Turing Complete Catalytic Particle Computers

Anthony M.L. Liekens[1] and Chrisantha T. Fernando[2]

[1] Department of Biomedical Engineering,
Technische Universiteit Eindhoven, Eindhoven, the Netherlands
anthony@liekens.net
[2] School of Computer Science,
University of Birmingham, Edgbaston, United Kingdom
c.t.fernando@cs.bham.ac.uk

Abstract. The *Bare Bones* language is a programming language with a minimal set of operations that exhibits universal computation. We present a conceptual framework, Chemical Bare Bones, to construct Bare Bones programs by programming the state transitions of a multifunctional catalytic particle. Molecular counts represent program variables, and are altered by the action of the catalytic particle. Chemical Bare Bones programs have unique properties with respect to correctness and time complexity. The Chemical Bare Bones implementation is naturally suited to parallel computation. Chemical Bare Bones programs are constructed and stochastically modeled to undertake computations such as multiplication.

1 Introduction

1.1 Chemical Computing

An approach for programming a chemical computer is to design a complex 'particle' capable of a controlled transition between configurations, where each configuration is capable of catalyzing a specific set of reactions. Ribozymes can be artificially selected that catalyze specific reactions [1]. Multi-enzyme complexes are common in cells, e.g. PDGF and Tar Complexes [2]. Just as we require, they possess multiple catalytic activities and exist in many states. Programmability arises because the state of a complex subunit is dependent on the states of other subunits on the complex. The topology of the complex can be designed, e.g. clusters, chains, rings, to allow appropriate 'conformational spread' [3]. Approximately digital solid-state circuitry can be produced in proteins [4].

Our approach differs from other work in chemical computing with reaction networks in the following ways. Chemical Bare Bones (CBB) does not make explicit the implementation details of the catalytic reactions; the substrates may be proteins, RNAs or metabolites. Although DNA hybridization catalyst circuits have been proposed by Seelig et al., they model at the algorithm level, circuits of logic gates, not serially executable programs [5]. CBB describes computations carried out by only *one* particle complex with multiple states, and not networks of catalytic particles computing in a distributed manner. Such neural network

F. Almeida e Costa et al. (Eds.): ECAL 2007, LNAI 4648, pp. 1202–1211, 2007.

metaphors utilize coupled cascade cycles, where the weights are the extent of allosteric and covalent modification of the equilibrium position between binary protein configurations that represent activities [6,7,8]. Although it is possible to produce logic gates with an enzyme cascade cycle it will be a formidable task to assemble many of these gates together into a network [9]. The demonstration that CBB is Turing universal lies in the isomorphism between chemical reactions and the Bare Bones language. This overlays the underlying Turing universality of chemical kinetics on which our system depends [10]. Other approaches to demonstrating the Turing universality of a chemical computing system depend for example on forming an isomorphism with Wang tiles [11]. CBB does not produce analog reaction networks, e.g. integral feedback controllers [12] or analog networks capable of computing mathematical functions at their steady state [13]. Such analog networks cannot easily be hand-designed whereas CBB allows hand-design of similar functionalities, plus the incorporation of analog networks where necessary.

1.2 The Bare Bones Programming Language

The Bare Bones programming language contains only 2 assignment statements and one control structure, besides chaining of instructions. The assignment statements are **increase** v and **decrease** v where v denotes a variable name representing a strictly positive integer. Variables are created in memory when they are used for the first time, with a random initial value. The sole control structure is represented by a specific **while** loop, as a **while** $v \neq 0$ **do** ... **end** statement pair. The Bare Bones programming language only allows one condition to control the **while** loop, $v \neq 0$, where v can be any variable.

The Bare Bones language can express programs for any partial recursive function, as shown by Brookshear [14], which ensures Turing completeness. As an elementary example of a Bare Bones program, the above instructions can be used to clear a variable v with **while** $v \neq 0$ **do decrease** v **end**. In Algorithm 1, the values in variables v and w are multiplied and the result is stored in u. Versions that destroy the initial values of v or w during the computation are also possible and result in simpler algorithms.

