

This document is subject to the following notice:

NOTICE: This material may be protected by copyright law.

The copyright law of the United States (Title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material; the person using this equipment is liable for any infringement.

e-3070
 Answer: Such an assumption would indeed be chauvinist, but I am not assuming it. The point against the Turing-test conception of intelligence is not that the Aunt Bubbles machine wouldn't process information in the way we do, but rather that the way in which it does process information is unintelligent despite its performance in the Turing test.

Ultimately, the problem with the Turing test for theoretical purposes is that it focuses on performance rather than on competence. Of course, performance is evidence for competence, but the core of our understanding of the mind lies with mental competence, not behavioral performance. The behaviorist cast of mind that leads to the Turing-test conception of intelligence also leads to labeling the sciences of the mind as "the behavioral sciences." But as Chomsky (1959) has pointed out, that is like calling physics the science of meter readings.

11.1.2 Two Kinds of Definitions of Intelligence

We have been talking about an attempt to define intelligence using the resources of the Turing test. However, there is a very different approach to defining intelligence.

To explain this approach, it will be useful to contrast two kinds of definitions of water. One might be better regarded as a definition of the word 'water'. The word might be defined as the colorless, odorless, tasteless liquid that is found in lakes and oceans. In this sense of "definition," the definition of "water" is available to anyone who speaks the language, even someone who knows no science. But one might also define water by saying what water really is—that is, by saying what physicochemical structure in fact makes something pure water. The answer to this question involves its chemical constitution: H_2O . Defining a word is something we can do in our armchair, by consulting our linguistic intuitions about hypothetical cases, or, bypassing this process, by simply stipulating a meaning for a word. Defining (or explicating) the thing is an activity that involves empirical investigation into the nature of something in the world.

What we have been discussing so far is the first kind of definition of intelligence, the definition of the word, not the thing. Turing's definition is not the result of an empirical investigation into the components of intelligence of the sort that led to the definition of water as H_2O . Rather, he hoped to avoid muddy thinking about machine intelligence by stipulating that the word "intelligent" should be used in a certain way, at least with regard to machines. Quite a different way of proceeding is to investigate intelligence itself as physical chemists investigate water. We consider how this might be done in the next section, but first we should recognize a complication.

There are two kinds (at least) of kinds: *structural* kinds such as *water* or *tiger*, and *functional* kinds such as *mousetrap* or *gene*. A structural kind has a

livi012

LIVING STONE
 PFL
 4670
 "hidden compositional essence"; for water, the compositional essence is a matter of its molecules consisting of two hydrogen molecules and one oxygen molecule. Functional kinds, by contrast, have no essence that is a matter of composition. A certain sort of function, a causal role, is the key to being a mousetrap or a carburetor. (The full story is quite complex: Something can be a mousetrap because it is made to be one even if it doesn't fulfill that function very well.) What makes a bit of DNA a gene is its function with respect to mechanisms that can read the information that it encodes and use this information to make a biological product.

Now the property of being intelligent is no doubt a functional kind, but it still makes sense to investigate it experimentally, just as it makes sense to investigate genes experimentally. One topic of investigation is the role of intelligence in problem solving, planning, decision making, and so on. Just what functions are involved in a functional kind is often a difficult and important empirical question. The project of Mendelian genetics has been to investigate the function of genes at a level of description that does not involve their molecular realizations. A second topic of investigation is the nature of the realizations that have the function in us, in humans: DNA in the case of genes. Of course, if there are Martians, their genes may not be composed of DNA. Similarly, we can investigate the functional details and physical basis of human intelligence without attention to the fact that our results will not apply to other mechanisms of other hypothetical intelligences.

11.1.3 Functional Analysis

Both types of projects just mentioned can be pursued via a common methodology, which is sometimes known as *functional analysis*. Think of the human mind as represented by an intelligent being in the head, a "homunculus." Think of this homunculus as composed of smaller and stupider homunculi, and each of these being composed of still smaller and still stupider homunculi until you reach a level of completely mechanical homunculi. (This picture was first articulated in Fodor 1968; see also Dennett 1974 and Cummins 1975.)

