The Only Way is Up
On A Tower of Abstractions for Biology

Jasmin Fisher[1], Nir Piterman[2], and Moshe Y. Vardi[3]

[1] Microsoft Research Cambridge, UK
[2] University of Leicester, UK
[3] Rice University, USA

Abstract. We draw an analogy between biology and computer hardware systems and argue for the need of a tower of abstractions to tame complexity of living systems. Much like in hardware design, where engineers use a tower of abstractions to produce the most complex man-made systems, we stress that in reverse engineering of biological systems; only by using a tower of abstractions we would be able to understand the "program of life".

1 Introduction

System-level approaches in biology have gained mainstream attention in the past decade, in an effort to better understand biological complexity. An important activity in system biology is the development of mathematical and computational models. Abstraction is well understood to be a key to modeling complex systems in general, and biological systems in particular, where by "abstraction" we refer to a model at a certain level of description, suppressing lower-level details in a principled way. All models used in system biology employ abstraction, but they vary in their level of abstraction from low-level differential equations all the way to Boolean logic. Today's systems biology offers a tool set of many different types of abstraction, but without an overall organizing principle. Furthermore, the overwhelming majority of cellular models focus on the levels of genes, proteins, and metabolites, as well as metabolic or regulatory networks. Our claim is that abstraction alone is unlikely to be sufficient as a tool to understand biological systems; what is needed, we believe, is a *tower of abstractions*; that is, a sequence of models of increasing degree of abstraction, each level building on the level below it. Biology, we believe, must "climb up the ladder of tower of abstractions."

To show how a tower of abstractions can be used to tame complexity, it is useful to draw an analogy between biological systems and computing hardware systems. We note that, in recent years, many tools and formalisms that were originally designed for the development and analysis of computing systems have been successfully used for modeling biological systems [14, 10, 24]. Perhaps the most striking resemblance between biology and hardware is the ability to do concurrent computation. Biological systems operate with inherent concurrency events (e.g., biochemical reactions, intercellular signaling, and the like) do not

occur one after the other, but rather concurrently in different compartments over the entire organism just as the logical elements in computing hardware execute concurrently. In order for computing hardware to make sequential progress, for example, to sum up a vector of numbers, one has to add to the hardware memory elements, referred to as registers, which make it possible to transfer values from one machine cycle to the next. Analogously, in a cell, the accumulation of a certain protein may serve as a memory device and triggers events that depend on it. It is exactly this concurrency, however, that makes it difficult to understand the behavior of hardware and biological systems.

Our thesis is that in order to better understand complex biological behaviors, which will hopefully (and eventually) help us understand how genotype gives rise to phenotype, one must think of multiple useful levels of abstraction, similar to the tower of abstractions used by computer scientists and engineers in designing computing hardware. The argument is that to tame biological complexity we must find the right levels of abstraction to model biological systems, and that without such a tower of abstractions it would probably be impossible to understand the machinery behind complex living systems. Furthermore, the analysis through multi-leveled abstraction can serve to identify emergent behaviors of biological systems. A computer cannot be understood by pondering the behavior of transistors, or logic gates; similarly, the behavior of a cell cannot be predicted by understanding its chemistry at a molecular level. In order to understand the protocols employed by biological systems, which Caste and Doyle have suggested will give the necessary tools to reason about biological systems [8], we have to first identify the right levels of abstraction.

The Process of Hardware Design

To pursue the analogy of biology and hardware, it is useful to give a short (and rather simplified) overview of the process of hardware design [29]. The most notable feature of the design process is that it is a top-down process. Hardware design starts with the *formulation of requirements*, typically provided in a natural-language document. The next stage is the development of a *software model* of the intended system. This software model is intended to serve as an initial prototype for the system, which ultimately is implemented in hardware. The software model is an executable model, which can be experimented with, modified, and tuned.

The second design stage is a *transformation* of the software model into a hardware-description language (HDL). Such a language is essentially a programming language for hardware; it includes specialized features that talk about clocks and concurrency. While traditional software programming languages are designed to produce procedural code, executed one command at a time, HDLs assume that everything happens concurrently. An HDL model describes the behavior of the hardware in terms of signal flow and data transfer between registers (memory elements) and the operations performed on these signals and data. Note that the HDL model is not meant to run the software model; the software model

