Strong AI

It’s hard to exaggerate the enthusiasm that workers in the field greeted the idea of a computational theory of the mind. I remember the enthusiasm of those early days as some of the most famous figures in the field, many of them had archived great intellectual distinction in other fields already, were saying such things as Herb Simmons and Allen Newell said, what they called their Physical Symbol System hypothesis, the idea was that we the have now discovered that the mind is as symbol system. It’s a system that operates on symbols, and it doesn’t matter the medium it’s in, it could be in brains, it could be in vacuum tubes, it could literally be in anything that was rich enough and stable enough to sustain the symbol manipulations which were the essential component of thinking. So great was the enthusiasm that Simon and Newell compared themselves to the greatest scientific discoverers of history, perhaps with something less that total modesty, they said the discovery of the physical symbol system hypothesis, was comparable to the germ theory of disease, the DNA theory of genetic inheritance, the atomic theory of matter, and the tectonic theory of geology. They had other examples but those are the ones that spring immediately to mind.

Now, you wonder what is going on? I mean, if somebody’s comparing their rather, I think, doubtful view, to the atomic theory of matter or to the germ theory of disease, or to the greatest scientific discoveries of history, they must think they’ve got some real substance to this. So, before subjecting this to criticism, I want to tell you some of its intellectual foundations. I want to give you some idea, of the intellectual underpinnings of the computational theory of the mind. I have to mention, in passing, that Simon and Newell were not the most immodest, or the most exaggerated, in their enthusiasm. I guess my favorite was Marvin Minsky of M.I.T. who said were are now making computers so smart that we’ll be lucky of their willing to keep us around the house as household pets. I have heard Marvin say that so I know that it’s a correct quote, he was quoted in the New York Times, I find this hard to believe, but anyway, the New York Times quoted, well, maybe it isn’t so hard to believe, he said since we’re programmed into this rather stupid wet-ware that’s gonna decay, and die, and all kinds of terrible things happen to it, what we should do is get our minds reprogrammed onto some decent hardware, where we could have something like immortality; we could go on living as Vacs Computers, or maybe Cray Super-Computers after this dumb wet-ware that we were born with is fully decayed. Anyway, I want to get you the idea that a lot of people excepted this, it was well funded, there were lots of research grants and still are, though they're more doubtful now than they were in the past, and it wasn't just out of the blue. There was a historical development that I tried to explain in the last lecture, but there were also serious intellectual underpinnings, for the idea that we might discover the secret of our own existence, the secret of our own nature, by seeing ourselves as digital computers, and our minds as computer programs, programmed into the wet-ware, or the implementation mechanism, whatever it is, in our brains. There are five ideas, which I think are essential to understanding this. The first idea is the notion of a Turing machine. That's named after Alan Turing, T-U-R-I-N-G the inventor of these ideas. Turing was one of a group of brilliant logicians, and mathematicians, who made a large number, of important discoveries, really as early as the nineteen thirties, but continuing right on into the nineteen fifties. The idea of a Turing machine is an abstract mathematical idea of how problems can be solved by implementing them in a binomial system with only two symbols, a zero and one, and a Turing machine is ludicrously simple in its basic operation.

A Turing machine can perform exactly four operations. It can erase a zero and print a one. It can erase a one and print and zero. It can move its head one step to the left. It can move its head, or its scanner, it has a tape, and it can move it one step to the right. So it can move left or right, it can erase symbols, and it can print symbols. It's a purely abstract mathematical idea, it's a mistake to call it a Turing "machine," it ought to be called a Turing program, or a Turing abstract idea. However, the amazing thing about this abstract idea is you can actually put Turing programs on real machines, on real electronic machines. And though as I said, there aren't any Turing machines in real life, and can't be because, for example they have an Infinite amount of tape, and in real life there aren't any machines that have an infinite tape, and they are purely abstract. Nonetheless, and this is the important thought, for practical purposes the ordinary computer that you by in a store is a Turing machine, it performs these operations, though not in the way I described it, it doesn't have an actual tape, but it performs something that's functionally equivalent to that, by manipulating symbols, with binary symbols. With binary symbols, zeros and ones, you can do an enormous amount of intellectual operations.

