naturally suggests looking at the general question of how far philosophy has influenced the development of computer science. My own view is that the influence of philosophy on computer science has been very great. In the first three or four decades of the computer, this influence came mainly from earlier work in the philosophy of mathematics. In the last two decades, however, there has been an increasing influence of ideas from the philosophy of science, particularly ideas connected with probability, induction, and causality. In this paper, however, I will focus on the philosophy of mathematics. In section 2 I will give a brief sketch of the development of philosophy of mathematics during the so-called 'foundational' period (c. 1879 – 1939). This period saw the emergence of three main schools: logicism, formalism, and intuitionism. As a matter of fact, all three subsequently influenced the development of computer science, but in this paper I will concentrate on logicism, partly for reasons of space, and partly because it is the philosophical position most relevant to Kowalski's work. Section 3 therefore is devoted to logicism and computer science, and I try to show two things. First of all that the ideas of logicism were developed (particularly by Frege and Russell) for purely philosophical reasons, and second that these ideas proved very fruitful in computer science. This naturally raises a problem. Why did concepts and theories developed for philosophical motives before computers were even invented, prove so useful in the practice of computing? I will attempt to sketch the beginnings of a possible answer to this question. In section 4, however, I will turn to an influence in the opposite direction. The logic invented by the logicists proved to be useful in computer science, but the application of logic in computer science changed logic in many ways. In section 4, therefore, I will examine some of the ways in which applications in computing have changed the nature of logic. Section 5 closes the paper by considering some ideas of Wittgenstein. During his later ('ordinary language') period, which began around 1930, Wittgenstein developed a criticism of logicism. I am very far from accepting this criticism in its entirety, but it does raise some interesting points. In particular, in conjunction with some of Wittgenstein's later ideas on meaning, it suggests some further reasons why formal logic has proved so fruitful in computer science.

2. Philosophy of Mathematics in the Foundational Period

The foundational period in the philosophy of mathematics (c. 1879 – 1939) is characterised by the emergence and development of three different schools, each of which aimed to give a satisfactory foundation for mathematics. These schools were:-

(i) logicism (the view that mathematics is reducible to logic),
(ii) formalism (mathematics as the study of formal systems), &
(iii) intuitionism (mathematics based on the intuitions of the creative mathematician).

Logicism was started by Frege. Strictly speaking his aim was not to show that the whole of mathematics was reducible to logic, but only that arithmetic was reducible to logic. Frege adopted a non-logicist, Kantian view of geometry. To accomplish his goal, Frege devised a way of defining number in terms of purely logical notions. The existing Aristotelian logic was not adequate for his purpose. So he devised a new kind of formal logic which he published in his *Begriffsschrift* (literally concept writing) of 1879. This is essentially the same as the formal logic taught today, except that Frege used a curious two dimensional notation, which has been abandoned in favour of the more usual one dimensional manner of writing. Frege then went on to set up a complicated formal system with what were intended to be purely logical axioms, and tried to show that the whole of arithmetic could be logically deduced within this system using his definition of number. The first volume of this formal system took Frege 9 years to complete and it appeared in 1893. By the summer of 1902 , Frege had worked for another 9 years on the second volume, which was nearing completion, and it must have seemed to him that he had successfully completed the project to which he had devoted almost his whole adult life. At this moment, however, disaster struck. Frege received a letter dated 16 June 1902 from a young logician named Bertrand Russell who showed that it was possible to derive a contradiction from what Frege had taken as the basic axioms of logic. This is what is now known as Russell's paradox. Here is an extract from Frege's reply to Russell dated 22 June 1902 (Frege, 1902, pp. 127-8):

> 'Your discovery of the contradiction caused me the greatest surprise and, I would almost say, consternation, since it has shaken the basis on which I intended to build arithmetic. It seems, then, … that my Rule V … is false … . I must reflect further on the matter. It is all the more serious since, with the loss of my Rule V, not only the foundations of my arithmetic, but also the sole possible foundations of arithmetic seem to vanish. … In any case your discovery is very remarkable and will perhaps result in a great advance in logic, unwelcome as it may seem at first glance.' [1]

Russell's discovery of the paradox did not cause Russell to give up logicism. On the contrary, Russell tried to provide logicism with new foundations. He derived what is known as *the theory of types* to resolve his paradox, and, using this theory, he constructed with A.N.Whitehead a new massive system of formal logic in which it was hoped that the whole of mathematics could be derived. When the three huge volumes of this system, known as *Principia Mathematica*, were published in 1913, it looked as if the logicist programme had been brought to a successful conclusion. However, once again, this apparent success proved short-lived. In 1931 Kurt Gödel, a logician and member of the Vienna Circle, published his two incompleteness theorems. The first of these, in its modern form, shows that if *Principia Mathematica* is consistent, then there is an arithmetical statement which cannot be proved within the system, but which can be shown to be true by an informal argument outside the system. In effect not all the truths of arithmetic can be derived in *Principia Mathematica* which thus fails in its logicist goal of reducing arithmetic to logic. If *Principia Mathematica* were inconsistent, the situation would be no better – indeed it would be worse. In that event any statement whatever could be proved in the system which would therefore be useless. Gödel showed that the results of his paper applied not just to *Principia Mathematica* but to any similar logicist system. He had thus demonstrated that it was impossible to carry out the logicist programme of Frege and Russell.

