3

Semantic Engines: An Introduction to Mind Design

John Haugeland

I Cognitive Science

"Reasoning is but reckoning," said Hobbes (1651, ch. V), in the earliest expression of the computational view of thought. Three centuries later, with the development of electronic "computers," his idea finally began to catch on; and now, in three decades, it has become the single most important theoretical hypothesis in psychology (and several allied disciplines), and also the basis of an exciting new research field, called "artificial intelligence." Recently, the expression *cognitive science* has been introduced to cover all these varied enterprises, in recognition of their common conceptual foundation. This term, therefore, does not apply to every scientific theory of cognition, but only to those sharing a certain broad outlook – which is sometimes called the "information processing" or "symbol manipulation" approach. Perhaps, at last, Hobbes's philosophical insight has found its home in a proper scientific paradigm (Kuhn, 1970). [...]

Often, discussion of cognitive science focuses on *artificial intelligence* – "AI," among friends – because it amounts to a kind of distilled essence of cognitive science. But again, it is important to realize that "AI" (like "cognitive science") is more specific in its meaning than the words themselves might suggest. Crudely, we can put the point in terms of different technologies: a project at IBM to wire and program an intelligent robot would probably be AI, whereas a project at DuPont to brew and mold a synthetic-organic android probably would not. But this can be misleading; the crucial issue is not protoplasm versus semiconductor ("wetware" versus "hardware"), but rather whether the product is designed and specified in terms of a computational structure. If it is, then a working model could probably be manufactured much more easily by means of electronics and programming; and that's the *only* relevance of the technology. Indeed, the guiding inspiration of cognitive science is that, at a suitable level of abstraction, a theory of "natural" intelligence should have the same basic form as the theories that explain sophisticated computer systems. It is this idea which makes *artificial* intelligence seem not only possible, but also a central and pure form of *psychological* research.

A better perspective on all the excitement can be gained by asking why it took three hundred years for Hobbes's original proposal to be appreciated. Mainly, three famous philosophical dilemmas stood in the way: (i) the metaphysical problem of mind interacting with matter; (ii) the theoretical problem of explaining the relevance of meanings, without appealing to a question-begging homunculus; and (iii) the methodological issue over the empirical testability (and, hence, respectability) of "mentalistic" explanations. The computational idea can be seen as slicing through all three dilemmas at a stroke; and this is what gives it, I think, the bulk of its tremendous gut-level appeal.

Haugeland, J., "Semantic engines: An introduction to mind design," from J. Haugeland (ed.), *Mind Design* (MIT Press, Cambridge, MA, 1981).

Descartes, a contemporary of Hobbes, gave the mind/matter problem its modern form in his doctrine of metaphysical *dualism*. Mind and body, he said, are two entirely different *kinds* of substance: the one can have (as distinguishing characteristics) various thoughts and feelings, whereas the other can have shapes, motions, and the causal interactions described by physical laws (and not vice versa). Intuitively, this is much more appealing than *materialism* (the main alternative to dualism), according to which everything, including minds, is really just matter, in óne form or another. Not only are we reluctant to ascribe thought and feeling to "mere" matter, but we also find it very hard to ascribe shape and location to minds or ideas. There is, however, one basic problem, which no dualist has ever really solved: how can mind and body *interact?* On the one hand, they certainly *seem* to interact, as when a mental decision leads to a physical action, or when a physical stimulus leads to a mental perception; indeed, it's not clear how perception and action could be possible at all without mind/body interaction. On the other hand, however, physical laws are supposed to describe all motions of all bodies *completely* in terms of their interactions with one another.[1] In other words, physics leaves no room for causal intervention by the mental; hence the price of mind/body interaction is violation of the laws of physics – a price that few philosophers (or scientists) are willing to pay.

Thought itself (quite apart from matter) is not static and not random: it progresses and develops in ways that obey (at least much of the time) various rules of inference and reason. Superficially, this suggests an analogy with material particles obeying the laws of physics. But the analogy breaks down at a crucial point: particles have neither choice nor difficulty in "obeying" physics – it happens infallibly and automatically. People, on the other hand, often have to work to be reasonable; following the rules of reason is hardly infallible and can be very difficult. But this means there cannot be an explanatory dynamics of thought, which is at all comparable to physical dynamic theories; the respective roles of rules and laws in the two cases are deeply different. In particular, since correct application of the rules of reason to particular thoughts depends on what those thoughts *mean*, it seems that there must be some active rule-applier, which (or: who) *understands* the thoughts (and the rules), and which applies the rules to the thoughts as well as it can. If the activity of this rule-applier, following the rules of reason, is to explain the rationality of our thought processes, then it must be regarded as a complete little person – or *homunculus* (in Latin) – inside the head, directing the thoughts like a traffic cop. The trouble is: a theory that invokes an homunculus to *explain* thinking has begged its own question, because the homunculus itself has to think, and *that* thinking has *not* been explained.

Finally, there is the question of how the psychology of thought could ever be properly scientific. Thoughts, it seems, cannot be observed; and the difficulty is not that, like electrons or distant galaxies, they are too small or too far away. Rather, they are somehow essentially subjective – we don't even know what it would be like to observe (or measure) them objectively. So all that science has to go on, even in principle, is objective *behavior* (which might include internal "physiological behavior"); hence, thoughts can enter the picture only as inferred or hypothesized intermediates. Unfortunately, in any given case, invoking beliefs and desires to explain an action is just too easy. Whatever an organism does, one can always trump up a million different ideas and motives that would explain it. If there can be, in principle, no independent, empirical check on which of these hypotheses is the right one, it seems scientifically disreputable to accept any of them, even tentatively. This roughly, is the stance taken by *behaviorism*. The other half of the story, however, is that explaining behavior *without* invoking intervening mental processes is just too hard. In general, the subtle regularities, connection, and nonrandom variations in real-life behavior cannot so much as be *described* in nonmentalist (pure behaviorist) terms, let alone *explained* – hence, the overwhelming mechanicalness and stupidity of all the phenomena that the behaviorists were ever really able to account for.[2] The upshot is another philosophical standoff, with mentalists and behaviorists gleefully accusing one another of scientific bankruptcy.

Cognitive scientists can be materialists (nondualists) and mentalists (nonbehaviorists) at the same time; and they can offer explanations in terms of meaning and rule-following, without presupposing any unexplained homunculus. It all depends on a marvellously rich analogy with computers – the outlines of which we can see with a quick look at everybody's favorite example: a chess-playing machine. It would be very awkward and peculiar to start assigning geometrical shapes and locations to the internal program routines and operations (decision processes and data structures, say) of such a system; yet we are quite confident it has no immaterial soul. These same decisions clearly cause physical behavior (e.g., in teletypewriters or TV screens), yet no one is worried that the laws of physics are being violated. When the machine plays, it follows rules in at least two senses: it always abides by the rules of the game, and it employs various reasonable rules of thumb to select plausible moves. Though these rules are in no way laws of nature, the machine's behavior is explained (in part) by citing them – and yet, no unexplained "compunculus" is presupposed. Finally, this explanation will necessarily invoke the system's internal reasoning processes; yet it is far from easy to figure out (or design) processes that will consistently lead to the observed (or desired) behavioral responses. Moreover, for any given machine, on any given occasion, there seems to be a determinate right answer about which reasonings it in fact went through.

That may have been a bit swift, but still, *what an inspiration!* If there are no philosophical dilemmas about chess-playing computers, then why should there be any about chess-playing people – or, indeed, about human intelligence in any other form? To put it coldly: why not suppose that people *just are* computers (and send philosophy packing)? Well... nothing very interesting is ever that simple. Various questions come up, among the first of which is: What exactly is being proposed, anyway?

