Outline

• The Scientific Method
  Its eventual automation

• Models (that know nothing about protocols)
  Chemical Reaction Networks

• Lab Protocols (that know nothing about models)
  Digital Microfluidics

• Integration
  Closed-loop modeling and protocol execution
  The Kaemika App
The Scientific Method

Hasan Ibn al-Haytham (1027) Book of Optics

Galileo Galilei (1638) Two New Sciences
Discovery through Automation

The Scientific Method ~ 2020’s

Central question: How to close the automation loop?

while (true) {
    predict();
    experiment();
    falsify();
}
The Inner Loop

• A *model* is refined by testing a (fixed) *protocol* against a *systems*
• A *protocol* is refined by testing a (fixed) *model* against a *systems*

• Today: publication does not accurately reflect execution

  • Model: poorly-maintained matlab script
  • Protocol: poorly-described manual steps in the lab
  • System: poorly-characterized and hardly “resettable”

• ⇒ Crisis in biology: experiments are done once and are hard to reproduce
  [http://www.nature.com/news/reproducibility-1.17552](http://www.nature.com/news/reproducibility-1.17552)
Models

(those things that know nothing about protocols)
Chemical Reaction Networks (CRN)

\[ X + Y \rightarrow^r Z + W \]

• A \textit{phenomenological model} of kinetics in the natural sciences
  By (only) observing naturally occurring reactions

• A \textit{programming language, finitely} encoded in the genome
  By which living things manage the \textit{unbounded} processing of matter and information

• A \textit{mathematical structure}, rediscovered in many forms
  Vector Addition Systems, Petri Nets, Bounded Context-Free Languages, Population Protocols, ...

• A description of \textit{mechanism} (“instructions” / “interactions”) rather than \textit{behavior} (“equations” / “approximations”)
  Although the two are related in precise ways
  Enabling, e.g., the study of the evolution of \textit{mechanism} through unchanging \textit{behavior}
<table>
<thead>
<tr>
<th>spec</th>
<th>program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y := 2X</td>
<td>X -&gt; Y + Y</td>
</tr>
<tr>
<td>Y := ⌊X/2⌋</td>
<td>X + X -&gt; Y</td>
</tr>
<tr>
<td>Y := X1 + X2</td>
<td>X1 -&gt; Y</td>
</tr>
<tr>
<td>Y := min(X1, X2)</td>
<td>X1 + X2 -&gt; Y</td>
</tr>
</tbody>
</table>
### Advanced Programming Examples

**spec**

\[
\begin{align*}
Y & := \max(X_1, X_2) \\
\end{align*}
\]

**program**

\[
\begin{align*}
X_1 & \rightarrow L_1 + Y \\
X_2 & \rightarrow L_2 + Y \\
L_1 + L_2 & \rightarrow K \\
Y + K & \rightarrow 0 \\
\end{align*}
\]

\[
\max(X_1, X_2) = (X_1 + X_2) - \min(X_1, X_2)
\]

(but is not computed “sequentially”)

#### Approximate Majority

\[
(X, Y) :=
\begin{align*}
\text{if } X & \geq Y \text{ then } (X+Y, 0) \\
\text{if } Y & \geq X \text{ then } (0, X+Y) \\
\end{align*}
\]

\[
\begin{align*}
X + Y & \rightarrow Y + B \\
Y + X & \rightarrow X + B \\
B + X & \rightarrow X + X \\
B + Y & \rightarrow Y + Y \\
\end{align*}
\]
Programming any algorithm as a CRN

A CRN is a finite set of reactions over a finite set of species

CRNs are not Turing complete
  Like Petri nets: reachability is decidable

But unlike Petri nets, CRNs are approximately Turing complete
  Because reactions have also rates
  This makes it possible to approximate Turing completeness by approximating test-for-zero in a register machine. The probability of error (in test-for-zero) can be made arbitrarily small over the entire (undecidably long) computation.

