Robert Kowalski: A Short Story of My Life and Work
April 2002 - revised June 2015 and June 2025

Schooldays

I was born in Bridgeport, Connecticut, USA, on 15 May 1941. I went to Saint Michael's, a Catholic primary school in Bridgeport, attached to a Polish parish, but I didn't learn much Polish. My parents would speak Polish when they didn't want us children to understand. I was a good student, but not outstanding. There were 60 children in my class, 17 girls and 43 boys.

I went to Fairfield Prep, a boys-only Jesuit High School. It took half an hour's walk and two buses each way to get to school. In my second year, my Latin teacher, Father Walsh, trained four of us to compete in Latin sight-translation contests. The task was to translate a previously unseen Latin text into English without a dictionary.

The skill needed for the task was the ability to guess the most coherent English translation of the Latin text, constrained by our limited knowledge of Latin and of the subject matter of the text. Many years later, I learned that the required technique - of generating assumptions to solve problems subject to constraints - is called "abduction". Our team took first prize in New England.

I also started to have an intellectual life outside of school. I started reading Freud, Ruth Benedict and Joad's "Guide to Philosophy". I found these books very exciting, but they undermined my Catholic upbringing. I still believed there had to be a single truth, and I wanted to find out what it was. I also wanted to get away from home and to be free to come and go as I pleased.

University of Chicago and University of Bridgeport

For these reasons, I was attracted to the University of Chicago, and intellectually I was not disappointed. Among the other great ideas to which I was exposed in my first year, I was introduced to mathematical logic; and it seemed to me that it might lead the way to the truth.

I got "A"s in all my subjects, except English writing skills, in which I got a "D". I couldn't understand what was wrong with my writing, but I was determined to improve it. I reasoned that if I could understand and solve the problems with my writing, then I would do even better in my other subjects.

At the beginning of my second year, I began to find the social life at the University of Chicago very difficult. To make matters worse, I was overwhelmed by the assigned reading of Gibbon's "Decline and Fall of the Roman Empire". Reading about the seemingly endless *impedimenta* that the Roman troops had to carry to their battles was the last straw. I left Chicago in November of my second year.

I spent the rest of that academic year trying to find myself. I signed up for an expedition to find gold in Honduras, only to abandon the journey somewhere in Ohio. I worked for half a year in a chemical factory as a quality control inspector.

The following academic year, I enrolled at the University of Bridgeport and commuted, with my brothers, Bill and Dan, from my parents' home in Milford. Being a student at the University of Bridgeport was easy after Chicago. I decided to major in Mathematics.

I couldn't get a scholarship at first. So, I supported myself by working in Peoples Savings bank in the evening, processing paper tapes of the day's banking transactions. I discovered how to cut the time of the work in half, mostly by performing multiple tasks in parallel. But, because I was paid by the hour, I then had the more difficult problem of preventing my pay from also being cut in half.

When I went to the scholarship office to argue my case for getting a scholarship, I was turned down because I didn't participate in the extracurricular life of the University. The fact that I was busy working to support my studies was not deemed to be relevant. I was told that the only solution was to join a student club. But, because I had neither the interest nor the time to join any of the existing clubs, I advertised in the student newspaper to announce the formation of a new club for people who didn't want to belong to any clubs. Soon afterwards, I was awarded a scholarship, which allowed me to quit my job at Peoples Savings and to work full time as a student.

Academically, after getting my scholarship, I was left with plenty of time for independent study. Mostly I studied Logic. My favourite title was "The Meaning of Meaning" by C. K. Ogden and I. A. Richards. I also worked on the problems with my English and started to make big improvements in my writing style.

I took the Graduate Record Examination in Mathematics and scored higher than any previous student at the University of Bridgeport. The comparison isn't completely fair, because I had taken the exam a year earlier, and some of the questions were virtually the same. But it had the desired effect. I won Woodrow Wilson and National Science Foundation Fellowships for graduate study. They published my photograph in the Bridgeport Post.

Stanford and University of Warsaw

I went to Stanford to study for a PhD in Mathematics, but my real interest was Logic. I was still looking for the truth, and I was sure that Logic would be the key to finding it. My best course was axiomatic set theory with Dana Scott. He gave us lots of theorems to prove as homework. At first my marks were not very impressive. But Dana Scott marked the coursework himself, and he wrote lots of comments. My marks improved significantly as a result.

Jon Barwise was among the other students entering Stanford as a PhD student that year, in 1963. We were friends, but also competitors. He discovered that Stanford had an exchange program with the University of Warsaw, noted for its work in mathematical logic. We both applied for the program. I got in, but he didn't, because he was judged to be too young.

The exchange program started with an intensive Polish course at the end of the summer. I didn't receive any formal credit for the courses I attended at the University of Warsaw, but I didn't have to take any exams, and I could focus exclusively on the logic courses. I took courses with Helena Rasiowa, Andrzej Grzegorczyk and Andrzej Mostowski.

I spent much of my time on extracurricular activities. I met and visited my Polish relatives, including my grandparents, who lived near the Soviet border. I also met my future wife, Danusia, a student in the Mathematics Department at the University. After only a few months, we got married, in February 1965.

Before going to Poland, I had no interest in politics or current affairs. But I had been brought up during the Cold War, and the Jesuits were rabidly anti-communist. I expected Poland to be totally devoid of freedom, and I was surprised that it wasn't nearly so bad. However, I didn't fully appreciate how much worse it had been soon after the war, and how much worse it was in many other countries in the Soviet bloc. I became much more interested and more educated about such matters after I retired.

When I returned to Stanford at the beginning of the next academic year, I found it hard to convince myself that studying complex variables and recursion theory would lead me to the truth, and I was upset by the war that was developing in Vietnam. I became one of the organizers of the protest movement and found my niche dreaming up ideas and convincing other people to put them into action. I had the idea of dropping anti-war leaflets from airplanes. My flatmate, Ray Tiernan, a childhood friend from Bridgeport, organized the bombing campaign.

Ray and I both went up in the first few bombing missions. We practiced over Stanford and other places in the San Francisco area. Our first attempt nearly ended in disaster, when the leaflets got caught in the tail of the plane.

Our goal was to bomb the Rose Bowl football game in Los Angeles. Ray and I worked out an elaborate plan to conceal the registration number on the side of the plane, covering it with a false number, which we would rip off during our getaway in mid-fight. Unfortunately, when we landed to cover the number in the Mohave Dessert, before the game, the plane burst a tire, and we were too late to get to the Rose Bowl in time. We bombed Disney Land instead.

Eventually, Ray was arrested on our last mission, when he went up without me.

Puerto Rico and Edinburgh

I left Stanford in the middle of the academic year. Fortunately, I had taken enough courses to leave with a Master's degree. I applied to teach Mathematics at various universities, mostly outside the United States. I eventually accepted a job as Assistant Professor and Acting Chairman of the Mathematics and Physics Department at the Inter-American University in San Juan, Puerto Rico.

I was excited by the prospect of living and working in Puerto Rico, and I studied as much Spanish as I could before leaving. I don't have very clear memories of the one year I worked in Puerto Rico, but it convinced me that I had to start again and finish a PhD if I wanted my colleagues to take me as seriously as I desired. I applied to several universities in Great Britain. Eventually I accepted the offer of a Fellowship from Bernard Meltzer, Head of the Meta-mathematics Unit at the University of Edinburgh.

