How Downwards Causation Occurs in Digital Computers

A Preprint

George Ellis
Department of Mathematics
University of Cape Town
Rondebosch, Cape Town 7701
george.ellis@uct.ac.za

Barbara Drossel
Institute of Condensed Matter Physics
Technische Universität Darmstadt
Hochschulstr. 6, 64289 Darmstadt
drossel@fkp.tu-darmstadt.de

August 17, 2019

Abstract

Digital computers carry out algorithms coded in high level programs. These abstract entities determine what happens at the physical level: they control whether electrons flow through specific transistors at specific times or not, entailing downward causation in both the logical and implementation hierarchies. This paper explores how this is possible in the light of the alleged causal completeness of physics at the bottom level, and highlights the mechanism that enables strong emergence (the manifest causal effectiveness of application programs) to occur. Although synchronic emergence of higher levels from lower levels is manifestly true, diachronic emergence is generically not the case; indeed we give specific examples where it cannot occur because of the causal effectiveness of higher level variables.

Keywords Downward causation, Digital computers, Logical control, Transistor Physics

1 Introduction

The issue of whether strong emergence is possible is a highly contested topic, see for example [Hohwy and Kallestrup (2008)], [Humphreys (2016)], and [Gibb *et al* (2019)] for discussion. This is closely linked to the issue of downwards causation ([Ellis *et al* (2012)]), because if higher emergent levels have real causal powers in their own right, as claimed by [Noble (2011)] in the case of biology, they must have the ability to act down to lower levels in order to shape lower level dynamics so as to meet higher level needs. The purpose of the present paper is to show how this occurs in digital computers, where algorithms embodied in computer programs, together with data, are causally effective by shaping outcomes at the electron level. This is what enables the causal power of computer programs in the physical world world, as for instance in car manufacture, landing an aircraft, playing music, playing chess, or searching the web ([McCormack (2011)]). Thus strong emergence certainly takes place in those cases: these outcomes cannot be derived in a solely bottom-up way, because they enable conditional branching logic to have causal power, for example ("IF {*aircraft too high*} THEN {*reduce engine power*}"), and so on, where "too high" is a highly contextual variable.

Here 'causation' is understood in a counter-factual way [Menzies, (2017)]: if the application program were different, which would be the case if a different algorithm were employed (for example computing airflow over wings, rather than the next move in a chess game), there would be different flows of electrons through gates at the transistor level. Thus both the dynamics of the gates (the time-dependent way they implement Boolean logic) and of the electrons (which carriers flow through which semi-conductor materials at what time) is controlled jointly by the application programs implemented in the computer, the data loaded, and the time the program is turned on. These are in turn determined by the computer operator in the social setting which is the context in which this all happens, which social context also led to the existence of the computer in the first place. Physics is not causally complete by itself; it is only causally complete

in this full context (§7).

It has been claimed in Chapter 2 of [Ellis (2016)] that downwards causation takes place in digital computers, and this is implicit in Simon's discussion of technological systems ([Simon H A (1996)]), Tannebaum's description of hierarchical computer organisation ([Tanenbaum (2006)]), and Mellisinos' presentation of how the logical and physical levels in digital computers interrelate ([Mellisinos (1990)]). However none of these writings comment on how the branching logic of algorithms can be realised on the basis of the alleged underlying Hamiltonian physics, whose unitary nature is the essential basis for claims that supervenience - the statement that a specific lower level state uniquely determines a specific higher level state, and any difference in higher level state must be reflected in a difference in lower level states - undermines the possibility of strong emergence.

In this paper, §2.1 sets the stage by discussing the relevant hierarchical structures. (§3-§5) identifies the mechanism whereby logical variables reach down to cause branching of the physical dynamics at the electron level, and allows abstract structures to have causal powers (§7.1). While synchronic supervenience (that is, supervenience occuring at one time) can be claimed to always occur in digital computers, in many cases diachronic supervenience (that is, supervenience occurring at an earlier time in relation to outcomes at a later time) cannot occur, as discussed in §6. This is analogous to the way such downward causation happens in biology, as discussed in [Ellis and Koppel (2019)]. However the mechanism whereby the underlying microscopic processes at the electron level get conscripted to fulfill higher level purposes is completely different in the two cases (§7.2).

2 Modular hierarchical structures

Like all truly complex systems, digital computers are *Modular Hierarchical Structures* in both physical and logical terms. Because they are structured, higher levels can constrain and control lower level dynamics ([Blachowicz (2013)]). In order to achieve complex emergent outcomes, that structure must be both hierarchical (§2.1) and modular (§2.2). A key feature of such structures is multiple realisability of higher level structure and functions at lower levels (§2.3).

2.1 The orthogonal hierarchies

In digital computers, there is a logical hierarchy and an implementation hierarchy ([Ellis (2016)]:§2). The latter implements the former. Upwards emergence of higher levels and downwards constraints on lower levels takes place in both cases.

The logical hierarchy (software) This tower of 'virtual machines' M_I, each with associated language L_I, is discussed in depth in ([Tanenbaum (2006)]:pp.2-8), and is represented in Table 1.

Level	Logic	Downward Process	Agent
M_6	L_6: Application program	$\Downarrow$ Logic and data	Program code
M_5	L_5: High level language	$\Downarrow$ Translation: Compiler	Syntax, Semantics
M_4	L_4: Assembly language	$\Downarrow$ Translation: Assembler	Syntax, Semantics
M_3	L_3: Operating system level	$\Downarrow$ Partial Interpretation	Operating system
M_2	L_2: Machine language	$\Downarrow$ Interpretation	ISA
M_1	L_1: Microprograms	$\Downarrow$ Directly executed	Instructions
M_0	Digital logic level (Binary)	Hardware	Gates

Table 1: *Multilevel machine with a tower of virtual machines M_I each with a language L_I. The lowest logical level is M_1 with machine language L_1, with binary instructions directly executed by electronic circuits. ISA is the Instruction Set Architecture.* Adapted from [Tanenbaum (2006)], Figs. 1-1, 1-2.

Each language is characterised by its Syntax and its Semantics. The user interacts with the application program level M_6, which might be a word processor, internet browser, image processor, etc; or with the high level language level M_5: languages such as FORTRAN, C, JAVA, PYTHON, etc. Compilers or assemblers chain the high level logic down to the microprogramming level; the resulting binary code can be directly executed by the hardware ([Tanenbaum (2006)]).

The implementation hierarchy (hardware) This hierarchy, see Table 2, is implied by [Mellisinos (1990)] and [Tanenbaum (2006)]. It is what enables the logical hierarchy to influence physical reality. However there is not a 1-1 correspondence between the levels in the two hierarchies.