Adding polymerization to the model makes it fully Turing complete
Programming any dynamical system as a CRN

For example, take the canonical oscillator: sine/cosine

∂^2 s = c
∂c = -s

let s = s⁺ - s⁻
let c = (c⁺ - c⁻)

s⁺ -> s⁺ + c⁺
s⁻ -> s⁻ + c⁻
c⁺ -> c⁺ + s⁺
c⁻ -> c⁻ + s⁻
s⁺ + s⁻ -> Ø

∂s⁺ = c⁺
∂s⁻ = c⁻
∂c⁺ = s⁻
∂c⁻ = s⁺

s⁺₀ = max(0, s₀)
s⁻₀ = max(0, -s₀)
c⁺₀ = max(0, c₀)
c⁻₀ = max(0, -c₀)

Δ^2 θ = -g/l sinθ

Equation of motion of a simple pendulum

"elementary"
Programming any dynamical system as a CRN

For example, take the canonical oscillator: sine/cosine

1. Polynomization: All "elementary" ODEs (all those that include polynomials, trigonometry, exponentials, fractions, and their inverses) can be exactly reduced to just polynomial ODEs.

\[
\begin{align*}
\frac{d\theta}{dt} &= \omega \\
\frac{d\omega}{dt} &= -\theta \\
\end{align*}
\]

\[
\begin{align*}
\delta(s^+ - s^-) &= (c^- - c^+) \\
\delta(c^+ - c^-) &= -(s^+ - s^-) \\
\end{align*}
\]

Abstraction of Elementary Hybrid Systems by Variable Transformation

Jiang Liu\(^1\), Naijun Zhan\(^2\), Hengjian Zhao\(^1\), and Liang Zou\(^2\)

---

12
Programming any dynamical system as a CRN

For example, take the canonical oscillator: sine/cosine

1. Polynomization: All "elementary" ODEs (all those that include polynomials, trigonometry, exponentials, fractions, and their inverses) can be exactly reduced to just polynomial ODEs.

2. Positivation: All polynomial ODEs can be exactly reduced to polynomial ODEs in the positive quadrant (as differences).

Biomolecular implementation of linear I/O systems

K. Olavi  E. Klevins
Programming any dynamical system as a CRN

For example, take the canonical oscillator: sine/cosine

1. Polynomization: All "elementary" ODEs (all those that include polynomials, trigonometry, exponentials, fractions, and their inverses) can be exactly reduced to just polynomial ODEs.

2. Positivation: All polynomial ODEs can be exactly reduced to polynomial ODEs in the positive quadrant (as differences).

3. All positivized ODEs are Hungarian: i.e., all negative monomials have their l.h.s. differential variable as a factor.
Programming any dynamical system as a CRN

For example, take the canonical oscillator: sine/cosine

1. Polynomization: All “elementary” ODEs (all those that include polynomials, trigonometry, exponentials, fractions, and their inverses) can be exactly reduced to just polynomial ODEs.

2. Positivation: All polynomial ODEs can be exactly reduced to polynomial ODEs in the positive quadrant (as differences).

3. All positivized ODEs are Hungarian: I.e., all negative monomials have their l.h.s. differential variable as a factor.

4. Hungarization: All Hungarian ODEs can be exactly reduced to mass action CRNs.

\[
\begin{align*}
\frac{ds}{dt} &= c \\
\frac{dc}{dt} &= -s \\
\frac{d(s^+ - s^-)}{dt} &= (c^- - c^+) \\
\frac{d(c^+ - c^-)}{dt} &= -(s^+ - s^-) \\
\end{align*}
\]

let \( s = (s^+ - s^-) \) let \( c = (c^+ - c^-) \)

\[
\begin{align*}
\frac{ds^+}{dt} &= c^+ \\
\frac{ds^-}{dt} &= c^- \\
\frac{dc^+}{dt} &= s^- \\
\frac{dc^-}{dt} &= s^+ \\
\frac{d(s^+ + s^-)}{dt} &= 0 \\
\frac{d(c^+ + c^-)}{dt} &= 0 \\
\end{align*}
\]

Linearity

\[
\begin{align*}
\frac{d(s^+ - s^-)}{dt} &= (c^+ - c^-) \\
\frac{d(c^+ - c^-)}{dt} &= -(s^+ - s^-) \\
\end{align*}
\]

Renaming (Optional)

\[
\begin{align*}
\frac{ds^+}{dt} &= c^+ - s^- \cdot s^+ \\
\frac{ds^-}{dt} &= c^- - s^- \cdot s^+ \\
\frac{dc^+}{dt} &= s^- - c^- \cdot c^+ \\
\frac{dc^-}{dt} &= s^+ - c^- \cdot c^+ \\
\end{align*}
\]

\[
\begin{align*}
s^+_0 &= \max(0, s^0) \\
s^-_0 &= \max(0, -s^0) \\
c^+_0 &= \max(0, c^0) \\
c^-_0 &= \max(0, -c^0) \\
\end{align*}
\]

