CHAPTER #

WHY COMPUTERS CAN'T FEEL PAIN

Mark Bishop

Department of Computing

Goldsmiths College, University of London, UK

m.bishop@gold.ac.uk

1. Background

In Science and Science Fiction the hope is periodically reignited that a computer system will one day be conscious in virtue of its execution of an appropriate program; indeed the UK funding body EPSRC recently awarded an 'Adventure Fund' grant of around £500,000 to a team of 'Roboteers and Psychologists' at Essex and Bristol universities[1], with a goal of instantiating machine consciousness - in a 'humanoid-like' robot called Cronos - through

1 The project, '*Machine consciousness through internal modelling*', is funded by the EPSRC Adventure Fund. The total funding is £493,000 split between the departments of Computer Science, University of Essex and the Department of Psychology, University of Bristol. The project is led by Professor Owen Holland, University of Essex.

appropriate computational 'internal modelling'. What I will outline below is a brief reductio style argument that either suggests such optimism is misplaced or that panpsychism - the belief that the physical universe is composed of elements each of which is conscious - is true.

First though, it is helpful to outline exactly what is meant by the term consciousness in the context of this paper. By phenomenal consciousness I refer to the first person, subjective phenomenal states - sensory tickles, pains, visual experiences and so on. Current research into perception and neuro-physiology suggests that physically identical brains will instantiate identical phenomenal states; however, pace Maudlin (1989), "if a causal theory of reference is correct, a molecule-for-molecule identical replica of my brain, if just brought into existence, may not be capable of entertaining the proposition that ice is made of water. Still our best guess is that such a brain would support identical states of consciousness to mine, identical phenomenal states." As Maudlin (ibid) observes, this thesis is not analytic; however something like it underpins computational theories of mind; for computational structure supervenes on physical structure – physically identical brains must be computationally identical. Hence Maudlin (ibid) formulates the 'supervenience thesis', "two physical systems engaged in precisely the same physical activity through a time will support precisely the same modes of consciousness (if any) through that time."

The core argument I outline in this paper derives from ideas originally outlined by Hilary Putnam (1988), Tim Maudlin (1989) and John Searle (1990) and subsequently criticised by David Chalmers (1996), Colin Klein (2005) and Ron Chrisley (2006) amongst others[2]. In what follows, instead of seeking to justify Putnam's claim that "every open system implements every Finite State Automaton (FSA)", and hence that psychological states of the brain cannot be functional states of a computer, I will seek to establish the weaker result that, over a finite time window, every open physical system implements the trace of a Discrete State Machine Q, as it executes its control program on fixed, specified input (x). That this result leads to panpsychism is clear as, equating Q (x) to a specific computational system that is claimed to instantiate phenomenal states as it executes, and following Putnam's procedure, identical computational (and ex-hypothesi phenomenal) states can be found in every open physical system.

The route-map for this endeavour is as follows. In the first part of the paper I introduce Discrete State Machines, DSMs, and show how, with input to them defined, their behaviour is described by a simple unbranching sequence of state transitions analogous to that of an inputless DSM. Then I review Putnam's 1988 argument that purports to show how every open physical system implements every inputless FSA. This argument is subsequently applied to a robotic system that is claimed to instantiate genuine phenomenal states as it operates. The paper concludes with a brief discussion of some objections raised following presentation of these ideas at the "Computers and Philosophy" conference, Leval 2006.

2. Discrete State Machines

In his 1950 paper, 'Computing Machinery and Intelligence', Turing defined Discrete State Machines, DSMs, as "machines that move in sudden jumps or clicks from one quite definite state to another", and explained that modern digital computers fall within the class of them. An example DSM from Turing is one that cycles through three computational states (Q_1, Q_2 & Q_3) at discrete clock clicks. Turing demonstrated that such a device, which cycles through a linear series of state transitions 'like clockwork', may be implemented by a simple wheel-machine that revolves through 120^0 intervals.

