What Cognitive Scientists Need to Know about Virtual Machines

Aaron Sloman (A.Sloman@cs.bham.ac.uk)
School of Computer Science, University of Birmingham, Birmingham, B15 2TT, UK

Abstract

Many people interact with a collection of man-made virtual machines (VMs) every day without reflecting on what that implies about options open to biological evolution, and the implications for relations between mind and body. This tutorial position paper introduces some of the roles different sorts of *running* VMs (e.g. single function VMs, "platform" VMs) can play in engineering designs, including "vertical separation of concerns" and suggests that biological evolution "discovered" problems that require VMs for their solution long before we did. This paper explains some of the unnoticed complexity involved in making artificial VMs possible, some of the implications for philosophical and cognitive theories about mind-brain supervenience and some options for design of cognitive architectures with self-monitoring and self-control.

Keywords: virtual machine; virtual machine supervenience; causation; counterfactuals; evolution; self-monitoring; self-control; epigenesis; nature-nurture; mind-body;

Introduction

Two concepts of a "virtual machine" (VM), have been developed in the last half century to aid the theoretical understanding of computational systems and to solve engineering design problems. The first, Abstract virtual machine (AVM), refers to an abstract specification of a type of system, e.g. a Turing machine, a Universal Turing machine, the Intel pentium, or the virtual machine defined by a language or operating system, e.g. the Java VM, the Prolog VM, the SOAR VM, the Linux VM, etc. An AVM is an abstract object that can be studied mathematically, e.g. to show that one AVM can be modelled in another, or to investigate complexity issues.

The second concept refers to a *running* instance of a VM (Running virtual machine, RVM), e.g. the chess RVM against which an individual is playing a game at a certain time, the editor RVM I am using to type these words, the Linux RVM running on my computer, the networked file-system RVM in our department and the internet – a massive RVM composed of many smaller RVMs, instantiating many different AVMs.

The RVM concept has deep features that have not been widely recognized, or adequately described. Importantly different sub-cases are defined below, e.g. *specific* VMs (SVMs) and *platform* VMs (PVMs). I conjecture that biological evolution "discovered" some of the problems for which different sorts of VM are solutions long before we did and produced designs for RVMs whose functions we need to understand better. Moreover, understanding relations between these VMs and the underlying physical machines (PMs) in which they are implemented can help to clear up some old philosophical problems and pose new ones, including problems about mental causation. This will add clarity and precision (including engineering design information) to the closely related pair of concepts "realization" and "supervenience", and can provide new insights into old puzzles about emergence and the existence and causal efficacy of non-physical states and processes e.g. mental and socio-economic states and processes.

Philosophers have offered metaphysical, epistemological and conceptual theories about the status of such entities, for example in discussing dualism, "supervenience", "realization", mind-brain identity, epiphenomenalism and other "isms". Many deny that non-physical events can be causes, unless they are *identical* with their physical realizations. Non-philosophers either avoid the issues by adopting various forms of reductionism (if they are scientists) or talk about levels of explanation, or emergence, without being able to give a precise account of how that is possible. I shall try to show how recent solutions to engineering problems produce a kind of emergence that can be called "mechanism supervenience", and conjecture that biological evolution produced similar solutions.[1] Moore (1903) introduced the idea of supervenience. Davidson (1970) transferred it from ethics to the context of mind/body relations, inspiring much subsequent discussion, e.g. (Kim, 1993, 1998). Space limits rule out discussing relations between "supervenience" and "realization". A useful introduction is in the *Stanford Encyclopedia of Philosophy*, though it ignores the engineering (computing) examples.

Several varieties of supervenience can be distinguished:

- **property supervenience**: (e.g. having a certain temperature supervenes on having molecules with a certain kinetic energy);
- **pattern supervenience**: (e.g., supervenience of various horizontal, vertical and diagonal rows of dots on a rectangular array of dots, or the supervenience of a rotating square on changes in the pixel matrix of a computer screen);
- **mereological, or agglomeration, supervenience**: possession of some feature by a whole as the result of a summation of features of parts (e.g. supervenience of the centre of mass of a rock on the masses and locations of its parts, each with its own mass);
- **mathematical supervenience**: e.g. Euclidean geometry can be modelled in arithmetic, using Cartesian coordinates, and in that sense geometry supervenes on arithmetic.
- **mechanism supervenience**: supervenience of one machine on another: a set of interacting objects, states, events and processes supervenes on a lower level reality (e.g., supervenience of a running operating system on the computer hardware).

