Module Locking in Biochemical Synthesis

Brian Fett and Marc D. Riedel

Department of Electrical and Computer Engineering
University of Minnesota
200 Union St. S.E., Minneapolis, MN 55455
{fett, mriedel}@umn.edu

***Abstract*—We are developing a framework for computation with biochemical reactions with a focus on synthesizing specific logical functionality, a task analogous to technology-independent logic synthesis. Our method synthesizes biochemical reactions that compute output quantities of molecular types as a function of input quantities, either deterministically or probabilistically. An important constraint is the timing, captured in the relative rates of the biochemical reactions: all the outputs of a given phase must be produced before the next phase can begin consuming them as inputs. To achieve this synchronization, the reaction rates must sometimes be separated by orders of magnitude: some much faster than others, some much slower. This might be costly or infeasible given a specific library of biochemical reactions.**

In this paper, we describe a novel mechanism for *locking* the computation of biochemical modules – analogous to *handshaking* mechanisms in asynchronous circuit design. With locking, our method synthesizes robust computation that is nearly rate independent, requiring at most two speeds ("fast" and "slow"). The trade-off is with respect to the size of the solution: more reactions are needed. We characterize this trade-off for inter- and intra-module locking in general and for a variety of specific modules that we have designed. In particular, we discuss locking in detail for a stochastic module that implements probabilistic computation, producing different combinations of molecular types according to specified probability distributions.

I. Introduction

A. Bio-Design Automation

In the nascent field of synthetic biology, practitioners are striving to create new form and functionality in biological systems through genetic manipulations. Recent accomplishments portend a coming revolution in the biosciences. From *Salmonella* that secretes spider silk proteins [26], to yeast that degrades biomass into ethanol [22], to *E. coli* that produces antimalarial drugs [21], the potential impacts are far-reaching. Still in its early stages, the field has been driven by experimental expertise; the remarkable exploits are attributable to the skill of the researchers in specific domains of biology. There has been a concerted effort to assemble repositories of standardized components [3]. However, creating and integrating synthetic components remains an *ad hoc* process.

The field of synthetic biology is now reaching a stage where it calls for computer-aided design tools (an effort dubbed "biodesign automation" [19]). The EDA community has unique expertise to contribute to this endeavor. As with integrated circuits, the key in synthetic biology is to develop design flows that systematically explore configurations at different levels of abstraction. Randomness is inherent to biochemistry: at each instant, the sequence of reactions that fires is a matter of chance. We argue that biochemical design problems can be cast in terms of discrete, probabilistic computation performed on protein quantities.

In prior work, we have described a modular framework for synthesizing computation with biochemical reactions – performing a task for synthetic biology analogous to technology-independent logic synthesis in circuit design [6]. This includes a flexible toolkit of functional modules for standard arithmetic operations (analogous to those performed by an ALU) as well as regulatory functions (analogous to those performed by control circuitry).

B. Timing in Biochemical Computation

An important constraint in our design methodology is the *timing*, captured in the relative rates of the biochemical reactions. Both for intra- and inter-module computation, strict synchronization is often required: all the outputs of a given phase must be produced before the next phase can begin consuming them. To achieve this synchronization, the reaction rates must sometimes be separated by orders of magnitude: some much faster than others, some much slower. This might be costly or infeasible given a specific library of biochemical reactions. Consider the following basic design problem:

Example 1
Suppose that we wish to design a *multiplication* module: a set of biochemical reactions that produces an output quantity of a type z that is proportional to the input quantities of both type x and type y,

$$|z| = |x| \cdot |y|.$$

(We use the notation $|\ |$ to refer to the *quantity* of the corresponding type.) The following set of reactions accomplishes this [6]:

$$\begin{aligned} x &\xrightarrow{\text{slowest}} a \\ a + y &\xrightarrow{\text{fastest}} a + y' + z \\ a &\xrightarrow{\text{fast}} \varnothing \\ y' &\xrightarrow{\text{slow}} y. \end{aligned} \tag{1}$$

The notation $\longrightarrow$ simply specifies a rule regarding how types of molecules, designated as the *reactants*, combine to produce other types, designated as the *products*, e.g., "two of hydrogen"

and "one of oxygen" combine to produce "one of water". The rates are relative: if a "fast" reaction can fire, it is assumed to do so – repeatedly, until it runs out of reactants – before a "slow" reaction ever fires. Depending on the quantities involved, and the accuracy that is called for, an order of magnitude difference between two rate categories might suffice. Typically, it would be several orders of magnitude. Here a and y' are intermediate types; it is assumed that no molecules of these types are present initially. (The symbol $\varnothing$ as a product indicates "nothing", meaning that the type degrades into products that are no longer tracked or used.)