In the meanwhile, my first daughter, Dania, was born. I left Puerto Rico, knowing less Spanish than when I started, because everyone wanted to practice their English.

We used our savings from Puerto Rico to buy a car when we got to England, and we drove it to Edinburgh, after a detour to Poland and Italy, arriving in October 1967. I remember arriving at the doorway of the Meta-mathematics Unit, and seeing the sign: "Department of Computer Science". My heart sank. I hated computers, technology and engineering more generally, but I decided I would stick it out, get my PhD as quickly as possible, and resume my search for the truth.

Bernard Meltzer was working on the automation of mathematical proofs. Although I wasn't convinced about the value of the research topic, I was determined not to drop out of another PhD. I was lucky. Alan Robinson, the inventor of resolution, was in Edinburgh spending a year's sabbatical. He had just finished a paper on semantic trees applied to theorem proving with equality. Pat Hayes, another fresh PhD student, and I studied Alan's paper in minute detail. A few months later, I wrote my first research paper, on semantic trees, with Pat as coauthor. Pat, in the meanwhile, visited John McCarthy at Stanford, and together they wrote their famous paper on the situation calculus. The two papers were published together in 1969 in the Edinburgh Machine Intelligence Workshop series.

I finished my PhD in just over two years; and, with a second daughter, Tania, born in Edinburgh, I was free to start a new life. I decided to look for an academic job in the UK. But it wasn't as easy as I had hoped. I was eventually interviewed for two jobs – one a Fellowship at Pembroke College at Oxford University, the other a Lectureship in the Mathematics Department at the University of Essex.

I knew I wasn't going to be offered the Fellowship at Pembroke College when the Master of the College introduced me to one of the Fellows as "Mr. Kowalski from the University of Bridgeport". I didn't get the other job either. I had to settle instead for a postdoctoral fellowship in the Meta-mathematics Unit in Edinburgh. My third daughter, Janina, was born that same year.

The postdoctoral fellowship gave me plenty of time to explore my real interests. In those days they were mainly in philosophy of science and epistemology. I remember reading and being influenced by Lakatos' "Proofs and Refutations", Nelson Goodman's "Fact, Fiction and Forecast", and Quine's "Two Dogmas of Empiricism". I didn't fully appreciate it at the time, but in retrospect I was lucky that Bernard encouraged me to explore these broader interests.

Lakatos documented how the history of Euler's theorem could be viewed as a repeated cycle of conjectured theorems, attempted proofs, counterexamples, and revised conjectures. My reading of Lakatos reinforced my own experience with research on automated theorem-proving. It encouraged me in the view that it is both harder and more important to identify whether a theorem is worth proving than it is to prove the theorem, whether or not the theorem is worth proving. The downside is that it is easy to make a claim about a supposed theorem that cannot later be justified.

Other developments were attracting attention in the world of Logic and Artificial Intelligence. Attacks against Logic were being launched from MIT, with declarative representations declared as bad, and procedural representations as good. In the face of these attacks, many of the researchers

working on Logic for theorem proving moved into other areas. But I couldn't accept the view that Logic was dead.

I had been working on a form of resolution, called SL-resolution, with Donald Kuehner, who had been one of my mathematics teachers at the University of Bridgeport. (I visited Donald on one of my visits back home and convinced him to come to Edinburgh to do his PhD. Like me, he thrived in the British PhD environment.)

SL-resolution uses logic in a goal-directed way. We pointed this out at the end of our paper, and I set out to convince my colleagues that the goal-directed approach of SL-resolution reconciles logic with the procedural approach advocated at MIT.

In the summer of 1971, I received an invitation from Alain Colmerauer to visit him in Marseille. He was working on the automation of question-answering in natural language, and he had already done significant work on machine translation between English and French for weather forecasting in Canada. When he invited me to Marseille, he was using logic to represent the meaning of natural language sentences and using resolution to derive answers to questions.

Alain was interested in my work on theorem proving and on SL-resolution in particular. My family and I stayed with him and his family for several days in their small flat. We worked late into the night, discovering how to use logic to represent grammars and how to use theorem provers for parsing. We saw that some theorem provers, like hyper-resolution, behaved as bottom-up parsers, deriving sentences and other phrases from sequences of words. Other theorem-provers, like SL-resolution, behaved as top-down parsers, decomposing the problem of recognising or generating sentences into subproblems of recognising or generating other phrases and ultimately sequences of words.

Alain invited me to visit him again for a longer period of two months the following summer of 1972. It was during that second visit that logic programming, as I see it, was born.

Logic programming

In its simplest form, a logic program is a set of facts and rules of the form *if conditions then conclusion*. The simplest way to understand such a logic program is to think of using it to reason *bottom-up* (or *forwards*), to derive new facts as logical consequences of the given facts. For example, to derive *bob likes bob* from the fact that *bob likes logic* and the rule *if X likes logic then bob likes X.*

In general, bottom-up reasoning starts with given facts, and repeatedly unifies facts with the *conditions* of rules, deriving new facts that are instances of the *conclusions* of the rules. Given a set of goals to be solved, reasoning terminates when all the goals unify with the given or derived facts directly, or when no more new facts can be derived.

But logic programs can also be used to reason *top-down* (or *backwards*), to reduce goals to subgoals. For example, given the goal to find an *X* such that *bob likes X*, it reduces the goal to the subgoal of finding an *X* such that *X likes logic*. Top-down reasoning starts with a given goal, and repeatedly unifies goals with the conclusions of rules, reducing the goals to subgoals that are instances of the

conditions of the rules. Different rules whose conclusion unifies with the same goal are different ways of trying to solve the goal. Reasoning terminates successfully when all the subgoals derived from one way of solving the initial goal unify with the given facts directly.

Used to reason top-down, logic programs can be interpreted as computer programs, in which rules behave as goal-reduction procedures. To emphasize this use of backward reasoning, rules in logic programs are usually written backwards, in the form *conclusion if conditions*. For example, with top-down reasoning, the rule *bob likes X if X likes logic* can be used as a procedure: *to find an X such that bob likes X, find an X such that X likes logic.*

I have done my best to document the birth of logic programming, most recently in an article about the Marseille-Edinburgh Connection. In general terms, it is probably fair to say that my ideas were more theoretical and perhaps more philosophical than Alain's, and Alain's ideas were more practical. Alain once even referred to my work as "poetic", in a sense that I'm sure was intended to be complementary. Alain's work led to the design and implementation of the logic programming language Prolog in the same summer of 1972.

Back in Edinburgh, I embarked excitedly on recruiting converts to the logic programming cause, with Maarten van Emden and David Warren being the most prominent of the earliest recruits. Initially, Bob Boyer and J Moore were also attracted to the idea, and it led to their subsequent work on proving properties of programs written in Lisp.

I worked closely with Maarten, and we discovered that computation in logic programming can be viewed in two very different, but equivalent ways: It can be viewed as solving a goal by proving that the goal is a theorem which is logically implied by the program viewed as a set of axioms. But it can also be viewed as solving a goal by showing that the goal is true in a unique minimal model that makes the program true.

The facts that are true in the minimal model are all the facts that can be derived by reasoning bottom-up from the facts and rules in the program. We had no idea at the time that the minimal model view would later become the dominant view of the semantics of logic programming.

