Outline

• Reactive Systems
  o Processes vs. Functions

• Processes and Chemistry
  o Chemical modeling
  o Biochemical modeling (complexation etc.)

• Modeling Combinatorial Systems
  o General strong points of “agent-based” or “reactive” modeling languages
Dynamic Interactions

possible bindings

binding

fewer bindings

modification

different bindings

Mathematical modeling: What function does a protein compute?
What function does a cell compute?

Macromolecules and their Machines

Gene Machine
Nucleotides

Regulation
- Gene Regulatory Networks

Gene Interactions
- Holds receptors, actuators
- Hosts reactions

Protein Machine
Amino acids

- Metabolism, Propulsion
- Signal Processing
- Molecular Transport

Machine Networks
- Makes proteins, where/when/how much
- Directs membrane construction and protein embedding

Membrane Machine
Phospholipids

Confinement
- Storage
- Bulk Transport

Biochemical Networks
- Glycan Machine
  - Surface and Extracellular Features

Transport Networks
Reactive Systems

- Developed in Computer Science to get away from the strict notion of input/output computation
- Even more fundamentally, to get away from the very notion of functions as mathematical modeling tools

- We realized in the 60's–80s' that we could not practically model/implement operating systems and computer networks as imperative programs, because of combinatorial state explosion (which is also pervasive in biology)
- And we struggled theoretically to model them via appropriate mathematical functions, because of concurrency and nondeterminism (which are also pervasive in biology)
- What function does the internet (or a protein / gene / membrane / cell) compute?

- This led to a fundamental change in point of view of how to model systems in general
How does X compute

• Change the question from What to How
  o How does an Operating System compute?
  o How does the Internet compute?
  o How does a Cell compute?

• I.e.: what steps does X perform?
  o We can talk about what steps a subsystem performs in reaction to a stimulus from the environment
  o And what are the local consequences of those steps
  o (And this turns out to include functions as a special case)
What is reactive computation?

• Functions
  o Functions are supposed to \textit{terminate} and give a final answer
  o But if e.g. the internet terminates, it's useless: whatever answers it gives, they are never final. And if a cell terminates, it's dead.

• Reactive Systems (by many names: \textit{agent-based}, etc.)
  o Systems that accept inputs and provide outputs, but not necessarily in a "functional" way, e.g. because they have multiple interaction points, they have internal state, or have "a mind of their own" that does not even depend on the inputs.
  o They \textit{react} to the environment. The environment \textit{reacts} to them.
  o In this view computation is \textit{reaction}, or more symmetrically is \textit{interaction}, or is \textit{communication} when something is exchanged during interaction.
Functions vs. Processes

We analyze the notion of "computation step" for functions, and we generalize it to processes

(1) A "Process" is a component of a reactive system
Function Diagrams

- We use diagrams to emphasize simple computation steps

![Function Diagram]

- (A function of two inputs would be a function of a single input that is a pair)

- Function composition \((g \circ f)\)
  
  N.B. we rename \(out\) to \(temp\) in \(f\), and \(in\) to \(temp\) in \(g\), so they connect through their now common channel \(temp\)
Process Diagrams

- Multiple input and multiple output channels

  ![Diagram of a process P with inputs and outputs]

  This is not a function of 2 inputs. Don’t think And–gate (a Boolean function with 2 in 1 out)

  Think Flip–Flop (a "thing" with 2 independent set/reset inputs and 2 separate outputs)

- In “pure interaction” there is actually no difference between input and output channels: they just interact symmetrically with the environment

- But in “communication” there is a direction in which messages are received or sent. We may or may not label receive–channels as “in” and send–channels as “out”

- Specific “channel names” become fundamental to describe how processes interconnect (unlike functions, which have 1 default input and 1 default output channel name)
• Process composition \((Q \mid R)\)

A specific wiring is obtained by (re)naming of the connections ahead of time

Rules:

\[
\begin{align*}
P \mid 0 &= P \\
P \mid Q &= Q \mid P \\
P \mid (Q \mid R) &= (P \mid Q) \mid R
\end{align*}
\]

0 is the null process that has no channels

\((P \mid P \neq P \text{ in general})\)
Functions as Processes

\[ f(x) = \sqrt{x} \]