Now, what has happened with the development of the technology, is that we can now make Turing machines that manipulate these symbols at a very rapid rate, like several million per second, so you've got millions of these symbolic operations going on, in ordinary household computers. I mean if Turing were alive today, it would take his breath away to see what you can buy for a couple of thousand dollars in any computer store, because it is a level of computing power that far exceeded anything imaginable at the time he developed the idea of a Turing machine. O.K., that's the first notion then idea that there are Turing machines that perform these complex operations such as complex mathematical operations by breaking them down into a series of simple operations that involve only the manipulation of zeros and ones. It's an amazing and powerful idea.

The second idea that we need is the idea of an algorithm, and the idea of an algorithm is the idea of a procedure that will solve a problem by going through a precisely stateable series of steps. A finite series of steps will solve a problem. So for example if you learn to do addition, or long division, there is an algorithm for performing those, you do a precisely stateable series of steps and you take in an input, a bunch of numbers and you get an output, you get a result which is precisely determined by following the algorithm. Now the importance of the algorithm, for our present discussion, that a computer program is algorithm. What a computer program does is give the machine a set of rules for manipulating those symbols. Now let's talk about that for a second.

I said the machine manipulates symbols, but it does that according to a set of rules, and those rules always have the same form. They always say under condition C, perform act A. They're of the form if C then A. So, if you have a symbol zero in this slot, erase that symbol and print a one and move one square to the left, that would be a typical rule that a computer would follow. Now a set of those rules is called a computer program, and computer programs are algorithms in the sense that the computer doesn't worry about ambiguities, or about how to think its way out of complex, muddley ambiguous situations, it does what it's told. It takes in an input and it has a computer program that tells it exactly what to do with that input and it and prints out an output, and that sequence of steps is called a program, or alternatively an algorithm.

Now why is that important? Well there was a very important idea that was also put out at about this time, and it was arrived at independently by a bunch of thinkers, but it's usually called Church's Thesis after the famous mathematical logician Alonzo Church. It was arrived at independently by other people, among them Turing, and I think also Post, but I think it is usually called Church's Thesis, or the Church-Turing Thesis. Now again we're putting all these ideas together, and the idea now is, any algorithm at all, can be run on a Turing machine. Any problem at all, that you can solve, with a series of precisely stateable steps, can be solved on a computer, can be done computationally, can be solved on a Turing machine. In slightly more mathematical jargon, any computable function, that is any mathematical function, where there's a definite solution that you can arrive at by going through a series of steps, that can be programmed onto a computer. A computer can do anything that any algorithm can do, that's a very exciting idea. This is called Church's Thesis because, it's not actually proven as a theorem since the notion of an algorithm is not all that well defined, I just give you an intuitive idea of what an algorithm is, but nonetheless it's a very powerful idea, and it's actually one of the most powerful ideas in the 20th Century, it the idea that this simple device that has only a binary set of symbols, zero and ones, can solve any problem that can be solved algorithmically, any problem at all that can be done in a precisely specifal series, finite number of steps.

Now there's one further idea, this is our fourth idea, and that was also arrived at by Turing, and it's usually called Turing's Theorem, and the idea there is, and Turing actually proved this, as a theorem, that if you've got these Turing machines, which are these devices which manipulate zeros and ones, and implement these programs that are algorithms, then there is a universal Turing machine that can simulate the behavior of any other Turing machine. Now let me explain that idea a little bit. Again, there are abstract mathematical ideas, but the idea is basically very simple.