I didn't work so closely with David, because he was more focused on such practical matters as improving the Prolog language design and implementation, and I was happier just focussing on the theory. David wrote the first Prolog compiler, and he was responsible for developing many of the features found in Prolog today.

Edinburgh at that time was a world-renowned centre of research in Artificial Intelligence, and I benefited from the opportunity to discuss ideas with other researchers, including Alan Bundy, Rod Burstall, Michael Gordon, Donald Michie, Robin Milner and Gordon Plotkin, who were working in Edinburgh at that time, and with such distinguished visitors as Aaron Sloman and Danny Bobrow.

We also had visitors who were attracted to the logic programming idea. They included Luis Pereira from Lisbon, Sten Ake Tarnlund from Stockholm, Peter Szeredi from Budapest and Maurice Bruynooghe from Leuven. I travelled extensively in Europe, giving talks about the new cause.

Before leaving Edinburgh and before finishing my work on automated theorem proving, I developed the connection graph proof procedure. Until now there has been no proof of its completeness and only counterexamples to certain limiting cases. The history of unsuccessful attempts to prove completeness reinforced my conviction that identifying theorems is more important than proving them. Or to put it a little differently, truth is more important than proof.

Imperial College 1974-1981

Sometime in 1973 or 1974, I was invited to apply for a Readership in the Theory of Computing in the Department of Computing and Control at Imperial College in London. A British Readership is like a tenured Associate Professorship at an American university, with the additional feature of being primarily for research. It was a great opportunity to advance my career, and it had the added attraction of being in London, one of the most cosmopolitan and desirable places to live in the world. I jumped at the chance.

It took about a year to confirm my appointment, partly because there was another strong candidate, and partly because doubts were raised about my suitability for the post. I started in January 1975, and I was assigned to teach a course on formal languages and automata theory immediately upon my arrival. I knew next to nothing about automata theory, and I had little interest in it. Fortunately, Keith Clark, then working as a lecturer at Queen Mary College in London, was a keen convert to logic programming, and he provided me with guidance for the course. I muddled through, but it was an unhappy introduction to the Department.

However, it wasn't long before I was able to redirect my teaching to the areas of logic, logic programming, and artificial intelligence, which were central to my interests. I had to cheat a little in the beginning, for example by setting students the problem of writing a Prolog interpreter in Cobol, as a programming exercise in the comparative programming languages course.

My first few years at Imperial College were focused on learning enough of the basics of Computing to do my teaching, writing my book "Logic for Problem Solving" and promoting the cause of logic programming in general. In this latter pursuit, I was especially fortunate in recruiting Chris Hogger and in helping to bring Keith Clark into the Department. I also organised the first Logic Programming Workshop, at Imperial College in 1976.

The book was very hard work, and it seemed to take forever. To make matters worse, in those days I didn't type, and I had to rely entirely on others to do all the typing. The final draft was a camera-ready copy produced on a line printer, using ancient word-processing technology. When I finished, I knew it would be a long time before I wrote another book.

Alan Robinson invited me to Syracuse University in 1981, and during my visit I collaborated with Ken Bowen on amalgamating object level and meta-level logic programming. Our goal was to combine the two levels of language somewhat like the way that natural language uses sentences both to talk about the world and to talk about sentences themselves.

We had the practical objective of using logic programs to define and implement other logics and other programming languages without introducing such inconsistencies as the liar's paradox: *this*

sentence is false. Moreover, we were also inspired in part by the fact that Prolog includes such an amalgamation capability, although its logic was not very well understood.

I recently revisited this early work on amalgamation by incorporating it into the language Logical English (LE), which is syntactic sugar for the logical core of Prolog. We use amalgamation in LE to represent propositional attitudes, such as belief and obligation, and speech acts, such as telling, asking and denying. Here is a silly example:

> *Bob likes a person*
> *if the person likes logic and Bob believes that the person likes logic.*
> *Bob believes a proposition*
> *if Bob trusts a person and the person tells Bob that the proposition is true.*
> *Bob trusts Mary.*
> *Mary tells Bob that Bob likes logic is true.*

In 1978 I started a course of logic lessons for 12-year-old children at my daughters' middle school. We used Prolog to represent and solve logic problems on the Departmental computer, using a pay phone connection. The connection would be lost when our coins ran out.

Once we demonstrated the feasibility of teaching logic to children, I succeeded in getting support from the Science Research Council to develop microProlog, a microprocessor implementation of Prolog, for use in schools. The project employed Frank McCabe to do the implementation and Richard Ennals to develop and test the teaching materials.

Perhaps the worst thing about my work in those days was the fact that the MSc. course lasted throughout the summer and deprived me of the opportunity to get away from my normal commitments. Earlier, both when I was a student and when I was a postdoctoral researcher in Edinburgh, I relied upon such opportunities to clear my mind of details and to explore broader intellectual horizons.

The Fifth Generation Project and the Alvey Programme

Then everything changed. In 1981, MITI in Japan announced the Fifth Generation Project, whose stated goal was to leapfrog IBM in ten years' time. The governments of Britain, France and Germany were invited to participate, and logic programming was to play a dominant role. At the time, our group at Imperial College was recognised as the leading centre for logic programming internationally, and it was the obvious choice for a British centre to collaborate or compete with Japan.

The British government responded by forming a committee chaired by John Alvey, the Director of Research at British Telecom. The academic community, led by the Science Research Council, formed its own committees to advise the Alvey Committee. I was enlisted along with many others to help draft recommendations for the British response. Although I was not yet a full Professor, I was the most senior academic in Britain arguing the case for logic programming,

It was chaos. Academics argued with fellow academics, industrialists argued both with academics and other industrialists - all presided over by the British civil service. We all wanted a slice of the action. Some of us went further, arguing that we should follow the lead of the Fifth Generation Project and focus on logic programming to the detriment of other areas. That was a big mistake.

My position in the Department deteriorated, as I came into conflict with my academic colleagues, who wanted the government to focus on mainstream software engineering and formal methods. It wasn't much better on the national level, where logic programming was seen as a newcomer (and some would say an intruder) on the Computing scene. In the end, by the time the Alvey Committee produced its recommendations, virtually every area of Computing and related Electronics was singled out for special attention, except for logic programming, which received hardly a mention.

The British government decided to decline the Japanese invitation and to go it alone. The "Alvey Programme" was established, and eventually, after much further debate, logic programming was identified, along with all the other areas, as worthy of special promotion. By around 1985, as a result of the Alvey Programme and with a lot of help from Keith Clark, the logic programming group at Imperial College expanded to approximately 50 people, including PhD students, research assistants, academics and support staff. These were supported by thirteen separate, three-year research grants. The administrative and managerial burden was enormous. For my reward - or consolation - I was promoted to a full Professorship in 1982.

My position in the Department and that of the logic programming group were strained. We wanted to establish ourselves as a separate entity, and the Department wanted to keep us in our place. In the autumn of 1987, I took a six-month leave of absence, to get away from it all.

Logic Programming for Legal Reasoning

From 1981 to 1987, my professional life was dominated by academic politics. It was not an area of activity to which I was naturally drawn, but an area into which I was pushed by events around me. Inevitably the politics interfered with my research.

