6

The computational theory of mind

the only game in town

—Jerry Fodor

Science fiction is full of computers and robots that can think. HAL, the computer in Stanley Kubric's *2001: A Space Odyssey*, is perhaps the most famous example, but there are lots of others: K9 in *Doctor Who*; Data in *Startrek TNG*; and—my personal favorite—the Terminator in (you guessed it) the *Terminator*. We have become comfortable, it would seem, with the idea that a machine could have thoughts. The **computational theory of mind** takes this idea one step further. According to **computationalism** the mind is, quite literally, a computer.

In this chapter we will explore the idea that the mind is a computer. In order to understand that idea we need to be clear about what a computer is. That's the task of Section 6.2. However, before we can understand what a computer is, we first need to understand the distinction between **syntax** and **semantics**. That's the task of the next section.

6.1 Syntax and semantics

It will be helpful to have a rough-and-ready distinction between *basic symbols* and *complex symbols*. A basic symbol is one which has no meaningful parts; a complex symbol is one which is made up of two or more basic symbols. I will use English words as examples of basic symbols, and I will use English sentences as examples of complex symbols. Thus, 'Fodor' is a basic symbol whereas 'Fodor wrote *The Modularity of Mind*' is a complex symbol. (It might be argued that English words can't be basic symbols because they contain meaningful parts—the letters out of which they are assembled. For present purposes I will ignore this complication.)

The syntactic properties of a symbol are the properties which can be detected simply by examining the symbol in isolation. Here are some examples. Consider the basic symbol 'Fodor'. Just by examining that symbol you can work out that it consists of five letters in a certain arrangement. (Strictly, it consists of five letter

tokens. For the distinction between types and tokens see Section 3.1.) You can also work out that the symbol is black on white and that it is less than three centimeters long. These are all syntactic properties of the symbol 'Fodor'. In contrast, you can't work out that the symbol 'Fodor' refers to a famous American philosopher just by examining it. Nor can you tell the name and address of the typesetter who arranged that symbol on the page. So the properties 'refers to a famous American philosopher' and 'was typeset by Bloggs of 44 Carbuncle St London' are *not* syntactic properties.

It's very common for the syntactic properties of a symbol to be called the symbol's 'shape'. That's not surprising as shape is a good example of a syntactic property. In written English it is customary to end a question with a special symbol—the question mark. That symbol has a characteristic shape—?. We can easily recognize the question mark symbol by its shape. Indeed—and this will come up again later—a *machine* can recognize the question mark symbol by its shape. So shape is a syntactic property. If you're having trouble keeping in mind what the syntactic properties are, just think of them as a symbol's shape. You won't be far wrong.

What about the semantic properties of symbols? Roughly speaking, semantic properties are properties connected with the *meaning* of a symbol. For example, the **reference** of a symbol—what it refers to—is a semantic property. To continue with our example, the basic symbol 'Fodor' refers to a famous American philosopher. The **truth value** of a symbol—whether it is true or false—is also a semantic property. Note, though, that not all symbols have a truth value. The symbol 'Fodor', for example, is neither true nor false. Whilst it is true that Fodor wrote *The Modularity of Mind* and false that Fodor wrote *Alice in Wonderland*, 'Fodor' by itself is neither true nor false.

What kinds of symbols have truth values? Some (but not all) complex symbols have truth values. The complex symbol 'Fodor wrote *The Modularity of Mind*' is true; the complex symbol 'Fodor wrote *Alice in Wonderland*' is false. Both of these complex symbols make a claim about the world. The first example claims that the world is one in which Fodor wrote *The Modularity of Mind*; the second claims that the world is one in which Fodor wrote *Alice in Wonderland*. In general, symbols have the semantic properties of truth or falsity if and only if they make a claim about the world. It is now obvious why the symbol 'Fodor' is neither true nor false. By itself, 'Fodor' makes no claim about the world.

