

Towards Programmable *In Vivo* Computation

D.K. Arvind and Marc Blenkiron
Institute for Computing Systems Architecture
School of Informatics, The University of Edinburgh
Mayfield Road, Edinburgh, EH9 3JZ, Scotland.
Email: dka|marcb@dcs.ed.ac.uk

June 1st 2003

Abstract

This paper is concerned with *in vivo* computation, i.e., the idea of using living cells, such as bacteria, as general computational devices, albeit slow ones. We speculate on ways in which functional units such as timers, counters, random number generators and programmable memory could be realised, and how one could engineer a class of cells that could then be programmed by external stimuli to trigger a desired computation. A notation has been devised for describing sequences of DNA in terms of variables representing units such as promoters, operators and messenger proteins and also provides a means to specify how a polymerase interacts with these sequences. A bespoke simulation environment, BiSim, will be used to evaluate and search the utility of emergent genetic networks, before testing them in wetware.

1 Introduction

Biological organisms continuously sense their environment and react to both internal and external stimuli, which can be exploited in the design of biological sensors. Weiss et al, for instance, have adopted the approach of engineering precise and reliable *in vivo* logic circuitry [WBKKMN02], which is embedded in the cells to process the information. In this paper we propose an alternative approach, which rather than imposing a strict digital regime, exploits the analogue and probabilistic behaviour of the cellular mechanisms to devise programmable *in vivo* computation fabrics.

The DNA of a living organism contains the totality of information required for its life-cycle, which is encoded in sequential blocks, called the *genes*. Whereas the DNA remains locked inside the nuclei within the chromosomes, some of its information can be imparted to a *messenger RNA (mRNA)* copy, from which proteins can be made. The DNA sequence is first *transcribed*, i.e., read, and then

translated, i.e., converted into proteins. The genetic code may *express* or *repress*, i.e., turn on or off, respectively, particular proteins when required. The expression is controlled by a promoter which avoids the synthesis of unnecessary products. Control regions in the promoter, called *operators*, in the form of binding sites for regulator proteins, can up- or down-regulate transcription. The proteins produced from genes decay at different rates.

Enzymes called *polymerases* are responsible for the reading of DNA from a *promoter* to a *terminator* site. These enzymes can be modified when passing over a *utilisation* site, given the presence of a required protein. A modification might allow a *polymerase* to ignore a particular *termination* site and read beyond it.

We introduce the bacterial virus, phage λ , that is used to illustrate our examples. An attraction of studying such viruses is that their DNA sequence is typically shorter than that of other forms of life. This well-studied virus is able to infect a host bacterium, *E. Coli*, and can decide whether to combine itself with its host's DNA or make many copies of itself and break open its host, through a competition of *expression* between two genes, *cI* and *cro*.

When designing new genetic networks, as opposed to modifying existing ones, it can be difficult to find the correct set of proteins and sites (out of thousands that occur naturally) to produce the desired behaviour. A number of different options have to be tested in the target environment. We have to bear in mind that cells have evolved to suit their survival and reproduction, not to run our computational instructions. Cells have defence mechanisms that can spot certain sequences of DNA that should not be there, and destroy them. The cells could mutate to give them a selective advantage such as reducing the burden on its resources, if we do not guard against this.

In the rest of this paper, Section 2 presents a notation for describing genetic networks; we propose synthetic genetic networks that could form the basis for functional units, and speculate on ways to use them as analogue components and programming them. We briefly describe the B²Sim simulation environment for modelling and simulating the behaviour of synthetic genetic networks for *in vivo* computation. Section 4 makes reference to wetware issues of concern, with concluding remarks in Section 5.

