Journal of Experimental & Theoretical Artificial Intelligence,
Vol. 17, No. 4, December 2005, 385–417

Taylor & Francis
Taylor & Francis Group

Implementation is semantic interpretation: further thoughts

WILLIAM J. RAPAPORT*

Department of Computer Science and Engineering, Department of Philosophy,
and Center for Cognitive Science, 201 Bell Hall,
State University of New York at Buffalo, Buffalo, NY 14260-2000, USA

(*Received June 2005; in final form August 2005*)

This essay explores the implications of the thesis that implementation is semantic interpretation. Implementation is (at least) a ternary relation: *I* is an implementation of an 'Abstraction' *A* in some medium *M*. Examples are presented from the arts, from language, from computer science and from cognitive science, where both brains and computers can be understood as implementing a 'mind Abstraction'. Implementations have side effects due to the implementing medium; these can account for several puzzles surrounding qualia. Finally, an argument for benign panpsychism is developed.

Keywords: Implementation; Panpsychism; Qualia; Semantic interpretation; Simulation; Syntactic semantics

1. Implementation, semantics and syntactic semantics

In an earlier essay (Rapaport 1999), I argued that implementation is semantic interpretation. The present essay continues this line of investigation.

What is an implementation? Let us begin by considering some examples. Table 1 shows pairs of syntactic and semantic domains[†] that are clear examples in which the semantic domain (or model) implements the syntactic domain (or formal system) (cf. Rapaport 1995 for a more elaborate survey). The first three are paradigmatic cases: we implement an algorithm when we express it in a computer programming language; we implement a program when we compile and execute it; and we implement an abstract data type such as a stack when we write code (in some programming language) that specifies *how* the various stack operations

*Email: rapaport@cse.buffalo.edu
[†]I explain my use of these terms in section 1.3.

Journal of Experimental & Theoretical Artificial Intelligence
ISSN 0952–813X print/ISSN 1362–3079 online © 2005 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/09528130500283998

Table 1. Semantic domains that are implementations of syntactic domains.

Semantic domain		Syntactic domain
1. a computer program	is an implementation of	an algorithm
2. a computational process	is an implementation of	a computer program
3. a data structure	is an implementation of	an abstract data type
4. a performance	is an implementation of	a musical score or play-script
5. a house	is an implementation of	a blueprint
6. a set-theoretic model	is an implementation of	a formal theory

(such as *push* and *pop*) will work. Cases 4 and 5 are clearly of the same type as these paradigms, even though we do not, normally, use the term 'implementation' in discussing them. Case 6, another example that arguably can be thought of in the same way, suggests, in addition, that all semantic interpretations can be seen as implementations. In general, a 'real', 'concrete' (but see below), 'fully detailed' entity is an implementation of a 'theoretical', 'abstract' one.

We can see that implementation is a relative notion.[†] An implementation is always an implementation *of* something, which I call an 'Abstraction'. But it should be noted right away that Abstractions are not necessarily 'abstract' in any pre-theoretic sense, nor are implementations necessarily 'concrete'.

But implementation is not a binary relation. It has one more term: *I* is an implementation of an Abstraction *A in some medium M*. For the examples in table 1, the media might be, respectively, a computer programming language, a computer, a computer programming language, an orchestra or acting troupe, bricks and mortar, and set theory. Arguably, there are one or two more terms: A *cognitive agent C uses M* to implement *A* as *I*, possibly *for some purpose P* (cf. Giere 2004). However, further exploration of this point is beyond the scope of this essay (but see ‡ on page 29).

1.1. *Implementation in computer science*

Consider the abstract data type *Stack*, i.e. a 'last-in, first-out' structure specified by axioms allowing new items to be added to it only by 'pushing' them on 'top', and allowing items to be removed from it only by 'popping' them from the top. Here is one way to implement this abstract data type in a programming language:

1. A Stack, s, can be implemented as a 1-element array, $A[0], \ldots, A[n]$, for some n;
2. *top*(s) is defined to be a 1-argument function that takes as input the stack s and returns as output $A[0]$ (i.e. $A[0]$ is the implementation of the 'top');
3. *push*(s, i) is defined to be a 2-parameter procedure that takes as input the stack s and an item i (of the type allowed to be in the array), and yields as output the stack modified so that $A[0] := i$, and $A[j] := A[j-1]$ (i.e. each item on the stack is 'pushed down');

[†]Alternatively, it is a relational category; cf. Gentner (2005).

4. And, almost finally, *pop*(*s*) is defined to be a 1-argument function that takes as input a stack *s* and returns as output the item on the top of *s* (i.e. *top*(*s*)) while moving all the rest of the items 'up' (i.e. $A[j] := A[j+1]$).

I said 'almost finally' because—as should be obvious—some bookkeeping must be taken care of:

5. We have to specify what happens if the stack 'overflows' (as when we try to push an $(n+2)$nd item onto a stack implemented as an $(n+1)$-element array).
6. We have to specify what happens to the 'last' item when the top is popped (does the array cell that contained that item still contain it, or does it become empty?), etc.

These (as well as the limitations due to the type of object allowed to be in the array) can be called 'implementation details', since the abstract data type Stack 'doesn't care' about them (i.e. does not—or does not *have to*—specify what to do in these cases).

Another way to implement a stack is as a 'linked list'. First, a linked list ('list', for short) is itself an abstract data type. It is a sequence of items whose three basic operations are (1) *first*(*l*), which returns the first element on the *list l*, (2) *rest*(*l*), which returns a list consisting of all the original items except the first, and (3) *make-list*(*i*, *l*) (or *cons*(*i*, *l*)), which recursively increments (or *cons*tructs) a list by putting item *i* at the beginning of list *l*.[†] Lists can be implemented in a programming language that does not have them 'built in' by, for example, 2-element arrays (here, the first item in each two-cell unit of the array is the list item itself, and the second item in the two-cell unit is an index to the location of the next item) or by means of 'pointers' (each item on the list is implemented as a two-element 'record', the first element of which is the list-item itself and the second element of which is a pointer to the next item). Finally, a stack *s* can be implemented as a list *l*, where *top*(*s*): = *first*(*l*), *push*(*s*, *i*) := *make-list*(*l*, *i*), and *pop*(*s*) returns *top*(*s*) and redefines the list to be *rest*(*l*).

Thus, a Stack can be implemented in the medium of another abstract data type, *List*. That is, abstract data types can implement each other. So, an implementation need not be 'concrete'. Guttag *et al.* (1978: 74) give an 'example of the implementation of one data type, Queue ..., in terms of another, CircularLists'. This is done as follows: 'We first give, in a notation very similar to that for the specification, an implementation of the Queue type consisting of a *representation* declaration and a *program* for each of the Queue operations in terms of the representation'. In the example, the representation 'medium' is CircularList, and the 'programming language' consists of the operations of CircularLists.

So, an implementation of an abstract data type consists of a representation and programs, where the programs implement the abstract data type's operations, as follows: each operation of the abstract data type is ... defined? explicated? implemented? ... in terms of an operation of the implementing medium (the implementing abstract data type) *after* first representing each abstract-data-type

[†]Some arbitrary object, e.g. 'nil' in Lisp, is used as the base-case list. This is an implementation detail; cf. section 2.

entity (term) by a term of the implementing abstract data type. So, terms get interpreted by, or mapped into, elements of the interpreting domain, and predicates (operations) are mapped into predicates of the interpreting domain.

Moreover, a list implementing a stack can be further implemented in the medium of a particular programming language, say, Lisp (in fact, a particular implementation of that programming language, say, Allegro Common Lisp); that program will itself be implemented in the medium of some machine language, and, eventually, in the medium of some particular computer, say, the Sun sitting on the desk in my office. Each link in this chain is slightly more concrete than its predecessor. (For another example, consider items 1 and 2 in table 1.) The whole sequence of Abstractions and implementations constitutes what Smith has called a 'correspondence continuum' (Smith 1987, cf. Rapaport 1995).

To implement is to *construct* something, out of the materials at hand, that has the properties of the Abstraction; it could also be to *find* a counterpart that has those properties. *Both tasks are semantic*. Thus, each (relatively abstract) link in this chain semantically interprets its (relatively more concrete) implementation (Rapaport 1999). Is it better to view the implementation relation as (an instance of) semantic interpretation, or is it better to view the semantic-interpretation relation as (an instance of) implementation? Or are they simply two names for the same thing? Although I am sympathetic to this third option, semantic interpretation seems to be more general than implementation, because there are correspondence relations that are semantic interpretations but that are not implementations, at least, not in any obvious way, such as the relation between an optic-nerve signal and a retinal intensity pattern (see Rapaport 1995 for discussion). Note that Smith would disagree that this correspondence is a semantic one. However, any correspondence between two domains in which one is used to understand or interpret the other is a semantic correspondence (Rapaport 1995). This generality of semantic interpretation over implementation argues in favour of taking semantic interpretation as basic.[†]

Hayes, indeed, speaks of semantics in this context (1988: 209, bold face added):

> Because of the complexity of the operations, data types, and syntax of high-level languages, few successful attempts have been made to construct computers whose machine language directly corresponds to a high-level language.... There is thus a *semantic gap* between the high-level problem specification and the machine instruction set that **implements** it, a gap that a compiler must bridge.

What is this gap? Presumably, that (say) the specific operations, data types, etc., of the high-level language do not correspond directly to anything in the machine language: Pascal, for example, has the 'record' data type, but my Sun's machine

[†]Albert Goldfain (personal communication) asks, 'What corresponds with semantic misinterpretation? In what ways might X incorrectly implement Y? If S is a musical score and P is a note-for-note performance of S, and P' is exactly like P except for one missed note, then is P' an implementation of S with a side effect, or a misimplementation of S? What about P'', which is like P except for the fact that it is played on different instruments? Also, if I print out a musical score on my printer, is the printer "performing" the score? Why not?' These are all good questions that I hope to explore in future work. For now, note that the relation of P to P' is similar to the relation between an *algorithm* for solving a problem and a *heuristic* for solving that same problem, as characterized in Rapaport (1998; section 2).

language probably does not. So, a compiler is needed to show how to construct or implement records in the machine language.

Why does Hayes call this a *semantic* gap? It is a bit like the fact that one natural language might not have a single word corresponding to some single word in another natural language. Consider Russian, which has a term, 'ruka', referring to what in English has to be referred to as the hand + forearm.[†] Of course, one can translate between the languages by defining the word in terms of others (perhaps with a cultural gloss; cf. Jennings 1985, Rapaport 1988a: 102). But why is this *semantic* rather than *syntactic*?

A possible interpretation[‡] of the 'semantic gap' specifies four relations:

A. A program in a high-level programming language is semantically interpreted by real-world objects. Presumably, the semantic interpretation of the program is the relation between, on the one hand, data structures (say) in a Pascal program (e.g. a record representing a student viewed as consisting of a name, a class, a major, a student-number and a grade-point average) and, on the other hand, an actual student in the real world.

B. The program is also compiled into a machine-language implementation. The compilation relation is, or includes, the relation between that student-record data structure and a construct of data types in the machine language. Both A and B are semantic relations.

C. The machine-language implementation, in turn, is semantically interpreted by bits in a computer. It may seem odd to semantically interpret the machine-language program by bits, rather than by the real-world objects. But recall that all semantic relations are correspondences (and vice versa). Thus, this relation between the machine-language program and bits is just another correspondence. After all, we could also have mapped the Pascal program into computer bits—in fact, via B and C, we have! So, a machine-language program can be interpreted in terms of bits in the computer. Arguably, in fact, having these two distinct interpretations of two distinct (albeit input-output-equivalent) programs is appropriate. Where the machine-language program talks of registers, the Pascal program talks of 'students' (or student-records). So it is appropriate to understand the Pascal program as a 'mathematical model' of such real-world objects as students, and to understand the machine-language program as a 'mathematical model' of such (also real-world) objects as bits in a computer.

D. The semantic gap concerns the relation between A's real-world objects (such as students) and C's computer bits, since *both* are semantic interpretations of the Pascal program.

What, then, is this relation D? It could be *simulation*: the computer bits simulate the student. But simulation is, after all, a kind of *implementation*. The computer bits are a computer implementation of the student, i.e. an implementation of the student in the medium of the computer.

[†]John Sowa, personal communication, 29 November 1993.

[‡]Due to my computer-science colleague Bharadwaj Jayaraman (personal communication).