Unlike the syntactic properties, the semantic properties *cannot* be detected by examining a symbol in isolation. Think again about the symbol 'Fodor', and assume you have no idea who (or what) Fodor is. You can stare at that symbol for as long as you like and you will never work out what it refers to. In order to know that the reference of 'Fodor' is a famous American philosopher, you have to look beyond the symbol itself. In particular, you have to work out which person is connected in the right way to that symbol. Similarly, examining the complex

symbol 'Fodor wrote *Alice in Wonderland*' in isolation will not allow you to determine its truth value. You have to look at the world beyond the symbol in order to determine that 'Fodor wrote *Alice in Wonderland*' is false.

In summary, the syntactic properties of a symbol can be detected just by examining the symbol. Shape is a good example of a syntactic property. Semantic properties are connected with meaning and include properties like reference and truth value. Since you cannot determine a symbol's semantic properties by examining it in isolation, semantic properties are not syntactic properties. With the distinction between syntactic and semantic properties in place, we can turn to the question, 'What's a computer?'.

6.2 What's a computer?

In this section I'm going to describe what a computer is. I'm not going to talk about keyboards and monitors; rather, I'm going to give a very general description of computation which abstracts away from a great many of the details of real computers.

Speaking very generally, a computer has two features.

1. Computers recognize and manipulate symbols *solely on the basis of their syntactic properties*. Here's an example. I can use the 'find' function of my word processing program to locate the 'Fodor' symbols in the document I'm presently typing. My computer recognizes those symbols by their syntactic properties—perhaps my computer recognizes 'Fodor' symbols by their shape. (Electronic computers don't usually recognize symbols by their shape, but let's pretend they do.) What's especially important is that computers don't recognize symbols by their semantic properties. That's not surprising. After all, my computer has no idea at all what the symbol 'Fodor' refers to. Perhaps 'Fodor' refers to a cat; perhaps it refers to the seventh moon of the seventh planet of Alpha Centuri. My computer simply does not know. *All* my computer has to go on are the properties of symbols which can be detected in isolation; that is, *all* my computer has to go on are the syntactic properties of symbols. So computers are 'syntactic engines'—devices which recognize and manipulate symbols on the basis of their syntactic properties.
2. Whilst computers recognize and manipulate symbols solely on the basis of their syntactic properties, they can nevertheless be arranged so that the way the symbols are manipulated respects the semantic properties of those symbols. Thus it is easy to program a computer so that it begins with the following two complex symbols:

   A. All philosophers are funky.

   B. Fodor is a philosopher.

And ends up with the following complex symbol:

C. Fodor is funky.

The computer recognizes and manipulates the various symbols involved by their syntactic properties; it *does not* recognize and manipulate those symbols on the basis of their semantic properties. After all, the computer does not know who Fodor is, does not know what a philosopher is, and does not know what it is to be funky. Nevertheless, the transition from A and B to C is truth preserving: if A and B are true then C is too. So, whilst the computer is sensitive only to syntactic properties, it manages to respect semantic properties like truth value.

Notice that the transition the computer makes from A and B to C is rational in this sense: the first two complex symbols *provide evidence for* the last one. This is an example of the way that computers can be programmed to carry out at least some kinds of rational inference. We noted way back in the Introduction that the causal relations between thoughts often mirror the rational relations between those thoughts. Computers provide us with our first hint of how that might be achieved.

To sum up, a computer is a device which recognizes and manipulates symbols on the basis of their syntactic properties, but still manages to respect semantic properties like truth value.

It's worth briefly mentioning that computational states and processes are multiply realizable. That is, devices which are physically quite different can nevertheless be in the same computational states and realize the same computational processes. It happens to be convenient to realize computational states and processes in electronic circuits, but in principle they can be realized by a wide range of devices. (See Weizenbaum 1976: Ch. 2 for an explanation of how to build a computer out of toilet paper and stones!) Even amongst electronic computers there is considerable diversity. Your PC and my Mac can undertake exactly the same computations even though they are, from an engineering point of view, quite different.

In the next section I will briefly describe an incredibly simple computer. The simplicity of the computer helps make it clear that it's a syntactic engine. However, it is also clear that its symbolic manipulations respect certain semantic properties.

