RATE-INDEPENDENT CONSTRUCTS FOR CHEMICAL COMPUTATION*

PHILLIP SENUM and MARC RIEDEL

Electrical and Computer Engineering, University of Minnesota
Minneapolis, MN 55455
http://cctbio.ece.umn.edu
E-mail: {senu0004, mriedel}@umn.edu

This paper presents a collection of computational modules implemented with chemical reactions: an inverter, an incrementer, a decrementer, a copier, a comparator, and a multiplier. Unlike previous schemes for chemical computation, ours produces designs that are dependent only on coarse rate categories for the reactions ("fast" vs. "slow"). Given such categories, the computation is exact and independent of the specific reaction rates. We validate our designs through stochastic simulations of the chemical kinetics. Although conceptual for the time being, our methodology has potential applications in domains of synthetic biology such as biochemical sensing and drug delivery. We are exploring DNA-based computation via strand displacement as a possible experimental chassis.

Keywords: synthetic biology; molecular programming; molecular computing; chemical reaction networks

1. Introduction

The theory of reaction kinetics underpins our understanding of biological and chemical systems.[1] It is a simple and elegant formalism: chemical reactions define *rules* according to which reactants form products; each rule fires at a *rate* that is proportional to the quantities of the corresponding reactants that are present. On the computational front, there has been a wealth of research into efficient methods for simulating chemical reactions, ranging from ordinary differential equations (ODEs)[2] to stochastic simulation.[3] On the mathematical front, entirely new branches of theory have been developed to characterize the dynamics of chemical reaction networks.[4]

Most of this work is from the vantage point of *analysis*: a set of chemical reaction exists, designed by nature and perhaps modified by human engineers; the objective is to understand and characterize its behavior. Comparatively little work has been done at a conceptual level in tackling the inverse problem of *synthesis*: how can one design a set of chemical reactions that implement specific behavior?

Of course, chemical engineers, genetic engineers and other practitioners strive to create novel functionality all the time. Generally, they begin with existing processes and pathways, and modify these experimentally to achieve the desired new functionality.[5,6] In a sense, much of the theoretical work on the dynamics of chemical reactions also addresses the synthesis problem by delineating the range of behaviors that are possible. For instance, theoretical work has shown that fascinating oscillatory and chaotic behaviors can occur in chemical reaction networks.[7,8]

Perhaps the most profound theoretical observation is that chemical reaction networks are, in fact, *computational processes*: regardless of the complexity of the dynamics or the subtlety of

*This research is supported an NSF CAREER Award, #0845650.

the timing, such networks transform *input* quantities of chemical species into *output* quantities through simple primitive operations. The question of the computational power of chemical reactions has been considered.[9] (The answer is interesting and subtle: *stochastic* chemical reactions can compute any function – they are "Turing-universal" in the jargon of computer science. However, *deterministic* chemical reactions are not so powerful – they are not Turing-universal.)

One of the great successes of integrated circuit design has been in abstracting and scaling the design problem. The physical behavior of transistors is understood in terms of differential equations – say, with models found in tools such as SPICE.[10] However, the design of circuits occurs at more abstract levels – in terms of switches, gates, and modules. Many analogous levels of abstraction exist for biological systems. These range from molecular dynamics, to protein networks, to genetic regulatory networks, to signaling pathways, to complete cellular systems, to multicellular organisms. Several authors have made implicit or explicit connections between biochemical reactions and digital electronics.[11–13]

Our contribution is tackle the problem of computation with chemical reactions from a conceptual vantage point focusing on robustness. Unlike previous schemes for chemical computation, ours produces designs that are dependent only on coarse rate categories for the reactions ("fast" and "slow"). Given such categories, the computation is exact and independent of the specific reaction rates. In particular, it does not matter how fast any "fast" reaction is relative to another, or how slow any "slow" reaction is relative to another – only that "fast" reactions are fast relative to "slow" reactions.

In our prior and related work, we have described a variety of computational constructs with chemical reaction networks, including programming constructs such as "for" and "while" loops,[14] signal processing operations such as filtering,[15] and arithmetic operations such as multiplication, exponentiation and logarithms.[14]

In this paper, we present designs of chemical reaction networks that implement specific computational modules: inverters, incrementers, decrementers, copiers, comparators, and multipliers. In contrast to some of our earlier published constructs, all of these constructs depend on only two rate categories. Although conceptual for the time being, our methodology has potential applications in domains of synthetic biology such as biochemical sensing and drug delivery.