1.2. *Implementation outside of computer science*

Although the prototypical examples of implementation come from computer science, there are examples in other areas, too.

1.2.1. The arts. Some of the clearest examples outside of computer science of what could be called 'implementation' come from music. This ought not to be surprising: after all, a music score is very much like a computer program or algorithm, and the musician-plus-instrument (or conductor-plus-orchestra) plays a role very much like that of the computer. A musical score is *not*, of course, *mathematically* an algorithm, since much is left open to 'interpretation' by the musician (e.g, tempo, dynamics, optional repeats, phrasing, etc.). Nonetheless, it *is* a set of 'instructions' that, when followed or executed, produce a certain output. The 'process' consisting of the musician playing that music on an instrument can plausibly be said to *implement* the score. The score is a piece of syntax; the playing of the score is a 'semantic interpretation' of it.

An implementation requires an implementing medium. And, as should be evident, there can be many different media, hence many different implementations (the common core of which can be captured by the mathematical techniques of Goguen *et al*. 1978). We find the same thing in music: a given score can normally be played on a variety of instruments, modulo a few changes necessitated by the nature of the instrument. Such changes, as well as the particular features of the instrument, constitute 'implementation details'. Often, these change the nature of the work, for good or bad: 'a [piano] transcription [of a symphony] can hold a prism up to a familiar work, showing it in a new light' (Pincus 1990). That is, a piano transcription of a symphony is an *interpretation* of it—or, rather, *another* interpretation of 'the work', i.e. of an abstract data type (the score) of which both the symphony *and* the piano transcription are (semantic) interpretations or implementations. The implementation is also, of course, an 'interpretation' in the ordinary sense: Rosen speaks of 'the essential gap between the composer's conception of a work of music and the *multiple possibilities of realizing it in sound*' (1991: 50, emphasis added). The 'conception' is the abstract data type; the 'multiple possibilities' are different implementations.

Much the same can be said, *mutatis mutandis*, for scripts and productions of plays (or scripts and movies). Where English talks of a *director*, French talks of a *réalisateur* (a realizer): At least for francophones, plays and movies are *implementations* (of scripts).

1.2.2. Language. Language provides a variety of non-computer-science examples of implementation. For one thing, words can be considered as representations—hence, implementations—of ideas (cf. Harris 1987: xi, ch. 6). For another, if language can be thought of as an Abstraction (as, perhaps, Chomsky's theory of universal grammar would have it), then it can be implemented in a variety of ways: first, by spoken languages (implemented in the medium of speech) as well as by signed languages (implemented in the medium of space; cf. Coughlin 1991), and, second, in many ways in both spoken and signed languages (e.g., French, English, etc., and American Sign Language, British Sign Language, etc.): Bickerton (2004: 504) considers 'language' to be what I am calling an Abstraction, i.e. 'a system of

expression … that may function by means of [such 'modalities'—or implementations—as] speech, sign, Morse code, talking drums, smoke signals, naval flags …; or it may keep its productions within the individual's brain, not employing any modality at all'.

1.3. *Syntactic semantics*

This view that implementation and semantic interpretation are intimately related is part of a more general theory of the nature of the relation between syntax and semantics that I have dubbed 'Syntactic Semantics' (and which I have explored in a series of previous essays).[†] This theory, very briefly, holds that the semantic enterprise is recursive and, fundamentally, a syntactic—i.e. a symbol-manipulation—enterprise. Syntax is the study of the relationships among symbols of a formal system; it includes grammar (the specification of which sequences of symbols are well-formed formulas) and proof-theory (the specification of which sequences of well-formed formulas are proofs—i.e. which well-formed formulas are theorems), and is essentially a computable theory of symbol manipulation. Semantics is the study of the relationship between a formal system that is itself syntactically specified (i.e. a syntactic domain) and a semantic domain that can be specified by an 'ontology' (in the knowledge-representation, as opposed to the philosophical, sense of this overloaded term). The semantic domain is typically taken to provide meanings or interpretations of the symbols in the syntactic domain; the syntactic domain is typically taken to provide a language for talking about the semantic domain. These two domains can be combined. This happens, e.g. in a brain that perceives the external world: Some neuron firings correspond to external objects—i.e. they are caused by perceiving these objects and thereby represent them—and other neuron firings are (or represent) concepts. These latter can serve as meanings of the former. But, crucially, both sets of neurons are part of the same brain (and are, presumably, distinguishable only by their function, not their physical structure). In such cases, the semantic interpretation function becomes an internal relation among a set of symbols (some of which come from the syntactic domain and the others of which come from the semantic domain) and is itself a computable syntactic relation. Moreover, there is nothing privileged about one of these domains with respect to the other: A given syntactic domain for some semantic domain could itself be considered a semantic domain for some further syntactic domain, as in the correspondence continuum mentioned earlier.[‡] In the base case of this recursion, a domain must be understood in terms of itself, i.e. syntactically. In this way, the semantic enterprise is seen to be, ultimately, a syntactic endeavor, which I have called 'syntactic semantics'.

[†]Rapaport (1985, 1986, 1988a,b, 1995, 1996, 1999, 2000, 2002, 2003).

[‡]'It is worth noting that each metalanguage has itself all properties of a sign system. Thus, the syntactic metalanguage not only has a syntactic dimension insofar as it contains signs of various classes with various serial relations holding among them, it also has a semantic dimension since its signs are interpreted to represent meaning and refer to something, namely, to the signs of the object-code' (Posner 1992: 41).

Syntactic semantics applies these ideas to provide a computable understanding of cognition that is both syntactic and methodologically solipsistic (or, perhaps less controversially, 'internal'). The interested (or sceptical) reader can consult my essays cited in a previous footnote for details and argumentation. In the present essay, I wish to explore the implications of viewing implementation as semantic interpretation: the role of 'implementation details', the question of whether an implementation is 'the real thing', and the problem of whether any old thing can be an implementation of any other thing.

2. Implementation-dependent details

2.1. *In the details lie the differences*

Suppose we have two different implementations of an Abstraction. They may be implementations in *different* media, as, for example, implementations of the Stack abstract data type in Pascal using records and in Lisp using lists, or implementations of a computational cognitive agent (such as the SNePS Research Group's 'Cassie')[†] on a Vax running Franz Lisp (as Cassie was in the 1980s) and on a Sun running Allegro Common Lisp (as Cassie is in the 2000s). Or they may be implementations in the *same* medium: for example, two implementations of a fully-equipped Toyota Sienna XLE; here what I have in mind is that the vehicles would be identical except, of course, that the metal, plastic, fabric, etc., of one of them would perforce be distinct physical objects from the metal, plastic, fabric, etc., of the other. For another example, consider two implementations of the Stack abstract data type using Pascal arrays; here, one implementation might use an n-element array A with $\texttt{top} = A[n]$, while the other uses an $(n + 1)$-element array A with $\texttt{top} = A[0]$.

Clearly, the members of each pair of implementations will differ. The operations of the Pascal stack will be defined in terms, say, of record operations, while the operations of the Lisp stack will be defined in terms of list operations. The two computational cognitive agents' input–output behaviours ought to be the same, but the code will differ, so debugging will be a different process on each. One Sienna might dent more easily than the other, or get better gas mileage. Even identical twins differ. (For a novelistic treatment of this, see Duhamel 1931.) And clearly there will be implementation-dependent differences between the array-implemented stacks, some of which are behaviourally irrelevant and some of which have behavioural consequences. For example, 'where' `top` is, i.e. how `top` is implemented, is behaviourally irrelevant (i.e. irrelevant in terms of input–output behaviour). But the size of the array has behavioural consequences: An abstract stack needs no size, but every actual implementation of it will have a size, and this size constraint will affect the behaviour of the implemented stack (because finite stacks can overflow, whereas abstract ones cannot).

[†]Shapiro and Rapaport (1987, 1991, 1992, 1995); Shapiro (1989, 1998); Rapaport (1991, 1998, 2000, 2002, 2003); Rapaport *et al.* (1997); Ismail and Shapiro (2000); Shapiro *et al.* (2000); Shapiro and Ismail (2003); Santore and Shapiro (2004).

My philosophy colleague Jorge Gracia has discussed the relation of an artist's 'general idea of what he [sic] wants to do' and the final product, e.g. a sculpture:

> ... the sculptor's description is too general and does not identify those features of the sculpture that set it apart from others [that satisfy the description]. ... [t]he particular sculpture that the sculptor produces is not the result of his idea alone, but involves also the materials with which he works as well as the creative process itself that produces it. (Gracia 1990: 511–512.)

In general, then, when a given Abstraction is implemented, whether in different media or in different ways in the same medium, there will be implementation-dependent differences. Nevertheless, there will be some core, some essence, common to all of them in virtue of which they can be said to be the 'same'. (This, I take it, is the point of Goguen *et al.*'s (1978) isomorphism construction, as discussed in Rapaport (1999: section 3.2).)

Implementations are always more specific or detailed than their Abstractions, due to the implementing medium. This gives rise to *implementation-dependent side effects*. For instance, ideas can be implemented in different languages (or differently implemented even in the same language). Clarity of exposition, literary art and even cultural variety thrive on the implementation-dependent side effects due to the implementation-dependent differences. *Vivent les différences!*

Consider the implementation of a mind. That is, suppose that (at some future time) we have a collection of algorithms that 'account for' cognition—the Mind abstract data type, as it were. Suppose that we have neurological evidence that these algorithms are implemented in the human brain, and suppose that 'intelligence artificers' (Dennett's term) have implemented them on a supercomputer. We should expect that there will be *implementation-dependent* differences between human minds and such computer minds. Does this mean that the computer mind is not a 'mind'? I understand this question in the following way: is the computer mind an implementation of the Mind abstract data type? The answer, by hypothesis, would clearly be 'Yes'. Are the differences 'important'? That, of course, depends on what counts as being 'important'. Perhaps there will be a need to talk of *degrees* of 'mindhood' (cf. Rapaport 1993). Perhaps, for example, the Mind abstract data type will not be able to be fully implemented in dogs, or in chimps. Or perhaps we will be able to distinguish between a *Human Mind* abstract data type and a *Dog Mind* or *Chimp Mind* abstract data type, or, for that matter, a *Robot Mind* abstract data type.

Perhaps, in the long run, the *only* differences that will be of any significance will be the implementation-dependent ones—the physical differences—and even these will be of no more (or perhaps no less) significance than the implementation-dependent differences that *currently* exist due to the fact that *your* mind is implemented in *your* body and mine in mine. Suppose, for example, that androids like Lt. Commander Data of *Star Trek: The Next Generation* become commonplace. Would we—*should* we—behave differently towards them only because of their physical (i.e. implementation-dependent) differences? Suppose that some cognitive agents are 'aware' or have 'subjective experiences' (measured by, for example, whether they have faces or are human, or by some primitive 'feeling' or 'intuition' that they are aware), while others are *not* thus 'aware' (e.g. some computers). Suppose further that these two kinds of cognitive agents are not behaviourally distinguishable (perhaps only physically distinguishable—i.e. distinguishable on the basis of certain perceptual aspects of their implementation). Given this behavioural indistinguishability, I would say that

we would *not* behave differently towards them. (Better: We *should* not behave differently towards them; consider, after all, the ugly varieties of racism.) We would (or should) not behave differently even towards the *non*-'aware' ones (cf. the Turing Test): for even they, because they were behaviourally indistinguishable, would *claim* to feel pain, say; so it would be morally wrong to inflict (what they call) pain on them. What, then, would be the difference between them? Only a linguistic convention.

2.2. *Implementation-dependent side effects*

Consider an object that is a model of something. Which parts, aspects, or features of it contribute to its role as a model, and which pertain to its own nature—to its implementing medium?[†] From the fact that a globe is plastic, we do not infer that the world is plastic (nor do we require that a model of the world be made of the same stuff—rocks, water, soil, etc.—that the world is). Nor—as in a *Family Circus* cartoon showing a little boy next to a globe, asking 'Does the real world have writing all over it?'—do we infer that the world has writing on it from the presence of place names and lines of latitude and longitude on globes, even though these *are* part of its role as a model. Where do implementation-dependent side effects come from, and what, if anything, do they do?