2 Notation

We present a notation for defining relationships between DNA, the action of polymerases, proteins, and transcripts. A statement in this notation can contain variables which can be assigned to sets of proteins or sites, from a database of experimental properties and ranges. The notation has arisen, in part, to aid in communicating with biologists, and has since been formalised to enable the modelling, analysis and prediction of biological processes. It is a particularly difficult task, in general, to choose suitable proteins and sites with the desired properties and interactions. We have available quite detailed information regarding some protein shapes, their reaction sites, and host environmental characteristics, and a rich database of experimental results to draw upon. Yet it is very unlikely that the instantiated components will interact in the desired manner, first time. Weiss et al [WBKKMN02] are developing a genetic tool box of optimised components which is a first step in this direction.

2.1 Format

2.1.1 Γ , a tape

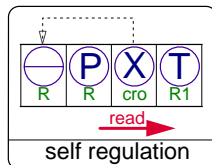
$$\Gamma = [T_{id} | P_{id} | \oplus_{id} | \ominus_{id} | U_{id} | X_{id}]^*$$

Symbol	Meaning
T_{id}	terminator
P_{id}	promoter
\oplus_{id}	up-regulator (operator)
\ominus_{id}	down-regulator (operator)
U_{id}	utilisation site
X_{id}	gene (for protein transcription)

A tape, of type, Γ , is an ordered list of components made up of terminators, promoters, operators, utilisation sites and gene encodings. Each component is identified by a name, *id*. Any two components of the same type and with

the same name are considered to have the same molecular composition. The name of a promoter will begin with either an *L* or *R* to indicate the direction of transcription. We draw on a small part of the genetic code of λ to illustrate a short tape in this notation.

$$\Gamma_{sample} = \ominus_R P_R X_{cro} T_{R1}$$



Promoter, P_R , and terminator, T_{R1} , mark the start and the end of transcription, respectively. During transcription, a gene, X_{cro} , produces an *mRNA* template from which the protein *cro* is made. This attaches itself to the operator site, \ominus_R , which down-regulates the promoter, P_R . This process hinders further transcription. The products of *mRNA* and *cro* are subject to decay and thus the negative feedback loop regulates the level of *cro*.

2.1.2 Φ , a polymerase

$$\Phi = [(P_{id}, Q)]^*, [(\ominus_{id}, M_{id}, P_{id}, Q)]^*, [(\oplus_{id}, M_{id}, P_{id}, Q)]^*, [(T_{id}, Q)]^*, [(U_{id}, T_{id}, Q)]^*, \kappa_\Phi$$

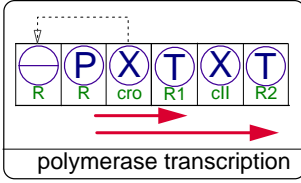
Symbol	Meaning
M_{id}	A type of protein
M_{null}	The absence of a protein
Q	$x \in Q \iff 0 \leq x \leq 1$
κ_Φ	The cost of making this enzyme

Group	Meaning
(P_{id}, Q)	Affinities for promoters
$(\oplus_{id}, M_{id}, P_{id}, Q)$	Up regulation effect
$(\ominus_{id}, M_{id}, P_{id}, Q)$	Down regulation effect
(T_{id}, Q)	Terminator effectiveness
(U_{id}, T_{id}, Q)	Utilisation site's effect

A polymerase of type, Φ , describes the behaviour of a transcribing enzyme, such as the chances of it starting transcription from a particular place, and how operators and terminators may interfere. Given a protein is attached to a utilisation site and a polymerase passes over it during transcription, we can encode a change in the polymerase, such as the reduction in the effectiveness of a termination site. The following example illustrates part of the behaviour of the E. Coli polymerase, as it interacts with a simplified short

tape corresponding to a longer sequence of phage λ .

$$\Phi_{sample} = (\mathbf{P}_R, 0.95), (\ominus_R, M_{cI}, \mathbf{P}_R, 1.0), (\ominus_R, M_{cro}, \mathbf{P}_R, 0.7), (\mathbf{T}_{R1}, 0.5), (\mathbf{T}_{R2}, 1.0)$$



This polymerase has a strong affinity for \mathbf{P}_R and therefore has a high chance of starting transcription from here. But when a M_{cro} is in \ominus_R , this promoter is partially blocked, which lowers its chance. The terminator, \mathbf{T}_{R1} , is only effective half of the time, whereas the terminator, \mathbf{T}_{R2} , is effective all of the time. In other words, half of the time, \mathbf{X}_{cro} will produce a transcript, and in the other half, a transcript from both \mathbf{X}_{cro} and \mathbf{X}_{cII} will be produced.

