Integrated Scientific Modeling and Lab Automation

Luca Cardelli, University of Oxford
SPLASH’21, 2021-10-21
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
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
The Scientific Method

Ḥasan Ibn al-Haytham (1027) Book of Optics

Galileo Galilei (1638) Two New Sciences
Discovery through Observation

The Scientific Method ~ 1638

Discovery through Collaboration

The Scientific Method ~ 2000’s

1 protein = 30 people / 30 years
Humans have >250,000 proteins 😊
Discovery through Automation

The Scientific Method ~ 2020’s

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

- A model is refined by testing a protocol against a systems
- A protocol is refined by testing a 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”

- \(\Rightarrow\) 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)
The Inner Loop

- Tomorrow, automation
  - Model: unambiguous (mathematical) description (CompBio)
  - Protocol: standardized (engineered) parts and procedures (SynthBio)
  - System: characterized (biological) organism and foundries (SysBio)
- Verification: simulation / analysis / model checking / theorem proving
- Observation: lab automation
- Falsification: statistical inference / model reduction
- Performance evaluation/optimization: of model+protocol+system combined
- Management: version control, equipment monitoring, data storage
The Inner Loop

• A specific domain

• Aiming for closed-loop automated modelling and experimentation

• Via Molecular Programming
Models

(those things that know nothing about protocols)

We could choose Differential Equations as our modeling language, as in most of science.

Instead, we choose Chemical Reaction Networks (this is roughly equivalent).

It turns out that in order to "implement differential equations" we need to "implement chemical reactions" anyway (or some other physical realization).
Chemical Reaction Networks (CRN)

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

- A *phenomenological model* of kinetics in the natural sciences
  By *(only) observing* naturally occurring reactions

- A *programming language*, *finitely* encoded in the genome
  By which living things manage the *unbounded* processing of matter and information

- A *mathematical structure*, rediscovered in many forms
  Vector Addition Systems, Petri Nets, Bounded Context-Free Languages, Population Protocols, ...

- A description of *mechanism* ("instructions" / "interactions")
  rather than *behavior* ("equations" / "approximations")
  Although the two are related in precise ways
  Enabling, e.g., the study of the evolution of *mechanism* through unchanging *behavior*
<table>
<thead>
<tr>
<th>spec</th>
<th>program</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Y := 2X )</td>
<td>( X \rightarrow Y + Y )</td>
</tr>
<tr>
<td>( Y := \lfloor X/2 \rfloor )</td>
<td>( X + X \rightarrow Y )</td>
</tr>
</tbody>
</table>
| \( Y := X_1 + X_2 \) | \( X_1 \rightarrow Y \)  
|                           | \( X_2 \rightarrow Y \) |
| \( Y := \min(X_1, X_2) \) | \( X_1 + X_2 \rightarrow Y \) |
Advanced Programming Examples

**spec**

\[
Y := \max(X_1, X_2)
\]

**program**

\[
X_1 \rightarrow L_1 + Y \\
X_2 \rightarrow L_2 + Y \\
L_1 + L_2 \rightarrow K \\
Y + K \rightarrow 0
\]

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

(but is not computed “sequentially”)

**Approximate Majority**

\[
(X, Y) :=
\]

if \(X \geq Y\) then \((X + Y, 0)\)

if \(Y \geq X\) then \((0, X + Y)\)

\[
X + Y \rightarrow Y + B \\
Y + X \rightarrow X + B \\
B + X \rightarrow X + X \\
B + Y \rightarrow Y + Y
\]
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 make 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 CRNs makes them fully Turing complete
Programming any dynamical system as a CRN

Galileo Galilei 1602
Christiaan Huygens 1673

\[ \frac{d^2 \theta}{dt^2} = -\frac{g}{l} \sin \theta \]

Equation of motion of the simple pendulum

https://en.wikipedia.org/wiki/Pendulum

A **dynamical systems** is anything characterized by a system of differential equations (ODEs).

**Elementary dynamical systems** are those that include on the r.h.s. only polynomials, trigonometry, exponentials, fractions, and their inverses. *(All of biochemistry, all of electronics, most of physics.)*

**STEP 1, Polynomization**: Elementary ODEs can be exactly reduced to just polynomial ODEs.
Programming any dynamical system as a CRN

Consider the canonical polynomial oscillator: sine/cosine

\[ \begin{align*}
\partial s &= c \\
\partial c &= -s
\end{align*} \]

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

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

Positivation

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

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

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

Renaming

\( s^+_0 = \max(0, s_0) \)

\( s^-_0 = \max(0, -s_0) \)

\( c^+_0 = \max(0, c_0) \)

\( c^-_0 = \max(0, -c_0) \)

“elementary” A very simple elementary ODE system.

But variables go negative: we can’t have that in a CRN (no negative concentrations).

**STEP 2, Positivation:** Split potentially negative variables of polynomial ODEs into the difference of two positive variables. Obtain the same trajectories as differences.