By labelling the three discrete positions of the wheel $\{W_A, W_B, W_C\}$ we can map computational states of the DSM (Q_1, Q_2, Q_3) to the physical positions of the wheel $\{W_A, W_B, W_C\}$, such that, for example, ($W_A \Rightarrow Q_1$; $W_B \Rightarrow Q_2$; $W_C \Rightarrow Q_3$). Clearly this mapping is observer relative: position W_A of the wheel could equally map to computational states Q_2 or Q_3 and, with other states appropriately assigned, the machine's state transition sequence (and hence its function) would remain unchanged. It is central to the argument to be developed in this paper that **all** computational states are observer relative in this fashion; they are not intrinsic to the physics of the system – that is, their determination always involves an 'observer-specified' function that maps from physical system state onto computational state.

2 Cf. Minds and Machines, 4: 4, 'What is Computation?', November 1994.

In general, we can generate the behaviour of any K-state (inputless) DSM, (f (Q) ⇒ Q'), by a K-state wheel-machine (e.g. a digital counter), and a function that maps each wheel/counter state W_n/C_n to each computational state Q_n as required.

In addition, Turing's machine may be stopped by the application of a brake and whenever it enters a specific computational state a lamp will come on. Input to the machine is thus the state of the brake, (I = {ON | OFF}), and its output, (Z), the state of the lamp. Hence the operation of a DSM with input is described by a series of 'contingent branching state transitions', which map from current state to next state, f (Q, I) => Q' and define the machines output - in the Moore form - as f (Q') ⇒ Z.

However, (over a finite time interval), defining the input to the DSM entails that such 'contingent behaviour' reverts to 'clockwork', (f (Q) => Q'). E.g. If Turing's DSM starts in Q_1 and the brake is OFF for two clicks, its behaviour, (execution trace), is fully described by the sequence of state transitions, (Q_1, Q_2, Q_3); conversely if Turing's DSM starts in Q_1 and the brake is ON for two clicks, its behaviour - execution trace - is described by the sequence of state transitions, (Q_1, Q_1, Q_1).

Hence, over a finite time window, if the input to a DSM is defined, we can map from each wheel/counter state W_n/ C_n to each computational state Q_n, as required. In Bishop (2002) I demonstrated, pace Putnam, how to map any computational state sequence with fixed [defined] input onto the [non-repeating] natural state sequence generated by any open physical system.

3. Putnam's mapping

Discussed in a brief appendix to Hilary Putnam's 1988 book *Representation and Reality* is a short argument that endeavours to prove that every open physical system is a realisation of every abstract Finite State Automaton and hence that functionalism fails to provide an adequate foundation for the study of the mind.

Central to Putnam's argument is the observation that every open physical system, S, is in different 'maximal' states[3] at every discrete instant and hence can be characterised by a discrete series of non-cyclic natural state transitions, [s_1, s_2 .. s_t .. s_n]. Putnam argues for this on the basis that every such open system, S, is continually exposed to electromagnetic and gravitational signals from, say, a natural clock. Hence by quantizing these natural states appropriately, every open physical system can by considered as a generator of discrete non-repeating modal state sequences, [s_1, s_2 .. s_][4].

Considering Turing's inputless DSM state machine, Q, and a six state digital counter [c_1 ... c_6], it is trivial to observe that, over time interval [t_1 .. t_6], if we map the state [Q_1] to the disjunction of counter states, [c_1 v c_4], DSM state [Q_2] to the disjunction of counting machine states, [c_2 v c_5] and DSM state [Q_3] to the disjunction of counting machine states, [c_3 v c_6], then the counting machine will fully implement Q as it transits counter states [c_1 .. c_6] over time interval [t_1 .. t_6]. Further, given any [counting] machine state, say [Q_1] ∈ {c_1, c_4}, at time [t_1], we can modally predict that the DSM will enter state [Q_2] at time [t_2].

To show that being in state [Q_1] at time [t_1] caused the counter to enter state [Q_2] at [t_2] we observe that at [t_1] the counter is in state [c_1], (which the mapping function labels DSM state [Q_1]), and that being in state [c_1] at [t_1] causes the counter to enter state [c_2], (which the mapping function labels DSM state [Q_2]) at [t_2]. Hence, given the current state of the counter at time [t], we can predict its future state and hence how the states of DSM Q evolve over the time interval under observation.