To see that these reactions implement multiplication, note that no reaction can fire until the first one does, producing a molecule of type a. When it does, it initiates an iteration of a *loop*: the quantity of z increases as the second reaction fires repeatedly until there is no more y remaining. Once this process terminates, the third and fourth reactions fire, ending the iteration and restoring y to its initial value. In each iteration, the quantity of x is decremented by one and the quantity of z is incremented by y. The final result is a quantity of z equal to the initial quantity of x times the initial quantity of y. □

Now suppose that we wanted to use this multiplication module as part of a larger computation, say that of a polynomial. Suppose that the operands x and y are produced by other modules. There is no issue if x trickles in as the multiplication is being performed; the computation will loop the correct number of times regardless. However, the full quantity of y must be present at the start; otherwise, the result of the computation will be incorrect.

Two solutions to this problem exist. The first is to enforce the synchronization by having earlier stages in the computation complete much faster than later stages: given two modules, the first producing an input for the second, one must select reactions such that the initiating reaction of the first module is much faster than in the second. Although typically there is considerable leeway to choose among reactions with differing rates, this solution clearly does not scale well.

C. Module Locking

In this paper, we discuss a more efficient scheme for synchronization that we call *locking*. It entails adding reactions involving a specific molecular type to each module – the module's *key*. Without the key, the sequence of reactions in the module is prevented from firing. Each module's key is produced by a reaction involving another type – the module's *keysmith*. The keysmiths for the different modules are produced slowly and randomly. When one pops into existence, it is generally consumed by another reaction involving types called *indicators*. If none of these indicators are present, then it is this module's turn to execute. Only then does the keysmith produce the key for the corresponding module. The set of reactions are:

$$\begin{aligned} \varnothing &\xrightarrow{\text{slow}} \text{keysmith} \\ \text{keysmith} &\xrightarrow{\text{slow}} \text{key} \\ \text{keysmith} + \text{indicator} &\xrightarrow{\text{fast}} \text{indicator}. \end{aligned} \tag{2}$$

(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; in such cases we can assume that the quantity is simply unity and adjust the rate accordingly). If there are multiple input dependencies in a module, we will need an indicator and an associated reaction for each. The first reaction of a locked module must be modified so that it depends on the key:

$$\text{key} + \text{reactants} \rightarrow \text{products} + \text{key}. \tag{3}$$

Typically, the key will be a catalyst, appearing as both a reactant and a product, but this need not be the case.

This mechanism for locking is, in fact, analogous to *handshaking* mechanisms in asynchronous circuit design, with the creation of a keysmith acting as a probe and the generation of a key being an affirmative response [15]. With locking, our method synthesizes robust computation that is nearly rate independent, requiring at most two speeds ("fast" and "slow"). The trade-off is with respect to the size of the solution: more reactions are needed. We characterize this trade-off: for inter-module locking, for intra-module locking on a variety of examples, and for the stochastic module specifically.

II. Background

A. Discrete Biochemistry

Interesting biochemistry typically involves complex molecules such as proteins and enzymes. Within the confines of a cell, the *quantities* of such molecules are often surprisingly small: on the order of tens, hundreds, or thousands of molecules of each type. At this scale, individual reactions matter, and the problem must be analyzed discretely [8]. The complexity stems from the dynamics at play among the multitude of coupled reactions. Randomness is inherent: at each instant, the exact sequence of reactions that fires next is a matter of chance.

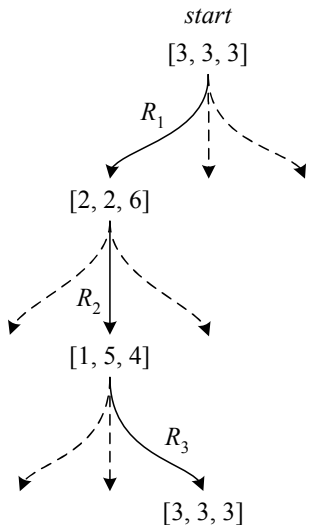

Fig. 1. Biochemical Reactions as Discrete Events – Beginning from the state $[3, 3, 3]$, R_1 fires, followed by R_2, followed by R_3.