Suppose one wants to explain how we understand language. Part of the system will recognize individual words. This word-recognizer might be composed of three components, one of which has the task of fetching each incoming word, one at a time, and passing it to a second component. The second component includes a dictionary, that is, a list of all the words in the vocabulary, together with syntactic and semantic information about each word. This second component compares the target word with words in the vocabulary (perhaps executing many such comparisons simultaneously) until it gets a match. When it finds a match, it sends a signal to a third component whose job it is to retrieve the syntactic and semantic



3 9346 00917635 7

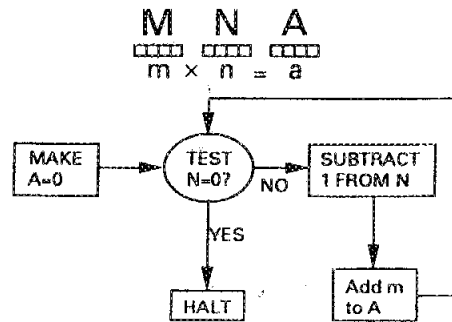


Figure 11.2 Program for multiplying. The user begins the multiplication by putting representations of m and n , the numbers to be multiplied, in registers M and N. At the end of the computation, the answer will be found in register A. See the text for a description of how the program works.

information stored in the dictionary. This speculation about how a model of language understanding works is supposed to illustrate how a cognitive competence can be explained by appeal to simpler cognitive competences, in this case the simple mechanical operations of fetching and matching.

The idea of this kind of explanation of intelligence comes from attention to the way in which computers work. Consider a computer that multiplies m times n by adding m to zero n times. Here is a program for doing this computation. Think of m and n as represented in the registers M and N in figure 11.2. Register A is reserved for the answer, a . First, a representation of 0 is placed in the register A. Second, register N is examined to see if it contains (a representation of) 0. If the answer is yes, the program halts and the correct answer is 0. (If $n = 0$, m times $n = 0$.) If no, N is decremented by 1 (and so register N now contains a representation of $n - 1$), and (a representation of) m is added to the answer register, A. Then the procedure loops back to the second step: register N is checked once again to see if its value is 0; if not, it is again decremented by 1, and again m is added to the answer register. This procedure continues until N finally has the value 0, at which time m will have been added to the answer register exactly n times. At this point, the answer register contains a representation of the answer.

This program multiplies via a "decomposition" of multiplication into other processes, namely addition, subtraction of 1, setting a register to 0, and checking a register for 0. Depending on how these things are themselves done, they may be further decomposable, or they may be the fundamental bottom-level processes, known as *primitive processes*.

The cognitive-science definition or explication of intelligence is analogous to this explication of multiplication. Intelligent capacities are under-

stood via decomposition into a network of less intelligent capacities, ultimately grounded in totally mechanical capacities executed by primitive processors.

The concept of a primitive process is very important; the next section is devoted to it.

11.1.4 Primitive Processors

What makes a processor primitive? One answer is that for primitive processors, the question, "How does the processor work?" is *not a question for cognitive science to answer*. The cognitive scientist answers "How does the multiplier work?" for the multiplier described above by giving the program or the information-flow diagram for the multiplier. But if components of the multiplier, say the gates of which the adder is composed, are primitive, then it is not the cognitive scientist's business to answer the question of how such a gate works. The cognitive scientist can say, "That question belongs in another discipline, electronic circuit theory." Distinguish the question of *how something works* from the question of *what it does*. The question of *what* a primitive processor does is part of cognitive science, but the question of *how* it does it is not.

This idea can be made a bit clearer by looking at how a primitive processor actually works. The example involves a common type of computer adder, simplified so as to add only single digits.

To understand this example, you need to know these simple facts about binary notation:² 0 and 1 are represented alike in binary and normal (decimal) notation, but the binary representation that corresponds to decimal '2' is '10'. Our adder will solve these four problems:

$$0 + 0 = 0$$

$$1 + 0 = 1$$

$$0 + 1 = 1$$

$$1 + 1 = 10$$

The first three equations are true in both binary and decimal, but the last is true only if understood in binary.

The second item of background information is the notion of a gate. An AND gate is a device that accepts two inputs, and emits a single output. If both inputs are '1's, the output is a '1'; otherwise, the output is a '0'. An EXCLUSIVE-OR (either but not both) gate is a "difference detector": it

2. The rightmost digit in binary (as in familiar decimal) is the 1s place. The second digit from the right is the 2s place (corresponding to the 10s place in decimal). Next is the 4s place (that is, 2 squared), just as the corresponding place in decimal is the 10 squared place.

emits a '0' if its inputs are the same (that is, '1'/'1' or '0'/'0'), and it emits a '1' if its inputs are different (that is, '1'/'0' or '0'/'1').