Note, after Chalmers, that the counter-machine described above will only implement a particular execution trace of the DSM[5] and Chalmers remains unfazed at this result because he states that inputless machines are simply an

3 A 'maximal' state is a total state of the system, specifying the system's physical makeup in absolute detail.

4 Chalmers (1996) observes, "Even if it [the claim that 'every open physical system is a realisation of every abstract Finite State Automaton'] does not hold across the board (arguably, signals from a number of sources might cancel each other's effects, leading to a cycle in behaviour), the more limited result that every non-cyclic system implements every finite-state automaton would still be a strong one".

5 Clearly there may be other state transition sequences that have not emerged in this execution trace. To circumvent this problem and fully implement an inputless FSA by an infinite state [counter] system, Chalmers posits the system with an extra dial - a sub-system with an

"inappropriate formalism" for a computationalist theory of mind[6].

Clearly the addition of input makes the DSM formalism non-trivial. There can now be branching in its execution trace, as the next state is contingent on both its current state and the input. This gives the system a combinatorial structure. But, as Chalmers observes, Putnam's revised construction does not properly encapsulate this structure – rather it merely manifests one trace of the FSA with a specific input/output dependency. So we are left with the counter intuitive notion that, for example, when using a rock to implement a two plus two program, we mark two on the input area of the rock and four on the output and credit the rock with computing the result..

In his 1996 paper, Chalmers introduces a more suitable FSA formalism, which makes explicit such input/internal-state dependencies, the Combinatorial State Automaton, CSA. A CSA is like - and no more powerful than - a conventional Finite State Automaton, FSA, except that its internal states, [S], are structured to form a set, $\{s_1, s_2 \ldots s_n\}$, where each element $\{s_i\}$ can take on one of a finite set of values or sub-states and has an associated state transition rule.

Chalmers then demonstrates how to map a CSA onto a physical system in such a way as to deal with such input/ internal-state dependencies correctly and preserve the internal functional organisation of the original program, but only at the price of a combinatorial increase in the number of states required for the implementation. In fact, as he illustrates in his paper, executing even the most trivial FSA with input and output, over a small number of time steps would rapidly require a physical system with more states than atoms in the known universe to implement it. So it seems that "we can rest reasonably content with the knowledge that the account as it stands provides satisfactory results within the class of physically possible system", and functionalism is preserved.

The problem that the CSA makes explicit is that of fully encapsulating the complex inter-dependencies between machine state and the input. To implement these using an open physical system requires an astronomical number of internal states, whereas the simple implementation of an inputless FSA that Putnam describes functions only because of the subsequent loss of generality. However, as we observed with Turing's DSM, when input is defined over a specific time interval, the combinatorial state structure collapses to a bounded linear path which can be simply generated using Putnam's mapping and any open physical system.

Returning to a putative conscious robot such as Cronos; at the heart of such a beast there is a computational system – typically a microprocessor; memory and memory mapped peripherals. Such a system forms a Discrete State Machine, DSM in interaction with its environment[7]. Thus, recalling that the computational states of DSMs are 'observer-relative' - requiring a mapping function to be fully determined from the physical state of the system - we note that with input to the robot specified and fixed over a finite time interval, we can simply map the execution trace of its control program onto the state evolution of any digital counter (or, pace Putnam, any open physical system).

Hence, if the state evolution of the robot DSM instantiates phenomenal experience, then so must the state evolution of any open physical system and we are inexorably led to embrace a panpsychist worldview where phenomenal consciousness is found everywhere.