Biomolecular implementation of linear I/O systems

K. Oishi  E. Klavins
Programming any dynamical system as a CRN

Translate positive ODEs to chemical reactions

The Law of Mass Action tells us how to produce polynomial ODEs from CRNs. The inverse process is called Hungarization, it works for Hungarian ODEs (polynomial ODEs where each negative monomial has the l.h.s. differentiated variable as a factor).

**STEP 3, Hungarization:** Translate polynomial ODEs to chemical reaction networks: each monomial on the r.h.s. produces one reaction.

Subject to the ODEs being Hungarian, but that is always satisfied after positivation!
Programming *any* dynamical system as a CRN

Translate those CNRs to (real, DNA) molecules

Chemistry tells us (sometimes) what reactions molecules obey. The inverse process is possible for DNA molecules, because we can “program” them.

**STEP 4, Molecular programming**: Translate any mass action chemical reaction network into a set of DNA molecules that obey those reactions.

Works up to an arbitrarily good approximation of Mass Action kinetics, and up to time rescaling.
Programming *any* dynamical system as a CRN

Thus, CNRs are “Shannon complete”, and can be physically realized.
Chemistry is (also) a formal language that we can use to implement any dynamical system with real (DNA) molecules.

- 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.

- N.B.: DNA can be used to manipulate and organize programmatically other forms of matter, so this is not really restricted to DNA experiments.
Model Semantics (deterministic)

• ODE semantics of CRNs

Definition  (CRN Flux) Let \((A, R)\) be a CRN. Let \(F(V, T) \in \mathbb{R}^{|A|}_{\geq 0} \to \mathbb{R}^{|A|}\) be the flux of the CRN at volume \(V \in \mathbb{R}_{>0}\) and temperature \(T \in \mathbb{R}_{\geq 0}\). For a concentration vector \(x \in \mathbb{R}^{|A|}_{\geq 0}\) we assume \(F(V, T)(x) = \sum_{\tau \in R} v_\tau \alpha_\tau(V, T, x)\), with stoichiometric vector \(v_\tau\) and rate function \(\alpha_\tau\).

Law of Mass Action \(F(V, T)\) makes up the r.h.s. of an ODE system \(\partial A = F(V, T)\).

State produced by a CRN \(C = (A, R)\) (species \(A\), reactions \(R\)) with flux \(F\) (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 \(T\)):

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

- Recall: we are aiming for models that can be placed into a closed-loop automated model+protocol cycle.
Protocols

(those things that know nothing about models)
A Protocol

For DNA gate assembly and activation in vitro

Protocol steps (liquid handing)
Digital Microfluidics

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

OpenDrop speed test
https://www.youtube.com/watch?v=pSls9L_h3Q0

Purple Drop (UW)

Manipulating droplets by electrical fields
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

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

Each program denotes a final state <concentrations, volume, temperature>

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

\[
[x]_\rho = \rho(x)
\]
\[
[x_0, V, T]_\rho = (x_0, V, T)
\]
\[
[Mix(P_1, P_2)]_\rho =
\]
\[
\text{let} \ (x_1^1, V_1, T_1) = [P_1]_\rho
\]
\[
\text{let} \ (x_2^2, V_2, T_2) = [P_2]_\rho
\]
\[
\left(\frac{x_0^1 V_1 + x_0^2 V_2}{V_1 + V_2}, V_1 + V_2, \frac{T_1 V_1 + T_2 V_2}{V_1 + V_2}\right)
\]
\[
[let \ x = P_1 \ in \ P_2]_\rho =
\]
\[
\text{let} \ (x_0, V, T) = [P_1]_\rho
\]
\[
\text{let} \ \rho_1 = \rho\{x \leftarrow (x_0, V \cdot (1 - p), T)\}
\]
\[
[P_2]_{\rho_1}
\]

(Equilibrate semantics)

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

(CRN semantics)

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

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

sample A {3μL, 20C}
split B, C, D, E = A
mix F = E, C, B, D
dispose F
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
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:

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

Each program denotes a final state <concentrations, volume, temperature>

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

$$[x]^\rho = \rho(x)$$
$$[x_0, V, T]^\rho = (x_0, V, T)$$
$$[Mix(P_1, P_2)]^\rho =$$
$$\quad \text{let } (x_1^1, V_1, T_1) = [P_1]^\rho$$
$$\quad \text{let } (x_2^2, V_2, T_2) = [P_2]^\rho$$
$$\quad \frac{x_1^1V_1 + x_2^2V_2}{V_1 + V_2}, V_1 + V_2, \frac{T_1V_1 + T_2V_2}{V_1 + V_2}$$
$$[let \; x = P_1 \; \text{in} \; P_2]^\rho =$$
$$\quad \text{let } (x_0, V, T) = [P_1]^\rho$$
$$\quad \text{let } \rho_1 = \rho\{x \leftarrow (x_0, V \cdot p, T), y \leftarrow (x_0, V \cdot (1 - p), T)\}$$
$$[P_2]^\rho_1$$

