

# Abstract Machines of Systems Biology<sup>1</sup>

*Luca Cardelli*

Microsoft Research

**Abstract.** Living cells are extremely well-organized autonomous systems, consisting of discrete interacting components. Key to understanding and modeling their behavior is modeling their system organization. Four distinct chemical toolkits (classes of macromolecules) have been characterized, each combinatorial in nature. Each toolkit consists of a small number of simple components that are assembled (polymerized) into complex structures that interact in rich ways. Each toolkit abstracts away from chemistry; it embodies an abstract machine with its own instruction set and its own peculiar interaction model. These interaction models are highly effective, but are not ones commonly used in computing: proteins stick together, genes have fixed output, membranes carry activity on their surfaces. Biologists have invented a number of notations attempting to describe these abstract machines and the processes they implement. Moving up from molecular biology, *systems biology* aims to understand how these interaction models work, separately and together.

**Preface to this reprint.** In this paper from 2005 I tried to summarize what I found most remarkable about cellular organization, that is the fact that *cells compute*, and that computation is one of their most important functions. We can easily understand “why” cells need to compute: unicellular organisms, in the active search for food and the active avoidance of predators, need to deploy increasingly sophisticated and highly optimized biochemical *algorithms*, in an information processing arms race with each other. (It’s hard for me to use a different term than *algorithms*, although *pathways* and *networks* are common terms.) However, we do not really understand “how” cells compute, except in very basic cases. Among the many, many things that are *not* known about biology, this is the one that should concern us most, particularly because information processing at the cellular level is so basic. Unlike more evolved information processing systems in higher organisms, here we know a lot about cellular information coding and basic processing steps. And yet, the programming models that arise from those “abstract machines” of biochemistry are unfamiliar ones. What I find even more remarkable is that cells have three separate Turing-complete (in principle) mechanisms at their disposal: proteins, genes, and membranes, which all often cooperate on information processing tasks.

## 1 Introduction

Following the discovery of the structure of DNA, just over 50 years ago, molecular biologists have been unraveling the functioning of cellular components and networks. The amount of molecular-level knowledge accumulated so far is absolutely amazing. And yet we cannot say that we understand *how a cell works*, at least not to the extent of being able to easily modify or repair a cell. The process of understanding cellular components is far from finished, but it is becoming clear that simply obtaining a full part list will not tell us how a cell works. Rather, even for substructures that have been well characterized, there are significant difficulties in understanding how components interact as *systems* to produce the observed behaviors. Moreover, there are just too many components, and too few biologists, to analyze

---

<sup>1</sup> Published in Transactions on Computational Systems Biology. III, LNBI 3737, pp 145-168, © Springer-Verlag Berlin Heidelberg 2005. With kind permission of Springer Science and Business Media.

each component in depth in reasonable time. Similar problems occur also at each level of biological organization above the cellular level.

Enter *systems biology*, which has two aims. The first is to obtain massive amounts of information about whole biological systems, via high-throughput experiments that provide relatively shallow and noisy data. The Human Genome Project is a prototypical example: the knowledge it accumulated is highly valuable, and was obtained in an automated and relatively efficient way, but is just the beginning of understanding the human genome. Similar effort are now underway in *genomics* (finding the collection of all genes, for many genomes), in *transcriptomics* (the collection of all actively transcribed genes), in *proteomics* (the collection of all proteins), and in *metabolomics* (the collection of all metabolites). *Bioinformatics* is the rapidly growing discipline tasked with collecting and analyzing such *omics* data.

The other aim of systems biology is to build, with such data, a science of the *principles of operation* of biological systems, based on the *interactions between components*. Biological systems are obviously well-engineered: they are very complex and yet highly structured and robust. They have only one major engineering defect: they have not been designed, in any standard sense, and so are not laid out as to be easily understood. It is not clear that any of the engineering principles of operations we are currently familiar with are fully applicable. Understanding such principles will require an interdisciplinary effort, using ideas from physics, mathematics, and computing. These, then, are the promises of systems biology: it will teach us new principles of operation, likely applicable to other sciences, and it will leverage other sciences to teach us *how cells work* in an actionable way.

In this paper, we look at the organization of biological systems from an information science point of view. The main reason is quite pragmatic: as we increasingly map out and understand the complex interactions of biological components, we need to *write down* such knowledge, in such a way that we can inspect it, animate it, and understand its principles. For genes, we can write down long but structurally simple strings of nucleotides in a 4-letter alphabet, that can be stored and queried. For proteins we can write down strings of amino acids in a 20-letter alphabet, plus three-dimensional information, which can be stored and queried with a little more difficulty. But how shall we write down *biological processes*, so that they can be stored and queried? It turns out that biologists have already developed a number of informal notation, which will be our starting points. These notations are abstractions over chemistry or, more precisely, are abstractions over a number of biologically relevant chemical toolkits.

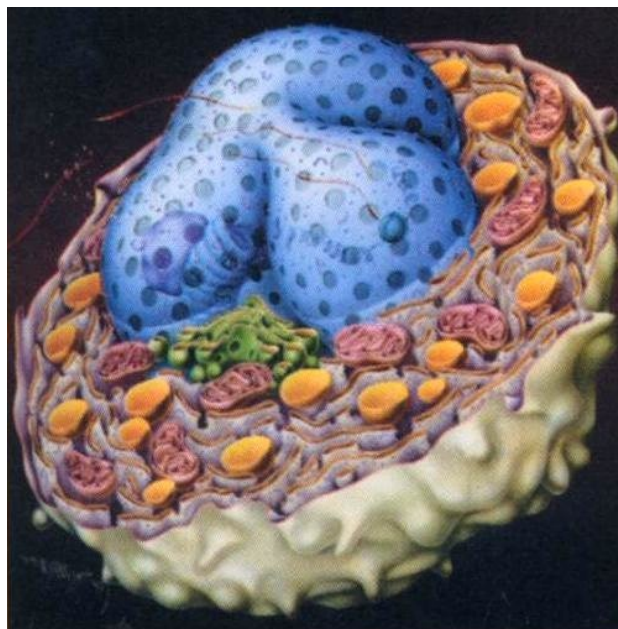
## 2 Biochemical Toolkits

Apart from small molecules such as water and some metabolites, there are four large classes of *macromolecules* in a cell. Each class is formed by a small number of units that can be combined systematically to produce structures of great complexity. That is, to produce both individual molecules of essentially unbounded size, and multi-molecular complexes.

The four classes of macromolecules are as follows. Different members of each class can have different functions (structure, energy storage, etc.). We focus on the most combinatorial, information-bearing, members of each class:

- *Nucleic acids*. Five kinds of *nucleotides* combine in ordered sequences to form two nucleic acid polymers: *DNA* and *RNA*. As data structures, RNA is **lists**, and DNA is **doubly-linked lists**. Their most prominent role is in coding information, although they also have other important functions.

- *Proteins*. About 20 kinds of *amino acids* combine linearly to form proteins. Each protein folds in a specific three-dimensional shape (sometimes from multiple strings of amino acids). The main and most evolutionary stable property of a protein is not the exact sequence of amino acids that make it up, nor the exact folding process, but its collection of surface *features* that determine its function. As data structures, proteins are **records** of features and, since these features are often active and stateful, they are **objects** in the object-oriented programming sense.
- *Lipids*: Among the lipids, *phospholipids* have a modular structure and can self-assemble into closed double-layered sheets (membranes). Membranes differ in the proportion and orientation of different phospholipids, and in the kinds of proteins that are attached to them. As data structures, membranes are **containers**, but with an active surface that acts as an **interface** to its contents.
- *Carbohydrates*: Among the carbohydrates, *oligosaccharides* are sugars linked in a branching structure. As data structures, oligosaccharides are **trees**. They have a vast number of configurations, and a complex assembly processes. *Polysaccharides* form even bigger structures, although usually of a semi-regular kind (rods, meshes). We do not consider carbohydrates further, although they are probably just as rich and interesting as the other toolkits. They largely have to do with energy storage and with cell surface and extracellular structures. But it should be noted that they too have a computational role, in forming unique surface structures that are subject to recognition. Many proteins are grafted with carbohydrates, through a complex assembly process called glycosylation.



From *MOLECULAR CELL BIOLOGY, 4/e* by Harvey Lodish, et. al. ©1986, 1990, 1995, 2000 by W.H. Freeman and Company. Figure 1-1, Page 1. Used with permission.

**Figure 1 Eukaryotic Cell**

Eukaryotic cells have an extensive array of membrane-bound compartments and organelles with up to 4 levels of nesting. The nucleus is a double membrane. The external membrane is less than 10% of the total.