Implementation-dependent side effects are due to implementation-dependent details. If we think of an Abstraction as a syntactic domain and of an implementation as a semantic model of the Abstraction, then it appears that the details come from situations in which the semantic domain is 'larger' than the syntactic domain. These are situations in which everything in the syntactic domain is interpreted in the semantic domain, yet in which not everything in the semantic domain is an interpretation of something in the syntactic domain. To adapt some terminology from logic, the syntactic domain is 'sound' but 'incomplete' (i.e. it is abstract). In this way, individuals can have properties that their universal lacks:

> ... written, spoken, and mental texts are all individual insofar as they are not instantiable themselves. ... As individual instances, moreover, they presuppose corresponding universals, but the universal is not the same for the three types of texts. *For the written text, it would be a written type of universal even though the universal would not be something written anywhere.* (Gracia 1990: 505–506, emphasis added.)

Such implementation-dependent properties, we see, *can* be essential properties of the individual; we will come back to this in section 2.3.

Another source of implementation-dependent details is non-isomorphic models (cf. Rapaport 1995: section 2.2.2). For example, consider non-isomorphic models of the group axioms (i.e. of the Group abstract data type): (1) two groups of different cardinalities (e.g. the cyclic groups of orders 2 and 3) or (2) an infinite cyclic group such as (a) the integers under addition and (b) the Cartesian product of that group with itself (which, unlike the former, has two disjoint subgroups except

[†]Perhaps this question is related to the medieval question of haecceity—the 'thisness' that makes Plato *this* man and Socrates *that* man, even while both are instances of (or implementations of?) the universal Man (cf. Rapaport 1999: section 4.1).

for the identity).[†] In each case, the implementation—the model—has features that are left *unspecified* by the Abstraction (in this case, the group axioms); they are implementation-dependent details. Indeed, even *isomorphic* models give rise to implementation-dependent differences: 'In any isomorphic class there are models which differ on *all* non-empty extensions. For example, in any isomorphism class there is one model at least whose domain consists of odd integers and one whose domain consists of even integers' (Jardine 1973: 231; Jardine points out that this gives rise to Quinean indeterminacy of reference).

Sometimes, the implementation-dependent details are not important and can—or even *must*—be ignored. This is because the *purpose* of an implementation or model is often to aid in understanding the Abstraction. There are two sides to this coin: if the *Abstraction*—better, the syntactic domain; i.e. the domain to be modelled; i.e. the domain to be understood in terms of the model—is itself complex, we will want the *model* to be simpler.[‡] Nonetheless, it will still have features that do not represent any part of the Abstraction: 'It may be *richer* in properties, but these would then not be ones relevant to its object [i.e. the Abstraction]; it [i.e. its object] wouldn't possess them, and so the model couldn't be taken to represent them in any way' (Wartofsky 1966: 6–7). The extra properties are implementation-dependent details, to be ignored. (Goguen *et al.* 1978 employs a construction to 'divide out' such irrelevancies; cf. Rapaport 1999: section 3.2.)

Often, however, the details *do* contribute something: This is the realm of the implementation-dependent *side effects*—phenomena contributed by the implementing medium, not by the Abstraction. Some are behaviourally relevant, others not. That a stack's top is implemented as $A[0]$ rather than $A[n]$ is not behaviourally relevant. A high-level program that cares only about stacks and not about their implementation can—and does—ignore this. Any modern programming language with built-in data-abstraction mechanisms *literally* ignores—does not know—about the implementation details (cf. Parnas 1972).[§]

But as side effects become more and more behaviourally relevant, they become more than mere *side* effects and can be of central importance. Let us consider

[†]I am grateful to my mathematics colleague Nicolas Goodman for this example.

[‡]'It is rather paradoxical to realise that when a picture, a drawing, a diagram is called a model for a physical system, it is for the same reason that a formal set of postulates is called a model for a physical system. This reason can be indicated in one word: simplification. The mind needs in one act to have an overview of the essential characteristics of a domain; therefore the domain is represented either by a set of equations, or by a picture or by a diagram. The mind needs to see the system in opposition and distinction to all others; therefore the separation of the system from others is made more complete than it is in reality. The system is viewed from a certain scale; details that are too microscopical or too global are of no interest to us. Therefore they are left out. The system is known or controlled within certain limits of approximation. Therefore effects that do not reach this level of approximation are neglected. The system is studied with a certain purpose in mind; everything that does not affect this purpose is eliminated' (Apostel 1961: 15).

[§]Albert Goldfain (personal communication) 'imagine[s] a poorly designed operating system (with a bad memory-management scheme) where $A[n]$-top stacks might overwrite some piece of reserved memory while $A[0]$-top stacks do not. There is usually a 'larger' context in which behaviourally irrelevant "implementation details" become relevant'. On the importance of suitably large contexts, see Rapaport (2005).

some examples. For instance, consider the following case of a chess game played with non-standard pieces:

> In today's chess, only the familiarly shaped Staunton pieces are used. ... [One] reason is the unfamiliarity, to chess players, of other than Staunton pieces. ... [In Reykjavik, in 1973, two grandmasters] started to play [with a non-Staunton set], and the conversation ran something like:
>
> 'What are you doing? That's a pawn.'
> 'Oh. I thought it was a bishop.'
> 'Wait! Maybe it is a bishop.'
> 'No, maybe it really is a pawn.'
>
> Whereupon the two grandmasters decided to play without the board. They looked at each other and this time the conversation ran:
>
> 'D5'
> 'C4'
> 'E6'
> 'Oh, you're trying *that* on me, are you? Knight C3.'
>
> And they went along that way until they finished their game. (Schonberg 1990: 38–39.)[†]

In this anecdote[‡], the implementation of the Abstract chess pieces had confusing implementation side effects.

Wilfrid Sellars (1955 [1963]) discussed another chess-related example of implementation:

> [A]ttention must be called to the differences between 'bishop' and 'piece of wood of such and such shape'. ... [The former] belongs to the rule language of chess. And clearly the ability to respond to an object of a certain size and shape *as a bishop* presupposes the ability to respond to it as an object of that size and shape. But it should not be inferred that 'bishop' is 'shorthand' for 'wood of such and such size and shape' .... 'Bishop' is a counter in the rule language game and participates in linguistic moves in which ... the ... longer expression does not .... (Sellars 1955: section 56 [1963: 343].)

'Being a bishop' is a nice example of what I call an Abstraction. Here, a bishop is implemented as a certain piece of wood. It could also, as Sellars observes, be implemented by a Pontiac if the chess game is played in Texas, where everything is supposed to be bigger:

> [T]he term 'bishop' as it occurs in the language of both Texas [where it is 'syntactically related ... to expressions mentioning different kinds of cars' (section 59, p. 344)] and ordinary chess can be correctly said to have a common meaning—indeed to mean the bishop role, embodied in the one case by pieces of wood, and in the other by, say, Pontiacs .... (Sellars 1955: section 62 [1963: 348].)

Here, we have an Abstraction (Chess) and two implementations (the ordinary Staunton pieces and the Texas pieces). We assume that the pieces that play the role of the bishop are both *called* 'bishops'; 'bishop' means the same thing in both implementations, namely, the Bishop Abstraction. That role is 'embodied as'—i.e. *is implemented by*—a Pontiac in Texas and a certain shaped piece of wood in the Staunton set. The words 'bishop' as they occur in the two different languages refer

[†]Cf. a similar conversation, in a language of 'nerve states', in Eco (1988).
[‡]Which, I should add, is highly doubted by at least one chess-playing philosopher I've mentioned it to!

to different entities (the language-entry and -departure rules in Sellars's language games differ). Sellars's Texas chess, played with Pontiacs implementing bishops, will have, if not *confusing* side effects, certainly *significant* ones—the chess board will have to be pretty large, and perhaps a speed limit will have to be imposed on the bishops.

Less frivolously, perhaps, problems with an implemented computer system may be due to details of the implementation that are not part of the original specifications. That is, the system might mathematically 'satisfy' the specifications, yet still fail due to hardware faults:

> . . . hardware does from time to time fail, causing the machine to come to a halt, or yielding errant behaviour (as for example when a faulty chip in another American early warning system sputtered random digits into a signal of how many Soviet missiles had been sighted, again causing a false alert . . .). (Smith 1985: 635)

This, I take it, is at the heart of James H. Fetzer's arguments against program verification (Fetzer 1988, 1991, cf. Nelson 1992, 1994).

To some extent, the notion of an implementation-dependent detail and its attendant 'side' effects is a relative one. Recall Gracia's example of the individual written text and its non-written 'written type of universal'. There would, however, be a further universal, of which the 'written-type' and 'spoken-type' of universals are instances. For example, a high-level universal might be Lincoln's Gettysburg Address, of which the written-universal and the spoken-universal are species; one written individual falling under the former would be the one Lincoln allegedly wrote on the back of an envelope, and one spoken individual falling under the latter would be the one Lincoln uttered on 19 November 1863. Or compare Euclid's algorithm for computing greatest common divisors with that algorithm implemented in Pascal, and with that algorithm implemented in Lisp; each of these can be (further) physically implemented as processes on a variety of machines.

Each level of Abstraction or implementation ignores or introduces certain details. One level's implementation detail is another's Abstraction. That is, we can (via a kind of reverse engineering) 'abstractify' an implementation's details, after which they are no longer 'details' *relative to* the new (more detailed) Abstraction. Consider, for example, the Stack abstract data type and the *N*-Element Stack abstract data type. A Pascal *n*-element array-implementation of a stack (*simpliciter*) will have as an implementation detail (yielding behaviourally observable side effects) that it can only store *n* elements. Yet the very same code will *also* be an implementation of an *N*-Element Stack and, as such, will *neither* have that feature as an *implementation*-dependent detail *nor* as a *side*-effect—indeed, it will be an essential feature.

Note that we have two senses of 'abstract' here: the sense in which abstract data types, specifications, and blueprints are 'abstract' (relative to implementations) and the sense in which to abstract is to eliminate (or ignore) 'inessential' 'details': 'every model deals with its subject matter *at some particular level of abstraction*, paying attention to certain details, throwing away others, grouping together similar aspects into common categories, and so forth' (Smith 1985: 637). Note, too, that the model need only be '*assumed* simpler' (Rosenblueth and Wiener 1945: 317, emphasis added): The implementation-dependent details are *ignored*, not *eliminated*. They are parts of the model that are *not* (intended to be) representations of the system being modelled.

2.3. *Qualia: that certain feeling*

The view of implementation as semantic interpretation, with its implementation-dependent details giving rise to implementation side-effects, offers an interesting angle on the puzzles of qualia. Qualia, roughly, are the subjective, qualitative 'feelings' or 'sensations' or 'experiences' that accompany various mental states and processes. Examples are the 'look' of blue (as opposed to yellow, and of yellow as opposed to blue) and the 'feel' of pain (or, for that matter, tactile sensation *simpliciter*). The puzzle is that these are 'private' or subjective phenomena: only I can know what my sensation of blue looks like or what my pain feels like (or that I am in pain). You cannot know what my sensation of blue is like or what my pain feels like, or know that I have any blue-sensation or that I am in pain. You can, perhaps, feel a pain that 'is like' my pain—though how would you (or anyone, for that matter, including me) really know that it 'is like' mine, since you can only feel your own? (Cf. Smith's (1985) 'gap' in our knowledge about whether our models match the world; for discussion, cf. Rapaport 1995: section 2.5.1.) In any case, your pain is not *my* pain. You can, perhaps, determine that I am in pain—but only on the basis of my publically observable physical behaviour, and that, of course, could be mere show or—more radically—be 'real' pain behaviour unaccompanied by any qualitative painful sensation (so-called 'absent' qualia; see section 2.3.1). So, qualia are private, hence 'mental' (according to a long-established tradition). Hence, they ought to be explainable functionally or as part of the Mind Abstraction. Yet, functionalism seems incapable of explaining them, because mental phenomena with different qualia are functionally indistinguishable, or so the puzzle goes.

A possible way out, I propose, is *to view qualia as dependent on implementation side-effects*. This does not resolve the puzzle completely, however, for we still have to account for the privacy of qualia.