This talk of '1' and '0' is a way of thinking about the "bistable" states of computer representers. These representers are made so that they are always in one or the other of two states, and only momentarily in between. (This is what it is to be bistable.) The states might be a 4-volt and a 7-volt potential. If the two input states of a gate are the same (say 4 volts), and the output is the same as well (4 volts), and if every other combination of inputs yields the 7-volt output, then the gate is an AND gate, and the 4-volt state realizes '1'. (Alternatively, if the 4-volt state is taken to realize '0', the gate is an "inclusive or" (either or both) gate.) A different type of AND gate might be made so that the 7-volt state realized '1'. The point is that '1' is conventionally assigned to *whatever* bistable physical state of an AND gate it is that has the role mentioned, that is, '1' is conventionally assigned to whatever state it is such that two of them as inputs yield another one as output, and nothing else yields that output. And all that counts about an AND gate from a computational point of view is its input-output function, not how it works or whether 4 volts or 7 volts realizes a '1'. Notice the terminology I have been using: one speaks of a physical state (4-volt potential) as "realizing" a computational state (having the value '1'). This distinction between the computational and physical levels of description will be important in what follows, especially in section 11.3.

Here is how the adder works. The two digits to be added are connected both to an AND gate and to an EXCLUSIVE-OR gate, as illustrated in figures 11.3a and 11.3b. Let's look at 11.3a first. The digits to be added are '1' and '0', and they are placed in the input register, which is the top pair of boxes. The EXCLUSIVE-OR gate, which, you recall, is a difference detector, sees different things, and so outputs a '1' to the rightmost box of the answer register, which is the bottom pair of boxes. The AND gate outputs a '0' except when it sees two '1's, and so it outputs a '0'. In this way, the circuit computes $1 + 0 = 1$. For this problem, as for $0 + 1 = 1$ and $0 + 0 = 0$, the EXCLUSIVE-OR gate does all the real work. The role of the AND gate in this circuit is *carrying*, and that is illustrated in figure 11.3b. The digits to be added, '1' and '1', are placed in the top register again. Now, both inputs to the AND gate are '1's, and so the AND gate outputs a '1' to the leftmost box of the answer (bottom) register. The EXCLUSIVE-OR gate puts a '0' in the rightmost box, and so we have the correct answer, '10'.

The borders between scientific disciplines are notoriously fuzzy. No one can say *exactly* where chemistry stops and physics begins. Because the line between the upper levels of processors and the level of primitive processors is the same as the line between cognitive science and one of the

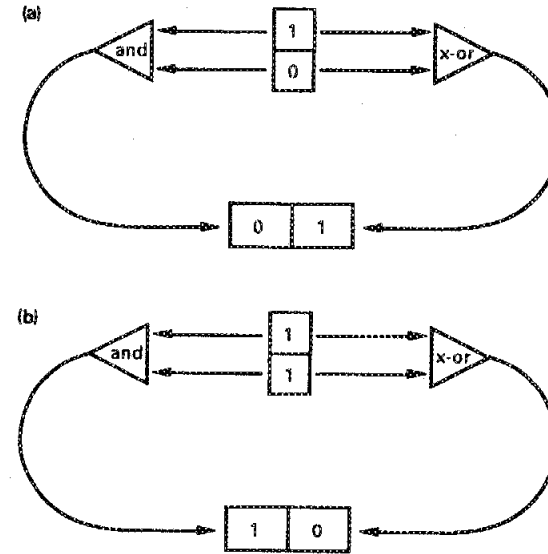


Figure 11.3

(a) Adder doing $1 + 0 = 1$ (b) Adder doing $1 + 1 = 10$.

"realization" sciences such as electronics or physiology, the boundary between the levels of complex processors and the level of primitive processors will have the same fuzziness. Nonetheless, in this example we should expect that the gates are the primitive processors. If they are made in the usual way, they are the largest components whose operation must be explained, not in terms of cognitive science, but rather in terms of electronics or mechanics or some other realization science. Why the qualification, "If they are made in the usual way"? It would be *possible* to make an adder each of whose gates were *whole computers*, with their own multipliers, adders, and normal gates. (We could even make an adder whose gates were people!) It would be silly to waste a whole computer (or a person) on such a simple task as that of an AND gate, but it could be done. In that case, the real level of primitives would not be the gates of the original adder, but rather the (normal) gates of the component computers.