---

arbitrary number of states, $[c_{[\text{dial-state, counter-state}]}]$. Now, associate dial-state [1] with the first run of the FSA. The initial state of the counter machine will thus be $[c_{[1, 1]}]$ and we associate this with an initial state of the FSA. Next associate counter states $[c_{[1, 2]}]$, $[c_{[1, 3]}]$ with associated FSA states using the Putnam mapping described earlier. If at the end of this process some FSA states have not come up, we choose a new FSA state, [C], increment the dial of the counting machine to position [2] and associate this new state $[c_{[2, 1]}]$ with [C] and proceed as before. By repeating this process all of the states of the FSA will eventually be exhausted. Then, for each state of the inputless FSA there will be a non-empty set of associated counting machine states. To obtain the FSA implementation mapping we use Putnam's mapping once more and the disjunction of these states is mapped to the FSA state as before. Chalmers remarks, "It is easy to see that this system satisfies all the strong conditionals in the strengthened definition of implementation [above]. For every state of the FSA, if the system is (or were to be) in a state that maps onto that formal state, the system will (or would) transit into a state that maps onto the appropriate succeeding formal state. So the result is demonstrated." (Chalmers 1996, p.317). However this extension is not required for the argument developed herein.

6 "To see the triviality, note that the state-space of an inputless FSA will consist of a single unbranching sequence of states ending in a cycle, or at best in a finite number of such sequences. The latter possibility arises if there is no state from which every state is reachable. It is possible that the various sequences will join at some point, but this is as far as the 'structure' of the state-space goes. This is a completely uninteresting kind of structure, as indeed is witnessed by the fact that it is satisfied by a simple combination of a dial and a clock. (ibid., p.318).

7 NB. It is central to the computationalist underpinning of cronos that its putative conscious states are not contingent upon it physically interacting with a physical environment; in personal communication, Prof. Holland envisaged a possible follow up project in which the entire cognitive architecture of cronos and its environment are entirely implemented in software, in a large scale virtual reality simulation.

4. Objections: (1) Do counterfactuals matter?

In Bishop (2002) I discuss several objections to this reductio with, perhaps, the most potent coming from David Chalmers who argues that 'as the above only implements one execution trace of the DSM it is not sensitive to counterfactuals; and it is only the possibility of appropriate counterfactual behaviour that guarantees phenomenal experience'

My initial response to this line of argument (Bishop 2002a; Bishop 2002b); followed from Maudlin's Supervenience thesis. Consider what happens if a putatively conscious robot, R_1, with full counterfactual sensitivity, is step-by-step transformed into new robot R_2, such that its resulting behaviour is determined solely by a linear series of state transitions; substituting each conditional branching state transition sequence in the evolution of R_1, with a linear state transition defined by current state and the defined input. It seems clear that, over a finite time interval and with identical input, the phenomenal experience of R_1 and R_2 must be the same. Otherwise we have a robot, R_n, ($R_1 < R_n \_ R_2$), whose phenomenal experience is somehow contingent upon the presence or absence of non-entered state sequences contravening Maudlin's 'supervenience thesis' (outlined earlier)[8]. However at the 2006 Tucson consciousness conference, in a paper entitled 'Counterfactual computational vehicles of consciousness', Ron Chrisley suggested that as we morph between R_1 and R_2, with the deletion of each conditional non-entered state sequence real physical differences between the robots emerge. Effectively, with each replacement of each of the non-entered conditional state sequences, we crucially no longer execute their concomitant conditional test and branch instructions[9]; hence the core reductio no longer holds.

To address this criticism I will endeavour to illustrate that the mere execution of a conditional branch instruction where the result of the test is known and fixed also cannot affect any putative phenomenal states instantiated by the program.

Some conditional branch instructions:

- IF (A > B> THEN GOTO {statement sequence A} ELSE {B}
- IF (A > 10) THEN GOTO {statement sequence A} ELSE {B}
- IF (11 > 10) THEN GOTO {statement sequence A} ELSE {B}

A non-conditional branch instruction:

- GOTO {statement sequence A}

The first conditional branch simply states that IF the value of variable A is greater than that of variable B then execute statement sequence {A} otherwise execute statement sequence {B}. The second conditional is of the same form, however this time we are comparing the value of variable A with the literal value '10'. However in the third example, the 'conditional' compares the value of two literals (11 and 10), hence the result of the test will always be true and the program will always follow statement sequence {A}. The fourth example is of a simple branch instruction, whereby control of the program will unconditionally shift to statement sequence {A}.