2.3.1. Absent qualia. Let us begin with the problem of 'absent qualia': the possibility that, for example, I feel no pain in circumstances in which others do, yet I am not oblivious to the pain stimulus—I behave appropriately. Thus, an experimenter sticks pins in my right hand and in yours. We both wince, withdraw our hands, perhaps cry out; we both say that the pin-pricking hurts, perhaps we both bleed, and we complain of residual soreness over the next several hours. Yet you *feel pain* and I do not (or so we suppose for the sake of argument). The questions are: (1) Is this possible? (2) Am I any 'less' of a cognitive agent because of my lack of feeling? The issue is sharpened when I am replaced by a computer or, better, an android: does the android *feel* pain? Many suppose not. But why? The central issue here is one of subjectivity, the same issue that is at the heart of the Chinese Room Argument (cf. Rapaport 2000): does an entity that passes a Turing-like test—in this case, one for pain or pain-behaviour—'really' have the phenomenon being tested for? And, if *not*, does that mean, despite its behavioural indistinguishability from a human that *does* have the phenomenon, that it is only 'going through the motions' and not 'really' feeling, using natural language, or thinking?

I have mixed feelings about this (if you will excuse the pun). On the one hand, I want to say that insofar as having—or lacking—the private sensation has *no*

behavioural consequences (not even to my being able to describe my pain-sensation in exquisite and poetic detail—whether I have it or not), then it is *not* part of the Mind Abstraction. If I *do* feel pain, then my sensation must be due to the *medium* in which it is implemented, namely, my *body*—it is an implementation side-effect. I can, of course, perceive the pain sensation. Moreover, it *is* possible that the Mind Abstraction can deal with this despite the fact (if fact it be) that, despite the privacy, it is not a mental phenomenon: for the Mind Abstraction will have, let's say, a variable or data structure of some sort whose value *would* be the sensation if I *had* a sensation and whose value is unassigned otherwise. The assignment of a value to this variable or data structure is input from my body. That is how it is implementation dependent.[†]

On the other hand, I think it is plausible that there are never any absent qualia. Take pain, and consider the following computational implementation of it suggested by my computer-science colleague Stuart C. Shapiro (in conversation, *c*. late 1980s; all of this ought, by the way, to be able to be done with current technology): imagine a computer terminal with a pressure-sensitive device hooked up to the central processing unit in a certain way that I will specify in a moment. Program the computer with a *very* user-friendly operating system that allows the following sort of interaction (comments in parentheses):

(User logs in, as, say 'rapaport')

System: Hi there, Bill! How are you? What can I do for you today?
(Assume that this only occurs at the first login and that the operating system is capable of some limited, but reasonable, natural-language conversation.)
User: I'd like to finish typing the paper I was working on yesterday—file 'book.30sep92'.
System: No problem; here it is!
(The file is opened. The user edits the file, closes it, and then hits the terminal sharply on the pressure-sensitive device.[‡] Assume that this device is wired to the computer in such a way that any sharp blow sends a signal to the central processing unit that causes the operating system to switch from *very*-user-friendly mode to 'normal' mode.)
System: File 'book.30sep92' modified and closed. Next command:
User: I'd like to read my mail, please.
(System runs mail program without comment. User exits mail program.)
System: Next command:
(User logs off; logging off in the context of having struck the pressure-sensitive device causes the operating system to switch to yet another mode. The next day, User logs in . . .)
System: Rapaport. Oh yeah; I remember you. You hit me yesterday. That hurt!

Now, what might be going on here? We have a computer with an artificial-intelligence operating system that is exhibiting pain behaviour. Modulo the differences between the computer and a human, and the limitations of the natural-language interface, behaviourally (or, from the intentional stance) it is reasonable to infer (or assume) that the computer was in pain when I hit it. But did it *feel* pain?

[†]For a recent version of this theory, see McDermott (2001).

[‡]A cartoon by Nick Hobart that appeared in *The Chronicle of Higher Education* a few years ago showed two people discussing a computer monitor displaying the message, 'NOW SLAP MONITOR ON SIDE AND SWEAR', while one person says, 'Now this one's *really* user-friendly'.

Well, how do *humans* feel pain? We feel pain when certain neurons are stimulated and certain signals are sent to the brain. Now, in our computer, certain wires connecting the pressure-sensitive device with the central processing unit are 'stimulated' and certain signals are sent to the central processing unit. Where's the difference between human and computer? Perhaps the difference is that, for humans, there is a 'pain-sensing' neuron in the brain that is stimulated when a human is hurt. It gets its input from the pain neurons (C-fibres, or whatever), which also send their input to certain motor neurons that results in typical pain behaviour (or perhaps the pain-sensing neuron sends its output to the motor neurons). Fine; build a similar such device into the central processing unit and operating system. The cases are parallel: either there is a quale in both cases, or there isn't one in either case. Since, by hypothesis, I feel pain, there should be a quale in both cases.

What is that quale? It is tempting to say that it is what I feel when the 'pain-sensing neuron' fires, but this raises the specter of a homunculus doing the feeling. It is, perhaps, better to say that the quale (the feeling) just *is* the firing of that neuron (perhaps *as experienced from* the first-person point of view). It is one thing to say that there *is* a feeling, another to *describe* it: what does the computer's pain *feel like*? I do not know. Do you know what *my* pain *feels like*? We will come back to this in the next section. My point, for now, is that pain qualia can *and will* arise whenever there is pain-behaviour, and the same holds, *mutatis mutandis*, for any qualia.

2.3.2. Inverted qualia. Consider, next, the problem of 'inverted' or 'shifted' qualia: the general problem of accounting for the particular 'feel' of a qualitative experience, assuming the *presence* of qualia: does your pain feel like mine? Does your sensation of blue look like mine? In the most perverse case—the inverted spectrum case—your sensation of blue is just like my sensation of yellow, and vice versa, all across the spectrum.[†] In an inverted-pain case, your feeling of pain might be just like my feeling of pleasure, and vice versa. In what I am calling 'shifted' qualia, red might be shifted down to orange, orange to yellow, etc. Or in a 'shifted' pain case, a pain of intensity 10 might be shifted down to a pain of intensity 9, and so on. (And this might even account for why some people have different tolerances for pain.)

Can this be? How? Well, first, it seems plausible that something like this, if not quite so extreme, *can* be. There are the experiments with inverting lenses (Stratton 1897) in which the subject becomes acclimated to seeing the world upside down—behavioural indistinguishability with distinct qualia. There appears to be no reason in principle not to be able to adapt this to inverted spectra (Cole 1990). And many of us can experience 'shifted' qualia by closing one eye: in my own case, at least, colours appear distinctly different to each of my eyes (colours seem 'shifted' to a darker shade in one eye); by crossing my eyes so as to produce a double image, I can even compare the differences in colour.

Again, I suggest, this is merely an implementation-dependent side effect. Rather than speculating on how the brain might be wired, let us again consider a computer example. Consider two computer programs with the same input–output behaviour, written in Pascal using stacks. Suppose that one of them implements the Stack

[†]Possibly excepting a fixed point? For various options along these lines—various implementations of the inverted-spectrum Abstraction, if you will—see Byrne (2005).

abstract data type as an n-element array $A[0], \ldots, A[n-1]$ with $\texttt{top} = A[0]$, while the other implements it as an n-element array $A[0], \ldots, A[n-1]$ with $\texttt{top} = A[n-1]$. The internal mechanisms—the *implementations* of the stacks—are 'inverted' with respect to each other, yet this is behaviourally undetectable and irrelevant. Granted, here there is no issue of 'qualitative feel', perhaps. Yet the point is that the differences—and there clearly are differences, although not input–output ones—are implementation dependent. The analogue of qualia are implementation-dependent side effects. Similarly for pain: the *sensation* or *feeling* of pain, in humans, might be something that your body has (or does, or undergoes) when, for instance, you step on a tack. But that it feels the way it does is an epiphenomenon (so to speak) *of the body*. Were the same mind implemented in a different body (as in Justin Leiber's 1980 novel, *Beyond Rejection*), perhaps the feeling would be different (or absent).

2.3.3. The syntactic semantics of qualia. Are qualia 'mental' phenomena? They are private, yet (I hold) they are implementation dependent. Does that mean that functionalism (or strong artificial intelligence) fails to 'model' some mental phenomena? That is certainly one interpretation, one move that can be made in the philosophical game. Or does it mean that what it fails to model (pain, spectra inversion) is not mental? That is, of course, another equally plausible interpretation, another move that's open, *unless* one defines the mental in terms of what is 'private' (i.e. not publicly accessible). Yet another option is that *some* of what we call 'mental' is body (or implementation) dependent, though this is not available for those who define bodily phenomena in 'public' terms. (On the relation of a (philosophical) problem to other assumptions that need to be made in order to solve it, see Rapaport 1982.)

The position I find congenial is to make the 'syntax' 'complete'. Recall my suggestion that implementation side-effects were due to situations where the semantic domain exceeded the syntactic domain. In such cases, we can *extend* the syntactic domain to make it match the semantic domain, in a way reminiscent of Hilbert's (1925) notion of 'ideal' elements in mathematics (see section 3; cf. Rapaport 1981, where I show how to do this in a Meinongian fashion to handle non-referring terms). Although any Mind Abstraction may be incomplete in *this* sense of having implementation side-effects, the *fact* of having such implementation side-effects can be made part of the Abstraction, as indicated earlier with my discussion of variables whose values are assigned externally. In this way, to paraphrase Tolstoy, every cognitive agent will 'feel pain', but everyone's pain will 'feel' different.

The random digits 'sputtered' by a faulty chip that were interpreted as enemy missiles (Smith 1985) were also implementation side-effects—(physical) implementation details that yielded or gave rise to 'mental' behaviour: the computer interpreted certain physical configurations as meaning something[†]—it 'felt' them in a certain way, so to speak. A feeling of pain is the mind's *perception* of a physical event.

[†]'[I]nformation itself has no meaning. Any meaning can be assigned to a particular bit pattern as long as it is done consistently. It is the interpretation of a bit pattern that gives it meaning. ... A method of interpreting a bit pattern is often called a *data type*. ... [A] type is a method for interpreting a portion of memory' (Tenenbaum and Augenstein 1981: 6, 45).

Thus, qualia can be thought of as the locus of 'interaction' of mind and body, of Abstraction and Implementation.

It is not, therefore, unreasonable that qualia would be physical, yet 'private'. The actual 'feeling' belongs, and only belongs, to the implementation. Consider *Hamlet's* sadness (at, say, his killing of Polonius) as opposed to *Olivier's* sadness (at, say, learning of the death of a good friend) and as opposed to *Olivier-qua-Hamlet's* sadness, i.e. the sadness manifested by Olivier playing Hamlet (or by Hamlet as played by Olivier). In the Method School of acting, Olivier-qua-Hamlet's sadness would be an implementation of the Abstraction Hamlet's Sadness in the medium of Olivier's sadness. This is to be distinguished from Olivier merely 'acting' sad (perhaps a case of absent qualia?). (For more discussion of this, see Rapaport 1985, 1988b.)

The privacy of qualia just *is* its subjectivity. Compare the following three experiences:

1. Suppose that you and I are both looking in a mirror at my reflection and that we both see, in the mirror, something on my eyelash: this is an 'objective', public, and external perception—a perception from the third-person point of view: We are both perceiving the same thing in the same way, only from different angles (literally from different perspectives).
2. Now suppose that you are looking directly at me, seeing the object on my eyelash, and that I see it out of the corner of my eye directly on my eyelash: this is also objective, public and external, since both you and I are looking at the same thing, but from different perspectives. But my first-person perspective is somewhat closer to the source, so to speak. Here, it is not only the angle that I see it from that differs from yours, but, also, *only* I can perceive it from that angle. In theory, however—were you small enough—you could also perceive it from that angle.
3. Finally, suppose that I *feel* it on my eyelash: This is no different from the other perceptions, except that it is not perceivable by anyone other than me. It is subjective, private and from the first-person point of view. Here, it is impossible for you to perceive what I am perceiving (perhaps not *logically* impossible, but *physically* impossible).

By a continuous sequence of perspective shifts (not unlike those cited in some versions of the Argument from Illusion; cf. Ayer 1956: 87–95, Rapaport 2000, section 6.2), we move from a public to a private phenomenon.

Is there any *more* or *other* difference between these? I think not. Pain, etc., are just the way things are perceived in certain circumstances, some of which cannot be experienced by anyone other than the subject. 'That certain feeling' *ought* to be private, because it's due to the *experiencer's* implementing medium, not anyone else's. *Privacy is not a mental phenomenon, but merely a perspective accessible only to the perceiver or cognizer.*

3. The real thing

One aspect of the question whether machines can think is this: is a computer 'simulation' of a mind 'really' a mind? Compare this, for the moment, to another

question: is a computer simulation of a hurricane 'really' a hurricane? (Cf. Dennett 1978, Hofstadter 1981, Rapaport 1988a.) What is the relation of a simulation to 'the real thing'?