Reactive Syntax

\[ f = ?\text{in}(x); !\text{out}(\sqrt{x}) \]

Reactive Diagram

“read from in into x; then write \( \sqrt{x} \) to out”
Composing Functions

\[ g(x) = (f \circ f)(x) \quad (\text{=} f(f(x))) \]

\[ x \rightarrow \sqrt{x} \rightarrow \text{out} \quad \text{temp} \]

\[ y \rightarrow \sqrt{y} \rightarrow \text{out} \]

\[ x \rightarrow \sqrt{x} \rightarrow \text{temp} \]

\[ x \rightarrow \sqrt{\sqrt{x}} \rightarrow \text{output} \]
Composing Functions

\[ g(x) = (f \circ f)(x) \]

```
g = (\nu \text{temp})
?in(x); !\text{temp}(\sqrt{x}) |
?\text{temp}(y); !\text{out}(\sqrt{y})
```

“create a new channel and use it to compose two copies of f”
A Process that is not a Function

Reactive Diagram

Reactive Skeleton

Reactive Syntax

\[ P = \ ?in_1(x); \ ?in_2(y); \ !out_1(x+y); \ P \oplus \ ?in_3(z); \ !out_2(\sqrt{z}); \ !out_1(2z); \ 0 \]
Ex. Vending Machine

A candy for 2 coins

VM =

?coin(c1);
(  ?coin(c2); !deliver(candy); VM
⊕  ?abort(); !refund(c1); VM
⊕  ?shake(); 0)
⊕  ?shake(); 0

Note that we are not describing the environment.
Dynamic Networks

- Here is where (static) diagrams fail, and syntax *has* to take over

  - Proliferation/degradation
    \[
    P = \ ?\text{in}(); (P \mid P)
    \]
    \[
    P = \ ?\text{in}(); 0
    \]

  - Interface modification
    \[
    P = \ ?\text{in}(x); \ !x(); Q
    \]
    channel passing!

  - Biological (and software) networks do both things a lot, as opposed to e.g. electrical networks
That’s $\pi$–calculus

• To compose processes $P$ we need:
  o Composition: $P | P$ (with identity elem. $0$)
  o Channel creation: $(\nu x)\, P$ (with $x$ bound in $P$)
  o Recursion: $*P$ (equal to $P | *P$)

• To perform computation steps we need:
  o Channel reading: $?c(x); \, P$ (with $x$ bound in $P$)
  o Channel writing: $!c(M); \, P$ (with message $M$)
  o Choice: $P \oplus P$ (with identity elem. $0$)

• … and channels can be sent as messages!
Generalizing Functions and Automata

• Unlike functions...
  o Processes have multiple, explicitly named, input and output channels.
  o Processes can run in parallel, can deadlock on their inputs, and can be nondeterministic in their outputs.

• Unlike automata (FSA)...
  o Processes can transmit data (not just change state).
  o While automata ‘talk’ to input strings, processes ‘talk’ to other processes: processes are communicating automata.
  o Processes are not “finite state”; they can express unbounded computation in time (divergence) and space (proliferation).
  o They have dynamic connectivity: networks can reshape themselves.
The Kinetics\(^1\) of Computation

1) how things *move*, in this case what steps they take and possibly at what rate.
Function Kinetics

• Functions have a single kinetic law:
  o Application rule: \(f(x) =_{\text{def}} M\{x\}\) then \(f(a) \rightarrow M\{a/x\}\)
    
    e.g.; \(f(x) =_{\text{def}} \text{not}(x)\) then \(f(\text{true}) \rightarrow \text{not}(x)\{\text{true}/x\} = \text{not}(\text{true}) \rightarrow \text{false}\)

• No other kinetic rule is strictly required
  o E.g., no arithmetic: “variable shuffling” is enough to compute anything [Church/Turing]

• The application rule is how functions compute
Process Kinetics

- Processes have one kinetic law:
  - Communication rule
    
    $(?c(x); P\{x\}) \oplus P' \mid (!c(a); Q) \oplus Q' \rightarrow P\{a/x\} \mid Q$

- plus one important identity:
  - Extrusion rule
    
    $((\nu x)P) \mid Q = (\nu x)(P\mid Q)$ for $x$ not occurring in $Q$

- No other kinetic rule is strictly required
  - “channel shuffling” is enough to compute anything [Milner]