Out of these four toolkits arises all the *organic chemicals*, composing, e.g., eukaryotic cells (Figure 1, [32] p.1). Each toolkit has specific structural properties (as emphasized by the bolded words above), systematic functions, and a peculiarly rich and flexible mode of operation. These peculiar modes of operation and systematic functions are what we want to emphasize, beyond their chemical realization.

Cells are without doubt, in many respects, information processing devices. Without properly processing information from their environment, they soon die for lack of nutrients or for predation. The blueprint of a cell, needed for its functioning and reproduction, is stored as digital information in the genome; an essential step of reproduction is the copying of that digital information. There are hints that information processing in the genome of higher organisms is much more sophisticated than currently generally believed [33].

We could say that cells are based on chemistry that also perform some information processing. But we take a more extreme position, namely that cells are chemistry *in the service* of information processing. Hence, we look for information processing machinery within the cellular machinery, and we try to understand the functioning of the cell in terms of information processing, instead of chemistry. In fact, we can readily find such information processing machinery in the chemical toolkits that we just described, and we can switch fairly smoothly from the classical description of cellular functioning in terms of classes of macromolecules, to a description based on abstract information-processing machines.

### 3 Abstract Machines

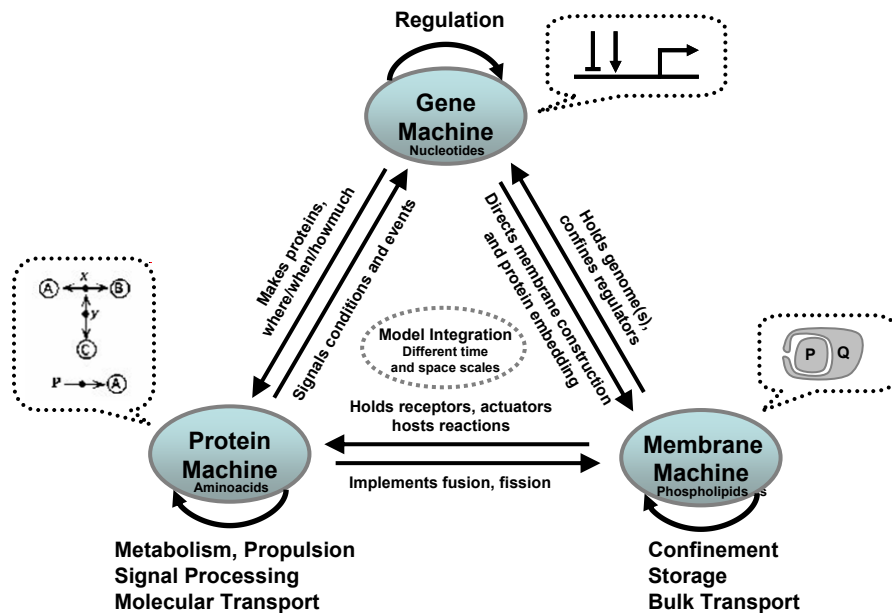
An *abstract machine* is a fictional information-processing device that can, in principle, have a number of different physical realizations (mechanical, electronic, biological, or even software). An abstract machine is characterized by:

- A collection of discrete states.
- A collection of operations (or events) that cause discrete transitions between states.

The evolution of states through transitions can in general happen concurrently. The adequacy of this generic model for describing complex systems is argued, e.g., in [22].

Each of the chemical toolkits we have just described can be seen as a separate abstract machine with an appropriate set of states and operations. This abstract interpretations of chemistry is by definition fictional, and we must be aware of its limitation. However, we must also be aware of the limitations of *not* abstracting, because then we are in general limited to work at the lowest level of reality (quantum mechanics) without any hope of understanding higher principles of organization. The abstract machines we consider are each grounded in a different chemical toolkit (nucleotides, amino acids, and phospholipids), and hence have some grounding in reality. Moreover, each abstract machine corresponds to a different kind of informal *algorithmic notation* that biologists have developed (Figure 2, bubbles): this is further evidence that abstract principles of organization are at work.

The *Gene Machine* (better known as Gene Regulatory Networks) performs information processing tasks within the cell. It regulates all other activities, including assembly and maintenance of the other machines, and the copying of itself. The *Protein Machine* (better known as Biochemical Networks) performs all mechanical and metabolic tasks, and also some signal processing. The *Membrane Machine* (better known as Transport Networks) separates different biochemical environments, and also operates dynamically to transport substances via complex, discrete, multi-step processes.



**Figure 2 Abstract Machines, Molecular Basis, and Notations**

These three machines operate in concert and are highly interdependent. Genes instruct the production of proteins and membranes, and direct the embedding of proteins within membranes. Some proteins act as messengers between genes, and others perform various gating and signaling tasks when embedded in a membrane. Membranes confine cellular materials and bear proteins on their surfaces. In eukaryotes, membranes confine the genome, so that local conditions are suitable for regulation, and confine other reactions carried out by proteins in specialized vesicles.

Therefore, to understand the functioning of a cell, one must understand also how the various machines interact. This involves considerable difficulties (e.g. in simulations) because of the drastic difference in time and size scales: proteins interact in tiny fractions of a second, while gene interactions take minutes; proteins are large molecules, but are dwarfed by chromosomes, and membranes are larger still. Before looking at the interactions among the different machine in more detail, we start by discussing each machine separately.

## 4 The Protein Machine (Biochemical Networks)

### 4.1 Principles of Operation

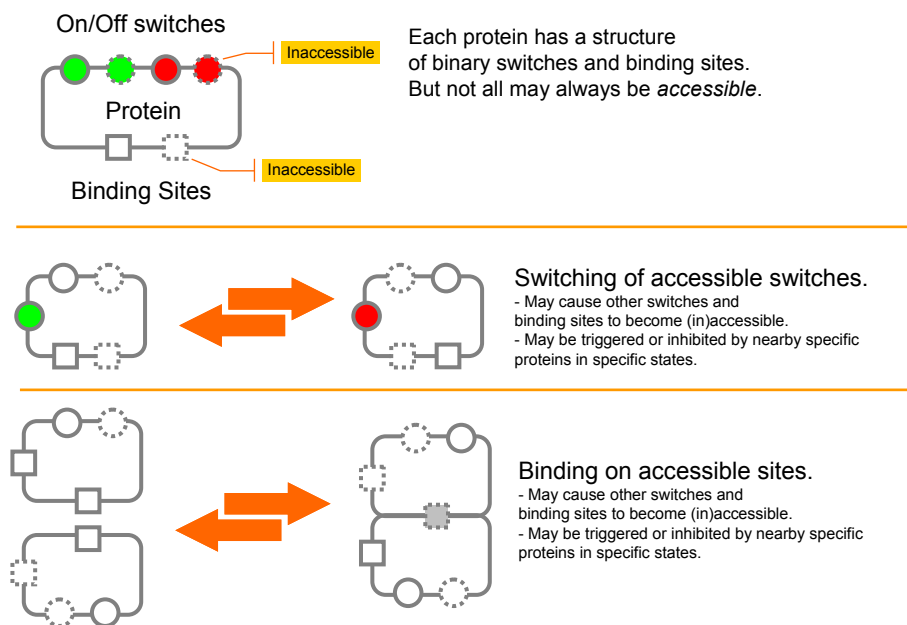
Proteins are long folded-up strings of amino acids with precisely determined, but often mechanically flexible, three-dimensional shapes. If two proteins have surface regions that are complementary (both in shape and in charge), they may stick to each other like Velcro, forming a protein **complex** where a multitude of small atomic forces creates a strong bond between individual proteins. They can similarly stick highly selectively to other substances. During a **complexation** event, a protein may be bent or opened, thereby revealing new interaction surfaces. Through complexation many proteins act as enzymes: they bring together compounds, including other proteins, and greatly facilitate chemical reactions between them without being themselves affected.

Proteins may also chemically modify each other by attaching or removing small phosphate groups at specific sites. Each such site acts as a boolean switch: over a dozen of them can be present on a single protein. Addition of a phosphate group (**phosphorylation**) is performed by an enzyme that is then called a **kinase**. Removal of a phosphate group (**dephosphorylation**) is performed by an enzyme that is then called a **phosphatase**. For example, a *protein phosphatase kinase kinase* is a protein that phosphorylates a protein that phosphorylates a protein that dephosphorylates a protein. Each (de-)phosphorylation may reveal new interaction surfaces, and each surface interaction may expose new phosphorylation sites.

It turns out that a large number of protein interactions work at the level of abstraction just described. That is, we can largely ignore chemistry and the protein folding process, and think of each protein as a collection of features (binding sites and phosphorylation sites) whose availability is affected by (de-)complexation and (de-)phosphorylation interactions. This abstraction level is emphasized in Kohn's Molecular Interaction Maps graphical notation [29][27] (Figure 4).