Let us now turn to *formalism*. The formalist philosophy of mathematics was developed by the German mathematician David Hilbert. Hilbert took over the concept of formal system from the logicists. The logicists tried to construct a single formal system based on the axioms of logic within which the whole of mathematics (or in Frege's case the whole of arithmetic) could be derived. Hilbert, however, suggested that a different axiomatic formal system could be constructed for each branch of mathematics, e.g. arithmetic, geometry, algebra, set theory, probability theory, etc. Frege's work had shown that there was a danger of a contradiction appearing in a formal system. To avoid this difficulty, Hilbert sugested that the formal systems of mathematics should be proved to be consistent using only the simple informal methods of finitistic arithmetic. Unfortunately Gödel's second incompleteness theorem showed that such consistency proofs could not be given for nearly all the significant branches of mathematics. Thus

Gödel had shown in a single paper published in 1931 that two of the three major positions in the philosophy of mathematics were untenable.

This leaves us wit the last of the three major schools – *intuitionism*. This was not in fact refuted by Gödel's incompleteness theorems, but it had other difficulties which made it unacceptable to most mathematicians. A systematic working out of the idea that mathematics was the intuitive construction of creative mathematicians seemed to indicate that some of the logical laws assumed in standard mathematics, notably the law of the excluded middle, had no proper justification. The intuitionists therefore created a new kind of mathematics not involving the law of the excluded middle and other suspect laws. Unfortunately this new mathematics turned out to be more involved and intricate than standard mathematics, and, as a result, it was rejected by most mathematicians as just too complicated to be acceptable.

The Wall Street crash of 1929 ushered in the depression of the 1930's. One could say that the Gödel crash of 1931 initiated a period of depression in the philosophy of mathematics. The three main schools all appeared to have failed. Not one had carried out its promise of providing a satisfactory foundation for mathematics. Yet fate was preparing an odd turn of events. In the post-War period the ideas of these philosophical programmes turned out, surprisingly, to be of the greatest possible use in the new and rapidly expanding field of computer science. In the next section I will examine how this came about in the case of the logicist programme. For reasons of space I cannot analyse the contributions of all three programmes, and I have chosen to concentrate on logicism, as it is the programme most closely connected to Bob Kowalski's work.

3. Logicism and Computer Science

Let us begin with the predicate calculus introduced by Frege in his *Begriffsschrift* of 1879 which opened the foundational period in the philosophy of mathematics. This has become one of the most commonly used theoretical tools of computer science. One particular area of application is in automated theorem proving. In his 1965 paper, Alan Robinson developed a form of the predicate calculus (the clausal form) which was specifically designed for use in computer theorem proving, and which has also proved useful in other applications of logic to computing. At the beginning of his paper, Robinson has an interesting section in which he discusses how a logic designed for use by a computer may differ from one suitable for human use. I will now expound his ideas on this point as they will be very helpful in dealing with the issues raised in the present paper.

Robinson begins by pointing out that in a logic designed for humans, the rules of inference have usually been made very simple. As he says (1965, p. 23):

> 'Traditionally, a single step in a deduction has been required, for pragmatic and psychological reasons, to be simple enough, broadly speaking, to be apprehended as correct by a human being in a single intellectual act. No doubt this custom originates in the desire that each single step of a deduction should be indubitable, even though the deduction as a whole may consist of a long chain of such steps. The ultimate conclusion of a deduction, if the deduction is correct, follows

> logically from the premisses used in the deduction; but the human mind may well find the unmediated transition from the premisses to the conclusion surprising, hence (psychologically) dubitable. Part of the point, then, of the logical analysis of deductive reasoning has been to reduce complex inferences, which are beyond the capacity of the human mind to grasp as single steps, to chains of simpler inferences, each of which is within the capacity of the human mind to grasp as a single transaction.'

If the logic is to be used by a computer, then the requirement that the rules of inference be simple no longer applies. A rule of inference which requires a great deal of computation for its application poses no problem for a computer, as it would for a human. On the other hand, for computer applications, it might well be desirable to reduce the number of rules of inference as much as possible. If a system has a large number of simple rules of inference, a human endowed with some intuitive skill could see which of these rules would be the appropriate one to employ in a particular situation. A computer, lacking this intuitive skill, might have to try each of the rules of the list in turn before hitting on the appropriate one. So we could say, that a logic for humans could have a large number of simple rules of inference, while a logic for computers would be better with fewer but more complicated rules. In fact Robinson introduced a system with a single rule of inference – the *resolution principle*. As he says (1965, p. 24):

> 'When the agent carrying out the application of an inference principle is a modern computing machine, the traditional limitation on the complexity of inference principles is no longer very appropriate. More powerful principles, involving perhaps a much greater amount of combinatorial information-processing for a single application, become a possibility.
>
> In the system described in this paper, one such inference principle is used. It is called the *resolution principle*, and it is machine-oriented, rather than human-oriented, in the sense of the preceding remarks. The resolution principle is quite powerful, both in the psychological sense that it condones single inferences which are often beyond the ability of the human to grasp (other than discursively), and in the theoretical sense that it alone, as sole inference principle, forms a complete system of first-order logic. …
>
> The main advantage of the resolution principle lies in the ability to allow us to avoid one of the major combinatorial obstacles to efficiency which have plagued earlier theorem-proving procedures.'