Fortunately, I was able to continue to make contributions to research by working with PhD students. I worked with Marek Sergot on the application of logic programming to legal reasoning, and along with several other members of the group, including a new PhD student, Fariba Sadri, we investigated the formalisation of the British Nationality Act as a logic program. In the atmosphere of the Alvey era, even this caused controversy: Some of our critics accused us of racism, because it was supposed that the work must have been supported by the British government to further its racist policies. I ended up writing to the Guardian, a national newspaper, to try to clear our names.

Fortunately for us, the assessment of our work improved over time. In 2021, Marek, Fariba and I received the Inaugural Stanford University CodeX Prize. The citation states: "The authors' seminal article, "The British Nationality Act as a Logic Program," published in 1986 in the Communications of the ACM journal, is one of the first and best-known works in computational law, and one of the most widely cited papers in the field."

The Event Calculus

Marek and I also worked on the representation of temporal reasoning, developing a calculus of events, in the spirit of McCarthy and Hayes' situation calculus, but focusing on the way in which events initiate and terminate facts, representing local states of affairs. This work became a major thread of a European Community research project, which explored, among other applications, an application to air traffic flow management. Murray Shanahan further developed the event calculus and featured it in his book about the *frame problem*, which is the problem of how to reason about the passage of time, given that most facts about the world persist from one state of the world to the next.

The event calculus solves the frame problem by representing and reasoning about how events initiate and terminate facts. The solution can be expressed by means of a single *frame axiom*, which can be written (in Logical English) as a meta-level logic programming rule:

> *a fact holds at a time T2*
> *if the fact is initiated by an event that occurred at a time T1 that is earlier than T2*
> *and it is not the case that the fact is terminated by an event that occurred at a time T*
> *that is on or after T1 and that is before T2*.

For example, an event in which an agent gives an object to another agent at a time initiates the fact that the other agent possesses the object at the time, and it terminates the fact the agent possesses the object at the time. So, if Bob possesses a book and he gives it to Mary, then Mary possesses the book from the time of Bob's giving the book to Mary, and until she gives the book to someone else.

The event calculus solution of the frame problem depends on interpreting negation as *negation as failure*, so that a condition of the form *it is not the case that p* is understood as meaning that *it cannot be shown that p.*

Negation as failure can be understood as exploiting the c*losed world assumption* that the logic program contains all the information about its subject matter. For example, it is natural to assume that a railway timetable contains all the information about train journeys within its geographical area. So, if is cannot be shown that there is a train connection between two places at a time, then there is no train connection between the two places at the time.

Keith Clark showed in a famous paper presented at a Logic and Databases Workshop in 1977 in Toulouse that *negation as failure* can be justified by rewriting logic programs as definitions in which, roughly speaking, *if* is shorthand for *if and only if*. He showed that the structure of a proof of *not p* by negation as finite failure using the if-form of logic programs is similar to the structure of a natural deduction proof that *not p* is a theorem that is logically implied by the if-and-only-if form of the program.

Later, other authors developed various minimal model semantics for negation as failure. The most notable of these were the well-founded semantics of Van Gelder, Ross and Schlipf and the stable model semantics of Gelfond and Lifschitz. In these semantics for negation, *not p* holds if *not p* is true in an appropriate minimal model.

Integrity constraints

Fariba and I worked on integrity checking for deductive databases. This work was motivated by an understanding, shared with many other researchers, that data in relational databases can be understood as facts in logic programs, and that rules in logic programs can be understood as defining abstract data in terms of more concrete data. In those days, such logic programs were known as deductive databases. These days, they are better known as Datalog.

However, it was not until I attended a talk by Herve Gallaire and Jean-Marie Nicolas at the 1977 Logic and Databases Workshop that I learned that there are two kinds of rules in database systems: logic programming rules, which define or *describe* data, and integrity constraint rules, which constrain or *prescribe* data.

It is easy to confuse the two kinds of rules. For example, the sentence *everyone on the bus has a ticket to ride the bus* can be understood as a *description* of the passengers on the bus. Or it can be understood as a *prescription* that needs to be satisfied by the passengers on the bus. Both interpretations of the sentence can be expressed with the same rule-like syntax, *if a person is on the bus then the person has a ticket to ride the bus*. But the semantics of logic programming interprets the rule as a description, and the semantics of integrity constraints interprets the rule as a prescription.

I later discovered that this confusion about the meaning of rules can help to explain apparent errors of human reasoning in the Wason selection task, which is possibly the most famous psychological experiment in deductive reasoning. In the standard version of the task, participants are given four cards lying on a table, with numbers on one side of the cards and letters on the other side. They are also given a rule that *if a card has a vowel on one side, then it has an even number on the other side.* The task is to determine which cards need to be turned over to test whether the rule is true or false. Typically, only 10 % of the participants reason in accordance with the standards of classical logic.

Cognitive psychologists have proposed a wide variety of explanations for human performance on the Wason task, including the explanation that human reasoning is performed by domain-specific methods as opposed to general-purpose reasoning mechanisms. Arguably, a better explanation is that it can be hard to tell the difference between descriptive and prescriptive rules, and that apparent errors of reasoning can occur when the experimenter and the participant have different interpretations of the rules in mind.

I did not know about the Wason selection task when we started to work on integrity checking in deductive databases. Nor did I have the more general understanding, which I have now, that logic programming rules can be understood as representing an agent's beliefs, and integrity constraints can be understood as representing an agent's goals.

The integrity checking method we developed had the more modest objective of monitoring updates that add new facts to a deductive database, and of checking whether a database that satisfies integrity constraints before an update continues to satisfy the integrity constraints after the update.

The method repeatedly reasons forward (or bottom-up), starting from the candidate facts to be added to the database, unifying facts with the conditions of both logic programming rules and integrity constraint rules, and deriving instances of the conclusions of the rules. An update satisfies the integrity constraints if all the conclusions derived in this way are satisfied in the updated database.

Abductive Logic Programming (ALP) and Argumentation

During my six-month leave of absence, I had hoped to work on a second book, which I tentatively titled *Logic for Knowledge Representation*. Instead, I worked mainly with another PhD student, Kave Eshghi, on abductive logic programming (ALP).

Abduction is a form of reasoning that solves a goal, such as explaining an observation or creating a plan to achieve a desired state of affairs, by generating hypothetical "facts". ALP includes integrity checking, to ensure that the generated facts satisfy any integrity constraints. For example, the integrity constraint than *nothing can be in two places at the same time* can be used to exclude a hypothesis that a person committed a crime if it can be shown that the person was at another place at the time of the crime.

Kave and I also showed that negation as failure can be understood in abductive terms, as generating a negative condition *not p* as an abductive hypothesis, to solve a goal.

Informally speaking, the abductive interpretation of negation is more cautious than the closed world assumption. It does not conclude that *not p* is true if *p* cannot be proved. It concludes, more tentatively, that *not p* can be assumed to be true if *p* cannot be proved. This interpretation of negation can be formalised in ALP by imposing the integrity constraint: (1) *not p* and *p* do not both hold. We also imposed a second integrity constraint (2) either *p* or *not p* holds, to try to show that our abductive procedure for negation was equivalent to the stable model semantics of Gelfond and Lifschitz.

Our abductive procedure for negation used an adaptation of our integrity checking procedure for deductive databases. The abductive procedure simulates negation as failure, by using the integrity constraint (1) to show that *not p* can be assumed to hold when *p* fails to hold.