Primitive processors are the only computational devices for which *behaviorism is true*. Two primitive processors (such as gates) count as computationally equivalent if they have the same input-output function, that is, the same actual and potential behavior, even if one works hydraulically and the other electrically. But computational equivalence of *nonprimitive* devices is not to be understood in this way. Consider two multipliers that work via different programs. Both accept inputs and emit outputs only in

decimal notation. One of them converts inputs to binary, does the computation in binary, and then converts back to decimal. The other does the computation directly in decimal. These are not computationally equivalent multipliers despite their identical input-output functions.

If the mind is the software of the brain, then we must take seriously the idea that the functional analysis of human intelligence will bottom out in primitive processors in the brain.

11.1.5 The Mental and the Biological

One type of electrical AND gate consists of two circuits with switches arranged as in figure 11.4. The switches on the left are the inputs. When only one or neither of the input switches is closed, nothing happens, because the circuit on the left is not completed. Only when both switches are closed does the electromagnet go on, and that pulls the switch on the right closed, thereby turning on the circuit on the right. (The circuit on the right is only partially illustrated.) In this example, a switch being closed realizes '1'; it is the bistable state that obtains as an output if and only if two of them are present as an input.

Another AND gate is illustrated in figure 11.5. If neither of the mice on the left is released into the right-hand part of their cages, or if only one of the mice is released, the cat does not strain hard enough to pull the leash. But when both are released, and are therefore visible to the cat, the cat strains enough to lift the third mouse's gate, letting it into the cheesy part of its box. And so we have a situation in which a mouse getting cheese is output if and only if two cases of mice getting cheese are input.

The point illustrated here is the irrelevance of hardware realization to computational description. These gates work in very different ways, but they are nonetheless computationally equivalent. And of course it is possible to think of an indefinite variety of other ways of making a primitive AND gate. How such gates work is no more part of the domain of cognitive science than is the nature of the buildings that hold computer factories. This question reveals a sense in which the computer model of the mind is profoundly *unbiological*. We are beings who *have* a useful and interesting biological level of description, but the computer model of the mind aims for a level of description of the mind that abstracts away from the biological realizations of cognitive structures. As far as the computer model goes, it does not matter whether our gates are realized in gray matter, switches, or cats and mice.

Of course, this is not to say that the computer model is in any way incompatible with a biological approach. Indeed, cooperation between the biological and computational approaches is vital to *discovering* the program of the brain. Suppose one were presented with a computer of alien design

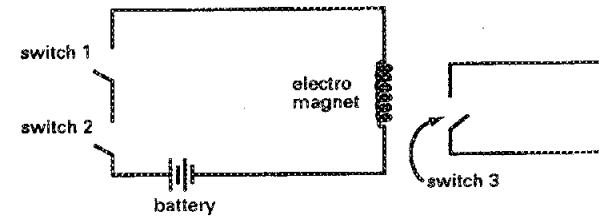


Figure 11.4
Electrical AND gate.
Open = 0, closed = 1

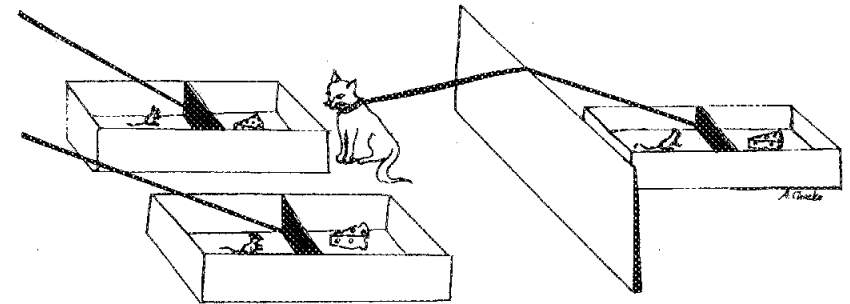


Figure 11.5
Cat and mouse AND gate.
Hungry mouse = 0, mouse fed = 1

and set the problem of ascertaining its program by any means possible. Only a fool would choose to ignore information to be gained by opening the computer up to see how its circuits work. One would want to put information at the program level together with information at the electronic level, and likewise, in finding the program of the human mind, one can expect biological and cognitive approaches to complement each other.