3.1. *Understanding abstraction and implementation*

The first observation I would like to make in this regard is that experience with an *implementation* of some Abstraction can change our understanding of the nature of the *Abstraction*. Can 'straight lines' be implemented in a non-Euclidean geometry? The answer is 'Yes', but they are not 'straight' anymore; they can only be implemented as *geodesics*: shortest distances between two points. So, on a sphere, the implementation of the Abstraction Straight Line is a great circle. Similarly, consider the implementation by airplanes of flying (cf. Rapaport 2000): Airplanes fly, but not the exact way birds do (e.g. although their wings might have more or less the same shape, planes do not flap them). Planes fly only (or at least?) in the sense of moving through the air without touching the ground. No doubt that needs to be refined, so as to rule out long jumps (but might not a *very* long jump *be* flying?). Yet another refinement might replace the reference to air with a general term for a fluid medium: It has been suggested that the knowledge-representation community's favourite flightless bird—the penguin—does indeed fly ... in water! (Ackerman 1989: 45–46; cf. Rapaport 2000.) The point is, as we saw earlier (section 2.1), that when an Abstraction is implemented in different media, there will be implementation-dependent differences, yet there will be some common essence to both, in virtue of which they can be said to be the same. Thus, what is 'really important' about straight lines is that they are geodesics; that geodesics are 'straight' in Euclidean space is an implementation-dependent side effect—an 'accidental' property, if you wish.

Dijkstra (1984: 2) said that Turing's 'question of whether Machines Can Think ... is about as relevant as the question of whether Submarines Can Swim'. Assertions of equivalence such as this are notoriously ambiguous. Does Dijkstra think that it is obvious that submarines *cannot* swim (and therefore that computers *cannot* think)? Or that it is obvious that they can? Or that it is merely a question of whether we will extend the meaning of 'swim' to cover whatever it is that submarines do? Suppose the latter. What is it that submarines do? They move in the water. But that is what swimming is,[†] though perhaps before the advent of submarines we thought that swimming *had* to be done by animals. Do fish swim? Surely. Do people? Perhaps only by extending the term. Extending the meaning of a term occurs when we realize or decide that a property that we thought was essential is not. This goes a long way toward explaining the unease people feel when they're told that computers can think.

So, is this extension of terms such as 'fly' to planes, 'swim' to submarines and 'think' to computers 'merely' a metaphorical extension? It may be metaphorical, but it is not 'mere':

> **Eus[ebius]**: ... I do wish you would stop using terms borrowed from human behavior [to describe monkey behaviour]! You're being anthropocentric!

[†]Unless, of course, swimming is flying in water!

> **Soc[rates]**: Well, monkeys are anthropoids. Besides, do you want me to make up a new word for a phenomenon for every species that shows it? Should geneticists stop talking about inheritance because that term was borrowed from economics? (Altmann 1989: 260)

There are two points: First, *refraining* from such extensions, metaphorical or otherwise, would force us to miss important generalizations. Second, as Lakoff and Johnson (1980) have shown us, metaphor is an unavoidable and central feature of our language and thought.

3.2. *Segregation of implementation*

What we do have to be careful about is mixing our metaphors. That is, an implementation must be complete unto itself; we must not import or apply (implementation-dependent) features from one implementation of an Abstraction to another implementation of the same Abstraction. Thus, to take the classic case, it is of course not true that computer-*simulated* hurricanes get *real* people wet. But they *do* get *simulated* people *simulatedly* wet (Hofstadter 1981; cf. Rapaport 1988a, Shapiro and Rapaport 1991). 'Obviously, a computer simulation of a stomach would only digest simulated food' (Johnson 1990: 46). And a 'simulated engine wouldn't generate any ''here in the world outside the computer'' power—but if you put it in a suitably simulated car, and engage the suitably simulated clutch, it will just fine drive down the simulated road' (Minsky 1991). In each of these cases, we do seem to have 'the real thing': A simulation of digestion *is* a kind of digestion, a simulated hurricane *is* a kind of hurricane. More accurately, I propose, a computer simulation of human digestion is an *implementation* of the Digestion Abstraction, as is human digestion itself (in fact, the scientific study of digestion has recently been aided by a 'virtual stomach'; cf. Eisenberg 2002). The latter may be more familiar, more prototypical (cf. Rosch 1978), but both, just as Dijkstra may have observed of swimming, are really digestion.

The difference between a 'simulation' of flying or of a hurricane and what we normally think of as 'real' flying or a 'real' hurricane is that the former 'are one step removed from reality ... [because they use] symbolic parameter values that represent physical behavior' (Johan Lammens, personal communication, 17 August 1990). I am not sure about the cause, but I agree with the effect: Computer simulations are not part of the 'real' world (except, of course, in the sense in which *everything* is part of the real world). They exist in their own simulated world, and we must be careful about 'transworld' attributions. Although a simulated hurricane will not get *us* wet—because that would require a 'transworld' causal relation of a kind that does not exist—the simulated hurricane must have some of the 'same' (or analogous) cause-effect relationships with denizens of *its* computer universe (e.g. getting simulated people simulatedly wet) in order for it to count as a simulation—in order for both it and 2004's Hurricanes Charley, Frances, Ivan and Jeanne to be *implementations* of the Hurricane Abstraction.

3.3. *Non-segregated implementations*

There is, however, an important family of exceptions to this principle of segregation. Computer simulations of semantic or information-processing systems are not only

implementations of them, but also can interact with other such implementations. In other words, if they were simulated hurricanes, they *could* get *us* wet.

A clear example of this is the computer simulation of computation itself. In one of my introductory computer science courses, I once used a piece of software called the 'P88 Assembly-Language Simulator'. 'P88' was (a fragment of) an assembly language for a hypothetical machine (Biermann 1990). (We can ignore for now whether it really *was* an (incomplete) assembly language *even if* 'just' a toy one, since that is irrelevant to the point I want to make.) The P88 Assembly-Language Simulator was a Pascal program (actually, a ThinkPascal program). As such, it had to be compiled into the machine language for the Macintosh computer on which it ran. My students and I could write P88 programs and 'assemble' them into (a simulation of) a P88 machine-language program, which, in turn, was interpreted by Pascal as a certain Pascal program, which, in its own turn, was compiled into a Macintosh machine-language program, and executed. The levels are shown in the middle column of figure 1.

Suppose, now, that I write a program *in P88 assembly language* that takes two integers as input and returns their sum as output. When I cause this P88 program to be executed, a prompt appears on the screen, I input an integer, another prompt appears, another integer is input, and their sum is printed on the screen as output. Question: was this a P88 computation? Yes and no.

Yes: It was, because the *algorithm* that computationally caused the sum of the two inputs to be output was a P88 algorithm that used data structures and instructions from the P88 language. In other words, the two integers that were the input to the P88 program were input *to that program*, and their sum was output *by that program* (this is the top row of figure 1).

No: It was not a P88 computation, because the P88 algorithm was executed by having a *Pascal* program perform *Pascal* computations that used data structures and instructions from the Pascal language. So, was it, then, a *Pascal* computation? (This is the third row of figure 1.) Well, in some sense not really, because the Pascal algorithm was executed by having a Macintosh machine-language program perform *Macintosh machine-language* computations that used data structures and instructions from the Macintosh machine language. (Curiously, these are more like the data structures and instructions from P88 than from Pascal, but they were not executing or simulating P88 instructions or using P88 data directly.) At bottom, then, only a Macintosh machine-language program was really being executed and really computing the sum of two integers. (This is the bottom row of figure 1.) In other

(apparent?) input (e.g. 2 integers a, b)	$\Rightarrow$	P88 program $\downarrow$ P88 machine-language program	$\rightsquigarrow$	(apparent?) output (e.g. $a + b$)
		$\downarrow$		
(apparent?) input (a, b)	$\Rightarrow$	Pascal program $\downarrow$	$\rightsquigarrow$	(apparent?) output
(actual?) input (a, b)	$\Rightarrow$	Macintosh machine-language program	$\rightsquigarrow$	(actual?) output $(a + b)$

Figure 1. Hierarchy of virtual-machine levels. ($A \rightarrow B$ means: A is implemented in B; $A \Rightarrow B$ means: A is input to B; $A \rightsquigarrow B$ means: A outputs B.)

words, the two integers that apparently were the input to the P88 program were actually input to the Macintosh machine-language program, and their sum was actually output by *that* program.

Yet I can, and do, use the P88 Assembly-Language Simulator to compute the sum of two integers. In other words, a computation by the P88 program was simulated by a Macintosh machine-language computation, but there *was* a computation nonetheless. The simulated computation *was* a computation. Moreover, it was a computation in two senses (one could say that there were two (simultaneous) computations (with the same input and output)): ignoring *how* the P88 program was implemented, a P88 computation did yield the sum of the two inputs. And, ignoring the fact that it was simulating a Pascal simulation of a P88 program, the Macintosh machine-language computation also yielded the sum of the two inputs. It is important, I think, to note that the Macintosh machine-language program did so in a roundabout way: it was *not* the simplest possible Macintosh program to output the sum of two inputs, because it did that *not* simply by performing an addition, but by doing a number of other ('bookkeeping') operations that simulated a Pascal program simulating a P88 program.

In an early 'For Better or For Worse' cartoon, Elizabeth (a child of about 5 or 6) is sitting at her school desk, looking at her fingers, and thinking 'Umm ... two an' seven arrrre ...'. Her teacher yells, 'Fingers!! I see fingers! No, Elizabeth. When we do our arithmetic, we don't use our fingers, we do our work in our heads'. In the last panel, Elizabeth is *imagining* her fingers, and thinking, 'Two an' seven arre ...'. Her mental 'fingers' are an implementation (a semantic correlate) of actual fingers, and can serve (some of) the same purposes. Elizabeth is not using her actual fingers, and she *is* doing the work in her head—by using 'head-fingers'. Imagination and mental imagery can serve as a substitute for actual experience—one can solve problems by manipulating either the actual objects or models of them.[†]

Let us consider some other examples of the breakdown of segregation, all in informational contexts: a photograph of a map (as, for example, one that appeared in an ad for New York University, in *The New York Times* (20 August 1991: D5)) can be used to find out where some city is, even though that wasn't the purpose or intent of the map in the ad (cf. Rapaport 1995: section 2.5.4). Copying information (*sic*) from a book (by Xerox or by hand) and then using that copied information (those copied sentences) rather than the original source is done all the time. We do not think twice about it or say that it is not 'really' information.

Why *is* it information? Because of its syntax and the reader's interpretation of it. But it *carries* the information whether or not a reader interprets it. There is a possible problem: the information carried could be differently interpreted by another reader. But what is invariant is the syntactic structure. In any case, multiple interpretations are all equally good interpretations. Ah, but *is* the syntactic structure invariant? Examples such as the string 'NOWHERE' (which could be analysed as 'NOW HERE' or as 'NO WHERE') or weakly equivalent grammars (i.e. ones that parse the

[†]In a 'Hi & Lois' cartoon, Trixie (a baby) sits in a chair and thinks, 'I never knew I could do this! I just imagined that I was skiing like crazy and came to a cliff and sailed over and parachuted down to a jungle full of elephants! Wow! What fun! As life goes on my imagination is going to save me a lot of wear and tear'.

same sentences but assign them different structures) suggest that, the larger the context, the more aid there is in determining the syntactic structure.

What about computer simulations of minds? 'The difference between a symbolic airplane simulator and a symbolic intelligence simulator is that the former models a physical system through the intermediary representation of parameter values, while the latter models behavior by behaving' (Johan Lammens, personal communication, 17 August 1990). Here's the insight: *we* do not interact with simulated hurricanes[†], so they don't get *us* wet. (As we have seen, they *do* get simulated people simulatedly wet—that is where the 'internal representations' come in.) But we *do* interact with simulated minds. Now, how is that possible? *Is* it possible?

3.4. *Interacting with implementations*

Do we thus interact, or do we only seem to? Having a conversation on some topic with a Turing-Tested simulated mind *is* having a conversation on that topic and *not* merely simulating having the conversation. For the latter is what you would do in a play.[‡] 'When you get out of a TT session, something has changed: you have talked to a system about something, and most likely that has affected some of your own thoughts and beliefs' (Johan Lammens, personal communication, 17 August 1990). Of course, being in a play can do that, too, just as *reading* the play could. But in the Turing Test case, it is a dynamic, changing conversation.