The important point to note here, and to which we shall return later in the paper, is that, as regards logico-linguistic systems, the requirements of a computer may be very different from those of a human.

Alan Robinson's version of the predicate calculus has indeed been used with great success in automated theorem proving. It also led through the work of Kowalski, and of Colmerauer and his team, to the logic programming language PROLOG (for historical details, see Gillies, 1996, **4.1**, pp. 72-5). Muggleton's concept of *inductive logic programming*, originated from the idea of inverting Robinson's deductive logic to produce an inductive logic. PROLOG has been an essential tool in the development of Muggleton's approach, which has resulted in some very successful machine learning programs (for some further details see Muggleton, 1992, and Gillies, 1996, **2.4**, pp. 41-44).

The examples just given, and some further examples which will be mentioned below, show that Frege's invention of the predicate calculus provided a useful, perhaps indeed essential, tool for computer science. Yet Frege's motivation was to establish a particular position in the philosophy of mathematics, namely that arithmetic could be reduced to logic. Indeed in the entire body of his published and unpublished writings, Frege makes only one reference to questions of computation. His predecessor Boole had also introduced a system of formal logic, and Jevons, influenced by Babbage, had actually constructed a machine to carry out logical inferences in his own version of Boolean logic. Jevons had the machine constructed by a clockmaker in 1869, and describes it in his paper of 1870. Frege made a number of comments on these developments in a paper written in 1880-1, although only published after his death. He wrote (1880-1, pp. 34-5):

'I believe almost all errors made in inference to have their roots in the imperfection of concepts. Boole presupposes logically perfect concepts as ready to hand, and hence the most difficult part of the task as having been already discharged; he can then draw his inferences from the given assumptions by a mechanical process of computation. Stanley Jevons has in fact invented a machine to do this.'

Frege, however, made clear in a passage occurring a little later that he did not greatly approve of these developments. He wrote (1880-1, p. 35):

'Boolean formula-language only represents a part of our thinking; our thinking as a whole can never be coped with by a machine or replaced by purely mechanical activity.'

On the whole it seems that Jevons' attempts to mechanise logical inference had only a slight influence on Frege's thinking. So we can say that considerations of computing had almost no influence on Frege's development of the predicate calculus, and yet the predicate calculus has proved a very useful tool for computer science.

Let us now move on from Frege to Russell. Bertrand Russell devised the theory of types in order to produce a new version of the logicist programme (the programme for reducing mathematics to logic) when Frege's earlier version of the programme had been shown to be inconsistent by Russell's discovery of his paradox. Thus Russell's motivation, like Frege's, was to establish a particular position in the philosophy of mathematics (logicism), and there is no evidence that he even considered the possibility of his new theory being applied in computing. Indeed Russell's autobiographical writings show that he was worried about devoting his time to logicism rather than to useful applied mathematics. Thus in his 1959 *My Philosophical Development*, he writes of the years immediately following the completion of his first degree (p. 39):

'I was, however, persuaded that applied mathematics is a worthier study than pure mathematics, because applied mathematics - so, in my Victorian optimism, I supposed - was more likely to further human welfare. I read Clerk Maxwell's *Electricity and Magnetism* carefully, I studied Hertz's *Principles of Mechanics*, and I was delighted when Hertz succeeded in manufacturing electro-magnetic waves.'

Moreover in his autobiography, Russell gives a letter which he wrote to Gilbert Murray in 1902 which contains the following passage (1967, p. 163):

'Although I denied it when Leonard Hobhouse said so, philosophy seems to me on the whole a rather hopeless business. I do not know how to state the value that at moments I am inclined to give it. If only one had lived in the days of Spinoza, when systems were still possible ...'

In view of Russell's doubts and guilt feelings, it is quite ironical that his work has turned out to be so useful in computer science.

Russell's theory of types failed in its original purpose of providing a foundation for mathematics. The mathematical community preferred to use the axiomatic set theory developed by Zermelo and others. Indeed type theory is not taught at all in most mathematics departments. The situation is quite different in computer science departments where courses on type theory are a standard part of the syllabus. This is because the theory of types is now a standard tool of computer science.

