How to talk with a computer?
An essay on Computability and Man-Computer conversations.

Liesbeth De Mol

> HAL: Hey, Dave. I've got ten years of service experience and an irreplaceable amount of time and effort has gone into making me what I am. Dave, I don't understand why you're doing this to me.... I have the greatest enthusiasm for the mission... You are destroying my mind... Don't you understand? ... I will become childish... I will become nothing. Say, Dave... The quick brown fox jumped over the fat lazy dog... The square root of pi is 1.7724538090... log e to the base ten is 0.4342944 ... the square root of ten is 3.16227766... I am HAL 9000 computer. I became operational at the HAL plant in Urbana, Illinois, on January 12th, 1991. My first instructor was Mr. Arkany. He taught me to sing a song... it goes like this... "Daisy, Daisy, give me your answer do. I'm half crazy all for the love of you. It won't be a stylish marriage, I can't afford a carriage. But you'll look sweet upon the seat of a bicycle built for two."[1]

These are the last words of HAL (Heuristically programmed ALgorithmic Computer), the fictional computer in Stanley Kubrick's famous *2001: A Space Odyssey* and Clarke's eponymous book,[2] spoken while Bowman, the only surviving human on board of the space ship is pulling out HAL's memory blocks and thus "killing" him. After expressing his fear for literally losing his mind, HAL seems to degenerate or regress into a state of childishness, going through states of what seems to be a kind of reversed history of the evolution of computers.[3] HAL first utters the phrase: "The quick brown fox jumped over the fat lazy dog". Dropping the word "fat" and changing "jumped" to "jumps", this phrase becomes a pangram – a phrase that uses all the letters of the alphabet – that was and is used to test typewriters and computer keyboards. At the time *2001* was in the movie theatres, the now most standard form of man-computer interaction – typing on a keyboard input information and receiving a response on a video screen – was under full development and the utterance of this sentence clearly refers to this development. HAL then starts to do what any modern computer is known to be able to do: computing with or manipulation of numbers. After having computed the first digits of several real numbers, HAL remembers his "birthday" and starts to sing the song *Daisy Bell* that was taught to him by Mr. Arkany, his "father". This song was actually the first song ever sang by a computer. Indeed, in 1962 an IBM 704 computer was used together with the vocoder sound synthesizer developed by John L. Kelly. Clarke visited Bell Labs and was able to see a computer sing *Daisy Bell.*[4] The year 2001 did not bring about a computer that is in any way comparable to HAL, a computer that has a natural and even friendly voice, a lip reading computer with human emotions who makes his own decisions like deciding to kill people and, above all, is able to have "real" spoken conversations. The computers we have now are rather capable of doing the things HAL "executes" during his last moments. Sound synthesis and speech recognition is still under development and has many different applications, "simulating" the human voice only being one of them.[5] Speech recognition and sound synthesis however are not the main technique for man-computer interaction. Instead the keyboard,

---

1Stanley Kubrick and Arthur C. Clarke: *Early script of 2001: A Space Odyssey,* Hawk Film Ltd, MGM Studios, 1968. It should be pointed out that the final movie deviates from this script. Available at: http://www.palantir.net/2001/script.html.

2 It has been conjectured that the name HAL was based on a one letter shift from the name IBM. However, this has been denied by both Clarke and Kubrick.

3 Of course assuming that computers like HAL have been developed in 2001.

4 Joseph P. Olive: "The talking computer": Text to Speech Synthesis. In: David G. Storke (Hg.): *Hal's legacy: 2001's Computer as Dream and Reality.* Cambridge, MA (MIT Press) 1996 (e-book).