Example 2
Consider a system with three types of molecules x_1, x_2, and x_3. The *state* of the system is described by

$$[|x_1|, |x_2|, |x_3|],$$

where $|x_1|, |x_2|$, and $|x_3|$ are the numbers of molecules of types x_1, x_2, and x_3, respectively, as non-negative integer quantities. For instance, the system might be in the state $[3, 3, 3]$ with three molecules of each type. Consider the three reactions:

$$\begin{aligned} R_1: &\quad x_1 + x_2 \rightarrow 3x_3 \\ R_2: &\quad x_1 + 2x_3 \rightarrow 3x_2 \\ R_3: &\quad 2x_2 + x_3 \rightarrow 2x_1. \end{aligned}$$

Note that these reactions are *coupled*: the types appear both as reactants and as products in different reactions. Suppose that the system is in the state $[3, 3, 3]$ and reaction R_1 fires. One molecule of type x_1 and one of type x_2 are consumed; three of type x_3 are produced. This results in the state transition:

$$[3, 3, 3] \xrightarrow{R_1} [2, 2, 6].$$

As reactions fire, a cellular process follows a sequence of such transitions. Figure 1 illustrates the trajectory taken from the state $[3, 3, 3]$ by the sequence R_1, R_2, and R_3. □

B. Probabilistic Biochemistry

Ignoring environmental changes outside the cell, one can assume cellular biochemistry behaves as a *Markov process*: the probability of future events depends only on the present state of the cell. Indeed, at each point in time, the probability of a given reaction firing is a function of the quantities of different types of molecules present. Specifically, it is proportional to:

- the number of ways that the reactants can come together,
- and the reaction *rate*.

Although we will often refer to rates in relative and qualitative terms – e.g.,"fast" vs. "slow" – these are, in fact, real-valued parameters that are either deduced from biochemical principles or measured experimentally [10].

Example 3
Suppose that the system in Example 2 is in the state $S = [3, 4, 5]$. There are

$$3 \times 4 = 12, \quad 3 \times \binom{5}{2} = 30, \quad \binom{4}{2} \times 5 = 30$$

ways to choose the reactants of R_1, R_2, and R_3, respectively. Suppose that the rates of reactions R_1, R_2, and R_3 are 1, 2 and 3, respectively. Then the firing probabilities for R_1, R_2, and R_3 are

$$\frac{12 \times 1}{162} = 0.074, \quad \frac{30 \times 2}{162} = 0.370, \quad \frac{30 \times 3}{162} = 0.556,$$

respectively. □

Randomness has been well characterized in biological systems [13], [17]. Certain biochemical systems appear to exploit randomness, choosing between different outcomes with a probability distribution – in effect, hedging their bets with a portfolio of responses. Examples include the *pap pili* epigenetic response of bacteria [11], the lentiviral positive-feedback loop in the HIV virus [25], and the lysis/lysogeny switch of the *lambda* bacteriophage [1].

Computationally, such discrete probabilistic biochemical systems are characterized through *Monte Carlo* simulation [8], [7], [14]. Beginning from an initial state, reactions are chosen at random, based on propensity calculations. As reactions fire, the quantities of the different species change by integer amounts. Repeated trials are performed and the probability distribution of different outcomes is estimated by averaging the results.

C. Synthesizing Probabilistic Behavior

As in natural systems, randomness plays a pivotal role in synthetically engineered systems. Prior work discussed a method for designing a set of biochemical reactions that produces different combinations of molecular types according to a specified probability distribution [6]. Here is an abbreviated description of a stochastic module. Suppose that we are targeting n distinct outcomes, characterized by the mutually exclusive production of types $O_1, \ldots, O_n$ according to a probability distribution. For i from 1 to n, we have five reactions:

$$\begin{aligned} \forall i: &\quad e_i \xrightarrow{\text{slow}} d_i \\ \forall i: &\quad d_i + e_i \xrightarrow{\text{fast}} 2d_i \\ \forall i \neq j: &\quad d_i + e_j \xrightarrow{\text{fast}} d_i \\ \forall i \neq j: &\quad d_i + d_j \xrightarrow{\text{fastest}} \varnothing \\ \forall i: &\quad d_i \xrightarrow{\text{slow}} d_i + O_i \end{aligned} \tag{4}$$

For each i, the first reaction in the set initiates the response by producing a catalyst type d_i. Among the i, the first one to fire this way generally determines the outcome. The other reactions ensure that once this first reaction fires, producing a molecule of d_i, this choice quickly wins out: the production of more molecules of d_i is encouraged, while the production of the other types d_j, $j \neq i$, is strongly inhibited. So the firing probabilities for the initializing reactions at the outset dictate the probability distribution of the final outcome.

The response to this system is precise and robust to perturbations. Furthermore, it is programmable: the probability distribution is a function of the quantities of input types [6]. Our contribution is design automation for biochemical synthesis at the level of abstract arbitrary types (a, b, c, etc.) – "technology-independent synthesis." The solution is then mapped to specific types and reactions from a suitable toolkit [3], [5].

III. Module Locking

We discuss schemes for locking the looping mechanisms in modules as well as locking successive modules that are chained together. We also discuss specifically an efficient scheme for locking the stochastic module.

