Invited Lecture

Logic, Computer Science and Artificial Intelligence: A Historical Perspective *

J.A.Robinson
Professor
Syracuse University

Good Morning. It is an honor and a pleasure to be invited to give a lecture at this conference. The topic was proposed to me by Dr. Koichi Furukawa and Professor Hidehiko Tanaka, and I welcome the opportunity they have offered me to discuss the part played by logic and logicians, both now and in the past, in the development of computer science and artificial intelligence.

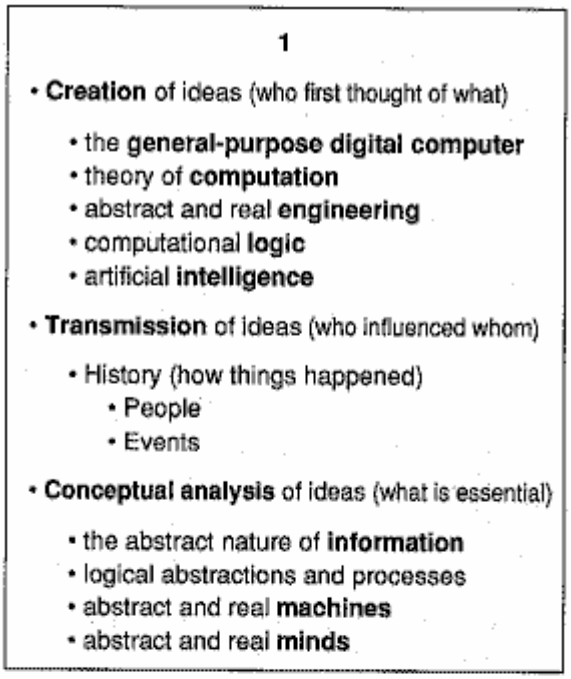
In my view, the conceptual origins of the computer are to be found in the work of certain pioneering logicians. Logic has, from the start, been the main component of computer science, and will continue to be that. Although logic has in the past also had an important apart to play in artificial intelligence, its contribution to the future of that particular sub-field of computer science is likely to be a relatively minor one.

When the Fifth Generation project was announced at the beginning of the 1980's, the organizers, as you all know, chose logic, and in particular logic programming, as the principle theme of the project. They emphasized the role of logic as a unifying and guiding principle in the design of computers and of programming languages. This choice, as you all remember, caused some surprise. Many people viewed it as a startling and risky change of direction in dealing with computers and computing. Some even saw it as the beginning of a kind of revolution. This reaction was quite understandable, but in my opinion it was a mistaken one. The emphasis on logic was indeed radical, but only in the literal sense. It took us back to the roots of the computer and of computer science. I believe that logic was a most natural choice.

The topic is structured, it seems to me, as shown in slide 1. We ask: who did what, and when? Who was it who, for the first time, thought of the idea of the general purpose digital computer, the so-called universal machine? Where did what we now call the theory of computation get its start? Where did what we now think of as computational logic begin? What about artificial intelligence? We should consider not only how these ideas got started, but also how they then flowed down to our own times; and who transmitted them, and how.

We must not only talk about the people and the events but also about the ideas themselves. The distinction between *abstract* and *real* will recur throughout the discussion. We will constantly abstract from processes and structures what seem to be their essential features. This means discarding the details that don't matter for the analysis and for understanding. So for example we will be talking about abstract and real machines. Indeed, we shall talk about *abstract and real people*, since

* The original title in FGCS'92 Proceedings is "The Role of Logic in Computer Science and Artificial Intelligence".



that, it seems to me, is a good way to look at what we do in artificial intelligence: we try to discard all irrelevant details in intelligent creatures so that we can deal only with what is *essential* to intelligence.

It is most important to keep the distinction between abstract and real clear when we seek to understand computers. I hasten to say that in concentrating on the abstract side of the distinction I am not at all somehow relegating the engineering details of computers to a lower level of importance. Quite the contrary. In fact in order to give a proper view of the marvels that the engineers have performed over the past several decades of actually building computers one really is forced to make this same distinction. It then becomes all the clearer how extraordinary it is that some of our most refined abstractions can in fact be realized in practical machines.

If you do look more closely (slide 2) at what we will now begin to call abstract computers, you realize that they are really formal logical systems. They are sets of definitions, rules of inference, and axioms, such as logicians and abstract mathematicians have been studying and developing for a very long time. As for real computers, well: they are physical machines, made out of electronic circuits and electromechanical subcomponents. Their behavior is causally determined, according to physical laws, by their design and by the inputs which are submitted to them from the outside world. Every real machine, and in particular every real computer, is in a straightforward sense the physical embodiment of an abstract machine. But to *understand* a real computer is to simply grasp the abstraction that it embodies: it is not necessary to be able to follow the 'causal logic' of its physical organization in order to program it intelligently.

Many people, myself included, learned to program on abstract Turing machines. There *were* of course no real ones, only abstract ones. We found their definitions and rules of operation in logic text books and in Turing's 1936 paper, and we programmed them on the basis of this purely logical description.

To 'run' the programs we had to 'simulate' the machines' behavior by the pencil-and-paper application of the transition rules. From the logical point of view, from the point of view of understanding the computational process, it was just the same as if we had 'real' Turing machines made out of electronic components to deal with. It was just a matter of irrelevant detail: the essence of the situation was no different.

We usually encounter Turing's name just as the label in the phrase 'Turing machine', but in this lecture I shall claim (and not everybody may agree) that Alan Turing was the major of two outstanding figures (John von Neumann was the other) in the history of the modern computer. Both of these men were extraordinary people. They were both intellectual giants, but they could hardly have been more different from each other in every other respect. Although Turing was younger than von Neumann by almost ten years he probably was, in some sense, the senior person intellectually. He was in fact the originator of ideas that we have come to associate with von Neumann.

Turing's life was not a very long one (slide 3). He was born in 1912, and died in 1954 just before his 42nd birthday. But it was an enormously busy and fruitful life. In 1936, he published what soon became a classic paper. It was only partly a solution to a long-standing problem in the logical

2 ABSTRACT AND REAL COMPUTERS

- This lecture looks at computers from the **abstract** point of view, but it is dedicated, with admiration and awe, to all those engineers whose utterly marvellous **technology** has turned abstractions into practical realities.
- Abstract computers are **formal logical systems**—sets of definitions, axioms and rules.
- Real computers are machines made out of electronic circuits and electromechanical components. They are totally governed by physical laws and the inputs they receive from the external world.
- Every real computer embodies an abstract one.
- When I started programming forty years ago, my first computers were abstract Tuning machines. Simulating them on paper was slow and tedious. *From the logical point of view* it was the same as dealing with 'real' machines.

foundations of mathematics, the so-called decision problem of Hilbert. *In solving this problem, Turing invented the computer.* The so-called universal Turing machine which was defined and fully developed in his paper is, in fact, the prototype, both conceptually and historically, of all subsequent general-purpose digital computers.