We can describe the operation of the *protein machine* as follows (Figure 3). Each protein is a collection of *sites* and *switches*; each of those can be, at any given time, either *available* or *unavailable*. Proteins can join at matching sites, to form bigger and bigger *complexes*. The availability of sites and switches in a complex is the *state* of the complex. A *system* is a multiset of (disjoint) complexes, each in a given state.

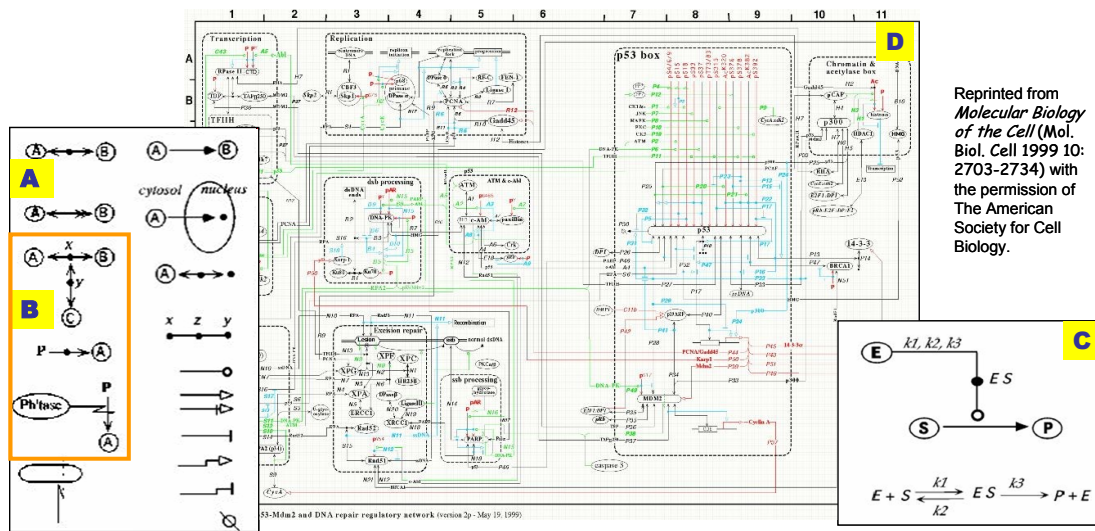
The protein machine has two kinds of operations. (1) An available switch on a complex can be turned on or off, resulting in a new state where a new collection of switches and sites is available. (2) Two protein complexes can combine at available sites, or one complex can split into two, resulting in a new state where a new collection of switches and sites is available.



**Figure 3 The Protein Machine Instruction Set**

Who is driving the switching and binding? Other proteins do. There are tens of thousands of proteins in a cell, so the protein machine has tens of thousands of “primitive instructions”; each with a specific way of acting on other proteins (or metabolites). For each cellular

subsystem one must list the proteins involved, and how each protein interacts with the other proteins in terms of switching and binding.



Reprinted from  
*Molecular Biology  
of the Cell* (Mol.  
Biol. Cell 1999 10:  
2703-2734) with  
the permission of  
The American  
Society for Cell  
Biology.

**Figure 4 Molecular Interaction Maps Notation**

From [29]. **A**: graphical primitives. **B**: complexation and phosphorylation. **C**: enzymatic diagram and equivalent chemical reactions. **D**: map of the p53-Mdm2 and DNA Repair Regulatory Network.

## 4.2 Notations

Finding a suitable language in which to cast such an abstraction is a non-trivial task. Kohn designed a graphical notation, resulting in pictures such as Figure 4 [29]. This was a tremendous achievement, summarizing hundreds of technical papers in page-sized pictures, while providing a sophisticated and expressive notation that could be translated back into chemical equations according to semi-formal guidelines. Because of this intended chemical semantics, the dynamics of a systems is implied in Kohn's notation, but only by translation to chemical (and hence kinetic) equations. The notation itself has no dynamics, and this is one of its main limitation. The other major limitation is that, although graphically appealing, it tends to stop being useful when overflowing the borders of a page or of a whiteboard (the original Kohn maps span several pages).

Other notations for the protein machine can be devised. Kitano, for example, improved on the conciseness, expressiveness, and precision of Kohn's notation [28], but further sophistication in graphical notation is certainly required along the general principles of [18]. A different approach is to devise a textual notation, which inherently has no "page-size" limit and can better capture dynamics; examples are Bio-calculus [38], and most notably  $\kappa$ -calculus [14][15], whose dynamics is fully formalized. But one may not need to invent completely new formalisms. Regev and Shapiro, in pioneering work [49][47], described how to represent chemical and biochemical interactions within existing process calculi ( $\pi$ -calculus). Since process calculi have a well understood dynamics (better understood, in fact, than most textual notations that one may devise just for the purpose), that approach also provides a solid basis for studying systems expressed in such a notation. Finally, some notations incorporate both continuous and discrete aspects, as in Charon [3] and dL-systems [45].

### 4.3 Example: MAPK Cascade

The relatively simple Kohn map in Figure 5 (adapted from [25]) describes the behavior of a circuit that causes Boolean-like switching of an output signal in presence of a very weak input signal. (It can also be described as a list of 10 chemical reactions, or of 25 differential/algebraic equations, but then the network structure is not so apparent.) This network, generically called a MAPK cascade, has multiple biochemical implementations and variations. The components are proteins (enzymes, kinases, phosphatases, and intermediaries). The circle-arrow Kohn symbol for “enzyme-assisted reaction” can signify here either a complexation that facilitates a reaction, or a phosphorylation/dephosphorylation, depending on the specific proteins.

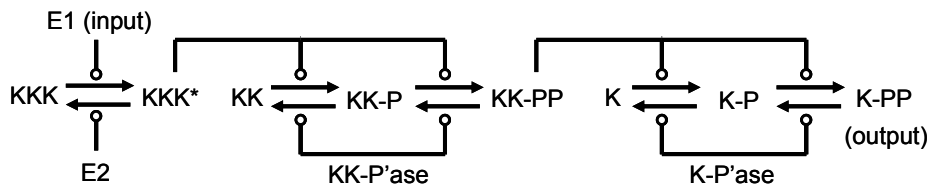


Figure 5 MAPK Cascade

The system initially contains reservoirs of chemicals  $KKK$ ,  $KK$ , and  $K$  (say, 100 molecules each), which are transformed by the cascade into the kinases  $KKK^*$ ,  $KK-PP$ , and  $K-PP$  respectively. Enzymes  $E2$ ,  $KK$ -Phosphatase and  $K$ -Phosphatase are always available (say, 1 molecule each), and tend to drive the reactions back. Appearance of the input enzyme  $E1$  in very low numbers (say, less than 5) causes a sharp (Boolean-like 0-100) transition in the concentration of the output  $K-PP$ . The concentrations of the intermediaries  $KK-PP$ , and especially  $KKK^*$ , raise in a much smoother, non-Boolean-like, fashion. Given the mentioned concentrations, the network works fine by setting all reaction rates to equal values.

To notice here is that the detailed description of each of the individual proteins, with their folding processes, surface structures, interaction rates under different conditions, etc. could take volumes. But what makes this signal processing network work is the structure of the network itself, and the relatively simple interactions between the components.

### 4.4 Summary

The fundamental flavor of the Protein Machine is: fast synchronous binary interactions. Binary because interactions occur between two complementary surfaces, and because the likelihood of three-party instantaneous chemical interactions can be ignored. Synchronous because both parties potentially feel the effect of the interaction, when it happens. Fast because individual chemical reactions happen at almost immeasurable speeds. The parameters affecting reaction speed, in a well-stirred solution, are just a reaction-specific rate constant having to do with surface affinity, plus the concentrations of the reagents (and the temperature of the solution, which is usually assumed constant). Concentration affects the likelihood of molecules randomly finding each other by Brownian motion. Note that Brownian motion is surprisingly effective at a cellular scale: a molecule can “scan” the equivalent volume of a bacteria for a match in 1/10 of a second, and it will in fact scan such a bounded volume because random paths in 3D do not return to the origin.



## 5 The Gene Machine (Gene Regulatory Networks)

### 5.1 Principles of Operation

The *central dogma of molecular biology* states that DNA is transcribed to RNA, and RNA is translated to proteins (and then proteins do all the work). This dogma no longer paints the full picture, which has become considerably more detailed in recent years. Without entering into a very complex topic [33], let us just note that some proteins go back and bind to DNA. Those proteins are called **transcription factors** (either **activators** or **repressors**); they are produced for the purpose of allowing one gene (or signaling pathway) to communicate with other genes. Transcription factors are not simple messages: they are proteins, which means they are subject to complexation, phosphorylation, and programmed degradation, which all have a role in gene regulation.