Let us now examine how Russell's ideas about types came in to computer science. A key link in the chain was Church who worked for some of his time on Russell's programme. Indeed Church's invention of the λ-calculus arose out of his attempts to develop the logicist position of Russell and Whitehead (1910-13). Russell and Whitehead had written the class of all a's such that f(a) as â f(a). Church wished to develop a calculus which focused on functions rather than classes, and he referred to the function by moving the symbol down to the left of a to produce ^ a f(a). For typographic reasons it was easier to write this as λx f(x), and so the standard notation of the λ-calculus came into being. (cf. Rosser, 1984, p. 338)

Church had intended his first version of the λ-calculus (1932) to provide a new foundation for logic in the style of Russell and Whitehead. However it turned out to be inconsistent. This was first proved by Kleene and Rosser in 1935 using a variation of the Richard paradox, while Curry in 1942 provided a simpler proof based on Russell's paradox. Despite this set-back the λ-calculus could be modified to make it consistent, and turned out to be very useful in computer science. It became the basis of programming languages such as LISP, Miranda, and ML, and indeed is used as a basic tool for the analysis of other programming languages. Functional programming languages such as Miranda and ML are usually typed, and indeed some form of typing is incorporated into most programming languages. It is desirable when specifying a function e.g. f(x,y) to specify also the types of its variables x, y, otherwise errors can be produced by substituting something of the wrong type for one of the variables which will often produce a nonsensical answer. Of course the type theories used in contemporary computer science are *not* the same as Russell's original type theory, but they are descendants nonetheless of Russell's original system. An important link in the chain was Church's 1940 version of the theory of types which was developed from Russell's theory, and which influenced workers in computer science. Davis sums up the situation very well as follows (1988b, p. 322):

‘Although the role of a hierarchy of types has remained important in the foundations of set theory, strong typing has not. It has turned out that one can function quite well with variables that range over sets of whatever type. So, Russell’s ultimate contribution was to programming languages!’

Robinson’s ideas about the different requirements of humans and computers regarding logico-linguistic systems help to explain what happened here. A system whose variables may be of a variety of different types is awkward and inconvenient for humans to handle, nor does it really confer any advantages. Humans can easily in most cases avoid making type errors in formulae, since their intuitive grasp of the meaning which the formula is supposed to convey will prevent them from writing down nonsense. The situation is almost exactly the opposite as regards computers. Computers have no problem at all about handling variables belonging to many different types. On the other hand, without the guidance provided by a strictly typed syntax, a computer can easily produce nonsensical formulae, since it lacks any intuitive grasp of the intended meaning of the formula. One again different systems are suitable for human and computers, so that it is not so in appropriate after all that set theory but not type theory is taught in mathematics departments, and type theory in computer science departments.

I have mentioned so far quite a number of uses of logic in computer science, but in fact there are several more. Logic is a fundamental tool for both program and hardware verification. As regards programming, the influence of logic is not restricted to the specifically logical programming languages such as PROLOG and LISP mentioned above. In fact logic has provided the syntactic core for ordinary programming languages.[2] At an even more fundamental level, the *Begriffsschrift* is the first example of a fully formalised language, and so, in a sense, the precursor of all programming languages.[3]

We must now try to tackle the problem which has arisen from the preceding discussion. The research of Frege and Russell was motivated by philosophical considerations, and they were influenced either not at all, or to a negligible extent, by considerations to do with computing. Why then did their work later on prove so useful in computer science?

Before the work of Frege and Russell, mathematics might be described as semi-formal. Of course symbolism was used, but the symbols were embedded in ordinary language. In a typical proof, one line would not in general follow from the previous ones using some simple logical rule of inference. On the contrary, it would often require a skilled mathematician to ‘see’ that a line followed from the previous ones. Moreover even skilled mathematicians would sometimes ‘see’ that a line in a proof followed from earlier lines when it did not in fact follow. As a result mistaken proofs were often published, even by eminent mathematicians. Moreover the use of informal language often resulted in ambiguities in the concepts employed, which could create confusions and errors.

Of course mathematics is still done today in this semi-formal style, but Frege, in his quest for certainty, thought that he could improve things by a process of formalisation. Concepts would have to be precisely defined to avoid ambiguities and confusions. The steps in a proof would have to be broken down, so that each individual step involved the application of a simple and obviously correct logical rule. By this process, which Frege thought of as the elimination of anything intuitive, he hoped to eliminate the possibility of error creeping in. As he put it (1884, 2): 'The aim of proof is ... to place the truth of a proposition beyond all doubt ...' It was this approach led him to develop a formal system of logic, his *Begriffsschrift* (or concept writing) which is equivalent to present day predicate calculus.

It is now easier to see how the methods which Frege used in his search for certainty in mathematics created a system suitable for use in computer science. What Frege is doing is in effect mechanising the process of checking the validity of a proof. If a proof is written out in the characteristic human semi-formal style, then its validity cannot be checked mechanically. One needs a skilled human mathematician to apply his or her intuition to 'see' whether a particular line follows from the previous ones. Once a proof has been formalised, however, it is a purely mechanically matter to check whether the proof is valid using the prescribed set of rules of inference. Thus Frege's work can be seen as replacing the craft skills of a human mathematician with a mechanical process.[4]

The process of mechanisation in general takes place in something like the following manner. The starting point is handicraft production by skilled artisans. The next step is the division of labour in the workshop in which the production process is broken down into smaller and simpler steps, and an individual worker carries out only one such step instead of the process as a whole. Since the individual steps are now quite simple and straightforward, it becomes possible to get them carried out by machine, and so production is mechanised.