A. Intra-Module Locking

Most of the modules found in [6] behave similarly to that in Example 1: there is a looping construct that iteratively works toward the correct answer. In all such modules, it is important that the reactions fire in the correct order. In addition to unlocking the reactions when it is time for them to execute, we must also have the ability to "re-lock" them when it is time for them to stop executing.

Our scheme involves adding a key requirement to most of the reactions in the loops of modules. Keysmiths are produced occasionally; if other keys are present, they quickly disappear – before they can produce their key. Only if no other keys are present will they produce their key. This ensures that at most one type of key is present (thus allowing only one part of the loop to fire at a time); also it ensures that only one key of that type is present (thus allowing for re-locking).

The set of reactions for this functionality is:

$$\begin{aligned}
\forall i: \quad & \varnothing & \xrightarrow{\text{slow}} \quad & \text{keysmith}_i \\
\forall i: \quad & \text{keysmith}_i & \xrightarrow{\text{slow}} \quad & \text{key}_i \\
\forall i: \quad & \text{key}_i & \xrightarrow{\text{slow}} \quad & \varnothing \\
\forall i,j: \quad & \text{key}_i + \text{keysmith}_j & \xrightarrow{\text{fast}} \quad & \text{key}_i.
\end{aligned} \tag{5}$$

Example 4
Figure 2 gives a locked version of the multiplication module in Example 1. The lines separate the reactions into three sets: the original reactions, the generic module-locking reactions, and some locking reactions specific to this example.

1) In the first set, notice that we do not add a lock to the second reaction; this is because the reaction is already locked by the "looping" type a. Notice, also, that the first and third reactions destroy the keys that they require. This prevents them from firing more than once. The fourth reaction does not destroy its key, since it fires repeatedly.
2) In the second set, the first three reactions produce the appropriate keysmiths; the next three allow those keysmiths to create keys; the next three cause keys that are no longer needed to disappear; the next nine ensure that there are no keysmiths left to create keys when there is already one in the system.
3) In the third set, the reactions ensure that no incorrect key will be produced. What is required here is that the keysmith of a locked reaction not appear when any of the other (related) locked reactions could be firing.

□

In general, most modules employing looping constructs will have four parts to the loop: a loop initiator, loop actions, a loop closer, and a loop reset. The loop actions are all reactions that require, but do not destroy, the looping type (a in the example above); no changes need to be made to lock these reactions. The loop initiator creates the looping type, while the loop closer destroys it. Both of these parts require a key that is destroyed by the reactions because they should only occur once per loop. The loop-reset reactions do not involve the loop type in any way; each will require a third, shared key, but they need not destroy it.

$$\begin{aligned}
\text{key}_1 + x & \xrightarrow{\text{slow}} a \\
a + y & \xrightarrow{\text{slow}} a + y' + z \\
\text{key}_2 + a & \xrightarrow{\text{slow}} \varnothing \\
\text{key}_3 + y' & \xrightarrow{\text{slow}} \text{key}_3 + y \\
\hline
\varnothing & \xrightarrow{\text{slow}} \text{keysmith}_1 \\
\varnothing & \xrightarrow{\text{slow}} \text{keysmith}_2 \\
\varnothing & \xrightarrow{\text{slow}} \text{keysmith}_3 \\
\text{keysmith}_1 & \xrightarrow{\text{slow}} \text{key}_1 \\
\text{keysmith}_2 & \xrightarrow{\text{slow}} \text{key}_2 \\
\text{keysmith}_3 & \xrightarrow{\text{slow}} \text{key}_3 \\
\text{key}_1 & \xrightarrow{\text{slow}} \varnothing \\
\text{key}_2 & \xrightarrow{\text{slow}} \varnothing \\
\text{key}_3 & \xrightarrow{\text{slow}} \varnothing \\
\text{key}_1 + \text{keysmith}_1 & \xrightarrow{\text{fast}} \text{key}_1 \\
\text{key}_1 + \text{keysmith}_2 & \xrightarrow{\text{fast}} \text{key}_1 \\
\text{key}_1 + \text{keysmith}_3 & \xrightarrow{\text{fast}} \text{key}_1 \\
\text{key}_2 + \text{keysmith}_1 & \xrightarrow{\text{fast}} \text{key}_2 \\
\text{key}_2 + \text{keysmith}_2 & \xrightarrow{\text{fast}} \text{key}_2 \\
\text{key}_2 + \text{keysmith}_3 & \xrightarrow{\text{fast}} \text{key}_2 \\
\text{key}_3 + \text{keysmith}_1 & \xrightarrow{\text{fast}} \text{key}_3 \\
\text{key}_3 + \text{keysmith}_2 & \xrightarrow{\text{fast}} \text{key}_3 \\
\text{key}_3 + \text{keysmith}_3 & \xrightarrow{\text{fast}} \text{key}_3 \\
\hline
\text{keysmith}_1 + y' & \xrightarrow{\text{fast}} y' \\
\text{keysmith}_1 + a & \xrightarrow{\text{fast}} a \\
\text{keysmith}_2 + y & \xrightarrow{\text{fast}} y \\
\text{keysmith}_3 + a & \xrightarrow{\text{fast}} a.
\end{aligned}$$

Fig. 2. A "Locked" Version of the Multiplication Module from Example 1.