Right after the war, in 1945, Turing designed the ACE computer in England. In the course of his very accessible and readable account of that disign, he proposed the idea of a stack, the idea of stored-program device, and the idea of subroutines. These now very familiar notions appear for the first time in this discussion of Turing's. A little later, in 1949, in his lecture *Checking a Large Routine*, he introduced the methodology of formal reasoning about programs, in particular of proving that they meet their specifications. Turing is of course very widely known today for his initiation of artificial intelligence (AI). He is often accorded the role of founding father of that subject, I think quite properly. What is not so widely known is that two years before his famous philosophical 'Turing Test' paper in 1950, he had given a very thorough discussion of the more technical content of the subject, going over its ideas and its difficulties as he then saw them. There is no doubt that Turing really does tower over those early years of AI.

Let us turn to von Neumann (slide 4). Whereas Turing was an immensely gifted genius, he was also a really quite eccentric person, highly unorthodox in most ways. You might say he was an 'outsider'. But von Neumann was just the opposite, an 'insider' in the fullest sense of the word. He had prodigious intellectual gifts. He outshone everybody, including even Turing, in that way. There may never have been anyone else quite like von Neumann.

He was uniquely brilliant. He was also a politically shrewd and a conservative, immensely influential practical person. He participated in scientific leadership at the highest levels of industry and government research, and he helped to make national policy right to the end of his life. He too had only a short life. Both he and Turing came to tragic ends while still relatively young. But even though von Neumann was extraordinarily influential in the history of computers, I believe that his role was secondary to, and derivative from, that of Turing.

The historians have been busy and have shed much light on the interactions between the two men. We can now see clearly that von Neumann was inspired by Turing and specifically by his 1936 paper, and that he then went on with his incomparable force of personality and mind to promulgate Turing's ideas and make them come alive. The so-called von Neumann computer really should be labeled the Turing computer.

3 ALAN M. TURING 1912-1954

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE *ENTSCHEIDUNGSPROBLEM*, 1936

- The **universal Turing machine** in this paper is the **prototype** of all general-purpose digital computers

SYSTEMS OF LOGIC BASED ON ORDINALS, 1938

- Turing's Princeton Ph. D. thesis

ACE REPORT, 1945

- stack, stored-program, subroutine concepts

CHECKING A LARGE ROUTINE, 1949

- first formal correctness proof of a program

INTELLIGENT MACHINERY, 1948

- detailed discussion of AI technology

COMPUTING MACHINERY AND INTELLIGENCE, 1950

- Introduces famous **Turing test** for artificial intelligence

Von Neumann was perhaps the leading, young logician of the 1920s. He was already famous when he and Turing first met, when Turing was a graduate student, 23 years old, at Cambridge University. Subsequently they had quite regular contacts in Princeton, where Turing spent the years 1936 and 1937 as a graduate student working with Alonzo Church. Both Turing and von Neumann had offices in the same small building, Fine Hall, in which were located the Mathematics Library and the offices of the Princeton University Mathematics Department. Fine Hall was designed to encourage mathematicians to get together, and it provided an ideal research environment. There was at that time a newly founded research organization in Princeton called the Institute for Advanced Study, of which von Neumann was a member, and which had nothing to do with the University except that at that time it was housed in Fine Hall. So von Neumann and Turing were thrown together. We now have a lot of evidence that they interacted with each other. It seems very probable that one of their main topics of discussion would have been the concepts and results from Turing's 1936 paper.

Once the war began, in 1939, both men were off to very important war work, Turing in cryptology, von Neumann in almost everything else. We know that from November 1942 to March 1943 Turing was in the United States, interacting with many people. Two of these months were spent visiting Claude Shannon at Bell Telephone Laboratories. It has not been shown that he and von Neumann got together during that time, but many people have conjectured that surely they must have. I believe that the two of them actually did meet, and that they did exchange ideas about automatic computation which represented the 1943 state-of-the-art.

In 1943 vacuum tube technology for digital computation was just beginning to be exploited in both countries. In the United States, this was primarily at the University of Pennsylvania's Moore School, in the famous ENIAC and EDVAC computer projects. In England, it was primarily the Bletchley Park code-breaking machines. It is quite clear that the British and the Americans were sharing all of the relevant technology with each

other at that time in the scientific war effort. So I conclude that the Turing-on-von Neumann influence continued throughout the war. We have the testimony of several people that von Neumann urged his colleagues to study Turing's 1936 paper, saying essentially that this paper was absolutely fundamental to what they were trying to do (in designing a general purpose digital computer). Von Neumann went on to play a major part, as you all know, in the design and construction of one the first general-purpose machines, the EDVAC. There is some question about which real computer was the first really universal machine, with stored program and so on, to work. Perhaps the EDSAC (at Cambridge University, based on the Moore School's EDVAC) was, or the Manchester University's prototype machine. But just which machines were the first to fly is not particularly interesting here. We are talking, after all, about the *abstraction* which was there for ten years before the first physical machine embodying it actually ran.

Von Neumann did have a great deal to do with physical machines—in particular, with the Institute for the Advanced Study machines, the IAS series, which were *his* machines in the sense that he oversaw their basic design. He soon went on beyond this to other matters which impinged more directly on AI. He essentially founded what we now think of as the theory of automata, especially that of cellular, quasi-biological automata. In particular he solved the impossible-sounding problem of specifying how his automata could automatically reproduce themselves. Amazingly, his construction is abstractly the same one nature seems to have hit upon, but von Neumann discovered and described his scheme before Crick and Watson's 1953 discovery of the structure of the DNA molecule. A remarkable feat!