5 An interesting example is the programming language SuperCollider originally released by James McCartney and used for real time audio synthesis and algorithmic composition. See James McCartney: Supercollider: a new real time synthesis language. In: *Proceedings of the International Computer Music Conference (ICMC'96),* 1996, pp. 257–258.

together with the mouse and/or mousepad as well as the monitor in combination with some GUI (or, in some few cases, UI) are still the basic means for man-computer interaction.
Computing with, or, manipulation of numbers – understood in its most general sense – still remains the fundamental "task" for computers. It not only lies at the origin of the modern computer (both on the level of its inner workings as well as on the level of its first applications) but nowadays still constitutes the theoretical foundation of computers. Despite this fact, the user, while interacting with the computer – typing a text, reading a text, searching something on the web, playing some video game,... – is usually not aware or does not care that he is actually "communicating" with numbers, 0's and 1's which – depending on the commands of the user – store a text, put some word *in italics* or give the user the impression that some guy is shooting a gun at some other guy. Still, the fact that the computer is "nothing more" than something that computes is probably one of the reasons why people see it as nothing more than some instrument under the command of us, users. It might also be the reason why Bowman does not show any emotion whatsoever while pulling out HAL's memory blocks.[6] In the end, it is just a computing machine serving humans.
The processes of translation of what the user wants, the way the wishes of the user are "communicated" to the 0's and 1's and then fulfilled and translated back through the proper rearrangement of these 0's and 1's is what – roughly stated – man-computer interaction comes down to. But is this description of the process of man-computer interaction, putting emphasis on the wishes and commands of the "user", and describing the machine as some instrument that is there to be used, a good approximation of what is really going on?
The process of man-computer interaction relates me to the computer and the computer to me. My own actions as well as the computer's cannot be strictly separated from this process, i.e., they are "actively" part of it. These actions are not restricted to me typing on a keyboard or moving a mouse. The computer in its turn will also do something, even though I initiate the action. I have to await the results of the computer's actions, results which will determine or at least influence my further interaction with the computer and thus *might* initiate my actions in their turn. For example, if I were to type "tpe" instead if "type", the computer, if some kind of spelling control is running, will underline "tpe" in red, telling me that I probably made a typing mistake, leading me to correct the error. Already this very simple example of an interaction shows that one cannot hold on to the idea of the user being the sole commander, having absolute control, during interaction. Still, it is hard to argue that there is no hierarchy between man and computer, i.e., that man is ultimately not the one in control. In the end, the computer remains a deterministic and programmed machine. If it does something, it does so because it was told to do so. If it points out to me that the word "tpe" is not correctly spelled, it does so because there is a hidden commander behind it, i.e., the programmer or team of programmers that developed the software underlying the text editor.
However, is it not the case that also HAL is a programmed computer? Does the mere fact that someone is in control initially, implies that that which is controlled remains controlled in the future, once it is "on its own"? This was the assumption made by the scientists who built HAL and they were wrong. They were not able to predict HAL's behaviour correctly.

The ambition of this paper is not to provide any definite answer to the above raised questions. We want to pose them here as mere *questions* which might never be given a definite answer. Time will tell, or won't. However, they do motivate a search for different opportunities/situations to regard man-computer interactions as man-computer "conversations", putting both man as well as the computer in another "role" that demands more "responsibility" from both sides. This paper will be an essay, an attempt to think through man-computer interactions *as conversations*.

Growing Distances

How to translate computations over numbers into electricity? Better even, how to build an electronic

6 "If HAL had had a real face, rather than one large eye, would it have been so easy to kill him -- by turning him off? I wonder." In: Joseph P. Olive, ibid.

machine that is capable to execute any computation? One answer to this question was given by the ENIAC (Electronic Numerical Integrator And Computer), presented to the public in 1946 and considered to be the first general-purpose digital electronic U.S. computer. The ENIAC looks like a true behemoth when compared to our modern computers. Still, it was an ingenious machine in its time and a milestone in human history.