MOE compilation

Molecular Dynamics

On the Inverse Problem of Reaction Kinetics

V. Hárs – J. Tóth
Programming any dynamical system as a CRN

For example, take the canonical oscillator: sine/cosine

1. Polynomization: All "elementary" ODEs (all those that include polynomials, trigonometry, exponentials, fractions, and their inverses) can be exactly reduced to just polynomial ODEs.

2. Positivation: All polynomial ODEs can be exactly reduced to polynomial ODEs in the positive quadrant (as differences).

3. All positivized ODEs are Hungarian: i.e., all negative monomials have their l.h.s. differential variable as a factor.

4. Hungarization: All Hungarian ODEs can be exactly reduced to mass action CRNs.

5. Molecular Programming: All mass action CRNs, up to time rescaling, can be arbitrarily approximated by engineered DNA molecules.
Chemistry is (also) a formal language that we can use to implement *any* algorithm and *any* dynamical system with *real* (DNA) molecules

- Turing complete and “Shannon complete”

- ANY collection of abstract chemical reactions can be implemented with specially designed DNA molecules, with accurate kinetics (up to time scaling).

- Approaching a situation where we can “systematically compile” (synthesize) a model to DNA molecules, run an (automated) protocol, and observe (sequence) the results in a closed loop.
A Model

A Chemical Reaction Network, provided explicitly or (in this case) generated from a higher-level description of the initial strands, according to the DNA strand displacement rules.
Model Semantics (deterministic)

- ODE semantics of CRNs

State produced by a CRN $C = (\mathcal{A}, \mathcal{R})$ (species $\mathcal{A}$, reactions $\mathcal{R}$) with flux $\mathbf{F}$ (r.h.s. of its mass action ODEs) at time $t$, from initial state $(x_0, V, T)$ (initial concentrations $x_0$, volume $V$, temperature $T$):

$$\begin{align*}
\llbracket((\mathcal{A}, \mathcal{R}, x_0), V, T)\rrbracket(H)(t) &= (G(t), V, T) \\
\text{let } G : [0...H) \to \mathbb{R}^{|\mathcal{A}|} \text{ be the solution of } G(t') = x_0 + \int_0^{t'} F(V, T)(G(s))ds
\end{align*}$$
Summarizing

- Our models are (chemical) programs
- We can compute their behavior (their final state)
- We can (virtually) run them by integration of the ODEs
- We can (physically) run them by DNA nanotech
Protocols

(those things that know nothing about models)
A Protocol
For DNA gate assembly and activation in vitro
Digital Microfluidics

OpenDrop
https://www.youtube.com/watch?v=ncfZWqPm7-4

OpenDrop speed test
https://www.youtube.com/watch?v=p5Is9L_h3Q0

Purple Drop (UW)
Digital Microfluidics

• A general, *programmable*, platform to execute the main liquid-handling operations

• To close the cycle, it can support many automated observation techniques on-board or off-board via peripheral pumps (sequencing, mass spec, ...) although these are all very hardware-dependent.
A Protocol Language

*Samples:* containers with volume, temperature, concentrations

\[
P = \begin{align*}
  & x \quad \text{(a sample variable)} \\
  & (x_0, V, T) \quad \text{(initial condition)} \\
  & \text{let } x = P_1 \text{ in } P_2 \quad \text{(define local variable)} \\
  & \text{Mix}(P_1, P_2) \quad \text{(mix samples)} \\
  & \text{let } x, y = \text{Split}(P_1, p) \text{ in } P_2 \quad \text{(split samples)} \\
  & \text{Equilibrate}(P, t) \quad \text{(equilibrate sample for } t \text{ seconds)} \\
  & \text{Dispose}(P) \quad \text{(discard sample)}
\end{align*}
\]
Protocol Semantics (deterministic)

Each program denotes a final state \(<\text{concentrations, volume, temperature}>\)