2.1.3 M, a protein type

$$M = [(\mathbf{P}_{id}, Q)]^*, [(\oplus_{id}, Q)]^*, [(\ominus_{id}, Q)]^*, [(\mathbf{U}_{id}, Q)]^*, \delta_{att}, \delta_{un}, \kappa_M$$

Symbol	Meaning
δ_{att}	The rate of decay when <i>attached</i>
δ_{un}	The rate of decay when <i>unattached</i>
κ_M	The cost of making this protein

Group	Meaning
(\mathbf{P}_{id}, Q)	Affinity for promoter activation site
(\oplus_{id}, Q)	Affinity for up-regulating operator
(\ominus_{id}, Q)	Affinity for down-regulating operator
(\mathbf{U}_{id}, Q)	Affinity for utilisation site

This statement describes a protein of type, M , with its affinities for promoters, operators and utilisation sites, decay rates and cost of its production. Catalysts and other effects on affinities are not directly stated.

2.1.4 X, a transcript encoding type

$$X = [(M_{id}, \tau)]^*, \delta$$

Symbol	Meaning
τ	A length of time
δ	Decay rate of its <i>mRNA</i>

Group	Meaning
(M_{id}, τ)	A protein type and its length

This statement lists a transcription encoding of type, X , which lists a number of protein products and their duration in time units, which gives an indication of how long the processes of transcription and translation take. A decay rate provides an indication of the stability of the *mRNA* template that is produced by transcription. For example:

$$X_{sample} = (M_{cro}, 20), (M_{cII}, 10), 0.05$$

So the transcript for M_{cII} would take 10 time units and the whole process would take 30.

2.1.5 Relationships

A set of relationships which describes the low-level interactions in a system may be specified by:

$$[\mathbf{T}_{id}]^*, [\Phi_{id}]^*, [M_{id}]^*, [X_{id}]^*$$

This statement describes a collection of tape configurations, and types of polymerase, proteins, and transcription. Note that a particular instance of a polymerase may be modified when it passes over a utilisation site, and that an instance of a tape configuration may be modified, for example, by mutation.

2.2 Cellular Functional Units

We next speculate on how functional units, such as counters, memory and random number generators, could be implemented in cellular fabric in order to realise programmable *in vivo* computation. Other functional units, such as ring oscillators, have already been demonstrated [EILe00]. It is very likely that once we begin to test even the most simple of these building blocks in wetware, then we would need to significantly tune and adjust the design and parameters in order to achieve their desired operation. A number of simplifications have been made to aid the clarity of presentation: in particular, it is often desirable that a protein is able to negatively regulate its own production, which would lead to a more stable level of concentration and not exhaust the host cell's resources.

2.2.1 Random Number Generator

A random number is considered to be a sequence of booleans, each assigned either true or false. The

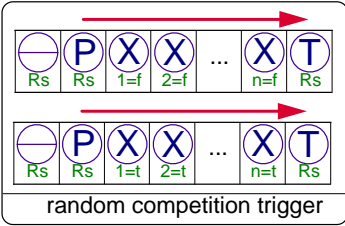
method postulated here would undoubtedly be subject to environmental biases. Let us consider a number with m booleans of form:

$$\sum_{i=0}^{m-1} 2^i \nu_i$$

Each ν_i will have a specific protein encoded to true, $X_{i=T}$, and a specific protein encoded to false $X_{i=F}$. At a trigger point, the following sequences will be transcribed for a short burst of time:

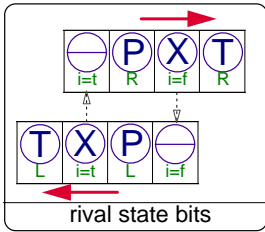
$$\ominus_{R_s} P_{R_s} X_{0=T} X_{1=T} \dots X_{m-1=T} T_{R_s}$$

$$\ominus_{R_s} P_{R_s} X_{0=F} X_{1=F} \dots X_{m-1=F} T_{R_s}$$



which would cause approximately similar quantities of true and false values for each variable to be present. A protein will be able to inhibit the production of its counterpart value, leading to a competition, such that after a period of time, the number will stabilise. The plausibility of such a two-state component has already been demonstrated in wetware [GCC00].

$$\ominus_{i=T} P_{R_s} X_{i=F} T_{R_s} T_{L_s} X_{i=T} P_{L_s} \ominus_{i=F}$$



2.2.2 Counters

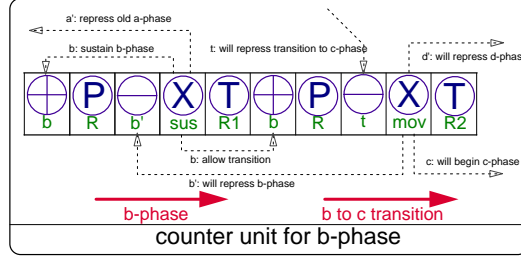
We may wish to count the number of times an event occurs, such as cell division, or perform a function every n generations. Our event trigger is a form of signal that is generated at intervals as a protein that also degrades rapidly.

In our case, a counter goes through a series of possible phases, where a diffuse signal fires an increment to the next phase. Although subject to behavioural fluctuations, we would wish to ensure

that the counter does not increment by more than a phase for a given trigger, and that only one phase is ever dominant for a length of time. Consider:

$$\pi(a, b, c, d, \Omega) = \oplus_b P_{R_s} \ominus_{b'} X_b X_{a'} T_{R1} \Omega_t \oplus_b P_{R_s} X_{d'} X_{b'} X_c T_{R2}$$

where a , b , c , and d are the four phases, and b is the current phase. Ω is one of the two types of operators.



When in phase b , production of b is maintained. The previous phase, a is suppressed by production of a' . When an event signal, corresponding to protein M_t is either bound to the Ω site or is absent through decay, then three different proteins are produced: b' suppresses the production of b , c starts the move to the next phase, and d' ensures that the phase does not change twice on the same trigger by inhibiting the phase following c .

A four phase counter could be represented by:

$$\pi(a, b, c, d, +) \pi(b, c, d, a, -)$$

$$\pi(c, d, a, b, +) \pi(d, a, b, c, -)$$

2.2.3 Timers

Given that a class of cells already has a counter and an oscillator to drive it, it would then be possible to approximate the synchronisation of a number of cells' counters. This can be achieved by passing particles between cells, as has been demonstrated in [McKoHaCo02]. On cell division, the phase of the clock would be passed on to the two resulting cells through the protein concentrations from the original cell.

2.2.4 Memory

We next propose the design of a simple serial write, serial read cyclic tape storage.

- The tape consists of a cycle n squares, where each square, $\sigma_{0 \leq i < n}$, corresponds to a point in an n phase counter. Each square can hold m locations.
- There are m distinct *write* location signals, each being encoded in a protein $M_{Wj:(j < m)}$. Each signal corresponds to a location offset. There are m distinct *read* location signals, each encoded in a protein $M_{Rj:(j < m)}$. Each signal corresponds to a location offset.
- At counter time, $i < n$, the protein, ρ_i , will be present inside the cell as produced by the timer. The presence of a write signal, M_{Wj} , will cause the square σ_i to have its j^{th} location to be set. Also at counter time, $i < n$, if σ_i has its j^{th} location set, then the read signal M_{Rj} will be produced.