Let us next look at Turing's non-abstract activities in the history of the computer (slide 5). He was responsible for the design of a real machine, called the ACE, in England in the immediate post-war period. It was only because he was a rather naive, non-administratively-subtle, nonpolitical person, that the ACE wasn't the first one to run. He was simply unable to push the actual construction along at the speed that would have made him first in the field. Others got there first, but that is not a part of the thesis here. What must be said is

4 JOHN von NEUMANN 1903-1957

- top young **mathematician** and **logician** in 1920s
- dominated **axiomatic set theory** and Hilbert's **formalist program** at Göttingen, 1921-1930
- after Gödel's 1930 bombshell, concentrated on **physics** and **applied mathematics**,1930-1944
- first met Turing in 1935 in Cambridge University
- regular contacts with Turing at Princeton, 1936-1937
- they may have met again in 1943
- urged his associates to study Turing's 1936 paper, as being **fundamental** to the design of computers
- major role in Eckert-Mauchly projects (ENIAC, EDVAC), 1944-1946
- designed and built **IAS** computer, 1945-1952
- self-reproducing automata; theoretical neuroscience, biological information-processing, 1943-1957

5 TURING'S **ACE** DESIGN, 1945

PROPOSAL FOR DEVELOPMENT IN THE MATHEMATICS DIVISION OF AN **A**UTOMATIC **C**OMPUTING **E**NGINE, 1945

- Real **stored-program** digital computers, the Pilot ACE and the DEUCE, based on the design in this proposal, were later constructed and operated successfully at the National Physical Laboratory in England

• stack **• subroutine** **• stored program**

- ACE was a conceptual descendant of the universal Turing machine
- In 1947 Turing wrote:

Some years ago I [made] an investigation of the theoretical possibilities and limitations of ***digital computing machines****. I considered a type of machine which had a central mechanism, and an infinite memory. Machines such as the ACE may be regarded as* ***practical versions of this same type of machine***

that regardless of which machines were actually first to operate, the ideas certainly were Turing's. In the 1947 survey of the situation for the London Mathematical Society he directly linked the real machines of that postwar period with his 1936 abstract one.

Slide 6 displays a few quotations from Turing's writings illustrating how he connected digital computing with logic. The last quotation, for example, shows that he clearly foresaw the idea of formal symbol manipulation. The general overall impression these passages give is that Turing was always aware that the importance of the basic ideas underlying computing machines lay in their essentially abstract and logical character. The fact that they could be realized or embodied in electronic devices or indeed in any other kind of physical device, although interesting and practically important, is relatively uninteresting from the purely scientific point of view. What gives these ideas their great power is that they are not dependent on the volatile nature of rapidly shifting technology.

Professor Michie and Professor Good have testified that towards the end of the war Turing was already making plans to build his universal machine. His motivation for doing so was to do AI. This is why he designed the ACE machine. He wanted, as he put it, to build a brain. Everything else was unimportant. He simply wanted to start AI and to have a useful tool. He believed that because of the universality concept there was no need to build more than one machine, the universal machine. Just get it built, and thereafter everything else was a matter of merely of programming it, of making software for the universal machine.

Slide 7 has some of Turing's thoughts about AI. You can see that he took seriously the distinction between real and abstract in the case of human beings. His approach to AI was to think away all of the irrelevant details of a physical real human being and just consider reproducing, in the form of computer software, computer programs for his machine, only what was *essential* about a human. It is what the brain is doing that is essential—this was what Turing thought. He thought that what the brain does could also be made to happen in his universal machine by suitable programming. This, then was his key to AI. So he said that to build a thinking machine, all you need to do is to take a man as a whole and replace him part by part with artificial subsystems. This, you might say, is *prosthetic extrapolation*. We can already, in today's technology, give a man artificial teeth, limbs, bones, joints and heart, and so on. Where will this stop? It is an entrancing thought, that you can just keep on replacing parts (as technology improves) until you are replacing the very neurons

6 TURING ON LOGIC AND COMPUTING

- *the property of being digital should be of greater interest than that of being electronic*
- *I expect that digital computing machines will eventually stimulate a considerable interest in* ***symbolic logic***
- *One could communicate with these machines in any language provided it was an exact language*
- *In principle* ***one should be able to communicate in any symbolic logic***
- *there will be much* ***more practical scope for logical systems*** *than there has been in the past*
- *Some attempts will probably be made to get the machine to do actual* ***manipulations of mathematical formulae***

7 TURING ON MACHINE INTELLIGENCE

- *A great positive reason for believing in the possibility of making thinking machinery is that* ***it is possible to make machinery to imitate any small part of a man.***
- *One way of setting about building a 'thinking machine' would be to take a man as a whole and to try* ***to replace all the parts of him by machinery***
- *What we want is a machine that can learn from experience. The possibility of letting the machine alter its own instructions provides the mechanism for this*
- *If a machine is expected to be infallible, it cannot also be intelligent*
- *No man adds very much to the body of knowledge, why should we expect more of a machine?*

in the brain.

In slide 8 we give some items of evidence supporting what I claimed earlier, that Turing's ideas were transmitted, amplified and promulgated through von Neumann. There are excellent biographies of each man now available: Andrew Hodges' *Alan Turing: the Enigma*, and William Aspray's *John von Neumann and the Origins of Modern Computing*.

In Hodges' book we read that von Neumann wrote a letter of support for Turing during Turing's graduate student years in Princeton. He wrote that he knew Turing very well, admired him and his work, and urged that he be given a fellowship. At the end of Turing's two years in Princeton, von Neumann offered him a research assistantship at the Institute of Advanced Study, quite an accolade, but Turing turned down the offer in order to return to Cambridge and, as it soon turned out, to the war work at Bletchley Park. I already mentioned how von Neumann praised Turing's work to other people who have put this fact in the record. For example, Julian Bigelow, who was von Neumann's chief engineer and chief designer on the Institute for Advanced Study project, was told that his first assignment on joining the project was to study Turing's paper. Stanislaus Ulam, the famous mathematician who was perhaps von Neumann's best friend, writes in his *Adventures of a Mathematician* how von Neumann often expressed to him his admiration for Turing and for Turing's brilliant ideas.