2. Chemical Model

We adopt the model of discrete, stochastic chemical kinetics.[3,16] Molecular quantities are whole numbers (i.e., non-negative integers). Reactions fire and alter these quantities by integer amounts. The reaction rates are proportional to (1) the quantities of the reacting molecular types; and (2) rate constants. We aim for robust constructs: systems that compute exact results independently of specific rate constants. All of our designs are formulated in terms of two coarse rate constant categories (e.g., "fast" and "slow"). Given such categories, the computation is exact and independent of the specific reaction rates.

Consider the reaction

$$X_1 \xrightarrow{\text{fast}} X_2 + X_3. \tag{1}$$

When this reaction fires, one molecule of X_1 is consumed, one of X_2 is produced, and one of X_3 is produced. (Accordingly, X_1 is called a *reactant* and X_2 and X_3 the *products*.) Consider what this reaction accomplishes from a computational standpoint. Suppose that it fires until all molecules of X_1 have been consumed. This results in quantities of X_2 and X_3 equal to the original quantity of X_1, and a new quantity of X_1 equal to zero:

```
X2 = X1
X3 = X1
X1 = 0
```

Consider the reaction

$$X_1 + X_2 \xrightarrow{\text{fast}} X_3. \tag{2}$$

Suppose that it fires until either all molecules of X_1 or all molecules of X_2 have been consumed. This results in a quantity of X_3 equal to the lesser of the two original quantities:

```
X3 = min(X1, X2)
X1 = X1 - min(X1, X2)
X2 = X2 - min(X1, X2)
```

We will present constructs different arithmetical and logical operations in this vein. Each sets the final quantity of some molecular type as a function of the initial quantities of other types. The challenge in setting up computation with chemical reactions is that they execute asynchronously and at variable rates, dependent on factors such as temperature. In spite of this, we aim to implement computation that does not depend on the rates. We will only speak of rates in qualitative terms, e.g., "fast" vs. "slow" (in our notation, such qualitative rates are listed above the arrows for reactions.)

We validate our designs through stochastic simulations of the chemical kinetics.[17] First proposed by Gillespie, stochastic simulation has become the workhorse of computational biology – the equivalent, one might say, of SPICE for electrical engineering.[10] Such simulation tracks integer quantities of the molecular species, executing reactions at random based on propensity calculations. Repeated trials are performed and the probability distribution of different outcomes is estimated by averaging the results.

3. Computational Constructs

In this section, we present a collection of constituent constructs for rate-independent computation: an inverter, an incrementer/decrementer, a copier, and a comparator. In the next section, we use some of these constructs to implement a multiplier.

An Inverter

We implement an operation that is analogous to that performed by an inverter (i.e., a NOT gate) in a digital system: given a non-zero quantity (corresponding to logical "1") we produce

a zero quantity (corresponding to logical "0"). Conversely, given a zero quantity we produce a non-zero quantity. We accomplish this with a pair of chemical types: the given type, call it a, and a corresponding "**absence indicator**" type, call it a_{ab}. The reactions generating the absence indicator are:

$$\varnothing \xrightarrow{\text{slow}} a_{\text{ab}} \tag{3}$$

$$a + a_{\text{ab}} \xrightarrow{\text{fast}} a \tag{4}$$

$$2\, a_{\text{ab}} \xrightarrow{\text{fast}} a_{\text{ab}} \tag{5}$$

Here the symbol $\varnothing$ as a reactant indicates that the reaction does not alter the quantity of the reactant types, perhaps because the quantity of these is large or replenishable.

The first reaction continuously generates molecules of a_{ab}, so in the absence of molecules of a we will have a non-zero quantity of a_{ab} in the system. If there are molecules of a present, then second reaction quickly clobbers any molecules of a_{ab} that are generated, so the quantity of a_{ab} will be close to zero. The third reaction ensures that the quantity a_{ab} remains small.