We understand what a Turing machine is, and we understand what an algorithm is, and we see that that's important because we see that any problem that you can solve algorithmically, anything at all, you can do on a Turing machine, but now it turns out, that there is a universal Turing machine, such that that machine can simulate the behavior of any other Turing machine. Any Turing machine at all, can be exactly duplicated on a universal Turing machine. Now once again, when we use word machine that is misleading, I don't want you to get the idea, I'm gonna go to a shop and buy a universal Turing machine, because these are abstract ideas, it should really be called a program and not a machine. But the importance of it is, that you can implement it, that is to say, you can, the, for practical purposes the computer that you buy is a universal Turing machine, because theoretically at least, I means you wouldn't actually go through the trouble to do this, it can take any program. Any Turing machine program can be run on an ordinary household computer, that's what a general purpose computer is as opposed to specifically dedicated computers.

O.K., so those are four basic theoretical ideas: the idea of a Turing machine, an Algorithm, of Churches' Thesis, and Turing's Theorem. But now in addition to them, there was a fifth notion, and this is not a theoretical idea, but it's the practical proposal, and it's called the Turing test. How much of this is due to Alan Turing it's really quite amazing. Turing proposed the following. If we're going to design programs, to simulate human intelligence, we need a test. A test for when we have succeeded. We need some way of telling, when are you winning in this game, and when are you losing. And Turing proposed the following. If you had, in one room, a human being giving answers to questions, and in another room a machine, giving answers to questions, and you had an expert in the middle passing in questions to each of these two rooms, and getting back the answers, if the expert couldn't tell the difference between the machine answer, and the human answer, then you'd have to say that the machine was as intelligent as the human. If the machine, can behave in a way that the expert can't distinguish from human behavior then the machine has human level intelligence. So, for example, if you were making a program to simulate the behavior of a Chinese speaker, and you got a machine that could pass the Turing test for the understanding of Chinese, you would have to say that the machine understands Chinese. The idea is, you'd have a Chinese expert giving questions to a Chinese person in one room, that he can't see, and to a machine in another room that he can't see, and if the expert gets back these answers, and the machines' answers are as good, as the human answers, then you'd have to say the machine now literally understands Chinese, it doesn't just behave as it understood Chinese, or simulate the behavior of understanding Chinese, it literally understands, because that's what the Turing test tests, it tests whether or not something literally understands.

Now those five ideas, the idea of an implemented algorithm or program, running on a Turing machine, implemented in a Turing machine, a machine that is such that it can implement any algorithm, and the idea of a universal Turing machine that can simulate the behavior of any other Turing machine, and the idea of a test, an objective valid test that will tell the difference between success and failure, between real understanding and real understanding, the idea of a scientific way of finding out when we have created machine intelligence, all of those led to an idea the very statement of which still sends shivers up and down my spine, it's this.

Maybe the Brain is a Universal Turing machine?

That is, maybe what we've got, in the human brain, inside our skulls, is we got a Turing machine. Not an abstract one cause, of course, they're irrelevant considerations, like that we're all gonna die, and we have finite memories; we can't have an infinite amount of tape. But if you subtract this kind of irrelevant stuff, like the fact that it's made of wet stuff, and we go to sleep at night, and get drunk, and otherwise muck up the program, if you subtract that, it looks like—at last, we've made it, we've discovered the answers to our questions; the brain is a digital computer, in fact it is a specific kind of computer, it's the kind that can implement a Turing program, and that's what a digital computer does, it implements these algorithmic programs, and so now, furthermore, not only have we got a theory that intelligence is a matter of a computer, of implementing computer programs, but we've got an objective test, and the test tells us, that if you can design a program that can pass the Turing test, then you have actually created human level intelligence, you have created a mind, and then this had another feature, which was immensely attractive, and that feature was—there's a research project. Ah, there's a research project that a whole lot of people can do. You have a bunch of people who program the computers so that they can pass the Turing test, but then there's real work for the psychologist to do, because what the psychologist do is try to figure out whether or not they way the machine solves the problem, is the same as the way human beings solve the problem, and that was done by psychologists with such things as reaction time experiments. Where you study the reaction time that the human being has, in recognizing faces, or the reaction time in remembering numbers in sequences, random numbers, whether or not that is parallel to the different processing times, that computer programs take to carry out various cognitive tasks.