Nonetheless, the computer model of the mind has a built-in antibiological bias, in the following sense. If the computer model is right, we should be able to create intelligent machines in our image—our *computational* image, that is. And the machines we create in our computational image may not be biologically similar to us. If we can create machines in our computational image, we will naturally feel that the most compelling theory of the mind is one that is general enough to apply to both them and us, and this will be a computational theory, not a biological theory. A biological theory of the *human* mind will not apply to these machines, though the biological theory will have a complementary advantage: namely, that such a biological theory will encompass us together with our less

intelligent biological cousins, and thus provide a different kind of insight into the nature of human intelligence. Both approaches can accommodate evolutionary considerations, though for the computational paradigm, evolution is no more relevant to the nature of the mind than the programmer's intentions are to the nature of a computer program.

Some advocates of connectionist computer models of the mind claim that they are advocating a biological approach. But examining typical connectionist models shows that they are firmly within the computationalist paradigm. For example, the popular backpropagation models allow weights of network connections to shift between positive and negative, something that cannot happen with neural connections. It would be a simple matter to constrain the network models so that connections could not change between excitatory and inhibitory, but no one wants to do this because then the models would not work. Thus, although these models are biologically inspired, biological fidelity is rejected when it does not make computational sense.

11.2 Intelligence and Intentionality

Our discussion so far has centered on the computational approach to one aspect of the mind, intelligence. But the mind has a different aspect that we have not yet discussed, one that has a very different relation to computational ideas, namely *intentionality*.

For our purposes, we can take intelligence to be a capacity for various intelligent activities such as solving mathematics problems, deciding whether to go to graduate school, and figuring out how spaghetti is made. (Notice that this analysis of intelligence as a capacity to solve, figure out, decide, and the like, is a mentalistic, not a behaviorist analysis.)

Intentionality is aboutness. Intentional states represent the world as being a certain way. The thought that the moon is full and the perceptual state of seeing that the moon is full are both about the moon and they both represent the moon as being full. Thus both are intentional states. (See volume 2, chapter 9.) We say that the *intentional content* of both the thought and the perceptual state is *that the moon is full*. A single intentional content can have very different behavioral effects, depending on its relation to the person who has the content. For example, the fear that there will be nuclear war might inspire one to work for disarmament, but the belief that there will be nuclear war might influence one to emigrate to Australia. (Don't let the spelling mislead you: intending is only one kind of intentional state. Believing and desiring are others.) Intentionality is an important feature of many mental states, but many philosophers believe it is not "the mark of the mental." There are bodily sensations, the experi-

ence of orgasm, for example, that are genuine mental states but have no intentional content. (Well, maybe there is a bit of intentional content to this experience, for example, locational content, but the phenomenal content of the experience, what it is like to have it, is clearly not exhausted by that intentional content.)

The features of thought just mentioned are closely related to features of language. Thoughts represent, are about things, and can be true or false; and the same is true of *sentences*. The sentence "Bruce Springsteen was born in the USSR" is about Springsteen, represents him as having been born in the Soviet Union, and is false. It would be surprising if the intentional content of thought and of language were independent phenomena, and so it is natural to try to reduce one to the other or to find some common explanation for both. We pursue this idea below, but before we go any further, let's try to get clearer about just what the difference is between intelligence and intentionality.

One way to get a handle on the distinction between intelligence and intentionality is to recognize that in the opinion of many writers on this topic you can have intentionality without intelligence. Thus John McCarthy (creator of the artificial-intelligence language LISP) holds that thermostats have intentional states in virtue of their capacity to represent and control temperature (McCarthy 1980). And there is a school of thought that assigns content to tree rings in virtue of their representing the age of the tree (see the references below in section 11.3). But no school of thought holds that the tree rings are actually intelligent. An intelligent system must have certain intelligent capacities, capacities to *do* certain sorts of things, and tree rings can't do these things. Less controversially, words on a page and images on a television screen have intentionality. For example, my remark earlier in this paragraph to the effect that McCarthy created LISP is about McCarthy. But words on a page have no intelligence. Of course, the intentionality of words on a page is only derived intentionality, not original intentionality. (See Searle 1980 and Haugeland 1980.) Derived intentional content is inherited from the original intentional contents of intentional systems such as you and me. We have a great deal of freedom in giving symbols their derived intentional content. If we want to, we can decide that "McCarthy" will now represent Minsky or Chomsky. Original intentional contents are the intentional contents that the representations of an intentional system have *for* that system. Such intentional contents are not subject to our whim. Words on a page have derived intentionality, but they do not have any kind of intelligence, not even derived intelligence, whatever that would be.