Both generic module locking reactions (i.e., those in Equation 5) as well as some reactions specific to the module must be added. These include reactions for:

- destroying the keysmith for the loop initiator if the looping molecule is present,
- destroying the keysmith for the loop initiator if any loop reset reactions can fire,
- destroying the keysmith for the loop closer if any loop actions can take place,
- destroying the keysmith for the loop reset reactions if the looping molecule is present.

With these modifications, the only requirement on the rate of the reactions is that all reactions that destroy a keysmith be "fast" and the others "slow."

Figure 3 compares the accuracy of the locked vs. unlocked versions of various functional modules using different separations in the rate constants. For the unlocked version, we used the rates $1, \lambda, \lambda^2$, and λ^3 as the values for "slowest," "slow,"

"fast," and "fastest." For the locked modules, we used 1 and λ for "slow" and "fast." Note that the total range of rates in the unlocked case is λ^3. For a fair comparison, we defined the "accuracy gain" for the scheme to be the error of the unlocked method at $\lambda = 10$ divided by the error of the locked scheme at $\lambda = 1000$, since both sets of reactions would then require that the fastest reaction be 1000 times faster than the slowest.

Multiplication: 10×10		
	%error	
λ	unlocked	locked
1	77.75%	48.46%
10	27.07%	24.67%
100	4.20%	3.09%
1000	0.45%	0.33%
10000	0.05%	0.02%
# of reactions	4	26
Accuracy gain: 82.03×		

Exponentiation: 2^5		
	%error	
λ	unlocked	locked
1	75.02%	N/A
10	18.027%	N/A
100	2.37%	33.72%
1000	0.25%	2.58%
10000	0.02%	0.26%
# of reactions	4	26
Accuracy gain: 6.99×		

Logarithm: $log_2(64)$		
	%error	
λ	unlocked	locked
1	267.03%	169.99%
10	41.36%	69.89%
100	5.24%	11.57%
1000	0.53%	1.27%
10000	0.05%	0.14%
# of reactions	6	30
Accuracy gain: 32.56×		

Fig. 3. A Comparison of the Accuracy of the Locked and Unlocked Versions of Three Modules: Multiplication, Exponentiation, and Logarithm.

It is interesting to note that the unlocked versions of multiplication and exponentiation tend to under-compute the result, whereas our locked versions tend to over-compute it. This is because, in the unlocked case, the error that occurs is removing the loop molecule prematurely; in the locked case, the error comes from allowing the loop to reset while active.

B. Inter-Module Locking

To implement more complex biochemical computation, the simple modules outlined in [6] can be nested, with one module performing an arithmetic operation on an input and passing it to the next module. This generally requires the first module to complete execution prior to the second starting. As we suggested in the introduction, this either necessitates multiple speeds or else locking.

The first step for module locking is to identify all molecular types that indicate that the locked module should not be firing. Then, the reactions outlined in Equation 2 are added. Finally, the loop-initiator reaction is changed so that it requires the appropriate key, that is, one that will not be produced in the presence of the specified indicators. We illustrate with an example.

Example 5
Consider the simple case of a "linear" module,

$$|y| = |x|.$$

Suppose that the output of this module is the input to a stochastic module that produces an outcome A with probability $p_1 = y/100$ and an outcome B with probability $p_2 = (100 - y)/100$. Without locking, the reactions are:

$$\begin{aligned}
x &\xrightarrow{\text{fast}} y \\
y + e_2 &\xrightarrow{\text{fast}} e_1 \\
e_1 &\xrightarrow{\text{slow}} d_1 \\
e_2 &\xrightarrow{\text{slow}} d_2 \\
e_1 + d_1 &\xrightarrow{\text{fast}} 2d_1 \\
e_2 + d_2 &\xrightarrow{\text{fast}} 2d_2 \\
d_1 + d_2 &\xrightarrow{\text{fastest}} \varnothing \\
d_1 + &\xrightarrow{\text{slow}} d_1 + A \\
d_2 + &\xrightarrow{\text{slow}} d_2 + B.
\end{aligned} \tag{6}$$