A **gene** is a stretch of DNA consisting of two (not necessarily contiguous or unbroken) regions: an *input (regulatory) region*, containing **protein binding sites** for transcription factors, and an *output (coding) region*, coding for one or more proteins that the gene produces. Sometimes there are two coding regions, in opposite directions [46], on count of DNA being a doubly-linked list. Sometimes two genes overlap on the same stretch of DNA.

The output region functions according to the **genetic code**: a well understood and almost universal table mapping triplets of nucleotides to one of about 20 amino acids, plus start and stop triplets. The input region functions according to a much more complex code that is still poorly understood: transcription factors, by their specific 3D shapes, bind to specific nucleotide sequences in the input region, with varying binding strength depending of the precision of the match.

Thus, the gene machine, although entirely determined by the digital information coded in DNA, is not entirely digital in functioning: a digitally encoded protein, translated and folded-up, uses its “analog” shape to recognize another digital string and promote the next step of translation. Nonetheless, it is customary to ignore the details of this process, and simply measure the effectiveness with which (the product of) a gene affects another gene. This point of view is reflected in standard notation for gene regulatory networks (Figure 7).

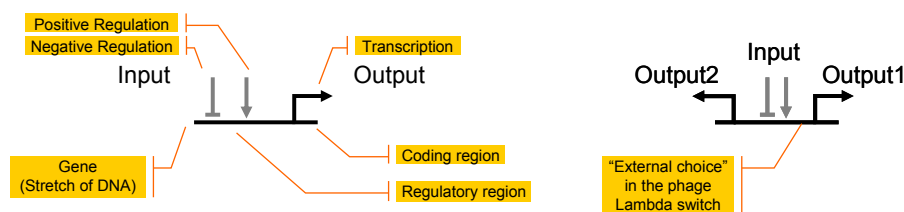


Figure 6 The Gene Machine Instruction Set

In Figure 6, a gene is seen as a hardware gate, and the genome can be seen as a vast circuit composed of such gates. Once the performance characteristics of each gate is understood, one can understand or design circuits by combining gates, almost as one would design digital or analog hardware circuits. The performance characteristics of each gene in a genome is probably unique. Hence, as in the protein machine, we are going to have thousands of “primitive instructions”: one for each gene.

A peculiarity of the gene machine is that a set of gates also determines the network connectivity. This is in contrast with a hardware circuit, where there is a collection of gates out of a very small set of “primitive gates”, and then a separate wiring list. Each gene has a

*fixed output*; the protein the gene codes for (although post-processing may vary such output). Similarly, a gene has a *fixed input*: the fixed set of binding sites in the input region. Therefore, by knowing the nucleotide sequence of each gene in a genome, one (in principle) also knows the network connectivity without further information. This situation is similar to a software assembly-language program: “Line 3: Goto Line 5” where both the input and output addresses are fixed, and the flow graph is determined by the instructions in the program. However, a further difference is that the output of a gene is not the “address” of another gene: it is a protein that can bind with varying strength to a number of other genes.

The state of a gene machine is the concentrations of the transcription factors produced by each gene (or arriving from the environment). The operations, again, are the input-output functions of each gene. But what is the “execution” of a gene machine? It is not as simple as saying that one gene stimulates or inhibits another gene. It is known that certain genes perform complex computations on their inputs that are a mixture of boolean, analog, and multi-stage operators (Figure 7-B [54]). Therefore, the input region of each gene can itself be a sophisticated machine.

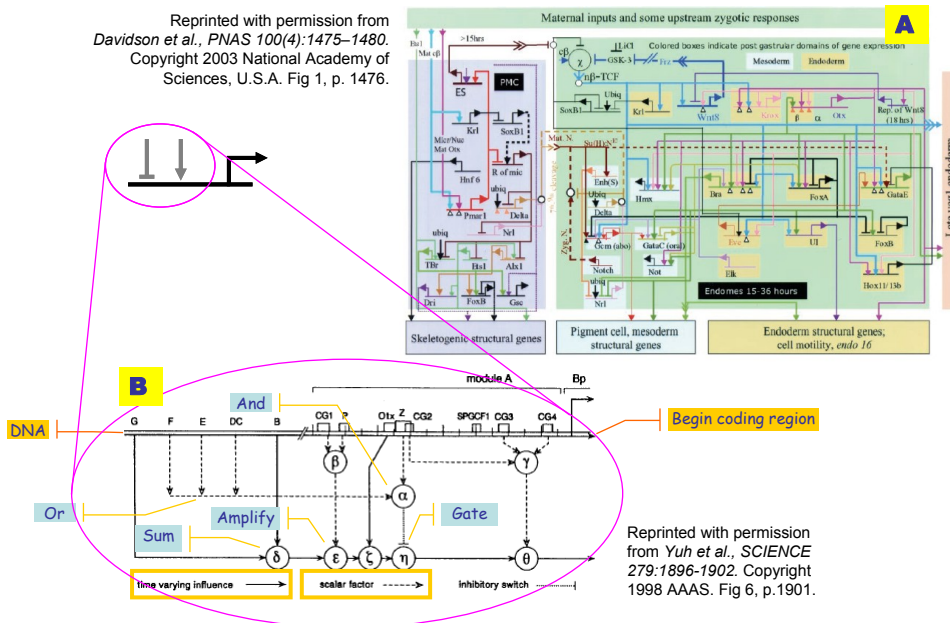
Whether the execution of a gene machine should be seen as a continuous or discrete process, both in time and in concentration levels, is already a major question. Qualitative models (e.g.: random and probabilistic Boolean networks [26][50], asynchronous automata [52], network motifs [36]) can provide more insights than quantitative models, whose parameters are hard to come by and are possibly not critical. On the other hand, it is understood that pure Boolean models are inadequate in virtually all real situations. Continuous, stochastic, and decay aspect of transcription factor concentrations are all critical in certain situations [34][53].

## 5.2 Notations

Despite all these difficulties and uncertainties, a single notation for the gene machine is in common use, which is the gene network notation of Figure 7-A. There, the gates are connected by either “excitatory” (pointed arrow) or “inhibitory” (blunt arrow) links. What such relationships might mean is often left unspecified, except that, in a common model, a single constant weight is attached to each link.

Any serious publication would actually start from a set of ordinary differential equations relating concentrations of transcription factors, and use pictures such as Figure 7-A only for illustration, but this approach is only feasible for small networks. The best way to formalize the *notation* of gene regulatory networks is still subject to debate and many variations, but there is little doubt that formalizing such a notation will be essential to get a grasp on gene machines the size of genomes (the smallest of which, *M.Genitalium*, is on the order of 150 Kilobytes, and one closer to human cellular organization, Yeast, is 3 Megabytes).

Reprinted with permission from Davidson et al., PNAS 100(4):1475-1480. Copyright 2003 National Academy of Sciences, U.S.A. Fig 1, p. 1476.

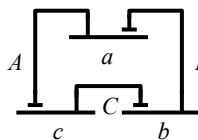


**Figure 7 Gene Regulatory Networks Notation**

**A** [16]: gene regulatory network involved in sea urchin embryo development: **B** [54]: boolean/arithmetic diagram of module A, the last of 6 interlinked modules in the regulatory region of the *endo16* sea urchin gene; G,F,E,DC,B are module outputs feeding into A, the whole region is 2300 base pairs.

### 5.3 Example: Repressilator

The circuit in Figure 8, artificially engineered in *E. coli* bacteria [19], is a simple oscillator (given appropriate parameters). It is composed of three genes with single input that inhibit each other in turn. The circuit gets started by *constitutive transcription*: each gene autonomously produces output in absence of inhibition, and the produced output decays at a certain stochastic rate. The symmetry of the circuit is broken by the underlying stochastic behavior of chemical reactions. Its behavior can be understood as follows. Assume that gene *a* is at some point not inhibited (i.e. the product *B* of gene *b* is absent). Then gene *a* produces *A*, which shuts down gene *c*. Since gene *c* is no longer producing *C*, gene *b* eventually starts producing *B*, which shuts down gene *a*. And so on.