	Entity	Nature
Level 7	Internet	Maximal Network
Level 6	Network	Linked computers, printers, file servers
Level 5	Computer	Components linked by bus, clock
Level 4	Components	ALU, CPU, Memory, I/O devices
Level 3	Gates	Boolean logic: AND, OR, NOT
Level 2	Transistors	Binary ON/OFF function
Level 1	Crystalline structure	Band Structure
Level 0	Carriers	Current Flow

Table 2: *Implementation hierarchy (schematic). This is the physical context within which upward emergence and downward causation takes place. Table 3 amplifies the lower levels.*

The interface between the computer's software and hardware at level 4 is the Instruction Set Architecture (ISA), which is the external (abstract) view of the computer chip (hardware).

2.2 Modularity

As discussed clearly by ([Booch (2006)]:3-26), developing from [Simon H A (1996)], in order to obtain complex behaviour, each level L_N should be made of networks of modules at level L_{N-1}. The key point is that we split a complex task up into simpler tasks, design units to handle the simpler tasks, and then knit them together in networks so that overall when taken together they handle the complex task. To do so, modules are characterised by (a) Information Hiding, (b) Abstraction, (c) Interface Specification, and (d) being Modifiable. The situation is similar for physical modules and for logical modules, and the whole is done in a hierarchical way (sub-modules of modules do even simpler tasks).

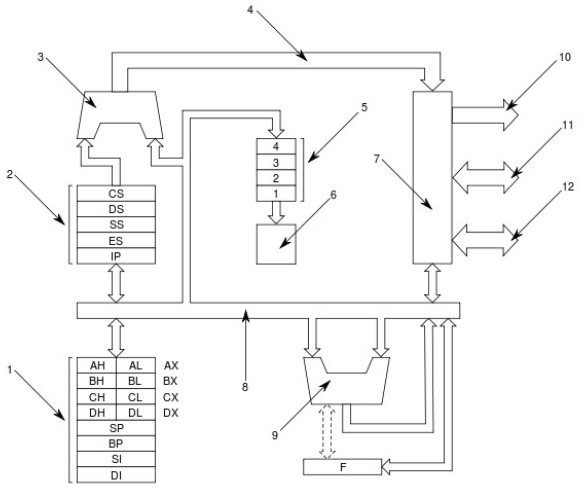

Figure 1: *Simplified block diagram of Intel 8088. 1=main registers; 2=segment registers and IP; 3=address adder; 4=internal address bus; 5=instruction queue; 6=control unit; 7=bus interface; 8=internal databus; 9=ALU; 10/11/12=external address/data/control bus.* (Wikipedia, Intel 8086: open source)

Interface Specification The interface between each module and the rest of the system must be carefully specified, as in the case of the ISA: what control variables, parameters, and data will be sent to the module, and what will be sent out to other modules in the system from the module.

Information Hiding The module's internal variables are hidden from outside view, all that is required is that the module does what it is required to do, as viewed from outside. The user does not mind what internal variables are chosen and what logic is used.

Abstraction An abstraction is an external description of what the module does in a reliable way. This should include a name by which one can refer to the module, which indicates what it does (it should have a unique identifier). This can be seen in the case of the modules identified in Figure 1 (a representation of a specific case of Level 6 in Table 2).

Modifiable The previous points are directly related to multiple realisability (§.2.3) We can change the internal variables and even the internal logic of the module, as long as the externally viewed functionality, as defined by the interface and abstraction, is maintained. The system as a whole only cares that the module does what it is designed to do, not how it does it (although some implementations are preferred over others if they are either faster, or use less resources).

Figure 1 shows a typical set of physical modules that make up level 5 in the Implementation Hierarchy (Table 2).

2.3 Multiple realisability

Multiple realisability is a key indicator that downward causation is taking place [Auletta, Ellis & Jaeger (2008)], [Ellis (2016)], because it indicates that higher level needs are driving lower level structure and outcomes. It does not matter which lower level entities are chosen to carry out higher level needs as long as they do the job; the effective causal entity at the lower level is the *equivalence class* at that level that does what is needed ([Anthony (2008)], [Ellis (2016)], [Bickle (2019)]). This is a central feature of digital computers: both hardware and software can be realised in multiple ways. A modular structure plays a key role in facilitating this (you can alter the module logic and internal variables as long as the interface to the outside world is unchanged).

> **Module Significance:** *Looked at this way, the central reason why modules are important is precisely that they allow multiple realisability to happen*.

Hardware There are many different microprocessors with different instruction sets that can be used in a digital computer while allowing running of the same high-level software [Tanenbaum (2006)]. A classic example is a Java Virtual Machine [Lindholm *et al (2014*], allowing standard Java to be run on any hardware. Wikipedia ("Java Virtual Machine") describes it thus:

> "*A Java virtual machine (JVM) is a virtual machine that enables a computer to run Java programs as well as programs written in other languages that are also compiled to Java bytecode. The JVM is detailed by a specification that formally describes what is required in a JVM implementation. Having a specification ensures interoperability of Java programs across different implementations so that program authors using the Java Development Kit (JDK) need not worry about idiosyncrasies of the underlying hardware platform*."

One can perhaps regard this as the central theme of Alan Turing's great work: a Universal Turing Machine can be implemented in any hardware, provided it does indeed implement the required logic [Hodges (1992)].

Software Any algorithm can be run by any properly defined computer language - of which there are a great many. A particular algorithm (e.g. Bubblesort, see §3.2 below) can be loaded in any desired high level language (FORTRAN, BASIC, COBOL, LISP, PYTHON, C, etc.) and then via compilers and interpreters [Aho *et al* (2006)] the same logic will be realised in the Virtual Machine hierarchy (Table 1) in a different language at each level. A key point here is that algorithms go hand in hand with data structures [Lafore (2002)]; both the software and the data have to be represented correctly at each level.

The relation between hardware and software A further key point is that the boundary between what is done in hardware and what is done by software is fluid. Tannenbaum ([Tanenbaum (2006)]:8) states "*A central theme of this book is that hardware and software are logically equivalent"*. It is a matter of convenience - and performance speeds - how the split is made. Thus for example memory can be virtual ([Tanenbaum (2006)]:429-453) or physical ([Tanenbaum (2006)]:159-173).

2.4 Black boxing not coarse graining

As pointed out long ago by Ross Ashby [Ross Ashby (1992)], when one is dealing with modules that perform logical operations, the appropriate method of going from a lower to a higher level is not coarse graining, it is *black boxing*. One assembles lower level units that perform specific logical operations O_i in networks into an entity that then comprise a next-higher level module performing operations O_I. Examples are how transistors can be connected to give modules that perform arithmetic and logic operations, including being decoders, multiplexers, flip-flops (the basis of memory), registers, and counters ([Mellisinos (1990)]:44-58; [Tanenbaum (2006)]:136-164). A key feature here is multiple realizability, as discussed above: different circuits can give equivalent logical results, as e.g. in the case of the half-adder ([Mellisinos (1990)]:45). Boolean algebra can be used to determine equivalent circuits ([Tanenbaum (2006)]:141-145).