II Formal Systems

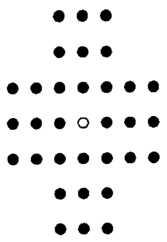
To start at the beginning, we must first say a little bit more carefully what a computer is. It is an automatic formal system. To see what this means, we first consider what a formal system is, and then what it is to automate one.

A *formal system* is like a game in which tokens are manipulated according to rules, in order to see what configurations can be obtained. Basically, to define such a game, three things have to be specified:

1 what the tokens are;
2 what the starting position is; and
3 what moves are allowed in any given position.

Implicit in (2) and (3) is a specification of what positions are possible (for instance, what the board is, if it's a board game). Also, there is sometimes a specified *goal* position, which the player (or each player) is trying to achieve – such as a "winning position."

For example, there is a familiar solitaire game in which the tokens are pegs, arranged as follows in the starting position:



The solid dots are the tokens (pegs), and the circle in the middle is an empty space, into which a token could fit. The only move allowed by the rules is jumping one token over an adjacent one into an empty space, and then removing the token jumped over. This game has a goal: to perform such jumps until only one token remains, and it is in the center space.

Three points should be noticed about this (simple-minded) formal system. First, it is entirely *self-contained.* Only its own tokens, positions, and moves make any difference to it, and these only insofar as they matter to the application of the rules. In other words, the "outside world" (the weather, the state of the economy, whether the building is on fire, and so on) makes no difference whatsoever in the game. And, further, any aspects of the tokens and positions themselves which are irrelevant to determining which moves are legal – e.g., (in this game) color, size, weight, market value – are equally outside the game. Politics and courtship, by contrast, are not at all self-contained (even though they are sometimes called games) because just about anything could be relevant in some situation or other. Second, every relevant feature of the game is *perfectly definite*; that is, barring outright mistakes or breakdowns, there are no ambiguities, approximations, or "judgment calls" in determining what the position is, or whether a certain move is legal. For each peg and slot, that peg is either definitely (obviously and 100 percent) in that slot, or definitely (obviously and 100 percent) not in that slot – there are no in-between or borderline cases. Third, the moves are *finitely checkable*, in the sense that for each position and each candidate move, only a finite number of things has to be checked to see whether that move would be legal in that position. This is pretty trivial for our example, but it's nontrivial and very important for more complicated formal systems. Obviously, being self-contained, perfectly definite, and finitely checkable go nicely hand-in-hand; we will say that a game or system that has all three properties is *digital*. All formal systems are digital in this sense.

The digitalness of formal systems has the following important consequence: two systems that seem to be quite different may nevertheless be essentially the same. Clearly, the peg-jumping game would be essentially unchanged if the pegs were replaced by marbles, or even by helicopters (given a big enough board) – so long as the same rules were followed. But the differences can be more dramatic. Imagine a game played with two baskets and thirty-three dominoes, each with one letter and one numeral written on it. At the beginning, all the dominoes are in the start basket, except the one marked D4, which is in the finish basket; and the object of the game is to reverse that situation, by a process of "switching triads." A *triad* is three dominoes which have the same letter and sequential numerals, or the same numeral and sequential letters – so B4, C4, and D4 form a triad, because they have the same numeral and sequential letters. *Switching* a triad is just moving each of its members to the opposite basket; and this is legal whenever the middle member and one other are in the start basket and the third is in the finish basket. Though one would hardly suspect it at first, it turns out that this game (played with a certain domino set) is essentially the same as the peg-jumping game. It is easy to see why as soon as the members of that domino set are listed in the following revealing order:

A3, A4, A5,
B3, B4, B5,
C1, C2, C3, C4, C5, C6, C7,
D1, D2, D3,   D5, D6, D7,   D4,
E1, E2, E3, E4, E5, E6, E7,
F3, F4, F5,
G3, G4, G5.

Thus switching the D2-D3-D4 triad in the starting position would be equivalent to jumping the peg in slot D2 over the peg in slot D3, thereby emptying both of those slots and filling slot D4.

This kind of essential sameness among formal systems is called *formal equivalence*. Two formal systems are formally equivalent if they can be translated back and forth in roughly the following sense:

(1) for each position in one system, there is a unique corresponding position in the other system;
(2) the two starting positions correspond; and
(3) whenever you can get from one position to another in one system, you can get from the corresponding position to the corresponding position in the other system.

Actually, this definition is a little more stringent than necessary; but it gives the right idea. In particular, it leaves room for equivalent systems to be *very* different on the surface, so long as appropriate correspondences can be found.

There are, of course, an unlimited number of formal systems and most of the interesting ones are significantly more complicated than our peg-jumping example (or its domino-switching equivalent). Two forms of complication are especially widespread and important; we introduce them by considering chess and algebra. The first is that there can be different *types* of tokens, such that what the rules permit depends on the type(s) of token(s) involved. Thus, in chess, each side begins with sixteen tokens of six different types; and whether it would be legal to move a certain token to a certain square always depends (among other things) on what type it is – what would be legal for a rook would not be legal for a bishop, and so on. In fact, whether two tokens are treated equally by the rules (in the same circumstances) is what determines whether they are tokens of the same type. For example, in some fancy chess sets, the pawns are little figurines, each one different from the next; but they are all the same type (namely, pawns), *because* the rules specify the same moves (namely, pawn moves) for all of them. To put it another way, tokens of the same type are formally interchangeable. Note, by the way, that the type of each token has to be perfectly definite (and also finitely checkable, and independent of the outside world) if the overall system is to remain digital.[3]

The second complication is that the *positions* of one formal system can function as the *tokens* in another ("higher level") system. We can see both how this works and why it is important, by considering high-school algebra as a formal system. Though one would not usually call algebra a game, the rules have actually been formalized to the point where it can be played like one. In the algebra game the tokens are equations or formulae, and the rules specify various transformations that can be made in these formulae – or (what comes to the same thing) various new formulae that can be written down, given some that are already written down. The starting position consists of a collection of formulae that are "given" (including, of course, all the *axioms*); and making a "move" is adding a formula (*theorem*) to this list, by following the rules – that is, giving a formal *deduction* or *proof* (or, at least, a step in one). Now, the difficulty is that each different algebraic formula is a different type of token in this system, and there are indefinitely many of them – so how can the moves be finitely checkable? And the answer, of course, is that all these different tokens are built up in a systematic way out of a comparatively small collection of letters and standard symbols.

More specifically, each algebra-game token is a legal position in another game, which we might call the "well-formed-formula game." The tokens of the latter (i.e., the letters and symbols) are usually called "simple" or "atomic" tokens, to distinguish them from the tokens of the algebra game (i.e., the well-formed equations), which are "compound" or "molecular" tokens. The point is that the rules of the algebra game can apply to the various (molecular) tokens in terms of their structure; thus for *any* equation that has the same addend on both sides, you can "cancel" that addend. So, the same rule can apply in a parallel fashion to tokens of indefinitely many different types, so long as they have the specified structural features; hence, finitely many rules can suffice for the algebra game, even though there are indefinitely many types of well-formed algebraic formulae. This general strategy of using the positions of one formal system as the tokens of another makes large complicated systems much easier to deal with, and it is found throughout mathematics, formal logic, and virtually everywhere else that formal methods are used.

It might seem odd to include mathematics and logic here in the same category with chess and the peg-jumping game – because, roughly, their tokens *mean* something (and thus might be true or false, say). But their inclusion is perfectly serious and very important. Most mathematical and logical systems are *formal* in *exactly the same sense* that chess, checkers, and the like are formal: they have starting positions, and rules for making moves in any given position, and they are digital, in the sense explained above. From this point of view, any meanings that their tokens might have are utterly irrelevant; meaning has to do with the outside world, and is in no way part of any self-contained formal system, as such. There are, of course, other points of view, in which meaning is very important (we will discuss some of these when we come to interpretation and semantics). But, considered only *as formal systems*, games, logic, and mathematics are all equally meaningless, and entirely on a par.