**Figure 8 Repressilator Circuit**

### 5.4 Summary

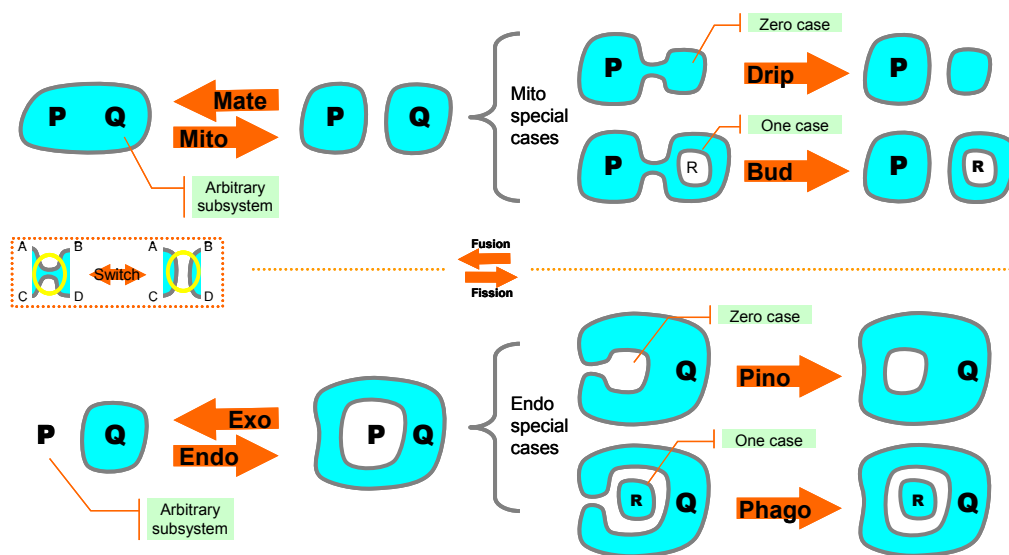
The fundamental flavor of the Gene Machine is: slow asynchronous stochastic broadcast. The interaction model is really quite strange, by computing standards. Each gene has a fixed output, which is not quite an address for another gene: it may bind to a large number of other genes, and to multiple locations on each gene. The transcription factor is produced in great quantities, usually with a well-specified time-to-live, and needs to reach a certain threshold to have an effect. On the other hand, various mechanisms can guarantee Boolean-like switching when the threshold is crossed, or, very importantly, when a message is *not* received.

Activation of one gene by another gene is slow by any standard: typically one to five minutes, to build up the necessary concentration<sup>2</sup>. However, the genome can slowly direct the assembly-on-need of protein machines that then act fast: this “swap time” is seen in experiments that switch available nutrients. The stochastic aspect is fundamental because, e.g., with the same parameters, a circuit may oscillate under stochastic/discrete semantics, but not under deterministic/continuous semantics [53]. One reason is that a stochastic system may decay to *zero* molecules of a certain kind at a given time, and this can cause switching behavior, while a continuous system may asymptotically decay only to a non-zero level.

## 6 The Membrane Machine (Transport Networks)

### 6.1 Principles of Operation

A cellular membrane is an oriented closed surface that performs various molecular functions. Membranes are not just containers: they are coordinators and sites of major activity<sup>3</sup>. Large functional molecules (proteins) are embedded in membranes with consistent orientation, and can act on both sides of the membrane simultaneously. Freely floating molecules interact with membrane proteins, and can be sensed, manipulated, and pushed across by active molecular channels. Membranes come in different kinds, distinguished mostly by the proteins embedded in them, and typically consume energy to perform their functions. The consistent orientation of membrane proteins induces an orientation on the membrane.



**Figure 9 The Membrane Machine Instruction Set (2D)**

One of the most remarkable properties of biological membranes is that they form a two-dimensional fluid (a lipid bilayer) embedded in a three-dimensional fluid (water). That is, both the structural components and the embedded proteins freely diffuse on the two-dimensional plane of the membrane (unless they are held together by specific mechanisms).

<sup>2</sup> Consider that bacteria replicate in only 20 minutes while cyclically activating hundreds of genes. It seems that, at least for bacteria, the gene machine can make “wide” but not very “deep” computations [36].

<sup>3</sup> “For a cell to function properly, each of its numerous proteins must be localized to the correct cellular membrane or aqueous compartment.” [32] p.675.

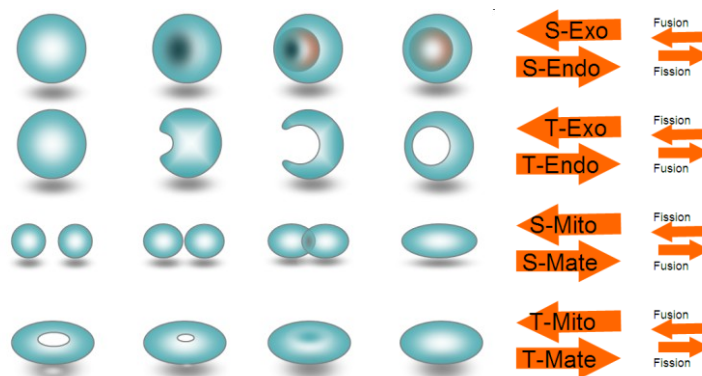
Moreover, membranes float in water, which may contain other molecules that freely diffuse in that three-dimensional fluid. Membranes themselves are impermeable to most substances, such as water and protons, so that they partition the three-dimensional fluid. This organization provides a remarkable combination of freedom and structure.

Many membranes are highly dynamic: they constantly shift, merge, break apart, and are replenished. But the transformations that they support are naturally limited, partially because membranes must preserve their proper orientation, and partially because membrane transformations need to be locally-initiated and continuous. For example, it is possible for a membrane to gradually buckle and create a bubble that then detaches, or for such a bubble to merge back with a membrane. But it is not possible for a bubble to “jump across” a membrane (only small molecules can do that), or for a membrane to turn itself inside-out.

The basic operations on membranes, implemented by a variety of molecular mechanisms, are *local fusion* (two patches merging) and *local fission* (one patch splitting in two) [8]. We discuss first the 2D case, which is instructive and for which there are some formal notations, and then the 3D case, the real one for which there are no formal notations.

In two dimensions (Figure 9), at the local scale of membrane patches, fusion and fission become indistinguishable as a single operation, *switch*, that takes two membrane patches, i.e. to segments A-B and C-D, and switches their connecting segments into A-C and B-D (crossing is not allowed). We may say that, in 2D, a switch is a fusion when it decreases the number of whole membranes, and is a fission when it increases such number.

When seen on the global scale of whole 2D membranes, switch induces four operations: in addition to the obvious splitting (Mito) and merging (Mate) of membranes, there are also operation, quite common in reality, that cause a membrane to “eat” (Endo) or “spit” (Exo) another subsystem (P). There are common special cases of Mito and Endo, when the subsystem P consists of zero (Drip, Pino) or one (Bud, Phago) membranes. All these operations *preserve bitonality* (dual coloring); that is, if a subsystem P is on a dark (or light) background before a reaction, it will be on a dark (or light) background after the reaction. Bitonality is related to preservation of membrane orientation, and to locality of operations (a membrane jumping across another one does not preserve bitonality). Bitonal operations ensure that what is or was *outside* the cell (light) never gets mixed with what is *inside* (dark). The main reactions that violate bitonality are destructive and non-local ones (such a digestion, not shown). Note that Mito/Mate preserve the nesting depth of subsystems, and hence they cannot encode Endo/Exo; instead, Endo/Exo can encode Mito/Mate [12].



**Figure 10 The Membrane Machine Instruction Set (3D)**

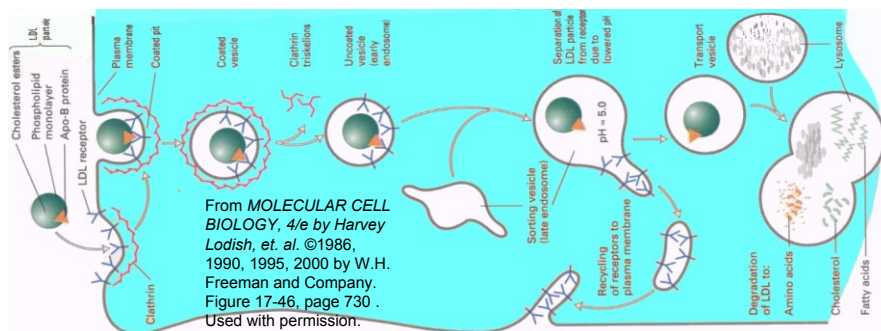
Each row consists of initial state, two intermediate states, and final state (and back).

In three dimensions, the situation is more complex (Figure 10). There are 2 distinct local operations on surface patches, inducing 8 distinct global operations that change surface topology. *Fusion* joins two Positively curved patches (in the shapes of domes) into one Negatively curved patch (in the shape of a hyperbolic cooling tower) by allowing the P-patches to kiss and merge. *Fission* instead splits one N-patch into two P-patches by pinching the N-patch. Fusion does not necessarily decrease the number of membranes in 3D (it may turn a sphere into a torus in two different ways: T-Endo T-Mito), and Fission does not necessarily increase the number of membranes (it may turn a torus into a sphere in two different ways: T-Exo, T-Mate). In addition, Fusion may merge two spheres into one sphere in two different ways (S-Exo, S-Mate), and Fission may split one sphere into two spheres in two different ways (S-Endo, S-Mito). Note that S-Endo and T-Endo have a common 2D cross section (Endo), and similarly for the other three pairs.