3 Branching logic at all Levels

Causation is a contested word. However in the case of digital computers, it is quite clear it is taking place when the computer is used. A program is loaded, and the operator runs the program. What happens at the physical level is determined by what program is loaded. Thus as mentioned above, we rely on a counterfactual view of causation [Menzies, (2017)]: if a different program is loaded, different flows of electrons will take place at the bottom level.[1] This takes place by a flow of causation from the logical level to the physical level, whereby logical branching (§3.1), exemplified by Bubble Sort (§3.2), leads to physical branching (§3.3), (§3.4).

3.1 Logical Branching

At user program level, one has logical branching of the form IF .. THEN .... ELSE, or WHILE ... DO ...., or REPEAT UNTIL ..... This logic is chained down by compilers or interpreters to lower abstract machine levels, as discussed above (see Table 1). Given this logic, branching is governed by the relevant data.

At each level M_I there is a language L_I with a syntax that allows such branching; the details of the syntax varies at each level, and with the specific software implementation.

3.2 Example: Bubblesort

To give a specific example of logic implemented in a computer, *Bubblesort* is one of the simplest sorting procedures ([Lafore (2002)]:79-89). In pseudocode, it can be written (Wikipedia) as follows

```
1 procedure bubbleSort ( A : list of sortable items )
2     n = length(A)
3     repeat
4         newn = 0
5         for i = 1 to n-1 inclusive do
6             if A[i-1] > A[i] then
7                 swap(A[i-1], A[i])
8                 newn = i
9             end if
10        end for
11        n = newn
12    until n ≤ 1
13 end procedure
```

Note the branching operations at lines 6 and 12 (there is always an implicit assumption "IF NOT, just carry on"). How this is implemented in C^{++} is shown in https://www.geeksforgeeks.org/cpp-program-for-bubble-sort/ which also gives links to implementation in Java and Python. For an assembly language version, see https://www.geeksforgeeks.org/8085-program-bubble-sort. This variety of implementations of the same logic is of course a demonstration of multiple realisability.

[1] There are of course lower levels than the electron level, as described by the standard model of particle physics, which may in turn depend on even lower levels such as string theory/M theory.They are of no concern to us here.

Furthermore there are a variety of other sorting algorithms, e.g. Mergesort, Quicksort, Heapsort, etc ([Lafore (2002)]:89-108,279-294,315-364). They have been extensively compared for efficiency (*Wikipedia: Sorting Algorithm*). The point here is again that of multiple realisability and modularity: many programs need a sorting subroutine, so the logical need is simply an algorithm which will sort a list; the operational need is a module that will be efficient in terms of the type of list that will need to be sorted. It is that functional requirement that drives the designer's choice of a specific sorting algorithm in a particular program. Thus higher level need acts down to select lower level options from amongst a variety of choices (a classic example of adaptive selection).

3.3 Physical Branching

Corresponding to the logical branching, physical branching takes place at each level in Table 2:

At transistor level, the branching is ON or OFF, depending on the gate voltage $V(t)$ (see (1)).

At gate level, the branching is in terms of truth values of basic logical operations: AND, NOT, NOR which can then lead to branching in comparators, adders, decoders, etc. ([Mellisinos (1990)]:37-58, [Tanenbaum (2006)]:135-164), with outcomes depending on the data.

At computer level: branching occurs via CPU control of the basic instruction FETCH-EXECUTE cycle ([Mellisinos (1990)]:75, [Tanenbaum (2006)]:173-202), with activation of different instruction memory and data memory locations and output modules via a bus [Tanenbaum (2006)]:176-220) controlled by a clock. Branching occurs according to the relevant data.

At network level: branching is for instance via requests represented by a Hypertext Transfer Protocol (HTTP), sent between computers identified by their Internet Protocol address (IP address), as for example sending a request to Google for information.

3.4 Relation

Logical branching governs physical branching, which is demonstrable by changing the program loaded. Outcomes alter, although the physical structure involved (the computer hardware) is identical. This happens via compilers [Aho *et al* (2006)] and interpreters ([Tanenbaum (2006)]) that chain logic down to the machine code level. The logic of the algorithm [Knuth (1973)], for example Bubblesort (§3.2), is preserved during the downward chaining process, as it gets rewritten in different languages L_I with different variables and syntax (Table 1). The abstract becomes physical at the machine level, where digital logic is expressed in a timed sequence of electron flows that turn transistors ON or OFF ([Tanenbaum (2006)]:2-7, 231-326; [Mellisinos (1990)]:75-80).

4 Contextual branching at the transistor level

Time dependent control signals at the machine level alter gate voltages $V(t)$ of integrated circuit transistors (see Figs. 2 and 3). This changes a transistor from OFF to ON or vice versa, depending on a threshold level In the MOSFET case shown in Figs. 2 and 3, the logic is,

$$\{IF\ V(t) > V_{threshold}\ THEN\ current\ flows\ ELSE\ not\} \tag{1}$$

The question is how this branching logic relates to a unitary description at the microscopic level, which does not *per se* allow branching (cf. [Ellis and Koppel (2019)]). We discuss in turn, the interacting levels (§4.1); the disconnect between descriptions at different levels (§4.2); and the downward links for branching causation (§4.3).

4.1 The interacting levels

To understand how the physical makeup of transistors can enable logical branching, one needs to look in more detail at the physical hierarchical structuring at the transistor level (Table 3, in effect an expansion of the lower levels of Table 2).

	Level	Structure	Outcomes
T5	gates	transistor combinations	Boolean Logic
T4	transistors	base, emitter, collector; carrier channels	ON/OFF
T3	crystal structure	ions; symmetry, broken by impurities	phonons, band structure
T2	Electron population	densities, average flows, resistance	electron diffusion, current
T1	Individual ions, electrons	ion bonding; electron velocities, collisions	electron drift

Table 3: *The physical (implementation hierarchy) at the transistor level.*

Level T5: Gates Transistors linked via wires and resistors form the basic logical gates. Some of them can be made of a single integrated set of transistors.