We also used the disjunctive integrity constraint (2), to ensure that *not p* holds *by default* if *p* fails to hold. However, Phan Minh Dung argued that the integrity constraint (2) is too strong, and he showed that a corrected variant of our abductive procedure implements a weaker and arguably more natural semantics of negation as failure, namely his "admissibility semantics".

Compulog

Just as my six-months leave of absence was ending, I received an invitation from Brussels to help organise a basic research project involving the main academic groups in the European Community working on logic programming. The resulting project, Compulog, aimed to extend logic programming, by developing a more powerful Computational Logic. The project employed Fariba as an academic replacement for College work, so that during the period 1989-91 I could work full time

as a researcher and as the project's coordinator. I continued the research that I started earlier, but with greater focus than before.

I mostly worked on ALP with Francesca Toni and Tony Kakas. We discovered that Dung's admissibility semantics for negation as failure can be understood in terms of arguments that defend themselves against attack from other arguments.

Francesca and I collaborated with Dung during his several visits to Imperial College. During one of these visits, Dung developed a more abstract argumentation interpretation of negation as failure, and he showed that it could be used to give semantics to other logics for default reasoning. This work was very well received; and, arguably, Dung's abstract argumentation theory is the dominant approach to argumentation theory today.

Fujitsu

Soon after the start of the Compulog project, Fujitsu Research Laboratories, which was one of the main partners in the Japanese Fifth Generation Project, supported a five-year project in our group, focused on ALP, during the period 1990-95.

During the Fulitsu project, I established good contacts with Ken Satoh, who had been a visitor in our group in the early 1980s. I continued my contacts with Ken, and I visited him later in Sapporo and Tokyo after he left Fujitsu and I retired from Imperial College.

Initially, the Fujitsu project supported Francesca Toni, as a PhD student. But, when the first three-year grant for the Compulog project ended, I transferred to the Fujitsu project and extended the leave of absence from my College work.

Towards the end of the project, Fujitsu encouraged me to investigate the application of logic programming to multi-agent systems. This made me look more closely at production systems, active databases and BDI (Belief, Desire, Intention) agent programming languages. These investigations convinced me that integrity constraints provide the functionality of condition-action rules in production systems, active rules in databases and commitment rules in BDI agent systems. This functionality is missing in basic logic programming. But ALP shows how it can be combined with logic programming in a natural way.

Back in the Department

When the Fujitsu project ended, I became slowly reintegrated into the life of the Department. Logic programming was beginning to go out of fashion, and the logic programming group was no longer seen as a threat. Indeed, my own rehabilitation was so complete that, during the period 1994-97, I became a member of a four-person Departmental Executive Committee, and I was even given the title of "Senior Deputy Head of Department".

I'm not sure what motivated me to get so involved in the running of the Department. Perhaps I wanted to show that I could rise above the parochial interests of the Logic Programming Group and could help to look after the interests of the Department as a whole.

The Department had both external and internal problems. Externally, we suffered the same fate as many other Computing Departments elsewhere. We were the poor relation of the more established departments, and we were inadequately resourced in comparison. When the College decided it should do more to promote Information Technology, it looked primarily to the Electronics and Electrical Engineering (EEE) Department for its lead.

To some extent, our low standing in the College was partially our own doing, the result of a long history of internal conflicts between competing groups. Perhaps it was because I had once been in conflict with the rest of the Department myself and because I had now made my peace that I was so welcome on the Department's Executive Committee.

I began to find my teaching increasingly tedious. The main problem was the inhibiting effect of the need to prepare the examination questions before presenting the course material. These preparations were needed to ensure that there was enough time to submit a draft of the questions to an independent external examiner and to make any changes required by the examiner. Although this requirement significantly contributed to the rigour of the examination process, I found that it increasingly inhibited the spontaneity and enthusiasm I could generate for my teaching.

Head of Department

In November 1996, the then Head of Department was so unhappy with the state of the Department and with our relations with the College that he resigned from his post. He agreed to stay on as Head until the Rector found a replacement. By the beginning of March 1997, there was still no news from the Rector, and the rumour went around that the Department would be broken up and distributed between the Mathematics and the EEE Departments. In desperation, as Senior Deputy Head of Department, I went to talk to the Rector myself.

My real goal was to return to full time research, to work on my book and to be my own boss. Instead, the Rector invited me to become Head of Department, and I accepted. One reason that I agreed to become Head was that I thought that it would give me the opportunity to apply Logic to the practical problems of the Department.

I planned to try to develop general rules to solve problems that would otherwise involve individual, *ad hoc* negotiations – such problems as deciding academic workloads, the overheads that should be charged on research grants, and the distribution of overheads between the Department and grant holders. I thought that establishing a clear set of rules that applied to everyone alike, without favour or malice, would take the politics out of decision making.

At first, I looked to the College for examples of best practice. I found a variety of methods used in other departments to calculate and regulate workloads, but I couldn't convince the academic staff in the Computing Department to try them out. Believing in Logic to the extent that I did, I wasn't inclined to impose by force what I couldn't achieve by logical argument.

I was even less successful in getting advice from the College about how to calculate the amount and distribution of research grant overheads; and this was one of the areas where some of the most

difficult problems arose in the Department. People couldn't agree whether research overheads should mainly support the groups doing the research or should support the Department as a whole. The College had no general policy about this, and different departments had widely different policies and practices. Discussions in our Department didn't produce any consensus either.

Although I tried hard to formulate general rules, I didn't succeed in convincing the Department. In addition, there were too many other problems that needed attention. These ranged from external problems of trying to get more resources from the College to internal problems of allocating scarce resources, such as office space, within the Department. I was surprised and disappointed to discover the extent to which people were unwilling to sacrifice their own personal interests for the greater good of the community as a whole.

I resigned as Head of Department, handing over to my successor in July 1999, and taking early retirement, at the age of 58, on 1st September 1999.

Professor Emeritus

Having left the Department, I planned to focus on writing my book about the application of Computational Logic to everyday life, aimed at a general, non-technical audience. But first there were other matters that needed to be cleared out of the way, some academic and others purely domestic.

On the domestic side, I moved with my wife from our home in Wimbledon to a small hamlet in the West Sussex countryside. We extended the original seventeenth century cottage, added an oak, timber-framed summerhouse, and created a parking area. I did most of the planning and project management myself, and some of the timber framing and masonry. I enjoyed the change from academic work.

I also enjoyed the opportunity to combine academic work with extended visits to Japan, Australia, Portugal, Switzerland and Venezuela. These helped me to return to research and to recover from my period as Head of Department.

Writers' Workshops

Before leaving Imperial College, I started a series of Writers' Workshops on Logic and English for PhD students in the Department. I continued the Workshops after leaving the College, during several visits to Japan, organised by Ken Satoh. In these Workshops, the students presented short, written abstracts of their work, and we discussed and debated how to improve their writing by using concepts of clarity, simplicity and coherence inspired by Computational Logic.

I enjoyed these workshops more than my other teaching. Compared with my normal lecture courses, which were often a stale recitation of predetermined conclusions, the workshops were generally an exciting, mutual learning experience. The students seemed to enjoy them as much as I did. I could test my theories about the logical nature of human thought, and the students could see how the theories might apply to their own practical problems of communicating their thoughts more effectively to other people.