We use this simple construct in many of our computational modules.[15,18] In general, it can be used to synchronize steps. Suppose that we want to perform the following:

$$a \rightarrow b \tag{6}$$

$$b \rightarrow [\textit{operate on } b] \tag{7}$$

Here the second step is an operation that depends on the quantity of b. We do not want to start consuming molecules of b until the full quantity of it is generated from a. We can accomplish this with an absence indicator a_{ab}:

$$a \rightarrow b \tag{8}$$

$$a_{\text{ab}} + b \rightarrow [\textit{operate on } b] \tag{9}$$

3.1. *Increment and Decrement Operations*

We describe constructs to implement incrementation and decrementation. These operations form the basis of more complex arithmetical operations, such as multiplication. The inputs consist of two molecular types g, the "start signal," and x, the quantity to be incremented or decremented. We assume that some external source injects molecules of g. Any quantity can be injected; regardless, the quantity of x is incremented or decremented by exactly one. The system consumes all the molecules g. Once the quantity reaches zero, another increment/decrement operation can be performed. The operations proceed as follows:

1) The system waits for the start signal g to be some non-zero quantity.
2) It transfer the quantity of x to a temporary type x'.
3) It sets g to zero.
4) It transfers all but one molecule of x' back to x.

5a) For a decrement, it removes the last molecule x'.

5b) For an increment, it removes the last molecule of x' and adds to two molecules to x.

The following reactions implement this scheme. Given molecules of g, a reaction transfers molecules of x to molecules of x':

$$x + g \xrightarrow{\text{slow}} x' + g \tag{10}$$

The following reaction sets the quantity of g to zero. Using the absence indicator mechanism described in the preceding section, it does so only once all molecules of x have been transfered to x':

$$g + x_{\text{ab}} \xrightarrow{\text{slow}} \varnothing \tag{11}$$

Reactions of the form of 3– 5 are needed to generated x_{ab}; we omit them here. The following reaction transfers all but one molecule of x' back to x. It does so by repeatedly selecting pairs of x'. In essence, this is a repeated integer division by two. Again, using the absence indicator mechanism, it proceeds only once all molecules of g have been removed:

$$g'_{\text{ab}} + 2\,x' \xrightarrow{\text{fast}} x + x' + x'' \tag{12}$$

This reaction also produces molecules of a supplementary type x''. Note that this reaction is in the "fast" category. The new type x'' is consumed by the reaction:

$$x'' \xrightarrow{\text{slow}} \varnothing. \tag{13}$$

Note that this reaction is in the "slow" category. We introduce x'' because we cannot directly use an absence indicator for x' to detect when Reaction 12 has completed; here x' is not completed consumed. Instead, in reactions below we use an absence indicator for x''. Again, reactions of the form of 3– 5 are needed to generated x''_{ab}; we omit them here.

In Reaction 12, we do not directly use an absence indicator for g_{ab}, since that reaction is in the "fast" rate category. A design restriction for the absence indicator types is that they should never be directly involved in "fast" reactions. They are produced slowly and consumed quickly if the corresponding type is present in the system; if they were involved in a "fast" reaction, there would be competition. We avoid this by transferring the corresponding absence indicator g_{ab} to a secondary type g'_{ab} via a "slow" reaction:

$$\varnothing \xrightarrow{\text{slow}} g_{\text{ab}} \tag{14}$$

$$g_{\text{ab}} \xrightarrow{\text{slow}} g'_{\text{ab}} \tag{15}$$

We setup the absence indicator reactions for both types:

$$g + g_{\text{ab}} \xrightarrow{\text{fast}} g \tag{16}$$

$$g + g'_{\text{ab}} \xrightarrow{\text{fast}} g \tag{17}$$

$$2\,g_{\text{ab}} \xrightarrow{\text{fast}} g_{\text{ab}} \tag{18}$$

$$2\,g'_{\text{ab}} \xrightarrow{\text{fast}} g'_{\text{ab}}. \tag{19}$$

Finally, we include the following reaction to perform a decrement:

$$x''_{\text{ab}} + x' + g'_{\text{ab}} \xrightarrow{\text{slow}} \varnothing \quad \text{[Decrement]} \tag{20}$$

Or we include the following reaction to perform an increment:

$$x''_{\text{ab}} + x' + g'_{\text{ab}} \xrightarrow{\text{slow}} 2\,x \quad \text{[Increment]} \tag{21}$$

3.2. *A Copier*

In digital computation, one of the most basic operations is copying a quantity from one register into another. The programming construct is "set the value of b to be the value of a":

```
let b = a;
```

To implement an equivalent operation with chemical reactions, we could use a reaction that simply transfers the quantity of a to b:

$$a \rightarrow b \tag{22}$$

However, this is not ideal because this reaction consumes all the molecules of a, setting its quantity to zero. We would like a chemical construct that copies the quantity without altering it. The following reaction does not work either:

$$a \rightarrow a + b \tag{23}$$

It just creates more and more molecules of b in the presence of a. A more sophisticated construct is needed.