Level T4: Transistors Transistors are based in semiconductors such as silicon that have been doped with donor (type n) or acceptor (type p) impurities. Considering *Field Effect Transistors* there is an (n,p,n) structure. Fig.2 shows it when

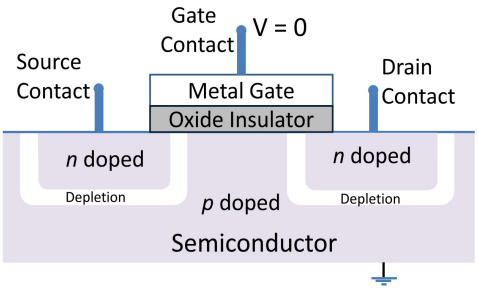

Figure 2: MOSFET with no bias applied to gate. The depletion layers separates the n-doped and p-doped regions, so no current flows. Source: [Simon (2013)]:204. With Permission from OUP and the author

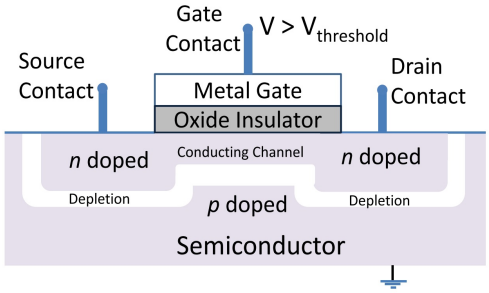

Figure 3: MOSFET with bias applied to gate. The depletion layer shifts, so a continuous channel forms between the source and drain and current flows. Source: [Simon (2013)]:205. With Permssion from OUP and the author

not conducting and Fig.3 when it is conducting. The gate is separated from the p-and n-doped semiconductor regions by an oxide insulator, making a capacitor. An applied voltage on the gate attracts electrons from the source and thereby opens a conducting channel between the source and drain, through alterations to the chemical potential and depletion region.

Level T3: Crystal structure, phonons, electronic bands The crystal structure with its particular symmetries and degrees of freedom gives rise to the various types of phonons and to the electronic band structure. Depending on the distance between the bands and their filling, one obtains the distinction between conductors, insulators, and semiconductors ([Mellisinos (1990)]). In a semiconductor, conduction occurs only at nonzero temperature as electrons need to become excited from the valence band to the conduction band.

Level T2: Electron population Electron/carrier flow is due to (1) Diffusion (due to a density gradient), leading to depletion regions; (2) Drift due to an electric potential, leading to a current. Resistance, characterised by a collision time, occurs due to collisions with impurities, phonons, and other electrons. Electron conduction is time asymmetric because of interaction with a heat bath. Due to its dissipative nature, electron conduction is not a Hamiltonian process.

Level T1: Ions and electrons The description on the most microscopic level is based on a Hamiltonian for the ions and electrons ([Phillips (2012)]:16):

$$\begin{aligned} H &= -\sum_i \frac{\hbar^2}{2M_i}\nabla^2_{\mathbf{R}_i} - \sum_i \frac{\hbar^2}{2m_e}\nabla^2_{\mathbf{r}_i} + \sum_i \sum_{j>i} \frac{Z_i Z_j e^2}{4\pi\epsilon_0 \left|\mathbf{R}_i - \mathbf{R}_j\right|} \\ &\quad - \sum_i \sum_j \frac{Z_i e^2}{4\pi\epsilon_0 \left|\mathbf{R}_i - \mathbf{r}_j\right|} + \sum_i \sum_{j>i} \frac{e^2}{4\pi\epsilon_0 \left|\mathbf{r}_i - \mathbf{r}_j\right|} \end{aligned} \tag{2}$$

However, in order to derive from this Hamiltonian the actual lattice structure and electron charge distribution in the transistor, a series of approximations is required:

A: First, the electrons are separated into conduction band electrons (essentially unbound and so free to move) and valence band electrons (closely bound to ions and so localised). This transforms the Hamiltonian (2) to the form,

$$H = T_i + T_e + V_{ii} + V_{ee} + V_{ei} + E_{core} \tag{3}$$

B: Next, the Born-Oppenheimer (adiabatic) approximation is used, which assumes that the electrons are at all times in equilibrium with the positions of the ions, see for instance [Schwabl (2007)], Ch. 15. To this purpose, the wave function

is factorized into an electron part $\Psi_e(\mathbf{r}, \mathbf{R})$ for given positions of the ions, and an ion part $\Phi(\mathbf{R})$,

$$\Psi(r, \mathbf{R}) = \Phi(\mathbf{R})\Psi_e(\mathbf{r}, \mathbf{R}) \,, \tag{4}$$

leading to the electron equation

$$(T_e + V_{ee} + V_{ei})\Psi_e(\mathbf{r}, \mathbf{R}) = E_e(\mathbf{R})\Psi_e(\mathbf{r,R}) \tag{5}$$

and the ion equation

$$(T_i + V_{ii} + E_{core} + E_e(\mathbf{R}))\Phi(\mathbf{R}) = E\Phi(\mathbf{R}) \,. \tag{6}$$

C: The electron equation (5) is used to obtain the electronic band structure. Since the band structure is based on a picture of non-interacting electrons, the interaction term V_{ee} must be dropped so that the electron Hamiltonian becomes a sum of one-particle Hamiltonians. Given a periodic lattice, the solutions are Bloch waves, giving the energy bands $E_n(\mathbf{k})$, which are the lattice analog of free particle motion ([Phillips (2012)]:22).

D: Electron-lattice interactions occur via phonons ([Simon (2013)]:82-84, 90-95). The ion equation (6) can be used to derive the phonon modes of the ions by making several further approximations (localized ions, harmonic expansion of the energy around the equilibrium ion positions). The normal modes of the resulting harmonic model determine the dispersion relations of phonons.

E: Finally, in order to model electron-phonon interactions explicitly, a quantum field theoretical formalism is required that is based on creation and annihilation operators for electrons and phonons, see for instance [Solyom (2009)], Ch. 23. The dispersion relations of electrons and phonons obtained by the above-mentioned methods determine the electron and phonon propagators, but the interaction term requires a separate evaluation of the cross section for the scattering of electrons via the emission or absorption of a phonon.

Doping Effects Doping with impurities adds electrons to the conduction band (donor, or n-doping) or holes to the valence band (receptor, or p-doping) ([Simon (2013)]:187-194).

p-n and n-p Junction effects The junctions between n-doped and p-doped regions are the key feature of transistors ([Mellisinos (1990)]:14-20; [Simon (2013)]:199-203). Electron diffusion at such junctions leads to depletion regions, stabilised by induced electric fields ([Mellisinos (1990)]: Fig.1.9). Transistors are created by suitably shaped such junctions ([Simon (2013)]:203-205).

Electric field effects We do not want to model the dynamics of a transition between ON/OFF states in the transistor, but rather the steady state situation when it is in one or other of these states (conducting or not) according to whether the applied voltage is above a threshold or not. This means that electric field effects must be modelled.