III Automatic Formal Systems (Turing Machines and Computers)

An *automatic* formal system is a physical device (such as a machine) which automatically manipulates the tokens of some formal system according to the rules of that system. It is like a chess set that sits there and plays chess *by itself*, without any intervention from the players, or an axiomatic system that writes out its own proofs and theorems, without any help from the mathematician. The exciting and astonishing fact is that such systems can be built. Looked at in the right way, this is exactly what computers are. There are two fundamental problems in building an automatic formal system. The first is getting the device to obey the rules (in principle, this problem has already been solved, as will be explained in the remainder of this section). The second is the "control" problem – how the device selects which move to make when there are several legal options. We will consider this briefly in the next section.

The theoretical ancestor of all automatic formal systems is a class of devices invented (in the abstract) by the mathematician Alan Turing and now called *Turing machines*. A Turing machine has:

1 an unlimited number of *storage bins*;
2 a finite number of *execution units*; and
3 one *indicator unit*.

The indicator unit always indicates one execution unit (the "active" unit), and two storage bins (the "in" and "out" bins, respectively). Each storage bin can contain one formal token (any token, but only one at a time). Each execution unit has its own particular rule, which it obeys whenever it is the active unit. What that rule specifies will depend on what token is in the current in-bin; and in each case it will specify two things: first, what token to put in the current out-bin (discarding the previous contents, if any), and second, what the indicator unit should indicate next. The machine proceeds by steps: the active execution unit checks the in-bin, and then, according to what it finds there and what its rule is, it refills the out-bin and resets the indicator unit; then the next step begins. Usually there is one execution unit which does nothing; so if it ever gets activated, the machine stops.[4]

Clearly, any Turing machine is an automated version of some formal system or other. The starting position is the initial contents of the storage bins, the moves are the machine steps, and the rules are those which the execution units obey (the control problem is handled in the rules for resetting the indicator unit). Not so obvious, but almost certainly true, is the converse: any automatic formal system can be formally imitated by some Turing machine.[5] "Formal imitation" is like formal equivalence, except for two things. First, since we are talking about *automatic* systems – that is, systems that actually "choose" which move to make – an imitating system has not merely to offer corresponding legal options in each corresponding position, but also to make the corresponding choice in each case. Let's call systems which are formally equivalent, and which make equivalent choices in each position, *dynamically* equivalent. Second, the imitating system is often divided into two parts: the part which directly corresponds to the system being imitated, and another part which works behind the scenes, making everything come out right. The first part is called the *virtual machine*, and the second part the *program*.

For example, suppose some automatic formal system, *A*, is being formally imitated by some Turing machine, *T*. Then there is some virtual machine, *V*, which is both a part of *T*, and dynamically equivalent to *A*. So some portion (say half) of *T*'s storage bins will be allocated to *V*, as *V*'s storage bins; the tokens that appear in these bins will constitute *V*'s positions (which, in turn, correspond to *A*'s positions). The rest of *T*'s storage bins contain the *program* (and perhaps some "scratchpad" workspace). The reason a program is necessary is that in general *V*'s rules will be different from *T*'s rules. The program is a (finite) set of tokens so contrived that when *T* obeys its own rules with respect to *all* of its storage bins, it will, in effect, obey *V*'s rules with respect to the tokens in those bins that have been allocated to *V*. Intuitively, we can think of the program as "translating" *V*'s rules into *T*'s rules, or even as "telling" *T* what *V*'s rules are, so that *T* can follow them in moving the tokens in *V*'s bins. In fact, it's not quite that straightforward, and *T*'s rules have to have a certain versatility in the first place, in order to make such an imitation possible. But the point is: for any formal system, there is a Turing machine that can be programmed to imitate it.

But the fundamental importance of Turing machines rests on yet another truly amazing fact – a theorem first proved by Turing – which has, perhaps more than any other single result, shaped modern computer science. It is that there are special Turing machines, called *universal Turing machines*, which can be programmed to imitate any other Turing machine. In particular, one could imitate a Turing machine that was itself imitating some other automatic formal system – which means that, indirectly, the universal machine is also imitating that other automatic formal system. So, combining Church's thesis and Turing's theorem, a universal Turing machine can (suitably programmed) imitate any automatic formal system whatsoever! To put it another way: if you have just one universal Turing machine, and you are prepared to do some programming, you can have (a formal imitation of) any automatic formal system you care to specify.

It was soon discovered that there are a number of different kinds of universal machine, which are not (in the strict sense) Turing machines. But since they are universal, they can formally imitate any Turing machine; and, of course, any universal Turing machine can formally imitate any of them. In principle, therefore, it doesn't matter which one you have – *any* universal machine will do as well as any other (except for differences in efficiency, elegance, and the like). The reason this is important is that, with one qualification, universal machines can be built; that is what digital computers are. The one qualification is that a true universal machine must have unlimited storage, whereas any actual machine will have only a certain fixed amount (though it can be very large). So, aside from limitations of memory size, any standard digital computer can, with appropriate programming, formally imitate any automatic formal system yet discovered; that, basically, is why computers are so powerful.

Interestingly, computer programmers almost never program any of the machines that are actually constructed out of transistors, wires, and so on; rather, most programs are written for *virtual* machines, which are themselves merely being *imitated* by the hardware machine. The reason is that some universal machines are cheaper to build, while others are more convenient to use. The sensible course for a manufacturer, therefore, is to build the cheaper one, hire a few experts to write complicated programs (called "interpreters" or "compilers") which will make it imitate the more convenient ones, and then sell the package. The computer "languages" that you hear about – BASIC, FORTRAN, PASCAL, LISP, and so on – are really just some of these more convenient universal machines, which are widely imitated by various hardware machines. In fact, there are often more layers than that: thus the hardware directly imitates some intermediate virtual machine, which, in turn, is programmed to imitate the higher level machines, like those mentioned above. And, of course, when a programmer programs the latter, he or she is really designing yet another (virtual) machine, which the FORTRAN or LISP machine, or whatever, will formally imitate. This last machine, incidentally, will typically be designed for some special purpose, and so probably will not be a universal machine (though it could be).

A fundamental point should now be obvious: a particular physical object can be, at one and the same time, any number of *different machines*. There is no single correct answer to the question: which machine is that (really)? Of course, at the hardware level it is some particular model from some manufacturer; but at the same time it *is* (just as "really") all the other machines that are being imitated at various other levels. So a particular object is a particular machine (or automatic formal system) only at a particular level of description – at other levels of description it is other machines. Once we see this point, we see how foolish it is to say that computers are nothing but great big number crunchers, or that all they do is shuffle millions of "ones" and "zeros." Some machines are basically numerical calculators or "bit" manipulators, but most of the interesting ones are nothing like that. And the fact that most actual commercial equipment can be described as such machines (e.g., most can be described as bit manipulators, on some level) is of no theoretical consequence. The machine one cares about – perhaps several levels of imitation up from the hardware – may have nothing at all to do with bits or numbers; and that is the only level that matters. For instance, an automatic system that played the peg-jumping game would probably not refer to numbers. Part of the reason for using the expression "automatic formal system" in place of "computer" is that the latter suggests a device which only "computes," and that is just wrong.

