Compiling Program Control Flows into Biochemical Reactions *

De-An Huang[1], Jie-Hong R. Jiang[1,2], Ruei-Yang Huang[1], and Chi-Yun Cheng[2]
[1]Department of Electrical Engineering; [2]Graduate Institute of Electronics Engineering
National Taiwan University, Taipei 10617, Taiwan

ABSTRACT

Computing with biochemical reactions emerges in synthetic biology. With high-level programming languages, a target computation can be intuitively and effectively specified. As control flows form the skeleton of most programs, how to translate them into biochemical reactions is crucial but remains ad hoc. This paper shows a systematic approach to transforming control flows into robust molecular reactions. Case studies demonstrate its usefulness.

1. INTRODUCTION

The advancement of *systems biology* [2] reveals that biochemical reactions in living organisms intertwine and conduct complex "computation" resulting in intelligent reactive behaviors. The curious exercise to build artificial systems with biochemical components gives birth to the field of *synthetic biology* [1, 10]. A well-known example, among myriad others, is the synthetic oscillator [4]. Engineering computation using biochemical reactions not only satisfies human curiosity, but also sharpens our understanding about nature's design principles of biological circuits.

With proper quantitative interpretation on molecular concentrations, a set of biochemical reactions can be thought of as a computing system. Given initial molecular concentrations as input, the reactions (can be described by differential equations) evolve the concentrations in the configuration space toward some target concentrations as output. Under such quantitative interpretation, biochemical systems are similar to hardware and software systems. Therefore the design methodology of hardware and software systems can be brought to biochemical systems. Implementation substrates, such as the DNA strand displacement technique [17], have been demonstrated for molecular computing.

To build complex systems, hardware/software design with high-level programming languages has become a standard approach due to their supported intuitive and effective features. To synthesize a program down to an actual physical realization, several transformation steps often have to be conducted at various abstraction levels. Compiling a program to molecular realization of biochemical systems shares strong similarity to compiling a program to silicon realization of integrated circuits. As control flows form the skeleton of most if not all programs, one key step in the compilation is to transform the control flows.

Synthesizing molecular reactions has been pursued, e.g., in [5, 6] for some arithmetic operations, [9] for digital signal processing, [15, 16] for writing and compiling code into biochemistry, etc. Nevertheless the involved control flows are relatively simple, e.g., without composition of looping and branching statements. Also the compilation of program control flows may seem somewhat ad hoc. A general principle and methodology remain lacking to handle complex control flows.

This paper proposes a systematic framework converting control flow statements into biochemical reactions. Several techniques, including the dimerized absence indicator, reaction buffer, restoration reaction, error-tolerant precision control, etc., are devised to enhance reaction robustness and practicality. To show concrete applications, we perform case studies on arithmetic division and on greatest common divisor computation, which were not shown before. Simulation results confirmed the usefulness and robustness of the proposed methodology. In particular, our synthesized reactions work robustly at both the mesoscopic scale (involving tens to thousands of individual molecules) under discrete stochastic simulation [7] and the macroscopic scale under continuous deterministic simulation, in contrast to the mesoscopic restriction of [16].

The rest of this paper is organized as follows. Preliminaries are given in Section 2. Methods of regulating reactions are discussed Section 3. Section 4 presents the conversion of control-flow statements to reactions. Section 5 performs evaluation based on case studies. Section 6 compares our results with prior work. Finally, Section 7 concludes this paper.

2. PRELIMINARIES

2.1 Chemical Kinetics

To simplify discussion, this paper adopts the *classical chemical kinetic (CCK) model* [12] of biochemical reactions (although our formulation works under stochastic simulation as well). We assume that the molecules involved in reactions are of large quantities and thus satisfying the following basic assumptions of the CCK model. First, molecules are equally distributed in space and their spatial non-uniformity effects are negligible. Second, the reactions happen continuously and deterministically. Under these assumptions, the dynamic behavior of a biochemical systems can be characterized with ordinary differential equations (ODEs). From an application viewpoint, it can be exploited as a resource for computing in term of molecular concentrations, which correspond to numbers and logic values.

Consider the following biochemical reaction

$$\alpha_1 X_1 + \cdots + \alpha_m X_m \xrightarrow{k} \beta_1 Y_1 + \cdots + \beta_n Y_n,$$

where coefficients α_i's and β_j's specify the stoichiometric amounts, molecules X_i's and Y_j's are the *reactants* and *prod-*

*This work was supported in part by the National Science Council under grants NSC 99-2221-E-002-214-MY3, 99-2923-E-002-005-MY3, and 100-2923-E-002-008.

IEEE/ACM International Conference on Computer-Aided Design (ICCAD) 2012, November 5-8, 2012, San Jose, California, USA

ucts, respectively, and k is the *rate constant*. Let $[A]$ denote the concentration of molecule A. The dynamics of these molecules can be described by

$$-\frac{1}{\alpha_1}\frac{d[X_1]}{dt} = \cdots = -\frac{1}{\alpha_m}\frac{d[X_m]}{dt} = \frac{1}{\beta_1}\frac{d[Y_1]}{dt} = \cdots = \frac{1}{\beta_n}\frac{d[Y_n]}{dt} = k[X_1]^{\alpha_1}\cdots[X_m]^{\alpha_m},$$

where the last term is the *reaction rate*, proportional to the rate constant and the reactant concentrations.

For a set of reactions, the slowest reaction that determines the speed of the whole process is called the *rate-limiting step*. In a reaction, the reactant totally consumed when the reaction is complete is called the *limiting reactant*.

2.2 Reaction Rate and Execution Precedence

The rate, or speed, of a biochemical reaction is determined by its reactant concentrations and rate constant. Given two coexisting reactions, the more the difference is between their reaction rates, the less their executions are overlapping. With a substantial rate difference between two reactions, one can approximately consider the slow reaction executes only after the fast reaction has finished. However since the molecules coexist and the reactions are concurrent, they are not perfectly well interleaved and there always exists some extent of "leakage" that the slow reaction executes before fast reaction finishes.