8 TRANSMISSION OF TURING'S IDEAS VIA VON NEUMANN

- *The **universal Turing machine** is a **model of the stored-program computer** (developed eight years later)*

  William Aspray, 1990

- von Neumann's **letter of support** for Turing, 1937
- von Neumann offered Turing a **research job**, 1938
- **von Neumann praised Turing's work to others**. He made Julian Bigelow, his chief IAS designer and engineer, study Turing's 1936 paper
- Stanislaus Ulam reported that in 1939 von Neumann expressed great admiration for Turing and his "brilliant ideas"
- *Many people have acclaimed von Neumann as the father of the computer but he would never have made that mistake himself. **He firmly emphasized to me that the fundamental conception is due to Turing**. Von Neumann was well aware of **the fundamental importance of Turing's paper of 1936**. [He] introduced me to that paper and at his urging I studied it with care*

  Stanley Frankel, 1972

Stanley Frankel and Nicholas Metropolis wrote the first major program to run on the ENIAC when that machine started. It was some kind of shock wave computation, very secret. Later, Frankel wrote a letter to Professor Brian Randell, the computer historian, from which we take the last quotation in slide 8. Frankel was neither a computer engineer nor a programmer; he was a user, a physicist who needed to solve a practical computation problem.

Slides 9 and 10 are glimpses of Turing's contacts with another person who, of course, is very well known, Claude Shannon.

I don't think many people in computer science today are intimate with Shannon's work, but I think his ideas were probably also in the back of the von Neumann's mind when he moved strongly into the computer field. Shannon's 1938 paper, in which he linked Boolean logic (that is, the abstract notation of essentially the propositional calculus) with relay and switching circuits, had a major impact on many people at that time. It introduced a theoretical way of getting a grip on logic design and invited the development of algorithms for the minimization of the number of switching components, and so on, topics which suddenly became very much everybody's concern who was involved in the problems of computer design. I was struck by the fact that Turing and Shannon had a good relationship and spent a lot of time talking to each other in late 1942 and early 1943. Ostensibly and officially, they met only to discuss speech scrambling devices. However, it defies the imagination to suppose that they did not also talk about the somewhat different topic of digital computing. I think it was, during this visit that Turing and von Neumann probably had some quite secret, and indeed still secret, meetings. These meetings, I am sure, will eventually emerge into light of day when the relevant archives are at last opened to public view.

As slide 10 reminds us, Shannon was the great pioneer of the mathematical theory of communication, or information theory. Since Turing was interested in everything, certainly this theory of Shannon's would have fascinated him. It is another, excellent, example of an exercise in abstracting the essential features of a situation from the irrelevant physical details that embody them. It is very interesting that in the 1956 volume *Automata Studies*, which Shannon edited together with John McCarthy, Shannon contributed a curi-

9 CLAUDE SHANNON

- A SYMBOLIC ANALYSIS OF RELAY AND SWITCHING CIRCUITS, 1938
- The first explicit study of the realization of abstract logical expressions in electrical circuits This was a major theoretical advance. It exploited the formal analogy between syntactic and semantic properties of the propositional (Boolean) calculus and operational behavior of relay and switching circuits.
- **Turing visited Shannon** for two months at Bell Laboratories in early 1943, to discuss speech scrambling. Surely they would also have discussed digital computing? During this same visit to the USA Turing may also have met with von Neumann to discuss wartime computing problems.

ous, ingenious paper on the 'plasticity' of the structure of the universal Turing machine.

In slide 11, we find two important names which, although they may be half-forgotten now, will perhaps soon become well known again because of the current renaissance of artificial nerve-networks ("neural networks").

McCullogh and Pitts invented the artificial nerve-network idea in 1943, the same year in which Turing visited Shannon (and possibly also von Neumann).

McCullogh and Pitts introduced a certain two-dimensional dataflow-like notation in which they could represent discrete-state automata as networks of basic units. The basic units were thought of as artificial neurons, which had input and output lines connecting them with other similar units, and which behaved as finite-state switching elements. Their work was an attempt to do *abstract neuroscience*. It had an enormous impact, but of a different kind than perhaps they had anticipated. In particular, their notation with its accompanying abstract model was immediately seized upon by von Neumann as the right way to think about computers. He saw that computers can be represented abstractly as networks of connected finite-state devices, which would have physical realizations in electromechanical or electronic components. But with the McCullogh and Pitts notation you design computers abstractly and study their behavior logically without needing actually to build them. When you felt you had got it abstractly and logically right, but only then, you could go ahead and realize the abstractions in actual hardware. So

10 CLAUDE SHANNON

A MATHEMATICAL THEORY OF COMMUNICATION, 1948

- The first deep analysis of the concept of **information in the abstract**. Separated information (and computation) from its embodiment in hardware. Emphasized **representation** and **coding**

A UNIVERSAL TURING MACHINE WITH 2 INTERNAL STATES, 1956

- Ingenious construction to convert **any** Turing machine with n states and m symbols into an equivalent Turing machine with 2 states and no more than 4m (n+1) symbols.
- Shannon and Turing surely also talked about Shannon's ideas on digital communication and Turing machine theory during their two months of "speech scrambling" discussions at Bell Laboratories in early 1943

11 W. S. McCULLOGH AND W. H. PITTS

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY, 1943

- **artificial nerve networks** are general abstract digital switching circuits
- McCullogh and Pitts cited Turing's universal machine, pointing out that **any** Turing machine can be implemented as an artificial nerve network
- artificial nerve networks were used by von Neumann and Turing in their 1945 designs and in expounding the EDVAC, IAS and ACE computers
- Kleene 1951 introduced the concept of **regular expressions** and proved that they exactly represent those events which are recognizable by McCullogh and Pitts nerve networks

McCullogh-Pitts nerve-networks became a computer design methodology for von Neumann and others. Turing, in fact, in his ACE report, also adopted this notation.

Some of the specialists in the audience might also be aware of the connection between McCullogh-Pitts notation and the so-called *regular expressions* of abstract computer science.

Slide 12 concerns an influential pioneer of the logical approach to both computer science and artificial intelligence—John McCarthy—who is still active in research. He is perhaps most widely known and honored for his invention of the immortal programming language LISP (literally immortal: people will no doubt be writing LISP programs centuries from now!) Historically, LISP was the first great *logic programming* language. It is essentially a system of formal logic, the so-called *lambda calculus* of Church, with only a few additions and modifications. McCarthy has had a big influence on our thinking, starting with the first paper mentioned in the figure. This paper expounded the logical ideas underlying LISP, but went far beyond that in studying the role played by inductive definitions and recursion. The now-familiar conditional expression was McCarthy's invention, introduced in that paper. It is a very important paper. It was not only in the theory of computation and programming languages, in this way, that McCarthy started things rolling, but he gave a strong impetus to the logical approach to artificial intelligence. His 1958 paper, interestingly, was first given at National Physical laboratory in Teddington, England where Turing had worked for a brief time writing the ACE report. McCarthy proposed this wonderful idea of a program to which you could simply *tell* knowledge. The program would be in effect the embodiment of a axiomatic deductive system—a deductive knowledge base which would be the platform for planning and for controlling actions in a robot, and so on. A commonplace idea now, but here is where it began. It is almost as if Turing handed the logical torch in 1954 to the young McCarthy, fresh out of graduate school, as indeed also was Marvin Minsky, about whom we shall soon say more, and who was setting out at that time to "make a brain" in the spirit of Turing, but not by a *purely* logical approach.