To do this, the gate voltage $V(t)$ must be added to the model. This leads to a potential energy term in the Hamiltonian of the electrons:

$$H_V(t) = \sum_i eV(\mathbf{r}_i, t) \tag{7}$$

where the Level T_4 variable $V(t)$ determines the Level T_1 variables $V(\mathbf{r}_i, t)$ in a downward way. It leads to a displacement of the electrons until a new equilibrium is reached where the electrical field created by the modified charge distribution cancels the electrical field due to the gate voltage. In order to calculate this new equilibrium, a self-consistent calculation based on the charge density due to doping, gate potential, and thermal excitation must be performed.

This alters the band structure ([Mellisinos (1990)]:19, [Simon (2013)]: Fig.18.6) and thereby either creates a conduction channel by changing the depletion zone, or not, according to the bias voltage applied (compare Figs. 2 and 3 above). In the case of the ON state, an additional electric field between the two contacts leads to electron conduction, which will be discussed in more detail further below.

4.2 The disconnect

The previous description has shown that even though one starts with the electron-ion Hamiltonian, one makes subsequently a series of approximations that effectively replace the initial model by other models. These other models involve classical elements and elements from statistical physics, in addition to Hamiltonians. It is this combination of elements that enables the switching between the ON and OFF state of the transistor.

(1) **Level** T_1: The Born-Oppenheimer approximation is based on the idea that ions are localized objects, neglecting entanglement between ions and electrons. This is an element from classical physics, and it is essential for obtaining the lattice structure, which breaks the symmetry of the underlying Hamiltonian [Primas (1998), Chibbaro *et al* (2014]. A purely quantum-mechanical calculation cannot give such a symmetry-broken ground state. However, this symmetry-broken state is the basis for deriving electronic band structure and phonons (cf [Anderson (1972)], ([Phillips (2012)]:1-2). This means that the new Hamiltonians that describe electrons in bands and quasiparticles called phonons are based on a new, phenomenological theory that includes essential classical ingredients.

(2) **Level** T_2: Electron conduction on the most microscopic level is typically described using the Boltzmann equation ([Phillips (2012)]:179-187), which includes collision rates as basic ingredient. This means that a model with localized electrons and stochastic transitions is used, neither of which feature in the initial Hamiltonian. This description is therefore not derived from a microscopic Hamiltonian, but it is the appropriate form of physical dynamics that applies at that level: it has its own logic that is not fully determined by the underlying quantum physics.

(3) **Level** T_3: The derivation of the phonon dispersion relations involves the quantization of classical harmonic oscillations, which in turn are obtained by approximating the ion equation from the Born-Oppenheimer approximation by an harmonic expansion around the ion equilibrium positions. This means that one is using a classical intermediate model, the low-lying excitations of which are quantized by hand. This leads to a new Hamiltonian, which again is not really derived from the supposedly fundamental Hamiltonian for the electrons and ions.

(4) **Level** T_4: In order to calculate the charge in the depletion region, one performs a classical electrostatics calculation based on the electron density in the conduction band obtained from statistical physics calculations of the thermally excited electrons. Again, this is not derived directly from the base Hamiltonian, because contextual information about the structure of the transistor, the doping, the gate voltage, and the temperature need to be taken into account.

Overall there is no continuous derivation of the relevant properties from the Hamiltonian 2. There are rather a series of empirically adequate phenomenological models building on each other but not derivable from them. As stated by Leggett [Leggett (1992)],

> *"No significant advance in the theory of matter in bulk has ever come about through derivation from microscopic principles. (...) I would confidently argue further that it is in principle and forever impossible to carry out such a derivation. (...) The so-called derivations of the results of solid state physics from microscopic principles alone are almost all bogus, if 'derivation' is meant to have anything like its usual sense. Consider as elementary a principle as Ohm's law. As far as I know, no-one has ever come even remotely within reach of deriving Ohm's law from microscopic principles without a whole host of auxiliary assumptions ('physical approximations'), which one almost certainly would not have thought of making unless one knew in advance the result one wanted to get, (and some of which may be regarded as essentially begging the question)."*

> *"This situation is fairly typical: once you have reason to believe that a certain kind of model or theory will actually work at the macroscopic or intermediate level, then it is sometimes possible to show that you can 'derive' it from microscopic theory, in the sense that you may be able to find the auxiliary assumptions or approximations you have to make to lead to the result you want. But you can practically never justify these auxiliary assumptions, and the whole process is highly dangerous anyway: very often you find that what you thought you had 'proved' comes unstuck experimentally (for instance, you 'prove' Ohm's law quite generally only to discover that superconductors don't obey it) and when you go back to your proof you discover as often as not that you had implicitly slipped in an assumption that begs the whole question. "*

> *"Incidentally, as psychological fact, it does occasionally happen that one is led to a new model by a microscopic calculation. But in that case one certainly doesn't believe the model because of the calculation: on the contrary, in my experience at least one disbelieves or distrusts the calculation unless and until one has a flash of insight and sees the result in terms of a model. I claim then that the important advances in macroscopic physics come essentially in the construction of models at an intermediate or macroscopic level, and that these are logically (and psychologically) independent of microscopic physics."*

Such approximations and models are introduced at each step up in the representation of the implementation hierarchy.

4.3 Branching causation: The downwards links

The depletion region grows or shrinks depending on the applied voltage. This cannot be described in a Hamiltonian way, because it is based in diffusion processes. Furthermore the current flow is determined by collisions with impurities and phonons, as characterised by a scattering time τ (a micro variable) associated with the resistance of the material (a macro variable) ([Simon (2013)]:19). In both cases one has irreversible dynamics arising from the underlying unitary dynamics in the context of the transistor structure, and the issue of the arrow of time occurs. Ultimately this is resolved by Contextual Wavefunction Collapse, where heat baths in contact with the external environment determine the local arrow of time from the cosmological direction of time ([Drossel and Ellis (2018)]).

This opening or closing of the conduction channel happens in accord with the instructions at the binary logic level, via the basic computer functioning cycle (fetch half cycle, execute half cycle) ([Mellisinos (1990)]:75-76; [Tanenbaum (2006)]:54-58) executed via modules such as depicted in Figure 1. This happens according to the logic of the algorithm being implemented (see Table 1).

The bottom line The entire design of the transistor is chosen such that it can act as an ON/OFF switch triggered by the gate voltage:

> **The crucial dynamic** *When a time dependent bias voltage $V(t)$ is applied to a transistor gate, it alters the underlying Hamiltonian through the time-dependent potential term $H_V(t)$, see (7), so usual existence and uniqueness theorems do not apply: outcomes are not determined by initial data. Furthermore the dynamic at work is anyway not just about changing a Hamiltonian, but also about the macroscopic world, which has classical features that arise from ongoing localization and symmetry breaking due to interaction with heat baths, which affect what happens at the transistor level. This changes outcomes in a way determined by the potential $V(t)$, as expressed in (1). This is how the logic represented in the abstract structure of the computer program, expressed in machine code, controls the underlying physics at the electron level; the electron dynamics is no longer unitary.*

Outcomes are determined by the time dependent function $V(t)$ determined by the machine code (Level M_1 in Table 1), and applied in the context of the specific detailed structure of the transistor concerned (Wikipedia, Transistor; cf. Figs. 2,3). We represent that structure in a phenomenological way rather than by detailed constraint equations in the Hamiltonian formulation.