Cellular structures have very interesting dynamic topologies: the eukaryotic nuclear membrane, for example, is two nested spheres connected by multiple toroidal holes (and also connected externally to the Endoplasmic Reticulum). This whole structure is disassembled, duplicated, and reassembled during cellular mitosis. Developmental processes based on cellular differentiation are also within the realm of the Membrane Machine, although geometry, in addition to topology, is an important factor there.

## 6.2 Notations

The informal notation used to describe executions of the Membrane Machine does not really have a name, but can be seen in countless illustrations (e.g., Figure 11, [32] p.730). All the stages of a whole process are summarized in a single snapshot, with arrows denoting operations (Endo/Exo etc.) that cause transitions between states. This kind of depiction is natural because often all the stages of a process *are* observed at once, in photographs, and much of the investigation has to do with determining their proper sequence and underlying mechanisms. These pictures are usually drawn in two colors, which is a hint of the semantic invariant we call bitonality.



**Figure 11 Transport Networks Notation**

LDL particle (left) is recognized, ingested, and transported to a lysosome vesicle (right). [32], p.730.

Some membrane-driven processes are semi-regular, and tend to return to something resembling a previous configuration, but they are also stochastic, so no static picture or finite-state-automata notation can tell the real story. Complex membrane dynamics can be found in the protein secretion pathway, through the Golgi system, and in many developmental processes. Here too there is a need for a precise dynamic notation that goes beyond static pictures; currently, there are only a few such notations [42][48][12].

### 6.3 Example: LDL Cholesterol Degradation

The membrane machine runs real algorithms: Figure 11 depicts LDL-cholesterol degradation. The “problem” this algorithm solves is to transport a large object (an LDL particle) to an interior compartment where it can be degraded; the particle is too big to just cross the membrane. The “solution”, by a precise sequence of discrete steps and iterations, utilizes proteins embedded in the external cellular membrane and in the cytosol to recognize, bind, incorporate, and transport the particle inside vesicles to the desired compartment, all along recycling the active proteins.

### 6.4 Summary

The fundamental flavor of the Membrane Machine is: fluid-in-fluid architecture, membranes with embedded active elements, and fusion and fission of compartments preserving bitonality. Although dynamic compartments are common in computing, operations such as endocytosis and exocytosis have never explicitly been suggested as fundamental. They embody important invariants that help segregate cellular materials from environmental materials. The distinction between active elements *embedded* on the surface of a compartment, vs. active elements *contained* in the compartment, becomes crucial with operations such as Exo. In the former case, the active elements are retained, while in the latter case they are lost to the environment.

## 7 Three Machines, One System

### 7.1 Principles of Operation

We have discussed how three classes of chemicals, among others, are fundamental to cellular functioning: nucleotides (nucleic acids), amino acids (proteins), and phospholipids (membranes). Each of our abstract machines is based primarily on one of these classes of chemicals: amino acids for the protein machine, nucleotides for the gene machine, and phospholipids for the membrane machine.

These three classes of chemicals are however heavily interlinked and interdependent. The gene machine “executes” DNA to produce proteins, but some of those proteins, which have their own dynamics, are then used as control elements of DNA transcription. Membranes are fundamentally sheets of pure phospholipids, but in living cells they are heavily doped with embedded proteins which modulate membrane shape and function. Some protein translation happens only through membranes, with the RNA input on one side, and the protein output on the other side or threaded into the membrane.

Therefore, the abstract machines are interlinked as well, as illustrated in Figure 2. Ultimately, we will need a single notation in which to describe all three machines (and more), so that a whole organism can be described.

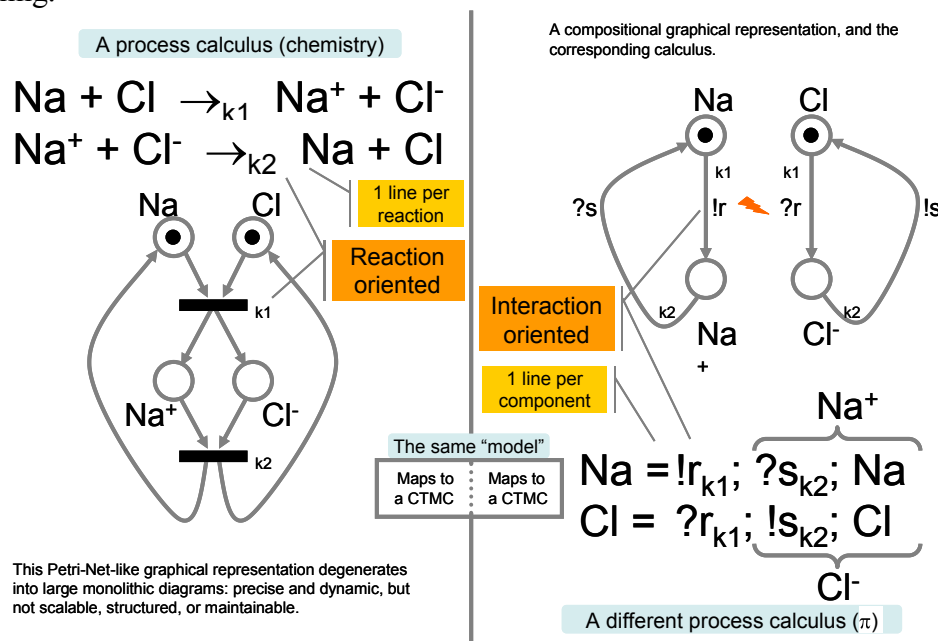
### 7.2 Notations

What could a single notation for all three machines (and more) look like? All formal notations known to computing, from Petri Nets to term-rewriting systems, have already been used to represent aspects of biological systems; we shall not even attempt a review here. But none, we claim, has shown the breadth of applicability and scalability of process calculi [35], partially because they are not a single notation, but a coherent conceptual framework in which one can derive suitable notations. There is also a general theory and notation for such calculi [37], which can be seen as the formal umbrella under which to unify different abstract machines.

Major progress in using process calculi for describing biological systems was achieved in Aviv Regev's Ph.D. thesis [47], where it is argued that one of the standard existing process calculi,  $\pi$ -calculus, enriched with a stochastic semantics [24][43][44], is extraordinarily suitable for describing both molecular-level interactions and higher levels of organization. The same stochastic calculus is now being used to describe genetic networks [30]. For membrane interactions, though, we need something beyond basic process calculi, which have no notion of compartments. Ambient Calculus [13] (which extends  $\pi$ -calculus with compartments) has been adapted [47][48] to represent biological compartments and complexes. A more recent attempt, Brane Calculus [12], embeds the biological invariants and 2D operations from Section 6.

These experiences point at process calculi as, at least, one of the most promising notational frameworks for unifying different aspects of biological representation. In addition, the process calculus framework is generally suitable for relating different levels of abstractions, which is going to be essential for feasibly representing biological systems of high architectural complexity.

Figure 12 gives a hint of the difference in notational approach between process calculi and more standard notations. Ordinary chemical reaction notation *is* a process calculus: it is a calculus of chemical processes. But it is a notation that focuses on reactions instead of components; this becomes a disadvantage when components have rich structure and a large state space (like proteins). In chemical notation one describes each state of a component as a different chemical species (Na, Na<sup>+</sup>), leading to a combinatorial blowup in the *description* of the system (the blowup carries over to related descriptions in terms of differential equations). In process calculus notation, instead, the components are described separately, and the reactions (occurring through complementary event pairs such as !r and ?r) come from the interactions of the components. Interaction leads to a combinatorial blowup in the *dynamics* of interactions, but not in the description of the systems, just like in ordinary object-oriented programming.



**Figure 12 Chemical vs. Process Calculi Notations**



On the left of Figure 12 we have a chemical description of a simple system of reactions, with a related (non-compositional) Petri Nets description. On the right we have a process calculus description of the same system, with a related (compositional) description in terms of interacting automata (e.g., Statecharts [22] with sync pseudostates). Both kinds of descriptions can take into account stochastic reaction rates ( $k_1, k_2$ ), and both can be mapped to the same stochastic model (Continuous-Time Markov Chains), but the descriptions themselves have different structural properties. From a simulation point of view, the left-hand-side approach leads to large sparse matrices of chemical species vs. chemical reactions, while the right-hand-side approach leads to large multisets of interacting objects.