At this juncture it is critical to note that many modern 'optimising compilers' will automatically convert the third conditional statement to a simple branch instruction (as these execute more efficiently). Further, if the compiler can deduce that the value of A can never be less than or equal to ten during any possible execution of the program, an optimising compiler may also convert the second conditional into a simple branch; similarly, if it can be deduced a priori that A is always going to be greater than B then it may even convert the first statement into a simple branch;

8 "Suppose that a system exists whose activity through a period of time supports a mode of consciousness, e.g. a tickle or a visual sensum. The supervenience thesis tells us that, if we introduce into the vicinity of the system an entirely inert object that has absolutely no causal or physical interaction with the system, then the same activity will support the same mode of consciousness. Or again, if the activity of a system supports no consciousness, the introduction of such an inert and causally unconnected object will not bring any phenomenal state about … if an active physical system supports a phenomenal state, how could the presence or absence of a causally disconnected object effect that state?" (Maudlin, 1989).

9 A 'conditional branch' instruction is an instruction in a computer program of the form, "IF (TEST IS TRUE) THEN GOTO {statement sequence A} ELSE GOTO {statement sequence B}".

hence it is clear that no special phenomenal properties can result from the mere execution of a conditional statement, otherwise the phenomenal properties of a putative robotic system would be in a strong sense conditional on the type of compiler used to compile its control program.

I will now describe four segments of code, used in four, otherwise identical, robots [A .. D], each of which has a red Munsell colour card placed in view of its optical sensor. Electronic circuitry ensures that the value registered by the optical sensor is stored in a digital latch circuit, positioned at location $FFFF[10] in the computer's memory. If, say, the colour sensor indicates red light falling on it, it will register say $FF, otherwise, if say it is in darkness, it will register say $00.

ROBOT A: Sensor reading genuinely contingent on the current ambient light conditions.
LDA $FFFF
IF (A = 0) THEN execute statement sequence {A} ELSE {B}

ROBOT B: Red light permanently illuminates the sensor, so it always registers $FF and $FF is always stored by the latch at location $FFFF.
LDA $FFFF
IF (A = 0) THEN execute statement sequence {A} ELSE {B}

ROBOT C: Sensor faulty so it always registers $FF hence $FF is always stored by the latch at location $FFFF.
LDA $FFFF
IF (A = 0) THEN execute statement sequence {A} ELSE {B}

ROBOT D: The latch is forced to always store $FF at location $FFFF; hence the value subsequently loaded from $FFFF will always be $FF.
LDA $FFFF
IF (A = 0) THEN execute statement sequence {A} ELSE {B}

The question for the computationalist roboteer is which of the four robots [A .. D] will experience phenomenal red. It would appear that, ex-hypothesi, robot A must experience red, as the value obtained from the latch is an accurate reflection of the light signal falling on the sensor. By similar logic, robot B must also experience phenomenal red. – if a different coloured light was shone onto the sensor, the value on the latch would change appropriately.

But consider robot C. It is clear that the program itself has no means of knowing if the sensor is operating properly and hence if value stored in the latch is an accurate representation of the light detected by the sensor; however the value in the latch is now not in any way contingent on the ambient light conditions that pertain. Nonetheless, as the software executed is unchanged, the supervenience thesis suggests that the phenomenal states generated by the program must be the same; robot C must continue to 'see' red

For robot D, data from the colour sensor is no longer stored in the latch; instead the engineer has designed the circuitry so that the latch always stores the value $FF; once again, the program code executed by the robot is unchanged. And again the supervenience thesis suggests that the phenomenal states experienced by the robot will remain same. However, if the control-program for robot D was compiled using an 'optimising compiler' then the subsequent conditional branch would be replaced by a non-conditional branch; demonstrating that non-entered conditional state sequences can be completely removed and the putative phenomenal states of the program must be unchanged, hence Chrisley's objection is invalid[11] and the original reductio holds.

10 The $ sign indicates a hexadecimal number; i.e. a number to the base 16; dgit range is [0 .. 9 A .. F], hence hexadecimal $FF is 15 x 16 + 15 = 255 (decimal).

11 Clearly, if the phenomenal experience of robot D differed from robot A, then the putative phenomenal states of a robot will always be contingent upon the particular type of compiler used by the roboteer (not on the semantics of actual program he or she wants to compile).