Here e_1 and e_2 are initialized to 0 and 100, respectively. Modifying the reactions such that the stochastic module is locked until the linear module completes, the reactions are:

$$\begin{aligned}
x &\xrightarrow{\text{slow}} y \\
y + e_2 &\xrightarrow{\text{slow}} e_1 \\
\text{key} + e_1 &\xrightarrow{\text{slow}} \text{key} + d_1 \\
\text{key} + e_2 &\xrightarrow{\text{slow}} \text{key} + d_2 \\
e_1 + d_1 &\xrightarrow{\text{fast}} 2d_1 \\
e_2 + d_2 &\xrightarrow{\text{fast}} 2d_2 \\
d_1 + d_2 &\xrightarrow{\text{fastest}} \varnothing \\
d_1 + &\xrightarrow{\text{slow}} d_1 + A \\
d_2 + &\xrightarrow{\text{slow}} d_2 + B \\
\varnothing &\xrightarrow{\text{slow}} \text{keysmith} \\
\text{keysmith} &\xrightarrow{\text{slow}} \text{key} \\
x + \text{keysmith} &\xrightarrow{\text{fast}} x \\
y + \text{keysmith} &\xrightarrow{\text{fast}} y.
\end{aligned} \tag{7}$$

Here we have added four reactions, two for the key generation and one for each of the indicator molecules, x and y.

Figure 4 compares the locked version to the original version. Curves of the probabilistic response for outcome A are plotted for different rates. We see how effective locking is, even if "fast" is only twice as fast as "slow." □

Similar examples are shown for both logarithm and exponentiation in Figure 5 and Figure 6, respectively. While the low quantities involved in the logarithm example do not play to

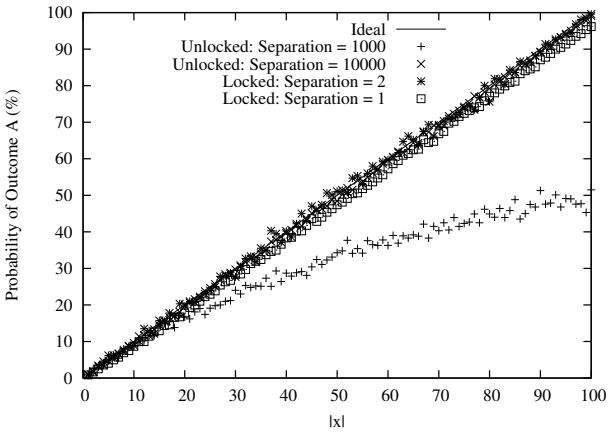

Fig. 4. Inter-module Locking – the probabilistic response of the locked and unlocked versions of the modules in Example 5. The first is for the unlocked version with the rates "slow," "fast," and "fastest," each separated by a factor of 1000; the next with these rates separated by a factor of 10,000; the next for the locked version with rates "slow" and "fast" separated by a factor of 2; and the last with these rates identical.

the strength of this scheme, the exponentiation example shows how even with *no* rate separation, a better approximation to the true value is reached than with a rate separation of 1000 in the unlocked case.

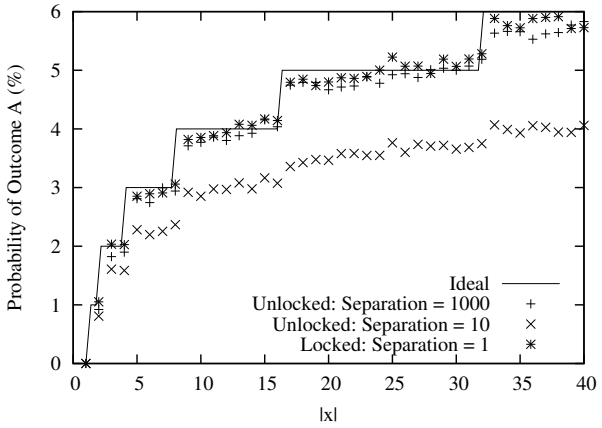

Fig. 5. Inter-module Locking (Logarithm) – an example with a logarithm module feeding a stochastic module; the correct value of the probability is the ceiling of the base two logarithm in percent (shown as ideal).

C. Locking the Stochastic module

The stochastic module described in Section II-C is at the core of our synthesis methodology [6]. In order to produce different combinations of molecular types according to a specified probability distribution, layers of amplification and inhibition are needed. The method necessitates reactions with widely separated rates to accomplish this. Here we propose an alternative approach, based on locking. All initialization reactions share one key; as such, the choice between the reactions is made independently of the types and quantities of keys present. Thus, the probability distribution is still a function of the input types.