\([P]^\rho\) is the final state produced by a protocol \(P\) where \(\rho\) binds its free variables:

\[
[x]^\rho = \rho(x) \\
[x_0, V, T]^\rho = (x_0, V, T) \\
[Mix(P_1, P_2)]^\rho = \\
\text{let } (x_1, V_1, T_1) = [P_1]^\rho \\
\text{let } (x_2, V_2, T_2) = [P_2]^\rho \\
\left(\frac{x_0 V_1 + x_0 V_2}{V_1 + V_2}, V_1 + V_2, \frac{T_1 V_1 + T_2 V_2}{V_1 + V_2}\right)
\]

\[
[\text{let } x = P_1 \text{ in } P_2]^\rho = \\
\text{let } (x_0, V, T) = [P_1]^\rho \\
\text{let } \rho_1 = \rho\{x \leftarrow (x_0, V \cdot p, T), y \leftarrow (x_0, V \cdot (1 - p), T)\} \\
[P_2]^{\rho_1}
\]

(Equilibrate semantics)

\[
[\text{let } x, y = \text{Split}(P_1, p) \text{ in } P_2]^\rho = \\
\text{let } (x_0, V, T) = [P_1]^\rho \\
\text{let } \rho_1 = \rho\{x \leftarrow (x_0, V \cdot p, T), y \leftarrow (x_0, V \cdot (1 - p), T)\} \\
[P_2]^{\rho_1}
\]

(CRN semantics)

\[
[\text{Dispose}(P)]^\rho = (0[A], 0, 0)
\]
Kaemika Microfluidics Compiler

- Mix, split, equilibrate, dispose
- Automatic routing – no geometrical information
- Hot/cold zones
Summarizing

- Our protocols are (liquid handling) programs
- We can compute their behavior (their final state)
- We can (virtually) run them (by simulation)
- We can (physically) run them (by digital microfluidics)
Models *together with* Protocols
Automating “the whole thing”

- Protocols: sets of steps to direct lab machinery (or people)
  - Published in specialized journals. With varying accuracy.
- Models: sets of equations to predict the results of lab experiments
  - Published in Auxiliary Online Materials. With lots of typos.

- Protocols know nothing about models
  - What hypothesis is the protocol trying to test? It is not written in the protocol.
- Models know nothing about protocols
  - What lab conditions are being used to test the model? It is not written in the model.
- While presumably talking about the same system
  - Through the experiment.

- Reproducibility crisis
  - Experiments are hard to reproduce. (materials, conditions, shortcuts)
  - Even models are hard to reproduce! (typos in equations, sketchy diagrams, unexplained graphs, mysterious scripts)

- Similar to classical lifecycle problems in C.S.
  - Documentation (model) gets out of step from code (protocol) if their integration is not automated.
An Integrated Description

Samples: containers with volume, temperature, concentrations

\[
P = x \quad \text{(a sample variable)} \\
(x_0, V, T) \quad \text{(initial condition)} \\
\text{let } x = P_1 \text{ in } P_2 \quad \text{(define local variable)} \\
\text{Mix}(P_1, P_2) \quad \text{(mix samples)} \\
\text{let } x, y = \text{Split}(P_1, p) \text{ in } P_2 \quad \text{(split samples)} \\
\text{Equilibrate}(P, t) \quad \text{(equilibrate sample for } t \text{ seconds)} \\
\text{Dispose}(P) \quad \text{(discard sample)}
\]

Each sample evolves (via Equilibrate) according to a given overall CRN:

\[
\mathcal{C} = (\mathcal{A}, \mathcal{R}) \quad \text{(species, reactions)}
\]
Program Semantics (deterministic)

Each program denotes a final state \( \text{<concentrations, volume, temperature>} \)

\[ [P]^\rho \text{ is the final state produced by a protocol } P \text{ for a fixed CRN } \mathcal{C} = (\mathcal{A}, \mathcal{R}) : \]

\[ [x]^\rho = \rho(x) \]
\[ [x_0, V, T]^\rho = (x_0, V, T) \]
\[ [Mix(P_1, P_2)]^\rho = \]
\[ \text{let } (x_1^1, V_1, T_1) = [P_1]^\rho \]
\[ \text{let } (x_0^2, V_2, T_2) = [P_2]^\rho \]
\[ \frac{x_1^1 V_1 + x_0^2 V_2}{V_1 + V_2}, V_1 + V_2, \frac{T_1 V_1 + T_2 V_2}{V_1 + V_2} \]

\[ [let x = P_1 \text{ in } P_2]^\rho = \]
\[ \text{let } (x_0, V, T) = [P_1]^\rho \]
\[ \text{let } \rho_1 = \rho\{x \leftarrow (x_0, V \cdot p, T), y \leftarrow (x_0, V \cdot (1 - p), T)\} \]

\[ [P_2]^\rho_1 \]

\[ [Equilibrate(P, t)]^\rho = \]
\[ \text{let } (x_0, V, T) = [P]^\rho \]
\[ \text{let } (A, R, x_0, V, T)(H)(t) \]

\[ [Dispose(P)]^\rho = (0^{|A|}, 0, 0), \]