When a high-level program is transformed into a set of reactions, different reaction rates can be exploited to prioritize the computations of the program. To reduce the number of required reaction rates, concepts such as module locking [6] and absence indicators [9] were proposed. Essentially only two, fast and slow, reaction rates are needed. This paper adopts a similar approach with a robustness improvement to be discussed in Section 3.2. In the sequel, we shall denote fast and slow reaction rates with r_f and r_s, respectively.

2.3 Boolean Abstraction

For molecule A, the Boolean interpretation of its concentration with respect to some non-negative constant θ is a function defined as

$$f_\theta(A) = \begin{cases} 1, & \text{if } [A] \geq \theta, \text{ and} \\ 0, & \text{otherwise,} \end{cases}$$

where θ is a reaction-dependent threshold. With this abstraction, we interpret molecule A is of high and low concentrations if $f_\theta(A) = 1$ and $f_\theta(A) = 0$, respectively. When molecule A participates in a reaction, $f_\theta(A)$ effectively indicates whether A is active or inactive in the reaction. In the sequel, for simplicity we assume a uniform threshold θ for all reactions, and abbreviate $f_\theta(A)$ as A_θ. Our formalism however can be easily extended to a general setting with different θ's.

The propositional conditions on multiple molecule concentrations can be expressed by a Boolean formula using logic connectives: conjunction $\wedge$, disjunction $\vee$, and negation $\neg$. For example, formula $(A_\theta \wedge B_\theta) \vee \neg C_\theta$ expresses the condition that A and B are of high concentrations, or C is of low concentration.

3. REACTION REGULATION

In information processing, computation must be performed in a proper order such that data dependencies are maintained to ensure operational correctness. How to arrange reactions in a desired temporal order is an important issue in molecular computation. This section introduces some key elements for reaction regulation.

3.1 Reaction Pre/Postconditions

In a programming language, the *precondition* and *postcondition* of a line of code are the state/variable conditions before and after executing this line of code. Similarly, when translated to biochemical reactions, the *precondition* and *postcondition* of a reaction are the respective molecular concentration conditions before and after the exhaustive firing of this reaction.

In compiling a program into biochemical reactions, often we need to regulate a reaction based on the presence or absence of some molecule. Without loss of generality, consider the simple reaction $X \to Y$, which is to be activated by the presence of molecule Z. Then this reaction can be expressed, among other possibilities, as $(X + Z \to Y + Z)$, where Z serves as a catalyst or an enzyme. For this reaction to be active, Z_θ is a precondition, and $\neg X_\theta$ and Y_θ are postconditions. For brevity, in the sequel the above reaction is denoted as

$$X \xrightarrow{Z_\theta} Y$$

where the catalyst precondition Z_θ is shown above the arrow and should be distinguished from the rate constants r_f and r_s.

A precondition can involve the presence and/or absence of multiple molecules and thus can be a complex Boolean expression. For example,

$$X \xrightarrow{A_\theta \wedge B_\theta} Y$$

corresponds to

$$X + A + B \to Y + A + B$$

and

$$X \xrightarrow{A_\theta \vee B_\theta} Y$$

corresponds to

$$X + A \to Y + A$$
$$X + B \to Y + B$$

It should be noted however that, by the CCK model, reaction activation can only be possible by the presence, instead of absence, of some molecule. Therefore to activate a reaction by the absence of some molecule, we need to introduce the so-called *absence indicator* [9].

3.2 Absence Indicator

Let A' be the complementary molecule of A such that the presences of molecules A and A' are mutually exclusive. In [9, 16], A' is called the *absence indicator* of A with the following basic realization:

$$\emptyset \xrightarrow{r_s} A'$$
$$A + A' \xrightarrow{r_f} A$$
$$(2A' \xrightarrow{r_f} A')$$

That is, the environment constantly and slowly generates A', and A fast degrades A'. So A' exists (to some small extent due to the third reaction) only when A is absent.

The validity of these reactions for absence indication relies on the mesoscale assumption. At the mesoscopic level, the absence of some type of molecules can be clearly defined as absolute zero of a molecular count. So the number of molecule A' can be effectively suppressed to zero when molecule A is present. When it comes to the macroscopic level, however, this notion of absence no longer holds. By the mass action kinetics, at equilibrium

$$r_s = r_f \cdot [A] \cdot [A']$$

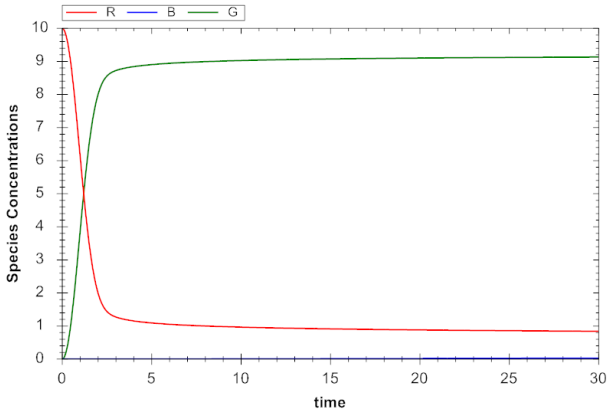

Figure 1: RGB reactions regulated by prior absence indicator (by ODE simulation with SBW simulator)

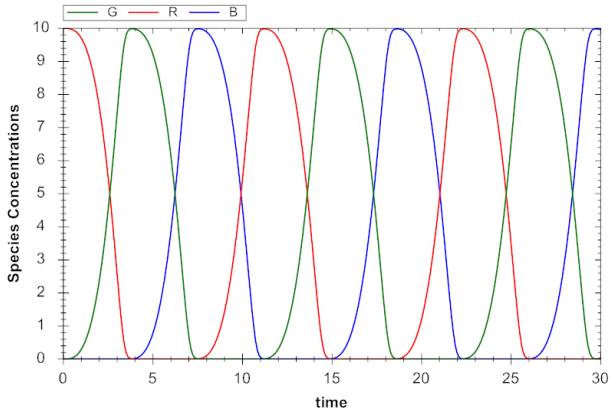

Figure 2: RGB reactions regulated by dimerized absence indicator (by ODE simulation with SBW simulator)

As the equilibrium concentration $[A']$ is proportional to r_s/r_f ($[A]$ can be treated as a constant at equilibrium), it cannot be exactly zero. (Note that the third reaction is neglected for simplicity without affecting the conclusion.) This "leakage" of A' strongly degrades the regulatory robustness of absence indicator in bulk reactions.