O.K., so far so good, but you might say that there is something fishy about this because Turing machines can be run in anything. We happen to run ours now in silicon chips, but there's no essential connection between computers and silicon chips. It just happens that that's the best technology, at present, no doubt it will be superceded, prior to that we had transistors, prior to that we had vacuum tubes. But you can also use an abacus, you can have a bunch of men sitting on high stools, wearing green eyeshades, calculating, ah adding up columns of figures, all of those systems are Turing machines, because all of them are implementing programs, they're all carrying out algorithms. Now this is another exciting idea-it doesn't matter what the physical system is, whether it's a brain, or a bunch of a transistors, one writer says, look; you can implement a Turing machine program with a bunch of pigeons pecking. Another author Ned Block said that you can have a computer program that is entirely on, that uses mice, cats, and cheese. When I figure, I forget how it goes, but the mice is released and that's a zero, and then the cat jumps on the mouse and that's a one, it doesn't matter anything goes, provided only that it has this level of symbolic description. Now this phenomena by which anything stateable enough and rich enough to carry the program, is a computer, that's got a name, a very important name, we're gonna see more about it later is called multiple-realizability. The computer program is realizable in a large number of different forms. In fact, it's indefinitely realizable, provided that any realization as I said before, has to be stable enough to carry the program, and rich enough to carry the program. But anything that can carry a program is a computer, it doesn't matter how, what it's made of, or how fast it works, it's just can it implement the program, can it carry out the steps in the program.

O.K., that's the basic idea and I've done my best to try and make it sound appealing, and if I didn't make that sound at all appealing, then I haven't done my job, as an intellectual of the 20the Century, because those ideas are among the most powerful and exciting, of the last 20th Century. And now I'm gonna refute it, I think the whole idea is a massive error, and it's, the beautiful thing in this field, the Computational Theory of the Mind, Strong A.I. has two wonderful features that almost none of those other theories that I mentioned, has.

First of all you can state it simply; the mind is a computer program, the brain is computer hardware. That's just very simple, and we're all familiar enough with computers that we have some idea of what that means, and furthermore, no only can you state it simply, you can refute it in a few minutes. I mean it takes five minutes to state the refutation and I'm now gonna do it, somebody can time me if you'd like, here's how the refutation goes.

When anybody gives you a theory of the mind, always ask yourself—what would it be like for me? What would it be like if my mind actually works the way these guys say, that all minds work. I mean that how I tried Behaviorism, when somebody says that there's nothing to your mind but your behavior, you know that's false(!), cause you know in your own case when you feel a pain, there's a difference between behaving that you feel a pain and actually feeling a pain—right, it's obvious. I think it's obvious for any theory of the mind, that the first test is always try it on yourself, and there's a deep reason for that, namely, our minds have a first person existence, our mental states only exist as experienced by individual human beings. So any theory of the mind, has gotta pass the ME test, it's gotta get past me and my experiences. Well let's try it out with Strong A.I..

Let's suppose, that somebody designs a program, which will enables a computer to pass the Turing test, in some domain that I don't understand. Then what I image, is that I carry out the steps in that program, now I don't understand Chinese, I can't understand a word of Chinese, I can't even tell Chinese writing from Japanese writing, so if you can understand Chinese pick a different example, but I'd like each person hearing this to think of this in their own case, in his or her own case, how would it be and here's how I think of it for me. Let's suppose I am locked in a room, I can't see outside the room, I am locked in a room and in this room there are a bunch of boxes full of symbols. These are Chinese symbols, now I don't know what these symbols mean because I don't know any Chinese. I don't know a word of Chinese, and let's suppose also, that I have a set of rules written in English for manipulating symbols, and the rules say things like, go into box number one and take out a squiggle-squiggle sign, and then go to box number two and match the squiggle-squiggle sign next to the squaggle-squaggle sign, and you can imagine any Chinese symbols you would like here. Now just to look ahead a bit rules like that have a name, those are called computational rules. They tell you under condition C perform act A, they are the kind of thing Turning machines do, I put it in terms of Chinese symbols instead of zeros and ones, but it all comes out the same in the end.