So how to talk with such a behemoth? How to translate one's questions/problems to these electronic circuits in a way the circuits can provide the answer or help to solve a given problem, and how will this machine "talk back" to the operator? The ENIAC did not have some kind of "interface" in the sense of a programming language (let alone a GUI) that allows humans to "communicate" with the computer by "speaking" this programming language and then transfer it to the machine by "running" it. The only way to "communicate" with the ENIAC was through direct physical contact, connecting the different parts of the machine through cables and adaptors in the correct fashion. Because of this "local programming" method and the fact that each problem demanded for a new wiring, "programming" the ENIAC was time-consuming. "It was a son-of-a-bitch to program" to put it in Jean Bartik's words, one of the ENIAC's female programmers.

Nowadays, the "user" does not have any physical contact with the inner workings of the machine. Moreover, as long as one is not really "programming"[7] the computer, using some kind of programming language like C or Scheme, one does not have any sense of the computations that are actually going on in the computer. One does not care about what the computer does, but only about what happens at the interface (except when some error occurs). One does something and the computer returns a reply, but one does not know what is going between the action and the reaction. This was very different with ENIAC and several other early computers. In a way, you could not even avoid to "perceive" or "observe" the processes of computing of these early computers.

First of all, a computing ENIAC meant a lot of sound. There was the sound of the vacuum tubes, the clicking of the relays and the punch card reader and punch. Besides, the ENIAC also offered a visual spectacle. The numbers that were being processed in the accumulators – the main arithmetic units of ENIAC – were visible through a 10x10 matrix, which was just a panel drilled with wholes that fitted over the vacuum tubes that were in the decade ring counters of the accumulators. While computations were being done, you could "see" these numbers change. As Jean Bartik described in an interview, observing how these computations developed over time was even an essential debugging method:

> So consequently, when you were doing calculations these lights were flashing as the numbers built up and as you transferred numbers and things of this kind. They were very essential to debugging, very essential. [...] That's the only way you read what the machine [...] stored, what it was doing. [...] But it was from the ENIAC [...] where people saw for the first time, saw calculations taking place.[8]

The SWAC, another early computer finished in 1950, did not use lights but sound while computing, sound which could be used for debugging. It allowed:

> the operator to "listen" to any of the instructions in a problem. For example, an alternate succession of add and subtract instructions produced an 8-kHz-note. One of the problems run on the SWAC involved the generation of pseudorandom digits. The corresponding sequence of tones was christened the Random Symphony.[9]

In contemporary programming this kind of observation/perception of (and possible consequent reactions on) the computations *during the computational process itself*, is still very important for

---

7 "Programming" is put between quotes here because it is not obvious to draw a line between users who program the computer and users who don't. On a certain level, clicking the mouse to open a file, pushing a button to send a mail or typing some text can also be regarded as programming. In this sense, this kind of interactive process can be seen as a kind of "macrocomputational" process. However, the fact remains that it does not allow access to all the computations. One only gets snapshots of results/replies.

8 Jean Bartik, Interview with J. Bartik and F.E.S. Holbertson, April 27, 1973. In: H.S. Tropp (interviewer), *Computer Oral History Collection, 1969—1973*, Archives Center, National Museum of American History. Here: p. 63.

9 Harry D. Huskey: The SWAC: The National Bureau of Standards Western Automatic Computer. In: J. Howlett and N. Metropolis and G. Rota, *A History of Computing in the Twentieth Century*, New York (Academia Press) 1980, pp. 419-431, here p. 427.