WHO and UNICEF

I had another opportunity to apply Computational Logic to practical problems, when Tony Burton, working at WHO in Geneva, contacted me in 2009.

Tony belonged to a WHO/UNICEF working group tasked with producing annual estimates of global, country by country, annual infant immunisation coverage. Since 2000, the group had been collecting immunisation data from national authorities, together with data from international surveys. The different kinds of data are often inconsistent, both independently and in combination. The group needs to reconcile inconsistencies and publish an independent estimate of the actual immunization coverage. These estimates are often controversial and may be disputed by both national authorities and expert specialists.

Tony contacted me to see if I could help the group to formulate their informal rules and heuristics in more rigorous, logical terms, to make their decision making more transparent and more consistent. Computer implementation of the rules was not a major objective.

The group had been considering various possibilities for formalising the rules, including the use of both logic programs and production rules. We had many discussions about the differences and the relationships between the alternatives. Eventually, we agreed on a formulation of the rules in logic programming terms, which we then implemented using tabling in XSB Prolog.

In addition to helping to ensure consistency, the Prolog program documents the argument for every estimate. Because the rules are transparent, the estimates can be challenged; and if someone puts forward a convincing counterargument, the rules can be refined to produce better estimates both in the disputed case and more generally.

The WHO/UNICEF working group used the Prolog program from 2010 to 2024. In 2024, the program was reimplemented in R, which is now one of the standard programming languages used in statistical computing and data visualization. The logic programming rules are still being used, but they have been hand-compiled into R.

The Book: Computational Logic and Human Thinking – How to be Artificially Intelligent

Both the Writers' Workshops and the work with WHO/UNICEF confirmed my conviction that Computational Logic can really help people to think and behave more intelligently. This helped to encourage me in the work on the book.

When I first put this story on my webpage in 2002, I had made enough progress to acknowledge that I was actually writing the book. But it was proving more difficult than I had expected to make the book accessible to a non-technical audience.

I finally completed and published the book in 2011. Although it was not intended as a textbook, it has been used as a text in several universities, in both computing and philosophy departments. I

taught a course based on the book at Kyoto University in 2012. A Japanese translation of the book was published in 2025.

2011 was a good year for me. Not only did I complete book, but it was the year in which I received the IJCAI Award for Research Excellence. The citation says: "for his contributions to logic for knowledge representation and problem solving, including his pioneering work on automated theorem proving and logic programming".

ALP Agents

The book builds upon the use of ALP as a logical model of human thinking. It extends ALP by employing ALP as the thinking component of an intelligent agent that is embedded an ever-changing world. The agent's life is a continuous cycle, in which it observes the current state of the world and any events that happen, thinks, and acts to change the world in return, to satisfy its goals as well as it can.

An ALP agent's highest-level goals are *maintenance goals* of the form *if antecedent then consequent*, where the *antecedent* describes some features of the world until a certain time, and the *consequent* describes some features of the world after that time. For example, *if I am hungry at a time then I will eat some food at a future time*.

The agent monitors the world, to determine whether any antecedents of any of its maintenance goals are true; and, if an antecedent is true, the agent derives an *achievement goal* to make the consequent of the maintenance goal true in the future.

The agent's mission in life is to satisfy its goals as well as it can. For example, if you are hungry and you need to eat, then it is better to eat sooner rather than later; and it is better to eat something that you like than to eat something that tastes awful.

To help it with this mission, an intelligent agent maintains a database of beliefs, and it updates its beliefs with facts that describe its observations of the state of the world and of any events that happen. The agent's beliefs also include rules, which define abstract views of its concrete observations, define composite events as combinations of primitive events, and define plans of actions from combinations of primitive actions.

The rules that are included in an agent's beliefs can be used, among other things, to reason backwards to reduce achievement goals to achievement subgoals. Achievement subgoals include subgoals that are hypothetical, candidate actions. These hypothetical actions become facts if they are chosen for execution and they are executed successfully.

Beliefs that are rules can also be used to reason forwards, to derive logical conclusions from both existing facts and hypothetical facts. In particular, they can be used to determine logical consequences of hypothetical action facts, and to determine whether those actions might have desirable or undesirable side effects. This use of forward reasoning, together with estimates of the probability of circumstances that are outside the agent's control, can help the agent to make better decisions and obtain better solutions for its goals.

The Meaning of Life

The book includes a chapter on the Meaning of Life. Admittedly, the title of the chapter was designed to attract attention, but one reviewer seemed to dismiss the title altogether by pointing out that the Life in question is the life of a humble wood louse. I was disappointed by the review, because I intended the wood louse to be a metaphor for agents in general. I hoped that readers would notice that it is perfectly logical for an agent's life to be controlled by a production system of instinctive, condition-action rules, and for the agent not to be aware that its behaviour has been designed to satisfy the higher-level goals of an intelligent designer.

The Frame Problem Revisited

Having completed the book, and having argued the case for understanding the goals, beliefs and actions of an intelligent agent in ALP terms, I returned to more technical work with Fariba Sadri, developing a computer implementation of ALP for practical applications. We soon discovered that there was a huge obstacle to be overcome, namely the problem of dealing efficiently with change of state.

The event calculus and other solutions of the frame problem reason correctly about change of state, but they are not efficient enough for most practical applications. These solutions all make it necessary to reason, in one way or another, that, for every fact that holds before the occurrence of an event, the fact continues to hold after the event, unless it is terminated by the event. Given a history of events, these solutions, in effect, either compute or store the entire history of all states, from the beginning to the end of time. This certainly is not practical for even a moderately large amount of data.

All practical computer languages solve the frame problem by computing and storing only a single current state. Given one or more events that occur simultaneously, theses languages destructively update the current state, deleting any data that is terminated by the events, and adding any data that is initiated by the events. They leave any existing data that is not terminated by the events simply untouched, without reasoning that they are untouched. This solution enables efficient computation, but it creates the new problem of understanding its logical semantics. To solve this new problem, we had to reconsider the semantics of logic programming and ALP.

We solved the new problem by replacing the theoremhood view of the relationship between goals and beliefs by the model-generation view. In the theoremhood view, an agent's beliefs are a set of axioms, and a goal is solved by proving that the goal is a theorem that is a logical consequence of the axioms. In the model-generation view, a goal is solved by extending a model that makes the agent's beliefs true, so that the extended model also makes the agent's goals true. In formal logic, such worlds are called *interpretations* or *models*, and they determine whether sentences in the language are true or false.

The model-generation view justifies the use of destructive updates, because it simply constructs a model piecemeal. The model is constructed in the same way that the real world unfolds, existing at any given time only in its current state, and changing state by destroying its past. But in its totality,

the real world is the complete history of all its states and events, past, present and future. The frame axiom is true, not because it is used to reason about change of state, but because it is an emergent property that is true in this complete history of the world. In contrast, in the theoremhood view, destructive change of state is hard to justify, because it amounts to changing the axioms in the middle of a proof.

Logic Production Systems (LPS)

Fariba and I employed destructive change of state in a variant of ALP, for use as a practical computer language for programming, databases and artificial intelligence. We called the language LPS, because the language is a logical reconstruction of production systems.