State produced by CRN \( \mathcal{C} = (\mathcal{A}, \mathcal{R}) \) with flux \( F \) at time \( t \):

\[ ((A, R, x_0, V, T))(H)(t) = \]
\[ \text{let } G : [0...H] \rightarrow \mathbb{R}^{|A|} \text{ be the solution of } G(t') = x_0 + \int_0^{t'} F(V, T)(G(s)) \text{ds,} \]
\[ (G(t), V, T) \]
A Joint Semantics

This semantics gives us a joint simulation algorithm, connecting chemical simulation with protocol simulation.

In this presentation everything is deterministic. The state of the protocol is passed to the chemical simulator, which computes a new state that it passes to the protocol simulator, and so on.

Kaemika uses such a joint simulation algorithm for stochastic simulation, passing also variance information back and forth between chemical and protocol simulation. This requires an extension of the above semantics using the Linear Noise Approximation of chemical kinetics, which computes mean and variance of concentrations (both by ODEs, not e.g. by Gillespie algorithm), and a similar extension of the protocol operations.
We can ask: what is the probability of a certain outcome given uncertainties in both the protocol and the model?

Conversely: which parameters of both the protocol and the model best fit the observed result?

Also, we can use Statistical Modelchecking:

1500 executions including protocol uncertainty due timing and pipetting errors (red).
1500 executions including only model uncertainty about rates of the CRN (yellow).
1500 executions including both sources of uncertainty (blue).

We may estimate by Statistic Model Checking, e.g. the probability that Output will fall in a certain range, given distributions over uncertain model and protocol parameters.
Simulating Reaction Networks together with Digital Protocols
An integrated language for chemical models & experimental protocols

Deterministic (ODE) and stochastic (LNA) simulation

Chemical reaction networks (CRNs) and liquid-handling protocols

Reaction scores

Functional scripting

GUI

Search "Kaemika" in the app stores

http://lucacardelli.name/kaemika.html
Kaemika

- A prototype language for chemical models & protocols

- [http://lucacardelli.name/kaemika.html](http://lucacardelli.name/kaemika.html)

- Search "Kaemika" in the App stores

- CRN simulation
- Microfluidics simulation
- Reaction graphs
- ODE equations
- Stochastic noise (LNA)
Main features

- *Species and reactions*
  - Characterized by initial values and rates

- "*Samples" (compartments) and Protocols*
  - Isolate species and reactions in a compartment, and mix compartments

- *Kinetics (simulation)*
  - Deterministic (ODE) or stochastic (LNA) for chemical models
  - Digital microfluidics for chemical protocols

- *Programming abstractions*
  - Assemble models and protocols as compositions of modules
Species and Reactions

//======================================
// Lotka 1920, Volterra 1926
// (simplified with all rates = 1)
//======================================

number x1₀ <- uniform(0,1) // random x1₀
number x2₀ <- uniform(0,1) // random x2₀

species x1 @ x1₀ M      // prey
species x2 @ x2₀ M      // predator

x1 -> x1 + x1  {1}    // prey reproduces
x1 + x2 -> x2 + x2  {1} // predator eats prey
x2 -> Ø        {1}    // predator dies