6.3 Turing machines

Turing machines are very simple computers which, nevertheless, are extremely powerful. Strictly speaking, there are no Turing machines in the sense that no one could ever build one. However, thinking about the design and capabilities of

Turing machines has been very important in the development of mathematical logic and the theory of computation. (Turing machines were the brainchild of Alan Turing (1912–54), an extraordinary British mathematician who, besides making important contributions to mathematical logic and the theoretical foundations of computer science, was involved in cracking the German navy's 'Enigma' code during the Second World War. For a fascinating account of his life and achievements see Hodges 1983.)

A Turing machine consists of three parts.

1. A very long tape (for example, a very long strip of paper) divided into squares. (Strictly speaking, the tape should be infinitely long; hence, strictly speaking, Turing machines can't be built.) Symbols can be written in the squares. There is a finite 'alphabet' or list of symbols.
2. A 'head' which can do the following four things:
   (i) read the symbol in a square;
   (ii) erase the symbol in a square and replace it with another symbol from the alphabet;
   (iii) move one square either left or right; and
   (iv) halt (that is, cease all activity).

   What the head will do on any one occasion is determined by the symbol it is reading and the 'state' it's in. (More about states in a moment.)
3. A 'machine table' which specifies for each symbol in the alphabet and each state of the head what the head should do. The machine table will specify whether the head should leave the symbol as it is or replace it; whether the head should move left or right; and what state the head should be in next. (Alternatively, the machine table might simply order the head to halt.)

Let's look at a simple example: a Turing machine for adding together pairs of numbers (strictly, for adding together pairs of positive integers). The alphabet consists of two symbols: '—' and 'X'. (The '—' symbol simply indicates that the square is blank.) If we want the machine to add together 2 and 3, we set up the tape as shown in Figure 6.1.

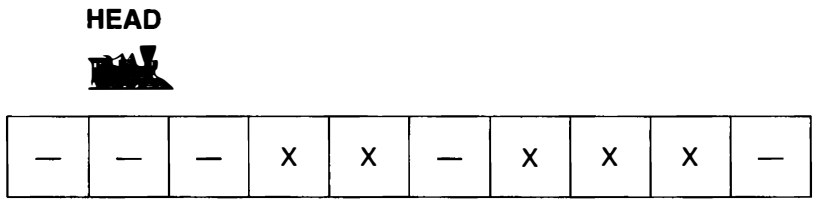

Figure 6.1 The head and tape of a Turing machine about to add 2 and 3

Table 6.1 The Turing machine table for adding two numbers.

	Input: X	**Input: —**
State: 1	no change/R/2	no change/R/1
State: 2	no change/R/2	X/L/3
State: 3	no change/L/3	no change/R/4
State: 4	—/halt	

The left sequence of Xs represents the number 2; the right sequence represents the number 3. The head is initially located to the left of the left sequence of Xs and is in state 1.

The machine table (Table 6.1) instructs the machine to write down the sum of two numbers. You read the machine table as follows. Take for example the top left square. It tells the head what to do when it is in state 1 and is reading a square which contains an X. The instruction in that square is 'no change/R/2', which means 'don't change the symbol, move one square to the right, and move into state 2. The instruction 'X/L/3' means 'replace the present symbol with an X, move one square left, and move into state 3'.

The basic idea behind the machine table is as follows. The head moves right until it reaches the left sequence of Xs. It then keeps going until it reaches the blank (i.e. —) that separates the two sequences of Xs. It replaces the blank with an X and heads left. When it reaches the next blank (i.e. after it has passed all the Xs) it moves one step right and replaces the leftmost X with a blank. The result (in our example) is a series of 5 Xs; that is, the machine has calculated that the sum of 2 and 3 is 5. Obviously, the same process works for any two sequences of Xs separated by a blank; in other words, the machine table will suffice for the addition of any two positive integers. You might like to play around with the machine table and satisfy yourself that it actually works.