- The communication rule is how reactive systems compute
Ex. VM in environment

\[
\text{VM} = \ ?\text{coin}(c1); \\
\quad (\ ?\text{coin}(c2); \ !\text{deliver}(\text{candy}); \ \text{VM} \\
\quad \oplus \ ?\text{abort}(); \ !\text{refund}(c1); \ \text{VM} \\
\quad \oplus \ ?\text{shake}(); \ 0) \\
\oplus \ ?\text{shake}(); \ 0
\]

\[
\text{BUYER} = \\
\quad !\text{coin}(10p); \\
\quad \!\text{coin}(10p); \\
\quad \?\text{deliver}(\text{yum}); \\
\quad \text{EAT}
\]

<table>
<thead>
<tr>
<th>VM</th>
<th>BUYER</th>
</tr>
</thead>
</table>

\[
\rightarrow \ ?\text{coin}(c2); \ !\text{deliver}(\text{candy}); \ \text{VM} \quad | \quad !\text{coin}(10p); \ ?\text{deliver}(\text{yum}); \ \text{EAT} \\
\quad \oplus \ ?\text{abort}(); \ !\text{refund}(10p); \ \text{VM} \\
\quad \oplus \ ?\text{shake}(); \ 0
\]

\[
\rightarrow \ !\text{deliver}(\text{candy}); \ \text{VM} \quad | \quad \?\text{deliver}(\text{yum}); \ \text{EAT}
\]

\[
\rightarrow \ \text{VM} \quad | \quad \text{EAT}
\]

There can be a rate associated with each channel, determining the speed of interactions.
Equivalence

• The "what does X compute?" question can be replaced by equivalence: two systems compute the same thing (whatever that is) if they can replace each other in every possible situation, according to the kinetic laws.

• Two functions are equivalent if they can replace each other in every context made up by other functions (Extensionality)
  o If they produce the same outputs from the same inputs.
  o The Church–Rosser theorem states that the kinetic law for functions leads to deterministic answers: this is non-trivial.

• Two processes are equivalent if they can replace each other in every environment made up by other processes (Contextual congruence)
  o A proof technique for process equivalence is based on showing bisimilarity: for each step one process can make, the other can choose to make a similar step ending up again in equivalent processes.
  o One of the main theorems states that bisimilarity is a congruence, i.e., that if two systems are bisimilar, then they can be exchanged for one another in every environment with no observable difference.
Chemical Networks as Reactive Systems
Chemical Systems

Reactions:

\[ A_1 + A_2 \rightarrow^r B_1 + B_2 \]

Deterministic reaction kinetics

\[ \frac{d[A_i]}{dt} = -r[A_1][A_2] \quad \text{Mass Action Law} \]

(assuming \( B_i \neq A_j \) for all \( i,j \))

Stochastic reaction kinetics

Chemical Master Equation \( \rightarrow \) CTMC
As Reactive Systems

Uniquely-named *asymmetric* \((A_1 \neq A_2)\) reaction \(c\):

\[
c : A_1 + A_2 \rightarrow^r B_1 + B_2
\]

A reaction is *ambiguous* about how the lhs species transfer to the rhs

Processes:

\[A_1 = ?c_r; B_1\]
\[A_2 = !c_r; B_2\]

*(the name of the reaction becomes the channel)*

The reactive system must resolve the ambiguity

or:

\[A_1 = ?c_r; B_2\]
\[A_2 = !c_r; B_1\]

or:

\[A_1 = ?c_r; (B_1|B_2)\]
\[A_2 = !c_r; 0\]

The (quantitative) kinetic laws for processes lead to a CTMC semantics.

