Biochemical Reactions and Biology

- Complex behaviors of a living organism originate from systems of biochemical reactions
- Engineering biochemical reactions may sharpen our understanding on how nature design living organisms (in contrast to how human design electronic systems)

Outline
- Compiling program control flows into biochemical reactions
  - Joint work with Chi-Yun Cheng, De-An Huang, Ruei-Yang Huang [ICCAD 2012]
- Beyond logic computation
Computational Biochemistry

- In living organisms, biochemical reactions carry out some form of “computations” which result in complex behaviors.
- Biochemical reactions may be exploited for computation by combining a proper set of biochemical reactions.
- Computation with biochemical reactions has potential applications in synthetic biology:
  - In synthetic biology, known biochemical parts (DNA, mRNA, proteins, etc.) are assembled either naturally or artificially to realize desired functions.

Previous Work

- Synthesizing molecular reactions has been pursued, e.g.,
  - Arithmetic operations
    - Fett et al. (2007, 2008)
  - Digital signal processing
    - Jiang et al. (2010)
  - Writing and compiling code into biochemistry
    - Shea et al. (2010), Senum et al. (2011)

Previous Work

- Still lack systematic methodology to construct complex program control flows.
- Heavily rely on modularized reactions.
- Assume reactions are of small quantities:
  - Work under stochastic simulation but not ODE simulation.

Our Focus

- Robustness
  - Improved reaction regulation
  - Enhanced fault tolerance
  - Valid under both stochastic and ODE simulations
- Optimality
  - Reduced number of reactions
  - Not limited to modularized reactions
- Systematic compilation methodology
Model of Computation

- Computation with biochemical reactions
  - Computation in terms of molecular quantities
  - Quantity changing rules are defined by reactions

- Example
  
  \[ A + B \rightarrow C \]
  
  \[ [C] = \min\{[A],[B]\} \]

Petri net representation

Reaction Model

- Classical chemical kinetic (CCK) model
  - ODE based simulation
    - (Empirical study shows our construction works for discrete stochastic simulation as well)

- Example:

  \[ \alpha A + \beta B \xrightarrow{k} \gamma C \]

  \[ \frac{-1}{\alpha} \frac{d[A]}{dt} = \frac{-1}{\beta} \frac{d[B]}{dt} = \frac{1}{\gamma} \frac{d[C]}{dt} = k[A]^\alpha [B]^\beta \]

Boolean & Quantitative Abstraction

- Data represented by molecular concentrations

- Control signals in terms of Boolean abstraction

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

Reaction Execution Precedence

- In information processing, computation must be performed in a proper order

- Data dependencies must be maintained to ensure operational correctness
Precondition

- For reaction $X + Z \rightarrow Y + Z$
- $Z_\theta$ is the precondition

$Z \rightarrow Y$

\[ X \xrightarrow{A_\theta \land B_\theta} Y \quad \Rightarrow \quad X + A + B \rightarrow Y + A + B \]

\[ X \xrightarrow{A_\theta \lor B_\theta} Y \quad \Rightarrow \quad \frac{X + A}{X + B} \rightarrow \frac{Y + A}{Y + B} \]

Absence Indicator

- Prior method [Senum and Riedel 2011]
  - Reaction rate $r_f \gg r_s$
  - At equilibrium (when $A$ is present),
    \[ [A'] = \frac{r_f}{r_s} [A] \]

  - The amount of $A'$ is sensitive to the amount of $A$
  - This “leakage” degrades the robustness
  - Can only be applied to stochastic model

Absence Indicator

- Dimerized absence indicator
  - At equilibrium (when $A$ is present),
    \[ A \xrightarrow{r_f} A' \]
    \[ 2A' \xrightarrow{r_s} A^* \]
    \[ [A^*] = \frac{r_s}{r_f} [A']^2 \]

  - $A^*$ is further suppressed by the presence of $A'$
  - $A^*$ remains little even if there is a leakage of $A$
Absence Indicator

\[ R \xrightarrow{-B} G \]
\[ G \xrightarrow{-R_g} B \]
\[ B \xrightarrow{-G_g} R \]

(by ODE simulation with SBW simulator)

Reaction Buffer

- Reaction series:

\[(1) \quad A \rightarrow B \quad B \xrightarrow{-A_g} A \]
\[(2) \quad A \rightarrow B \quad B \xrightarrow{-A_g} C \quad C \xrightarrow{-B_g} A \]

The “buffer” reaction avoids the leakage problem.

Compilation Strategy

1. Identify **Linear, Looping, Branching** statements and create corresponding control flow reactions.
2. Resolve violation of **preconditions** or **postconditions**, and introduce buffer if necessary.
3. Decompose reactions for practical realization
4. Optimize

Linear Flow

- Without regulation, reactions are concurrent

<table>
<thead>
<tr>
<th>Main Reactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 ( A \rightarrow B )</td>
</tr>
<tr>
<td>02 ( C \rightarrow D )</td>
</tr>
<tr>
<td>03 ( E \rightarrow F )</td>
</tr>
</tbody>
</table>
When reactions are intended to be in a sequential manner, precondition can be imposed as follows:

**Main Reactions**

01. $A \to B$
02. $C \to D$  \[\neg A_{\theta}\]
03. $E \to F$  \[\neg C_{\theta}\]

**Branching Statements**

Design judgment carefully. Use mutually exclusive postconditions to enter different code block.
Branching Statements

Different reaction leads to the same ending reaction

Petri net visualization of branching statements:
Using $A + B \rightarrow C$ to judge $(A \geq B)$?
Looping Statement

Main Reactions  Preconditions
01 $Q \rightarrow R$
02 while $P(A, B)$ $\neg Q_\vartheta \cdot \neg F_\vartheta$
03 ... $\rightarrow F$ $\text{Post}(P)$
... ... ... ...
04 $F \rightarrow ...$
05 $X \rightarrow Y$ $\text{Post}(\neg P)$

Judgment is made when
(1) Previous reaction finished
(2) End of every single loop

Looping Statement

Main Reactions  Preconditions
01 $Q \rightarrow R$
02 while $P(A, B)$ $\neg Q_\vartheta \cdot \neg F_\vartheta$
03 ... $\rightarrow F$ $\text{Post}(P)$
... ... ... ...
04 $F \rightarrow ...$
05 $X \rightarrow Y$ $\text{Post}(\neg P)$

Use judgment result as precondition to determine whether entering loop or not.