IV The Control Problem

In the last section we considered systems in which each move is completely determined by the rules and the current position. This is essential in the design of an *automatic* system, because each step has to be made "automatically" – that is, it must be completely fixed by the state of the machine at the time.[6] But in most positions in most formal systems, any one of various moves would be legal; usually, that is what makes them interesting, as games or whatever. Does it follow that such interesting formal systems cannot be automated? No; it only follows that some device for deciding among the several legal options at any point must be automated as well.

The easiest way to think about it is to consider the machine as divided into two parts or "submachines": one to generate a number of legal options, and another to choose from among them. This, of course, is just an extension of the basic point that a given device can be various machines, depending on how you look at it; only now we are looking at it as *two* separate (virtual) machines, interacting with each other on the same level. The advantage is that we can see how the above dilemma is resolved: the move-generating submachine automates an "interesting" system, in which a variety of moves might be legal at any point; but the combined machine, with both parts together, satisfies the requirement that some *particular* next move be determined by the overall state of the device at each step. Designing the second submachine – the one that makes the choices, given the options – is the *control problem*. In most cases, control design turns out to be the hardest part of automating an interesting formal system.

In an average chess position, for example, a player will have 30 or 35 legal possibilities to choose from. A beginning chess player could discover and list all of them without too much trouble; and designing a machine to do the same would not be terribly difficult either. The hard part – the entire difference between amateurs and world champions – is deciding *which* move to make, given these few possibilities; and that is the hard part to program as well. At first it might seem that big modern computers, with their tremendous speed and memory, could just look ahead to every possible outcome and see which moves lead to ultimate victory. In principle, this would be possible, since chess is technically finite; and such a machine would be literally invincible. In practical terms, however, such a computation is nowhere near possible. Assuming an average of 31.6 options per play gives a thousand (31.6 × 31.6) possible combinations per full move (each side having a turn). Thus looking ahead five moves would involve a quadrillion (10^{15}) possibilities; forty moves (a typical game) would involve 10^{120} possibilities. (For comparison, there have been fewer than 10^{18} seconds since the beginning of the universe.) These numbers are just preposterously large for any physically conceivable computer. They get that big because the number of choices at each additional step *multiplies* the total number of possible combinations so far. For understandable reasons, this is called the *combinatorial explosion*; it plagues control design for all but the most straightforward problems.

Obviously, human chess-players don't make that many calculations either; in fact, the available evidence indicates that they make rather few. The trick is to consider only the *relevant* possibilities and ignore the rest. Thus most of those 30 or 35 options in a typical chess position would be totally pointless or manifestly stupid; hence, it would be a waste of time to consider all the possible developments that could follow after them. If the number of relevant alternatives could be reduced, say, to three at each stage, then looking ahead five complete moves would involve only 59,049 possible combinations – still too many for a person to consider (consciously, anyway), but well within reach of computers. So, the approach to the control problem in this case will almost certainly concentrate on determining which possible moves are the relevant ones, deserving further consideration.

Unfortunately, there is no fail-safe way to tell what is and isn't relevant. Everybody knows how a seemingly pointless or terrible move can turn out to be a brilliant stroke (once the opponent takes the bait). Any method that systematically bypasses certain moves as not worth pursuing will inevitably overlook some of these brilliancies. What we want is a method that is efficient at bypassing moves that *really are* worthless, but not easily fooled into ignoring moves that only *seem* worthless. Such methods in general are called "heuristics" (from the Greek word for "discover"), in contrast to "algorithms" (from the Latin word for the Arabic system of numerals, named after an Arabic mathematician). In current usage, the essential difference is this: an *algorithm* is a rule or procedure that is guaranteed to give a result meeting certain conditions (you just turn the crank and out it pops); a *heuristic*, on the other hand, is a rule or procedure that is more or less reliable, but not infallible – just a rule of thumb. We have algorithms for multiplying integers, arranging words in alphabetical order, and finding a checkmate in certain chess end-games (king and rook against king, for example). But there are no feasible algorithms for finding the best move in most other chess positions, or arranging words into poetry, or discovering proofs of arbitrary theorems in number theory. In these cases people proceed by intuition, inspiration, and a few explicit heuristic rules of thumb. (Intuition and inspiration might just be unconscious heuristics, of course – that's a disputed point among psychologists.)

Automatic systems, in any case, *must* proceed according to explicit rules (explicit, at least, in the program of the virtual machine). And in one sense these rules have to be algorithms – they have to determine the next move of the machine definitively, at each step. But often, whether a given rule is an algorithm or a heuristic depends on how the desired result is specified. Thus a reasonable rule of thumb for winning chess games is: never trade your queen for a pawn. Occasionally, of course, a queen sacrifice is a brilliant play; so this rule is only a heuristic when the result is specified as "Find the best move." But if the specified result is "Avoid swapping a queen for a pawn," then this rule is (trivially) infallible. The point is that the rules followed by the machine only *have to* be algorithms in this trivial sense. That is, it can perfectly well follow a bunch of inconclusive rules of thumb, relative to the interesting result specification (winning, say), so long as there are algorithms defining the heuristics themselves. The machine can infallibly follow quite fallible rules.

This shows, by the way, what's wrong with the idea that a (properly functioning) computer never makes a mistake. It just depends on what counts as a mistake – i.e., relative to which result specification. A chess-playing computer can function perfectly, never making a single mistake in following its internal heuristics but making lots of mistakes *in the game*, because its heuristics are rather clumsy. It is only at special tasks (like arithmetic and sorting), where there are algorithms for the interesting results, that a (perfect) computer can be infallible.

If we construe "heuristics" broadly to include any methods that improve a system's chances of making "correct" decisions while leaving some possibility of "mistakes," then any techniques for providing default assignments (standard assumptions), jumping to conclusions, or reading between the lines will be heuristic procedures. In this broad sense, a major part of the effort in artificial intelligence research goes toward finding better heuristics – ones that are more efficient and harder to fool – and better ways of implementing them on the machines (i.e., better algorithms for defining the heuristics). Indeed, in certain specialized areas, like game playing, theorem proving, and most tasks in "micro-worlds," work on heuristics is most of the problem. These areas are distinguished by the fact that the desired result is already known, and easy to specify in formal terms; for instance, in chess the goal is winning, and winning positions (checkmates) can be defined precisely. Many other forms of intelligent behavior, however, like writing good poetry (to take an extreme case) or carrying on a normal conversation (to take a case that does not *seem* so extreme) are not well defined in this way. In these areas, a major effort is required even to characterize the results toward which the heuristics are supposed to guide the system – and this involves the semantic and pragmatic issues to which we will turn shortly.

V Digital and Analog

Automatic formal systems are, by definition, *digital* computers. There is another kind of device, sometimes called an *analog* computer, which is really quite different. Digital systems, remember, are self-contained, perfectly definite, and finitely checkable – all with regard to which moves are legal in which positions, according to the rules. An analog device, on the other hand, doesn't even have clearly defined moves, rules, and positions – though it may have states (which may change), and there is usually some way that it is supposed to work. The crucial difference is that in analog systems the relevant factors have not been defined and segregated to the point where it is always *perfectly definite* what the current state is, and whether it is doing what it is supposed to do. That is, there will often be slight inaccuracies, and marginal judgment calls, even when the device is working normally. To take the very simplest example, it is like the difference between a multi-position selector switch and a continuous tuning dial on a stereo. A switch has a number of click-stops, and it is always set definitely at one or another of them – you cannot set it between AM and FM. Of course, when you rotate it, it passes through intermediate angles; but these are irrelevant to its function, and (if it is a good switch) can be ignored. A tuning dial, on the other hand, moves smoothly without click-stops, and each angle tunes a different frequency. It is perfectly possible to set the dial between two stations, and, in fact, getting it "right on" a station can require a sensitive judgment call.