2 Methods

We show a conceptual implementation of the 3 basic Bare Bones instructions as networks of chemical reactions. Molecules and their counts represent program variables and their values. The number of molecules `V` denotes the value of variable v. We assume that the reaction networks have access to resource particles `R`. A multifunctional catalytic particle controls the flow of the program, analogous to a program or instruction counter in computers. The state of the particle corresponds to the current instruction that has to be processed by the program. In order to process the instruction, the controlling particle catalyzes a reaction

Algorithm 1 Multiplication ($u \leftarrow v * w$) in Bare Bones

clear u, **clear** t_1, **clear** t_2
while $v \neq 0$ **do**
  increase t_1, **decrease** v
end
while $t_1 \neq 0$ **do**
  decrease t_1, **increase** v
  while $w \neq 0$ **do**
    increase t_2, **decrease** w
  end
  while $t_2 \neq 0$ **do**
    increase w, **increase** u, **decrease** t_2
  end
end

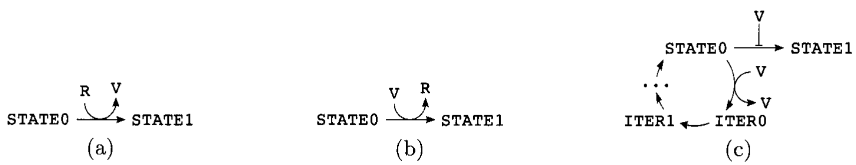

Fig. 1. Chemical Bare Bones primitives (a) **increase** v, (b) **decrease** v and (c) **while** $v \neq 0$ **do** ... **end**

that acts on molecules representing the variables of the program. During the instruction, the program counter particle changes to a conformation representing the next instruction for the program.

Simple catalytic reactions that implement the **increase** v and **decrease** v primitives as basic reaction networks are depicted in Figure 1(a-b). Both primitives can be written as a single reaction. In the case of **increase** v, the control particle in state STATE0 reacts with an abundant resource molecule R, where a new molecule V and the control particle in configuration STATE1 are the products of the reaction. When the control particle signals STATE1, the **increase** v operation is terminated. Similarly, we let the program counter react with particle V to release a resource particle R to instantiate the **decrease** v primitive.

These two basic primitives can be chained as an ordered series of instructions. As an example, Figure 2 represents a program that consecutively goes through instructions **increase** u, **decrease** v and **increase** w. To construct this program, the controller goes through three successive states where it catalyzes one reaction.

The **while** is implemented by means of two reactions as shown in Figure 1(c). One reaction sets the program counter molecule from state STATE0 to the first instruction of the iteration (ITER0) if molecules V are present. An iteration of the while loop is a sequence of Bare Bones primitives. At the end of the iteration, the program counter is returned to its STATE0 state. The second reaction that makes up a **while** control structure moves the program counter out of the loop, and is inhibited by V molecules. This requirement for inhibition limits the application of the model, as some forms of reaction systems (e.g., metabolic networks) do not have this property. For now, we assume that this inhibition is *strict*, i.e., one V molecule locks the transition of the program counter from STATE0 to STATE1. In a later section, we analyze the behaviour of the loop with stochastic, competitive inhibition. Under the assumption of strict inhibition, the **while** loop cycles for as long as there are molecules V in the system. Note that if the value of v is not decreased during the iteration, the control structure loops unboundedly.

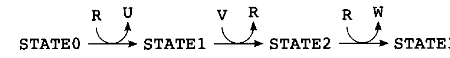

Fig. 2. Chaining three instructions

The above construction allows Bare Bones primitives to be ported to a platform of conceptual catalytic and inhibitory reactions. As a consequence, any Bare Bones program can be implemented as a reaction network. Since the Bare Bones language is Turing complete, our interpretation of the language as chemical reaction networks results in a universal language as well.

Because of this universality, more complex instructions that are being added to the language do not improve upon its expressive power. Higher level primitives can all be implemented as Bare Bones instructions, but may have simpler interpretations that can be added to the primitives, and increase the readability of the programs. As an example, an **if** $v \neq 0$ **then** ... **else** ... **endif** control structure could be implemented as two consecutive **while** loops, or it can be implemented more straightforwardly as a pair of reactions. One reaction would move the controller molecule to one series of instructions if V is present, where the second reaction, inhibited by V molecules, moves the pointer to a second block. At the end of both conditional blocks, the controller points to the instruction that follows the control structure.