Frege and his successors in the logicist tradition were carrying out an analogous process for mathematics. Mathematical proofs were broken down into simple steps which at a later stage could be carried out by a machine. From a general philosophical point of view, Frege and Russell were engaged in the project of mechanising thought. Since they lived in a society in which material production had been so successful mechanised and in which there was an ever increasing amount of mental (white collar) labour, this project for mechanising thought was a natural one. Moreover it was equally natural that mathematics should be the area chosen to begin the mechanisation process, since mathematics was already partially formalised, unlike other areas of thought.

These considerations perhaps explain why the philosophy of mathematics has assumed such importance within the philosophy of our time. Naturally as well as the thinkers who have pressed forward with the mechanisation of mathematics, there have been those who have objected to this mechanisation, and stressed the human and intuitive aspects of mathematics. Poincaré, Brouwer, Gödel, the later Wittgenstein, and, more recently, Penrose all belong to this trend. Although this line of thought is in many ways reactionary and of course has not halted the advances of mechanisation, there is nonetheless some truth in it, for, as long as mathematics continues to be done by humans at all, it will evidently retain some intuitive characteristics. This is another reason why the logicists although they thought they were building a secure foundation for mathematics and rendering its results certain, were in fact creating a form of mathematics suitable for computer science.

4. How Computer Science has affected Logic

So far we have examined how logical ideas, originating in the logicist programme for the philosophy of mathematics, proved useful in computer science. However the application of these logical concepts to computer science resulted in changes in the concepts themselves. We will next examine some of these changes. The earlier theoretical work of Robinson, Kowalski and others had been concerned with the problem of adapting ordinary classical 1st-order logic for the computer. In the course of actually implementing PROLOG it turned out that use had to made not of classical negation, but of a different type of negation called *negation as failure*. This issue was clarified by Clark in his 1978, which contains a study of this new type of negation. A logic with negation as failure is just one example of a new type of logic known as *non-monotonic logic*. Non-monotonic logic has been developed by computer scientists since the early 1980's, and is an example of an entirely new kind of logic which was introduced as the result of applying logic to computer science.

PROLOG, because of its negation as failure, turned out to be a non-monotonic logic. We must next examine what is a much more profound change – namely PROLOG's introduction of control into deductive logic. As we shall see, negation as failure is really just one consequence of PROLOG's control elements. We can perhaps most easily introduce the topic of logic and control by comparing a passage from Frege with one from Kowalski. In the conclusion of his 1884 book on *The Foundations of Arithmetic*, Frege claims to have made it probable that his logicist programme can be carried out. He goes on to describe what this means as follows (1884, §87, p. 99):

> 'Arithmetic thus becomes simply a development of logic, and every proposition of arithmetic a law of logic, albeit a derivative one. … calculation becomes deduction.'

Let us compare Frege's statement: 'calculation becomes deduction' with the following statement from Kowalski's (1979) *Logic for Problem Solving*, p. 129: 'computation = controlled deduction'. It is clear that Kowalski has added control to Frege's deduction. Let us now try to see what this means.

Suppose we have a PROLOG database (including programs). If the user inputs a query e.g. ?- p(a). (i.e. is p(a) true?), PROLOG will automatically try to construct a proof of p(a) from the database. If it succeeds in proving p(a), the answer will be: 'yes', while, if it fails to prove p(a), the answer will be: 'no' (negation as failure). In order to construct these proofs, PROLOG contains a set of instructions (often called the PROLOG interpreter) for searching systematically through various possibilities. The instructions for carrying out such searches are clearly part of a control system which has been added to the inference procedures of the logic.

One symptom of the addition of control is that logic programs often contain symbols relating to control which would not occur in ordinary classical logic. An example of this is the cut facility, written !. The PROLOG interpreter when conducting its searches automatically backtracks in many situations. In some problems, however, we may not wish the program to carry out so much backtracking which could result in a waste of time, the provision of unnecessary solutions etc. The facility ! controls, in a precise though somewhat complicated way, the amount of backtracking which occurs.

Negation as failure can be defined in terms of !, and another of PROLOG's control elements: fail, a primitive which simply causes the interpreter to fail. A logic program which defines negation as failure is the following:

```
not X  :-  X, !, fail.
not X.
```

The program works like this. Given the task of trying to prove not p, it matches to the leftmost part of the first sentence by setting X = p. It then moves on to trying to prove the first part of right side of the conditional, which with the substitution X = p is simply p. If PROLOG succeeds in proving p, it carries out ! which controls backtracking, and then reaches fail which causes the whole sentence to fail. Because of the operation of !, the interpreter is not allowed to consider the next sentence i.e. not X. Thus PROLOG has failed to prove not p. To sum up: if PROLOG can prove p, it fails to prove not p. If, however, PROLOG fails to prove p, then the first sentence fails *before ! is reached*. Backtracking is not therefore prevented, and so the PROLOG interpreter goes on to consider the second sentence not X. By substituting X = p, this sentence enables it to prove not p. Thus if PROLOG fails to prove p, it succeeds in proving not p. So the logic program does indeed define negation as failure. The interesting point here is that negation as failure is defined using the control elements !, and fail. Thus PROLOG's non-classical negation arises out of its control elements, and the difference between PROLOG and classical logic regarding negation can be seen as a symptom of the more profound difference that PROLOG introduces control into deductive logic.