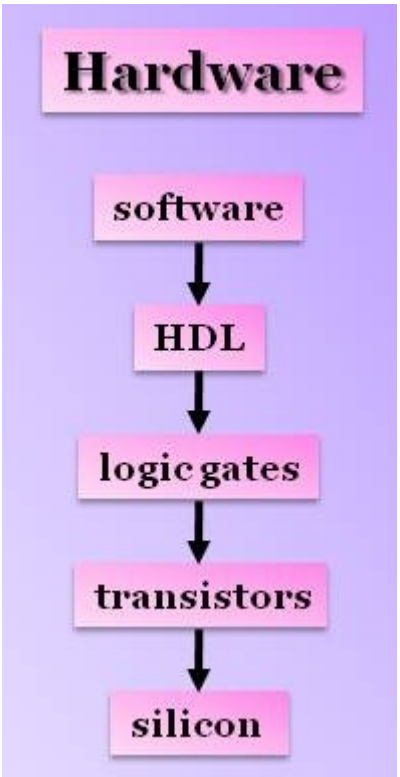

Fig. 1. The process of hardware design

and the HDL model are both models of the same system, but at different levels of abstraction.

The next stage is called *logic synthesis*; it converts the HDL model into a gate-level model, which describes the design implementation in terms logic gates and their connectivity. The conversion uses a predefined library of logic gates (e.g., AND gate with 2, 3, 4, or 8 inputs, etc.) that serve as elementary building blocks. Logic synthesis is typically an automated process, which implies that the two different descriptions (HDL and logic gates) should have exactly the same functionality.

The next stage is called *physical design*; here the logic has to be mapped to its physical implementation, in terms of components, component locations, component wiring, and the like. Here one deals with transistors and wires rather than with logic gates. While previously the main constraints were functional, here they are mainly physical. Length of wires, width of transistors, capacity, power consumption, and timing are the primary concerns. Ultimately, this design phase ends with a photomask, to be used in photolithography. Finally, the transistors and wires are actually printed on silicon.

Let us now illustrate this with a concrete example. We start with a software-level definition of integer multiplication, which can be described by K=I*J. At the HDL level, we choose (for the sake of this example) to implement multiplication via iterated addition. We need registers for I, J, and K (initially 0). We now iterate, at each iteration decrementing I and adding J to K. We stop when I reaches 0. At the gate level, we represent I, J, and K as 32-bit-vectors, that is, arrays of bits, each of length 32. We now need to implement bit-level decrementation and addition in terms of AND, OR, and NOT gates, and replicate that circuit 32 times. Finally, at the transistor level, we need to implement logic gates and registers using transistors. The final device will have thousands of transistors.

What is the point of this detailed description of computing hardware design? It is to emphasize the importance of having multiple levels of abstractions. Abstraction is the hardware designer's primary tool in dealing with complexity. The designers of the first microprocessor, in 1974, were able to work directly at the level of its 2300 transistors, but a modern microprocessor can have over two billion transistors. Today's tower of abstractions in hardware design (see Figure 1)software, HDL, logic-gates, transistors, silicon has emerged from close to 50 years of experience in hardware design. Hardware designers realized not only that abstraction is necessary for taming complexity, but also that several levels of abstraction are actually necessary.

Hardware and Wetware

It is important to note that the description above refers to the design of digital hardware systems, which have discrete behavior and form the basis for most computing systems. In continuous (analog) hardware systems, such as amplifiers, regulators, and filters, the focus is much more in the physical attributes of the devices, such as gain, power, and resistance. Which approach, discrete or continuous, is more appropriate for viewing biological systems? Many researchers believe that biology is completely continuous, doubting whether discrete abstractions can be found at all. This view, in our opinion misses an important point. Even digital computer systems are ultimately continuous systems, implemented in terms of transistors and wires. The value of discrete models is in their utility; they enable us to abstract away from the low-level continuous details. Thus, in hardware design continuous models are used only at the lowest level of abstraction, with higher levels, from logic gates and above, using discrete models. Discrete models are also extremely useful in biology. Indeed, the genetic code is discrete. Similarly, the opening and closing gating of ion channels in response to specific stimuli allowing cells to control their internal environment is just like having discrete switches. Biologists have been using discrete models, such as Boolean networks, since the 1960s [17]. For example, Boolean models correctly capture network motifs such as forward loops and dual-positive feedback loops [4, 19] and lead to better understanding of the Drosophila segment polarity gene network resilience [2, 7].

Clearly, biological systems are the "hardware of life", referred to as "wetware" by Rudy Rucker in his 1988 science-fiction novel [26]. The description above of the hardware-design process reveals, however, fundamental differences between hardware and wetware. Most fundamentally, computing hardware systems are designed by an intelligent designer, while biological systems are the result of billions of years of evolution. Furthermore, while hardware design proceeds in a sequence of distinct well-defined modelssoftware, HDL, logic gates, transistors, and silicon biology provides us with only the final 'model', so to speak, the living organism. What then is the value of the hardware-wetware analogy?