This paper is about mechanism supervenience, relating RVMs with PMs. It is not concerned with the simple case of how one property, pattern, or entity relates to others, but with how a complex *ontology* (collection of diverse entities, events, processes, states, with many properties, relationships and causal interactions) relates to another ontology. It could also be called "ontology supervenience", or perhaps "ontology instance supervenience". Davidson described supervenience as a relation between properties or "respects", whereas mechanism supervenience involves a relation between interacting parts and relations of complex ontology-instances, not just properties. A single object with a property that supervenes on some other property is a very simple special case.

[1] See http://www.cs.bham.ac.uk/research/projects/cogaff/09.html#vms for more details than this conference paper can include.

What Is a Machine?

A *machine* is a complex enduring entity with a (possibly changing) set of parts that interact causally with one another and/or with the machine's environment. The interactions may be discrete or continuous, sequential or concurrent. Different parts of the machine, e.g. different sensors and effectors, may interact with different parts of the environment concurrently. The machine may treat parts of itself as parts of the environment (during self-monitoring), and parts of the environment as parts of itself, e.g. tools, or diagrams (Sloman, 1978, Ch 6&7). The machine may be fully describable using concepts of the physical sciences (plus mathematics), in which case it is a physical machine (PM). Examples include gear mechanisms, clouds, electric motors, tornadoes, plate tectonic systems, and molecular machines in organisms. Many machines manipulate only matter and energy, whereas some, including biological organisms, manipulate information (often in order to control manipulation of matter and energy).

Not All Machines Are Physical Machines

Some information-processing machines have states, processes and interactions whose best descriptions use concepts that cannot be defined in terms of those of the physical sciences, e.g. "inferring", "checking spelling", "playing chess", "winning", "threat", "strategy", "desire". "belief", "plan", "crime", and "economic recession". There is no space to defend the indefinability claim here, though many will find it obvious. Examples of Non Physically Describable Machines (NPDMs) include socio-economic machines, ecosystems and some biological control systems, e.g. motivational mechanisms. NPDMs are ubiquitous in computing systems, but go unnoticed, or misunderstood, by many researchers trying to understand relations between physical and non-physical states and processes. If an RVM contains causally interacting components that cannot be described using concepts of the physical sciences, then some of its states and processes cannot be directly measured or detected using the techniques of physics alone, although some physical measures may *correlate* with non-physical phenomena. Examples are running computer programs doing things like checking spelling, playing chess, sorting email, computing statistics, etc. "Incorrect spelling" cannot be defined in terms of concepts of physics (e.g. "length", "velocity", "mass", "energy", "force", "charge", "temperature").

Computer scientists and engineers refer to NPDMs as "Virtual Machines" (VMs). This can cause confusion, since "virtual" sometimes implies "unreal", as in "virtual reality" (VR). However, a running NPDM/VM *really* has parts that interact with one another and with the environment, in computing systems, e.g. spread-sheets, word-processors, and flight controllers. They can make things happen outside the computer. A person in a VR game is an entity in a RVM, and its actions can cause things to happen even though it is not a real person.

Every RVM depends for its existence and efficacy on a PM on which it is implemented. This is *causal*, not *substance* dualism: an RVM and its main causal interactions can be transferred to a new PM, and the old PM destroyed, but no RVM can work without being implemented in a PM. A more complete discussion (Sloman, 1978, Ch. 6) would show that human-like RVMs are implemented partly in the physical environment (e.g. extending short term memories) and partly in other people, e.g. needed for reference to remote events and places (as P.F. Strawson showed), and for use of concepts like 'date', 'country', 'money', 'crime', 'fame', etc.