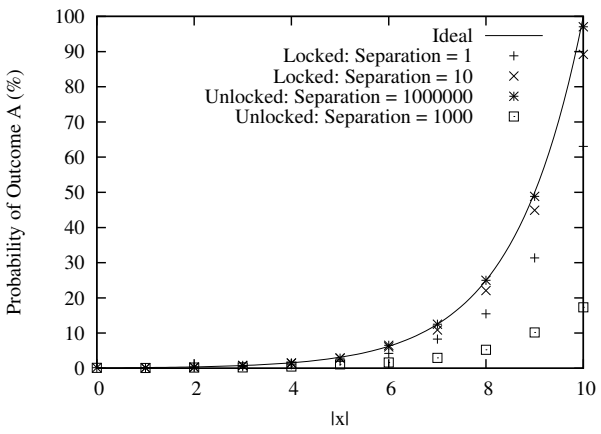

Fig. 6. Inter-module Locking (Exponentiation) – an example with an exponentiation module feeding a stochastic module; the correct value of the probability is proportional to the exponential (base 2) of the quantity of the input, hitting 100% at an input of 10.

Our stochastic module becomes:

$$\begin{aligned}
& & \varnothing & \xrightarrow{\text{slow}} & \text{keysmith} \\
& & \text{keysmith} & \xrightarrow{\text{slow}} & \text{key} \\
\forall i: & & \text{key} + e_i & \xrightarrow{\text{slow}} & d_i \\
\forall i: & & d_i + e_i & \xrightarrow{\text{fast}} & 2d_i \\
\forall i \neq j: & & d_i + e_j & \xrightarrow{\text{fast}} & d_i \\
\forall i: & & d_i + \text{keysmith} & \xrightarrow{\text{fast}} & d_i \\
\forall i: & & d_i & \xrightarrow{\text{slow}} & d_i + O_i.
\end{aligned} \tag{8}$$

The lock on each initiating reaction ensures that only a single random choice can be made. Once made, this choice inhibits all other choices both by consuming competing molecules and by destroying subsequent keysmiths. It is interesting to note that this version actually requires *fewer* reactions than our previous version for cases with five or more outcomes.

Figure 7 shows that the error at any given rate separation is more than an order of magnitude *lower* with locking than without; this is before taking into account that the unlocked version actually requires three levels, thus needing two such separations, while the locked version needs only one. Both the locked and unlocked versions were of a stochastic module with three outcomes; 100,000 random trajectories were run for each data point. With fewer requirements on reaction rates, much less error for a given separation in the rates and fewer required reactions in cases with large numbers of outcomes, the locked version is clearly superior.

IV. Discussion

We have implemented the modular designs described here in a tool called BAMBI (*Brian's Automated Modular Biochemical Instantiator*). Given:

- designated input and output types,
- specific quantities (or ranges of values) for the input types,
- target functional dependencies, and
- target probability distribution,

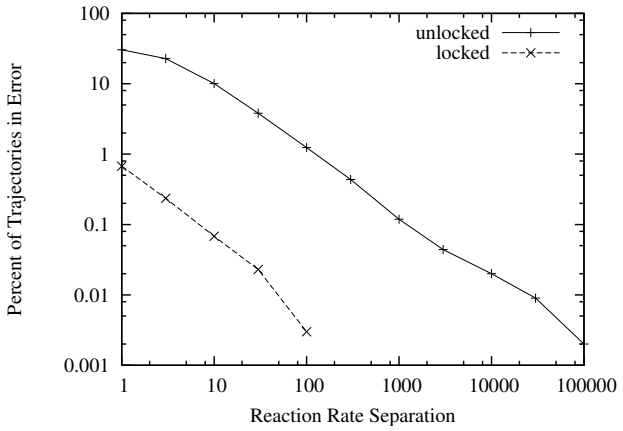

Fig. 7. Accuracy of the Locked vs. Unlocked Stochastic Modules – The percent of trajectories whose initial choice is not reflected in the final state is shown as a function of the separation between "fast" and "slow" rates.

the tool synthesizes a set of biochemical reactions. The targets can be nearly any analytic function or data set. The tool provides detailed measures of accuracy and robustness.

Our work, thus far, has focused on "technology-independent" methods of synthesizing biochemistry: synthesizing a design for a precise, robust, programmable probability distribution on outcomes, for arbitrary types and reactions. In future work, we will extend this methodology to the "technology mapping" phase by implementing designs with specific libraries and toolkits such as with MIT's BioBricks [3].

REFERENCES

[1] A. Arkin, J. Ross, and H. McAdams, "Stochastic Kinetic Analysis of Developmental Pathway Bifurcation in Phage λ-Infected E. Coli Cells," *Genetics*, Vol. 149, No. 1633, 1998.

[2] 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, pp. 423–429, 2004.

[3] BioBricks Parts List, *MIT Registry of Standard Biological Parts*, `http://parts.mit.edu`.