So far we have examined in some detail Turing machines and their machine tables. For our purposes, what's significant about Turing machines is that they are 'syntactic engines'; that is, the head of the Turing machine is sensitive only to the syntactic properties of the symbols on the tape. Nevertheless, given the right machine table, a Turing machine will respect the semantic properties of its inputs. Thus, in the case of the Turing machine just described, the machine is quite unaware of what the series of Xs means; nevertheless, it successfully completes the addition.

It seems that anything that can be computed by a more sophisticated computer can be computed by a Turing machine. (This is called 'Turing's thesis'.) Consequently, it seems that any computation whatsoever can be achieved by a machine which is sensitive only to syntax. The recognition and manipulation of symbols solely in virtue of their syntactic properties is thus surprisingly powerful.

6.4 The computational theory of mind

According to the computational theory of mind (hereafter CTM), thoughts are complex symbols which have both syntactic and semantic properties. We have seen that complex *linguistic* symbols like 'Fodor is funky' are made up of basic symbols like 'Fodor'. Similarly, according to CTM the *thought* that Fodor is funky is a complex symbol made up of basic symbols. However, we cannot assume that the mental symbol which refers to Fodor (the philosopher) is the English word 'Fodor'. It may be that the mind has a language or code of its own. We will return to this idea in Section 6.5. For the moment we will just accept that the mental symbol which refers to Fodor is the English word 'Fodor'.

So far we have seen that, according to CTM, thoughts are complex symbols with syntactic and semantic properties. What about thinking? According to CTM, thinking involves the recognition and manipulation of thoughts purely on the basis of their syntactic properties. However, whilst thoughts are manipulated solely on the basis of their syntactic properties, those manipulations respect the semantic properties of the thoughts involved. Thus, say that I believe that

D. All philosophers are funky,

and:

E. Fodor is a philosopher.

These thoughts lead me to believe that

F. Fodor is funky.

According to CTM, the mental processor which achieves the transition from D and E to F is sensitive only to the syntactic properties of the thoughts involved. Nevertheless, the transition is truth preserving; that is, if D and E are true, so is F. Moreover, the transition is rational in the sense that the thoughts D and E provide evidence for the thought F.

Putting all this together we can say that, according to CTM, thoughts are complex symbols with syntactic and semantic properties. Thinking—the manipulation of thoughts—is achieved by processors which, whilst sensitive only to the syntactic properties of the thoughts involved, nevertheless respect their semantic properties. In other words, thinking is computation.

This is, to put it mildly, a beautiful idea. Indeed, it might even be true! We have already seen that CTM offers an account of the rationality of thought; I will now briefly sketch three further virtues of CTM.

1. We have seen that, according to CTM, thinking is **computation**. And we know that computation is possible because computers—the one on your desk or at

the library—*do exactly that*. The existence of computers therefore gives modest support to CTM since it shows that at least some physical structures are capable of performing computations. (I say *modest* support because the existence of computers does not establish that the *mind* is a computer; it only shows that computation is physically possible.)

2. We saw in Chapters 3 and 4 that mental states can be multiply realized. If mental states are computational states, then computational states must be multiply realizable. In Section 6.2 we noted that computational states and processes can indeed be multiply realized. In other words, CTM accounts for the multiple realizability of mental states.
3. CTM requires that thoughts are complex symbols. We saw in Section 6.1 that a complex symbol is built up from basic symbols. (Recall the way that the complex symbol 'Fodor wrote *The Modularity of Mind*' contains the basic symbol 'Fodor'.) More precisely, complex symbols have a *structure*. That is, they consist of basic symbols *organized in a certain way*. Even though the complex symbols 'Lassie bit Bloggs' and 'Bloggs bit Lassie' are made up of *exactly the same basic symbols*, they are quite distinct. What makes them distinct is the way the basic symbols are organized. In other words, they are distinct because they have different structures. In Section 6.5 we will examine a powerful argument which aims to establish that thoughts must have a structure. Demonstrating that thoughts are structured is not enough to establish that CTM is true: perhaps thoughts are structured but thinking is not computation. Nevertheless, establishing that thoughts are structured would considerably advance the claim that the mind is a computer.

So far in this section we have garnered a certain amount of support for CTM; however, problems abound. Here I will briefly mention four difficulties which we will later take up in detail.