Not All RVMs Are Computer-based

Things mentioned in gossip, political debate, novels, plays, social sciences, etc., include RVMs, even though the label "machine" is unusual in this context. Such RVMs are involved in many *causal* interactions: e.g. jealousy can cause weeping, poverty can cause crime, and crime can cause misery. There are socio-economic NPDMs implemented (partly) in running mental VMs, implemented partly in brains and partly in aspects of the physical environment. These entities and their causal interactions, properties, relations, structures, states, events, and processes, are accurately describable only by using concepts that are not physically definable, even though the mechanisms are implemented (ultimately) in PMs. Neural mechanisms appear to be VMs implemented in physical/chemical machines. Chemical processes can be viewed as running VMs, whose entities, states, processes, and causal interactions are "fully implemented" in lower level PMs. Physics itself has layers, including commonsense physics of billiard ball interactions, implemented in the atomic/molecular layer, which physicists are trying to explain as implemented in deeper layers. Nobody knows how many layers will be required.

Misconceptions About VMs

Philosophers and others have written about VMs, but most ignore the variety of types, the complexity of relations between VMs and PMs, and the variety of causal interactions. Some suggest wrongly that talk of VMs is just metaphorical: this ignores advances in science and engineering that go far beyond making metaphors. The engineer who fixes a bug in a VM specification that caused a plane to crash is not dealing with metaphors: future runs of the VM will include different processes. Many still assume that every virtual machine is a finite state machine (FSM), with a collection of states between which it switches, triggered by input signals, with only one (atomic) state existing at a time. A theory that minds are like that (Block, 1996) could be called *atomic state functionalism*. Turing demonstrated that, with an infinite tape, this can provide a surprisingly powerful model of possible forms of information processing. But it is not general enough for our purposes. It cannot cope with what goes on in current multiprocessing computers with multiple external interfaces, because the machine table is not the only controller.

A richer model involves a RVM composed of multiple interacting FSMs, only some of which interact directly with the environment, possibly using external input and output interfaces including analog-to-digital and digital-to-analog con-

verters, all requiring "device driver" software.

Instead of having a *fixed* set of sub-processes, modern computing systems allow new VMs of varying complexity to be constructed dynamically, some running for a while then stopping, others going on indefinitely, some spawning new sub-VMs, which in turn can spawn new sub-VMs. If analog devices are included, there can be VM states that change continuously, instead of only discrete changes. Some VMs can effectively model continuous change while running on digital devices (e.g. in digital video and digital audio systems). Other VMs may not be able to tell whether they are interacting with discrete or physically continuous mechanisms.

Not only static parts and relations but also processes and causal interactions can supervene on physical phenomena. Many people think multiple processes cannot run in parallel on a single-CPU computer, because only one process can run at a time. This ignores how memory mechanisms work in computers. Different software processes have (overlapping) regions of memory allocated to them, which endure in parallel, so that a temporarily passive process can affect an active one that reads some of its memory, justifying the description of "*enduring* interacting sub-systems". In any case, how many CPUs share the burden of running a multi-component VM is a *contingent* feature of its implementation, since some operating systems on multi-cpu systems can dynamically distribute processes among processors as available. Moreover, interrupt handlers connected to constantly "on" physical devices, e.g. keyboard and mouse interfaces, video cameras, etc., allow some processes to constantly watch or control the environment even when they don't have control of the CPU.

Another common misconception is that there are fixed correlations between VM and PM entities and events. The relationships between RVMs and PMs can be continually altered by virtual memory systems (paging mechanisms), and systems using "garbage collection" (reclaiming physical or virtual memory that's no longer in use). The use of structure sharing, changing mixtures of precomputed and computed-on-the-fly structures, and dynamic process creation and termination, can cause enduring and fleeting VM entities, relationships, events, and processes, to have no fixed PM correlates. Different subsets of physical entities can implement an enduring VM entity at different times in its history.

A PM and the VM it implements need not be isomorphic. There need not be any part of the PM that is isomorphic with the VM, not even instantaneously. The structure of the VM can change significantly without structural changes occurring at the physical level though the physical states of millions of switches may need to change to alter conditional connections. A very large "sparse array" in the VM may contain many more locations than there are switches in the PM. Another process will find the sparse array indistinguishable from one implemented using an isomorphic PM structure. Distinct objects in a VM can have implementations that share parts of the PM. Redundant implementations used for reliability can map one VM entity to a set of PM entities. In a PM, circular containment is impossible, whereas it is possible in a VM, e.g. list A contains list B and B contains A (Sloman, 1978).