To overcome this shortcoming, we propose a robust *dimerized absence indicator* A^* of A with the following reactions:

$$\emptyset \xrightarrow{r_s} A'$$
$$A + A' \xrightarrow{r_f} A$$
$$2A' \underset{r_f}{\overset{r_s}{\rightleftharpoons}} A^*$$

At equilibrium, the concentration of A^* satisfies

$$r_s \cdot [A']^2 = r_f \cdot [A^*]$$

By substituting $[A']$ by its equilibrium concentration, $[A^*]$ is proportional to $(r_s/r_f)^3$. Compared to the prior absence indicator A', the new one A^* is further suppressed. This suppression makes the leakage of A^* negligible under the presence of A. Thereby A^* is much more robust than A' and is suitable even under the mass action kinetics.

Moreover, the dimerized absence indicator is advantageous in two aspects. First, its regulated reaction is flexible in either of fast and slow rates (preferably fast as implicitly assumed in the sequel). Second, its regulated reaction has rate almost independent of its reactant concentrations due to the new rate-limiting step. Therefore, undesirable interferences among reactions are reduced.

To assess the robustness of dimerized absence indicator, consider the following reaction cycle

$$R \xrightarrow{\neg B_\theta} G$$
$$G \xrightarrow{\neg R_\theta} B$$
$$B \xrightarrow{\neg G_\theta} R$$

where $\neg A_\theta$ stand for ${A^*}_\theta$ as a convention in the sequel unless otherwise stated. Let $R_\theta = 1$, $G_\theta = 0$, $B_\theta = 0$ initially. The reaction cycle is expected to behave as an oscillator. The simulation results by the SBW simulator [14] for cases using the prior ($\neg A_\theta = {A'}_\theta$) and dimerized ($\neg A_\theta = {A^*}_\theta$) absence indicators are shown in Figures 1 and 2, respectively. The ratios of rate constant r_f to r_s used in the simulations of Figures 1 and 2 are $10^4 : 1$ and $10^2 : 1$, respectively. (The x and y axes of the plots in this paper are of relative units by the default setting of the SBW simulator.) As seen from Figure 1,

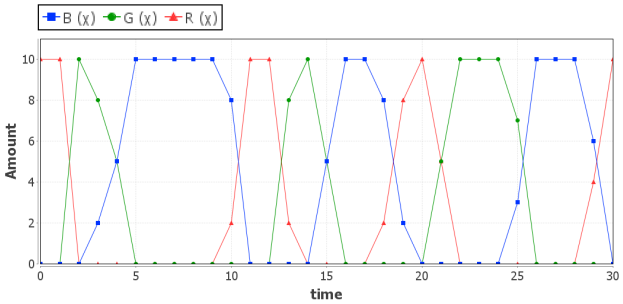

Figure 3: RGB reactions regulated by dimerized absence indicator (by stochastic simulation with iBioSim simulator)

the reactions regulated by the prior absence indicator fails to oscillate (G already transfers to B before the absence of R due to the leakage of non-negligible ${R'}_\theta$). In contrast, the dimerized absence indicator robustly sustains the oscillation due to its effective leakage suppression (in the presence of R the transfer rate from G to B is negligible compared to that from R to G).

On the other hand, the proposed dimerized absence indicator is applicable at the mesoscopic level, similar to prior absence indicator. Figure 3 shows the simulation result of iBioSim [8] using Gillespie's method [7].

3.3 Reaction Buffer

Even with the dimerized absence indicator, it is not possible to achieve instant reaction cycle

$$A \to B$$
$$B \xrightarrow{\neg A_\theta} A$$

since the product of the second reaction violates its precondition. To solve this problem, a buffer is needed to impose sufficient reaction delay. The reaction cycle

$$A \to B$$
$$B \xrightarrow{\neg A_\theta} C$$
$$C \xrightarrow{\neg B_\theta} A$$

with length three becomes achievable. Such molecule C is called a *buffer molecule*. In fact, buffering is useful not only in reaction cycles but also in other complex reactions to avoid violation of reaction preconditions. As will be seen, this technique is useful in the case studies of Section 5.

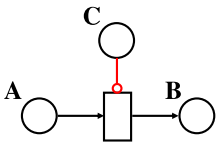

Figure 4: A Petri net example

3.4 Reaction Decomposition

In the previous discussion, the number of reactants in a reaction is unconstrained. Although it is not a serious problem for macroscale reactions, at the mesoscopic level a reaction with more than two reactants can be rare. For practicality concern, it may be necessary to decompose a reaction with many reactants to a set of reactions with fewer reactants. For example, as was shown in [17, 16], a trimolecular reaction

$$A + B + C \to D$$

can be decomposed into

$$A + B \underset{r_f}{\overset{r_f}{\rightleftharpoons}} E$$
$$E + C \to D$$

Generally, a reaction with n, say $2k+1$, reactants

$$A_1 + \cdots + A_{2k+1} \to B$$

can be converted into $\lfloor n/2 \rfloor$ bimolecular reactions

$$A_1 + A_2 \rightleftharpoons B_{1,2}$$
$$\vdots$$
$$A_{2k-1} + A_{2k} \rightleftharpoons B_{2k-1,2k}$$

and a reaction

$$B_{1,2} + \cdots + B_{2k-1,2k} + A_{2k+1} \to B$$

with $\lfloor n/2 \rfloor + (n \mod 2)$ reactants. Applying decomposition iteratively transforms a reaction with an arbitrary number of reactants into a set of reactions each with no more than two reactants.