5 The nature of variables

Strong emergence enabled by downward causation is clearly taking place when the relevant higher level variables cannot be obtained by coarse graining lower level variables (they are not 'nothing but' a summation of lower level effects) (§5.1); or when the lower level variables are created due to the nature of higher level structures ('downward emergence') (§5.2).

5.1 Irreducible Higher Level Variables

In terms of existence of intrinsically higher level variables, the key distinction is between global versus local variables,

Physical: The transistor structure as a whole (cf Figures 2 and3) is a Level T_4 feature, determining how lower level entities interact with each other. Lattice vibrations ([Phillips (2012)]:171-172) are a property of the crystal as a whole, which cannot be described or understood at any lower level. Thus phonons are a collective property arising from the macroscopic crystal structure ([Phillips (2012)]:169-172). The depletion zone is a property of the specific transistor construction, so is again a higher level variable.

Logical: A computer program as a whole [Abelson and Sussman (1990)] is an irreducible higher level entity. The scope of a variable is a key feature of a program; global variables are higher level quantities. Passing global parameters or variables to subroutines in a computer program changes logical constraints at the lower level.

The logical-physical interface Logical variables - which are defined in digital computers at each level of the virtual machine hierarchy (Table 1) - cannot be derived from physical variables because they are simply of a completely different nature. Rather they can be *represented* by physical variables - which cannot possibly be a bottom-up process, because the very concept of logical variables is not a physics concept.

5.2 Creating lower level variables

Logical: Declaration of variables in hierarchical computer programs includes a statement of scope: whether they are global variables, or only valid in particular sub-routines, in which case they are hidden local variables according to the principles of modular hierarchical structuring.

Physical: In condensed matter physics, many key lower level variables exist because of downward constraints due to the nature of higher level structures such as crystals and junctions:

Phonons come into existence due to lattice vibrations and are a key example of downwards causation ([Sartenaer (2015)], [Guay and Sartenear (2018)] discuss the nature of this downward creation of lower level elements.)

Band structures [Simon (2013)] result from the details of the crystal structure and are modified by tiny levels of impurities - the impurity density is a very carefully controlled global variable.

Depletion regions due to a balance between diffusion and electric field forces (Wikipedia: Depletion region) result from the detailed nature of the junction between p-doped and n-doped regions. Its geometrical design (a higher level variable relative to the electron level) is key.

6 Synchronic and Diachronic Supervenience

It is claimed that downward causation is not possible because of supervenience, see [Gibb *et al* (2019)] for an extensive discussion. *Synchronic supervenience* is the idea that any specific lower level state $L_1(t_0)$ at time t_0 leads to a unique higher level state $H_I(t_0)$ at that time. *Diachronic supervenience* is the idea that any specific lower level state $L_1(t_0)$ at time t_0 leads to a unique higher level state $H_I(t_1)$ at a later time t_1. Its justification is the combination of synchronic supervenience with the alleged unitary dynamics at the lower level: i.e the assumption that the initial lower level state $L_1(t_0)$ uniquely determines the later lower level state $L_1(t_1)$.

However we have seen above that the latter assumption is not generically true: lower level branching dynamics can occur due to higher level variables changing constraints at the lower levels (a simple example is a pendulum of varying length, see the Appendix of [Ellis and Koppel (2019)] for details). Here are some examples of the possibilities (§6.1-§6.4).

6.1 Fixed Programs and data

If the program $\mathcal{P}$ has fixed initial data $\mathcal{D}$ that is loaded with the program, so that at time t_1 both $\mathcal{P}$ and $\mathcal{D}$ are in memory, then diachronic supervenience will hold from that time on: the computer will run according to that program and data, and the outcome is uniquely determined at each time $t > t_1$ (provided the program halts: we will not deal with that issue here). But how did the program and data get there? They were not there at an earlier time t_0. The initial state $S(t_0)$ when the program and data have not been loaded cannot predict the later state $S(t_1)$ they are in memory. Thus diachronic supervenience will not be true from time t_0 to time t_1.

6.2 Interactive programs

Suppose data is input on an ongoing basis by sensors, for example in the case of aircraft automatic landing systems. The data $\mathcal{D}(t)$ used such as height, wind speed and direction, direction to the airport, position of other aircraft, and so on is updated on the basis of radar, GPS, and other readings. These are not predictable from the microphysics: they are high level variables. They are not uniquely predictable from the earlier data. Thus diachronic emergence does not hold. The training of artificial neural nets (ANNs) is another example. Weights of links between nodes are determined by backpropagation algorithms and supervised deep learning methods, which depend on the order in which training examples are used and the specific data set chosen. These are not predictable on the basis of the data initially available to the computer.

6.3 Adaptive programs

Programs that learn, for instance evolutionary computation based in genetic algorithms ([Mitchell (1996)]) go a step further: the program $\mathcal{P}(t)$ is a functions of time, and is optimised as execution proceeds. In such programs, a random number is chosen to seed the selection process, which can be based for example on the precise time when program execution started, and hence is unpredictable from both a microphysics and macroscopic viewpoint.

6.4 Unpredictable input

One can chose to run programs where program branch points are determined by quantum uncertainty. Thus one can have data specified by a sensor detecting particles emitted by decay of a radioactive atom, in which case the detection times and consequent program branchings are unpredictable even in principle because of the foundational nature of quantum uncertainty ([Ghirardi (2007)]). Diachronic supervenience is impossible in this case because the data $\mathcal{D}(t)$ cannot be known in advance of the quantum detection event.

6.5 Evolutionary development

Finally how did digital computers on the one hand, and transistors/integrated circuits on the other, come into existence? They did not exist 100 years ago, now they do - an example of diachronic *emergence* where the outcomes were not entailed by the initial data, so it is not an example of diachronic *supervenience*. They came into being via a creative evolutionary process of creative vision of possibilities leading to adaptive selection of technology and associated concepts, for example the idea of the transistor, its detailed development into a variety of implementations, and the development of the complex manufacturing methods for VLSI chips ([Mellisinos (1990)]:36-37).

6.6 The larger context

A computer is not a closed system. It is an open system and therefore a Hamiltonian description fails anyway. It is open in many respects: new bits from the computer program, needs electrical power, is in contact with heat baths, has finite temperature and emits therefore photons, and so on. The basis for diachronic emergence is not there.