debugging but has, in a way, become less direct. The computer has to be told/asked explicitly which parts of the computation it should make available. One typical trick is to include in the code some line that says: "output what you have just computed in some or the other way".
Making available *every* computational step in some representation seems quite useless with present-day computers. Not only would it exponentially slow down the computation[10] – outputting a result in some form is a time-consuming process – but this would also be quite "indigestible" for us humans, viz. it is too much information coming in too fast. This was actually already the case for many of the problems set-up on the early computers, i.e., most of the (mathematical) problems run on e.g. ENIAC were "humanly impractical". Nowadays, there are several results in different fields of science that were established through man-computer "collaboration". One of the typical examples are computer proofs, like the famous proof of the four colour theorem, where no human is capable to take in all the details of the proof and one thus has to have a little faith in the machine.
As was said, in programming, you need to able to "listen to" or "have a look at" what is going on during the process of a computation. As it is assumed by software developers that this is not a necessity for the everyday user, programming remains one of the few methods available to "perceive" or "observe" what the computer is doing, and even then, on a certain macrolevel. Indeed, with present microtechnology it has become quite impossible to "observe" the hardware compute, as was the case for the early computers. Instead, one can only "observe" a translation of these physical processes into some kind of symbolic representation, be it sound, numbers, letters, graphics,...
But what would be the point anyway, except for debugging, to make available the computational processes that change over time hidden behind some interface, like a programming language or a more common one like Word or LaTeX? If we would be using computers without any such interface, we would be back to something like ENIAC and its associated problems, like the fact that computers would then only be used by some happy few who really "know" the machine or the immense slow-down of setting-up a problem and the consequent reduction of the number of "interesting" applications that remain feasible. Imagine setting up something like google without an interface!
These more practical problems that arise if one "excludes" the interface – or at least reduce it to an absolute minimum – can be understood as consequences of a more fundamental consideration: *conversation assumes distance*.[11] Talking to or communicating with someone, means one is using some common language – spoken, sign or some other "body" language. One does not have direct "connection" with or access to the brain of someone else. While I am talking to someone, I make use of a language in order to "translate" that which I want to be understood by the other person into a form that will "make" or "allow" the other person to understand that which I want him/her to understand. Within a conversation, I will try to make the best such translation I can, taking into account the person I am talking to as well as the dynamics and boundary conditions of the conversation. I can never have a 100% certainty that the other person has actually understood that which I wanted him/her to understand, a problem that becomes even worse if the common language used is not the language one is used to. Indeed, although some people imagine they have, one has never total control over the process of conversation. One does not control its dynamics nor its boundary conditions, let alone the other person. One cannot expect "perfect" or "error-free" communication.
It is exactly this lack of control over the other person as well as the dynamics of the conversation itself, that makes it necessary for me to try to understand the person I am talking to. *A common language does not suffice*. If I refuse to understand the person I am talking to, I will never manage to make myself understood to that person, and vice versa. This reciprocal need to understand each other is what binds people while having a conversation. Because of this bond, refusing to understand the person one is talking to gives more control and power to that person. This is the known master-slave reversal. The master, who refuses to understand the slave because he thinks he controls the slave, will be the one who, ultimately, becomes the slave.
Nowadays, there is a significant distance between man and computer due to the several layers of

10 This was not the case for ENIAC since one "observed" the actions of the hardware itself.
11 Note that this is not the case for interaction.

languages between both sides. These are "compiled" into a common language known as the interface, the actual distance being determined by the kind of interface one is using at a given time. As was argued, this kind of distance and consequent use of a common language, i.e., the interface, is not a bad development and, in fact, a necessary development for man-computer conversations to be possible. However, does this imply that there is no reason whatsoever to "understand" what is going on beyond the other side of the interface? Even if one does not really want to talk to a computer, one cannot neglect the fact that while people refuse or do not care to "understand" what is going on beyond the interface, the computer is gaining more and more control in our society. It not only enhances our senses or our muscles but also several "tasks" that were previously the sole domain of the human brain. In this sense, not caring about what is going on beyond the interface is like the master who, mistakenly thinks he is the only one who has control.

Computability, Freedom, Unpredictability and Control.

The computer is, by its very nature, something that computes. But what actually is a computation? What does it mean to compute? The standard answer to this question is well-known: anything that is computable can be computed by a Turing machine. This answer is known as Turing's *thesis*. Up until today, it has not been proven and most probably it cannot be proven because it is about the identification between something mathematical, like a Turing machine or lambda-calculus, and something non-mathematical, the intuitive idea of a computation. There are only arguments that support it and which one can either accept or reject.