I will now argue that these developments in PROLOG are a natural extension of the mechanisation process which gave rise to modern logic in the first place. In the previous section I claimed that the work of Frege and Russell can be seen as a mechanisation of the process of checking the validity of a proof. Still their classical logic leaves the construction of the proof entirely in the hands of the human mathematician who has to use his or her craft skills to carry out the task. PROLOG carries the mechanisation process one stage further by mechanising the construction of proofs. In this respect, then, it goes beyond classical logic, and this is also why PROLOG has to introduce control into logic.

A major theme of this paper has been the different conceptual requirements of a computer and of a human mathematician. Further light will be cast on this issue by considering an argument against logicism which Wittgenstein formulated in his later period. This will be the subject of the fifth and final section of the paper.

4. A Criticism of Logicism by Wittgenstein and its Significance

Wittgenstein began his career in philosophy as a student of Russell's, and his first published book, the *Tractatus* of 1921, is full of enthusiasm for Russell's logic. Indeed Wittgenstein claims that the new logic reveals the underlying structure of language. After finishing the Tractatus, Wittgenstein gave up philosophy for about a decade, and engaged in a variety of other activities. He was a village schoolmaster for several years, and also helped with the construction of his sister's mansion in Vienna. Perhaps partly because of these experiences, when he returned to philosophy he developed new views about language which were very different from those of the Tractatus. These were eventually published in 1953, after his death, in the *Philosophical Investigations*. Wittgenstein's later theory is that the meaning of a word is given by its use in a language-game. By a 'language-game' he means some kind of rule-guided social activity in which the use of language plays an essential part. He himself introduces the concept as follows: 'I shall also call the whole, consisting of language and the actions into which it is woven, the "language-game."' (1953, §7, p. 5). And again: 'Here the term "language-*game*" is meant to bring into prominence the fact that the *speaking* of language is part of an activity, or of a form of life.' (1953, §23, p. 11)

Wittgenstein illustrates his concept of language-game by his famous example involving a boss and a worker on a building site. The boss shouts 'slab', for example, and the worker has to fetch a slab. Wittgenstein's point is that the meaning of the word 'slab' is given by its use in the activity carried out by boss and worker.

Wittgenstein also devoted a great deal of thought to the philosophy of mathematics during his later period. His reflections on this subject were eventually published as *Remarks on the Foundations of Mathematics* in 1956, though they were written much earlier. In these remarks, Wittgenstein displays great hostility both to logicism and the use of logic in mathematics. He speaks of '"The disastrous invasion" of mathematics by logic.' (1956, V-24, p. 281), and of 'The curse of the invasion of mathematics by mathematical logic …' (1956, V-46, p. 299)

These harsh words about logic are of course connected with his new views of language and meaning. Wittgenstein now thought that it was absurd to claim that the whole of mathematics could be reduced to a single system such as *Principia Mathematica*. On the contrary mathematics consists of a whole variety (or motley) of techniques carried out in different language-games; as he says: ' … what we call mathematics is a family of activities with a family of purposes …' (1956, V-15, p. 273). These mathematical language-games are also connected with the language-games of everyday life, as, for example, arithmetic may be used on the building site.

From this point of view, Russell's *Principia Mathematica* does not provide a foundation for mathematics, but is simply a new piece of mathematics, a new mathematical language-game. As Wittgenstein says (1956, III-4, p. 146):

'But still for small numbers Russell does teach us to add; for then we take the groups of signs in the brackets in at a glance and we can take *them* as numerals; for example '*xy*', '*xyz*', '*xyzuv*'.

Thus Russell teaches us a new calculus for reaching 5 from 2 and 3; and that is true even if we say that a logical calculus is only – frills tacked on to the arithmetical calculus.'

In my view this is partly right and partly wrong. I agree with Wittgenstein that mathematical logic is a new mathematical calculus but does not provide a foundation for the rest of mathematics as the logicists thought it would. On the other hand Wittgenstein clearly thought that this new mathematical calculus was useless, and that 'a logical calculus is only – frills tacked on to the arithmetical calculus.' The passages I have quoted from Wittgenstein were written in the period 1939-44, and it was not unreasonable at that time to think that the formal systems produced by the logicists would be useless. Contrary, however, to Wittgenstein's expectations, these same logicist systems turned out to be very useful for computer science. I next want to argue that Wittgenstein's later theory of meaning, with which I largely agree, helps to explain why formal logic has proved valuable in computer science.