equilibrate for 40

<= Demo: LotkaVolterra
2AM Oscillator

\[ \partial \mathrm{lo}_1 = - h_1 \cdot \mathrm{lo}_1 - 0.5 \cdot h_2 \cdot \mathrm{lo}_1 + \mathrm{md} + 0.5 \cdot \mathrm{lo}_2 \cdot \mathrm{md} \]
\[ \partial \mathrm{hi}_2 = - 0.5 \cdot h_1 \cdot \mathrm{hi}_2 - h_2 \cdot \mathrm{lo}_2 + h_2 \cdot \mathrm{md} + 0.5 \cdot \mathrm{lo}_1 \cdot \mathrm{md} \]
\[ \partial \mathrm{lo}_2 = 0.5 \cdot h_1 \cdot \mathrm{md} + h_2 \cdot \mathrm{lo}_2 - 0.5 \cdot \mathrm{hi}_1 \cdot \mathrm{lo}_2 + \mathrm{lo}_2 \cdot \mathrm{md} \]
\[ \partial \mathrm{hi}_1 = - h_1 \cdot \mathrm{lo}_1 - 0.5 \cdot \mathrm{hi}_1 \cdot \mathrm{lo}_2 + \mathrm{hi}_1 \cdot \mathrm{md} + 0.5 \cdot \mathrm{hi}_2 \cdot \mathrm{md} \]
\[ \partial \mathrm{md} = 2 \cdot \mathrm{hi}_1 \cdot \mathrm{lo}_1 + 0.5 \cdot \mathrm{hi}_1 \cdot \mathrm{lo}_2 + 0.5 \cdot \mathrm{hi}_2 \cdot \mathrm{lo}_1 - 0.5 \cdot \mathrm{hi}_2 \cdot \mathrm{lo}_2 + \mathrm{lo}_1 \cdot \mathrm{md} + \mathrm{lo}_2 \cdot \mathrm{md} \]
\[ \partial \mathrm{md}’ = 0.5 \cdot \mathrm{hi}_1 \cdot \mathrm{hi}_2 - 0.5 \cdot \mathrm{hi}_1 \cdot \mathrm{md}’ + 2 \cdot \mathrm{hi}_2 \cdot \mathrm{lo}_2 + 0.5 \cdot \mathrm{hi}_2 \cdot \mathrm{lo}_1 - 0.5 \cdot \mathrm{lo}_1 \cdot \mathrm{md}’ + 0.5 \cdot \mathrm{lo}_2 \cdot \mathrm{md}’ \]
\[ \partial \mathrm{var} = \partial \mathrm{cov} = \partial \mathrm{cov} \]
Writing Models Compositionally

• Embedded chemical notation
  Programs freely contain both chemical reactions and control flow
  Can generate unbounded-size reaction networks

• Rich data types
  numbers, species, functions, networks, lists, flows (time-courses)
  flows are composable functions of time used in rates, plotting, and observation

• Modern abstractions
  Functional: programs take data as parameters and produce data as results
  Monadic: programs also produce effects (species, reactions, liquid handling)
  Nominal: lexically scoped chemical species (species are not “strings”)
function Predatorial(number n) {
    if n = 0 then
        define species prey @ 1 M
        prey -> 2 prey // prey reproduces
        report prey
        yield prey
    else
        define species predator @ 1/n M
        species prey = Predatorial(n-1)
        prey + predator ->{n} 2 predator // predator eats
        predator -> Ø // predator dies
        report predator
        yield predator
    end
}

species apexPredator = Predatorial(5)
equilibrate for 50

// Creates a stack of predator-prey relationships in Lotka-Volterra style, and returns the apex predator.

<= Demo: Predatorial
Describing a Protocol

- **Samples** (e.g., test tubes)
  - Are characterized by a volume and a temperature
  - Contain a specified set of species
  - Evolve according to reactions that operates on those species
  - Isolate species and reactions

- **Protocol Operations** (e.g., liquid handling)
  - Accept and produce samples
  - Accepted samples are *used up* (they can only be operated-on once)
Samples

- Samples contain concentrations of species, acted over by reactions.
- Each sample has a fixed volume and a fixed temperature through its evolution.
- Sample concentrations are in units of molarity $M = \text{mol/L}$.
- The default implicit sample is called the vessel $\{1 \text{mL, 20C}\}$

```plaintext
species \{c\} // a species for multiple samples
sample A \{1\mu L, 20C\} // volume and temperature
species a @ 10mM in A // species local to A
amount c @ 1mM in A // amount of c in A
a + c -> a + a

sample B \{1\mu L, 20C\}
species b @ 10mM in B // species local to B
amount c @ 1mM in B // amount of c in B
b + c -> c + c
```

An amount can also be given in grams (if molar mass is specified). The resulting concentration is then relative to sample volume.

```plaintext
species \{NaCl\#58.44\}
sample C \{1mL, 20C\}
amount NaCl @ 8g in C
```