So, how is the interaction possible? Because both systems deal with information, albeit implemented differently. But the implementations are 'transparent' (as in the game of chess played with Staunton pieces; cf. section 2.2). Jahren (1990: 315) claims 'that ''mentality'' equals human mentality'. He asserts that I 'objected that ... [this] would be like saying that flying is only real when it is implemented on [sic] birds' (Jahren 1990: 326n1). Jahren replied that I 'rel[y] on a metaphysical equivalence between natural phenomena and computational simulations that ... [is] rather bizarre' (Jahren 1990: 326n1). By 'metaphysical equivalence', Jahren apparently means the ability of a 'computational simulation ...[to] produce the physical effects that natural phenomena do', but he is 'unwilling to grant that elevated metaphysical status to what is ...only a numerical computation in a machine' (Jahren 1990: 326n1). I concede that computer simulations (i.e. implementations) of hurricanes are not thus 'metaphysically equivalent' to real ones (I would prefer to say that they are 'causally independent'). But I am claiming that there is an exception in cases of information (or intentional phenomena, more generally): information *can* flow across implementations. A mere 'numerical computation' cannot get me wet, but it can give me information (and so can a 'mere' neural computation—*you* can give me information, too).

[†]As Debra Burhans (personal communication) noted, sometimes, however, we *do*, as when we learn information about the behaviour of a hurricane from a computer simulation of it. But this is consistent with my point about being able to interact with informational simulations.

[‡]Except, perhaps, if the play is a quasi-improvisational 'virtual drama'; see Anstey *et al.* (2003) and Shapiro *et al.* (2005).

One must, of course, be careful to distinguish—if possible—between information that only concerns one world (either the simulated world or else the real world) and information that can transcend the boundary. Smith (1985) cites an example of a training tape that was interpreted to be a real Soviet attack. Another example would be a historical novel that is thought to be a historical text. The very fact that this can happen shows that some 'simulations' are indistinguishable from the 'real thing'. Note that novelistic language *can* be distinguished from reportorial language (Galbraith 1995: 33–35).[†] The point, however, is that it does not have to be. They need to be indistinguishable by whatever system needs to deal with both of them. And they need to be indistinguishable at the representational level, or at the level of Fregean *Sinne* or Meinongian objects, not at the level of Fregean *Bedeutungen* or of actual objects: in the one case, there is a real-world referent (or 'Sein-correlate', Rapaport 1978), but not in the other. At the level of *Sinn* or Meinongian object or mental representation, all things are on a par. If we cannot determine that they differ referentially, the default assumption should be that they do not. And even if we *can* determine it, it might not matter; i.e. 'ideal' elements (in Hilbert's sense; see section 2.3.3 above) are treated no differently from 'normal' elements. (For more discussion of this, see Rapaport 1991, Rapaport and Shapiro 1995.)

So there is a difference between the digestion and hurricane cases on the one hand and the natural-language and mind-brain cases on the other: 'brains, unlike stomachs, are information-processors. And if one information processor were made to simulate another information processor, it is hard to see how one and not the other could be said to think' (Johnson 1990: 46). That is, the difference is that it *is* the same stuff involved in the brain and computer cases: 'Simulated thoughts and real thoughts are made of the same stuff: information' (Johnson 1990: 46). Well—not quite: Information is abstract; 'simulated' and 'real' thoughts are different *implementations* of the same *Abstraction*. (One might, however, want to say that they deal with the Abstraction *directly*, via 'transparent' media.)

My claim, then, is that simulated cognition (or mentality, or intelligence) *is* cognition (or mentality, or intelligence). Recall the mental imagery debate, in which Kosslyn (1981, Kosslyn *et al.* 1981) argues that one really scans a mental (i.e. simulated) image, whereas Pylyshyn (1981) argues that one *pretends* to scan (*simulates* scanning) a real image. Or compare Searle's claim (1979) that in fictional language, one *pretends* to assert (one *simulates* asserting) rather than *really* asserting a pretended (or simulated) utterance—an utterance in a pretend-world.[‡] In both of these cases, I side with the 'really' people: Rather than saying that a computer *simulates* understanding something real, I would say that it *really* understands something simulated—and that in many cases, the simulated thing that it really understands is itself the real thing (internally represented).[§]

[†]At least sometimes. Debra Burhans (personal communication) notes that recent unfortunate developments in journalism have tried to blur the distinction!

[‡]Note the deictic shift; cf. Segal (1995), Galbraith (1995).

[§]Webb (1991: 247) argues 'that it is possible for a simulation to be a replication if the device used can not only represent but also instantiate the same capacities as the system'. This seems congenial to my claims.

4. From multiple realizability to panpsychism

Given the Principle of Multiple Realizability—the apparently obvious claim that there can be more than one implementation of an Abstraction, more than one model of a theory—an argument can be constructed for a variety of panpsychism (the view that everything is a mind). The argument, in its bare outlines, is this:

1. There is multiple realizability (of computational processes).
2. ∴ There is universal realizability (of computational processes) (either by an argument of Searle's or by an argument of Smith's).
3. ∴ Anything can be a model of anything (else) (from (2)).
4. ∴ Anything can be a model of a mind (from (3) by universal instantiation, or by an argument of Randall R. Dipert's).
5. ∴ Anything can be a mind (by the argument of § 3 that *models* of minds *are* minds).

Let us begin with Searle's argument from (1) to (2).

4.1. *Multiple realizability implies universal realizability*

In 'Is the Brain a Digital Computer?' (1990), Searle is concerned with the multiple realizability of computational processes. Hence, on the assumption that mental processes are computational, he is concerned with the multiple realizability of mental processes. According to Searle, a 'disastrous' consequence of multiple realizability is that it

> would seem to imply universal realizability. If computation is defined in terms of the assignment of syntax, then everything would be a digital computer, because any object whatever could have syntactical ascriptions made to it. You could describe anything in terms of 0's and 1's. (Searle 1990: 26.)

What evidence does Searle have for this claim that any (physical) object can be described computationally? And why is it disastrous? The latter question is easier to answer: According to Searle, universal realizability doesn't tell us what is *special* about the brain as opposed to other, less interesting, computational systems, such as the 'stomach, liver, heart, solar system, and the state of Kansas' (1990: 26). But if the ultimate panpsychic conclusion holds, then this merely begs the question; we will come back to this (section 4.3).

Searle claims that 'For any object there is some description of that object such that under that description the object is a digital computer' (1990: 26). This seems too strong. For one thing, there are certainly things that are non-computational in the strong sense of the Church-Turing Thesis. For another, merely assigning 0s and 1s to give an encoding (of, say, the atomic structure) of a physical object does not make it computational. To be computational, analogues are needed of Turing-machine instructions, control structures, states, the input–output tape, etc. At the very least, to be computational, an item must compute some function. So, if my pen, say, is a digital computer, what function does it compute? Well, I suppose it could be argued that it computes the constant function (or perhaps the identity function, or perhaps it loops forever—i.e. is undefined on all input). But that is trivial. On the other hand, consider the string of 0s and 1s that, according to Searle, encodes my pen. That's the Gödel number of some program (no doubt a trivial one, but

who knows?). Does that make my pen an *implementation* (a model) of that program? No; it is an interesting correspondence, but not an implementation, because the interpretations of the 0s and 1s for the description of the pen are not those required for the program. So, Searle's argument does not support universal realization.

Smith, however, has made similar observations. He describes a 'computer [that] ... calculates oriental [*sic*] trajectories', which is, in fact, a car that drives west[†], and he asks why this *isn't* a computer (1982: 2). He notes that it does share at least one important feature with computers, which is close to Searle's claim: The 'oriental'-trajectory calculator *is* equivalent to a Turing machine, but that is not sufficient to make it a computer. But *why* is it equivalent to a Turing machine? Perhaps because it is input–output equivalent to a Turing machine that calculates 'oriental' trajectories? But mere input–output equivalence is not sufficient: as I once argued in this journal (Rapaport 1998), to say that a *function* is comput*able* is merely to say that there is a Turing machine that computes it (i.e. that has the same input–output behaviour), but this does not require that a device with that input–output behaviour is *itself* a computer (i.e. that it *computes* its output from its input)—it could be a mysterious oracle. Now, the case of the car is not quite the case of such an oracle. The car, after all, has parts whose function and behaviour contribute to the car's overall behaviour (its output, if you will).

So why does Smith say it is not a computer? Because there are no *symbols* that 'act as causal ingredients in producing an overall behavior' (1982: 2)—symbols in the sense of markers of a formal syntactic system:

> In describing how a car works... the story is not computational, because the salient explanations are given in terms of mechanics—forces and torques ... and so forth. These are not *interpreted* notions; we don't posit a semantical interpretation function in order to make sense of the car's suspension. (Smith 1982: 3)

I dispute that. First, there is a mapping between the physical parts and actions of the car and terms from physics (i.e. physical theory). Second, there is a mapping between (at least) terms from physics and my concepts. So we *do* interpret the car's parts and actions.

Here is Smith's response to the claim

> ...that we 'interpret' steering wheels as mechanisms for getting cars to go around corners—... this is a broader notion of 'interpret' than I intend. I mean to refer to something like the relationship that holds between pieces of language, and situations in the world that those pieces of language are about. (Smith 1982: 4)

But that distinction is one that can not be drawn—*both* are interpretations (Rapaport 1995).

The threat of universal (or near-universal) realizability is expressed by Smith thus:

> ... the 'received' theory of computation—the theory of effective computability that traffics in recursive functions, Turing machines, Church's Thesis, and the rest ... does not intrinsically identify the class of artefacts that computer science studies. ... [I]t is too broad, in that it includes far more devices within its scope (like chairs and Rubik cubes) than present experts

[†]Shouldn't that be *east*? Perhaps 'oriental' is a typographical error for 'orbital' or 'orientational' ('oriental' as in 'pertaining to orienting oneself'?).

> would call computers. The problem stems from the fact that Turing equivalence (i.e. computing the same function) is a *weak, behavioral* metric, and we are interested in a theory that enables us to define *strong, constitutional* concepts. (Smith 1982: 5)

I am willing to accept Rubik's cube, however. The difficulty is that 'the class of artefacts that computer science studies' is an *intended interpretation* that the theory of computation just won't let us get our hands on, any more (but, equally, no less) than Peano's axioms let us get our hands on the natural numbers. If chairs are included (as Dipert argues; see below), so be it. Even if we strengthen the theory to talk about *algorithmic* equivalence, and not mere input–output equivalence, we will still get multiple, hence unwanted—or, better, *unexpected*—realizations.

4.2. *Everything models anything*

If there is universal realizability, then everything can be an implementation (or realization) of an arbitrary Abstraction. It follows by universal generalization that everything can be an implementation (or realization, or model) of anything (else).[†]

4.3. *Everything models mentality*

Clearly, if anything can be a model of anything (else), then anything can be a model of a mind. An argument explicitly for this consequent was offered by Dipert (1990).[‡]

Dipert begins by reminding us that David Hilbert's philosophy of formalism took numbers in 'purely structural, formal terms … [C]hairs, beer mugs, or whatever could just as well represent/exemplify numbers (under the right interpretation) as do numerals or our thoughts of numbers' (1990: 6).[§] Similarly, 'programs, together with their hardware implementation … may not look much like more usual [i.e. biological] embodiments of minds' (1990: 6), but they could be, in the same way that chairs can be embodiments of numbers. However—or so Dipert observes—not even adherents of so-called 'strong AI' would take *chairs* as embodying minds, because 'brains are much more complex than chairs, and so chairs and tables lack some of the structural features of mental properties' (1990: 6). However, I am an adherent of 'strong AI' and am as willing to accept the claim about chairs as I am about the standard water-pipes-and-valves model, which I am quite willing to do. Dipert (along with Searle and Smith, evidently) thinks this is problematic. Here's the argument that shows why:

> (**P1**) Ordinary, middle-sized objects at room temperature (let us call them OMSORTs, for short)—e.g. chairs, coffee mugs, baseballs (and, presumably, brains)—are highly complex, dynamic entities (1990: 7–8).