E.g., with initial conditions \(2A_1|2A_2\), the CTMC is:

\[
2A_1|2A_2 \quad \overset{r}{\rightarrow} \quad A_1|A_2|B_1|B_2 \quad \overset{r}{\rightarrow} \quad 2B_1|2B_2
\]
As Reactive Systems

Uniquely-named *symmetric* reaction $c$:

$$c : \ A + A \rightarrow^r B_1 + \ldots + B_n$$

Discrete reaction kinetics (A must interact with a copy of itself):

$$A = ?c_{r/2}; \ (B_1|\ldots|B_i) \oplus !c_{r/2}; \ (B_{i+1}|\ldots|B_n) \quad 0 \leq i \leq n$$

With initial conditions $A|A$ (two molecules), the CTMC is as follows; note that each copy of $A$ can do an input or an output, so there are two possible paths to the outcome:

That is:

$$A|A \quad B_1|\ldots|B_n$$
From Reactions to Processes

\[
\begin{align*}
\nu_1 &: A + B \rightarrow k_1 C + C \\
\nu_2 &: A + C \rightarrow k_2 D \\
\nu_3 &: C \rightarrow k_3 E + F \\
\nu_4 &: F + F \rightarrow k_4 B
\end{align*}
\]

**Interaction Matrix**

<table>
<thead>
<tr>
<th>Reaction</th>
<th>Processes</th>
<th>Channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A + B \rightarrow C + C )</td>
<td>( \nu_1(k_1) )</td>
<td>( \nu_2(k_2) )</td>
</tr>
<tr>
<td>( A + C \rightarrow D )</td>
<td>( \nu_1(k_1) )</td>
<td>( \nu_3(k_3) )</td>
</tr>
<tr>
<td>( C \rightarrow E + F )</td>
<td>( \nu_3(k_3) )</td>
<td>( \nu_4(k_4/2) )</td>
</tr>
<tr>
<td>( F + F \rightarrow B )</td>
<td>( \nu_4(k_4/2) )</td>
<td>( \nu_1(k_1) )</td>
</tr>
</tbody>
</table>

**Half-rate for symmetric reactions**

**Degradation reaction** \( \nu_i : X \rightarrow ki P_i \)
- add \( \tau;P_i \) to \( <X,\nu_i> \).

**Asymmetric reaction** \( \nu_i : X + Y \rightarrow ki P_i \)
- add \( ?;P_i \) to \( <X,\nu_i> \) and \( !;0 \) to \( <Y,\nu_i> \).

**Symmetric reaction** \( \nu_i : X + X \rightarrow ki P_i \)
- add \( ?;P_i \) and \( !;0 \) to \( <X,\nu_i> \).

**Read out the processes by rows:**

- \( A = ?\nu_1(k_1);(C|C) \oplus ?\nu_2(k_2);D \)
- \( B = !\nu_1(k_1);0 \)
- \( C = !\nu_2(k_2);0 \oplus \tau_k;3(E|F) \)
- \( D = 0 \)
- \( E = 0 \)
- \( F = ?\nu_4(k_4/2);B \oplus !\nu_4(k_4/2);0 \)
Different Processes for the same Reactions

• That was a systematic way to translate reactions to processes, akin to the way the stoichiometric matrix leads systematically to generating mass action ODEs.

• There can be multiple reaction systems that produce the same ODEs: reactions are finer than ODEs.

• Here there can be multiple process systems that produce the same reactions: processes are finer than reactions.

• Hence there can be better or worse ways to get processes that match certain reactions.

• That is, different processes that produce more compact and/or modular models, but with the same kinetics.
Ex: Catalysis

- Two reactions, same catalyst C
  - According to the general scheme the catalyst uses one channel for each reaction it catalyzes
    \[ C = !a_r; C \oplus !b_r; C \]
    a: \[ A + C \rightarrow^r C + B \]
    A = ?a_r; B
    b: \[ D + C \rightarrow^r C + E \]
    D = ?b_r; E
  - Modularizing: the catalyst has its own catalysis channel c, used for all the reactions it catalyzes:
    \[ C = !c_r; C \]
    A = ?c_r; B
    D = ?c_r; E
  - A very minor improvement in model size here, but these improvement compound in larger models
Chemistry vs. $\pi$-calculus

Chemical reactions

\[ r : A + B \rightarrow_{k_1} C + D \]
\[ s : C + D \rightarrow_{k_2} A + B \]

Does $A$ become $C$ or $D$?

A Petri-Net-like representation. Precise and dynamic, but not modular, scalable, or maintainable.