Reactions can be specified with Arrhenius parameters \{collision frequency, activation energy\}. The reaction kinetics is then relative to sample temperature $T$.

```plaintext
a + c -> \{2, 5\} a + a
// rate is 2*e^(-5/(R*T))
```

<= Demo: MixAndSplit
Ex: Serial Dilution (recursive protocol)

Prepare a series of increasingly diluted solutions and apply a network \( f \) to each (\( f \) can add species and reactions to the solutions)

RESULT:
- sample init {1mL, 25°C} \{A = 1M, B = 1M\}
- sample s2 {1mL, 298.2K} \{A = 100mM, B = 100mM\}
- sample s4 {1mL, 298.2K} \{A = 10mM, B = 10mM\}
- sample s7 {1mL, 298.2K} \{A = 1mM, B = 1mM\}
- sample s10 {1mL, 298.2K} \{A = 100uM, B = 100uM\}
Digital Microfluidics Compiler

- Mix, split, equilibrate, dispose
- Automatic routing – no geometrical information
- Hot/cold zones

<= Demo: MixAndSplit
Extracting the Model and the Protocol

From the script

```
species {c}
sample A
  species a @ 1M in A
  amount c @ 0.1M in A
  a + c -> a + a
  equilibrate A1 = A for 1

sample B
  species b @ 1M in B
  amount c @ 0.1M in B
  b + c -> c + c
  equilibrate B1 = B for 1

split C, D = A1 by 0.5
dispose C
mix E = D with B1
  a + b -> b + b
  equilibrate F = E for 20
dispose F
```

The protocol

The (final) model (sample E)
Extracting the Hybrid Transition System

From the script

species \{c\}

sample A
species a @ 1M in A
amount c @ 0.1M in A
a + c -> a + a
equilibrates A1 = A for 1

sample B
species b @ 1M in B
amount c @ 0.1M in B
b + c -> c + c
equilibrates B1 = B for 1

split C,D = A1 by 0.5
dispose C

mix E = D with B1
a + b -> b + b
equilibrates F = E for 20
dispose F

The full story (Hybrid system)
Kaemika: Extra features
Extra features

• General kinetic rates
  • Fractions, rational powers, exponentials, trigonometry. E.g., \( x \rightarrow y \left\{\frac{1}{x}\right\} \)
  • Work with both deterministic and stochastic simulation and equation-extraction
  • Even triggers (discontinuous waveforms)

• Direct ODE notation
  • Instead of a reaction, just write an ODE like \( \frac{dx}{dt} = s \cdot y - s \cdot x \)
  • This is translated to the reaction \( \emptyset \rightarrow x \left\{ s \cdot y - s \cdot x \right\} \) using general kinetic rates

• Timeflows (trajectories as first-class values)
  • Programmable plot reports (e.g., var(2 \cdot a - 3 \cdot b))
  • Capture timeflow outputs to combine (e.g., avg) and re-plot/export them later

• Mass action compiler
  • Turn any elementary ODE system (with fractions, rational powers, exponentials, trigonometry) into an equivalent system of pure mass action reactions.

• Programmable random numbers and distributions
  • As in MIT’s Omega probabilistic language, with rejection sampling.

• Export
  • SBML, ODE, Bitmap, SVG, GraphViz
Reaction scores (graphical representation of reaction networks)


- **Reactants and products**
  - \(a + b \rightarrow c + d\)

- **Repeated species**
  - \(2b \rightarrow c + d\)

- **Reactants but no products**
  - \(a \rightarrow \emptyset\)

- **Products but no reactants**
  - \(\emptyset \rightarrow a + b\)

- **Catalyst**
  - \(b + a \rightarrow a + c\)

- **Catalyst but no reactants**
  - \(a \rightarrow a + c\)

- **Catalyst but no products**
  - \(a + c \rightarrow c\)