To understand the value of this analogy we need to remember that the biologist is not a designer, but rather a reverse engineer, with the task of uncovering, given a device, the functionality of that device and its principles of operation. Consider now a hardware engineer who is given a hardware device with the task of reverse engineering it. That task can be quite difficult. Take the device described above for integer multiplication. An inspection of a semiconductor chip may reveal an intricate network of thousands of transistors, but may say nothing about the functionality of the device. The reverse-engineering process is helped enormously by the reverse engineer's understanding of the (forward engineering) design process. Understanding that the transistor network implements logic gates, which implement HDL, which implements software, is critical to the success of reverse engineering a hardware device. We believe that the main value of the hardware-wetware analogy is in its showing that abstraction, and multiple levels of it, are absolutely crucial to handling biological complexity.

The importance of abstraction has been implicitly understood for quite some time. As stated by Brenner, "while the genome sequence is central, it is a level of abstraction that is too cryptic to be used for the organization of data and derivation of theoretical models." [6] Boolean gene networks are an example of an abstract model, whose value is that it is much easier to work with than the network of differential equations that it approximates. When Bornholdt says "Less Is More in Modeling Large Genetic Networks" [4], he is pointing out to the value of abstract models. In our view, Biology needs to go beyond mere abstraction and develop its own tower of abstractions. Note that we are not referring here to the fact that biology requires models at different scales (e.g., molecular, cellular, organ), rather, even a single scale requires multiple levels of abstraction, just as hardware designers apply multiple levels of abstraction during the design process at the same scale, for example, multiplication can be performed in terms of iterated addition. In biology, one can also observe different level of abstraction at the same scale. For example, the process of cell-fate determination in the earthworm *C. elegans* can be observed at different levels of abstraction. On the low-level, we can look at signal transduction and describe gene-expression levels [30] and the change in protein quantities over time [11, 20], or, at a higher level of abstraction, we can observe the cell acquiring a specific cell fate according to morphology, cell division, or position of its daughter cells [28, 27]. The cellular module of circadian clocks, constructed from genes and proteins involved in interlocking feedback loops [13], is an example of functional module

that is best considered at a higher level of abstraction than that of regulatory, or metabolic pathways, which in turn are at a higher level of abstraction than that of genes, proteins, and metabolites [22]. The segmentation clock, a transcriptional oscillator that is responsible for vertebrate somitogenesis, is, in turn, an ensemble of numerous cellular oscillators [15].

One may argue that because biological systems are evolved rather than engineered, unlike hardware, they are unlikely to be amenable to hierarchical modeling. We argue otherwise. Evolution selects by fitness, and fitness is determined by phenotype. It is the very high-level attributes and traits of an organism that determine its fitness. Precisely because evolution typically works via reuse and modification of biological modules [16, 18], we should expect a tower of abstractions to bridge the large gap between the genotype and phenotype. The brain is another complex system that is the result of evolution. While at the lowest level, brain functionality is driven by neurons, a full understanding of the brain requires it to be understood in terms of systems, subsystems, and sub-subsystems [23]; one would expect this to also be the case in cellular biology. As an example, let us consider bacterial chemotaxsis, whereby bacteria migrate towards chemical attractants and away from chemical repellents. Chemotaxis is a behavior that contributed to fitness and is therefore selected for by evolution. The molecular mechanisms underlying chemotaxis are a subject of ongoing research, which shows that these mechanisms vary among different bacterial species [25]. The process of chemotaxis is very amenable to hierarchical modeling. Low level models consider the configuration of the molecules in the base of a flagellum and how changes in their phosphorylation leads to the binary choice of clockwise or anti-clockwise rotation [3]. The change in conformation is abstracted in the signaling network model of Rao et al., which includes the sensing (through ligand binding) and rotation-direction decision (through phosphorylation of controller). Higher-level models could, for example, abstract away the signaling network and connect directly sensing and motion.