7 Conclusion

This paper clarifies how the logical structure of computer programs, implementing suitable algorithms, controls the flow of electrons at the transistor/gate level in digital computers. To conclude, we discuss in turn in this section, how abstract entities can have causal powers, and hence physics *per se* is not causally complete (§7.1); the difference between the argument presented here and the case of biology (§7.2); the outcome that all levels are equally real (§7.3); and the social connection (§7.4),

7.1 Abstract entities have causal powers

It thereby provides a clear example of *downward causation* from algorithms and data to the relevant underlying physical level (electron flows in semiconductor materials); this is what enables the dual pair of computer programs and their associated data to have genuine causal power - as they undoubtedly do [McCormack (2011)] - through a computational process. The nature of such a process is described by Abelson and Sussman as follows ([Abelson and Sussman (1990)]:p.1):

> "*Computational processes are abstract beings that inhabit computers. As they evolve, processes manipulate other abstract things called data. The evolution of a process is directed by a pattern of rules called a program. In effect, we conjure spirits of the computer with our spells. A computational process is indeed much like a sorcerer's idea of a spirit. It cannot be seen or touched. It is not composed of matter at all. However it is very real. It can perform intellectual work. It can answer questions. It can affect the world by disbursing money at a bank or by controlling a robot arm in a factory*".

Thus they are examples showing that a simple materialist position - the idea that only physical entities can have causal powers - cannot be correct:

> **Abstract entities have causal powers** *It is clear from this discussion that (1) algorithms, (2) computer programs, and (3) data - all abstract entities - have causal powers because they alter physical outcomes in a real world social context.*

This furthermore shows that consideration of digital computers gives a concrete example of how the underlying physics *per se* is not causally complete, as is often alleged. Rather,

> **Causal completeness:** *In the real world, it is only the combination of physics with its logical, social, psychological, and engineering contexts (which includes the values guiding policy) that can be causally complete, because it is this whole that determines what computer programs will be written*

and what data utilised, hence what electron flows will take place in integrated circuits, as per the discussion in this paper.

As a specific example: the amount of money that can be dispersed to you from an ATM will be limited by an agreement you have reached with your bank. The program used to control the ATM will take into account the existence of such limits, and the specific amount you are able to take out in a given time period will be limited by a logical AND operation linking this agreed amount to the amount of money in your account. Thus these abstract variables will control electron flows in both the bank computers and the ATM dispenser mechanism.

7.2 The difference from biology

In the case of biology, much of the arguments about the effectiveness of top down causation, able to guide dynamic outcomes at the underlying electronic level, is the same as presented here [Ellis and Koppel (2019)]. However there is a key difference.

In that case, signalling molecules ([Berridge (2014)]) cause a change in distances between nuclei in other biomolecules and thereby alter the Hamiltonian governing the physics in a time dependent way. Here, the distances between nuclei are fixed to a high approximation: they are subject to temperature dependent fluctuations, but their average positions are essentially unaltered by the electronic signals conveyed to the transistors. It is the electron band structure that is changed via a change in electric potential difference between the base and the conductor, which alters the Fermi surface and depletion zone and hence alters the flow of carriers (electrons and holes).

In both cases it is the exquisitely shaped physical context that enables it to happen: here, the precise design and manufacture of the transistors; in that case, the precise shape of the biomolecules that implement logical choices as explained in [Ellis and Koppel (2019)]. In that case it is a result of natural selection and developmental processes; in this case, a result of intelligent design [Simon H A (1996)] and precision manufacture: a developmental process occurs whereby physicists and engineers try out all sorts of options to see what works best, leading to very precise doping of semiconductors, precise lithography to create integrated circuits, and so on ([Mellisinos (1990)]:25-29,35-37). The computers and transistors would not exist without this evolutionary process, which results in the existence of physical products such as digital computers, whose existence cannot be accounted for by physics alone [Ellis (2005)].

7.3 The outcome: all levels are equally real

In the case of biology, all emergent levels are equally real: because of downward causation, they are all causally effective in terms of their own logic, as emphasized by Noble [Noble (2011)].

In the case of digital computers, each physical level in the implementation hierarchy (Table 2) has real causal powers as expressed in terms of the variables relevant at that level. Causality at the different physical levels is coordinated via a combination of upwards emergence and downwards constraint ([Ellis (2016)]). Furthermore, each virtual machine in the logical hierarchy ([Tanenbaum (2006)] and Table 1) is as real as each other; they are all equally causally effective. It is for convenience that we write programs in terms of higher level variables, which are more attuned to the way the human mind thinks. We could in principle write them at any level!

This equal effectiveness of all higher levels is the explicit result of a largely uncelebrated aspect of computers: the development of compilers and interpreters that translate logic representations downwards between levels [Aho *et al* (2006)], without which computers would be largely unusable.[2] This is the explicit machinery that acts downward to enable abstract high level programs to control flows of electrons in transistors in such a way as to represent abstract logic. In physics terms, the logic of the relevant algorithm is represented by the time dependent pattern of change of the potential energy term (7) in the electron Hamiltonian, in accord with the digital representation of the algorithm in machine code.

7.4 The social connection

The high level programs and data are thus represented in machine code, which is realised in physical terms as a time sequence of current flows at the electron level that implements the algorithm logic. But it is individual or social needs

[2]Try writing a complex program in Assembly language ([Tanenbaum (2006)]:507-521), or much worse, Machine code! The name of Grace Hopper should be up there with the panoply of computer greats such as Charles Babbage, Ada Lovelace, Alan Turing, and John von Neumann: see Wikipedia, 'Grace Hopper'.

or purposes that control what programs are actually written and change the world, such as Google, Facebook, Paypal, Uber, AirBnB, and so on ([Thompson (2019)]). Ethical issues necessarily arise, for example as regards how social media such as Facebook use algorithms based in social engineering to increase impact factors ([Bissell (2019]), so ethical values are important causal factors in program design. The underlying values and understandings of meaning that guide social interactions are in fact the highest level causal factors in computer usage: they are key determinants in deciding what needs will be met in what way by what algorithms. However electron flows make it happen.

The results of this paper have possible implications for arguments regarding emergence and reductionism in the case of the mind/brain ([Hohwy and Kallestrup (2008)], [Humphreys (2016)]), and specifically those arguments against free will based in the alleged causal completeness of physics at the bottom level, related in a key way to the possibility of responsible action. We will not pursue those issues here; we simply comment that the results of §7.1, combined with those of [Ellis and Koppel (2019)], are potentially relevant to the argument.