OK, so there I am I'm locked in a room and I got these boxes of Chinese symbols, and I got a rulebook written in English for shuffling the Chinese symbols. Now imagine also that outside the room, there are a bunch of people who pass me little packets of Chinese symbols, and when I get these little packets I look up what I'm supposed to do, and it says match em next to that symbol, and match em next to that symbol, and eventually I go through all the steps, and it says give em back a symbol that has this shape. OK, so symbols come in, I shuffle em according to the rules, and I give back symbols. That's all that's gonna on, I don't understand a word of this, ahh, you could understand, you can think of me as motivated by the fact that they're paying me well, or I get free beer, or whatever, but I'm there knowing nothing, all I do is shuffle symbols. OK, now let's suppose, unknown to me, the people on the outside of the room, call the rulebook, according to which I shuffle the symbols, they call that the computer program, and those boxes of Chinese symbols, they call that the database. And the little bunches of symbols they give in to me they call questions, and the bunches of symbols I give back to them, by following the program and using the database they call answers to the questions. And lets' suppose also, that I get so good at answering the questions, that after while, my answers are indistinguishable, from a native Chinese speaker.

I pass the Turing test for understanding Chinese.

Has everybody got the picture? I mean they're asking me question, I get meaningless squiggle, it doesn't mean anything to me. Unknown to me the question is, ahh, what's the longest river in China? And I give back an answer that says, I don't know, could be the Yang Sea, could be the Yellow River, one or the other, which for me is probably the correct answer. But I know none of that, I don't know what any of these mean, all I know is I've got Chinese symbols that look meaningless to me and I shuffle them around and give back other Chinese symbols, until eventually, they get so good at writing the program, and I get so good at shuffling the symbols, my answers are indistinguishable from the answers of a native Chinese speaker. I'm passing the Turing test for understanding Chinese. OK, now here's the point, here's the punch line of this little parable.

I don't understand a word of Chinese. There is no way I could ever come to understand Chinese the way we've described this, because I don't know what any of these symbols mean! They're all just meaningless symbols. But now, and this is the moral of that punch line, if I don't understand Chinese on the bases of manipulating symbols, that is to say on the bases of implementing a computer program for understanding Chinese, then neither does any other digital computer understand Chinese, because no digital computer has anything I don't have. That is what a computer is, remember what a Turing machine is. A Turing machine is a device that manipulates symbols. It pays no attention to the meanings of the symbols because it doesn't need to. It operates on the symbols entirely in virtue of their form, entirely in virtue of their shape. So the basic moral of the story, I mean the bottom line of this particular story is that you couldn't possibly give machine any understanding by giving it a computer program, because on the basis of the computer program, for understanding Chinese which I took as an example, you don't understand anything. You understand nothing of the Chinese; the only thing I understood in the parable, were that I had English words in which the program was written, that's incidental, but I don't understand any Chinese on the basis of implementing a Chinese-understanding program. And if I don't do that, neither does any other digital computer, because no computer has anything I don't have, and when I understand English, I can't be doing it solely on the basis of a computer program, because we've just seen that a computer program is inadequate.

Now what's going on in this argument, let's stop and think about it a little bit. This argument occurred to me, oh, fifteen, sixteen years ago, and under odd circumstances, I'll tell you a little bit about em.

In the early days of cognitive science, the Sloan Foundation generously funded cognitive scientist to go around the country and give lectures, since we were financed by the Sloan Foundation, we were called, "Sloan Rangers," an obvious label, and I was gonna be a Sloan Ranger going to Yale, to lecture to the artificial intelligence lab, about their research project and I didn't know anything about Artificial Intelligence, I could barely spell it. I bought a book written by the people there, and I read it on the airplane, and they were designing these story-understanding programs. The idea was that they had these wonderful stories, and here's a typical story.