For some purposes, analog devices are very convenient. Scale models are a case in point. Suppose an architect wants to find out how the light and shadows will fall on a complicated structure at different times of day. A good analog approach is to build a model, illuminate it with a spotlight from various directions, and then *look* at it from various directions. A trickier but similar case is the use of laboratory animals in medical research. To estimate the physiological effects of weird drug combinations, experimenters can give corresponding (or exaggerated) doses to rats and then just wait to see what happens. Other common examples of analog devices are slide rules, electronic harmonic systems (modeling mechanical harmonic systems), and string-net pathfinders for railroad or highway networks. Though "analog" is itself a rather ill-defined notion, it clearly encompasses quite valuable tools.

Digital systems, however, have several inherent advantages. First, of course, universal machines are, by their very nature, extremely versatile; and that makes them more economical for each application. Second, analog systems can themselves often be *digitally simulated* – which makes the digital system even more versatile. For example, the architect's model could be simulated by writing equations for all the opaque surfaces in the building, and then calculating the paths of individual light rays at, say, one-inch intervals (for various positions of the sun). The whole thing might then drive a TV display, set for various viewing angles; and if all the intervals are small enough, the result can be pretty good. The amount of calculation involved can be prodigious, and the general technique has all the theoretical elegance of sandblasting; but computers are cheap, fast, and tireless, so it often works nicely.

It is sometimes said (with a grand air) that *any* analog device can be digitally simulated to *any* desired precision; but this is grossly misleading. Digital simulation is possible only when all the operative relationships in the analog system can be described in a relatively compact and precise way – e.g., not too many equations with not too many variables. The architectural simulation, for example, depends on assuming that all the light comes from the same direction and travels in a straight line until it's blocked by one of the few specified opaque surfaces. Without this simple structure to work from, the simulation could not get off the ground. Thus there is no comparable general description of the physiology of laboratory rats pumped full of odd chemicals – there are billions of potentially relevant subtle interactions and responses in a complex organic system, and the scope for combinatorial explosion is essentially indescribable. Hence, digital simulation is out of the question, except in very special cases where it is known in advance that only a certain few variables and relationships are relevant. It might seem that, in principle, simulation must be possible anyway, because rats are made of atoms, and each atom obeys known equations. But such a principle is utterly out to lunch. A single large molecule may have so many interdependent degrees of freedom, that no computer yet built could simulate it reliably in reasonable time; and one rat contains more molecules than there would be minicomputers if the entire volume of the earth were packed solid with them.

The real theoretical advantage of digital systems lies in quite another direction, and depends specifically on their digital nature. Consider two conventions for keeping track of money in a poker game. Each uses different colors for different denominations: blue, red, and white for a hundred, ten, and one, respectively. But in one system, the unit of each denomination is a colored plastic disk (i.e., a poker chip), whereas in the other system it is a tablespoon of colored sand. The latter arrangement does have some merits – particularly the fact that *fractional* bets are possible – using less than a full tablespoon of white sand. But the chip system has one overwhelming advantage: all the bets are *exact*. By contrast, in measuring volumes of sand, there is always some small error, no matter how careful you are. Using the chips, and a very modest degree of care, it is possible to wager *exactly* 13 units (plus or minus zero); but with the sand this is impossible, even given the finest instruments in the world. The difference is that the chip system is digital; each token is perfectly definite, and there is no need for judgment calls.

The more complex and interdependent a system becomes, the more vulnerable it is to errors that propagate and get out of control. A small error in one component, affecting a more sensitive component, can result in a larger error going to the next component, and so on. We can see a rudimentary form of this, even in the poker example. Suppose the measurement error on sand volumes is ±2 percent; then it would be perverse to try to bet, say, 613 units, because the measurement error on the six blue tablespoons is worth four times as much (on the average) as all three white tablespoons – the latter simply drop out as irrelevant. There are ways to control such errors, of course, but only to a certain extent, and they tend to get expensive. The perfect definiteness of digital tokens, though confining in some cases, pays off in the elimination of this sort of error; thus, though you cannot bet a fraction of a unit with poker chips, there is no problem in betting 613 units – since six blue chips can be counted exactly, the white chips remain perfectly significant. This advantage is progressively more important for larger and more complicated systems – hence the contemporary predominance of digital methods.

VI Semantics

Formal systems (and computers) can be more than mere games, because their tokens can have interpretations that relate them to the outside world. This is the domain of semantics and pragmatics.

Sometimes we say that the tokens in a certain formal system *mean* something – that is, they are "signs," or "symbols," or "expressions" which "stand for," or "represent," or "say" something. Such relations connect the tokens to the outside world (what they are "about"), making it possible to use them for purposes like record-keeping, communication, calculation, and so on. A regular, systematic specification of what all the tokens of a system mean is called an *interpretation*; and the general theory of interpretations and meanings is called *semantics*. Accordingly, what any token means or says, and hence also whether it is true or false, and so on, are all *semantic properties* of that token.

Semantic properties are not formal properties. A formal system as such is completely self-contained and, viewed in that way, is just a meaningless game. In strictly formal terms, interpretation and meaning are entirely beside the point – they are extraneous "add ons" that are formally irrelevant. (When discussing a system that is to be interpreted, we call its purely formal characteristics and structure its *syntax*; "syntactical" is just another word for "formal," but it is generally used only when a contrast to "semantic" is relevant.)

So, formal tokens can lead two lives: *syntactical* (formal) *lives*, in which they are meaningless markers, moved according to the rules of some self-contained game; and (if the system is interpreted) *semantic lives*, in which they have meanings and significant relations to the outside world. The story of how these two lives get together is the foundation of modern mathematics and logic; and it is also the philosophical inspiration of cognitive science. We review the mathematical/logical case first, however, because it is better understood and there are fewer complications.

Consider a formal system (like the algebra game, discussed above in section 2) in which the positions are just sets of tokens, and the legal moves are just to add more tokens to the current position (depending, of course, on what is already in it). And suppose this system is interpreted in such a way that each of its (well-formed, complete) tokens "asserts" something – that is, each token (according to the meaning assigned to it by the interpretation) expresses some claim about the world. Then, depending on what the claim is and what the world is like, each of these tokens will be either true or false (relative to that interpretation). Now, the *rules* of such a system will have the semantic property of being *truth-preserving* if and only if they meet the following condition: for any position which already contains *only* true tokens, any other token which can be added to that position (according to the rules) will also be true. In other words, if you start with tokens which are all true (under the interpretation), and if you obey the (purely formal) rules, then you can be sure there are no false tokens in any position you ever reach.

The rules of standard logical and mathematical systems, relative to their standard interpretations, are all truth-preserving; and, of course, the tokens in their starting positions (i.e., their axioms) are all true. Therefore, any token in any legal position of one of these systems is guaranteed also to be true! That is why we know in advance that their *theorems* (which are defined in purely syntactical/formal terms) are all *true* (which is a semantic property). Or, what comes to the same thing, in order to establish the semantic truth of a token in such a system, it suffices merely to prove it formally (play the game). This is how the "two lives" of the tokens get together; and it is the basic idea behind the formalization of modern logic and mathematics. In effect, given an interpreted formal system with true axioms and truth-preserving rules, if you take care of the syntax, *the semantics will take care of itself.*[7]

Most mathematical and logical systems are only partially interpreted, in the sense that some of their atomic tokens are left as variables, whose exact interpretation is to be specified by the user (in certain allowable ways). For example, in ordinary algebra, you can specify what the letters (variables) stand for in any way you want – so long as they stand for numbers. The important thing is that the rules are still truth-preserving, no matter what specific interpretation is given to the variables. For example, if a user knows, relative to some specific interpretation of the variables (as values in a physics problem, say) that the token

$$\frac{a-b}{c}+c=\frac{d-b}{c}+\frac{e(1+e)-e}{c}$$

is true, then he or she can apply the purely *formal* rules for multiplying through, deleting parentheses, collecting terms, etc., and be assured that the token

$$a+c^2=d+e^2$$

is also *true* (relative to the same interpretation). This is a vivid example of how useful it can be to have the semantics "take care of itself," if only one plays the syntactical game correctly.