If you open up such systems you will not see the virtual machine components, only the hardware components. Likewise, as explained above, the existence and properties of the RVMs (e.g. playing chess, or correcting spelling) cannot be directly detected by physical measuring devices. They can only detect more or less reliable physical *correlates*. Reliable detection of VM states, processes, and events by external devices may be impossible for systems with highly dynamic or idiosyncratic virtual-physical mappings. This may also be true of some mental RVMs.

Causation in computing systems

A common misconception about causation in computers is that PM events occur in sequence, every VM event is caused by a specific PM process associated with a particular physical part, and causation is one-way, as depicted crudely in Fig. 1. This is how the *supervenience* or *realization* of mind on matter is conceptualised by many philosophers. The popular model of computers as supporting a single ordered stream of discrete events is false: a modern computing system has many different physical components that are active in parallel, including memories, external sensor interfaces reading information into memory, and output interfaces transmitting information to external systems, some measuring or controlling continuously varying signals: voltages, speeds, temperatures, tilting (e.g. using gyroscopes) and chemical concentrations. As indicated in Fig. 2, the (spatial and temporal) granularity of VMs and PMs is very different, and there can be causal chains between coarse-grained VM events linking PM events, vastly reducing the combinatorics of programming, debugging, or modifying VMs, compared with controlling PMs. This contradicts the view that there is causation by VMs only

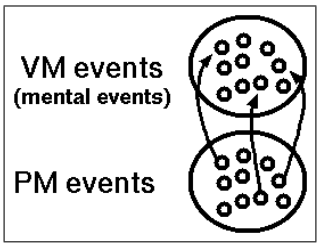

Figure 1: A tempting but incorrect causal model

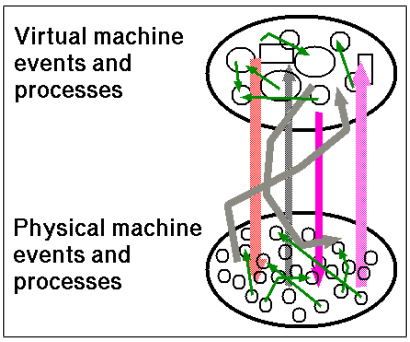

Figure 2: A (slightly) more accurate model – see text.

because the VM events just *are* the PM events described in a different way. On this "VM-PM identity theory" causation by VM events (e.g. decisions) just is causation by the corresponding physical events. However, there are statements true of VM events and false of PM events and *vice versa*. Moreover, if VMs were identical with their implementations, then PMs would be implemented in VMs. They aren't.

A combination of hardware and software technology ensures (most of the time!) the truth of a complex web of conditional statements relating what happens if so and so occurs in physical or virtual machines. The support for that network of relationships, including counterfactual conditional truths (about what would have happened if ....), is equivalent to support for **a complex web of causal connections**, between interacting VM and PM components crudely indicated in Fig. 2. A corollary is that events can be causally overdetermined.

Such causal webs in computers required decades of development of physical devices performing many functions, interfaces/transducers linking computers to other devices, and a plethora of interacting software or hybrid hardware-software sub-systems, including: schedulers, device drivers, file management systems, memory management systems, compilers, interpreters, interrupt handlers, caches, programmable firmware stores, error-correcting memory, wired and wireless network interfaces, network protocol handlers, email systems, web browsers, and many more. Evolution has had far more time, however, We'll return to the implications of that below. *Virtual machine functionalism*, using mechanism supervenience, unlike atomic state functionalism, postulates mind-brain relations more like Fig. 2.

Most man-made virtual machines have a fixed architecture at present. However there is no reason why RVMs should not extend themselves, e.g. through learning processes, just as (Chappell & Sloman, 2007) suggested some biological virtual machines grow themselves.