2.1 Basic Programs

Figure 3 shows an implementation of the **clear** v operator as a basic reaction network. For as long as there are molecules V in the reactor, these react with state STATE0 of the program controller, to produce a resource particle R. If no more molecules V are left in the reactor, the inhibitory reaction becomes unlocked, thereby ending the execution of the program by moving the program controller to STATE1. Figure 4 shows an implementation of a multiplier, as in Algorithm 1. The program initially clears the result and temporary variables t_1 and t_2. Then, molecules V are moved to temporary molecules T1. Consuming T1, the **while** loop starting at state STATE4 sums the value of v to u, t_1 times, using a similar loop with temporary molecule T2. When all temporary molecules T1 have been used, the program signals its end by setting the state of the program controller to STATE8.

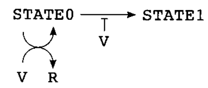

Fig. 3. Chemical Bare Bones program for **clear** v as a reaction network

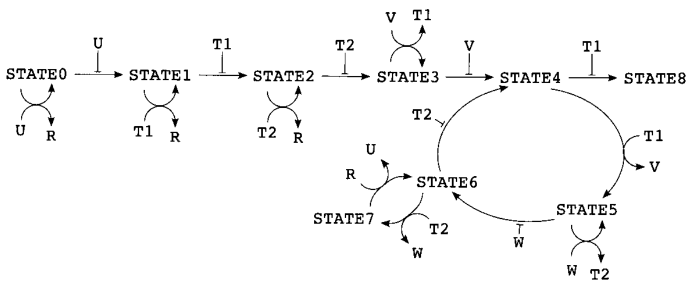

Fig. 4. Chemical Bare Bones program for multiplying the values of v and w as a reaction network. The result is stored in variable u

2.2 Networks as Subroutines

Complex programs in Chemical Bare Bones can be used as modular components, and called in similarity to subroutines. In order to do so, a main program sets the parameters of the module, and activates a new program controller set to the first state of the module. When the end state of the module appears, the extra controller has to be deactivated. The result of the subroutine's computation can now be copied to local variables before continuing the main program. In similarity to an instruction stack in computers, program controllers and their states guarantee the correct flow of the program. For now, we assume that only one program controller can be active at any given time. In a later section, we elaborate on the implementation of parallel program threads.

As an example, the multiplier reaction network from Figure 4 can be called from another program. In order to do so, the parameters for the multiplication have to be copied to variables v and w. Activating a new controller in state STATE0 of the multiplier starts the subroutine. When the multiplier reaches state STATE8, its controller must become deactivated, and the result in u copied to the local variable of the main program before continuing.

Chemical Bare Bones programs are built up out of digital computations, in contrast with other models that rely on analog circuits, see e.g., [13] for implementations of analog square root functions. If each of the reactions of such an analog circuit is catalyzed by the state of the program controller, and the program controller has a method to sense the termination of the analog program, then Chemical Bare Bones programs can, in theory, interface with such analog circuits.

3 Results

3.1 Reaction Networks as Machines

Stochastic Models and Correctness. To set up general Bare Bones programs in reaction networks, we have previously assumed that the inhibition rule to end a while loop is strict. However, this assumption is not feasible in real networks of reactions. Because of real chemistry's stochastic nature, the correctness of a chemical implementation of a program is not guaranteed.

Assuming a well-stirred mixture, and a basic model of mass-action kinetic laws, a reaction is said to occur with a propensity proportional to its reaction rate and the number of reactants available in the system. If we assign a sufficiently fast reaction rate to the reaction that starts an iteration of the while loop, and a relatively slow reaction rate to the competitively inhibitory reaction that exits the while loop, we can decrease the probability that a while loop is exited prematurely.

For now, we suppose that the program counter molecule of the program points to the start of the **while** $v \neq 0$ **do** ... **end** loop, where the program can either enter an iteration of the **while** loop, or exit the loop. We first want to determine the probability that the program exits the **while** loop at the entry state. Let k_{fast} be the reaction rate for the reaction that enters an iteration of the loop. The propensity for this reaction to occur is $k_{\text{fast}}v$. Similarly, let k_{slow} be the reaction rate to exit the **while** loop, and transform the program counter molecule to its next state, ending the **while** loop. Assuming competitive inhibition, the propensity of this reaction is k_{slow}. Using Gillespie's algorithm [15] for stochastic models of reactions, we can determine the probability that the system exits the **while** loop prematurely, dependent on the number of molecules V in the reactor. The probability that either reaction occurs first is proportional to its propensity,

$$\Pr\left[\text{next reaction enters iteration}\right] = \frac{k_{\text{fast}}v}{k_{\text{fast}}v + k_{\text{slow}}} \tag{1}$$

$$\Pr\left[\text{next reaction exits loop}\right] = \frac{k_{\text{slow}}}{k_{\text{fast}}v + k_{\text{slow}}}. \tag{2}$$