3.5 Petri Net Visualization

A system of biochemical reactions can be intuitively visualized with a *Petri net* [11], which is a bipartite graph consisting of two types of nodes, i.e., *circles* (or *places*) and *boxes* (or *transitions*), and directed and weighted edges connecting between circles and boxes. In particular, a circle represents a molecule and a box represents a reaction. The fanin and fanout circles of a box correspond to the reactants and products, respectively, of the corresponding reaction. The amount of a molecule is signified by the amount of *tokens* in the corresponding circle. The reaction of a box B can fire if the molecule of every fanin circle C of B has a sufficient amount specified by the weight on the edge connecting from C to B. In our context, we augment the Petri net with *inhibition arcs* [3]. An inhibition arc, denoted as a directed edge ended with a bubble, connects from a circle to a box and signifies that the corresponding molecule inhibits the corresponding reaction. For example, the Petri net of Figure 4 represents the reaction

$$A \xrightarrow{\neg C_\theta} B$$

Note that for brevity we shall omit representing tokens in circles and weights on edges in a Petri net.

4. CONTROL FLOW COMPILATION

By treating the reactant concentrations of a biochemical reaction as inputs and the product concentrations as outputs, the reaction itself can be seen as an atomic step in computation. By orchestrating a set of reactions in a well-organized flow, complex computation can be achieved. This section shows how to translate the program control flow into biochemical reactions (independent of the choices on absence indicator realizations).

4.1 Linear Flow

A system is often specified by a high-level program in a well-ordered sequential manner whereas biochemical reactions are intrinsically concurrent. One of the most basic elements in control flow compilation is to transform a simple control flow (consisting of only branchless and loop-free statements) into biochemical reactions executed step by step in a linear order. For example, consider the reactions:

$$A \to B$$
$$C \to D$$
$$E \to F$$

Without regulation, these reactions are concurrent. When their executions are intended to be in a sequential manner, preconditions can be imposed as follows.

Main Reactions		**Preconditions**
01	$A \to B$	
02	$C \to D$	$\neg A_\theta$
03	$E \to F$	$\neg C_\theta$

A Petri net representation of these reactions is shown in Figure 5 (a). Essentially the exhaustion condition of some reactant of a reaction can be used as a precondition for its subsequent reaction. In a general setting, where a reaction involves multiple reactants and products, the catalyst precondition can be built as a complex Boolean expression to ensure correct activation. Again, due to the $(r_s/r_f)^3$ suppression power of the dimerized absence indicator, a reaction can be fired only when its preceding reaction is almost exhausted. It robustly regulates the linear execution order of these reactions.

4.2 Branching Statement

We examine mainly the `if-else` construct whereas a similar methodology is applicable to other branching constructs, such as `switch`, as well. Without loss of generality, consider the following example of ordered conditional reactions.

Main Reactions		**Preconditions**
	$\cdots$	
01	$Q \to R$	
02	**if** $P_1(A, B)$	$\neg Q_\theta$
03	$S_1 \to T_1$	$Post(P_1)$
	$\cdots$	
04	$S_i \to T_i$	
05	$H \to I$	$\neg S_{i\theta}$
06	**else if** $P_2(A, B)$	$\neg Q_\theta$
07	$U_1 \to V_1$	$Post(P_2)$
	$\cdots$	
08	$U_j \to V_j$	
09	$H \to I$	$\neg U_{j\theta}$
10	**else**	$\neg Q_\theta$
11	$W_1 \to X_1$	$Post(\neg P_1 \wedge \neg P_2)$
	$\cdots$	
12	$W_k \to X_k$	
13	$H \to I$	$\neg W_{k\theta}$
14	$Y \to Z$	$\neg H_\theta$
	$\cdots$	

A Petri net representation of these reactions is shown in Figure 5 (b), where the reactions of lines 1 and 14 are omitted. In this example, P_1 and P_2 are *predicates*, which expresses the entering conditions of the `if` block and `else if`

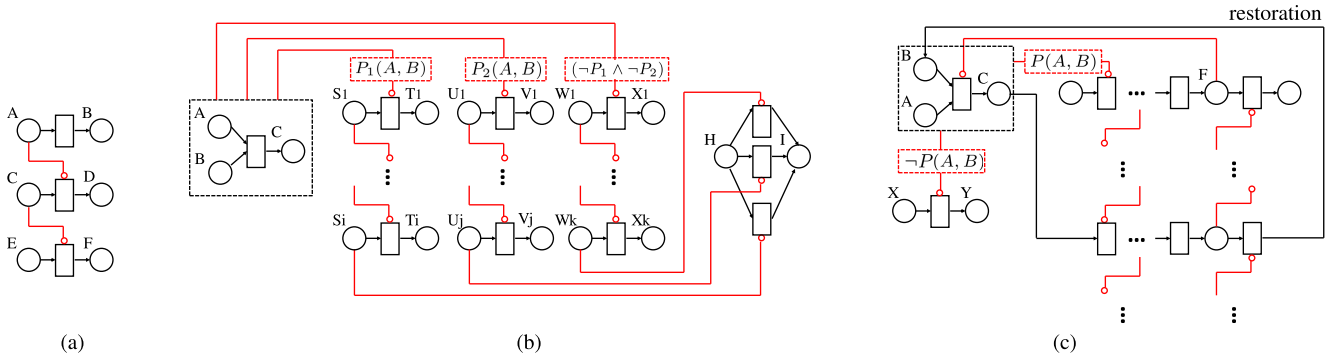

Figure 5: Petri net visualization of (a) linear flow, (b) branching statement, and (c) looping statement

block, respectively. Their truths depend on their arguments' molecular concentrations. In the sequel, we shall assume that the predicates express disjoint conditions, that is, $P_1 \wedge P_2$ equals false in this example. This assumption can always be maintained because, even if the original predicates are non-disjoint, they can be made disjoint by prioritizing them. In this example, P_2 can be modified as $P_2 \wedge \neg P_1$.