A guy goes into a restaurant, and orders a hamburger, and they bring him a hamburger and it's burned. And then the guy leaves the restaurant and goes away, and now they program all of that into a computer, and then you ask the computer, did the guy eat the hamburger. And they're very proud of the fact a Yale, that the computer will answer, no-the guy didn't eat the hamburger. You remember I didn't say in the story, I just said the guy left the restaurant and went away. I didn't say whether or not he ate it, but the computer can make an inference, so they think, well that proves the computer understands the story, because it can pass the Turing test, it can make the kind of inferences that humans can make. And it seemed to me at the time, and it still seems to me, that doesn't prove anything, cause I could be doing all that in Chinese, and I don't understand anything, if I don't understand anything, neither does the computer. I behave as if I understood, that's the Turing test, but what this shows is that the Turing test is no good, cause it can't distinguish, real understanding from behaving as if you understood.

Well, I was a little bit depressed when I thought of that, because I thought, well first of all, they must of thought of this already, it would occur to anybody, who's heard about these computer programs, and secondly I thought, I'm supposed to give a whole week of lectures in Yale, and that's only gonna take five minutes, they must surely have an answer to that. Well it turns out that they didn't' have an answer to that, everybody was convinced I was wrong, but no one, no two people agreed on exactly why I was wrong, and little did I know, I'm not sure that it was such a good idea there on United Airlines at thirty-thousand feet thinking about the Chinese Room while I was waiting for them to bring me dinner. In the last fifteen years there must have been published two hundred published attacks on that little argument, The Chinese Room Argument, and I'm gonna talk about some of those attacks later, but let's explore some of the implications of this.

I said, just implementing a computer program, by itself, is not enough for understanding anything, and I showed this by giving an example where I implement a computer program, for understanding Chinese, but don't understand any Chinese. But now, what's the difference, between understanding English, which I do understand, and carrying out a program for Chinese, which I don't understand? Let's go back to the Chinese Room. There I was, locked in the room, and I get these questions in Chinese and I give back answers in Chinese, and I don't understand any of it. Suppose that they also ask me questions in English, they ask me questions in English, they ask me, that's the longest river in the United States? And I say, well, it's the Mississippi, but I like to think of it as the Missouri-Mississippi because it's one long river with two names. Now, from the outside my answers to questions in English look no different from my answers in Chinese. In English and in Chinese I am passing The Turing Test, from the outside looks exactly the same. From the inside there's an obvious difference, well what's the difference?

I don't see any reason why we shouldn't accept the common-sense answer. The common-sense answer is, in English I know what these words mean. That is, I have not just meaningless formal symbols, which is what a computer has, but I actually have an understanding of the formal symbols, I have more than just the symbols, I've got a semantics, I have meaning I can attach to the symbols, that's the difference. All right, that leads to the next question, well—why couldn't we give that to the computer? Why couldn't we program the computer with the meanings, as well as with the symbols?

After all, if I know the meanings in English symbols, why couldn't be program in the computer, not just the symbols, but their meanings as well? But now to answer that, remember with the definition of a Turing machine was. It's a device that manipulates symbols; uninterrupted, formal, symbols. That's not the weakness of a Turing machine, that's not a weakness of a computer, that's the source of its power. It's powerful precisely because it doesn't have to know what any of that stuff means, it just has zeros and ones, or Chinese symbols, or yeses and no's, or X and Ys, any system at all that can be precisely formally defined, any system at all that can undergo a set of state transitions, in the operation of a computer, any system at all of that type, is a computer, and any system that can carry those state transitions, will be implementing a computer program.

So there isn't any way we could program a computer with meanings, in addition to symbols, because they by definition of the computer, what the computer does is implement symbols, and to repeat that's the power of the computer. The power of the computer comes precisely from the fact, that it doesn't have to know what any of this stuff means, it doesn't have to be conscious, all it has to do is to be able to manipulate the symbols to carry out, very rapidly, a series of symbol manipulations.