Objections: (2) Computational states are not observer-relative but are intrinsic properties of any genuine computational system[12]

In addressing this objection I will initially consider the most primitive of computational systems - a simple two input / single output logic gate [X], with physical behaviour fully specified by the following table of voltage levels:

INPUT-1	**INPUT-2**	**OUTPUT**
0v	0v	0v
0v	5v	0v
5v	0v	0v
5v	5v	5v

It is clear that under **MAPPING-A**, (*+5v = COMPUTATIONAL STATE TRUE and 0v = COMPUTATIONAL FALSE*), the gate [X] 'computes' the logical **AND** function.

Conversely, under **MAPPING-B**, (*0v = COMPUTATIONAL STATE TRUE and +5v = COMPUTATIONAL FALSE*), it is clear that the gate [X] computes the logical **OR** function.

It follows that, at a fundamental level in the physical realisation of any logical system, such 'observer-relativity' must hold: the computational function of the system must be contingent on the 'observer-determined' mapping used[13].

Further, it is clear that even if the physical-to-computational mapping is known, the function of the system remains observer-relative; that is, "different answers grow from the concerns of different individuals"[14]. Consider (a) a chess playing computational machine used to control the position of chess pieces in a game against, say, a human opponent and (b) the same program being used to control the illumination of a strip of coloured lights - the two dimensional chess board being mapped to a one dimensional strip of lights where the colour of each light is contingent on the value (king, knight, pawn etc) of the piece mapped onto it - in an interactive art exhibition. It is clear that the *purpose of the computations* is contingent on their *social use*. In Heideggerian terms, computing machinery doesn't exist in the world until it is put to some use - an event of '*breaking-down*' - such as playing chess when it becomes part of the background of '*readiness-to-hand*' required in the act of playing a game of chess; or interactively controlling an array of lights when it becomes an interactive piece of art. For Winograd and Flores, we see that, "for different people, engaged in different activities, the existence of objects and properties emerges in different kinds of *breaking down*". In these terms it is meaningless to talk about the existence of the computational system without concomitant purposeful activity and associated 'breaking-down'.

5. Conclusion

In this paper I have attempted to demonstrate (a) that computation is always fundamentally observer-relative and that (b) non-entered counterfactual state sequences of the control program of a robot cannot affect its putative phenomenal experience. Thus - being wary of panpsychism - the reductio argument presented herein should be seen to suggest that computers really *cannot* feel.

12 Objection raised by a member of the audience at the presentation of this paper at the 2006 'Computers and Philosophy' conference, Leval, France.

13 Although it is true that as the complexity of the logical system increases, the number of consistent computational functions that can be assigned to it diminishes, it remains the case that its computational properties will always be relative to the threshold logic value used. The 'physical-state' ⇒ 'computational-state' mapping will always co-determine the 'logical-function' that the physical computational system instantiates.

14 Cf. What is a word-processor?, in Winograd, T. & Flores, F. *Understanding Computers and Cognition*, Addison Wesley, 1986.

6. References:

Bishop, J.M., (2002a), Dancing with Pixies, in Preston, J. & Bishop, J.M., (eds), Views into the Chinese Room, (Oxford: Oxford University Press).

Bishop, J.M., (2002b), Counterfactuals Cannot Count: a rejoinder to David Chalmers, Consciousness & Cognition, 11(4), pp: 642-652.

Chalmers, D.J., (1996), Does a Rock Implement Every Finite-State Automaton?, Synthese, 108, pp.309-333.

Chrisley R., (2006), Counterfactual computational vehicles of consciousness, Toward a Science of Consciousness 2006, April 4-8, Tucson Convention Center, Tucson Arizona, USA.

Klein, C., (2004), Maudlin on Computation, (working paper).

Maudlin T, (1989), Computation and Consciousness, Journal of Philosophy 86, pp. 407-432.

Putnam, H., (1988), Representation & Reality, (Cambridge MA: Bradford Books).

Searle, J., (1990), Is the Brain a Digital Computer?, Proceedings of the American Philosophical Association, vol. 64, pp.21-37.