If $v = 0$, the probability that the next reaction exits the **while** loop is 1. When the number of molecules V is low, the probability to prematurely end the **while** loop is highest. The probability that the **while** loop stops iterating is lower if a faster reaction rate k_{fast} or slower rate k_{slow} is chosen.

We can, as an example, compute the probability that a **clear** v program, as in Figure 3, stops removing V molecules and moves to its end state too early. The probability that the loop is terminated prematurely, is given by

$$1 - \prod_{n=1}^{v} \frac{k_{\text{fast}}n}{k_{\text{fast}}n + k_{\text{slow}}} = 1 - \frac{v!\,\Gamma\left(\frac{k_{\text{slow}}}{k_{\text{fast}}} + 1\right)}{\Gamma\left(\frac{k_{\text{slow}}}{k_{\text{fast}}} + v + 1\right)}. \tag{3}$$

Figure 5(a) depicts the probability of terminating a **clear** v prematurely. For higher $k_{\text{fast}}/k_{\text{slow}}$ ratios, the probability of incorrectly exiting the while loop is

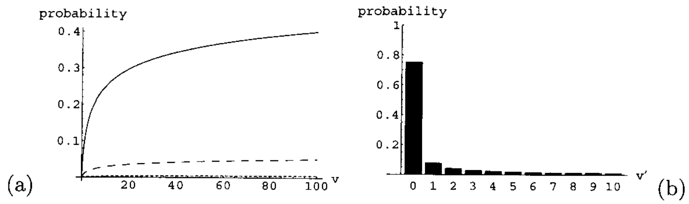

Fig. 5. (a) Probability of prematurely exiting a **clear** v program, dependent on the initial value of v. The continuous, dashed and dotted graphs represent the probabilities for $k_{\text{fast}}/k_{\text{slow}} = 10, 100$ and 1000, respectively. (b) Probability distribution over the possible end results v' of a **clear** v program with initial $v = 10$, with ill-defined parameters $k_{\text{fast}}/k_{\text{slow}} = 10$

lower. As more iterations of the **while** loop have to be carried out, the probability of exiting too early is higher. Independent on the kinetic rate settings, as v goes to infinity, the probability of prematurely exiting the loop tends to 1.

The probability to end the **clear** v program when there are still v' particles left, with $1 \leq v' \leq v$ is given by

$$\frac{k_{\text{slow}}}{k_{\text{fast}}v' + k_{\text{slow}}} \prod_{n=v'+1}^{v} \frac{k_{\text{fast}}n}{k_{\text{fast}}n + k_{\text{slow}}} = \frac{k_{\text{slow}}\Gamma\left(\frac{k_{\text{slow}}}{k_{\text{fast}}} + v'\right)\Gamma(v+1)}{\Gamma\left(\frac{k_{\text{slow}}}{k_{\text{fast}}} + v + 1\right)\Gamma(v'+1)} \quad (4)$$

Figure 5(b) shows the probability to end a **clear** v operation with v' particles left, where the initial value of v is set to 10. Parameter k_{fast} was chosen to be 10 times bigger than k_{slow}. Increasing the rate between these parameters results in higher probability to terminate the computation correctly, as shown in Figure 5(a).

Time Complexity. As a result of the previous section, a larger ratio between kinetic rates k_{fast} and k_{slow} increases the accuracy of programs that use **while** loops. There is, however, a trade-off in the expected running time of the program. Indeed, for high $k_{\text{fast}}/k_{\text{slow}}$ ratios, entering iterations of a **while** loop is fast, where exiting from a while loop is relatively very slow.