1. We have seen that computational processes respect the semantic properties of the symbols involved. This raises an important issue: how do the symbols get their semantic properties? The symbols in a conventional computer get their semantic properties from *us*. The 'Fodor' symbols in this document refer to a certain American philosopher *because I say they do*. It is because *I* am thinking about a certain American philosopher that my 'Fodor' symbols refer to the author of *The Modularity of Mind*. I could, if I wanted to, write a document about Russian literature in which my 'Fodor' symbols refer to the Russian novelist Fodor Dostoyevsky. It's my intention to use 'Fodor' to stand for the American philosopher rather than the Russian novelist which determines that my 'Fodor' symbols refer to the author of *The Modularity of Mind* rather than the author of *Brothers Karamazov*.

So far we have seen that the symbols of conventional computers get their semantic properties from the intentions of the humans using the computer. But where do human mental symbols get *their* semantic properties from? There seem to be, broadly speaking, two possibilities. Either mental symbols somehow get their semantic properties from each other, or they get their semantic properties by being related in some special way to things in the world beyond the mind. Neither possibility is without its difficulties. We will return to these issues in Chapter 9.

2. If thinking is computation, then anything which performs the right sort of computations is a thinker. In Section 6.6 we will consider an apparent counterexample to CTM: a set-up in which all the right computations appear to have been performed and yet no (relevant) thinking has gone on.
3. A further difficulty for CTM takes the form of a rival account of mental processes. Called '**connectionism**', the rival account is the topic of the next chapter.
4. Finally, it's not at all clear that CTM has the resources to account for consciousness. Many people have the intuition that a computer could carry out all the right computations and yet not be conscious. I will say a little bit more about computationalism and consciousness in Section 6.6.

6.5 The language of thought

We have seen that, according to CTM, thoughts are complex symbols which are made up of basic symbols arranged in certain ways. In other words, thoughts are structured. In this section we will consider an important argument which concludes that thoughts are indeed structured entities.

Let's begin by thinking about public languages like English. Consider the sentence, 'Lassie bit Bloggs'. If that sentence is meaningful then so is the one obtained by transposing the two proper names: 'Bloggs bit Lassie'. Similarly, if the sentence 'Smith is smarter than Jones' is meaningful, then so is 'Jones is smarter than Smith'. More generally, if 'X did Y to Z' is meaningful then so is 'Z did Y to X'.

The property of languages just described is called 'systematicity'. **Systematicity** is only guaranteed if the sentences in the language possess a structure. For imagine a language L in which sentences have no structure. Let's assume that in L the basic symbol 'S' means that Lassie bit Bloggs. Notice that there is no guarantee that L has another basic symbol which means that Bloggs bit Lassie: maybe there is; maybe there isn't. On the other hand, if the sentences in the language are structured as they are in English, with word order indicating who did what to whom, it will always be possible to transform one meaningful sentence of the form 'X bit Y' into another meaningful sentence of the form 'Y bit X'.

So the systematicity of language entails that sentences have structure. Let's turn now to a particularly striking feature of public languages—**productivity**. Languages are productive in that we can build new meaningful sentences out of old sentences or parts of old sentences. For example, from 'Bloggs is short' and 'Bloggs is silly' we can obtain the new sentence 'Bloggs is short and Bloggs is silly'. And from 'Bloggs is short and Bloggs is silly' and 'Bloggs is slimy' we can obtain 'Bloggs is short and Bloggs is silly and Bloggs is slimy'. You're not going to get an A+ in your English test for the last sentence, but it is grammatical. This process can be extended indefinitely, making longer and longer sentences until you die of boredom.

The example of productivity just given generates longer and longer sentences, but it doesn't make them more complex. However, it's possible to make sentences which are not only longer but of greater complexity (not to mention interest). For example:

The girl is on the swing.

The girl in the blue blouse is on the swing.

The girl in the blue blouse designed in Paris is on the swing.

The girl in the blue blouse designed in Paris last season is on the swing.