For a predicate P, we associate it with two forms of definitions: a *descriptive definition* and an *operational definition*. The former defines P with a logical statement, and the latter defines P with a set of biochemical reactions and a postcondition of these reactions. For example, $P(A,B) = A > B$ is a descriptive definition, and $P(A,B) = A_\theta \wedge \neg B_\theta$, the postcondition with respect to reaction $A + B \to C$, is an operational definition. Note that, given a descriptive definition for a predicate, its equivalent operational definition is not unique. In the sequel, we call the reactions (respectively postcondition) in the operational definition of P simply as the reactions (respectively postcondition) of P. In the sequel, we denote the postcondition of P as $Post(P)$.

The postcondition of a predicate and other stronger conditions (that imply this postcondition) can be exploited to selectively enable its subsequent reactions. When the predicates of an `if-else` statement are mutually contradicting, disjoint postconditions can be derived to uniquely activate the starting reaction(s) of the desired code-block. In the example, the `if` code-block can be enabled by $Post(P_1) = A_\theta \wedge \neg B_\theta$, `else if` block by $Post(P_2) = \neg A_\theta \wedge B_\theta$, and `else` block by $Post(\neg P_1 \wedge \neg P_2) = \neg A_\theta \wedge \neg B_\theta$. Note that the three predicates P_1, P_2, $(\neg P_1 \wedge \neg P_2)$, share the same reaction. Moreover, their preconditions are the same, i.e., $\neg Q_\theta$.

On the other hand, to ensure the first reaction right after an `if-else` statement, i.e., $Y \to Z$ in the example, is properly activated when leaving the statement, we insert a control reaction $H \to I$ at the end of every code-block of the `if-else` statement, and assume H is of high concentration initially. So this control reaction is activated by $\neg S_{i\theta} \vee \neg U_{j\theta} \vee \neg W_{k\theta}$. Thereby $\neg H_\theta$ can be used as the precondition for $Y \to Z$. As shown in the example, the corresponding precondition of a main reaction step is shown on the right-hand side.

4.3 Looping Statement

We study the `while` construct whereas a similar methodology is applicable to other looping constructs, such as `for`, as well. Without loss of generality, consider the following ordered looping reactions.

	Main Reactions	**Preconditions**
01	$Q \to R$	
02	**while** $P(A,B)$	$\neg Q_\theta \cdot \neg F_\theta$
03	$\dots \to F$	$Post(P)$
	$\cdots$	
04	$F \to \dots$	
05	$X \to Y$	$Post(\neg P)$

A Petri net representation of these reactions is shown in Figure 5 (c), where the reaction of line 1 is omitted. The `while` statement is similar to the `if-else` statement in that the former can be considered as an `if-else` construct with repeated execution on the `if` code-block until the `if`-predicate becomes false. So leaving the `while` loop is the same as switching to the `else` block. We simply use the negated postcondition of the predicate of `while` as the precondition of the first reaction after the `while` loop, i.e., $X \to Y$ in this example.

On the other hand, entering the `while` statement is slightly more complicated than entering the `if-else` statement. The entering condition is determined by two reactions: the reaction right before the `while` code-block, i.e., $Q \to R$ in our example, and the last reaction in the `while` block. To achieve proper activation of the `while` loop, we insert two reactions $\cdots \to F$ and $F \to \cdots$ in the beginning and at the end, respectively, in the `while` loop, where F is a new molecule not used elsewhere with a low initial concentration. We then use $\neg Q_\theta \wedge \neg F_\theta$ as the precondition for the reactions of predicate P. Reaction $\cdots \to F$ ensures the newly introduced molecule F is produced to halt the predicate reactions during the entire execution of the `while` loop, whereas $F \to \cdots$ ensures F is consumed at the end of the `while` loop to reactivate the predicate reactions. Moreover, another loop invariant needs to be maintained is $\neg Q_\theta$ to prevent the predicate reactions from being unintentionally halted.

Another issue about the `while` loop is the restoration of molecules. Sometimes the concentrations of some molecules must be restored. Suppose, for example, the reactions of predicate $P(A,B)$ change the concentrations of A, B, and other relevant molecules. Before next iteration the predicate needs to be checked again, the concentrations of these molecules must be restored to their correct amounts.

Let C be the molecule to be restored. We introduce a restoration mechanism with the following reaction cycle.

$$\begin{aligned} C + \cdots &\to D + \cdots \\ D &\to E \\ E &\to C \end{aligned}$$

Let D and E be two new molecules with initial zero amounts. So the amount of C used in reactions is accumulated in D. Restoration of the used amount of C can be done by enabling the second and then third reactions, and thus passing

the amount of D to E and back to C. Noted that not every reaction involving C should have D as a product. For the cases we intend to reduce the amount of C, we can neglect D as a product.

4.4 Compilation Strategy

The above constructs can be used as templates in sketching the control flow of a program. The following strategies show the general steps for program compilation.

1. Identify linear, looping, and branching statements, and based on their corresponding templates create control flow reactions.
2. Resolve violation of precondition and postcondition of reactions, and introduce reaction buffers if necessary.
3. Decompose reactions for practical realization.
4. Optimize and simplify reactions.

Although our program compilation currently remains a manual process, we believe that the proposed methodology is systematic and takes a step forward to automating the design of molecular computing.

5. CASE STUDY

We performed two case studies on division and greatest common divisor computation, whose realizations with biochemical reactions had not been shown before. The reactions were written in the Systems Biology Markup Language (SBML) [13] and simulated with the SBW simulator [14]. The dimerized absence indicator was used by default with the ratio of rate constant r_f to r_s setting as $1000:1$. Moreover, while our synthesized reactions work under both ODE simulation and Gillespie's simulation, only ODE simulation results are shown. For brevity, the reaction codes presented below are without reaction decomposition discussed in Section 3.4 even though decomposition is also performed for simulation.