Why VMs Are Important In Engineering

As explained more fully in (Sloman, 2008) the use of VMs makes a huge difference in the engineering of complex systems – for several reasons. It would be impossible to specify directly all the low level physical processes in a modern computing system as was done with the earliest computers, programmed at the bit level. Formalisms for programming abstract VMs using concepts determined by the tasks, rather than the usually much greater bit-level complexity, allow some designers to specify processes running on those VMs (games, browsers, databases, etc.), while others work on the mappings between VMs and PMs. Initially compilers could statically map VM instructions to PM instructions. Later, as concurrency, multiple users with different privileges and external interfaces were added, the variety and unpredictability of run-time possibilities ruled out static mapping of VMs to PMs, and the tasks had to be allocated to intermediate sub-systems taking decisions at run time. Allowing generic hardware and software designed by specialists to handle those decisions, further simplified tasks for designers of specific applications with concurrently executing subsystems.

This *vertical* separation of design tasks contrasts with *horizontal* separation, i.e. designing and implementing different concurrent or switchable functions. Both can make very complex systems much simpler and easier to specify, design, implement, and debug, because each engineer (or team) works on different, easier problems than getting the whole system to work. Over time, designers of high level systems need to know less and less about the details of what happens when their programs run, leaving low level designs for others, e.g. people writing compilers, device drivers, operating systems, etc. VMs also reduce complexity for system maintainers, allowing them to monitor and debug packages, by looking only at patterns of VM operations, to find out what is going wrong. Another important development was from *specialised VMs* (SVMs) implemented to perform specific functions, to *platform VMs* (PVMs) each capable of supporting many SVMs. Examples of PVMs include operating systems, which are RVMs on top of which many different SVMs can run. Using a high level PVM as the control interface makes a very complex system much more controllable: relatively few high level factors, shared between different functions, are involved in running the system, compared with monitoring and driving sub-processes at the transistor or neuron level. This advantage can apply to *self* monitoring and *self* control in complex systems that need to take high level control decisions McCarthy (1995); Minsky (2006).

Biological Virtual Machines

Evolution clearly produced horizontal modularity, using separate designs for sub-systems with separate functions: different neural or other control subsystems coexist and control different body parts, or produce different behaviours, e.g. eating, walking, breathing, circulating blood, repairing damaged tissue. But if developing new behaviours for the whole organism requires each new behaviour to be implemented in terms of low level states of muscles and sensors shared between competences (e.g. typing, piano playing, preparing food), that could be unmanageable, and very hard to change. Perhaps evolution also "discovered" the advantages of *vertical* modularity based on PVMs, long before we did, e.g. using layered VMs running on brains to provide high level control interfaces between subsystems, including self-monitoring. Moreover, if different control regimes (including motive generation, and conflict detection and management) are implemented on a multi-purpose VM layer, a PVM, the regimes will have much simpler specifications than neural implementations – e.g. making learning by self-monitoring much easier.

For self-monitoring biological and artificial VMs, the compressed, abstract information, ignoring much physiological detail, may often suffice. If self-monitoring and self-controlling systems are grown during an organism's lifetime, as in humans, platform-based vertical modularity could simplify a genome's support for multiple possible developmental trajectories, influenced by demands of the environment

(Chappell & Sloman, 2007). Biological PVMs could allow developing individuals to explore and select different kinds of new functionality, e.g. different behaviours, different languages, different control strategies, different policies for choosing goals or planning actions, and different ways of learning things. There would not be *fixed* machine table as suggested in (Pollock, 2008).

Designs can survive in evolution for the same reason as they are useful for human designers. A design modification giving an organism a kind of PVM-based self-understanding lacked by its competitors, could make it more successful. E.g. it may monitor its own reasoning, planning, and learning processes (at a certain level of abstraction) and find ways to improve them. If those improved procedures can also be taught, the benefits need not be rediscovered only by chance. So PVMs can directly support teaching and learning, and thereby cultural evolution. Using them in control mechanisms goes further than the common suggestion that a robot needs to build a self-model.

However, such mechanisms, while useful much of the time, can also produce incomplete self knowledge and errors in self analysis, etc. Simplifications in self-monitoring VMs could lead robot-philosophers to produce confused philosophical theories about their own mind-body relationships, e.g. theories about "qualia". "Ineffable" qualia arising out of learning in self-monitoring VMs could cause muddle (Sloman & Chrisley, 2003). Intelligent, human-like robots will start thinking about these issues. As science fiction writers have pointed out, they could become as muddled as human philosophers. So to protect future robots from muddled thinking, we shall have to teach them philosophy, provided that we have good philosophical theories to teach. Likewise children.