We must of course, at some point mention logic programming!

The famous equational aphorism in slide 13, coined by Professor Kowalski and often cited, succinctly points out that in programming you have to separate two components: the knowledge you are using and the way you are using it. You can then deal separately with each component. It must be emphasized that *there have to be both components*. You cannot ignore either of them. We can find both extremes in actual practice. Some people throw all the logic away and leave only control behind, by writing purely imperative programs. Oddly enough, programming Turing machines is like that. You simply tell them what to do, step by step—giving the control only.

As for using logic only: perhaps it is only the logicians and philosophers who have done this in a pure form. When one organizes knowledge in this

12 JOHN McCARTHY

A BASIS FOR A MATHEMATICAL THEORY OF COMPUTATION 1961

- *computer science is the science of how machines can be made to carry out intellectual processes*
- *it is reasonable to hope that the relationship between* ***computation*** *and* ***mathematical logic*** *will be as fruitful in the next century as that between analysis and physics in the last*

PROGRAMS WITH COMMON SENSE, 1958

- *The ADVICE TAKER is a proposed program for solving problems by manipulating* ***sentences in formal languages****.*
- *A program has common sense if it automatically* ***deduces*** *for itself a sufficiently wide class of immediate consequences of anything it is told and what it already knows*

way, nothing *happens*. It is just a Platonic and timeless set of propositions, some of which are axioms and the rest of which are theorems. These are the—static—consequences of the axioms, whether you derive them or not. Nothing happens. That is not a basis for computation. In computation, something actually happens. PROLOG, for example, illustrates both the Kowalski equation and the Colmerauer maxim: just describe it, simply say what it is, and then the computing system will take that description and will build it, if it is an object, or will do it, if it is an action. That's very neat way of expressing what logic programming feels like.

The logical origin of PROLOG is quite gratifying to me personally because it is based on ideas that I happen to have been involved in developing in the early 1960s—the *resolution* versions of the first-order predicate calculus. However, the history of logic programming goes back much earlier, to the early 1930s when Turing was an undergraduate in Cambridge. Jacques Herbrand and Kurt Gödel had just recently published their seminal work. Herbrand's particular contribution to logic was a specific precursor of what we now call computational logic. In Herbrand's Ph.D. thesis of 1931 we find, for example, what we now call *unification*. It is somewhat obscurely described, but definitely it was he who first thought of that. As we all know now, as for example in Colmerauer PROLOG-3, these ideas are being much generalized and extended, so that instead of just unification we now think more generally of *constraint solving* as being the guiding principle

13 LOGIC PROGRAMMING

- *Algorithm = **Logic** + Control* (Robert Kowalski)
- *Just **describe** it!* (Alain Colmerauer)
- logic programming in PROLOG is based on the Horn-clause special case of my 1965 **resolution** version of the **predicate calculus** which exploits Herbrand's **unification** process. Colmerauer's Prolog III goes beyond basic Prolog by replacing **unification** with **constraint solving**
- Logic programming in LISP uses a different logic, based on McCarthy's version of Church's lambda-calculus

behind logic programming systems. The basic idea is still the same. And whether we like it or not (and I don't mind at it all) LISP must be classified as another logic programming language: it is just that the logic is a different logic, the lambda calculus of Alonzo Church. Thus McCarthy was so to speak the Kowalski and Colmerauer of LISP, because he saw what you could do computationally with that abstract logical system.

Slide 14 is a list of names, in roughly chronological order, divided into three groups to suggest the way the history of computational logic has taken place. Frege invented the predicate calculus almost out of thin air, and Hilbert presided over the first several decades of its development at the start of this century. Von Neumann belongs in there because of his immensely important contributions to the logical formalization of set theory. Post, Godel, Church, Curry, Kleene and Rosser were all pioneers in the 1930s in developing logical systems which probed very deeply the logical theory of computation and computability. Most of the central results of the field were obtained before 1940. Turing's work was of course a major part of this. In about 1955 the computer began to be widely available for all kinds of research, including computational logic. Starting then, there was very important work done over a period of 15 years by Wang, Gilmore, Davis, Putnam and Prawitz, all of whom were directly influential on me and others of my generation. It was they who taught us, and who intrigued us with the enormous possibilities that they could see. They did not quite work these out for themselves, but they certainly laid the framework for what soon became possible. In the third group of names I put the names of Bledsoe and myself first because of our ages—the others in the group are all younger. Wos and George Robinson in the mid-1960s founded the Argonne group, which has since gone on to fame and success. Loveland's *Model Elimination* version of the predicate calculus is essentially the same as the Kowalski-Kuehner SLD-resolution underlying PROLOG. Huet generalized the unification algorithm to the higher order predicate calculus, a very important step towards higher-order logic programming and theorem proving. Woody Bledsoe has spent his career as a professional mathematician specializing in computational logic theory and in developing systems for proving hard 'real' theorems on the computer.

Kowalski and Colmerauer have given us logic programming in the predicate calculus sense, and we must, I think, mention Keith Clark here, who has made such an important addition to the abstract logical framework for logic programming with his identification and explication of the *negation by failure* concept as well as his pioneering work in concurrent logic programming. There are of course many more names which should be included as we come down to the present day, but we cannot undertake to write the complete history of computational logic in a two minutes. I just wanted to remind you of some of the people who were involved in computational logic and how various have been their contributions.