An even more vivid example is computers, because precisely what an automatic formal system can do is "take care of the syntax" – i.e., play by the rules. The machine does not have to pay any attention to the interpretation or meaning of any of its tokens. It just chugs along purely formally; and if the starting position, rules, and interpretation happen to be of the sort we have been discussing, then it will automatically produce only truths. Given an appropriate formal system and interpretation, the semantics takes care of itself. This, fundamentally, is why computers can be so useful for calculations and the like – why they can be *computers* and not just electronic toys.

An automatic formal system with an interpretation such that the semantics will take care of itself is what Daniel Dennett (1981) calls a *semantic engine*. The discovery that semantic engines are possible – that with the right kind of formal system and interpretation, a machine can handle meanings – is the basic inspiration of cognitive science and artificial intelligence. Needless to say, however, mathematics and logic constitute a very narrow and specialized sample of general intelligence. People are both less and much more than automatic truth-preservers. Consequently, our discussion of interpretation needs to be expanded and generalized considerably; unfortunately, the issues will get messier and murkier as we go.

VII Interpretation and Truth

Interpretation is especially straightforward in the special case of logic and mathematics because, in the final analysis, the only semantic property that matters is truth. Hence, it suffices if the theorems are guaranteed true, given the truth of the axioms (and special assumptions, if any) – that is, it suffices if the rules are truth-preserving. We shall see in the next section that there are many other cases where truth is far from all that matters in judging interpretations. But it is worth staying with the special case a little longer, to ask *why* truth has the importance it does; then it will be clearer why other considerations are important in other cases.

Imagine finding an automatic formal system that uses ordinary numerical tokens, and generates "theorems" (i.e., outputs) like the following:

$$\begin{array}{lll} = 5 = 1 \div & = \div 3 - 1 - 8 & + 1 - 5940 \\ 71 \div 92 \times & = 61 = 040 & 84 - 1 \times 5 = \end{array}$$

Formally, of course, there is nothing wrong with these tokens; we can imagine any number of (strange and boring) games in which they would be perfectly legal moves. *Semantically*, on the other hand, they look like nonsensical, random gibberish – "arithmetic salad." That is, it seems impossible to construe them as expressing claims about the relationships among numbers (e.g., equations). But hold on: this reaction depends on a tacit adoption of the familiar Arabic interpretation of what the numerals and signs mean (the digit "1" stands for the number one, the "+" sign stands for addition, and so on). Formally, however, these numerals and signs are just neutral marks (tokens), and many other (unfamiliar) interpretations are possible (as if the outputs were in a code). Suppose, for instance, we construed the atomic tokens according to the following non-customary scheme (using "⇒" to abbreviate "stands for"):

"1" ⇒ equals	"6" ⇒ zero	"+" ⇒ five
"2" ⇒ plus	"7" ⇒ one	"−" ⇒ six
"3" ⇒ minus	"8" ⇒ two	"×" ⇒ seven
"4" ⇒ times	"9" ⇒ three	"÷" ⇒ eight
"5" ⇒ div. by	"0" ⇒ four	"=" ⇒ nine

Then, with this table, we could translate the system's outputs back into the familiar notation as

$$\begin{array}{lll} 9 \div 9 = 8 & 98 - 6 = 62 & 5 = 6 \div 3 \times 4 \\ 1 = 83 + 7 & 90 = 94 \times 40 & 2 \times 6 = 7 \div 9 \end{array}$$

Superficially, these look more like equations, and hence not nearly as random or crazy as the raw (untranslated) outputs. Unfortunately, they're all false – *wildly* false. In fact, on closer inspection, the digits look just as random as before; we still have arithmetic salad, only disguised in regular equation format.

But there are over a trillion possible ways to interpret these fifteen atomic tokens (even sticking to permutations of the ordinary one). Here is just one more possibility, together with the translations it would yield for the original outputs:

"1" ⇒ equals	"6" ⇒ zero	"+" ⇒ five
"2" ⇒ div. by	"7" ⇒ nine	"−" ⇒ four
"3" ⇒ times	"8" ⇒ eight	"×" ⇒ three
"4" ⇒ minus	"9" ⇒ seven	"÷" ⇒ two
"5" ⇒ plus	"0" ⇒ six	"=" ⇒ one

$$\begin{array}{lll} 1 + 1 = 2 & 12 \times 4 = 48 & 5 = 4 + 7 - 6 \\ 9 = 27 \div 3 & 10 = 16 - 6 & 8 - 4 = 3 + 1 \end{array}$$

What a difference! These not only look like equations, they *are* equations – they are *true*. And, intuitively, that strongly inclines us to prefer this interpretation – to think it "better" or "righter" than the ones which yield random nonsense as readings. This intuition has several theoretical considerations behind it.

In the first place, formal tokens in themselves never intrinsically favor one interpretation scheme over any other – from their point of view, all interpretations are equally extraneous and arbitrary. So if some particular interpretation is to be adopted, over all the other possibilities, there must be something distinctive about it, which makes it stand out in the crowd. But, as our example suggests, an interpretation that renders a system's theorems as truths is a rare and special discovery (the alternatives yield mostly gibberish). Hence, such an interpretation is distinctive.

Second, if we regard an interpretation as *relating* or *connecting* a formal system to the outside world (i.e., to whatever its tokens are "about"), then the distinctiveness of the preferred interpretation should lie in that relation. But if a system's theorems are all true under an interpretation, then there is, in fact, a relation strong enough to support inferences about the world from the theorems, and about the theorems from (knowledge of) the world. Thus we can discover new facts about numbers, using an interpreted formalization of arithmetic; likewise, the above-noted (practical) utility of the semantics "taking care of itself" depends on the *truth*-preservingness of the rules. In the other direction, we (sometimes) can predict theorems, given what we know about numbers. For instance, if our example system produced the theorem fragment "7401 . . . ", we would be hard pressed to guess its completion, from its form alone. The first interpretation scheme (which gives "1 × 4 = . . . ") is no help either, because its "translations" come out randomly false. But knowing that the second scheme gives "9 − 6 = . . . ", and that the theorem will be true, makes the prediction easy: the completion will be whichever token means "three" (namely, "×"). The reliability of such inferences (both ways) indicates that *this* relation between the tokens on the one hand and the numbers and operations on the other is *not* arbitrary – i.e., this interpretation somehow genuinely connects them.

Finally, and most important, interpreting is tantamount to "making sense of"; hence, if the system doesn't end by making sense (but, rather, makes nonsense), then the interpretation attempt has failed. Arithmetic salad does not make sense (whether clothed in the outer form of equations, or not); and that, primarily, is why random interpretation schemes don't really give *interpretations* at all. In the context of arithmetic, true equations make eminently good sense – hence the preferability of our second scheme. (An occasional lapse could be understood as meaningful though false; but constant, random falsehood is, in effect, impossible – because the tokens cease to make any sense at all.) So truth matters to interpretations not only because it provides a nonarbitrary choice among candidate schemes and because this choice reflects some relation between the system and what it is (interpreted to be) "about," but also, and most of all, because wild falsehood amounts to *nonsense* – the antithesis of meaning.