In our construct, we have a "request-to-copy" type cr. When an external source injects molecules of cr, the copy operation proceeds. (The quantity of cr that is injected is irrelevant.) It produces an output quantity of b equal to the input quantity of a. It leaves the quantity of a unchanged. The reactions for the copier construct are as follows. Firstly, in the presence of cr, a reaction transfers the quantity of a to a':

$$cr + a \xrightarrow{\text{slow}} cr + a' \tag{24}$$

After all molecules of a have been transferred to a', the system removes all the molecules of cr:

$$cr + a_{\text{ab}} \xrightarrow{\text{slow}} \varnothing \tag{25}$$

Here, again, we are using the concept of an absence indicator. (The symbol $\varnothing$ as a product indicates "nothing", meaning that the type degrades into products that are no longer tracked or used.) Removing cr ensures that a is copied exactly once. After cr has been removed, a reaction transfers the quantity of a' back to a and also creates this same quantity of b:

$$cr_{\text{ab}} + a' \xrightarrow{\text{slow}} a + b \tag{26}$$

We also generate absence indicators a_{ab} and cr_{ab} by the method described above. We note that, while this construct leaves the quantity of a unchanged after it has finished executing, it temporarily consumes molecules a, transferring the quantity of these to a', before transferring it back. Accordingly, no other constructs should use a in the interim.

3.3. *A Comparator*

Using our copier construct, we can create a construct that compares the quantities of two input types and produces an output type if one is greater than the other. For example, let us assume that we want to compare the quantities of two types a and b:

```
if  (a > b) {
    t = TRUE
} else {
    t = FALSE
}
```

If the quantity of a is greater than the quantity of b, the system should produce molecules of an output type t; otherwise, it should not produce any molecules of t.

Our construct for a comparator is as follows. First, we create temporary copies, c and d, of the types that we wish to compare, a and b, respectively, using the copier construct described in the previous section. (We omit these reactions; they are two verbatim copies of the copier construct, one with a as an input and c as an output, the other with b as an input and d as an output.) We split the copy request so that the two copiers are not competing for it:

$$cr \xrightarrow{\text{fast}} cr_1 + cr_2 \tag{27}$$

Now we compare a and b via their respective copies c and d. To start, we first consume pairs of c and d:

$$c + d \xrightarrow{\text{fast}} \varnothing \tag{28}$$

Note that this is a *fast* reaction; we assume that it fires to completion. The result is that there are only molecules of c left, or only molecules of d left, or no molecules of c nor d left. Molecules of the type that originally had a larger quantity have persisted. If the quantities were equal, then both types were annihilated. We use absence indicators c_{ab} and d_{ab} to determine which type was annihilated:

$$\varnothing \xrightarrow{\text{slow}} c_{\text{ab}} \tag{29}$$

$$c + c_{\text{ab}} \xrightarrow{\text{fast}} c \tag{30}$$

$$2\, c_{\text{ab}} \xrightarrow{\text{fast}} c_{\text{ab}} \tag{31}$$

$$\varnothing \xrightarrow{\text{slow}} d_{\text{ab}} \tag{32}$$

$$d + d_{\text{ab}} \xrightarrow{\text{fast}} d \tag{33}$$

$$2\, d_{\text{ab}} \xrightarrow{\text{fast}} d_{\text{ab}} \tag{34}$$

If a was originally greater than b, there will now be a presence of c and an absence of d. We produce molecules of type t if this condition is met. We preserve the quantities of c and d_{ab}; the amount t that we produce depends on the quantity of a fuel type:

$$\text{fuel} + c + d_{\text{ab}} \xrightarrow{\text{slow}} c + d_{\text{ab}} + t \tag{35}$$

For robustness, we also add a reaction to destroy t in the case that the asserted condition is not true:

$$c_{\text{ab}} + d + t \xrightarrow{\text{slow}} c_{\text{ab}} + d \tag{36}$$

$$c_{\text{ab}} + d_{\text{ab}} + t \xrightarrow{\text{slow}} c_{\text{ab}} + d_{\text{ab}} \tag{37}$$

We can readily generalize the construct to all types of logical comparisons. Table 1 lists these operations and their corresponding reactions.