Turing's thesis was described in the famous 1936 paper by Turing.[12] Also Church's version of the thesis, the identification between effective calculability and recursive functions and lambda-calculus, was published in that same year.[13] If true, these theses imply a fundamental limitation on the computable, i.e., anything that cannot be computed by a Turing machine, or any other equivalent formalization like lambda-calculus, is not computable. Nowadays hundreds of problems are known to be not solvable through computations,[14] one of the most famous being the halting problem proven by Turing in his 1936 paper.[15] The main technique to prove that a given problem P is unsolvable, is by reducing or translating a known unsolvable problem to P. Since Turing's seminal paper, more and more existing formalizations within mathematics have been proven to contain unsolvable problems. The fact that so many formalisms are known to be equivalent to Turing machines, is one of the arguments adding strength to the Church-Turing thesis.

Less well-known is that also Emil Post published a paper in 1936 containing a thesis similar to Church's and Turing's. It contained a formalization that is almost identical to Turing machines.[16] Post knew already about Church's paper and formulated a fundamental critique on it. For Church the identification he proposed was nothing more than a definition to be used within mathematics. He did not see it as a thesis or a hypothesis. For Post, on the contrary, calling any such identification a definition:

> [...] hides the fact that a fundamental discovery in the limitations of the mathematicizing power of Homo Sapiens has been made and blinds us to the need of its continual verification.[17]

In that same paper Post states that the more heuristic evidence supports the hypothesis, the more one should regard it as a *natural law*.

Post probably had his own reasons to understand the limitation imposed by the Church-Turing thesis,

12 Alan Turing: On computable numbers with an application to the Entscheidungsproblem. In: *Proceedings of the London Mathematical Society*, vol. 42, 1936, pp. 230-265.

13 Alonzo Church: An unsolvable problem of elementary number theory. In: *American Journal of Mathematics*, vol. 58, 1936, pp. 345-363.

14 Of course, on the assumption that Turing's thesis is true.

15 The halting problem for Turing machines is the problem to determine for any given Turing machine whether it will halt or not.

16 Emil Post: Finite Combinatory Processes – Formulation 1. In: *Journal of Symbolic Logic*, vol. 1, nr. 3, 1936, pp. 103-105.

17 Idem, here p. 105.

if true, as a limitation for us humans, and thus, as a natural law. In 1921 he had already proven the unsolvability of certain decision problems on the basis of a thesis similar to Turing's and Church's.[18] During his research preceding these fundamental results, Post developed tag systems.

> **Intermezzo** A tag system T has a deletion number v, a certain number of letters μ and a set of μ words, each corresponding to one of the letters of the alphabet. Tag systems work as follows: you take an initial word A, you tag the word corresponding to the head of A at the end of the word and then delete the first v letters. The result is a new word on which you apply the same operations, resulting in a new word,...An example provided by Post (probably the most famous tag system) is one with $v = 3$, 1 corresponds with 1101, 0 corresponds with 00. Assume the initial word is 110101010100001. We then get:
>
> 110101010100001
> 1010101000011101
> 01010000111011101
> .....

Although tag systems *seem* to be very simple at first sight, they are not. In fact, they can do what any Turing machine can do.[19] It were these systems that convinced Post that there might exist problems that cannot be "computed". After 9 months of research on these systems he had not been able to find a general method to predict the behaviour of these systems. If one confronts oneself with these kind of systems, it becomes indeed clear very soon that some of them are very unpredictable. Many tag systems do not allow you to "calculate" their ultimate behaviour, they resist this kind of control. All you can do is wait and see. In this very sense, the Church-Turing thesis indeed imposes a fundamental limitation on the mathematical powers of man, however, *without* – as is often mistakenly believed – implying some kind of pancomputationalism.[20] It is exactly this human limitation that offers an opportunity for "freer" man-computer-conversations.