As an example, we analyze the running time of a successful **clear** v operation. Assuming that k_{fast} and k_{slow} are well-separated, the expected running time of the **clear** v program can be approximated with the sum of expected times that separate consecutive operations, with

$$\text{E}\left[\text{running time of successful } \textbf{clear } v\right] \approx \frac{1}{k_{\text{slow}}} + \sum_{n=1}^{v} \frac{1}{nk_{\text{fast}}}. \quad (5)$$

The first term denotes the expected time to exit the while loop when there's no V particles left, the second term sums up the expected times to execute an iteration of the **while** loop. Note that this is an approximation since we do not take the probability to exit the **while** loop prematurely into account. Since rate k_{slow} is chosen to be very small in comparison with k_{fast}, the expected running time of the **clear** v operator is dominated by the time it takes to exit the **while** loop, and not so much by carrying out the instructions in the **while** loop.

This result implies a unique property of Chemical Bare Bones programs with respect to time complexity. By choosing high $k_{\text{fast}}/k_{\text{slow}}$ ratios, the time complexity of Bare Bones programs in stochastic models becomes dependent on the number of **while** loops that need to be exited during a run, and not on the number of instructions that have to be processed by the program, since these are relatively fast in comparison with the time required to exit a **while** loop.

Parallelism. In the above sections, we have constructed essentially serial programs for an inherently parallel platform. By introducing multiple program counter molecules in the reactor, parallel programs can be carried out in reaction networks. However, these multiple program counters and their instructions act on shared memory variables.

Multi-threaded programs with shared memory require a mutual exclusion concept to control the flow of the program. The simplest implementation is to use binary semaphores as safeguards of critical sections, i.e., sections of the program that access shared memory. Before a thread of the parallel program executes a critical section, it reserves the semaphore which offers mutual exclusive access to the shared memory. At the end of the critical section, the semaphore is released to allow other threads to access shared memory.

The Bare Bones language in reaction networks allows for the implementation of parallel programs with semaphores. Threads of the parallel program can be started by activating multiple program counter molecules. For the implementation of a semaphore, a variable s is initialized with value 1 at the beginning of the program. If a thread wants to enter a critical section, it can reserve the semaphore with a **decrease** s statement. At the end of its critical section, the thread must release the semaphore by an **increase** s statement to allow other threads to enter their critical sections after locking the semaphore. If the semaphore was reserved by a thread, other threads that request the semaphore with a **decrease** s reaction are implicitly put on hold since the reaction can only be completed if the semaphore is available.

Other options for parallelization is to activate multiple program counters that act on local variables, in similarity to cellular automata [16], or as Chomsky grammars and P systems.

3.2 Simulation

Using the stochastic simulator BioNetS [17] we modeled the multiplier network in Figure 4. Figure 6 shows the typical behaviour of the multiplier for inputs $v = 13, w = 7$, computing the correct result $u = 91$. Figure 7 shows the approximate mean errors of the same network for input values of v and w in the range 1 to

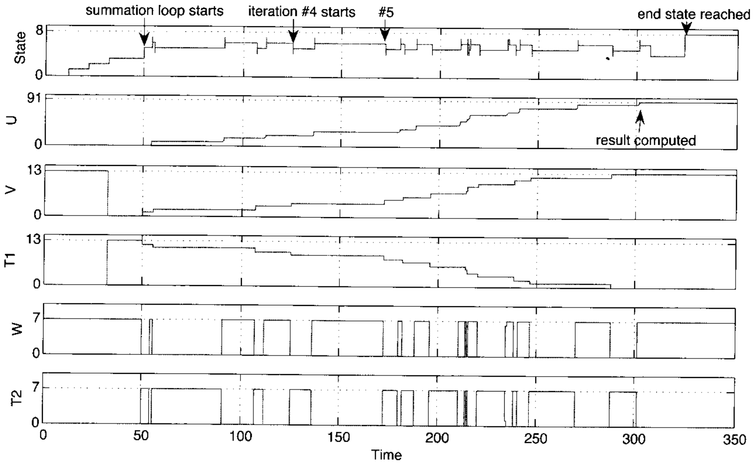

Fig. 6. Typical behaviour of the multiplier with initial values $v = 13, w = 7, u = t_1 = t_2 = 0$. The end result, $u = v, w = 91$ is achieved just before the program controller reaches state 8. The intra-loop rates are fast, with reaction rates $k_{\text{fast}} = 100$, whilst the inter-loop state transition steps are slow, with reaction rate $k_{\text{slow}} = 0.1$

100. For each (v, w) pair, 100 trials were conducted with different random seeds. The rare failure to complete the cycles of the outer loop result in large errors with high variance. As predicted, the greater the difference in rate between fast intra-loop reactions and slow between-loop reactions, the higher the accuracy of the multiplier. For higher values of v and w we see an increasing mean error.