Reactive system ($\pi$)

A compositional graphical representation (precise, dynamic and modular) and the corresponding calculus.

\[ A = \! r_{k_1} ; C \]
\[ C = ?s_{k_2} ; A \]
\[ B = ?r_{k_1} ; D \]
\[ D = \! s_{k_2} ; B \]

Maps to a CTMC

The same "model"
Biochemical Networks as Reactive Systems
Molecules with State

- Explosion of species, reactions, and state space.

\[ n \text{ modification sites} = 2^n \]
\[ n \text{ molecular states} = 2^n \]
\[ n \text{ ‘species’} = 2^n \]
\[ n \text{ ODEs (mass action)} \]

The master equation will have \(2^n\) ODEs for each molecule!
Connected Molecules

- Further combinatorial explosion

\[ n \text{ states} \times m \text{ states} = n \times m \text{ states} \]

\[ 2^{n_1} \times 2^{n_2} \times \ldots \times 2^{n_m} = \text{BIG} \]

Figure 6. An example of the process diagram for part of the fission yeast cell cycle process represented in Figure 3. Temporal sequence of biochemical processes are represented explicitly. Molecular species appear repeatedly along the interaction processes.

A graphical notation for biochemical networks
Hiroaki Kitano
Iterated Connections (Polymers)

• ‘Infinite’ explosion

An actually infinite number of species and ODEs

\[ p_1 \] (polymer of length 1)
\[ p_2 \] (polymer of length 2)
\[ p_3 \] (polymer of length 3)
...

Copolymer equation

An alternating copolymer has the formula: -A-B-A-B-A-B-A-B-, or \((-A-B-)_n\). The molar ratios of the monomer in the polymer is close to one, which happens when the reactivity ratios \( r_1 \) and \( r_2 \) are close to zero, as given by the Mayo-Lewis equation also called the copolymerization equation:\(^{[11]}\)

\[
\frac{d[M_1]}{d[M_2]} = \frac{[M_1]}{[M_2]} \left( \left[ r_1 [M_1] + [M_2] \right] \right)
\]

where \( r_1 = k_{11}/k_{12} \) and \( r_2 = k_{22}/k_{21} \)
**$\pi$–calculus for Biochemistry**

- **Biochemistry here means**
  - *Direct* modeling of complexation and polymerization, which are fundamental biochemical features.
  - That is, a complex is not a “new species”: it is a structure formed by existing basic species, which can also break apart.

- **We now need the full $\pi$–calculus**
  - We need to create new channels to represent new complexation bonds.
  - We need value-passing so the components of a complex can operate on those bonds: we need to pass *channels over channels*. 
Complexation

\[ A + B \xrightarrow{s \leftrightarrow r} AB \]

There is no good notation for this reaction in chemistry: \( AB \) is considered as a separate species (which leads to combinatorial explosion of models).

But there is a way to write this precisely in \( \pi \)-calculus. Let there be a single public association channel \( \alpha_r \) at rate \( r \), and many private dissociations channels \( \delta_s \) at rate \( s \), one for each complexation event (these are dynamically created by the new-channel operator \( \nu \)):

\[
\begin{align*}
A_{\text{free}} & = (\nu \delta_s) !\alpha_r(\delta_s) ; A_{\text{bound}}(\delta_s) \\
A_{\text{bound}}(\delta_s) & = !\delta_s ; A_{\text{free}} \\
B_{\text{free}} & = ?\alpha_r(\delta_s) ; B_{\text{bound}}(\delta_s) \\
B_{\text{bound}}(\delta_s) & = ?\delta_s ; B_{\text{free}}
\end{align*}
\]

More compactly:

\[
A = (\nu \delta_s) !\alpha_r(\delta_s) ; !\delta_s ; A \\
B = ?\alpha_r(\delta_s) ; ?\delta_s ; B
\]