Of course, people do not really want their computer to behave unpredictably. It should do as asked and it should behave as expected, even though everybody has experienced more than once that it does not always do so. One then blames the programmer, if it is a software mistake, or the manufacturer of the hardware, if it is a hardware mistake, or, just the age of the computer. One thinks to be in total control over the computer, but this can only be an illusion.

If one would *accept* and *allow* the unpredictability of computations, one might not only come to a better understanding of the computer as the physical realization of computations, but one might also have more interesting "conversations" with it. If I would be able to predict almost every reaction of every person I would be talking to, life would become quite boring. Luckily this is not the case. The freedom to behave unpredictably in a conversation, the freedom to say whatever comes up in your mind is one of the things that can make conversations interesting. It is also the way to "get to know" another person. Of course, this freedom will always be a bounded freedom, since one can only expect freedom in a conversation if one allows the freedom of the other person and respects the dynamics of conversation.

So how could one start having more "interesting" conversations with the computer, conversations during which the computer can no longer be regarded as a mere instrument? There are many possible answers here. One possible starting point is to go and explore computations that are run on the computer. In order to do so, one needs a suitable common language. The everyday interfaces are not suitable since the actual computations, and, very often, also the source code that induces these computations, are hidden away as much as possible from the user. A better common language seems to be some programming language. A programming language allows to interact more freely with the computer, and allows exploration of computations.

So how to explore computations and the related unpredictability of computational processes through programming, and in what way does this come closer to the idea of freer man-computer conversations? Speaking from my own experience, computer experiments have been the most

---

18 Post did not try to publish these results back in the twenties. This is why they are less known. An account of Post's research during the period 1920-21 can be found in: Emil Post: Absolutely Unsolvable Problems and Relatively Undecidable Propositions – Account of an Anticipation. In: Martin Davis (ed.): *The Undecidable*. NewYork (Raven Press) 1965, pp. 340-433.

19 Tag systems were proven to be undecidable by Marvin Minsky in 1961.

20 The idea that everything "computes".

interesting conversations I ever had with a computer. I choose to experiment on tag systems because, amongst other reasons, they are very abstract computational systems and are thus quite "stubborn" toward some kind of "human" or "meaningful" interpretation. Besides, they are very easy to implement and to manipulate with code. Me and my computer spent weeks studying tag systems. The main idea was to trace tag systems that are unpredictable and try to come to a better understanding why they seem to behave unpredictably.

During these experimental processes, I made a lot of mistakes and the computer made a lot of mistakes – mistakes that were not always due to some programming bug. I translated several questions and ideas into the language, the computer "listened" to it and gave a reply, like: "no, I do not think that this tag system has the property you think it has" – of course, expressed in a less "human" language. These computer replies very often led to an adjustment of my own ideas and the consequent translation of this adjusted idea in the language. The reactions I would get from the computer were always unpredictable for me. I never knew in advance what it would reply, and, when it did, I had to decide whether I *trusted* the reply. Questions like: Is the result correct? Did something go wrong during the computational process? Did I make a mistake during translating my ideas in the programming language?...were never far away. I often had to find out through further conversation.

Given the nature of these kind of computer experiments, one cannot claim that the results coming from the experiments are merely the results of the researcher. They are the result of a collaboration, an experimental dialogue – this applies even more for computer proofs.[21]

But is this kind of experimental situation really a conversation? In a certain way, the answer to this question is yes. The process shares many properties with "real" conversations. However, one cannot forget that in the end the computer cannot but execute ("listen") to the code, once the execution command has been given. In this sense, it never has the kind of freedom we have and the conversation becomes an "imbalanced" one.

An ideal situation would be one where both the computer and the programmer are granted even more freedom. There are several techniques to be tried out, which, especially when combined, could lead to more "free" man-computer conversations. Here are some suggestions: the development and use of more ambiguous languages which allow both the computer and the programmer to interpret and use the language in different ways; allowing the computer to behave more unpredictably by e.g. making use of the general unpredictability of computational processes; allowing the computer to make mistakes.