This raises many research questions, including: How could PVMs have evolved? Do some species allow the environment to influence features of VM construction during epigenesis (Chappell & Sloman, 2007)? How? Can genes specify construction of virtual machines under environmental influence?

It is often assumed that only mechanisms with biological benefits could evolve. However a VM can include low cost "decoupled" subsystems that process information, even though they have only intermittent or no connections with sensors or motors; e.g. a VM playing chess with itself or solving problems in number theory, without any connection to sensors or effectors. Such things may earn their biological keep by occasionally passing discoveries to VMs that are linked to the environment. Some VMs, e.g. sensory processing VMs, may have more going on in them than can possibly be expressed externally using the available external bandwidth. Some constantly active sub-VMs may be capable of being only very slightly influenced by sensor data, with very minor perturbations, while others are strongly driven by the environment.

A system that can change its own instructions can create its own new sub-systems while running, and if this is done largely under external influences, it may turn out to be a system whose development nobody planned, and nobody understands. So the widespread belief that computers can do only what a programmer specifies is false, except in simple cases (with write-protected code!). There are similar erroneous beliefs about genetic influences in humans.

Supervenience and Causation

Many philosophers who investigate mind-matter relationships are generally either ignorant of or simply ignore most of the facts about complex artificial VMs, e.g. the otherwise excellent (Kim, 1993, 1998). Notable exceptions include Bechtel, Boden, Clark, Dennett, Polger, and Pollock. Mental states and processes are said to supervene on or be realized in physical ones. Problems arising include: Can mental processes cause physical events ("downward causation")? If previous physical states and processes suffice to explain physical states and processes at any time, how can mental ones have any effect? How could your deciding to come here influence your movement, if physical causes (in brain and environment) suffice to produce the motion? Before trying to answer such questions for minds and brains it is worth considering systems we already understand much better, because humans designed, built, and can extend and debug them. E.g. a move in a chess VM can cause a computer display to change. Software engineers finding a software bug that causes an airliner to crash, can re-design the system so that early detection of a problem causes a control mechanism to take remedial action, saving lives. Virtual machines can *do* things.

Some philosophers believe that causal connections can exist *only* between physical events and processes; but that raises the problem: at what level of physics? Billiard-ball interactions and even chemical interactions can be argued to involve virtual machines implemented on lower level machines. So if they can be causes, why not allow causation in other virtual machines? Normal human decision making and policy making requires talk of ignorance causing poverty, poverty causing crime, preferences causing decisions, intentions causing actions, experiences causing learning, greed in one person causing harm to others, and many more. If we interpret causation in terms of truth of appropriate (often very complex) sets of conditional statements the mystery can be removed. Fig. 2 indicates (crudely) how causes can operate simultaneously at different levels because a tangled web of true counterfactual conditionals supported by complex technology links events at different levels, showing how virtual machines and physical machines are related so that the same thing can be caused in two very different ways, by causes operating at different levels of abstraction. Software engineers have an intuitive understanding of this, but don't do philosophical analysis.

A key feature of causation is its relationship with conditional and counterfactual conditional questions and statements: Would Fred have crashed if he had drunk less, if the road camber had been greater, if there had been no ice on the road, if he had driven more slowly, etc.? Without offering an analysis of these usages, I have tried to indicate how they relate to causation in VMs, loosely indicated in Fig. 2, which depicts a web of interconnected counterfactual conditional

statements corresponding to detailed implementational mechanisms. A lot more detail is needed to make the points precise.

Multiple layers of virtual machinery

Just as some physical machines (e.g. modern computers) have a kind of generality that enables them to support many different VMs (e.g. the same computer may be able to run different operating systems Windows, Linux, or ....) so are there some platform VMs with a kind of generality that enables them to support many different "higher level" VMs (e.g. the same operating system VM may be able to run many different applications – window managers, word processors, mail systems, spelling correctors, spreadsheets, compilers, games, internet browsers, CAD packages, virtual worlds, chat software, etc. ....). More generally, VMs may be layered: VM_1 supports VM_2 which supports VM_3, etc. The layers can branch, and also be circular, e.g. if VM_1 includes a component that invokes a component in a higher level VM_k, which is implemented in VM_1. Layered virtual machines are not the same as layered hierarchical control systems (e.g. Brooks' "subsumption architecture"), where control layers implement different functions for the whole system, and can be turned on and off independently (mostly). When a VM provides functionality that is implemented in lower levels: the lower levels can't be turned off leaving the higher levels running. Marr's third level, the "computational level" is sometimes regarded as a virtual machine level. However Marr did not allow multiple VM levels, and insofar as his third level was a mapping from sensor data to a scene description, it was a single function, not a machine specification as in Fig. 2.