5.1 Division

A pseudo-code of division over integers is given below, where A and B are the input dividend and divisor, respectively, and Q and R are the output quotient and remainder, respectively.

```
Division(A, B)
  begin
  01   while A ≥ B
  02     A := A − B
  03     Q := Q + 1
  04   R := A
  end
```

The pseudo-code can be translated into the following *reaction-code*, where molecule C is initially of a unit amount.

	Main Reactions	**Preconditions**
01	**while** $[A] \geq [B]$	
02	$(A + B \rightarrow D)$	$\neg G_\theta$
03	$C \rightarrow Q + E$	$A_\theta \wedge \neg B_\theta$
04	$D \rightarrow F$	$\neg C_\theta$
05	$E \rightarrow G$	$\neg D_\theta$
06	$F \rightarrow B$	$\neg E_\theta$
07	$G \rightarrow C$	$\neg F_\theta$
08	$D \rightarrow R$	$\neg A_\theta$

A Petri net representation of these reactions is shown in Figure 6. The predicate reaction, in line 2 (reaction-code), is parenthesized to distinguish it from other reactions. The same reaction is also usable for the reaction of code $A := A - B$ (in line 2, pseudo-code). Lines 2, 4, 6 in the reaction-code form a reaction cycle, and line 6 is the restoration step

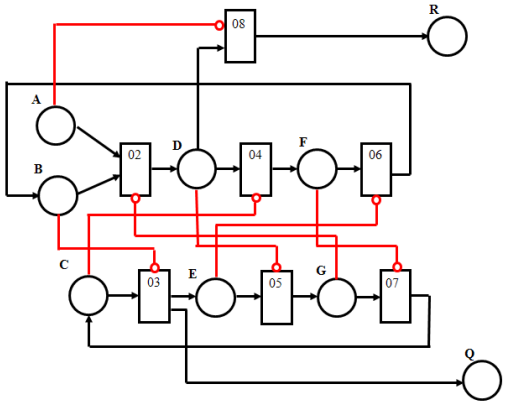

Figure 6: Petri net visualization of division computation

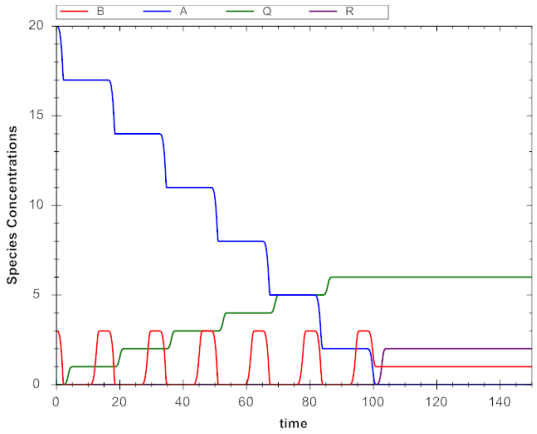

Figure 7: Computation of *Division*$(20, 3)$ without reaction decomposition

of B, which is required since the computation $A := A - B$ should not change the amount of B. Lines 3, 5, 7 form a reaction cycle to restore C. This cycle is triggered to add 1 to quotient Q, and the amount of C should remain the same after an iteration so restoration is required.

Figures 7 and 8 show the SBW simulation results of computing $20/3$ by the above reactions without and with reaction decomposition, respectively. The waveforms confirm their correctness despite the irregular curve in Figure 8 due to the reversible reactions discussed in Section 3.4.

5.2 Greatest Common Divisor Computation

A pseudo-code for the greatest common divisor (GCD) computation is given below, where A and B are the input integers, and GCD is the output.

```
GreatestCommonDivisor(A, B)
  begin
  01   while A ≠ B
  02     if A > B
  03       A := A − B
  04     else if B > A
  05       swap(A, B)
  06   GCD := A
  end
```

With the proposed constructs, the pseudo-code translates to the reaction-code below.

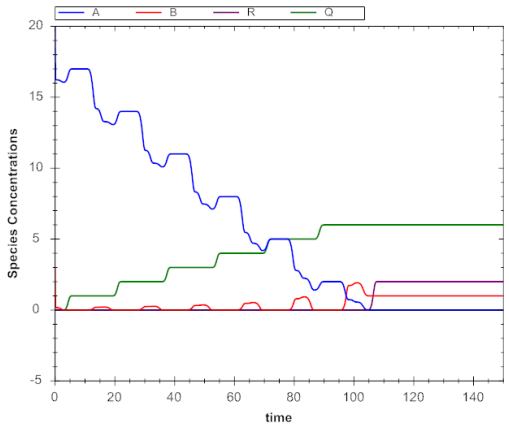

Figure 8: Computation of *Division*$(20, 3)$ with reaction decomposition

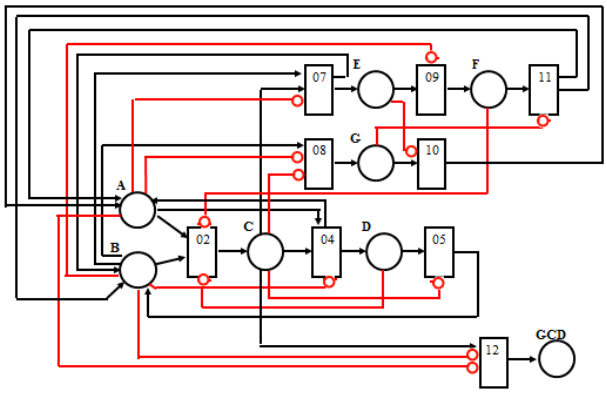

Figure 9: Petri net visualization of GCD computation

Main Reactions		**Preconditions**
01	**while** $[A] \neq [B]$	
02	$(A + B \rightarrow C)$	$\neg D_\theta \wedge \neg F_\theta$
03	**if** $[A] > [B]$	
04	$C \rightarrow D$	$A_\theta \wedge \neg B_\theta$
05	$D \rightarrow B$	$\neg C_\theta$
06	**else if** $[B] > [A]$	
07	$C \rightarrow E$	$\neg A_\theta \wedge B_\theta$
08	$B \rightarrow G$	$\neg C_\theta \wedge \neg A_\theta$
09	$E \rightarrow F$	$\neg B_\theta$
10	$G \rightarrow A$	$\neg E_\theta$
11	$F \rightarrow A + B$	$\neg G_\theta$
12	$C \rightarrow GCD$	$\neg A_\theta \wedge \neg B_\theta$