14 COMPUTATIONAL LOGIC

1879-1935	GOTTLOB	**FREGE**
	DAVID	**HILBERT**
	JOHN	**VON NEUMANN**
	JACQUES	**HERBRAND**
	EMIL	**POST**
	KURT	**GÖDEL**
	ALONZO	**CHURCH**
	ALAN	**TURING**
	HASKELL	**CURRY**
	STEPHEN	**KLEENE**
	J.BARKLEY	**ROSSER**
1955-1960	HAO	**WANG**
	PAUL	**GILMORE**
	MARTIN	**DAVIS**
	HILARY	**PUTNAM**
	DAG	**PRAWITZ**
1960-present	WOODY	**BLEDSOE**
	ALAN	**ROBINSON**
	LARRY	**WOS**
	GEORGE	**ROBINSON**
	DON	**LOVELAND**
	GERARD	**HUET**
	ROBERT	**KOWALSKI**
	ALAIN	**COLMERAUER**
	KEITH	**CLARK**

Slide 15 notes that the idea of the *logical interchangeability* between hardware and software, an idea which we all take for granted, is really just another version of Turing's universal machine concept: you only need to put a certain minimal 'something' in hardware form so that everything else can then be realized as software, created simply by programming that hardware. In today's terms, you meet this idea in *reduced instruction set computers*, which are almost a serious engineering form of the game of minimizing the universal machine, of seeing how small you make it. How much hardware do you really have to have, at a minimum, in order to be able to do everything by programming? We are reminded of Turing, then, even by the RISC concept.

Slide 16 mentions a practical way in which the distinction between real and abstract computers turns out to be really very important. For von Neumann in particular it became a very big thing. He became involved in litigation and some very unpleasant friction, with the ENIAC/UNIVAC designers Eckert and Mauchly.

What can you get a patent on? The law has always been generally that you can only patent hardware. You can't patent an *idea*. Yet, it is often the ideas which drive progress. In the balance between hardware and software, or between abstract and real, one would like to acknowledge the importance of the abstract side. In order to provide an incentive to concentrate on abstract invention would it not help if abstract inventors could obtain a patent on their discoveries? It is noteworthy, for example, that Turing could not have patented the universal machine idea, which is one of the most important and fruitful discoveries of the 20th century or indeed of all time.

Von Neumann's expository notational technique, as I have already mentioned, in setting out the design of the EDVAC, was to throw away all the hardware ideas and just leave behind the McCulloch-Pitts nerve-network description of the abstract layout and function of the digital computer, including the stored program idea. It was this which caused the friction with Eckert and Mauchly, whose engineering ideas he abstracted away. They resented that, since they felt that he was leaving them out of the picture and denying them credit for their innovations. In some sense he was indeed doing this, but his motives were completely different from those attributed to him by Eckert and Mauchly. He was not all averse to assigning credit to others—for example, to Turing—when he thought he was himself being unduly praised.

From this distance in time we can see how very important Eckert and Mauchly were, because it was they who made it possible to build a real machine that would go very fast, and all rest of it . In the lawsuits which have subsequently taken place concerning the contributions of Eckert, Mauchly, Atanasoff, and von Neumann the ques-

15 INTERCHANGEABILITY OF HARDWARE AND SOFTWARE

- first demonstrated theoretically by Turing in his 1936 paper
- 1945 ACE was a 'minimum hardware' machine
- 1945 EDVAC minimized parallelism for hardware economy
- this interchangeability concept is a commonplace today:
- RISC computers
  - interpreters instead of special architectures
  - simulation
- the basic question: what should be in **hardware** form, and what in **software** form?
- or: which one of the many different kinds of **universal machine** should we use?

16 PATENTABILITY OF REAL BUT NOT OF ABSTRACT MACHINES

- A basic legal fact about patentability: one can patent only *physical devices* (hardware)—an *idea or scientific principle* cannot be patented
- Turing could not have patented his universal machine!
- von Neumann's technique in the EDVAC Report was precisely to abstract *from* hardware and engineering specifics and present only the essential **idea** in abstract form using McCullogh-Pitts diagrams.
- This led to friction and even to litigation:
  - Eckert & Mauchly vs von Neumann
  - Eckert & Mauchly vs Atanasoff
- ***abstraction of logic from engineering*** *enabled von Neumann to do the logic of EDVAC without simultaneously doing the engineering, and thereby made it possible for him to essentially complete the design in one draft* Arthur W.Burks,1980

tion was *who invented what, and when*? There is no doubt that it was *hardware* that was being argued about. Arthur Burks, one of the people who worked directly with von Neumann, points out that it was precisely the abstraction of the logic from engineering which enabled von Neumann to do the logic of EDVAC *without simultaneously doing the engineering*, and therby made it possible for him to complete the design *in one draft*. So as a part of computer design methodology the distinction between real and abstract computers is, to put it mildly, a very important one.

In slide 17 Metropolis and Rota, two people who knew von Neumann very well, tell us that Turing and von Neumann were among the few who realized, in the 1930s, that mathematical logic was the magic key to both programming languages as well as computers. More recently, Dijkstra has emphasized tirelessly that if you look properly at programming and at computer science what you

17 LOGIC AND TECHNOLOGY

N.Metropolis and Gian-Carlo Rota, 1980:

- *Few except Turing and von Neumann realized in the 1930s that mathematical logic was* ***the magic key to programming languages as well as to computer design***
- *The symbolism of Peano, Russell and Whitehead, the analysis of proofs by Gentzen, the definition of computability by Church and Turing, mark* ***the beginning of the computer revolution***

E.W.Dijkstra, 1981:

- *Logic has changed from a descriptive science into a prescriptive one: the* ***new logician is an engineer***
- *It is no longer the purpose of programs to instruct machines, it is* ***the purpose of machines to execute programs***

see is, in effect, *logic becoming a branch of engineering*. It is however the *abstract engineering* that you do when you program. In programming you no longer wait for the engineers to come and say *here is a machine we have built, what can you program it to do*? As Dijkstra points out, it is now the other way round. The programmer in effect says: *this is the kind of machine we would like; please build it for us.*

(Slide 18). Programmers have always learned to program *abstract* versions of machines. The actual physical machines have never really been 'visible'.

A similar situation prevailed in the case of the early 'higher level' programming languages such as FORTRAN (slide 19) and LISP (slide 20). In this case the illusion enjoyed by the user was that there was 'back in there somewhere' a *machine* which directly embodied the rules of the language and which followed the steps of the evaluation algorithm when given a program express to process.

The fact that this was an illusion was of course known to us, but little or no use was made of this knowledge except, perhaps, for the pragmatic but (it now seems) disreputable practice of debugging the compiled programs with the aid of large memory dumps.