- **Autocatalyst**
  - \(a \rightarrow 2a\)
Reaction Scores vs. Reaction Graphs

- 2AM Oscillator

\[
\begin{align*}
hi1 + md &\rightarrow 2hi1 \\
lo1 + hi1 &\rightarrow lo1 + md \\
lo1 + md &\rightarrow 2lo1 \\
hi2 + lo1 &\rightarrow hi2 + md \{0.5\} \\
hi2 + md &\rightarrow hi2 + hi1 \{0.5\} \\
lo2 + hi1 &\rightarrow lo2 + md \{0.5\} \\
lo2 + md &\rightarrow lo2 + lo1 \{0.5\} \\
hi2 + lo2 &\rightarrow hi2 + md_{01} \\
hi2 + md_{01} &\rightarrow 2hi2 \\
lo2 + hi2 &\rightarrow lo2 + md_{02} \\
lo2 + md_{02} &\rightarrow 2lo2 \\
lo1 + lo2 &\rightarrow lo1 + md_{03} \{0.5\} \\
lo1 + md_{03} &\rightarrow lo1 + hi2 \{0.5\} \\
hi1 + hi2 &\rightarrow hi1 + md_{04} \{0.5\} \\
hi1 + md_{04} &\rightarrow hi1 + lo2 \{0.5\}
\end{align*}
\]
Mass Action Compiler

- Lorenz chaotic attractor

\[ \begin{align*}
\frac{dx}{dt} &= s \cdot y - s \cdot x \\
\frac{dy}{dt} &= r \cdot x - x \cdot z - y \\
\frac{dz}{dt} &= x \cdot y - b \cdot z
\end{align*} \]

\( s = 10 \)
\( b = \frac{8}{3} \)
\( r = 28 \)
\( x_0 = 1 \)
\( y_0 = 0 \)
\( z_0 = 28 \)

not mass action

\( x' + x^- \rightarrow \emptyset \)
\( y' \rightarrow y' + x' \) (10)
\( x' \rightarrow x' + x' \) (10)
\( y' \rightarrow y' + x' \) (10)
\( y' + y' \rightarrow \emptyset \)
\( z' + x' \rightarrow z' + x' + y' \)
\( z' + x' \rightarrow z' + x' + y' \)
\( x' \rightarrow x' + y' \) (28)
\( y' \rightarrow y' + y' \)
\( z' + x' \rightarrow z' + x' + y' \)
\( z' + x' \rightarrow z' + x' + y' \)
\( x' \rightarrow x' + y' \) (28)
\( y' \rightarrow y' + y' \)
\( z' + z' \rightarrow \emptyset \)
\( y' + x^- \rightarrow y' + x^- + z' \)
\( y' + x^- \rightarrow y' + x^- + z' \)
\( z' \rightarrow z' + z' \) (2.667)
\( y' + x^- \rightarrow y' + x^- + z' \)
\( y' + x^- \rightarrow y' + x^- + z' \)
\( z' \rightarrow z' + z' \) (2.667)

Initial:
\( x' = 1 \)
\( x^- = 0 \)
\( y' = 0 \)
\( y^- = 0 \)
\( z' = 28 \)
\( z^- = 0 \)

<= Demo: LorenzAttractor
Global Sensitivity Analysis (of a Lotka-Volterra system)

function f(number r1 r2 r3) {  
  define  
  sample S  
  species x1 @ 0.66 M in S  
  species x2 @ 0.44 M in S  
  x1 -> x1 + x1 {r1}  
  x1 + x2 -> x2 + x2 {r2}  
  x2 -> Ø {r3}  
  equilibrate S for 2.5  
  yield [observe(x1,S), observe(x2,S)]  
}

random X(omega w) {  
  f(1+(w(0)-0.5)/10, 1+(w(1)-0.5)/10, 1+(w(2)-0.5)/10)  
}

draw 2000 from X

N.B., consider also exporting your Kaemika model to SBML and use the Sobol’ method of global sensitivity analysis in e.g. Copasi.
Conclusions

Integrated modeling
  Of chemical reaction networks and protocols
  How the Kaemika app supports it
  Why it needs a new language for smooth integration

Closed-loop modeling, experimentation and analysis
  For complete lab automation
  To “scale up” the scientific method

Experimental biological protocols with formal semantics
Alessandro Abate, Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, Boyan Yordanov. CMSB 2018.

Kaemika app - Integrating protocols and chemical simulation
Luca Cardelli. CMSB 2020.

Kaemika User Manual

Thanks to:
  Gold (parser generator)
  OSLO (ODE simulator)
  C#/Xamarin (IDE)
  App store reviewers

NO thanks to:
  XAML (general obfuscator)
  App store certificates
  Dark mode support