A Petri net representation of these reactions is shown in Figure 9. The predicate reactions for the `while` and `if-else` statements can be realized with the same reaction $A+B \rightarrow C$ (line2, reaction-code). In addition, the same reaction works for $A := A - B$ (line 3, pseudo-code). If $[A] > [B]$ is true, lines 2, 4, 5 (reaction-code) form a reaction cycle restoring B. If $[B] > [A]$ is true, lines 7, 9, 11 (reaction-code) restore A and B with amount $[A]$. Lines 8 and 10 restore A with amount $[B] - [A]$. These two reaction paths perform the swap of A and B. In the reaction cycle formed by lines 7, 9, 11, molecule F serves as a buffer, which is needed because the reactant of reaction cycle involves B, which is one of the molecules restored in line 11. The operations of line 8 and line 11 must be interleaved such that the production of B in line 11 does not affect the precondition of the reactions from line 8 to 11. After $[A]$ and $[B]$ both becomes zero, the final result remains in C. So the last step, when $[A] = [B]$, we pass the amount of C to GCD.

Figures 10 and 11 show the simulation results for the computation of GCD(30,12) without and with reaction decomposition, respectively. The result is correct except that the last reaction $C \rightarrow GCD$ cannot be triggered. The cause is that the amount of molecules are assumed to be *exact* multiples of a predefined unit amount. In reality, the number of molecules cannot be exactly controlled. So $[A]$ and $[B]$ can hardly equal. Even though the rounding error can be ignored, it can still cause a severe problem when used as a precondition.

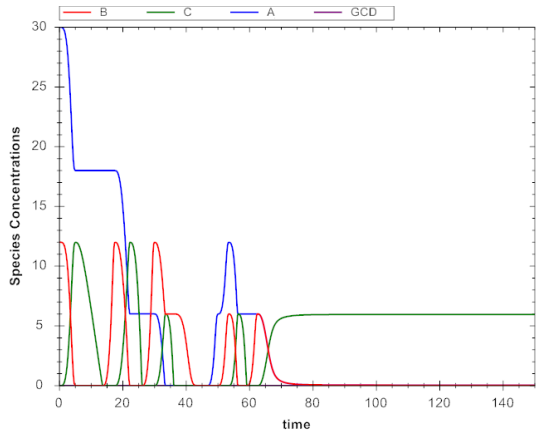

Figure 10: Computation of *GCD*$(30, 12)$ without reaction decomposition

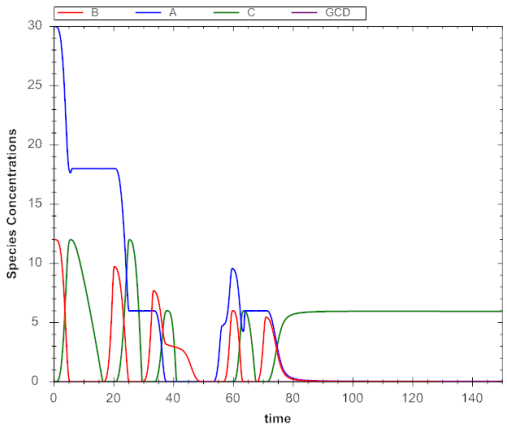

Figure 11: Computation of *GCD*$(30, 12)$ with reaction decomposition

This problem can be solved with the following modified pseudo-code, where the amount of molecule Z sets the precision for error tolerance. In the experiment, Z is of concentration 0.1 units.

```
GreatestCommonDivisor_err_toler(A, B, Z)
  begin
01    while |A − B| > Z
02      if A > B + Z
03        A := A − B
04      else if B > A + Z
05        swap(A, B)
06    GCD := A
  end
```

The program translates to the reaction-code below. The reactions added are lines 3, 4, 7 and 13. The reaction in line 3 (respectively 4) further tests whether $[A]-[B] > [Z]$ (respectively $[B] - [A] > [Z]$), and, if it holds, line 7 (respectively line 13) restores the amounts of Z and A (respectively Z and B). A buffer H is added to the reaction cycle formed by lines 6, 8, and 9 to avoid interference among restoring reactions.

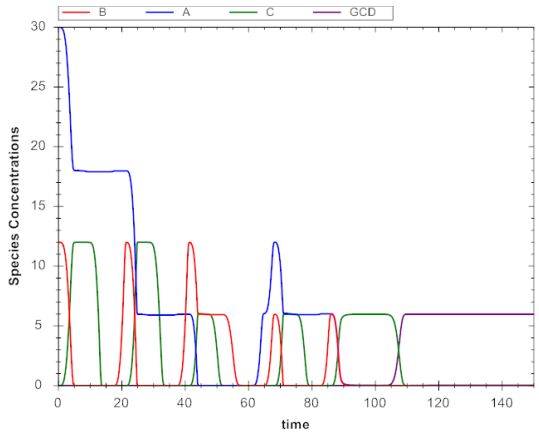

Figure 12: Error-tolerant computation of GCD(30, 12) without reaction decomposition