### 7.3 Example: Viral Infection

The example in Figure 13 (adapted from [2], p.279) is the “algorithm” that a specific virus, the Semliki Forest virus, follows to replicate itself. It is a sequence of steps that involve the dynamic merging and splitting of compartments, the transport of materials, the operation of several proteins, and the interpretation of genetic information. The algorithm is informally described in English below. A concise description in Brane Calculus is presented in [12], which encodes the infection process at high granularity, but *in its entirety*, including the membrane, protein, and gene aspects.

A virus is too big to cross a cellular membrane. It can either punch its RNA through the membrane or, as in this example, it can enter a cell by utilizing standard cellular phagocytosis machinery. The virus consists of a capsid containing the viral RNA (the nucleocapsid). The nucleocapsid is surrounded by a membrane that is similar to the cellular membrane (in fact, it is obtained from it “on the way out”). This membrane is however enriched with a special protein that plays a crucial trick on the cellular machinery, as we shall see shortly.

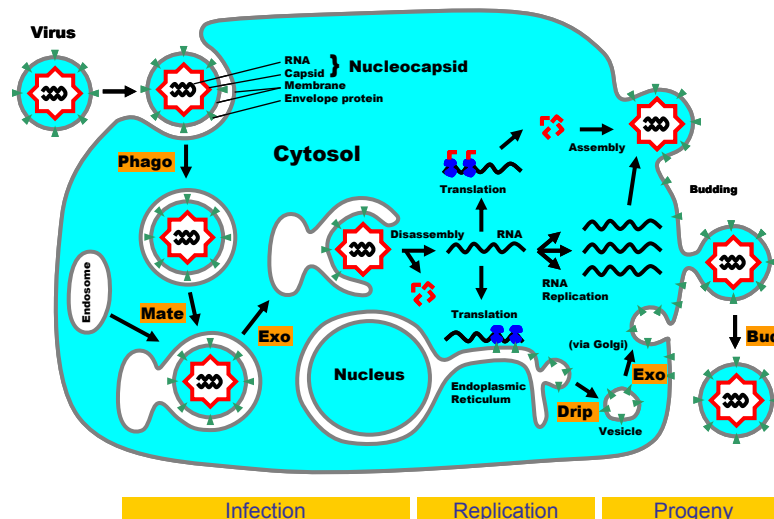


Figure 13 Viral Replication

*Infection:* The virus is brought into the cell by phagocytosis, wrapped in an additional membrane layer; this is part of a standard transport pathway into the cell. As part of that pathway, an endosome merges with the wrapped-up virus. At this point, usually, the endosome causes some reaction to happen in the material brought into the cell. In this case, though, the virus uses its special membrane protein to trigger an exocytosis step that deposits

the naked nucleocapsid into the cytosol. The careful separation of internal and external substances that the cell usually maintains has now been subverted.

*Replication:* The nucleocapsid is now in direct contact with the inner workings of the cell, and can begin doing damage. First, the nucleocapsid disassembles, depositing the viral RNA into the cytosol. This vRNA then follows three distinct paths. First it is replicated to provide the vRNA for more copies of the virus. The vRNA is also translated into proteins, by standard cellular machinery. The proteins forming the capsid are synthesized in the cytosol. The virus envelope protein is instead synthesized in the Endoplasmic Reticulum, and through various steps (through the Golgi apparatus) ends up lining transport vesicles that merge with the cellular membrane, along another standard transport pathway.

*Progeny:* In the cytosol, the capsid proteins self-assemble and incorporate copies of the vRNA to form new nucleocapsids. The newly assembled nucleocapsids make contact with sections of the cellular membrane that are now lined with the viral envelope protein, and bud out to recreate the initial virus structure outside the cell.

## 7.4 Summary

The fundamental flavor of cellular machinery is: chemistry in the service of materials, energy, and information processing. The processing of energy and materials (e.g., in metabolic pathways) need not be emphasized here, rather we emphasize the processing of information, which is equally vital for survival and evolution [1]. Information processing tasks are distributed through a number of interacting abstract machines with wildly different architectures and principles of operation.

## 8 Outlook: Model Construction and Validation

The biological systems we need to describe are massively concurrent, heterogeneous, and asynchronous: notoriously the hardest kinds of systems to cope with in programming. They have stochastic behavior and high resilience to drastic changes of environmental conditions. What organizational principles make these systems work reliably, and what conditions make them fail? These are the questions that computational modeling needs to answer.

There are two main aspects to modeling biological systems. *Model construction*, requires first an understanding of the principles of operation. This is what we have largely been discussing here: understanding the abstract machines of systems biology should lead us to formal notations that can be used to build (large, complex) biological models. But then there is *model validation*: a good scientific model has to be verified or falsified through postdiction and prediction. We briefly list different techniques that are useful for model validation, once a specific model has been written up in a specific precise notation.

*Stochastic simulation* of biochemical systems is a common technique, typically based on the physically well-characterized Gillespie algorithm [21], which originally was devised for reaction-oriented descriptions. The same algorithm can be used also for component-oriented (compositional) descriptions with a dynamically unbounded set of chemical species [44]. Stochastic simulation is particularly effective for systems with a relatively low number of interactions of any given kind, as is frequently the case in cellular-scale systems. It produces a single (high-likelihood) trace of the system for each run. It frequently reveals behavior that is difficult to anticipate, and that may not even correspond to continuous deterministic approximations [34]. It can be quantitatively compared with experiments.

*Static analysis* techniques of the kind common in programming can be applied to the description of biological systems [40]. Control-flow analysis and mobility analysis can reveal

subsystems that *cannot* interact [7][41]. Causality analysis can reconstruct the familiar network diagrams from process description [11]. Abstract interpretation can be used to study specific facets of a complex model [39], including probabilistic aspects [17].

**Modelchecking** is now used routinely in the analysis of hardware and software systems that have huge state spaces; it is based on the state and transition model we emphasized during the discussion of abstract machines. Modelchecking consists of a model description language for building models, a query language for asking questions about models (typically temporal logic), and an efficient state exploration engine. The basic technology is very advanced, and is beginning to be applied to descriptions of biological systems too, in various flavors. **Temporal** modelchecking asks qualitative questions such as whether the systems can reach a certain state (and how), or whether a state is a necessary checkpoint for reaching another state [9][20]. **Quantitative** modelchecking asks quantitative questions about, e.g., whether a certain concentration can eventually equal or double some other concentration in some state [4][6]. **Stochastic** modelchecking, based, e.g., on discrete or continuous-time Markov chain models, can ask questions about the probability of reaching a given state [31].

**Formal reasoning** is the most powerful and hardest technique to use, but already there is a long tradition of building tools for verifying properties of concurrent systems. Typical activities in this area are checking behavioral equivalence between different systems, or between different abstraction levels of the same system, including now biological systems [10][5].

While computational approaches to biology and other sciences are now common, several of the techniques outlined above are unique to computer science and virtually unknown in other fields; hopefully they will bring useful tools and perspectives to biology.

## 9 Conclusions

Many aspects of biological organization are more akin to discrete hardware and software systems than to continuous systems, both in hierarchical complexity and in algorithmic-like information-driven behavior. These aspects need to be reflected in the modeling approaches and in the notations used to describe such systems, in order to make sense of the rapidly accumulating experimental data.

*“The data are accumulating and the computers are humming, what we are lacking are the words, the grammar and the syntax of a new language...”*

Dennis Bray (TIBS 22(9):325-326, 1997)

*“The most advanced tools for computer process description seem to be also the best tools for the description of biomolecular systems.”*

Ehud Shapiro (Biomolecular Processes as Concurrent Computation, Lecture Notes, 2001)

*“Although the road ahead is long and winding, it leads to a future where biology and medicine are transformed into precision engineering.”*

Hiroaki Kitano (Nature 420:206-210, 2002)

*“The problem of biology is not to stand aghast at the complexity but to conquer it.”*

Sydney Brenner (Interview, Discover Vol. 25 No. 04, April 2004)