We focussed on production systems because we wanted to show that, like production systems, LPS can be understood both as a scaled down model of human thinking and as a practical language for computer applications. For our logical reconstruction, in addition to justifying destructive change of state, we needed to show that condition-action rules in production systems can be reformulated as goals in logical form.

Condition-action rules in ordinary production systems do not have a logical interpretation. They have a seemingly logical syntax as rules of the form *if conditions then actions*. But the "inference engine" that executes the rules employs a procedure called "conflict resolution", which does not have a logical semantics.

For example, given the rules

> *if I am hungry then I eat some food*
> *if I am sleepy then I go to sleep*

and given the facts that *I am hungry* and *I am sleepy* at the same time, production systems employ conflict resolution to select only one of the two actions *I eat some food* or *I go to sleep*, and it performs the selected action immediately. If the rules were sentences with a logical semantics, the inference engine should derive both actions, instead of only one, as logical consequences of the facts and rules.

In the logical reconstruction of condition-action rules in LPS, the rules are rewritten as maintenance goals with explicit times, and the constraint that *a person cannot eat and sleep at the same time* is expressed explicitly as an additional goal. The LPS inference engine correctly derives both the conclusion that *I eat some food at a future time* and the conclusion that *I go to sleep at a future time*, but where the two future times need to be different.

There can be many worlds that satisfy the goals, differing by the times at which they make the two actions true. Some of these worlds may be better than others, and the implementation of LPS needs to choose between them, either by making sensible decisions itself, or by enacting the preferences of the goals' designer. Deciding how to deal with preferences has been the biggest challenge in the design of LPS.

I investigated the related problem of how to deal with conflicting obligations and contrary-to-duty obligations with Ken Satoh. We argued that obligations can be understood as goals in ALP. To say that *p* is obligatory, is to say that *p* is a goal and it must be true in all best possible worlds. To say that *p* is obligatory, but that if *p* is violated then *q* is obligatory, means that the real goal is *p or q*, and that models in which *p* is true are better than models in which *q* is true. We argued that this approach solves several "paradoxes" in the logic of obligations, and we published our work in the Journal of Philosophical Logic.

Our work on LPS was concerned with more practical issues of developing a usable implementation of LPS. The first implementations were done in Prolog as MSc student projects. Occasionally, I would test an implementation using small examples written in LPS. When the examples did not work as I expected, I would study the Prolog implementation to see whether the implementation or the example was the source of the problem. Eventually, I decided to reprogram parts of the implementation myself. To my surprise, I discovered that I enjoyed programming in Prolog.

Computational Logic for Use in Teaching (CLOUT)

In 2016, we received a research grant from Imperial College to create a new implementation of LPS, together with example LPS programs, for teaching logic and computing to students in secondary school. We recruited Miguel Calejo in Lisbon, to produce a professional implementation of LPS using SWISH, an online interface for SWI Prolog.

I enjoyed writing programs in LPS, illustrating the wide range of pedagogical examples that can be implemented naturally in LPS. The programs included such examples as the prisoner's dilemma, the dining philosophers, rock-paper-scissors, map colouring, toy blocks worlds, Conway's game of life, self-driving cars and bank account transactions. To make the examples more appealing, Miguel developed an elegant declarative language for associating images with facts. The images change as the facts change over time, animating the history of the world generated by the program's actions and other events. Many of the examples and animations can be found at LPS on SWISH and in the paper Combining Logic Programming and Imperative Programming in LPS.

We organised several workshops at Imperial College for high school teachers to introduce them to our new, user-friendly implementation of LPS, and we advertised the workshops in the Computing at Schools (CAS) Forum of the British Computer Society. The workshops were well received, and the teachers who participated were very enthusiastic. But we didn't have any connections inside the educational establishment, and the new language and its applications that we were promoting were in direct competition with the computer science curriculum supported by the establishment.

Because we did not have sufficient resources to campaign successfully for our radically different approach to educational computing, we shifted our attention to other applications of LPS.

Logical Contracts

The competition for CLOUT came not only from more established approaches to teaching computing, but also from other applications of LPS competing for our attention. The most pressing of these were applications of LPS to blockchain systems, smart contracts and law.

We started to investigate blockchain applications around 2017, when the excitement surrounding blockchains and cryptocurrencies, such as Bitcoin, was near its peak. Miguel was the driving force behind this work, and we recruited my former PhD student, Jacinto Dávila, to join us. We tentatively explored the creation of a company, Logical Contracts, in association with Imperial College, to logically represent legal contracts in a language that is close to natural, human language, but executable by computer. The plan was to use logical contracts implemented in LPS to:

- monitor compliance of the parties to a contract,
- enforce compliance, by automatically performing actions to fulfil obligations, and/or by issuing warnings and remedial actions to respond to violations of obligations,
- explore logical consequences of hypothetical scenarios, and
- query and update the Ethereum blockchain.

We developed several proof-of-concept applications, and we showed how other smart contract applications could be reimplemented naturally in LPS.

We obtained support from several small companies, implementing applications in such areas as international swaps and derivatives contracts, and accountancy tax law. However, the greater the number of applications we developed, the more we could see that the blockchain technology was incidental to the main need, which is for a technology-agnostic computational representation that is close to natural language. This need led, in turn, to a new focus for our work, namely to the development of a controlled natural language, Logical English, which is syntactic sugar for Prolog or LPS, and which can be read and understood without technical training, but with only a reading knowledge of English.

Compared with ordinary English, not only is Logical English computable, but in many cases, it can be easier to understand. Just as importantly, because Logical English is unambiguous, it is harder to misunderstand.

The 50-Year Anniversary of Prolog and the Prolog Education Group (PEG)

In 2022, the Prolog Heritage Association and the Association for Logic Programming celebrated the birth of Prolog in 1972. In addition to a special Prolog Symposium held in Paris and to other celebrations that year, we established the Prolog Education Group (PEG), to promote logical and computation thinking through Prolog. The Group has been meeting online for this purpose biweekly since its inception.

Because the logical core of Prolog is much simpler than ALP and LPS, it is also much easier to learn. Students can learn logic and computing implicitly by using Prolog to explore the logical consequences of facts and rules, starting with given examples, and updating and modifying the examples with new facts, rules and assumptions.

Although the logic underpinning Prolog lacks some of the features of classical logic, it includes such powerful features as negation as failure and the amalgamation of object language and metalanguage, which are lacking in classical logic. Negation as failure makes it possible to represent

rules and exceptions; and the amalgamation of object language and metalanguage makes it possible to express propositional attitudes and speech acts. These features enable many applications, such as the representation and execution of legal texts, for which classical logic is inadequate. They also make it possible to use Prolog to support logical and computational thinking at all levels and in all areas of the educational curriculum.

These legal and educational applications of Prolog can be facilitated by employing Logical English syntax, using such simple sentences as *Alice likes a person if the person likes logic*, which can be understood even by young children without any formal training in logic, mathematics or computing.

Runnable and modifiable examples of Logical English can be found in Logical English on SWISH and in the papers Logical English for Law and Education and Logical English Demonstration.

Logic in the Age of AI

During most of the 20th century, symbolic approaches, many of them based on the use of logic, dominated AI. This changed, around 10-15 years ago, when so-called sub-symbolic approaches, using artificial neural networks trained on vast amounts of data and using powerful graphic processing units (GPUs), began to make huge advances. These advances include large language models (LLMs), which generate natural language text in response to human prompting. Because they are trained on virtually the whole of human knowledge on the internet, they are beginning to achieve human levels of general intelligence, and they are on the verge of achieving a form of superintelligence. These developments are creating a huge challenge for what it means to be human in the age of AI.