Table 1. Logical operations via chemical reactions.

Operation	Creation	Destruction	Operation	Creation	Destruction
`a == b`	$a_{\text{ab}} + b_{\text{ab}}$	$a + b_{\text{ab}}$ $a_{\text{ab}} + b$	`a >= b`	$a + b_{\text{ab}}$ $a_{\text{ab}} + b_{\text{ab}}$	$a_{\text{ab}} + b$
`a > b`	$a + b_{\text{ab}}$	$a_{\text{ab}} + b$ $a_{\text{ab}} + b_{\text{ab}}$	`a <= b`	$a_{\text{ab}} + b$ $a_{\text{ab}} + b_{\text{ab}}$	$a + b_{\text{ab}}$
`a < b`	$a_{\text{ab}} + b$	$a + b_{\text{ab}}$ $a_{\text{ab}} + b_{\text{ab}}$	`a != b`	$a_{\text{ab}} + b$ $a + b_{\text{ab}}$	$a_{\text{ab}} + b_{\text{ab}}$

4. A Multiplier

Building upon the constructs in the last section, we show a construct that multiplies the quantities of two input types. Multiplication, of course, consists of iterative addition. Consider the following lines of pseudo-code:

```
while x > 0 {
   z = z + y
   x = x - 1
}
```

The result is that z is equal to x times y. We implement multiplication chemically using the constructs described in the previous sections: the line `z = z + y` is implemented with a copy operation; the line `x = x - 1` is implemented using a decrement operation. Only one additional reaction is needed to handle the `while` statement.

Firstly, we have reactions that copy the quantity of y to z. We use a "copy-request" sa type to synchronize iterations; it is supplied from the controlling reaction 52 below.

$$sa + y \xrightarrow{\text{slow}} sa + y' \tag{38}$$

$$sa + y_{\text{ab}} \xrightarrow{\text{slow}} \varnothing \tag{39}$$

$$sa_{\text{ab}} + y' \xrightarrow{\text{slow}} y + z \tag{40}$$

Secondly, we have reactions that decrement the value of x. We use sb as the signal to begin the decrement.

$$x + sb \xrightarrow{\text{fast}} x' + sb \tag{41}$$

$$sb + x_{\text{ab}} \xrightarrow{\text{slow}} \varnothing \tag{42}$$

$$sb_{\text{ab}} \xrightarrow{\text{slow}} sb'_{\text{ab}} \tag{43}$$

$$2\,x' + sb'_{\text{ab}} \xrightarrow{\text{fast}} x' + x + x'' \tag{44}$$

$$x'' \xrightarrow{\text{slow}} \varnothing \tag{45}$$

$$x' + x''_{\text{ab}} + sb'_{\text{ab}} \xrightarrow{\text{slow}} \varnothing \tag{46}$$

$$2\,sb'_{\text{ab}} \xrightarrow{\text{fast}} sb'_{\text{ab}} \tag{47}$$

$$sb'_{\text{ab}} + sb \xrightarrow{\text{fast}} sb \tag{48}$$

Thirdly, we have a controlling set of reactions to implement the `while` statement. This set

generates sa and sb to begin the next iteration, preserving the quantity of x:

$$x + x'_{\text{ab}} + y'_{\text{ab}} \xrightarrow{\text{slow}} x + \text{start} \tag{49}$$

$$\text{start} + x' \xrightarrow{\text{fast}} x' \tag{50}$$

$$\text{start} + y' \xrightarrow{\text{fast}} y' \tag{51}$$

$$\text{start} \xrightarrow{\text{slow}} sa + sb \tag{52}$$

This set initiates the next iteration of the loop if such an iteration is not already in progress and if there are still molecules of x in the system. The types x' and y' are present when we are decrementing x or copying y, respectively; thus, they can be used to decide whether we are currently inside the loop or not. Finally, we generate the four absence indicators according to the template in Reactions 3– 5.