The girl in the blue blouse designed in Paris last season by a beer-swilling Australian cricket fan is on the swing.

You get the idea.

Like systematicity, productivity is only possible because sentences are structured. Old sentences and phrases can be recycled as parts of new, longer, and/or more complex sentences. Once again, imagine that there is a language which lacks structure. The language may have a basic symbol which means 'Bloggs is short' and another one which means 'Bloggs is silly', but there is no guarantee that it will have a symbol which means 'Bloggs is short and Bloggs is silly'.

So far we've been talking about public languages like English. Parallel considerations apply to thought which seems to be both systematic and productive. Let's do systematicity first: if you can think that Lassie bit Bloggs then you can think that Bloggs bit Lassie. Now productivity: if you can think that Bloggs is short and you can think that Bloggs is silly, then you can think that Bloggs is short and Bloggs is silly. We noted in the language case that systematicity and productivity require that language is structured. A similar conclusion arises in the case of thought: thoughts have structure. That is, thoughts are made up of basic symbols assembled into more complex structures.

The conclusion we have just reached amounts to saying that thoughts are, in important respects, language-like; in other words, we think in a language.

This idea is often called the 'language of thought' hypothesis (Fodor 1975). Fodor argues that the language we think in is not the public language we talk in. He holds his position because neither small children nor nonhuman mammals can *talk*, but it seems preposterous to claim that they can't *think*. The special language in which we are said to think is sometimes called 'mentalese' to distinguish it from public languages like English. Other philosophers have argued against this view, arguing that we do indeed think in a public language. In their view, we begin life with a rudimentary version of mentalese, but once we learn to speak a public language we abandon mentalese and think in (say) English.

I won't enter into the mentalese versus public language debate here. For our purposes what's important about the systematicity and productivity of thought is that they make it plausible that thoughts are complex, structured symbols. And that's exactly what CTM requires. Of course, CTM requires more than structured symbols: it requires processes that recognize and manipulate those symbols on the basis of their syntactic properties and in ways which respect their semantic properties. Nevertheless, the plausibility of the language of thought hypothesis lends considerable support to CTM.

6.6 The Chinese room

We saw in Section 6.4 that CTM is compatible with the multiple realization of mental states and processes because computational states and processes can be multiply realized. It follows that if CTM is true mental states and processes could in principle be realized by a digital computer. That is, a digital computer could have (or be) a mind. This idea underpins a research program called *strong artificial intelligence* (or 'strong AI'). The aim of strong AI is to develop programs such that the digital computers running those programs would—quite literally—have minds.

Many practical difficulties lie in the way of actually building an intelligent computer. However, setting aside the practical difficulties we can note that CTM entails that a digital computer could think. So if CTM is true then strong AI is possible. Conversely, if strong AI is *impossible*, CTM is false. The contemporary American philosopher John Searle has constructed a thought experiment which seems to show that strong AI is impossible and, consequently, that CTM is false. For reasons that will be clear in a moment, Searle's thought experiment is called the 'Chinese room'.

Strong AI researchers have attempted to program digital computers to understand simple stories (see Schank and Abelson 1977). For example, the computer might

be expected to understand a simple story about eating in a restaurant. The computer is given three kinds of input:

1. The story.
2. Some general information about restaurants and the kinds of things that typically occur there. For example: people eat in restaurants; people order their food from waiters; people are usually required to pay for what they have ordered; and so on. Researchers in strong AI call this information a 'script'.
3. Some questions about the story.

If the scientists have managed to program the computer properly then, according to strong AI, the computer will not merely answer the questions correctly, it will literally understand the story.

Now for Searle's ingenious counterargument. As it happens Searle speaks English but doesn't understand a word of Chinese. Searle imagines that he is sitting in a room (the 'Chinese room') and receives three sets of Chinese characters and a set of instructions in English. The instructions tell him how to manipulate the three sets of Chinese characters to obtain yet another set of Chinese characters. Following the instructions proves to be very difficult. As he doesn't read Chinese, Searle has to laboriously identify the Chinese characters purely by their shape. After a lot of effort he finally manages to obtain a fourth set of Chinese characters which he hands to the people waiting outside the room.