Conversely, there can be intelligence without intentionality. Imagine that an event with negligible (but, and this is important, nonzero) probability occurs: In their random movement, particles from the swamp come

together and by chance result in a molecule-for-molecule duplicate of your brain. The swamp brain is arguably intelligent, because it has many of the same capacities that your brain has. If we were to hook it up to the right inputs and outputs and give it an arithmetic problem, we would get an intelligent response. But there are reasons for denying that it has the intentional states that you have, and indeed, for denying that it has any intentional states at all. For because we have not hooked it up to input devices, it has never had any information from the world. Suppose the swamp brain and your brain go through an identical process, which in your case is the thinking of the thought that Bernini vandalized the Pantheon. The identical process in the swamp-brain has the phenomenal features of that thought, in the sense of "phenomenal content" indicated in the discussion of orgasm above. What it is like for you to think the thought is just what it is like for the swamp-brain. But, unlike you, the swamp-brain has no idea who Bernini was, what the Pantheon is, or what vandalizing is. No information about Bernini has made any kind of contact with the swamp-brain; no signals from the Pantheon have reached it, either. Had it a mouth, it would merely be mouthing words. Thus no one should be happy with the idea that the swamp-brain is thinking the thought that Bernini vandalized the Pantheon.

The upshot: what makes a system intelligent is what it can do, what it has the capacity to do. Thus intelligence is future oriented. What makes a system an intentional system, by contrast, is in part a matter of its causal history; it must have a history that makes its states represent the world, that is, have *aboutness*. Intentionality has a past-oriented requirement. A system can satisfy the future-oriented needs of intelligence while flunking the past-oriented requirement of intentionality. (Philosophers disagree about just *how* future-oriented intentionality is, whether thinking about something requires the ability to "track" it; but there should be little disagreement that there is *some* past-oriented component.)

Now let's see what the difference between intelligence and intentionality has to do with the computer model of the mind. Notice that the method of functional analysis that explains intelligent processes by reducing them to unintelligent mechanical processes *does not explain intentionality*. The parts of an intentional system can be just as intentional as the whole system. (See Fodor 1981.) In particular, the component processors of an intentional system can manipulate symbols that are about just the same things that the symbols manipulated by the whole system are about. Recall that the multiplier of figure 11.2 was explained via decomposition into devices that add, subtract, and the like. The multiplier's states were intentional in that they were about numbers. The states of the adder, subtractor, and so on, are also about numbers and are thus similarly intentional.

There is, however, an important relation between intentionality and functional decomposition that is explained in the next section. As you will see, though the multiplier's and the adder's states are about numbers, the gate's representational states represent *numerals*, and in general the subject matter of representations shifts as we cross the divide from complex processors to primitive processors.

11.2.1 The Brain as a Syntactic Engine Driving a Semantic Engine

To see the idea of the brain as a syntactic engine it is important to see the difference between the number 1 and the symbol (in this case, a numeral or digit) '1'. (The convention in this book is that italics indicate symbols, but in this chapter, numerals are indicated by italics plus single quotation marks.) Certainly, the difference between the city, Boston, and the word 'Boston' is clear enough. The former has bad drivers in it; the latter has no people or cars at all, but does have six letters. No one would confuse a city with a word, but it is less obvious what the difference is between the number 1 and the numeral '1'. The point to keep in mind is that many different symbols, such as 'II' (in Roman numerals), and 'two' (in alphabetical writing) denote the same number, and one symbol, such as '10', can denote different numbers in different counting systems (as '10' denotes one number in binary and another in decimal).

With this distinction in mind, one can see an important difference between the multiplier and the adder discussed earlier. The algorithm used by the multiplier in figure 11.2 is notation independent: *Multiply n by m by adding n to zero m times* works in any notation. And the program described for implementing this algorithm is also notation independent. As we saw in the description of this program in section 11.1.3, the program depends on the properties of the numbers represented, not the representations themselves. By contrast, the internal operation of the adder described in figures 11.3A and 11.3B depends on binary notation, and its description in section 11.1.4 speaks of numerals (notice the quotation marks and italics) rather than numbers. Recall that the adder exploits the fact that an EXCLUSIVE-OR gate detects symbol differences, yielding a '1' when its inputs are different digits, and a '0' when its inputs are the same digits. This gate gives the right answer all by itself so long as no carrying is involved. The trick used by the EXCLUSIVE-OR gate depends on the fact that when you add two digits of the same type ('1' and '1' or '0' and '0') the rightmost digit of the answer is the same. This result is true in binary, but not in other standard notations. For example, it is not true in familiar decimal notation. ($1 + 1 = 2$, but $0 + 0 = 0$.)