5. Simulation Results

We validated our constructs using stochastic simulation. Specifically, we performed a time homogeneous simulation using Gillespie's "Direct Method"[3] with the software package "Cain" from Caltech.[19] In each case, the simulation was run until the quantities of all types except the absence indicators converged to a steady state. We used a rate constant of 1 for the "slow" reactions. We tried rate constants between two to four orders of magnitude higher for the "fast" reactions. (We refer to the ratio of "fast" to "slow" as the *rate separation.*) For each of the graphs below, the initial quantity of each type is zero, with the exception of the types specified.

5.1. *Multiplier*

Graph 1 shows the output of a single simulated trajectory for our multiplier. We observe exactly the behavior that we are looking for: the quantity of y cycles exactly 10 times as it exchanges with y' and is copied to z; the quantity of z grows steadily up to 100; the quantity of x decreases once each cycle down to 0. Table 2 presents detailed simulation results, this time tested for accuracy. Errors generally occur if the system executes too many or too few iterations. As can be seen, the larger the quantity of x, the more accurate the result, in relative terms. As expected, the larger the rate separation, the fewer errors we get.

Table 2. Statistical simulation results for "Multiplier" construct

Trial	Rate Separation	Trajectories	x	y	z	Expected z	Error
1	100	100	100	50	4954.35	5000	0.91%
2	100	100	50	100	4893.18	5000	2.14%
3	1000	100	100	50	4991.56	5000	0.17%
4	1000	100	50	100	4995.78	5000	0.08%
5	10000	100	100	50	4998.69	5000	$< 0.01\%$
6	10000	100	50	100	4999.14	5000	$< 0.01\%$
7	10000	100	10	20	200.04	200	$< 0.01\%$
8	10000	100	20	10	200.03	200	$< 0.01\%$

5.2. *Copier*

Graph 2 shows an average simulated trajectory for our copier. Again, we observe exactly the behavior we expect: the quantity of a drops to 0 almost immediately as it turns into a'; this is followed by the removal of cr from the system. When the quantity of cr drops to nearly zero, both a and b rise steadily back to the original quantity of a. Table 3 shows additional simulation results from our copier, this time tested for accuracy. The copier construct appears to be quite robust to errors; however, large rate separations do not help as much as they do for the multiplier. The system seems to prefer a larger injection quantity of cr, but whether it is larger or smaller than the initial quantity of a is irrelevant.

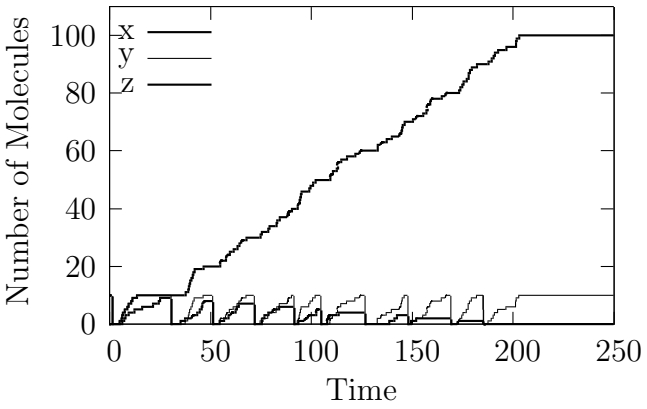
Table 3. Statistical simulation results for "Copier" construct

Trial	Rate Separation	Trajectories	cr	a	b	Expected b	Error
1	100	500	5	100	102.45	100	2.45%
2	100	500	50	100	104.826	100	4.826%
3	1000	500	5	100	100.312	100	0.312%
4	1000	500	50	100	100.516	100	0.516%
5	10000	500	5	100	100.022	100	0.022%
6	10000	500	50	100	100.034	100	0.034%
7	10000	500	5	5000	4938.39	5000	1.232%
8	10000	500	50	5000	4967.26	2	0.655%
9	10000	500	200	5000	4796.38	2	4.072%
10	10000	500	50	2	2	2	4.072%