The label 'emergence' can indicate that a VM non-definitionally extends an ontology: the new concepts required to describe the VM are not definable in terms of old ones. Engineers discussing implementation of VMs in computers and philosophers discussing supervenience of minds on brains are talking about the same 'emergence' relationship. This is not a metaphor: both are examples of the same type.

Conclusion

The idea of a virtual machine (or NPDM) is deep, full of subtleties and of great philosophical significance, challenging philosophical theories of mind, of causation, and of what exists. The use of virtual machines has been of profound importance in engineering in the last half century, even though many of those closely involved have not noticed the wider significance of what they were doing, especially the benefits of vertical separation of concerns, and the complexity of what has to be done to make it all work. The biological relevance has not been widely acknowledged, whereas it seems that evolution "discovered" both the problems and many solutions long before we did, long before humans existed. I expect the biological importance of VMs, including the importance of VMS that grow themselves, will be increasingly acknowledged. The resulting mind-brain theory is *Virtual Machine Functionalism*, not *Atomic State Functionalism*.

If VMs are objects of self-monitoring and self-control they can deceive themselves because many details are inaccessible, as humans have done, including some scientists who examine physical structures and processes in brains in the hope of explaining virtual machine phenomena, without understanding the complexity and sophistication of the mappings that can exist.

Acknowledgements

Many colleagues have helped with these ideas, especially Ron Chrisley, Matthias Scheutz, Jackie Chappell, and colleagues working on the EU-funded CoSy robotics project and its successor CogX, both making heavy use of VMs. Luiz Carlos Baptista reminded me about causation in VR systems. An early version of some of these ideas is (Sloman, 1993). Later ideas inspired by the CoSy project, including ideas about levels of interacting dynamical systems, are in my website and papers in press. Many of the ideas have been developed in parallel by other authors, e.g. (Dyson, 1997). There is much more relevant literature than I have space for here.

References

Block, N. (1996). What is functionalism? In *The Encyclopedia of Philosophy Supplement.* Macmillan.

Chappell, J., & Sloman, A. (2007). Natural and artificial meta-configured altricial information-processing systems. *Int. Jour. of Unconventional Computing*, *3*(3), 211–239.

Davidson, D. (1970). Mental Events. In L. Foster & J. W. Swanson (Eds.), *Experience and Theory.* London: Duckworth.

Dyson, G. B. (1997). *Darwin Among The Machines: The Evolution Of Global Intelligence*. Addison-Wesley.

Kim, J. (1993). *Supervenience and Mind: Selected philosophical essays*. CUP.

Kim, J. (1998). *Mind in a Physical World*. MIT Press.

McCarthy, J. (1995). Making robots conscious of their mental states. In *Aaai spring symposium on representing mental states and mechanisms.* Palo Alto, CA: AAAI.

Minsky, M. L. (2006). *The Emotion Machine*. New York: Pantheon.

Moore, G. (1903). *Principia ethica*. CUP.

Pollock, J. L. (2008). What Am I? Virtual machines and the mind/body problem. *Philosophy and Phenomenological Research.*, *76*(2), 237–309. (http://philsci-archive.pitt.edu/archive/00003341)

Sloman, A. (1978). *The computer revolution in philosophy*. Hassocks, Sussex: Harvester Press. (http://www.cs.bham.ac.uk/research/cogaff/crp)

Sloman, A. (1993). The mind as a control system. In C. Hookway & D. Peterson (Eds.), *Philosophy and the cognitive sciences* (pp. 69–110). CUP.

Sloman, A. (2008). The Well-Designed Young Mathematician. *AIJ*, *172*(18), 2015–2034.

Sloman, A., & Chrisley, R. (2003). Virtual machines and consciousness. *JCS*, *10*(4-5), 113–172.