While hardware is based on a well-defined and well understood tower of abstractions, a standard abstraction tower for biology has yet to emerge, see Figure 2 for a putative tower (of course, a biological tower of abstraction may not be as neat and orderly as the computing hardware tower). Even at the most basic level, we do not have a biological analogy to the most fundamental fact of hardware design, which is that transistors implement logic gates. Searching for the fundamental "bio-logic gates" [10] is a highly active research area. Brandman et al. [5] describe several general building blocks in genetic networks, such as excitatory feedback loop, inhibitory feedback loop, and the like. Nurse [21] calls for a program of describing and cataloguing cellular "logic circuits". In the context of synthetic biology, which is concerned with designing artificial biological systems, Endy [9] has argued for using functional modules and in turn to use these modules to create systems. In essence, these calls are for the development of a "bio-logic gate-level model" (obviously, in a biological setting the components are much more fluid than in an engineering setting, often performing different tasks in different contexts). While the development of such a model would constitute a

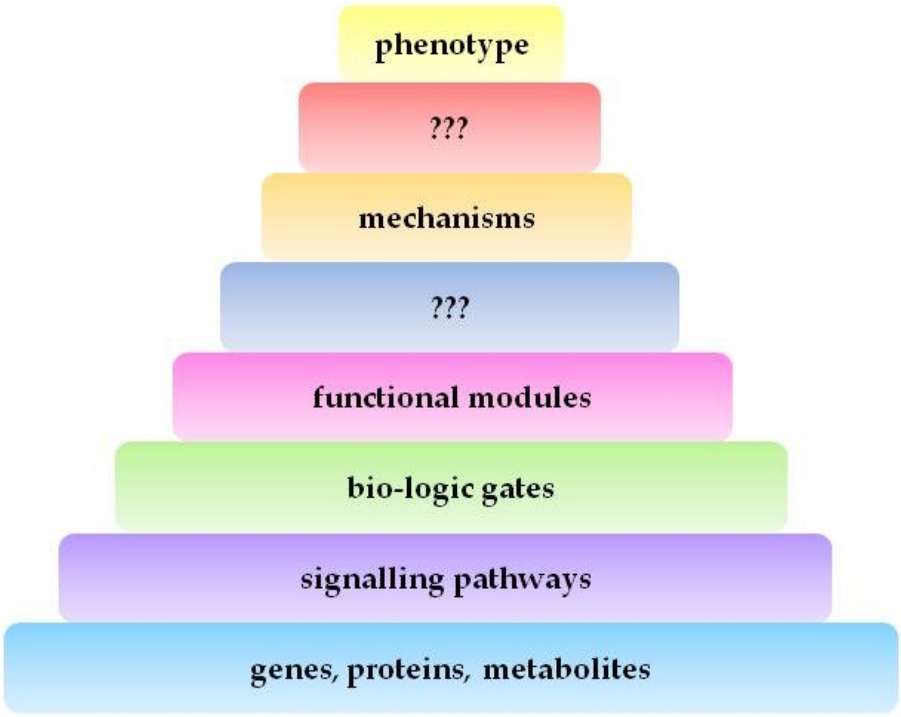

Fig. 2. Tower of abstractions in biology

significant step forward in system biology, we should remember that in hardware design the gate-level model is still a fairly low-level model. The reverse engineer who has uncovered the gate-level model of the multiplication device described above is still far from realizing that the device performs integer multiplication. Similarly, we must define models that are more abstract and higher-level than the "bio-logic gate-level model." The segmentation clock [15], mentioned above, which is an ensemble of numerous cellular oscillators seems to be an example of a functional module that is best considered at a level above that of "bio-logic gates."

The Software of Life

In the tower of abstractions of hardware design, the highest level was the software level, which describes the behavior of the hardware device. What is then the software of life? Let us go back to the example above. The software of the device we described above is the equation Z=X*Y. This equation is not directly represented in the silicon; nevertheless, the silicon implements it. Thus, Z=X*Y *emerges* from the simple and local interaction of the thousands of transistors that

constitute the circuit. It follows that the software of hardware can be viewed as an *emergent behavior* of the hardware. This behavior is the top level in our tower of abstractions; see [1] for a discussion of emergence and multi-levelled abstraction in science. Analogously, the "software of life" is an emergent behavior of biological systems (e.g., chemotaxis). To understand how genotype leads to behavior, we need to identify first the tower of abstractions bridging genotype and behavior. In genetics, the central dogma provides us with the appropriate level of abstraction, referring to the DNA-to-protein transfer. While system biology researchers are largely aware of the importance of abstraction, system biology has concentrated its efforts in models of the gene/protein/metabolite and regulatory network levels. We believe that biological models should have multiple levels of abstraction, starting from molecular-level models, going through bio-logic-gate models, and eventually getting to behavioral models, relating to the "software of life". Identifying these levels of abstraction is, in our opinion, one of the central challenges of system biology; and quoting a recent piece on systems biology theory by Gunawardena [12], "Molecular biology was reductionism's finest hour. Now, there is nowhere left to go but up."