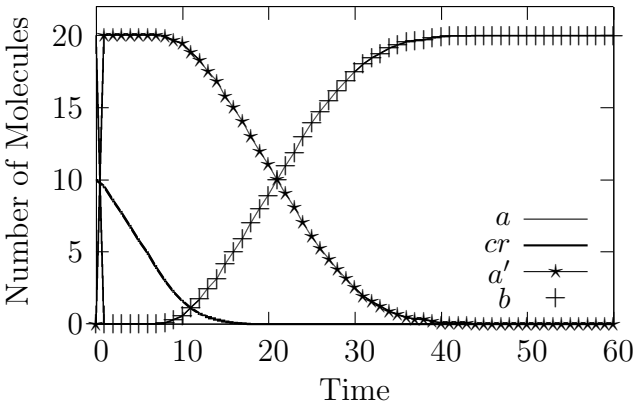
5.3. *Decrementer and Comparator*

Graph 3 shows the output of a single simulated trajectory of our decrementer. Exactly twenty peaks can be seen in the graph, including the initial peak on the far-left margin of the graph. This is exactly the behavior we are looking for – a decrement by exactly one each cycle. Graphs 4 and 5 display simulation results from our comparator. In Graph 4, t is asserted as we would expect; in Graph 5, t is not asserted, also as we would expect.

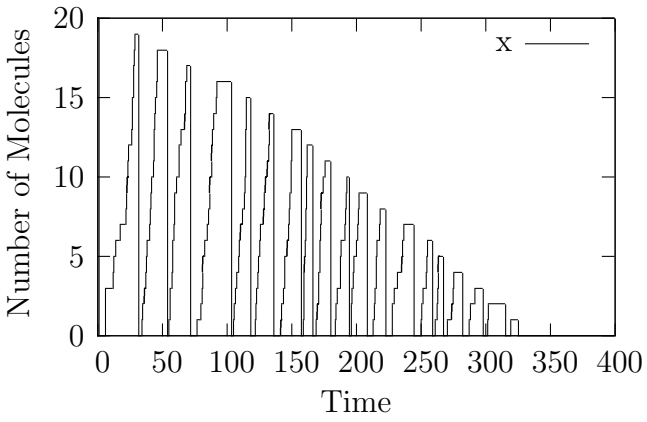
Graph 1: Simulated Multiplier, $x = 10$, $y = 10$



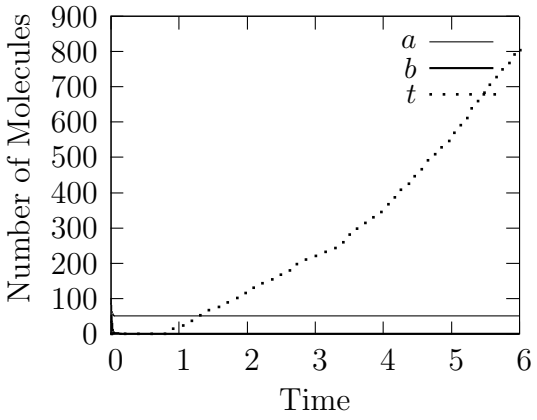
Graph 2: Simulated Copier, $a = 20$, $cr = 10$



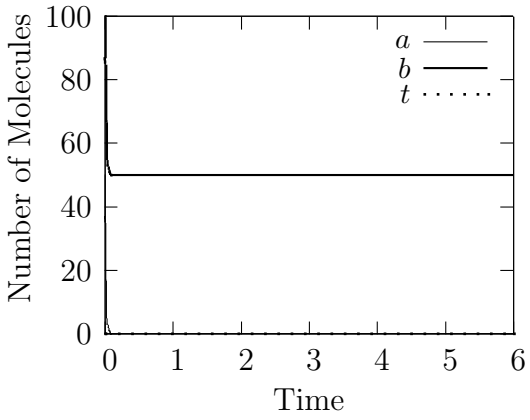
Graph 3: Simulated Decrement, $x = 20$



Graph 4: Comparator $(a > b)$, $a = 100$, $b = 50$



Graph 5: Comparator $(a > b)$, $a = 50$, $b = 100$



6. Discussion

Our contribution is to tackle the problem of synthesizing computation at a conceptual level, working not with actual molecular types but rather with abstract types. One might question whether actual chemical reactions matching our templates can be found. Certainly, engineering complex new reaction mechanisms through genetic engineering is a formidable task; for *in vivo* systems, there are likely to be many experimental constraints on the choice of reactions.[20] However, we point to recent work on *in vitro* computation as a potential application domain for our ideas.

Through a mechanism called DNA strand-displacement, a group at Caltech has shown that DNA reactions can emulate the chemical kinetics of nearly any chemical reaction network. Indeed, they provide a compiler that translates abstract chemical reactions of the sort that we design into specific DNA reactions.[21] Recent work has demonstrated both the scale of computation that is possible with DNA-based computing,[22] as well as exciting applications.[23] While conceptual, our work suggest a *de novo* approach to the design of biological functions. Potentially this approach is more general in its applicability than methods based on appropriating and reusing existing biological modules.