Loop body

Looping Statement

Main Reactions  Preconditions
01 $Q \rightarrow R$
02 while $P(A, B)$ $\neg Q_\vartheta \cdot \neg F_\vartheta$
03 ... $\rightarrow F$ $\text{Post}(P)$
... ... ... ...
04 $F \rightarrow ...$
05 $X \rightarrow Y$ $\text{Post}(\neg P)$

Executed when Looping condition not satisfied

Loop body

Case Study: Division

Division($A, B$)

begin
01 while $A \geq B$
02 $A := A - B$
03 $Q := Q + 1$
04 $R := A$
end
Case Study: Division

Main Reactions | Preconditions
---|---
01 while \([A] \geq [B]\) | 
02 \((A + B \rightarrow D)\) \(-G_a\) | 
03 \(C \rightarrow Q + E\) \(A_b \wedge \neg B_b\) | 
04 \(D \rightarrow F\) \(-C_b\) | 
05 \(E \rightarrow G\) \(-D_b\) | 
06 \(F \rightarrow B\) \(-E_b\) | 
07 \(G \rightarrow C\) \(-F_b\) | 
08 \(D \rightarrow R\) \(-A_b\) |

C of unit amount initially

If \(A > B\), B will exhaust and A will have amount \([A] - [B]\) remains.

\[ Q := Q + 1 \]

\[ A := A - B \]

The amount \(Q\) increment by \(1\) (We set \(C := 1\)).
Case Study: Division

Main Reactions

<table>
<thead>
<tr>
<th>No.</th>
<th>Reaction</th>
<th>Preconditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>(A + B \rightarrow D)</td>
<td>(-G_D)</td>
</tr>
<tr>
<td>02</td>
<td>(A + B \rightarrow C)</td>
<td>(-G_B)</td>
</tr>
<tr>
<td>03</td>
<td>(C \rightarrow D)</td>
<td>(-B_C)</td>
</tr>
<tr>
<td>04</td>
<td>(D \rightarrow E)</td>
<td>(-B_D)</td>
</tr>
<tr>
<td>05</td>
<td>(E \rightarrow F)</td>
<td>(-B_E)</td>
</tr>
<tr>
<td>06</td>
<td>(F \rightarrow G)</td>
<td>(-B_F)</td>
</tr>
<tr>
<td>07</td>
<td>(G \rightarrow H)</td>
<td>(-B_G)</td>
</tr>
<tr>
<td>08</td>
<td>(H \rightarrow I)</td>
<td>(-B_H)</td>
</tr>
</tbody>
</table>

\[B \leftarrow D\]

\(B\) must regain value before next judgment

Case Study: GCD

```
begin
  while \(A \neq B\)
  if \(A > B\)
    \(A := A - B\)
  else if \(B > A\)
    \(B := B - A\)
  else
    swap(A, B)
end
```

\(GCD := A\)

may cause a serious problem because \([A]\) almost never exactly equal \([B]\)

\(1000 \neq 1001\)
Case Study: GCD

\[ \text{GreatestCommonDivisor\_err\_toler}(A, B, Z) \]
begin
\begin{align*}
01 & \text{while } |A - B| > Z \\
02 & \quad \text{if } A > B + Z \\
03 & \quad \quad A := A - B \\
04 & \quad \text{else if } B > A + Z \\
05 & \quad \quad \text{swap}(A, B) \\
06 & \quad GCD := A \\
end
\end{align*}

is a necessarily small constant, indicating the error tolerant range.

Future work

- Biological realization of the compilation approach
- Minimization of molecular species
- From discrete/static to continuous/dynamic computation?
  - Feedbacks
  - Robustness issues

Correct answer obtained by enhanced error tolerance

(ODE simulation with SBW)
Outline

- Compiling program control flows into biochemical reactions

- Beyond logic computation

Beyond Logic Computation

- A signal processing perspective
  - Modulators, filters, and signal processing

- A control perspective
  - Sensors, actuators (motors), and decision making

Signal Processing Perspective

- Transcription factor translocation example
  
  Nuclear translocation of *S. cerevisiae* stress response TF Msn2

Signal Processing Perspective

- TF translocation example (cont’d)
Signal Processing Perspective

- TF translocation example (cont’d)

Control Perspective

- Quorum sensing example

Control Perspective

- Chemotaxis example

Symbiotic association between the squid Euprymna scolopes and the luminous bacterium Vibrio fischeri.
Control Perspective

- Chemotaxis example (cont’d)

Beyond Systems Biology

- Systems biology share strong similarities with systems neuroscience, although fundamental mechanisms are quite different
  - In biology, biochemical reactions are the fundamental mechanism
  - In neuroscience, electrical communications are the fundamental mechanism

Connection to Neuroscience

- C. elegans nervous system
Connection to Neuroscience

- Connectome of C. elegans (302) neurons

Summary

- Program control flows can be systematically compiled into biochemical reactions
- Discrete computation, though convenient abstraction for genetic circuits, is not a universal approach to systems and synthetic biology
- New computation models needed to decipher various open questions in systems biology and systems neuroscience