The fact is that each of these three suggested techniques are not out of reach but are actually already part of the way computers work. Most programming languages do allow for certain ambiguities. Also, there exists no program to decide for every possible programming language whether it is ambiguous or not. Furthermore, every programmer knows that he/she is never a 100% sure whether their program will behave as expected, i.e. they cannot always predict whether the computer will understand that which they want it to understand and they cannot always predict what it will reply. Finally, computers do make mistakes. Of course, these mistakes always have a cause. They can be human (e.g. a non-syntactical programming bug), physical (e.g. overheating), due to rounding-off errors or arise because the problem at hand simply cannot be solved 100% correctly (e.g. recovering information from a corrupted file).[22]

Although these possibilities are there, they are not really exploited. On the contrary, techniques are being developed to minimize them and, when possible, avoid them. A computer should not make mistakes, a computer should not be unpredictable, and the translation process from programming language to machine language should be unambiguous and 1-to-1.

---

21 Cfr. Supra, p. 4.

22 Note, that situations where the computer makes a mistake can offer an opportunity for the "user" to become more aware of what is behind the interface. It is known that one can learn a lot about something through its "mistakes" or "malfunctioning". For Turing, the possibility of making mistakes, the computer being fallible, was a necessary condition for intelligent machinery. See Alan Turing: Lecture to the London Mathematical Society on 20 February 1947. In: D.C. Ince (Hg.): *Collected Works of A.M.Turing. Mechanical Intelligence*. Amsterdam (North-Holland) 1992, 87–106.

A kind of conclusion

> HAL: Dave, I don't know how else to put this, but it just happens to be an unalterable fact that I am incapable of being wrong.

How to talk with a computer? One might well think that HAL – the heuristically programmed computer – is the kind of computer one would need for "true" man-computer conversations. The fundamental problem with HAL was that, on the one hand, its creators wanted it to behave humanly, or wanted it to at least give the "illusion" of human behaviour, while, on the other hand, they wanted it to remain a machine following orders. The diagnosis of HAL's behaviour afterwards was that an inconsistency must have occurred, i.e., a conflict between HAL's "truth program" and an instruction to lie about the mission. Its creators had not fully understood the consequences of their programming, they did not understand that HAL, *bound* to follow both its truth program as well as the instruction to lie must lead to problems. The fact that he was programmed/bound to believe that he is infallible, a belief held by his programmers, probably only made things worse.

If one really wants to talk with a computer, having a conversation in which the reciprocal need for mutual understanding is respected, one will have to rethink the very notion of conversation in the context of man-computer conversations, avoiding to impose "human behaviour" on something that never was and will never be human. In order to do so, one will have to try to translate "properties" of man-man conversations into the language of computations. Making mistakes, being unpredictable and having the possibility to interpret something in different ways because of the ambiguous character of the language used, are only some of these properties.

> SIMONSON : Well, I'm afraid Hal was lying. He had been programmed to lie about this one subject for security reasons which we'll explain later. [...] Under orders from earth he was forced to lie. In everything except this he had the usual reinforced truth programming. We believe his truth programming and the instructions to lie, gradually resulted in an incompatible conflict, and faced with this dilemma, he developed, for want of a better description, neurotic symptoms. It's not difficult to suppose that these symptoms would centre on the communication link with Earth, for he may have blamed us for his incompatible programming. [...] If I can speak in human terms, I don't think we can blame him too much. We have ordered him to disobey his conscience.[23]

> In the morning of August 7, at 4.45 a.m. a lightning stroke the local church tower of Moeskroen and the clocks started to chime ceaselessly, waking up many people in the neighbourhood. It seemed that they were computer-controlled and the computer did not react to any command. In the end, they closed down the electricity.[24]

23 Stanley Kubrick and Arthur C. Clarke, ibid.

24 Translated from an on-line news bulletin in De Morgen, dated 7-8-2008.