It turns out that the three sets of Chinese characters are, respectively, a simple story (in Chinese), some general information about the setting of the story (in Chinese), and a set of questions about the story (in Chinese). The instructions in English are the equivalent of a computer program, and Searle is the computer. In laboriously following the instructions he has carried out the computer program, and the set of Chinese characters he has produced are sensible answers to the questions about the story.

It seems overwhelmingly obvious that Searle does not understand the Chinese story. (Compare the situation in the Chinese room with a situation in which Searle is asked to read a story in English. Searle understands the English story; he doesn't understand the Chinese one.) But it follows that strong AI is in deep trouble. For, according to strong AI, a suitably programmed computer *could* understand a simple story. What Searle has apparently shown is that following a computer program is insufficient for understanding. After all, Searle follows the program to perfection, but he hasn't got a clue what the story is about. And if that's right, CTM is in deep trouble for, as we have seen, if strong AI is impossible then CTM is false.

So far we have seen that the Chinese room example prompts the intuition that strong AI is impossible. Searle follows up the Chinese room thought experiment

with the following argument:

1. Programs are formal (syntactical).

2. Minds have contents (semantic contents).

3. Syntax is not identical [with] nor sufficient by itself for semantics.

Therefore,

4. Programs are not sufficient for nor identical with minds; i.e. strong AI is false.

(Searle 1991: 526)

Searle's point is that understanding a story involves knowing what the various symbols (i.e. the words) *mean*. That is, understanding a story involves grasping the semantic properties of the symbols in the story. Clearly, when he is in the Chinese room, Searle has no grasp of the semantic properties of the symbols involved. He is only aware of the shapes of the symbols; that is, he is only aware of their syntactic properties. Searle concludes that, since any computer system is sensitive only to syntactic properties, no computer system is aware of the relevant semantic properties, and so no computer is able to understand a story.

The Chinese room argument has generated an enormous amount of discussion. Let's begin with a reply to the Chinese room offered by Fodor (Fodor 1980*b*). Fodor agrees with Searle about the importance of semantics. However, unlike Searle, Fodor thinks that the symbols of a computer *can* have semantic properties and, in other publications, he has indicated the theory of semantics he thinks will do the trick (see, for example, Fodor 1990*d*). We will look at Fodor's theory in Chapter 9. For the moment let's return to the Chinese room and see how Fodor's suggestion fares.

According to Fodor, the symbol 'Mt Everest' refers to a certain mountain on the border of Nepal and Tibet because there is an appropriate connection between the mountain and the symbol. Similarly, the Chinese symbol 'XYZ' refers to Mt Everest because there is an appropriate connection between the mountain and the Chinese symbol. In Chapter 9 we will see how Fodor spells out 'appropriate connection'. Notice, though, that the presence or absence of that connection won't be obvious to Searle when he is in the Chinese room. All Searle is aware of is the shape of the symbols in front of him. Say that he comes across the symbol 'XYZ' and that that symbol is appropriately connected to Mt Everest. Searle won't know that the symbol is appropriately connected to Mt Everest, nor will he know that 'XYZ' means Mt Everest. Consequently, Searle will not understand the story. In other words, *even if the symbols have a semantics*, Searle will not understand the story.

It's time to pause and consider what's involved in understanding a story. To understand a story is—amongst other things—to be in a certain conscious state.

We sometimes describe this state by saying that we are *aware* of what the story means. Thus it is very natural to say that when Searle reads a story written in English he is aware of its meaning, but when he 'processes' a Chinese story in the Chinese room he is not aware of its meaning. This suggests that a crucial difference between Searle when he processes a Chinese story and Searle when he reads an English story is a difference of consciousness: only in the latter situation is Searle conscious of the content of the story. It follows that the Chinese room set-up is insufficient for conscious awareness of the meanings of stories.

Something important follows from that conclusion. The activities Searle undertakes in the Chinese room are purely computational activities. Consequently, if the Chinese room set-up is insufficient for conscious awareness of the meanings of stories, then *computation* is insufficient for conscious awareness of the meanings of stories. And this result would seem to generalize: computation is insufficient for consciousness.