OK I think that argument is decisive, but I have to tell you my view is not universally shared, and I'm gonna spend some time explaining some of the most powerful objections to it. Now, frankly I am puzzled by the depth of the hostility that this little argument, which is not very ambitious, its rather simple, any beginner can understand the argument, all it says is look, understanding can't be just carrying out a computer program, because I can carry out a computer program for something I don't understand, and I still won't understand it. But, the fact that there was so much hostility to this argument, to this day I get so much anger about it, suggests to me that it's almost like a religion, the idea that the solution to our problems is essentially a computational solution. It almost has the status of religious faith, and people feel deeply challenged, if you say, no there's a fundamental error at the core of this.

Let me tell you some of the answers that were proposed to this. When I first presented this argument, the most common answer I got, I found, very puzzling. But it was this; there you are in the room, and you are manipulating the Chinese symbols, but remember you are not alone, because you admitted you had boxes of symbols. You probably had a scratch paper on which to write down the instructions. You had a rule book, there was a whole room, there was a window to the room in which the questions came in, and there was another slot in which answers went out, and it isn't you who understands in the room, it's the whole system who understands. It isn't the individual who understands, it is this entire computer system. In fact, you, are a rather minor feature of the system, here comes a magic word, you are just The Central Processing Unit. Nobody ever thought a CPU could understand anything. It isn't the CPU, it's the whole computer system, that understands. Now when I first heard this I was in a debate, and I said to the guy, who presented it, I said, you mean the room understands Chinese, and the guy said yes, the room understands Chinese. Well, I admire courage, you know, I mean I think it's terrific that people have the nerve of their preposterous views. But I have to tell you that I don't think that's gonna help us much, to say the room understands Chinese, but if you like that answer then think of it the following, just vary the experiment just a little bit. Instead of me, in the Chinese room, with all my symbols and my rule book, imagine that we get rid of the room, and I memorize the symbols, and memorize the contents of the rule book, and carry out all the operations in my head, and if you'd like I can work in the middle of an open field, so there isn't anything in the room that's isn't in me, of course this is science fiction, we couldn't do it in real life, but then you can't actually program computers to simulate understanding Chinese in real life either, so we—if, as a science fiction example, we can imagine that I work alone in the middle of a field. Now I gave this a name, this reply, the reply that says it's the whole system that understands Chinese, I called it the systems reply, and the answer to The Systems Reply I think is very simple. Let me be the whole system, and if I am the entire system, and I don't understand Chinese in virtue of implementing the computer program, then neither does the system, because there's nothing in the system that isn't in me.

Well, why not, what's going on, what are we lacking? Well to repeat, this is the key point I wanna to keep emphasizing, what is missing is any content. The formal symbols by themselves have not content, no meaning, they are, to use the linguist jargon here, purely syntactical, that is we're talking about formal symbols, and formal symbol manipulations. Now if we keep that in mind, that it's the formal structure of the symbols, which is, both the power and the limitation of the computer; it 's the power of the computer because you can implement the same program, in an indefinite range of hard wares, and one and the same hardware can implement an indefinite range of programs. Then we see that the underlying insight behind the Chinese room, was one that doesn't emerge, obviously, from the example itself, but it this. The formal syntax of the computer program, the structure of the program as a purely formal set of symbol manipulations, can't be itself be sufficient for the essential feature of the human mind, namely our thoughts have contents. I don't just think in meaningless symbols, I think in English words. I think in images, and pictures, I have a series of mental contents that I associate with the words. So if I think the sequence of words, my income tax will be handed in late this year, those are not just meaningless symbols going through my mind. On the contrary, they have a very deep meaning for me, they send a shudder down my spine every time I think of those, ah, but not because I got a bunch of zeros and ones, but because I have English words that I know, and the computer's power, derives precisely from the fact, that it knows nothing of meanings, or anything else. The power of the digital computer is precisely what makes it, inappropriate, as a model for the human mind, and in one sentence the computer program has only empty symbols, the human mind has contents. It's only, the only interest that the human mind in con, in symbols, is that it can attach contents to them, and that's true of computer symbols as well.

Well I would like to say that's the end of the story about The Computational Theory of the Mind. But unfortunately, that is the entering wedge into a whole series of philosophical issues that The Chinese Room raises, and we will consider those in the next lecture.