To address this challenge, we need to improve our understanding of what it means for a human or artificial agent to be intelligent. With such an understanding, we can hope to improve our own human intelligence and to exploit artificial intelligence for our own human goals.

Arguably, human intelligence can collaborate with artificial intelligence in much the same way that conscious logical thinking collaborates with subconscious intuitive thinking in the dual process cognitive model of human thinking. In humans, logical thinking monitors intuitive thinking, endorsing it in some cases and overriding it in other cases. Similarly, when humans and AI collaborate, human logical thinking can monitor AI thinking and can control the use of AI for human purposes. The Prolog Education Group is reorienting its mission for this task, and I am contributing to this effort.

Wikipedia

Although it is not directly related to PEG or to developments in AI, I have also participated in educational activities as an editor of Wikipedia, off and on since around 2006.

I am fascinated by Wikipedia as an experiment in democratic decision-making, where anyone can edit an article and present their point of view, and decisions are made by trying to reach consensus through argument and discussion. It intrigues me to think that such a consensus-building approach might work for solving other problems, such as deciding how to run a country or how to settle a dispute between different countries.

While my own experience has been generally positive, it hasn't been without its problems, including editing wars in the early days. More recently, several of my edits have been reverted, because it was argued that they did not represent a neutral point of view or that they exaggerated the importance of my own point of view. These reversions discouraged me for a while, but I am slowly recovering my confidence to begin anew.

But what amazes me, more than anything else, is the reluctance of other experts to contribute to Wikipedia even when there are glaring mistakes or imbalances that anyone with even a modest knowledge of the subject can recognise. It reminds me of the way that many people do not take part in political elections, because they think that their vote won't make any difference.

Life in the Stone Age

I don't work all the time. The best time for me to work is in the morning, and then intermittently throughout the day. Some days I don't work consciously at all.

Living as I do in West Sussex, I don't have to go far to immerse myself in the English countryside. The South Downs are not far away, and I can also walk straight out of my garden or across the road into the adjacent fields. My neighbour, who farms the fields, lets me wander over them with few constraints.

One day about eighteen years ago, I was walking in the field across the road when I noticed some worked flint lying on the ground. For several years, I had been looking for prehistoric flint artefacts off and on, mostly in the South Downs, where there are Neolithic flint mines. I soon discovered that all around me, within a mile of my home, there were the remains of prehistoric activity, mostly dating to the Mesolithic period about 8,000 years ago. Since my first discovery, I identified three separate Mesolithic sites and collected a large number of flint artefacts, including microliths, arrowheads, scrappers and knives.

About twelve years ago, I teamed up with the distinguished archaeologist and lithics expert, Andrew David, shortly after his retirement. Together, we explored my Mesolithic sites in greater detail, and we published an article in the Sussex Archaeological Collections, documenting our discoveries.

In the last few years, however, coinciding to some extent with disruptions caused by COVID, I have scaled back my archaeological activities, and I have been spending more of my time in the garden, planting and shaping trees in the Japanese, niwaki style.

Search for Truth

Looking back at my academic work, I like to see it as a search for truth, with Logic leading the way.

The search began in secondary school, triggered by my extracurricular reading of such books as Joad's "Guide to Philosophy". When I read about Plato's philosophy of ideas, I was convinced that it was true. And when I read about Aristotle's empiricism, I was convinced again, but this time that a

contrary philosophy was true. It couldn't be that both philosophies were true. But Joad offered no guidance to distinguish between the two apparent truths.

The first-year mathematics course at the University of Chicago introduced me to mathematical logic, which seemed almost magical in its use of symbolism. Mathematical logic seemed to be able to conjure truth out of nothing. I decided to major in mathematics at the University of Bridgeport, partly because mathematics is the language of mathematical logic, and partly because it seemed to show that indisputable truth is possible. I hoped it might help me to find other truths elsewhere.

My search continued at Stanford and the University of Warsaw. But I began to doubt that mathematics would help to solve such life and death problems as the war in Vietnam. I never questioned the relevance of Logic, because to my mind the logic of common-sense left no doubt that the war was wrong. But I questioned the purpose of mathematical logic, because it seemed to me that it had become a branch of pure mathematics, and that it had lost touch with the original purpose of Logic, to help people think more clearly and more effectively.

Ideally, I would have continued my studies of Logic in a philosophy department. But I didn't have the necessary academic background. I found myself doing a PhD in computer science at the University of Edinburgh instead. Fortunately, the PhD, which was about using symbolic logic to mechanically prove mathematical theorems, didn't require any knowledge of conventional computing.

The topic of my PhD was not one that I chose for myself. Nor was it on the shortest path to my ultimate goal. But it gave me an entry into the field of artificial intelligence, where I worked on the development of logical methods that could be implemented by means of computers. Although I had little enthusiasm for the goals of artificial intelligence, I learned that the same logical methods I was developing to prove mathematical theorems, could also be used for other, less mathematical kinds of problem solving. I was encouraged by the thought that the same logical methods, used to make computers more intelligent, could also be used by people to improve their own human intelligence.

My work has also benefited from attacks against logic by other researchers working in artificial intelligence. These attacks drew attention to weaknesses in my theories and helped me to identify areas where the theories needed to be improved.

Perhaps the biggest weakness of traditional mathematical and philosophical logics is that they focus on pure, disembodied thought. Even logics like the event calculus, which are concerned with actions, events and changing states of affairs, just deal with thinking about change, without actually performing it. I believe that the model-generation semantics solves this problem.

I am still searching for the truth. I started by believing that the truth comes from proving theorems as logical consequences of axioms. But now I believe that the truth comes from performing actions to satisfy our goals. But, because other agents have other goals and perform other actions, our combined actions can conflict with one another, and our actions can be self-defeating.

The actions we perform come from two sources: from subconscious, intuitive associations of conditions and actions, and from conscious reasoning to derive actions to satisfy goals (by abduction). Both sources of candidate actions are valuable, and both can be improved. Intuitive,

condition-action associations can be improved by gaining more experience and by reinforcement learning. Conscious reasoning can be improved by better logical reasoning and by gaining more knowledge, consisting of true beliefs.

But knowledge alone is useless. It becomes useful when Logic uses knowledge to derive candidate actions from goals. The more knowledge we have, the more options we have for satisfying our goals, and therefore the more options we have for avoiding conflicts with other agents. Logic can also use knowledge to derive possible consequences of candidate actions. This can help us to identify both positive and negative consequences of those actions, and it can help us to decide which actions to perform.

What are the implications for Education? Yes, we need to teach AI, because in doing so, we need to teach what it means for any agent, human or artificial, to be intelligent. But we shouldn't stop at honouring the intelligence of AI, no matter how powerful it may become. Moreover, there are not many lessons to be learned from subsymbolic AI for improving human condition-action associations. This needs to be learned from human experience.

But we can and should teach the lessons we have learned from developing symbolic, logic-based AI, because they can also be used by humans, to improve our own human, logical thinking skills. Moreover, we can use such logical systems as Prolog and Logical English, to help to support those lessons with entertaining and educationally relevant examples.