[4] D. Endy and R. Brent, "Modelling Cellular Behaviour," *Nature*, Vol. 409, pp. 391–395, 2001.

[5] D. Endy, "Foundations for Engineering Biology," *Nature*, Vol. 438, pp. 449–453, 2005.

[6] B. Fett, J. Bruck, and M. Riedel, "Synthesizing Stochasticity in Biochemical Systems," *Design Automation Conference*, pp. 640–645, 2007.

[7] M. Gibson and J. Bruck, "Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels," *Journal of Physical Chemistry A*, No. 104, pp. 1876–1889, 2000.

[8] D. Gillespie, "Exact Stochastic Simulation of Coupled Chemical Reactions," *Journal of Physical Chemistry*, Vol. 81, No. 25, pp. 2340–2361, 1977.

[9] C. Goble, S. Pettifer, R. Stevens, and C. Greenhalgh, "Knowledge Integration: *in silico* Experiments in Bioinformatics," in *The Grid: Blueprint for a New Computing Infrastructure*, 2nd Edition, eds. Ian Foster and Carl Kesselman, Morgan Kaufman, 2003.

[10] R. Heinrich and S. Shuster, "The Regulation of Cellular Systems," *Chapman and Hall*, 1996.

[11] A. Hernday, B. Braaten, and D. Low,"The Intricate Workings of a Bacterial Epigenetic Switch," *Advances in Experimental Medicine & Biology*, Vol. 547, No. 83-9, 2004.

[12] I. Herskowitz, "Life Cycle of the Budding Yeast *Saccharomyces cerevisiae*," *Microbiological Reviews*, Vol. 52, No. 4, pp. 536–553, 1988.

[13] E. Libby, T. Perkins, and P. Swain, "Noisy information processing through transcriptional regulation," *Proceedings of the National Academy of Sciences*, Vol. 104, No. 17, pp. 7151–7156, 2007.

[14] L. Lok and R. Brent, "Automatic Generation of Cellular Reaction Networks with Moleculizer 1.0," *Nature Biotechnology*, Vol. 23, No. 1, pp. 131–136, 2005.

[15] C. Myers, "*Asynchronous Circuit Design*," John Wiley and Sons, 2001.

[16] L. Nagel and D. Pederson, "Simulation Program with Integrated Circuit Emphasis," *Midwest Symposium on Circuit Theory*, 1973.

[17] S. Paliwal, P. Iglesias, K. Hilioti, A. Groisman, and A. Levchenko, "MAPK-mediated bimodal Gene Expression and Adaptive Gradient Sensing in Yeast," *Nature*, Vol. 446, Issue 7131, pp. 46–51, 2007.

[18] W. Qian and M. Riedel, "Robust Polynomial Arithmetic with Stochastic Logic," *Design Automation Conference*, 2008.

[19] J. Rabaey, "Design without Borders – A Tribute to the Legacy of A. Richard Newton," *Design Automation Conference*, 2007.

[20] M. Riedel, "The Computer-Aided Synthesis of Stochasticity in Biochemical Systems," *Advances in Synthetic Biology*, 2008.

[21] D. Ro, E. Paradise, M. Ouellet, K. Fisher, K. Newman, J. Ndungu, K. Ho, R. Eachus, T. Ham, M. Chang, S. Withers, Y. Shiba, R. Sarpong, and J. Keasling, "Production of the Antimalarial Drug Precursor Artemisinic Acid in Engineered Yeast," *Nature*, Vol. 440, pp. 940–943, 2006.

[22] M. Sedlak and N. Ho, "Production of Ethanol from Cellulosic Biomass Hydrolysate Using Genetically Engineered Yeast," *Applied Biochemistry & Biotechnology,* Vol. 114, No. 1–3, pp. 403–416, 2004.

[23] H. Smith, C. Hutchinson III, C. Pfannkoch, and J. Venter, "Generating a Synthetic Genome by Whole Genome Assembly: ϕX174 Bacteriophage from Synthetic Oligonucleotides," *Proceedings of the National Academy of Sciences*, Vol. 100, pp. 15440–15445, 2003.

[24] J. Glass, et al., "Essential Genes of a Minimal Bacterium," *Proceedings of the National Academy of Sciences*, Vol. 103, pp. 425–430. 2006.

[25] L. Weinberger, J. Burnett, J. Toettcher, A. Arkin, and D. Schaffer, "Stochastic Gene Expression in a Lentiviral Positive-Feedback Loop: HIV-1 Tat Fluctuations Drive Phenotypic Diversity," *Cell*, Vol. 122, pp. 169–182, 2005.

[26] E. Zielinska, "Chris Voigt: Biology's Toy Maker," *The Scientist*, Vol. 21, No. 9, 2007.