The inputs and outputs of both the multiplier and the adder must be seen as referring to numbers. One way to see this fact is to notice that

otherwise one could not see the multiplier as exploiting an algorithm involving multiplying numbers by adding numbers. What are multiplied and added are numbers. But once we go *inside* the adder, we must see the binary states as referring to *symbols themselves*. For as just pointed out, the algorithms are notation dependent. This change of subject matter is even more dramatic in some computational devices, in which there is a level of processing in which the algorithms operate over *parts* of decimal numerals. Consider, for example, a calculator, in which the difference between an '8' and a '3' is a matter of two small segments on the left of the '8' being turned off to make a '3'. In calculators, there is a level at which the algorithms concern these segments.

This fact gives us an interesting additional characterization of primitive processors. Typically, as we functionally decompose a computational system, we reach a point where there is a shift of subject matter from abstractions like numbers or from things in the world to the symbols themselves. The inputs and outputs of the adder and multiplier refer to numbers, but the inputs and outputs of the gates refer to numerals. Typically, this shift occurs when we have reached the level of primitive processors. The operation of the higher-level components such as the multiplier can be explained in terms of a program or algorithm that is manipulating numbers. But the operation of the gates cannot be explained in terms of number manipulation; it must be explained in symbolic terms (or at lower levels, for example in terms of electromagnets). At the most basic computational level, computers are symbol crunchers, and for this reason the computer model of the mind is often described as the symbol-manipulation view of the mind.

Seeing the adder as a syntactic engine driving a semantic engine requires noticing two functions: one maps numbers onto other numbers, and the other maps symbols onto other symbols. The symbol function is concerned with the numerals as symbols—without attention to their meanings. Here is the symbol function:

$$'0', '0' \rightarrow '0'$$

$$'0', '1' \rightarrow '1'$$

$$'1', '0' \rightarrow '1'$$

$$'1', '1' \rightarrow '10'$$

The idea is that we interpret something physical in a machine or its outputs as symbols, and some other physical aspect of the machine as indicating that the symbols are inputs or outputs. Then given that interpretation, the machine's having some symbols as inputs causes it to have other symbols as outputs. For example, having the pair '0', '0' as inputs

causes having '0' as an output. And so the symbol function is a matter of the causal structure of the machine under an interpretation.

This symbol function is mirrored by a function that maps the numbers represented by the numerals on the left onto the numbers represented by the numerals on the right. This function will thus map numbers onto numbers. We can speak of this function that maps numbers onto numbers as the *semantic* function (semantics being the study of meaning), because it is concerned with the meanings of the symbols, not the symbols themselves. (It is important not to confuse the notion of a semantic function in this sense with a function that maps symbols onto what they refer to; the semantic function maps numbers onto numbers, but the function just mentioned, which often goes by the same name, would map symbols onto numbers.) Here is the semantic function (in decimal notation—you must choose *some* notation to express a semantic function):

$$0, 0 \rightarrow 0$$

$$0, 1 \rightarrow 1$$

$$1, 0 \rightarrow 1$$

$$1, 1 \rightarrow 2$$

Notice that the two specifications just given differ in that the first maps quoted entities onto other quoted entities. The second has no quotes. The first function maps symbols onto symbols; the second function maps the numbers referred to by the arguments of the first function onto the numbers referred to by the values of the first function. (A function maps arguments onto values.) The first function is a kind of linguistic "reflection" of the second.

The key idea behind the adder is that of an isomorphism between these two functions. The designer has found a machine that has physical aspects that can be interpreted symbolically, and under that symbolic interpretation, there are symbolic regularities: some symbols in inputs result in other symbols in outputs. These symbolic regularities are isomorphic to rational relations among the semantic values of the symbols of a sort that are useful to us, in this case the relation of addition. It is the *isomorphism between these two functions* that explains how it is that a device that manipulates symbols manages to add numbers.

Now the idea of the brain as a syntactic engine driving a semantic engine is just a generalization of this picture to a wider class of symbolic activities, namely the symbolic activities of human thought. The idea is that we have symbolic structures in our brain, and that nature (evolution and learning) has seen to it that there are correlations between causal interactions among these structures and rational relations among the

meanings of the symbolic structures. A crude example: the way in which we avoid swimming in shark-infested water is that the brain-symbol structure 'shark' engenders the brain-symbol structure 'danger'. (What makes 'danger' mean *danger* is discussed below.)