[†]Alternatively, step (3) of the argument for panpsychism follows from the assumption that everything shares at least one property (and perhaps infinitely many) with everything (else). Cf. the discussion of Wartofsky (1979) in Rapaport (1995: section 2.5.3).

[‡]Dipert himself is sympathetic to the conclusion, even though he is playing devil's advocate in criticizing it; cf. Dipert (1990: 20n16).

[§]cf. Hilbert's *Gesammelte Abhandlungen* (vol. 3: 403) as cited in Coffa (1991: 135).

(**P2**) Suppose there is a good cognitive science theory T of the sufficient conditions for 'cognition and other mental processes' (1990: 8).
(**P3**) Suppose there is an AI system C that implements T (1990: 9).
(**C1**) ∴ Since T spells out the sufficient conditions for cognition, C has cognition (1990: 9).
(**P4**) For every OMSORT O, there is an interpretation such that O exemplifies T (although we might not be able to exhibit the interpretation that does the job) (1990: 9, 11).
(**C2**) ∴ Since T spells out the sufficient conditions for cognition, O has cognition (1990: 9–10).

> These reflections should also make us resist our initial temptation to say that exemplifying some humanly-graspable ... set of properties [is] *sufficient* for having mental properties—unless we are willing to say, with Leibniz, that everything is a mind (1990: 11).

Thus,

(**P5**) Conclusion (C2) is absurd or uninteresting.
(**C3**) ∴ Conclusion (C1) is absurd or uninteresting.

Now, one difference between the arbitrary OMSORT O and the AI system C is that for C we *do* know what the interpretation function is: We can *understand* how and why C behaves as it does; we can interpret C's behaviour as a mind. We accept it as such (this is what we do in our everyday solution of the problem of other minds).

Note, too, that some OMSORTs that we *might* very well be willing to accept as implementations of minds—e.g. connectionist implementations that have not been 'properly treated' (Smolensky 1988)—are such that we might very well *not* understand them (what, for example, do the connection weights 'mean'?).

What's wrong with Leibniz's position? Mainly that if everything is a mind, then we cannot explain the difference between a *human* mind and a coffee mug. In this regard, we might be no more worse off than a topologist who, as the joke has it, cannot distinguish a doughnut from a coffee mug, since both are toruses. But to say that there is a way to view doughnuts and coffee mugs such that they are alike is *not* to say that there is no way in which they differ. Similarly, if computational cognitive science tells us that, from a certain perspective, brains and mugs are alike, that's not to say that, from some other perspective, they are not. We *can* explain the difference between a mind and a mug: The mug mind cannot communicate with us and therefore is irrelevant. That is, the mug *qua* mind cannot communicate; the mug *qua* coffee-holder is a perfectly functional device. Brains *can* communicate, but they cannot hold coffee—so we do not use them for that purpose. Not to be too macabre, we *could* use a brain as a paperweight, I suppose; a Martian (or a Black Cloud; Hoyle 1957, cf. Rapaport 2003) might do so, and might never realize that its paperweight implemented a mind, any more than we realize that a coffee mug might. As a further analogy, compare a high-level program (e.g. a program to compute greatest common divisors, or even an AI program) with a machine-language version in the same way we have been comparing brains with mugs. The latter might not *look* like the former, but under the right circumstances it might very well behave like the former.

What, then, is the import of Dipert's argument? Is it merely that, for any theory, there are infinitely many models, many of which are non-isomorphic[†] and many of which are not the intended model(s) and are such that we did not antecedently take them to be models? If so, so what? Sure, there *have* to be unintended models, and there is no way to pick out or mark or identify the intended ones; that's one of the main lessons of the theory-model relationship. What we learn from the existence of unintended models (assuming that we are completely satisfied with the *theory* of which they are models) is that some things have properties and features that we did not expect them to have.

Of course, Dipert's transcendental argument merely shows that it is *highly likely* that OMSORTs can be taken as (models of) minds, not that they *are*. Two highly complex things, just because of their high complexity, need not be models of each other. (Two highly complex patterns need not be matchable.)

On the other hand, Dipert's claim might be the weaker one that (it is highly likely that) there are *some* OMSORTs that model the cognitive theory *T* but that we would not antecedently have taken as an intended model. I think we can only bite the bullet on this. But perhaps it can be made palatable: suppose the OMSORT is a (particular) baseball. Imagine complex dynamic *processes* 'within' the baseball, presumably at the subatomic level, that model the mental processes. What, for instance, might correspond to perception? (Does the baseball 'see'?) Perhaps nothing so corresponds in the sense of external causes of internal processes, but there might be internal processes that, in a methodologically solipsistic fashion, correspond to (or model) perception. Or perhaps there *are* such external causes, but they need not be actual events as *we* characterize (or see) them. The world that the baseball-mind 'perceives' might (indeed, probably would) have different categories than the human mind or the AI mind (as Kant told us long ago; for a discussion of Kant from an AI perspective, see Kirsh 1991: 12; cf. Rapaport 2003, section 10).[‡]

5. Summary

In Rapaport (1999), I proposed that implementation is best understood as semantic interpretation (rather than as individuation, instantiation, reduction, or supervenience). It is a relationship between an Abstraction (a generalization of the notion of an abstract data type) and an implementing medium. This relationship can be found in the arts and language, as well as in the theory of abstract data types. In general, something is an implementation of an Abstraction in an implementing medium (perhaps as created by some cognitive agent for some purpose). In the present essay, I have investigated the mind–brain relationship as a case of implementation. Mind is an Abstraction that can be implemented in brains as well

[†]'[M]athematics involves a considerable variety of models. the same experience can be modeled mathematically in more than one way.... [M]athematical models are determined 'up to a canonical isomorphism.' Indeed, that is all that matters.... For many mathematical purposes though, mathematicians use axiomatic systems which have many nonisomorphic models' (Mac Lane 1981: 467).

[‡]If implementation is a 5-place relation as suggested on page 2, then perhaps an OMSORT such as a baseball does not implement a computer or a mind unless someone uses or interacts with it in that way. Cf. Giere (2004: section 4.1, 747–748).

as in computers. Implementations, however, have implementation-dependent details that give rise to qualia—implementation side-effects. Finally, an argument for a benign form of panpsychism can be developed from this viewpoint.

If Mind can be implemented in a computer, could a computer that implemented a natural-language-understanding program really understand language? I would say 'Yes'; Searle, famously, says 'No'. In a forthcoming essay, I re-examine his Chinese Room in light of the present conclusions.

Acknowledgments

This essay is adapted from an unpublished manuscript, Rapaport 1996. I am grateful especially to Debra Burhans and Albert Goldfain and to Carl Alphonse, Josephine Anstey, Frances L. Johnson, Erwin Segal, and other members of the SNePS Research Group for comments on earlier drafts.

References

D. Ackerman, "Penguins", *The New Yorker*, pp. 38–67, 10 July 1989.

S.A. Altmann, "The monkey and the fig: a Socratic dialogue on evolutionary themes", *American Scientist*, 77, pp. 256–263, 1989.

J. Anstey, D. Pape, S.C. Shapiro and V. Rao, "Virtual drama with intelligent agents", in *Hybrid reality: Art, technology and the human factor, proceedings of the 9th International Conference on Virtual Systems and MultiMedia (VSMM)*, (International Society on Virtual Systems and MultiMedia), H. Thwaites, Ed., 2003, pp. 521–528.

L. Apostel, "Towards the formal study of models in the non-formal sciences", in *The concept and the role of the model in mathematics and natural and social sciences*, H. Freudenthal, Ed., Dordrecht: D. Reidel, 1961, pp. 1–37.

A.J. Ayer, *The problem of knowledge*, Baltimore: Penguin, 1956.

D. Bickerton, "Mothering plus vocalization doesn't equal language", *Behavioral and Brain Sciences*, 27, pp. 504–505, 2004.

A. Biermann, *Great ideas in computer science: a gentle introduction*, Cambridge, MA: MIT Press, 1990.

A. Byrne, "Inverted qualia", in *The Stanford Encyclopedia of Philosophy* (*Summer* 2005 *Edition*), E.N. Zalta, Ed., 2005. Available online: http://plato.stanford.edu/archives/sum2005/entries/qualia-inverted/.

J.A. Coffa, *The Semantic tradition from Kant to Carnap: To the Vienna station*, Cambridge, UK: Cambridge University Press, 1991.

D. Cole, "Functionalism and inverted spectra", *Synthese*, 82, pp. 207–222, 1990.

E.K. Coughlin, "A professor champions distinct culture of deaf people", *Chronicle of Higher Education*, 2 October 1991: A5.

D.C. Dennett, "Why you can't make a computer that feels pain", reprinted in D.C. Dennett, *Brainstorms*, Montgomery, VT: Bradford Books, 1978, pp. 190–229.

E.W. Dijkstra, "The threats to computer science", *ACM 1984 South Central Regional Conference* (Austin, TX, 16–18 November 1984), EWD898. Available online: http://www.cs.utexas.edu/users/EWD/ewd08xx/EWD898.PDF

R.R. Dipert, "Complexity and models of minds: a simple, Hilbertian argument that strong AI is doomed", paper presented at the 5th Annual Computers and Philosophy Conference, Stanford University, 9 August 1990); page references are to an unpublished preprint.

G. Duhamel, *Les jumeaux de Vallangoujard*, [*The twins of Vallangoujard*], M.E. Storer, Ed., Boston: DC Heath, 1940, 1931.

U. Eco, "On truth. A fiction", in *Meaning and mental representations*, U. Eco, M. Santambrogio and P. Violi, Eds., Bloomington: Indiana University Press, 1988, pp. 41–59.

A. Eisenberg, "The virtual stomach (no, it's not a diet aid)", *The New York Times*, 31 October 2002, G4.

J.H. Fetzer, "Program verification: the very idea", *Communications of the ACM*, 31, pp. 1048–1063, 1988.

J.H. Fetzer, "Philosophical aspects of program verification", *Minds and Machines*, 1, pp. 197–216, 1991.

M. Galbraith, "Deictic shift theory and the poetics of involvement in narrative", in *Deixis in narrative: a cognitive science perspective*, J.F. Duchan, G.A. Bruder and L.E. Hewitt Eds., Hillsdale, NJ: Lawrence Erlbaum Associates, 1995, pp. 19–59.

D. Gentner, "The development of relational category knowledge", in *Building object categories in developmental time*, L. Gershkoff-Stowe and D.H. Rakison Eds., Mahwah, NJ: Lawrence Erlbaum Associates, 2005.

R.N. Giere, "How models are used to represent reality", *Philosophy of Science*, 71, pp. 742–752, 2004.

J.A. Goguen, J.W. Thatcher and E.G. Wagner, "An initial algebra approach to the specification, correctness, and implementation of abstract data types", in *Current trends in programming methodology*, Vol. IV: *Data structuring*, R.T. Yeh, Ed., Englewood Cliffs, NJ: Prentice-Hall, 1978, pp. 80–149.

J.J.E. Gracia, "Texts and their interpretation", *Review of Metaphysics*, 43, pp. 495–542, 1990.

J.V. Guttag, E. Horowitz and D.R. Musser, "The design of data type specifications", in *Current trends in programming methodology*, Vol. IV: *Data structuring*, R.T. Yeh, Ed., Englewood Cliffs, NJ: Prentice-Hall, 1978, pp. 60–79.

R. Harris, *Reading Saussure: A Critical Commentary on the Cours de Linguistique Générale*, La Salle, IL: Open Court, 1987.

J.P. Hayes, *Computer architecture and organization*, 2nd edn, New York: McGraw-Hill, 1988.

D. Hilbert, "On the infinite", 1925, trans. E. Putnam and G.J. Massey, reprinted in *Philosophy of mathematics: selected readings*, P. Benacerraf and H. Putnam Eds., Englewood Cliffs, NJ: Prentice-Hall, 1964, pp. 134–151.

D.R. Hofstadter, "A coffeehouse conversation on the Turing test", *Scientific American*, 15–36, May 1981; reprinted with Reflections (by D.C. Dennett) in *The mind's I: Fantasies and reflections on self and soul*, D.R. Hofstadter and D.C. Dennett, Eds., New York: Basic Books, 1981, pp. 68–95.