## References

- [1] C.Adami. **What is complexity?** BioEssays 24:1085–1094, Wiley, 2002.
- [2] B.Alberts, D.Bray, J.Lewis, M.Raff, K.Roberts, J.D.Watson. **Molecular biology of the cell**. Third Edition, Garland.
- [3] R.Alur, C.Belta, F.Ivancic, V.Kumar, M.Mintz, G.J.Pappas, H.Rubin, and J.Schug.. **Hybrid modeling of biomolecular networks**. Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control, Rome Italy, March 28-30, 2001. LNCS 2034.
- [4] M.Antoniotti, B.Mishra, F.Park, A.Policriti, N.Ugel. **Foundations of a query and simulation system for the modeling of biochemical and biological processes**. In L.Hunter, T.A.Jung, R.B.Altman, A.K.Dunker, T.E.Klein, editors, The Pacific Symposium on Biocomputing (PSB 2003) 116-127. World Scientific, 2003.
- [5] M.Antoniotti, C.Piazza, A.Policriti, M.Simeoni, B.Mishra. **Modeling cellular behavior with hybrid automata: bisimulation and collapsing**. In Int. Workshop on Computational Methods in Systems Biology (CMSB'03), LNCS. Springer, 2003. To appear.
- [6] M.Antoniotti, A.Policriti, N.Ugel, B.Mishra. **Model building and model checking for biochemical processes**. In Cell Biochemistry and Biophysics, 2003. In press.
- [7] C.Bodei, P.Degano, F.Nielson, H.R.Nielson. **Control flow analysis for the pi-calculus**. Proc. 9th International Conference on Concurrency Theory, LNCS 1466:84-98. Springer, 1998.
- [8] K.N.J. Burger. **Greasing membrane fusion and fission machineries**. Traffic 1: 605–613. 2000.
- [9] G.Ciobanu, V.Ciobotariu, B.Tanasa. **A  $\pi$ -calculus model of the Na pump**. Genome Informatics 13:469-471, 2002.
- [10] G.Ciobanu. **Software verification of biomolecular systems**. In G.Ciobanu, G.Rozenberg (Eds.): Modelling in Molecular Biology, Natural Computing Series, Springer, 40-59, 2004.
- [11] M.Curti, P.Degano, C.Priami, C.T.Baldari. **Modelling biochemical pathways through enhanced pi-calculus**. Theoretical Computer Science 325(1):111-140.
- [12] L.Cardelli. **Brane calculi – Interactions of biological membranes**. Proc. Computational Methods in Systems Biology 2004. Springer. To appear.
- [13] L.Cardelli, A.D.Gordon. **Mobile ambients**. Theoretical Computer Science, Special Issue on Coordination, D. Le Métayer Editor. Vol 240/1, June 2000. pp 177-213.
- [14] V.Danos, M.Chiaverini. **A core modeling language for the working molecular biologist**. 2002.
- [15] V.Danos, C.Laneve. **Formal molecular biology**. Theoretical Computer Science, to Appear.
- [16] E.H.Davidson, D.R.McClay, L.Hood. **Regulatory gene networks and the properties of the developmental process**. PNAS 100(4):1475–1480, 2003.
- [17] A.Di Pierro, H.Wiklicky. **Probabilistic abstract interpretation and statistical testing**. Proc. Second Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification. LNCS 2399:211-212. Springer, 2002.
- [18] S.Efroni, D.Harel and I.R.Cohen. **Reactive animation: realistic modeling of complex dynamic systems**. IEEE Computer, to appear, 2005.
- [19] M.B.Elowitz, S.Leibler. **A synthetic oscillatory network of transcriptional regulators**. Nature 403:335-338, 2000.
- [20] F.Fages, S.Soliman, N.Chabrier-Rivier. **Modelling and querying interaction networks in the biochemical abstract machine BIOCHAM**. J. Biological Physics and Chemistry 4(2):64-73, 2004.
- [21] D.T.Gillespie. **Exact stochastic simulation of coupled chemical reactions**, Journal of Physical Chemistry 81:2340–2361. 1977.
- [22] D.Harel. **Statecharts: a visual formalism for complex systems**. Science of Computer Programming 8:231-274. North-Holland 1987.
- [23] L.H.Hartwell, J.J.Hopfield, S.Leibler, A.W.Murray. **From molecular to modular cell biology**. Nature. 1999 Dec 2;402(6761 Suppl):C47-52.
- [24] J. Hillston. **A compositional approach to performance modelling**. Cambridge University Press, 1996.
- [25] C-Y.F.Huang, J.E.Ferrell Jr. **Ultrasensitivity in the mitogen-activated protein kinase cascade**. PNAS 93:10078–10083, 1996.
- [26] S.Kauffman, C.Peterson, B.Samuelsson, C.Troein. **Random Boolean network models and the yeast transcriptional network**. PNAS 100(25):14796-14799, 2003.
- [27] H.Kitano. **The standard graphical notation for biological networks**. The Sixth Workshop on Software Platforms for Systems Biology, 2002.
- [28] H.Kitano. **A graphical notation for biochemical networks**. BIOSILICO 1:169-176, 2003.
- [29] K.W.Kohn. **Molecular interaction map of the mammalian cell cycle control and DNA repair systems**. Molecular Biology of the Cell 10(8):2703-34, 1999.
- [30] C.Kuttler, J.Niehren, R.Blossey. **Gene regulation in the pi calculus: simulating cooperativity at the Lambda switch**. BioConcur 2004, ENTCS.
- [31] M.Kwiatkowska, G.Norman, D.Parker. **Probabilistic symbolic model checking with PRISM: a hybrid approach**. J. Software Tools for Technology Transfer (STTT), 6(2):128-142. Springer-Verlag, 2004.
- [32] H.Lodish, A.Berk, S.L.Zipursky, P.Matsudaira, D.Baltimore, J.Darnell. **Molecular cell biology**. Fourth Edition, Freeman, 2002.
- [33] J.S.Mattick. **The hidden genetic program of complex organisms**. Scientific American p.31-37, October 2004.
- [34] H.H.McAdams, A.Arkin. **It's a noisy business! Genetic regulation at the nanomolar scale**. Trends Genet. 1999 Feb;15(2):65-9.

- [35] R.Milner. **Communicating and mobile systems: the  $\pi$ -calculus**. Cambridge University Press, 1999.
- [36] R.Milo, S.Shen-Orr, S.Itzkovitz, N.Kashtan, D.Chklovskii, U.Alon. **Network motifs: simple building blocks of complex networks**. Science 298:824-827, 2002.
- [37] R.Milner. **Bigraphical reactive systems**. CONCUR 2001, Proc. 12th International Conference in Concurrency Theory, LNCS 2154:16-35, 2001.
- [38] M.Nagasaki, S.Onami, S.Miyano, H.Kitano: **Bio-calculus: its concept and molecular interaction**. Genome Informatics 10:133-143, 1999. PMID: 11072350.
- [39] F.Nielson, R.R.Hansen, H.R.Nielson. **Abstract interpretation of mobile ambients**. Science of Computer Programming, 47(2-3):145-175, 2003.
- [40] F.Nielson, H.R.Nielson, C.Priami, D.Rosa. **Static analysis for systems biology**. Proc. ACM Winter International Symposium on Information and Communication Technologies. Cancun 2004.
- [41] F.Nielson, H.R.Nielson, C.Priami, D.Rosa. **Control flow analysis for BioAmbients**. Proc. BioCONCUR 2003, to appear.
- [42] G.Paun. **Membrane computing**. Springer, 2002.
- [43] C.Priami. **The stochastic pi-calculus**. The Computer Journal 38: 578-589, 1995.
- [44] C.Priami, A.Regev, E.Shapiro, W.Silverman. **Application of a stochastic name-passing calculus to representation and simulation of molecular processes**. Information Processing Letters, 80:25-31, 2001.
- [45] P.Prusinkiewicz, M.Hammel, E.Mjolsness. **Animation of plant development**. Proceeding of SIGGRAPH 93. ACM Press, 351:360, 1993.
- [46] M.Ptashne. **Genetic switch: phage Lambda revisited**. Cold Spring Harbor Laboratory Press. 3rd edition, 2004.
- [47] A.Regev. **Computational systems biology: a calculus for biomolecular knowledge**. Ph.D. Thesis, Tel Aviv University, 2002.
- [48] A.Regev, E.M.Panina, W.Silverman, L.Cardelli, E.Shapiro. **BioAmbients: an abstraction for biological compartments**. Theoretical Computer Science, to Appear.
- [49] A.Regev, E.Shapiro: **Cells as computation**. Nature, 419:343, 2002.
- [50] I.Shmulevich, E.R.Dougherty, W.Zhang. **From Boolean to probabilistic Boolean networks as models of genetic regulatory networks**. Proceedings of the IEEE 90(11):1778-1792, 2002.
- [51] **Systems biology markup language**. <http://www.sbml.org>.
- [52] D.Thieffry, R.Thomas. **Qualitative analysis of gene networks**. Pacific Symposium on Biocomputing 1998:77-88. PMID: 9697173.
- [53] J.M.Vilar, H.Y.Kueh, N.Barkai, S.Leibler. **Mechanisms of noise-resistance in genetic oscillators**. PNAS, 99(9):5988-5992, 2002.
- [54] C-H.Yuh, H.Bolouri, E.H.Davidson. **Genomic cis-regulatory logic: experimental and computational analysis of a sea urchin gene**. Science 279:1896-1902, 1998. [www.sciencemag.org](http://www.sciencemag.org).