2.2.5 Digital or Analogue?

Signals in cellular networks take the form of protein concentrations. We do not necessarily need to utilise these as digital-type components. For example, a number of processes may contribute to the production of proteins, which depending probabilistically on their *relative* levels will cause an action, or a more graded analogue response, such as movement. In the phage λ virus that infects E. Coli bacteria, for example, a decision is made whether or not to lay dormant in the host's DNA and multiply with it, or to take over the cell and multiply rapidly, thereby killing the host through a competition over promoter and operator sites by the relative levels of two signals, *cI* and *cro*. In addition, there also exist sets of proteins and sites that have differing affinities for each other.

2.3 A Case Study in Programmability

We present an example which illustrates *in vivo* computation to solve a boolean variable satisfaction problem. We describe how a sequence of DNA could be added, perhaps by virus, to a cell and cause the *utilisation* of a previously engineered functional units.

Given a CNF statement, ψ , with n clauses:

$$\psi = \bigwedge_{i=0}^{n-1} \omega_i$$

where, for ease of illustration, each clause contains two out of m possible boolean variables in one of the following combinations:

$$\omega_i = (\nu_j \vee \nu_k) \quad \text{and} \quad (\bar{\nu}_j \vee \nu_k)$$

$$\text{and} \quad (\nu_j \vee \bar{\nu}_k) \quad \text{and} \quad (\bar{\nu}_j \vee \bar{\nu}_k) \quad : \quad i < n; j, k < m$$

We wish to discover a sequence of boolean assignments to each of the m variables, such that ψ is true. We may negate both sides of the equation to obtain:

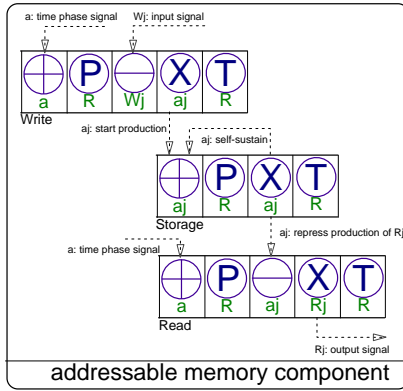
$$\bar{\psi} = \bigvee_{i=0}^{n-1} \bar{\omega}_i$$

Therefore, if any one of the negated clauses is found to be true, then the whole expression will be false. Clauses are converted to the following:

$$\bar{\omega}_i = (\bar{\nu}_j \wedge \bar{\nu}_k) \quad \text{or} \quad (\nu_j \wedge \bar{\nu}_k)$$

$$\text{or} \quad (\bar{\nu}_j \wedge \nu_k) \quad \text{or} \quad (\nu_j \wedge \nu_k) \quad : \quad i < n; j, k < m$$

Here is a partial sketch of the implementation. We acknowledge that there may be a degree of over-spill of reads and writes tied to a timer:



- Write, $\oplus_{\alpha} P_R \ominus W_j X_{\alpha_j} T_R$, when time signal α is active and there is no presence of the write signal, M_{Wj} , then some M_{α_j} will be produced.
- Storage, $\oplus_{\alpha_j} P_R X_{\alpha_j} T_R$, the signal protein, M_{α_j} , is self-sustaining. Regulation of the level of this protein is glossed over.
- Read, $\oplus_{\alpha} P_R \ominus_{\alpha_j} X_{Rj} T_R$, at time, α , the read protein, M_{Rj} , is produced only if there is no α_j bound to the down operator.

One could imagine a partially associative form of such a memory where a particular write signal activates a *number* of memory locations with *differing probabilities*. In this way we may also be able to create multiple location instructions.