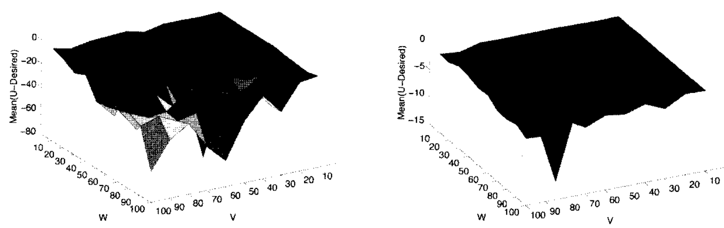

Fig. 7. Left: Low accuracy system, $(k_{fast}/k_{slow} = 100/0.1)$. **Right:** High accuracy system, $(kfast/kslow = 1000/0.1)$. The vertical axis shows the mean$(U_{end} - (v.w))$ obtained over 100 trials for each v, w pair. The errors are always to underestimate the value of $v.w$, due to a **while** loop being exited prematurely

4 Discussion

We have demonstrated a conceptual means by which Turing complete programs can be built using catalytic particles. The complexity of a program is likely in practice to be limited by the capacity to design particle complexes that can act as multi-state program counters. We make no claim that such serial programs exist in biology. If multiplication were to ever confer a fitness advantage, evolution would be more likely to find an analog solution.

Acknowledgments. We would like to thank Huub ten Eikelder for helpful discussion. Also we thank Jon Rowe and Peter Hilbers for guidance throughout the project. This work is supported by the European Community through the ESIGNET project of the Sixth Framework Programme.

References

1. Bartel, D., Szostak, J.: Isolation of new ribozymes from a large pool of random sequences. Science 261, 1411–1418 (1991)
2. Bray, D.: Signaling complexes: Biophysical constraints on intracellular communication. Ann. Rev. Biophys. Biomol. Struct. 27, 59–75 (1998)
3. Bray, D., Duke, T.: Conformational spread: The propogation of allosteric states in large multiprotein complexes. Ann. Rev. Biophys. Biomol. Struct. 33, 53–73 (2004)
4. Graham, I., Duke, T.: The logical repartoire of ligand-binding proteins. Phys. Biol. 2, 159–165 (2005)
5. Seelig, G., Yurke, B., Winfree, E.: Dna hybrization catalysts and catalyst circuits. DNA, 329–343 (2004)
6. Bray, D.: Protein molecules as computational elements in living cells. Nature 376, 307–312 (2004)
7. Arkin, A., Ross, J.: Computational functions in biochemical reaction networks. Biophys. J. 67, 560–578 (1994)
8. Hjelmfelt, A., Weinburger, E.D., Ross, J.: Chemical implementation of neural networks and turing machines. Proc. Natl. Acad. Sci. USA 88, 10983–10987 (1991)
9. Baron, R., Lioubashevski, O., Katz, E., Niazov, T., Willner, I.: Elementary arithmatic operations by enzymes: A paradigm for metabolic pathway-based computing. Angew. Chem. Int. Ed. (in press, 2006)
10. Magnasco, M.O.: Chemical kinetics is turing universal. Phys. Rev. Lett. 68, 1190–1193 (1997)
11. Winfree, E.: Dna computing by self-assembly. The Bridge 33(4) (2003)
12. Sauro, H.M., Kholodenko, B.N.: Quantitative analysis of signaling networks. Prog. Biophys. Mol. Biol. 86, 5–43 (2004)
13. Deckard, A., Sauro, H.M.: Preliminary studies on the in silico evolution of biochemical networks. Chembiochem. 5, 1423–1431 (2004)
14. Brookshear, J.G.: Theory of Computation, Formal Languages, Automata and Complexity. Benjamin-Cummings, Redwood City (1989)
15. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. J. Chem. Phys. 81(25), 2340–2361 (1977)
16. Sipper, M.: Evolution of Parallel Cellular Machines. Springer, Heidelberg (1997)
17. Adalsteinsson, D., McMillen, D., Elston, T.C.: Biochemical network stochastic simulator (bionets): software for stochastic modeling of biochemical networks. BMC Bioinformatics 5, 24 (2004)