One's programs were written according to the rules of certain abstract formal systems, presented in programming manuals, and it was a *relatively* unimportant fact that these programs could then be actually run, at very high speeds, on real (or simulated) hardware versions of these abstract formal systems.

(Slide 21). The same point of view can be taken in the case of all flavors of logic programming: underlying the practicalities of actual computation is an abstract *rewriting machine* with which one can pretend one is dealing directly and which is essentially equivalent to a formal logical system.

It is of course the same today, only more so (slide 22). Programmers really don't care, and often don't even know, what hardware their programs are running on. They are dealing with abstract machines, not real ones.

18 AN **ABSTRACT** OR A **REAL UNIVAC**?

- When I joined the du Pont company in 1956 my first assignment was to learn how to program the Eckert & Mauchly [Remington Rand] UNIVAC.
- All the relevant definitions and rules (thirty or so instructions, the logical structure of the registers and memory, and so on) were given in the UNIVAC PROGRAMMING MANUAL. I studied this **formal system** and thus learned to program the **abstract UNIVAC**.
- The du Pont company's **real UNIVAC** was kept behind locked doors. To run UNIVAC programs, we handed them to the receptionist.
- For all we knew, there might have been no real UNIVAC behind those locked doors. What if the company had simply hired a team of very fast humans to **simulate** it? Except for the running time, how could we have detected the difference?

19 THE ABSTRACT **FORTRAN** MACHINE

- In 1959 FORTRAN was a wonderful liberation from machine-language programming. Now one could program in a much higher-level and more natural language.
- To run FORTRAN programs, we again simply handed them to the receptionist, but now we knew for certain that there was no **real** FORTRAN machine behind those doors.

20 THE ABSTRACT **LISP** MACHINE

- John McCarthy and his MIT team designed the LISP language and the **abstract LISP machine** in order to be able to compute directly and naturally with recursive functions and data structures. At that time, real LISP machines did not exist.
- LISP was an even greater liberation than FORTRAN. It was at a much conceptual higher-level, directly based on a powerful formal logic—the lambda-calculus of Alonzo Church—and was an excellent approximation to a working version of this exceedingly abstract logical calculus.
- Later, **real** hardware LISP machines were indeed actually built, but ironically they did not survive competition from the **abstract** LISP machines simulated efficiently on Sun work stations and other general purpose computers.

21 IDEALIZED NORMAL FORM MACHINES

- These are the abstract 'reduction' machines implicit in e.g. the lambda-calculus and other **rewriting** systems such as Curry's **combinatory logic**
- The **states** of a normal form machine are expressions of a formal language

states = expressions
transitions = rewritings
terminal = unrewritable = in normal form

- to use such a normal form machine, one types in an expression as **input** and receives as **output** an equivalent expression in normal form
- this is the paradigm of all **descriptive** or **declarative** computation

22 INTERPRETERS AND COMPILERS

- The software technique of **simulating** one machine on another machine has made it possible to deal directly with abstract machines as though they are real
- this is a form of 'virtual reality'
- Today's FORTRAN, LISP, PROLOG (etc.) users can expect to be able to run their programs on almost any real machine. The user can simply ignore the question of what the actual hardware is that supports **the illusion that the abstract machine exists**.

Landin long ago brilliantly made the case (slide 23) that all modern programming languages are essentially just the lambda calculus in one or other shallow disguise, and that this will continue to be so far into the future. Perhaps we would today want to generalize his thesis to say that programming languages are and always will be really *systems of logic in disguise*.

The consequences of distinguishing between *abstract* and *real* machines extend even to the administrative patterns in educational and research organizations. It has become common to distinguish between computer science and computer engineering (slide 24).

In AI, itself, a similar distinction—between real and abstract humans—sheds light on much that goes on in the field. Slide 25 shows a couple of quotations from the writings of Marvin Minsky, who has stood all along for the relative unimportance of logic for AI.

Minsky's view is that logic is only one among many principles and ideas which are important for AI, and one must surely agree with him. It seems to me that if the mission of AI is to reproduce what it is that we find in real humans that we call *intelligent* thinking and behavior, then logic is indeed only a small part of it. Logic is based on a kind of retrospective reconstruction by logicians of what minds can do. But the mind doesn't seem actually seem to do it that way. We really don't yet know how it does it. Logic doesn't explain, as Minsky says, *how* we think. It's only a very artificial model of thinking which does not fit the facts. As you know, many people have argued passionately for and against this idea. McCarthy still insists that you could, even though nature does not seem to have done it this way, you *could* have an intelligent thinking agent which put logic much more in the center of the picture. The beauty of this claim is that it can be tested, and it will be tested. We will eventually know whether McCarthy is right about this. We just need a few more decades, I think, of trying to make it happen.

Slide 26 suggests that the distinction between real and abstract humans is in fact a very useful methodological one which explains much of what is done and written in artificial intelligence and cognitive science research. As long as we keep

23 PETER LANDIN'S, THESIS

THE NEXT 700 PROGRAMMING LANGUAGES, 1964

All programming languages are nothing but the lambda-calculus in disguise. They are simply the lambda-calculus, ***sweetened*** *with* ***syntactic sugar***

- Landin's own SECD machine is a sophisticated normal form machine for the lambda-calculus which uses four stacks to decompose and reassemble the structure of the expression being reduced to normal form

24 COMPUTER SCIENCE AND COMPUTER ENGINEERING

- **Computer science** is a logical discipline whose subject matter is that of **abstract computers** and **abstract computation**. Since one can always simulate an abstract machine, there is really no need to build a hardware version of it just to see how it behaves.
- On the other hand, **computer engineering** is all about **building real machines** and improving the necessary **hardware technology**.
- This discipline has produced miracles and shows every sign of continuing to do so

25 LOGIC AND PSYCHOLOGY

- Marvin Minsky, 1985:
- *I do not mean to say that there is anything wrong with logic; I only object to the assumption that ordinary reasoning is largely based on it*
- *Logic is only a small part of our great accumulation of different, useful ways to chain things together*
- Minsky would agree that our **essential** programming knowledge is **abstract** and **logical**
- Turing's **fundamental axiom of artificial intelligence** is that our essential knowledge of other humans, and even of ourselves, is of this same character. This is the idea behind the famous **Turing Test**

clear the difference between trying to understand and explain how, on the one hand, nature has in fact managed to create intelligent creatures and, on the other hand, how we might artificially create them, using whatever ideas and technologies are available and appropriate, there need be no controversy. Even so, I suppose there are those who prefer to believe that even in the attempt to make intelligence happen in artifacts, the best approach to follow is to imitate nature as far as possible.