F. Hoyle, *The black cloud*, New York: Harper and Row, 1957.

H.O. Ismail and S.C. Shapiro, "Two problems with reasoning and acting in time", in *Principles of Knowledge Representation and Reasoning: Proceedings of the 7th International Conference (KR 2000)*, A.G. Cohn, F. Giunchiglia and B. Selman, Eds., San Francisco: Morgan Kaufmann, 2000, pp. 355–365.

N. Jahren, "Can semantics be syntactic?", *Synthese*, 82, pp. 309–328, 1990.

N. Jardine, "Model-theoretic semantics and natural language", in *Formal semantics of natural language*, E.L. Keenan, Ed., Cambridge: Cambridge University Press, 1975, pp. 219–240.

R.C. Jennings, "Translation, interpretation and understanding", paper read at the American Philosophical Association Eastern Division, Washington, DC; abstract, in *Proceedings and Addresses of the American Philosophical Association*, 59, pp. 345–346, 1985.

G. Johnson, "New mind, no clothes", *The Sciences*, pp. 45–49, July/August 1990.

D. Kirsh, "Foundations of AI: The big issues", *Artificial Intelligence*, 47, pp. 3–30, 1991.

S.M. Kosslyn, "The medium and the message in mental imagery: A theory", in *Imagery*, N. Block, Ed., Cambridge, MA: MIT Press, 1981, pp. 207–244.

S.M. Kosslyn, S. Pinker, G.E. Smith and S.P. Schwartz, "On the demystification of mental imagery", in *Imagery*, N. Block, Ed., Cambridge, MA: MIT Press, 1981, pp. 121–150.

G. Lakoff and M. Johnson, *Metaphors we live by*, Chicago: University of Chicago Press, 1980.

J. Leiber, *Beyond rejection*, New York: Ballantine Books, 1980.

S. Mac Lane, "Mathematical models: A sketch for the philosophy of mathematics", *American Mathematical Monthly*, 88, pp. 462–472, 1981.

D. McDermott, *Mind and mechanism*, Cambridge, MA: MIT Press, 2001.

M. Minsky, Posting to comp.ai.philosophy bulletin board, 21 December 1991. Available online http://groups-beta.google.com/group/comp.ai.philosophy/msg/bf1b1554b22e1a48

D.A. Nelson, "Deductive program verification (a practitioner's commentary)", *Minds and Machines*, 2, pp. 283–307, 1992.

D.A. Nelson, Review of R.S. Boyer and J S. Moore, *A computational logic handbook*, and J S. Moore, "Special issue on system verification", *Minds and Machines*, 4, pp. 93–101, 1994.

D. Parnas, "A technique for software module specification with examples", *Communications of the ACM*, 15, pp. 330–336, 1972.

A.L. Pincus, "The art of transcription sheds new light on old work", *The New York Times*, Arts and Leisure (Sect. 2), 23 September 1990.

R. Posner, "Origins and development of contemporary syntactics", *Languages of Design*, 1, pp. 37–50, 1992.

Z. Pylyshyn, "The imagery debate: Analog media versus tacit knowledge", in *Imagery*, N. Block, Ed., Cambridge, MA: MIT Press, 1981, pp. 151–206.

W.J. Rapaport, "Meinongian theories and a Russellian paradox", *Noas*, 12, pp. 153–180, 1979; errata, *Noas*, 13, 1979, 125, 1979.

W.J. Rapaport, "How to make the world fit our language: An essay in Meinongian semantics", *Grazer Philosophische Studien*, 14, pp. 1–21, 1981.

W.J. Rapaport, "Unsolvable problems and philosophical progress", *American Philosophical Quarterly*, 19, pp. 289–298, 1982.
W.J. Rapaport, "Machine understanding and data abstraction in Searle's Chinese room", in *Proceedings of the 7th Annual Conference of the Cognitive Science Society* (*University of California at Irvine*), Hillsdale, NJ: Lawrence Erlbaum Associates, 1985, pp. 341–345.
W.J. Rapaport, "Searle's experiments with thought", *Philosophy of Science*, 53, pp. 271–279, 1986.
W.J. Rapaport, "Syntactic semantics: Foundations of computational natural-language understanding", in *Aspects of artificial intelligence*, J.H. Fetzer, Ed., Dordrecht: Kluwer Academic Publishers, 1988a, pp. 81–131.
W.J. Rapaport, "To think or not to think", *Noas*, 22, pp. 585–609, 1988b.
W.J. Rapaport, "Predication, fiction, and artificial intelligence", *Topoi*, 10, pp. 79–111, 1991.
W.J. Rapaport, "Because mere calculating isn't thinking: Comments on Hauser's 'Why isn't my pocket calculator a thinking thing?' ", *Minds and Machines*, 3, pp. 11–20, 1993.
W.J. Rapaport, "Understanding understanding: Syntactic semantics and computational cognition", in *Philosophical perspectives, Vol. 9: AI, connectionism, and philosophical psychology*, J.E. Tomberlin Ed., Atascadero, CA: Ridgeview, pp. 49–88, 1995.
W.J. Rapaport, *Understanding Understanding: Semantics, Computation and Cognition*, Technical Report 96–26, Buffalo: SUNY Buffalo Department of Computer Science.
W.J. Rapaport, "How minds can be computational systems", *Journal of Experimental and Theoretical Artificial Intelligence*, 10, pp. 403–419, 1998.
W.J. Rapaport, "Implementation is semantic interpretation", *The Monist*, 82, pp. 109–130, 1999.
W.J. Rapaport, "How to pass a Turing test: Syntactic semantics, natural-language understanding, and first-person cognition", *Journal of Logic, Language, and Information*, 9(4), pp. 467–490, 2000; reprinted in *The Turing test: The elusive standard of artificial intelligence*, J.H. Moor, Ed., Dordrecht: Kluwer, 2003, pp. 161–184.
W.J. Rapaport, "Holism, conceptual-role semantics, and syntactic semantics", *Minds and Machines*, 12(1), 3–59, 2002.
W.J. Rapaport, "What did you mean by that? Misunderstanding, negotiation, and syntactic semantics", *Minds and Machines*, 13(3), pp. 397–427, 2003.
W.J. Rapaport, "In defense of contextual vocabulary acquisition: How to do things with words in context", in *Proceedings of the 5th International and Interdisciplinary Conference on Modeling and Using Context (Context-05)*, A. Dey, B. Kokinov, D. Leake and R. Turner, Eds., Berlin: Springer-Verlag Lecture Notes in Artificial Intelligence 3554, 2005, pp. 396–409.
W.J. Rapaport and S.C. Shapiro, "Cognition and fiction", in *Deixis in narrative: A cognitive science perspective*, J.F. Duchan, G.A. Bruder and L.E. Hewitt, Eds., Hillsdale, NJ: Lawrence Erlbaum Associates, 1995, pp. 107–128.
W.J. Rapaport, S.C. Shapiro and J.M. Wiebe, "Quasi-indexicals and knowledge reports", *Cognitive Science*, 21, pp. 63–107, 1997.
E. Rosch, "Principles of categorization", in *Cognition and categorization*, E. Rosch and B.B. Lloyd, Eds., Hillsdale, NJ: Lawrence Erlbaum, 1978, pp. 27–48.
C. Rosen, Reply to letter, *New York Review of Books*, 14 February 1991, p. 50.
A. Rosenblueth and N. Wiener, "The role of models in science", *Philosophy of Science*, 12, pp. 316–321, 1945.
J.F. Santore and S.C. Shapiro, "Identifying perceptually indistinguishable objects", in *Anchoring symbols to sensor data, Papers from the AAAI Workshop, Technical Report WS-04-03*, S. Coradeschi and A. Saffiotti, Eds., Menlo Park, CA: AAAI Press, 2004, 1–9.
H.C. Schonberg, "Some chessmen don't make a move", *New York Times*, 15 April 1990, Sect. 2, pp. 38–39.
J.R. Searle, "The logical status of fictional discourse", in J.R. Searle, *Expression and meaning*, Cambridge: Cambridge University Press, 1979, pp. 58–75.
J.R. Searle, "Is the brain a digital computer?", *Proceedings and Addresses of the American Philosophical Association*, 64(3), pp. 21–37, 1990.
E.M. Segal, "Narrative comprehension and the role of deictic shift theory", in *Deixis in narrative: A cognitive science perspective*, J.F. Duchan, G.A. Bruder and L.E. Hewitt, Eds., Hillsdale, NJ: Lawrence Erlbaum Associates, 1995, pp. 3–17.
W. Sellars, "Some reflections on language games", in *Science, perception and reality*, London: Routledge and Kegan Paul, 1963, pp. 321–358.
S.C. Shapiro, "The CASSIE projects: An approach to natural language competence", in *EPIA 89: 4th Portuguese Conference on Artificial Intelligence, Proceedings (Lisbon)*, J.P. Martins and E.M. Morgado, Eds., Berlin: Springer-Verlag Lecture Notes in Artificial Intelligence 390, 1989, pp. 362–380.
S.C. Shapiro, "Embodied Cassie", *Cognitive robotics: Papers from the 1998 AAAI Fall Symposium, Technical Report FS-98-02*, Menlo Park, CA: AAAI Press, 1998, pp. 136–143.

S.C. Shapiro, J. Anstey, D.E. Pape, Devdas T. Nayak, M. Kandefer and O. Telhan, "The trial the trail, Act 3: A virtual reality drama using intelligent agents", in *Proceedings of the 1st Annual Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE-05)*, Menlo Park, CA: AAAI Press, 2005.

S.C. Shapiro and H.O. Ismail, "Anchoring in a grounded layered architecture with integrated reasoning", *Robotics and Autonomous Systems*, 43, pp. 97–108, 2003.

S.C. Shapiro and W.J. Rapaport, "SNePS considered as a fully intensional propositional semantic network", in *The knowledge frontier: Essays in the representation of knowledge*, N. Cercone and G. McCalla, Eds., New York: Springer-Verlag, 1987, 262–315.

S.C. Shapiro and W.J. Rapaport, "Models and minds: Knowledge representation for natural-language competence", in *Philosophy and AI: Essays at the interface*, R. Cummins and J. Pollock, Eds., Cambridge, MA: MIT Press, 1991, pp. 215–259.

S.C. Shapiro and W.J. Rapaport, "The SNePS family", *Computers and Mathematics with Applications*, 23, pp. 243–275, 1992.

S.C. Shapiro and W.J. Rapaport, "An introduction to a computational reader of narratives", in *Deixis in narrative: A cognitive science perspective*, J.F. Duchan, G.A. Bruder and L.E. Hewitt, Eds., Hillsdale, NJ: Lawrence Erlbaum Associates, 1995, pp. 79–105.

S.C. Shapiro, H.O. Ismail and J.F. Santore, "Our dinner with Cassie", *Working Notes for the AAAI 2000 Spring Symposium on Natural Dialogues with Practical Robotic Devices*, Menlo Park, CA: AAAI Press, 2000, pp. 57–61.

B.C. Smith, "Semantic attribution and the formality condition", paper presented at the 8th Annual Meeting of the Society for Philosophy and Psychology, University of Western Ontario, 15 May 1982, page references are to the "Second Draft" preprint.

B.C. Smith, "Limits of correctness in computers", *Technical Report CSLI-85-36*, Stanford, CA: Center for the Study of Language and Information, 1985, published in C. Dunlop and R. Kling, Eds., *Computerization and Controversy*, San Diego: Academic Press, 1991, pp. 632–646.

B.C. Smith, "The correspondence continuum", *Report CSLI-87-71*, Stanford, CA: Centre for the Study of Language and Information.

P. Smolensky, "The proper treatment of connectionism", *Behavioral and Brain Sciences*, 11, pp. 1–74, 1988.

G.M. Stratton, "Vision without the inversion of the retinal image", *Psychological Review*, 4, pp. 341–360, 463–481, 1897.

A.M. Tenenbaum and M.J. Augenstein, *Data structures using Pascal*, Englewood Cliffs, NJ: Prentice-Hall, 1981.

M.W. Wartofsky, "The model muddle: Proposals for an immodest realism", in *Models: Representation and the scientific understanding*, Dordrecht, Holland: D. Reidel, pp. 1–11, 1966.

M.W. Wartofsky, "Introduction", in *Models: Representation and the scientific understanding*, Dordrecht, Holland: D. Reidel, pp. xiii–xxvi, 1979.

B.H. Webb, "Do computer simulations really cognize?", *Journal of Experimental and Theoretical Artificial Intelligence*, 3, pp. 247–254, 1991.