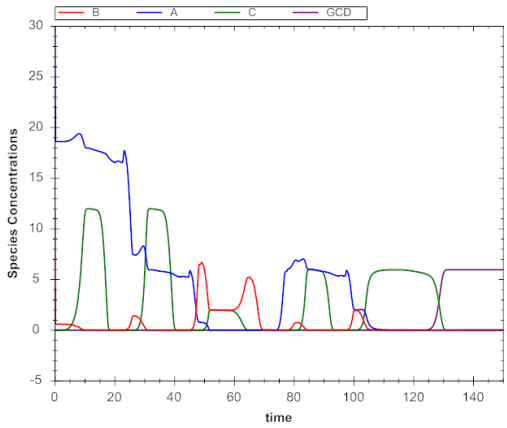

Figure 13: Error-tolerant computation of GCD(30, 12) with reaction decomposition

	Main Reactions	**Preconditions**
01	**while** $\|[A]-[B]\| > [Z]$	
02	$(A + B \to C)$	$\neg H_\theta \wedge \neg F_\theta$
03	$(A + Z \to X)$	$\neg H_\theta \wedge \neg F_\theta \wedge \neg B_\theta$
04	$(B + Z \to Y)$	$\neg H_\theta \wedge \neg F_\theta \wedge \neg A_\theta$
05	**if** $[A] > [B] + [Z]$	
06	$C \to D$	$A_\theta \wedge \neg B_\theta \wedge \neg Z_\theta$
07	$X \to A + Z$	$\neg C_\theta \wedge \neg B_\theta$
08	$D \to H$	$\neg C_\theta$
09	$H \to B$	$\neg D_\theta$
10	**else if** $[B] > [A] + [Z]$	
11	$C \to E$	$\neg A_\theta \wedge B_\theta \wedge \neg Z_\theta$
12	$B \to G$	$\neg C_\theta \wedge \neg A_\theta$
13	$Y \to B + Z$	$\neg C_\theta \wedge \neg A_\theta$
14	$E \to F$	$\neg B_\theta$
15	$G \to A$	$\neg E_\theta$
16	$F \to A + B$	$\neg G_\theta$
17	$C \to GCD$	$\neg A_\theta \wedge \neg B_\theta$

Figures 12 and 13 show the SBW simulation results without and with reaction decomposition, respectively. In both cases the previous precision problem is resolved. Although the waveform of Figure 13 is not as perfect as that of Figure 12, the computation remains correct.

6. RELATED WORK

Among related prior efforts, [16] is the closest to ours. The authors provide examples of arithmetic computation and transform them into biochemical reactions. However a systematic methodology remains missing. Moreover, prior transformation heavily relies on modularized reactions. Our framework in contrast works for non-modularized reactions as well, and can be more flexible thus achieving better optimality.

When molecular quantities are concerned, prior work assumes reactions are in small quantities. Hence discrete stochastic simulation is performed. The constructed reactions do not work under continuous deterministic simulation however. In contrast, our framework assumes more conservative realization to work for both discrete and continuous simulations. Our empirical experience suggests that discrete simulation in our considered computation tends to make more optimistic prediction than continuous simulation.

Compared to the prior absence indicator, our dimerized absence indicator works much more robustly under both continuous and discrete simulations. Other new techniques, such as reaction buffer insertion and a precision control mechanism for error tolerance, are introduced to enhance the reliability of molecular reactions. Also our restoration mechanism, though conceptually similar to the *copier* construct [16], offers greater flexibility for reaction optimization.

To overcome the leakage problem of absence indicator, prior work [9] proposed a positive feedback mechanism to boost the transfers of molecules. The transfers are accelerated and the reaction rates rely lesser on the concentration of absence indicator thus alleviate the impact of leakage. However it costs extra reactions for each transfer of molecules.

7. CONCLUSION AND FUTURE WORK

We have presented a systematic design principle in the compilation of program control flows into biochemical reactions. Robust design techniques, including the dimerized absence indicator, reaction buffer insertion, and a parameterized precision control mechanism, were proposed. Case studies on division and GCD computation have confirmed the usefulness and robustness of the proposed methodology. For future work, the compilation process and reaction optimization remain to be fully automated.

8. REFERENCES

[1] E. Andrianantoandro, S. Basu, D. Karig, and R. Weiss. Synthetic biology: New engineering rules for an emerging discipline. *Molecular Systems Biology*, 2006.

[2] U. Alon. *An Introduction to Systems Biology: Design Principles of Biological Circuits*. Champman & Hall/CRC, 2006.

[3] C. Chaouiya. Petri net modelling of biological networks. *Briefings in Bioinformatics*, 8(4): 210-219, 2007.

[4] M. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767): 335-8, 2000.

[5] B. Fett, S. Bruck, and M. Riedel. Synthesizing stochasticity in biochemical systems. In *Proc. Design Automation Conf.*, 2007.

[6] B. Fett and M. Riedel. Module locking in biochemical systems. In *Proc. Int'l Conf. on Computer-Aided Design*, 2008.

[7] D. T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J. Comput. Phys.*, 22(4): 403-434, 1976.

[8] The iBioSim Tool: `http://www.async.ece.utah.edu/iBioSim/`

[9] H. Jiang, A. Kharam, M. Riedel, and K. Parhi. A synthesis flow for digital signal processing with biomolecular reactions. In *Proc. Int'l Conf. Computer-Aided Design*, 2010.

[10] J. Lucks and A. Arkin. The hunt for the biological transistor. *IEEE Spectrum*, pages 38-43, March 2011.

[11] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4): 541-580, 1989.

[12] C. J. Myers. *Engineering Genetic Circuits*, Chapman & Hall/CRC Press, July 2009.

[13] The Systems Biology Markup Language (SBML) portal: `http://sbml.org/`

[14] Systems Biology Workbench (SBW): `http://sbw.sourceforge.net/`

[15] A. Shea, B. Fett, M. Riedel, and K. Parhi. Writing and compiling code into biochemistry. In *Proc. Pacific Symposium on Biocomputing*, 2010

[16] P. Senum and M. Riedel. Rate-independent constructs for chemical computation. *PLoS ONE*, 6(6), 2011.

[17] D. Soloveichik, G. Seelig, and E. Winfree. DNA as a universal substrate for chemical kinetics. In *Proc. of the National Academy of Sciences*, 107(12): 5393-5398, 2010.