Slide 27 indeed suggests that in the case of so-called expert systems it is probably the 'artificial' expert systems, using artificial techniques and unnatural methods, which will continue to perform better than those based on actual human expertise.

Finally, here are two quotations (slide 28) which seem to me to characterize very well the future relationship of logic to both to computing and to artificial intelligence.

Martin Davis is a distinguished pioneer of computational logic whose own contributions, starting in the early 1950s, have continued to be extremely influential in the development of logic programming and automated deduction.

Helmut Schnelle is a cognitive scientist who has closely studied von Neumann's attempt (in his theory of self-reproducing cellular automata) to provide a formal but 'natural' logical reconstruction of living and intelligent organisms. He finds, in the different approaches of Turing and von Neumann to this problem, essentially the same distinction between abstract and real that we have

26 ABSTRACT AND REAL HUMANS

- McCullogh-Pitts notation is intended as an abstract model of the biological neuron and thus there is an implicit claim that a **complete brain and nervous system** can be reproduced artificially as a suitable network of these elements
- *The gap between human and artificial intelligence is still immense. Gödel and Post believed that a satisfactory theory of mathematical intelligence must take account of nonfinitary and creative reasoning.* ***Should we hope for another Turing*** *to produce such a theory?*
- Turing's use of evident human limitations, the discovery by McCullogh and Pitts that neuronal nets can act as finite automata, and the rapid increase in the power of computers, have led to a widespread acceptance of **computers as the model for the workings of a central nervous system** and for intelligent behavior.

Robin Gandy, 1988

27 LOGIC AND EXPERT SYSTEMS

- **expert systems** are models of only very narrow and limited subsets of human knowledge
- it is ironic that **high-level expertise** (such as that of an equation solver, or a medical diagnostician or a mass spectroscopist) is easier to formalize as a collection of logical rules and axioms than is a **low-level skill** (such as riding a bicycle or driving a taxi)
- most human cognitive skills are **intuitive** and **subconscious**
- interviews with experts will not uncover their expertise if the experts cannot explicitly articulate the **intuitive methods** which they use
- **natural** expert systems will therefore probably not in future be as common or as effective as was once expected
- **artificial** expert systems need not be based on human originals—for example, **Deep Thought**, or Stiller's completely **inhuman** chess endgame programs—in order to be effective

28 LOGIC AND THE FUTURE

- Martin Davis, 1984:
- *the connection between **logic and computing** continues to be a vital one, and the lesson of **universality**, of the possibility of replacing the construction of diverse pieces of hardware by the programming of a **single all-purpose device** continues to be relevant.*
- Helmut Schnelle, 1988:
- *Turing's work provided an important starting point for von Neumann. Turing seemed to believe that his models were sufficient for a **logical and practical understanding of human behavior**. In contrast to this, von Neumann thought that insight into **architecture and composition of organisms** is essential, at least where proofs of feasibility in principle are not sufficient but practical understanding is required*

been concerned with throughout this lecture.

Let me then try to summarize the main points of my lecture (slide 29).

I have given reasons why I believe that logic is at the heart of computer science. I have also said why, although it is an important part of artificial intelligence, I believe it is not the central one. As time goes on, I am sure that we shall see logic become even more dominant in computer science. In artificial intelligence we will simply see it taking its place as one among many other relevant disciplines: psychology, neuroscience, linguistics, and so forth. I have maintained that in computer science and in the history of the computer, there have been only two really outstanding figures, Turing and von Neumann. They have both dominated the scene from the beginning, but it is Turing who was scientifically the more important of the two. His is the one single name which epitomizes the whole of computer science. As far as artificial intelligence is concerned, there is no doubt that Turing stands alone, far above the rest.

It is therefore fitting to remind ourselves that June 23, 1992 is the 80th anniversary of Turing's birth. This splendid conference—the grand finale of the great Fifth Generation Computer Systems

29 CONCLUSIONS

- **Logic** is, and always has been, a major theme in both **computer science** and **artificial intelligence**
- In the historical development of the **computer** and of **computer science**, we find that **logic** and **technology** have collaborated fruitfully
- contrary to popular beliefs, however, **logic** has always been very much the **leading partner** in the collaboration
- In the future of computer science, this partnership will continue and become even closer, but **logic** will assume **even greater importance** than in the past
- On the other hand, in **artificial intelligence**, logic shares the field with many other disciplines: **psychology**, **neuroscience**, **linguistics**, **robotics**, **cognitive science**, to name only a few, and it is of only limited relative importance
- In computer science in general, Turing and von Neumann are the **outstanding historical figures**, with Turing as the primary figure.
- In artificial intelligence, Turing's role is **unique**.

Project—and what it represents for all of us, is not only a most appropriate celebration of Turing's birthday, but also a direct commemoration of his remarkable scientific contributions and the ideas that he created and championed. He would surely have warmly appreciated the occasion.

Thank you.

Question: I appreciate your talk very much. I do have one comment about understanding the role of logic in computer science. It is that we may perhaps compare it with the contrasting roles of rationalism and empiricism in philosophy. Logic is the rationalism aspect of the discipline of computer science. Logic programming tries to express everything rationally, in terms of logical reasoning. However, the real world is empirical, and in trying to understand how the real world is structured we may have to use extra-logical, empirical techniques. For example, in computer science, object-oriented programming is an empirical approach to modeling the world, whereas logic programming is a rationalist approach. So as we do applications I think that we have to go beyond logic in computer science as well as in artificial intelligence.

Answer: Yes. I agree with that. It's essentially what I think Minsky is telling us in the particular case of understanding human behavior, but it also applies more generally across the board to all natural phenomena and processes. What you find in nature probably doesn't have much logic in it. An important thing to stress, however, and one which doesn't come out in your analogy, is that logic may well have a much more important part to play in *artifacts* (which, by definition, don't occur in nature). We ourselves design and build artifacts, and in doing so we are not necessarily limited to the ideas and techniques which nature has evolved for herself. In the *sciences of the artificial*, to use Herb Simon's phrase, logic can take a much more central role, and it does, and it has, and it will. That's the part of my thesis which I would like to emphasize. Computers are not found in nature: we build them. In doing so we can therefore use logic as our major guiding idea, and make them, as you say, rational.