VIII Interpretation and Pragmatics

In most activities involving meaningful tokens (other than logic and mathematics) truth is far from the only semantic property that matters. Take ordinary conversation, for example. In the first place, many speech acts – such as questions, commands, expletives, and even most banter and quips – are neither true nor false; so some other kind of appropriateness must be what matters. And even in the case of statements (which typically *are* true or false), much else is important besides just whether they are true. For instance, in conversation it is also important (usually) to stick to the topic, be polite, say only what is worth saying (not obvious, inane, redundant, etc.) – and, in general, to avoid making an ass of oneself. It is not clear to what extent such conditions should be called "semantic" (some authors prefer to call them "pragmatic"), but they are all relevant to the acceptability of an interpretation as soon as you get away from special cases like logic and mathematics.

In these cases we are driven back to the more general but also somewhat fuzzier notion of "making sense" as our criterion for the adequacy of interpretations. There is, to my knowledge, no satisfactory philosophical account of what it is to make sense; indeed, it is questionable whether a precise, explicit definition is even possible. Still, there is much to be said, and philosophers have worked out a variety of rough characterizations and prerequisites (sometimes referred to by the peculiar misnomer "principles of charity"). The truth of simple, uncontroversial declarative outputs (i.e., tokens interpreted as such) is, of course, one rough prerequisite on making sense in general; and a few philosophers (notably, Davidson, 1973) are inclined to stick with that. But most would agree there are other considerations.

The most widely discussed, *rationality*, is loosely analogous to truth-preservation in mathematical systems. The idea is that "obvious consequences" of tokens in the current position should be relatively easy to evoke from the system (as outputs), or to get it to add to the position. In general, the point applies not only to logically valid inferences, but to common sense inferences of all kinds, including the formation of subsidiary goals and attitudes (given the facts, goals, and attitudes already at hand) and also the ability to solve suitably simple "problems." Further, the system should have a tendency to root out tensions and incompatibilities among the tokens in its positions – e.g., by altering or eliminating one of the offending parties. It goes without saying that this notion is not very precisely defined; but it is equally clear that "rationality" in some such sense is an important factor in "making sense."

A second important factor is reliable interaction with the world via transducers. A *transducer* is a kind of automatic encoder or decoder which either reacts to the physical environment and adds tokens to the current position (an *input* transducer), or else reacts to certain tokens in the current position and produces physical behavior in the environment (an *output* transducer). So transducers are analogous to sense organs and motor systems in animals and people; and they become, in effect, part of the whole system, when it comes to determining whether (according to some interpretation) the system is making sense. Roughly, when an input transducer adds a token that is interpreted as a report, then (other things being equal) that report ought to be correct; and when an output transducer reacts to a token interpreted as a decision to do something, then (other things being equal) the behavior produced ought to be that action. For instance, if the token that regularly gets added in response to a rabbit running by were interpreted to mean "Lo, a flaming giraffe," then the system isn't making much sense – which is to say, the interpretation is probably defective. This is essentially Quine's (1960: ch. 2) criterion of preserving "stimulus meaning." Though he doesn't mention it, there is a corresponding condition on the reliability of output transducers.

A third condition on making sense is *conversational cooperativeness*, in more or less the sense introduced by Grice (1975) – assuming, of course, that the system is interpreted as "conversing" or communicating in some way. Suppose, for example, that you ask it what the capital of Illinois is (i.e., give it a token so interpreted); then "It is Springfield" or "I don't know" would be perfectly sensible replies, but "Easy on the mustard, please," or "Wow, 400 squirrels" would not. Even a false but relevant answer, like "It is Chicago," makes much more sense in context than some utterly irrelevant truth, such as "2 + 7 = 9." Answering the question one is asked is a very basic example of being cooperative; there are many subtler points as well, concerning how much information is given in the answer, the manner in which it is provided, how it pertains to the larger topic, whether it is rude, obvious, or funny, and so on. Cooperativeness is clearly a matter of degree, involving judgment and compromise; but if it is ever lacking altogether, a "conversation" quickly reduces to impossible nonsense.

A related consideration is what Austin (1970) called "felicity conditions" (compare also Searle's 1969 "constitutive rules"). Roughly, the idea is that one cannot or cannot "properly" make an unkeepable promise, threaten someone with something he or she wants anyway, give commands for which one lacks the authority, propose marriage to one's spouse, offer to sell (or buy) the planet Mars, and so on. That is, these "speech acts" or "performances" have various prerequisites or presuppositions; and if the prerequisites are not satisfied, there is something wrong with doing them. As with all of our conditions, a few such violations look merely foolish, or perhaps dishonest; but wholesale and flagrant disregard will yield an overall pattern of outputs which fails to make sense – and hence the interpretation will be unsatisfactory.

These last two points, more than the earlier ones, bring in the relevant "context," and indicate the importance of considering the outputs in relation to one another and in relation to the situation in which they are produced. Further work in this direction must confront issues like what is involved in the coherence of extended conversations, dramatic plots, and scholarly essays. Very little in the way of explicit theories or conditions has been proposed in this area – which is symptomatic of an impoverished philosophical literature on the problem of understanding and intelligibility in general. (For a brief overview of some of the difficulties, see Haugeland, 1979, and the works cited there.)

Interpreting an automatic formal system is finding a way of construing its outputs (assigning them meanings) such that they consistently make reasonable sense in the light of the system's prior inputs and other outputs. In the special case of logical and mathematical systems, it suffices if the outputs are consistently true; and this can be guaranteed by having only true axioms and truth-preserving rules. In more ambitious systems, however, including any with aspirations to artificial intelligence, truth is not a sufficient condition on the output tokens "making sense" – many other considerations are important as well. Hence, there is no reason to believe that truth-preservingness is the only, or even the most important, requirement to be imposed on the system's rules. Unfortunately, the alternative – nonasininity-preservation, perhaps? – is not at all clear. And the foregoing motley list of amorphous "further conditions" holds out little promise of any quick or clean solution. I think that this problem constitutes one of the deepest challenges that cognitive science must face.

IX Cognitive Science (Again)

The basic idea of cognitive science is that *intelligent beings are semantic engines* – in other words, automatic formal systems with interpretations under which they consistently make sense. We can now see why this includes psychology and artificial intelligence on a more or less equal footing: people and intelligent computers (if and when there are any) turn out to be merely different manifestations of the same underlying phenomenon. Moreover, with universal hardware, *any* semantic engine can in principle be formally imitated by a computer if only the right program can be found. And that will guarantee *semantic imitation* as well, since (given the appropriate formal behavior) the semantics is "taking care of itself" anyway. Thus we also see why, from this perspective, artificial intelligence can be regarded as psychology in a particularly pure and abstract form. The same fundamental structures are under investigation, but in AI, all the relevant parameters are under direct experimental control (in the programming), without any messy physiology or ethics to get in the way.

Of course, it is possible that this is all wrong. It might be that people just *aren't* semantic engines, or even that no semantic engine (in a robot, say) can be genuinely intelligent. There are two quite different strategies for arguing that cognitive science is basically misconceived. The first, or *hollow shell* strategy has the following form: no matter how well a (mere) semantic engine acts *as if* it understands, etc., it can't *really* understand anything, because it isn't (or hasn't got) "X" (for some "X"). In other words, a robot based on a semantic engine would still be a sham and a fake, no matter how "good" it got. The other, or *poor substitute*, strategy draws the line sooner: it denies that (mere) semantic engines are capable even of acting as if they understood – semantic engine robots are not going to get that good in the first place. The first strategy tends to be more conceptual and *a priori*, while the second depends more on experimental results. (Compare: No beverage made from coal tar would be wine, no matter what it tasted like; with: There's no way to turn coal tar into a beverage that tastes like wine.)