2.3.1 Translation

Each clause may be mapped to the notation as follows:

$$\begin{array}{ll}
 \bar{\nu}_j \wedge \bar{\nu}_k & \ominus_k \mathbf{P}_R \ominus_j \mathbf{U}_z \mathbf{T}_z \mathbf{X}_p \mathbf{T}_e \\
 \nu_j \wedge \bar{\nu}_k & \ominus_k \mathbf{P}_R \oplus_j \mathbf{U}_z \mathbf{T}_z \mathbf{X}_p \mathbf{T}_e \\
 \bar{\nu}_j \wedge \nu_k & \ominus_j \mathbf{P}_R \oplus_k \mathbf{U}_z \mathbf{T}_z \mathbf{X}_p \mathbf{T}_e \\
 \nu_j \wedge \nu_k & \oplus_j \mathbf{P}_R \oplus_k \mathbf{U}_z \mathbf{T}_z \mathbf{X}_p \mathbf{T}_e
 \end{array}$$

where, \mathbf{T}_z encodes a terminator that is skipped if utilisation site, \mathbf{U}_z , has a protein attached. Terminator, \mathbf{T}_e , is a termination site even for a modified polymerase. Each $\ominus_{j,k}$ encodes an operator that blocks its adjacent promoter if a corresponding j, k protein is attached. Each $\oplus_{j,k}$ encodes an up-regulator that requires activation by a corresponding j, k protein. We note that there are relatively few up-regulators that work well in conjunction, however, this can be resolved by causing the presence of a protein, j to block the production of an intermediate protein, j' thus allowing the use of j' and k in conjunction. \mathbf{X}_p could encode a protein that kills the cell or, less dramatically, resets the counter. A correct guess, noted by not having generated protein \mathbf{X}_p after another counter run, could cause the production of a fast-decaying protein that renders it immune to a virus that could be used to infect a lawn of bacteria to select the candidate answers.

2.3.2 Execution

Using a random number generator as described in Section 2.2.1 of size m locations for each variable, a guess is made for a correct set of assignments. While this happens, a counter will tick for a period of time which is sufficient for the random number generator to stabilise. Once the counter reaches its end value, then proteins will be transcribed that bind to the utilisation sites \mathbf{U}_z . Any incorrect assignments could result in dead, or reset bacteria. The remaining bacteria will either have guessed the correct assignment, or some other factor, such as mutation, may have interfered.

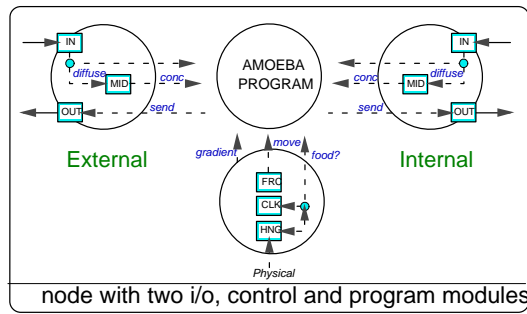
3 Simulation Environment

3.1 Platform

$\mathcal{B}^2\text{Sim}$ is a simulation platform which models the biological cell as a network of functional units at different levels of abstraction. Provisions have been made to efficiently support complex, time-changing models and the $\mathcal{B}^2\text{Sim}$ engine's (soft virtual machine, event-queue based) design is tailored to run

on a network of workstations such as a Beowulf cluster, as is its companion directed-search tool that is used in optimisation.

The support for modularity and flexibility allows devices to have multiple, time- and event-varying channels for inter-cellular and intra-cellular communication, with the ability to express characteristics such as external effects on signal strength or chemical diffusion characteristics and supports their use in a script file. For example, we may choose to model particle reactions using concentrations and rates, as stochastic kinetic reactions, or a combination through specification in an interpreted script file. Notably, script files allow *ranges* of settings, to provide for batch runs and directed search over numerical and structural parameters.



Nodes in such a network can range from collections of cells to cellular pathway junctions, which allows a simulation to take place in a mixture of different levels of abstraction for convenience and efficiency. A component of a node specifies its input/output, storage and environmental interactions. A number of exported instructions may be declared which allows a node's complete behaviour to be expressed in these terms.