References

1. R. Abott. Emergence explained-abstractions. *Complexity*, 12(1):13–26, 2006.
2. R. Albert and H. G. Othmer. The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in drosophila melanogaster. *J Theor Biol*, 223(1):1–18, 2003.
3. F. Bai, R. W. Branch, Jr. Nicolau, D. V., T. Pilizota, B. C. Steel, P. K. Maini, and R. M. Berry. Conformational spread as a mechanism for cooperativity in the bacterial flagellar switch. *Science*, 327(5966):685–9, 2010.
4. S. Bornholdt. Systems biology. less is more in modeling large genetic networks. *Science*, 310(5747):449–51, 2005.
5. O. Brandman, Jr. Ferrell, J. E., R. Li, and T. Meyer. Interlinked fast and slow positive feedback loops drive reliable cell decisions. *Science*, 310(5747):496–8, 2005.
6. S. Brenner. Sequences and consequences. *Philos Trans R Soc Lond B Biol Sci*, 365(1537):207–12, 2010.
7. M. Chaves, R. Albert, and E. D. Sontag. Robustness and fragility of boolean models for genetic regulatory networks. *J Theor Biol*, 235(3):431–49, 2005.
8. M. E. Csete and J. C. Doyle. Reverse engineering of biological complexity. *Science*, 295(5560):1664–9, 2002.
9. D. Endy. Foundations for engineering biology. *Nature*, 438(7067):449–53, 2005.
10. J. Fisher and T. A. Henzinger. Executable cell biology. *Nat Biotechnol*, 25(11):1239–49, 2007.
11. B. D. Grant and H. A. Wilkinson. Functional genomic maps in caenorhabditis elegans. *Curr Opin Cell Biol*, 15(2):206–12, 2003.
12. J. Gunawardena. Systems biology. biological systems theory. *Science*, 328(5978):581–2, 2010.
13. P. E. Hardin. The circadian timekeeping system of drosophila. *Curr Biol*, 15(17):R714–22, 2005.
14. D. Harel. On comprehensive and realistic modeling: some ruminations on the what, the how and the why. *Clin Invest Med*, 28(6):334–7, 2005.

15. K. Horikawa, K. Ishimatsu, E. Yoshimoto, S. Kondo, and H. Takeda. Noise-resistant and synchronized oscillation of the segmentation clock. *Nature*, 441(7094):719–23, 2006.
16. N. Kashtan and U. Alon. Spontaneous evolution of modularity and network motifs. *Proc Natl Acad Sci U S A*, 102(39):13773–8, 2005.
17. S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *J Theor Biol*, 22(3):437–67, 1969.
18. H. Kitano. Biological robustness. *Nat Rev Genet*, 5(11):826–37, 2004.
19. K. Klemm and S. Bornholdt. Topology of biological networks and reliability of information processing. *Proc Natl Acad Sci U S A*, 102(51):18414–9, 2005.
20. F. Long, H. Peng, X. Liu, S. K. Kim, and E. Myers. A 3d digital atlas of c. elegans and its application to single-cell analyses. *Nat Methods*, 6(9):667–72, 2009.
21. P. Nurse. Life, logic and information. *Nature*, 454(7203):424–6, 2008.
22. Z. N. Oltvai and A. L. Barabasi. Systems biology. life's complexity pyramid. *Science*, 298(5594):763–4, 2002.
23. M. Perus. Multi-level synergetic computation in brain. *NONLINEAR PHENOMENA IN COMPLEX SYSTEMS*, 4(2):157–193, 2001.
24. C. Priami. Algorithmic systems biology. *Communications of the ACM*, 52(5):80–88, 2009.
25. C. V. Rao, J. R. Kirby, and A. P. Arkin. Design and diversity in bacterial chemotaxis: a comparative study in escherichia coli and bacillus subtilis. *PLoS Biol*, 2(2):E49, 2004.
26. Rudy v B. Rucker and Copyright Paperback Collection (Library of Congress). *Wetware*. Avon Books, New York, 1988.
27. P. W. Sternberg and M. A. Felix. Evolution of cell lineage. *Curr Opin Genet Dev*, 7(4):543–50, 1997.
28. J. E. Sulston. C. elegans: the cell lineage and beyond. *Biosci Rep*, 23(2-3):49–66, 2003.
29. J.F. Wakerly. *Digital Design: Principles and Practices*. Pearson Education, 4th edition, 2008.
30. M. Wang and P. W. Sternberg. Pattern formation during c. elegans vulval induction. *Curr Top Dev Biol*, 51:189–220, 2001.