Note that we are describing \( A \) independently of \( B \): as in the catalysis example, \( A \) could form complexes with many different species over the \( \alpha_r \) channel.
Polymerization

\[
\begin{align*}
M_{\text{free,free}} & = \text{free,free} \\
M_{\text{free,bound}} & = \text{free,bound} \\
M_{\text{bound,bound}} & = \text{bound,bound} \\
M_{\text{bound,free}} & = \text{bound,free}
\end{align*}
\]

\[
\begin{align*}
M_{\text{free,free}} & = ?a(b); M_{\text{bound,bound}}(b) \oplus (\forall b) !a(b); M_{\text{free,bound}}(b) \\
M_{\text{bound,bound}}(l) & = !l; M_{\text{free,bound}}(b) \oplus (\forall b) !a(b); M_{\text{bound,bound}}(l,b) \\
M_{\text{free,bound}}(r) & = !r; M_{\text{free,bound}}(b) \oplus ?a(b); M_{\text{bound,bound}}(b,r) \\
M_{\text{bound,bound}}(l,r) & = !l; M_{\text{free,bound}}(r) \oplus !r; M_{\text{bound,free}}(l)
\end{align*}
\]

\[
M_{\text{free,free}} \mid M_{\text{free,free}} \rightarrow (\forall b) M_{\text{bound,bound}}(b) \mid M_{\text{free,bound}}(b)
\]
Polymerization

- Polymerization is iterated complexation
  - It can be represente in $\pi$–calculus *finitely*, with one process (definition) for each monomer state.
  - Note that polymerization cannot be described *finitely* in chemistry (or ODEs) because there it needs one reaction for each *length* of polymer.
  - The reason it works in $\pi$–calculus is because of the $\nu$ operator. It enables the finite representation of systems of potentially unbounded complexity.
  - As in real biochemistry, where the structure of each monomer is coded in a finite piece of DNA, and yet unbounded–length polymers happen.
Simulation

• It is possible to run simulations (particularly Gillespie–like stochastic simulation)
• For reactive systems with very large or even infinite numbers of species and reactions (like polymerization or unbounded complex formation) where it would be impossible to even write them all down.
• Without ever computing all the reactions or all the possible complexes
• By producing them only during the simulation, as the need arises
• Without any kind of size cut–off or approximation.
Biochemistry vs. $\pi$-calculus

Stoichiometric Matrix (species x reaction),

$$2^n \times 2^n(2^{n-1})$$

System description is exponential in the number of basic components.

The matrix is very sparse, so the corresponding ODE system is not dense. But it still has $2^n$ equations, one per species, plus conservation equations ($[ABC]+[A_pBC]=$constant, etc.).

System description is linear in the number of basic components.

(Its “run-time” behavior or analysis potentially blows-up just as in the previous case, but its description does not.)
Applications
Protein Machine

- Kappa is a (much) more convenient reactive system modeling framework for biochemical systems.

- It takes complexation and post-translational modification as primitives, instead of encoding them by $\pi$-calculus "plumbing" with channels-over-channels. In diagram form, it looks very much like this, but with its own peculiar syntax:

- Its kinetic laws are however more complex. The original paper on Kappa [Danos, Laneve] gives a non-trivial translation from Kappa to $\pi$-calculus.

- Kappa has since been used to model compactly, simulate, and analyze biochemical systems of huge combinatorial complexity [Fontana et al.] including stochastic kinetics.

- Its level of "rule-based" modeling approach is very close to biochemical statements such as: "protein A is phosphorylated on site 1 and then binds to protein B on site 2"
Gene Machine

Simulation of DNA replication fork dynamics and positional occupation over the DNA sequence, compared to nanopore data.
Membrane Machine

**Static Compartments**
Fragment of the hypothalamic system for body weight regulation handling molecular events (receptors, signaling pathways and gene expression) within a heterogeneous cell population sequestered to distinct anatomical compartments.

**Dynamic Compartments**
Qualitative model of a whole process of virus infection and reproduction, including membrane topology transitions.
Conclusions
Reactive Systems

• A model of computation
  o Many flavors, but sharing common principles and techniques
  o Suitable for nondeterministic concurrent systems with unbounded execution, unbounded proliferation, and dynamical connectivity
  o \( \pi \)-calculus as a minimal notation for general reactive systems

• A solution to combinatorial explosion
  o Models are exponentially (for phosphorylation/complexation) or infinitely (for polymerization) more compact.
  o The state space is explored incrementally, and even if the state space is actually infinite (as with polymers) we can still expand it on demand and simulate it with standard techniques.

• Further Reading
  o R. Milner: Communicating and Mobile Systems: The Pi Calculus
  o L. Cardelli: From Processes to ODEs by Chemistry. TCS 273, 261–281, 2008,