Let us return to the example of the boss and the worker on the building site. If the boss shouts 'slab', and the worker fetches a slab, then we can surely say that the worker has understood the meaning of the word 'slab', because he has acted appropriately, or, in Wittgenstein's terminology, has made the right move in the language-game. It is interesting in this context to consider the historical example of the Norman conquest of England. The Normans spoke French and the serfs on the estates which they had conquered spoke English. This must have created difficulties for the Norman overlords in giving orders to their serfs. Thus the lord might have said: 'Donnez-moi un de vos moutons', while the serf would only have understood: 'Give me one of your sheep'. Now the serfs would have lacked the educational facilities to learn French, and it might indeed have been in their interest to pretend to understand less French than they really did. Thus the Norman overlords must have been forced to learn English to be able to give orders to their serfs. This may perhaps explain why the speaking of French disappeared in England over the centuries, though not before it had modified the English language in many ways. Let us now see how all this might be applied to computers.

Several philosophers have denied that computers can understand language, but, if we adopt Wittgenstein's later theory of meaning, it looks as if they were wrong to do so. In Wittgenstein's example, we have only to replace the worker by a computer. I can certainly give orders to my computer, by, for example, typing in a program. If the computer carries out my instructions, surely it is sensible to say, just as in the human case, that it has understood those instructions. The computer and I are playing a language-game. Both of us are using the symbols involved correctly, and so, by Wittgenstein's criterion, we both understand the meaning of those symbols. In a similar fashion, we can say that dogs understand at least a few words of human language. Thus if my dog performs the appropriate actions when I say: 'sit', 'beg' and 'fetch', we can say that he understands the meaning of these three words. There is, however, a very significant difference between dogs and computers as regards language. Dogs can only understand commands consisting of essentially of one symbol (which may in practice be composed of a few words, e.g. sit down). Grammar is quite beyond them. Computers by contrast are much more finicky about grammar than humans. Humans often speak ungrammatically, and their utterances can usually be understood nonetheless. This applies even to the greatest of writers. Thus Shakespeare in describing the wound which Brutus gave Caesar wrote: 'This was the most unkindest cut of all' (Julius Caesar, act III, scene ii, line 188). Shakespeare's line is surely ungrammatical, and yet it is perfectly comprehensible to us. By contrast my computer has, all too frequently, failed to understand one of my instructions merely because that instruction has contained some trivial syntactical error!

This brings us back to the central theme of the different linguistic requirements of computers and humans. Computers find it easiest to understand very precise formal languages which are difficult for humans. The language which is easiest for computers is machine code which is quite opaque to all but a few highly trained humans. Conversely humans find loose informal natural (for humans) languages very easy to understand, and these cannot be understood at all by computers. This is the point of the analogy with the French-speaking Norman lords, and their English-speaking serfs. We humans are in the position of the Norman lords with regard to our computer serfs. These computers will do wonderful things for us, but we have to give them their orders in a language they can understand. This is a difficult task since computers cannot cope with languages which are easy and natural for us. This is where the language of formal logic has proved to be helpful. This language is intermediate between the machine code which is natural for computers, and an everyday language such as English which is natural for humans. Formal logic has the precise syntax which makes its sentences accessible to computers, while it has sufficient resemblance to ordinary language to be comprehensible to humans after a little training. Even within logic itself, there are, as Robinson pointed out in the passages quoted above in section 3, some formulations which are more suitable for computers and others that are more suitable for humans. Thus the clausal form of logic with its single, but complicated, rule of inference is more suitable for computers, whereas other systems of logic with several, but much simpler, rules of inference are more suitable for humans. In general terms, however, formal logic is a language system somewhat intermediate between those which are most suitable for computers, and those which are most suitable for humans. It is thus very helpful in facilitating human-computer interaction, and this I would see as the fundamental reason why it has proved so useful in computer science.

Frege in the *Begriffsschrift* where he introduces a formal system for logic for the first time explains the differences between his system and ordinary language by means of a striking analogy (1879, p. 6):

> ‘I believe that I can best make the relation of my ideography to ordinary language clear if I compare it to that which the microscope has to the eye. Because of the range of its possible uses and the versatility with which it can adapt to the most diverse circumstances, the eye is far superior to the microscope. Considered as an optical instrument, to be sure, it exhibits many imperfections, which ordinarily remain unnoticed only on account of its intimate connection with our mental life. But, as soon as scientific goals demand greater sharpness of resolution, the eye proves to be insufficient. The microscope, on the other hand, is perfectly suited to precisely such goals, but that is just why it is useless for all others.’

Similarly the language of formal logic is suited to the scientific goal of communicating with computers, since this task demands great precision of expression. It is less suited, however, to the task of communicating with other human beings.

The idea that different languages are suited to different purposes is already to be found in a reputed saying of the multi-lingual emperor Charles V. He is supposed to have said that he found French the most suitable language for talking to men, Italian for women, Spanish for God, and German for horses. If he had lived today, he could have added that the language of formal logic was the most suitable for talking to computers.

Notes

* I have been researching into the connections between philosophy and computer science for several years now. The specific focus on philosophy of mathematics arose out of discussions with Yuxin Zheng during his visit to King's College London from April to September 1997. This visit was made possible by Yuxin Zheng's receipt of a British Academy K.C.Wong Fellowship, and a travel grant from the Open Society Institute. I would like to thank the British Academy and the Open Society Institute for the support, which made this collaborative research possible, as well as Yuxin Zheng for many helpful suggestions.