The most obvious candidate for "X" in the hollow shell strategy is *consciousness*; thus "No computer really understands anything, no matter how smart it *seems*, because it isn't conscious." Now it is true that cognitive science sheds virtually no light on the issue of what consciousness is (though see Dennett, 1978, for a valiant effort); indeed, the term itself is almost a dirty word in the technical literature. So it's natural to suspect that something difficult and important is being left out. Unfortunately, nobody else has anything very specific or explanatory to say about consciousness either – it is just mysterious, regardless of your point of view. But that means that a cognitivist can say, "Look, none of us has much of an idea of what consciousness is; so how can we be so sure *either* that genuine understanding is impossible without it, *or* that semantic engines won't ever have it (e.g., when they are big and sophisticated enough)?" Those questions may seem intuitively perverse, but they are very difficult to answer.

A different candidate for "X" is what we might call *original intentionality*.[8] The idea is that a semantic engine's tokens only have meaning because we give it to them; their intentionality, like that of smoke signals and writing, is essentially borrowed, hence *derivative*. To put it bluntly: computers themselves don't mean anything by their tokens (any more than books do) – they only mean what we say they do. Genuine understanding, on the other hand, is intentional "in its own right" and not derivatively from something else. But this raises a question similar to the last one: What does it take to have original intentionality (and how do we know computers can't have it)? If we set aside divine inspiration (and other magical answers), it seems that original intentionality must depend on whether the object has a suitable structure and/or dispositions, relative to the environment. But it is hard to see how these could fail to be suitable (whatever exactly that is) if the object (semantic engine *cum* robot) always *acts* intelligent, self-motivated, responsive to questions and challenges, and so on. A book, for instance, pays no attention to what "it" says – and that's (one reason) why we really do not think it is the *book* which is saying anything (but rather the author). A perfect robot, however, would seem to act on its opinions, defend them from attack, and modify them when confronted with counter-evidence – all of which would suggest that they really are the *robot's own* opinions.

A third candidate for "X" in the hollow shell strategy is *caring*. Here the intuition is that a system could not really *mean* anything unless it had a stake in what it was saying – unless its beliefs and goals mattered to it. Otherwise, it is just mouthing noises, or generating tokens mechanically. The popular picture of computers as cold (metallic, unfeeling) calculators motivates the view that they could never really *care* about anything, hence that they could never genuinely *mean* (or understand) anything on their own. But, of course, the legitimacy of this picture is just what we're inquiring about. If cognitive science is on the right track, then some semantic engines – starting with people – *can* care about things, be involved with them, have personalities, and so on. And, indeed, if we had a robot which *seemed* (in appropriate circumstances) to be sympathetic, disappointed, embarrassed, ambitious, offended, affectionate, and so on, it would be very difficult to claim that it was merely an unfeeling hunk of metal – especially if this very remark "hurt its feelings." Again, we need some further criterion or intuitive test for what it is to have the appropriate inner quality, if we are to justify saying that a semantic engine which merely *acts as if* it is intelligent (etc.) is a hollow shell.

The "poor substitute" strategy – which I, myself, think is much more likely to succeed – argues instead that semantic engines will never even *seem* to have the full range of common sense and values of people. The basic suggestion is that those areas in which computers excel (or can be expected eventually to excel) are all of a special sort, where the relevant considerations are comparatively few and well defined. This includes formal games (by definition) and also a number of other routine technical or micro-world tasks. But it is an open question whether the intelligence manifested in everyday life, not to mention art, invention, and discovery, is of essentially this same sort (though presumably more complicated). Cognitive science, in effect, is betting that it is; but the results are just not in yet. The above issues of consciousness, original intentionality, and caring can all be raised again in the poor substitute strategy, in a more empirical form: Does the system *in fact* act as if it had the relevant "X"? And if, so far, the answer is always "No," is there any pattern to the failures which might give us a clue to the deeper nature of the problems, or the ultimate outcome? These, it seems to me, are the most important questions, but they are beyond the scope of this introduction.

Notes

[1] There remain, of course, quantum indeterminacies, even in physics; but these are no consolation to interactionism. It is generally best, in fact, to forget about quantum mechanics in any discussion of mind/body metaphysics.

[2] Which is not to say, by any means, that those accounts themselves were stupid, or scientifically worthless.

[3] The number of different types of tokens is not a "deep" property of a formal system, as can be seen from the fact that the peg-jumping game has only one type, whereas the domino-switching game has 33 types – yet, in a deep sense, they are the same game. In effect, the domino version trades more sophisticated token discriminations for less sophisticated position discriminations.

[4] This is a slight generalization of Turing's original definition (Turing, 1937). In his version, the storage bins are all connected in a row, called a "tape"; each bin can hold only a *simple* token; the in-bin and the out-bin are always the same bin; and this in/out-bin is always either the same as or right next to the in/out-bin for the previous step. So Turing's machine chugs back and forth along the tape, one bin at a time, dealing with one simple token at a time. The surprising and important point, as we shall see shortly, is that (apart from convenience and efficiency) these differences don't make any difference.

[5] This is a way of expressing *Church's thesis* (named after the logician, Alonzo Church); it has been proven true for all known, well-defined kinds of automatic formal system; but it cannot be proven in general, in part because the general idea of "automatic formal system" is itself somewhat intuitive, and not precisely definable.

6 We ignore the possibility of "randomizers" – they don't affect the point.

7 Unfortunately, even in mathematics, formalization is not all that one might have hoped. Ideally, one would like a system such that not only were all its theorems true, but also all its true tokens were theorems (i.e., *only* theorems were true); such a system is *semantically complete*. But it has been shown (originally by Kurt Gödel, in 1931) that no consistent formalization of arithmetic can be complete; and the same applies to many other important axiomatic systems. Most people are agreed, however, that this result doesn't make any difference to cognitive science. (For a possibly dissenting view, see Lucas, 1961.)

8 "Intentionality" is a philosopher's term for being about something, or having a meaning.

References

Austin, John L. (1970) *How to Do Things with Words*. New York: Oxford University Press.

Davidson, Donald (1973) Radical interpretation. *Dialectica*, 27, 313–28.

Dennett, Daniel (1978) Toward a cognitive theory of consciousness. In C. Wade Savage (ed.), *Perception and Cognition: Issues in the Foundations of Psychology*. Minnesota Studies in the Philosophy of Science, vol. 9. Minneapolis: University of Minnesota Press. Reprinted (1978) in *Brainstorms*. Montgomery, VT: Bradford Books.

Dennett, Daniel (1981) Three kinds of intentional psychology. In R. A. Healey (ed.), *Reduction, Time and Reality: Studies in the Philosophy of the Natural Sciences*. Cambridge: Cambridge University Press.

Gödel, Kurt (1931) Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38, 173–98.

Grice, H. Paul (1975) Logic and conversation. In G. Harman and D. Davidson (eds), *The Logic of Grammar*. Encino, CA: Dickenson.

Haugeland, John (1979) Understanding natural language. *Journal of Philosophy*, 76, 619–32.

Hobbes, Thomas (1651) *The Leviathan*.

Kuhn, Thomas (1970) *The Structure of Scientific Revolutions*, 2nd edn. Chicago: University of Chicago Press.

Lucas, J. R. (1961) Minds, machines and Gödel. *Philosophy*, 36, 120–4.

Quine, W. V. O. (1960) *Word and Object*. Cambridge, MA: MIT Press.

Searle, John R. (1969) *Speech Acts: An Essay in the Philosophy of Language*. Cambridge: Cambridge University Press.

Turing, A. M. (1937) On computable numbers, with an application to the *Entscheidungsproblem*. *Proceedings of the London Mathematical Society*, 42, 230–65.