References

1. F. Horn and R. Jackson, "General mass action kinetics," *Archive for Rational Mechanics and Analysis*, vol. 47, pp. 81–116, 1972.

2. P. Érdi and J. Tóth, *Mathematical Models of Chemical Reactions: Theory and Applications of Deterministic and Stochastic Models*. Manchester University Press, 1989.
3. D. Gillespie, "Exact stochastic simulation of coupled chemical reactions," *Journal of Physical Chemistry*, vol. 81, no. 25, pp. 2340–2361, 1977.
4. S. Strogatz, *Nonlinear Dynamics and Chaos with Applications to Physics, Biology, Chemistry, and Engineering*. Perseus Books, 1994.
5. M. Win, J. Liang, and C. Smolke, "Frameworks for programming biological function through RNA parts and devices," *Chemistry & Biology*, vol. 16, pp. 298–310, 2009.
6. J. Keasling, "Synthetic biology for synthetic chemistry," *ACS Chemical Biology*, vol. 3, pp. 64–76, 2008.
7. I. R. Epstein and J. A. Pojman, *An Introduction to Nonlinear Chemical Dynamics: Oscillations, Waves, Patterns, and Chaos*. Oxford Univ Press, 1998.
8. K. D. Willamowski and O. E. Rössler, "Irregular oscillations in a realistic abstract quadratic mass action system," *Zeitschrift fur Naturforschung Section A – A Journal of Physical Sciences*, vol. 35, pp. 317–318, 1980.
9. D. Soloveichik, M. Cook, E. Winfree, and J. Bruck, "Computation with finite stochastic chemical reaction networks," *Natural Computing*, vol. 7, no. 4, 2008.
10. L. Nagel and D. Pederson, "Simulation program with integrated circuit emphasis," in *Midwest Symposium on Circuit Theory*, 1973.
11. R. Weiss, G. E. Homsy, and T. F. Knight, "Toward in vivo digital circuits," in *DIMACS Workshop on Evolution as Computation*, 1999, pp. 1–18.
12. J. C. Anderson, C. A. Voigt, and A. P. Arkin, "A genetic AND gate based on translation control," *Molecular Systems Biology*, vol. 3, no. 133, 2007.
13. Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, and E. Shapiro, "An autonomous molecular computer for logical control of gene expression," *Nature*, vol. 429, no. 6990, pp. 423–429, 2004.
14. A. Shea, B. Fett, M. D. Riedel, and K. Parhi, "Writing and compiling code into biochemistry," in *Proceedings of the Pacific Symposium on Biocomputing*, 2010, pp. 456–464.
15. H. Jiang, A. P. Kharam, M. D. Riedel, and K. K. Parhi, "A synthesis flow for digital signal processing with biomolecular reactions," in *IEEE International Conference on Computer-Aided Design*, 2010.
16. D. T. Gillespie, "Stochastic simulation of chemical kinetics," *Annual Review of Physical Chemistry*, vol. 58, pp. 35–55, 2006.
17. ——, "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions," *Journal of Computational Physics*, vol. 22, no. 4, pp. 403–434, 1976.
18. A. Kharam, H. Jiang, M. D. Riedel, and K. Parhi, "Binary counting with chemical reactions," in *Pacific Symposium on Biocomputing*, 2011.
19. S. Mauch and M. Stalzer, "Efficient formulations for exact stochastic simulation of chemical systems," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 99, 2009.
20. R. Weiss, "Cellular computation and communications using engineering genetic regulatory networks," Ph.D. dissertation, MIT, 2003.
21. D. Soloveichik, G. Seelig, and E. Winfree, "DNA as a universal substrate for chemical kinetics," *Proceedings of the National Academy of Sciences*, vol. 107, no. 12, pp. 5393–5398, 2010.
22. L. Qian and E. Winfree, "A simple DNA gate motif for synthesizing large-scale circuits," in *DNA Computing*, 2009, pp. 70–89.
23. S. Venkataramana, R. M. Dirks, C. T. Ueda, and N. A. Pierce, "Selective cell death mediated by small conditional RNAs," *Proceedings of the National Academy of Sciences*, 2010 (in press).