Earlier versions of some of the ideas in this paper were presented at the Annual Conference of the British Society for the Philosophy of Science in September 1998, at the Logic Club, Department of Philosophy, University of California, Berkeley in November 1998, at a conference on Philosophy and Computing at King's College London in February 1999, and at the Applied Logic Colloquium at Queen Mary College London in November 1999. I am very grateful for the comments received on these occasions, and particularly for some points made by Martin Davis at Berkeley, one of which is mentioned in footnote 3 below.

I would also like to thank a number of computer scientists with whom I discussed this problem and who made many helpful suggestions, which have been incorporated in the paper. These include James Cussens, Mark Gillies, Stephen Muggleton, David Page, and Ashwin Srinivasan.

1. For further details of Frege's logicism, and the impact on it of Russell's paradox, see Gillies (1982).

2. I owe this point to Mark Priestley who is researching into this topic at the moment.

3. I owe this point to Martin Davis. See his 1988b, 316.

4. It should be stressed that this is my way of viewing Frege's work, and that Frege himself would not have seen things in this light. (I owe this point to Carlo Cellucci.)

References

Church, A. (1932). A Set of Postulates for the Foundation of Logic, *Annals of Mathematics*, **33**, pp. 346-66.

Church, A. (1940). A Formulation of the Simple Theory of Types, *Journal of Symbolic Logic*, **5**, pp. 56-68.

Clark, K. (1978). Negation as Failure. In H.Gallaire and J. Minker (eds.), *Logic and Data Bases*, Plenum Press, pp. 293-322.

Curry, H.B. (1942). The Inconsistency of Certain Formal Logics, *Journal of Symbolic Logic*, **7**, pp. 115-17.

Davis, M. (1988a). Mathematical Logic and the Origin of Modern Computing. In Rolf Herken (ed.), *The Universal Turing Machine. A Half-Century Survey*, Oxford University Press, pp. 149-74.

Davis, M. (1988b). Influences of Mathematical Logic on Computer Science. In Rolf Herken (ed.), *The Universal Turing Machine. A Half-Century Survey*, Oxford University Press, pp. 315-26.

Frege, G. (1879). *Begriffsschrift, Eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*, English translation in Jean van Heijenoort (ed.), *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*, Harvard University Press, 1977, pp. 1-82.

Frege, G. (1880-1). Boole’s Logical Calculus and the Concept-Script. English translation in *Gottlob Frege: Posthumous Writings*, Blackwell, pp. 9-52.

Frege, G. (1884). *The Foundations of Arithmetic: A Logico-Mathematical Enquiry into the Concept of Number*. English translation by J.L.Austin, Blackwell, 1968.

Frege, G. (1893). *Grundgesetze der Arithmetik, Begriffsschriftlich abgeleitet*. Vol. I. (1893) and Vol. II. (1903). Reprinted by G.Olms, 1962.

Frege, G. (1902). Letter to Russell. English translation in J. van Heijenoort (ed.) *From Frege to Gödel*, Harvard University Press, 1967, pp. 127-8.

Gillies, D.A. (1982). *Frege, Dedekind, and Peano on the Foundations of Arithemtic*, Van Gorcum.

Gillies, D.A. (1996). *Artificial Intelligence and Scientific Method*, Oxford University Press.

Jevons, W.S. (1870). On the Mechanical Performance of Logical Inference, *Philosophical Transactions of the Royal Society*, **160**, pp. 497-518.

Kleene, S.C. and Rosser, J.B. (1935). The Inconsistency of Certain Formal Logics, *Annals of Mathematics*, **36**, pp. 630-36.

Kowalski, R.A. (1979). *Logic for Problem Solving*, North-Holland.

Muggleton, S. (ed.) (1992). *Inductive Logic Programming*, Academic Press.

Rosser, J.B. (1984). Highlights of the History of the Lambda-Calculus, *Annals of the History of Computing*, **6**(4), pp. 337-9.

Robinson, J.A. (1965). A Machine-Oriented Logic Based on the Resolution Principle, *Journal for the Association for Computing Machinery*, **12**, pp. 23-41.

Russell, B. (1959). *My Philosophical Development*, George Allen and Unwin.

Russell, B. (1967). *Autobiography. Volume 1*, George Allen and Unwin.

Russell, B., and Whitehead, A.N. (1910-13). *Principia Mathematica*, Cambridge University Press.

Wittgenstein, L. (1921). *Tractatus Logico-Philosophicus*. English translation by D.F.Pears and B.F.McGuinness, Routledge & Kegan Paul, 1963.

Wittgenstein, L. (1953). *Philosophical Investigations*. English translation by G.E.M.Anscombe, Blackwell, 1967.

Wittgenstein, L. (1956). *Remarks on the Foundations of Mathematics*. English translation by G.E.M.Anscombe, Blackwell, 3rd edition, revised and reset, 1978.