The moral we have drawn from the Chinese room thought experiment is that computation is insufficient for consciousness. We cannot draw the quite distinct moral that computation is insufficient for *intelligence*. Searle's argument leaves untouched the claim of strong AI that a suitably programmed computer can respond with intelligence to a story. That Searle lacks conscious awareness of the meaning of the story does not show that *intelligent but unconscious* processing of the story is occurring.

It's hardly surprising that consciousness turns out to be a problem for CTM. As we saw over and over again in Part 1, consciousness is a problem. Period. We'll get to consciousness in Part 4.

6.7 Conclusion

CTM is a powerful idea. It offers an account of rational thought, and it is supported by the existence of computers, by the multiple realization of mental states, and by the structure-dependent properties of thought. However, it's not all blue skies and beer. As the Chinese room suggests, CTM has a problem with consciousness. Moreover, a quite extraordinary alternative conception of the mind exists. That's the topic of the next chapter.

SUMMARY

(1) Symbols have both syntactic and semantic properties. Syntactic properties are those which can be recognized by examining the symbol in isolation. Shape is an example. Semantic properties are concerned with meaning. They include

reference and truth value and cannot be recognized by examining the symbol in isolation.

(2) Very generally, computers are devices which manipulate symbols solely in virtue of their syntactic properties but which nevertheless manage to respect their semantic properties.

(3) Turing machines are very simple computers which are extremely powerful. It's obvious that Turing machines are just 'syntactic engines'.

(4) According to CTM, thoughts are complex symbols and thinking is computation.

(5) At first glance the Chinese room appears to show that computation is insufficient for intelligence. I argue that it shows something quite different: that computation is insufficient for at least some kinds of conscious awareness.

FURTHER READING

For a good introduction to the idea of computation, see John Haugeland's *Artificial Intelligence: The Very Idea* (1985), Chs 2 and 3. Haugeland 1985 also contains an excellent discussion of Turing machines (see pp. 133–40).

The great champion of CTM is Jerry Fodor; see in particular his *The Language of Thought* (1975). Fodor 1980*a* is an important presentation of his ideas, but it isn't easy. For a good, sympathetic discussion of the language of thought hypothesis see Sterelny 1990: Section 2.2. For a less sympathetic discussion see Braddon-Mitchell and Jackson 1996: Ch. 10. (It's always a good idea to read the less sympathetic authors!)

Michael Devitt and Kim Sterelny defend the view that whilst we begin life with mentalese, we gradually come to think in a public language. See Devitt and Sterelny 1987: Section 7.3. Peter Carruthers adopts a similar view. His *Language, Thought and Consciousness* (1996) is rather hard, but you might like to take a peek at Section 2.7.

It's worth noting that Fodor himself thinks that computation can't be the whole story about the mind. He expresses his doubts in *The Mind Doesn't Work that Way* (Fodor 2000, especially Chs 2 and 3). The book is in part a reply to Steven Pinker's *The Way the Mind Works* (Pinker 1997): hence the title.

The Chinese room was first discussed by John Searle in his 'Minds, Brains and Programs' (Searle 1980). The volume of *Behavioral and Brain Sciences* in which it appeared also contains a number of important commentaries. Fodor's reply (Fodor 1980*b*) is especially recommended. A more accessible version of the Chinese room argument is Searle 1990. For good textbook-style discussions of the Chinese room see Braddon-Mitchell and Jackson 1996: 107–11 and Sterelny 1990: Section 10.2.

TUTORIAL QUESTIONS

Easy ones

(1) Is CTM compatible with physicalism?

(2) Write a list of syntactic properties; write a list of semantic properties.

(3) Describe the Turing machine. Why is it impossible to build a real Turing machine?

(4) Fill in the blanks to complete the following sentences:

According to CTM, thoughts are . . .
According to CTM, thinking is . . .

A hard one

(5) Compare and contrast CTM and functionalism.

Finally, one for general discussion

(6) Could a computer be conscious?