The primitive mechanical processors "know" only the "syntactic" forms of the symbols they process (for example, what strings of zeroes and ones they see), and not what the symbols mean. Nonetheless, these meaning-blind primitive processors control processes that "make sense"—processes of decision, problem solving, and the like. In short, there is a correlation between the meanings of our internal representations and their forms. And this explains how it is that our syntactic engine can drive our semantic engine.³

In the last paragraph I mentioned a correlation between causal interactions among symbolic structures in our brain and rational relations among the meanings of the symbol structures. This way of speaking can be misleading if it encourages the picture of the neuroscientist opening the brain, just *seeing* the symbols, and then figuring out what they mean. Such a picture inverts the order of discovery, and gives the wrong impression of what makes something a symbol.

The way to discover symbols in the brain is first to map out rational relations among states of mind, and then identify aspects of these states that can be thought of as symbolic in virtue of their functions. Function is what gives a symbol its identity, even the symbols in English orthography, though this relation can be hard to appreciate because these functions have been rigidified by habit and convention. In reading unfamiliar handwriting we may notice an unorthodox symbol, someone's weird way of writing a letter of the alphabet. How do we know which letter of the alphabet it is? By its function! The function of a symbol is something on which we can appreciate by seeing how it appears in sentences containing familiar words whose meanings we can guess. You will have little trouble figuring out, on this basis, what letter in the last sentence was replaced by '%'.

11.2.2 Is a Wall a Computer?

John Searle (1990a) argues against the computationalist thesis that the brain is a computer. He does not say that the thesis is false, but rather that it is trivial, because, he suggests, everything is a computer; indeed, everything is *every* computer. In particular, his wall is a computer computing

3. The idea described here was first articulated to my knowledge in Fodor (1975, 1980); see also Dennett (1981) to which the terms "semantic engine" and "syntactic engine" are due, and Newell (1980). More on this topic can be found in Dennett (1987) by looking up "syntactic engine" and "semantic engine" in the index.

Wordstar. (See also Putnam 1988, for a different argument for a similar conclusion.) The points in the preceding section allow easy understanding of the motivation for this claim and what is wrong with it. In that section we saw that the key to computation is an isomorphism. We arrange things so that, if certain physical states of a machine are understood as symbols, then causal relations among those symbol-states mirror useful rational relations among the meanings of those symbols. The mirroring is an isomorphism. Searle's claim is that this sort of isomorphism is cheap. We can regard two aspects of the wall at time t as the symbols '0' and '1', and then we can regard an aspect of the wall at time $t + 1$ as '1', and so the wall just computed $0 + 1 = 1$. Thus, Searle suggests, everything (or rather everything that is big or complex enough to have enough states) is every computer, and the claim that the brain is a computer has no bite.

The problem with this reasoning is that the isomorphism that makes a syntactic engine drive a semantic engine is more full-bodied than Searle acknowledges. In particular, the isomorphism has to include not just a particular computation that the machine *does perform*, but also all the computations that the machine *could have performed*. The point can be made clearer by a look at figure 11.6, a type of X-OR gate. (See O'Rourke and Shattuck, forthcoming.)

The numerals at the beginnings of arrows represent inputs. The computation of $1 + 0 = 1$ is represented by the path $A \rightarrow C \rightarrow E$. The computation of $0 + 1 = 1$ is represented by the path $A \rightarrow B \rightarrow E$, and so on. Now here is the point. In order for the wall to be this computer, it isn't enough for it to have states that correspond to '0' and '1' followed by a state that corresponds to '1'. It must also be such that *had* the '1' input been replaced by a '0' input, the '1' output *would have been* replaced by the '0' output. In

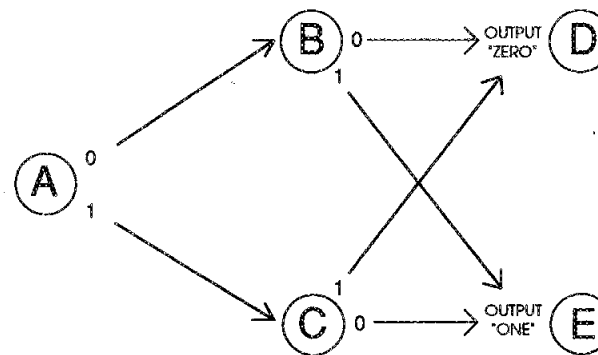


Figure 11.6
The numerals at the beginnings of arrows indicate inputs.