Acknowledgements We thank Steven Simon and Oxford University Press for permission to reproduce Figures 2 and 3 from [Simon (2013)]. This project was completed while both authors were visiting the Quantum Research Group at the University of KwaZulu Natal (UKZN), and we thank Francesco Petruccione for his hospitality at UKZN and support from his research grant number 64812: National Research Foundation (South African Research Chair).

References

[Abelson and Sussman (1990)] Abelson H and Sussman J S (1990) *Structure and Interpretation of Computer Programs* (MIT Press)

[Aho *et al* (2006)] Aho, A. V., Lam, M S, Sethi, R., & Ullman, J. D. (2006). *Compilers, principles, techniques, and Tools*. Addison Wesley.

[Anderson (1972)] Anderson, P. W. (1972) "More is different" *Science* **177**:393-396.

[Anthony (2008)] Anthony, L. M. (2008) "Multiple realisation: Keeping it real". In *Being Reduced*, Ed J Hohwy and J Kallestrup (Oxford University Press), 164-175.

[Auletta, Ellis & Jaeger (2008)] Auletta, G., Ellis, G. F., & Jaeger, L. (2008). "Top-down causation by information control: from a philosophical problem to a scientific research programme". *Journal of the Royal Society Interface* **5**:1159-1172.

[Berridge (2014)] Berridge, M. (2014) *Cell Signalling Biology.* (London: Portland Press)

[Bickle (2019)] Bickle, J (2019) "Multiple Realizability", *The Stanford Encyclopedia of Philosophy*, E N. Zalta (ed.), URL = https://plato.stanford.edu/archives/spr2019/entries/multiple-realizability/

[Bissell (2019] Bissell, T. (2019). *ZUCKED: Waking Up to the Facebook Catastrophe*. (Penguin Random House)

[Blachowicz (2013)] Blachowicz, J. (2013). "The constraint interpretation of physical emergence." *Journal for general philosophy of science* **44**:21-40.

[Booch (2006)] Booch G (2006) *Object oriented analysis and design with application*. Pearson Education India.

[Chibbaro *et al* (2014] Chibbaro, S., Rondoni, L., Vulpiani, A. (2014). *Reductionism, emergence, and levels of reality* (Springer Verlag Berlin Heidelberg).

[Drossel and Ellis (2018)] Drossel, B., and Ellis, G. (2018) "Contextual wavefunction collapse: An integrated theory of quantum measurement" *New Journal of Physics* **20**:113025.

[Ellis (2005)] Ellis, G F R (2005) "Physics, complexity and causality" *Nature* **435**:743.

[Ellis (2016)] Ellis G (2016) *How can Physics Underlie the Mind: Downward Causation in the Human Context* (Springer: Berlin/Heidelberg, Germany)

[Ellis and Koppel (2019)] Ellis, G and Kopel J (2019) "The Dynamical Emergence of Biology From Physics: Branching Causation via Biomolecules" *Front. Physiol.*, 25 January 2019

[Ellis *et al* (2012)] Ellis G F R, Noble D, and O'Connor T (2011) "Downward causation: an integrating theme within and across the sciences?" *Interface Focus* **2** 23 November 2011.

[Ghirardi (2007)] Ghirardi, G. (2007). *Sneaking a Look at God's Cards: Unraveling the Mysteries of Quantum Mechanics* (Princeton University Press).

[Gibb *et al* (2019)] Gibb, S., Hendry, R. F., and Lancaster, T. (Eds.) (2019). *The Routledge Handbook of Emergence*. Routledge.

[Guay and Sartenear (2018)] Guay, A., and Sartenaer, O. (2018) "Emergent quasiparticles. Or how to get a rich physics from a sober metaphysics". In *Individuation, Process, and Scientific Practices* Ed. O Bueno, R-L Chen, and M B. Fagan (Oxford University Press)

[Hodges (1992)] Hodges, A (1992) *Alan Turing: The Enigma* (Vintage Books)

[Hohwy and Kallestrup (2008)] Hohwy J and Kallestrup J (Eds) (2008) *Being Reduced* (Oxford University Press)

[Humphreys (2016)] Humphreys, P (2016) *Emergence: A Philosophical Account* (Oxford University Press)

[Knuth (1973)] Knuth, D. E. (1973). *The art of computer programming, Vol. 1: Fundamental algorithms*.

[Lafore (2002)] Lafore, R (2002) *Data Structures and Algorithms in Java* (SAMS: Indiana).

[Leggett (1992)] Leggett, A. J. (1992). "On the nature of research in condensed-state physics" *Foundations of Physics* **22**:221-233.

[Lindholm *et al (2014*] Lindholm, T., Yellin, F., Bracha, G., and Buckley, A. (2014). *The Java virtual machine specification*. (Pearson Education).

[McCormack (2011)] MacCormick, J. (2011). *Nine algorithms that changed the future: The ingenious ideas that drive today's computers*. Princeton University Press.

[Mellisinos (1990)] Mellisinos, A. C. (1990). *Principles of Modern Technology* (Cambridge: Cambridge University Press)

[Menzies, (2017)] Menzies, P) Menzies, P (2017) "Counterfactual Theories of Causation", *The Stanford Encyclopedia of Philosophy*, Ed. E N Zalta (ed.),
URL = <https://plato.stanford.edu/archives/win2017/entries/causation-counterfactual/>.

[Mitchell (1996)] Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press.

[Noble (2011)] Noble, D (2011) "A theory of biological relativity: no privileged level of causation" *Interface focus* **2**: 55-64.

[Phillips (2012)] Phillips, P. (2012). *Advanced solid state physics* (Cambridge University Press)

[Primas (1998)] Primas, H. (1998). "Emergence in exact natural science" *Acta Polytech. Scand.* **91**, 83–98.

[Ross Ashby (1992)] Ross Ashby, W (1952). *Design for a Brain* (Chapman and Hall)

[Sartenaer (2015)] Sartenaer, O. (2015). "Synchronic vs. diachronic emergence: a reappraisal" *European Journal for Philosophy of Science* **5**:31-54.

[Schwabl (2007)] Schwabl, F. (2007). *Quantum mechanics* (Springer Verlag Berlin Heidelberg)

[Simon H A (1996)] Simon, H. A. (1996). *The sciences of the artificial* (MIT Press).

[Simon (2013)] Simon, S. H. (2013). *The Oxford Solid State Basics* (Oxford University Press)

[Solyom (2009)] Solyom, J. (2009). *fundamentals of the physics of solids Volume II: Electronic Properties* (Springer Verlag Berlin Heidelberg)

[Tanenbaum (2006)] Tanenbaum, A S (2006) *Structured Computer Organisation* (Prentice Hall, Englewood Cliffs), 5th Edition.

[Thompson (2019)] Thompson, C. (2019). *Coders: Who they are, what they think, and how they are changing the world.* (Picador).

Version: 2019/08/14/arXiv