4 Wetware

4.1 "Hardware"

We propose that sequences representing functional units will be encoded inside a viral plasmid, such as T4, which will be inserted into our bacterial cell, such as the well-studied E. Coli. We propose that these ready-made computational devices could be stored and later used *off the shelf* for a desired computation, for example, by storing them in liquid nitrogen, an established practice. A slow and incremental approach will be necessary in creating functional units of even slight complexity.

4.2 Corruption

One of the many problems facing a hardware or software encoding in DNA will be corruption, whether from UV light or another source of mutation. If small proportions of cells were, in this respect, to malfunction, this is relatively straight forward to tackle, at least in principle, using means such as redundancy in groups. If however, a bacterium happened to mutate such that it reduced the length of its DNA, causing it a selective advantage, but also corrupting a functional unit, and then multiplying, this would cause a different type of problem. We plan to investigate whether analogues of error-checking as used frequently in computer communications could be applied to DNA messages - perhaps causing cell death if discovered. Graph colouring algorithms have already been illustrated as a means to add robustness [AmGiHo96] to a DNA based computer [Adelman96] and bacteria and viruses themselves can produce enzymes that recognise certain unwanted, short sequences of DNA and destroy them.

4.3 “Software”

Our colony of loosely synchronised bacterial cells may emit some perceivable signal each time their clock cycles wrap, such as a visible dye. We can send in modified sugars that can cause activation of genes inside cells to effectively generate external signals. We can encode a form of corruption checking by causing the removal of certain code to cause the production of an antibiotic such as ampicillin that will kill the bacterium. Similarly, we can send in our software encoded in a stretch of tamed λ or another virus that will not adversely interact with the previous method of inserting hardware functional units. We are investigating the possibilities of removal or deactivation of one software program for another, such as by selectively destroying the existing program's DNA and preventing the host bacterium from intervening (disabling the SOS response) for a period of time to provide a form of reprogrammability.

5 Conclusions

We have presented an alternative approach to programmable *in vivo* computation which exploits the analogue and probabilistic nature of cellular processes. A notation has been devised for describing genetic networks, with designs presented for functional units such as counters, memory and random

number generators. The BⁱSim simulation environment has been implemented and tested. Future work will investigate the implementation of the functional unit designs in wetware and refining the simulation models to be able to accurately predict their behaviour in the BⁱSim environment.

Acknowledgements

We would like to thank Jamie Davies, Department of Biomedical Sciences, University of Edinburgh for discussions on the wetware aspects of *in vivo* computation.

References

- [Adelman96] Adleman, L. 1996. *On constructing a molecular computer*, DNA Based Computers, Eds. R. Lipton and E. Baum. DIMACS: series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society. 1-21(1996).
- [AmGiHo96] Amos, M., Gibbons, A., and Hodgson, D. 1996. *Error-resistant implementation of DNA Computations*, Proceedings of the Second Annual Meeting on DNA Based Computers, Princeton, USA.
- [McKoHaCo02] McMillen, D., Kopel N., Hasty, J., and Collins, J. 2002. *Synchronizing genetic relaxation oscillators by intercell signaling*, Proceedings of the National Academy of Science 99, 670-684.
- [EILe00] Elowitz, M. B. and Leibler, S. 2000. *A synthetic oscillatory network of transcriptional regulators*, Nature 403, 335-338.
- [WBKKMN02] Weiss, R., Basu, S., Kalmbach, A., Karig, D., Mehreja, R., and Netatvali, I. 2002. *Genetic Circuit Building Blocks for Cellular Computation, Communications and Signal Processing*, International Journal of Natural Computing, Kluwer, In Press.
- [GCC00] Gardener, T., Cantor, C., and Collins, J. 2000. *Construction of a genetic toggle switch in Escherichia coli*, Nature 403, 339-342.