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

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

$$[\text{Dispose}(P)]^\rho = (0^{\{\mathcal{A}\}}, 0, 0),$$

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

$$[((\mathcal{A}, \mathcal{R}, x_0), V, T)](H)(t) =$$
$$\quad \text{let } G : [0...H] \rightarrow \mathbb{R}^{\{\mathcal{A}\}} \text{ be the solution of } G(t') = x_0 + \int_0^{t'} F(V, T)(G(s))ds$$
$$\quad (G(t), V, T)$$
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.
Each program denotes a final state \(<\text{concentrations}; \text{covariances}; \text{volume, temperature}>\)

**A Language for Modeling and Optimizing Experimental Biological Protocols**

Luca Cardelli *, Marta Kwiatkowska and Luca Laurenti

**Definition 3. (CRN Flux)** Let \((A, R)\) be a CRN. Let \(F(V, T) \in \mathbb{R}^{|A|} \rightarrow \mathbb{R}^{|A|}\) be the flux of the CRN at volume \(V \in \mathbb{R}_{\geq 0}\) and temperature \(T \in \mathbb{R}_{\geq 0}\). For a concentration vector \(\mu \in \mathbb{R}^{|A|}\), we assume \(F(V, T)(\mu) = \sum_{c \in C} v_c \alpha_c(V, T, \mu)\), with stoichiometric vector \(v_c\) and rate function \(\alpha_c\). We call \(f_T\) the Jacobian of \(F(V, T)\), and \(f_T^T\) its transpose. Further, define \(W(V, T)(\mu) = \sum_{c \in C} v_c \alpha_c(V, T, \mu)\) to be the diffusion term.

**Definition 4. (CRN Time Evolution)** Given a CRN \((A, R), (\mu, \Sigma, V, T)\), its evolution at time \(t < H\) (where \(H \in \mathbb{R}_{\geq 0} \cup \{\infty\}\) is a time horizon) is the state \((\mu(t), \Sigma(t), V, T)\) obtained by integrating its flux up to time \(t\), where:

\[
\begin{align*}
\mu(t) &= \mu + \int_0^t F(V, T)(\mu(s))ds \\
\Sigma(t) &= \Sigma + \int_0^t f_T(\mu(s))\Sigma(\mu(s)) + \Sigma(\mu(s))f_T^T(\mu(s)) + W(V, T)(\mu(s))ds,
\end{align*}
\]

with \(\mu(0) = \mu\) and \(\Sigma(0) = \Sigma\). If, for such an \(H\), \(\mu\) or \(\Sigma\) are not unique, then we say that the evolution is ill-posed. Otherwise, \(\mu(t)\) and \(\Sigma(t)\) define a Gaussian process with that mean and covariance matrix for \(t < H\).

\[
[x]^p = \rho(x) \\
[(p_1 \cdots p_{|A|}, r_V, r_T)]^p = ([p_1]^p \cdots [p_{|A|}]^p, \sigma^{|A| \times |A|}, r_V, r_T) \\
[\text{let } x = P_1 \text{ in } P_2]^p = [P_2]^p \\
[\text{where } \rho_1 = \rho(x \leftarrow [P_1]^p)] \\
[(\text{Mix}(P_1, P_2))]^p = \left(\frac{V_1 \rho_1 + V_2 \rho_2}{V_1 + V_2}\right) \left(\frac{\Sigma_1 + \Sigma_2}{V_1 + V_2}\right), V_1 + V_2, V_1 T_1 + V_2 T_2 \\
[\text{where } (\mu_1, \Sigma_1, V_1, T_1) = [P_1]^p \text{ and } (\mu_2, \Sigma_2, V_2, T_2) = [P_2]^p] \\
[\text{let } x, y = \text{Split}(P, p) \text{ in } P]^p = [P]^p \\
[\text{where } r = [p]^p, 0 < r < 1 \text{ and } (\mu, \Sigma, V, T) = [P_3]^p] \\
[\text{and } \rho_1 = \rho(x \leftarrow (\mu, \Sigma, r V, T)) = (\mu, \Sigma, (1-r)V, T)] \\
[(\text{Equilibrate}(P, p))]^p = ([\mu(t), \Sigma(t)](t), V, T) \\
[\text{where } t = [p]^p \text{ and } (\mu, \Sigma, V, T) = [P]^p] \\
[(\text{Dispose}(P))]^p = (\sigma^{|A| \times |A|}, 0, 0) \\
\]

Together with \([p]^p\) defined as:

\[
[z]^p = \rho(z) \\
[r]^p = r
\]
Stochastic Analysis

• 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?
• E.g., we can use Statistical Modelchecking to estimate the probability that the output will fall in a certain range, given the distributions over uncertain model and protocol parameters.
Summarizing

Automated discovery loop:

Model+Protocol → Realization (e.g. DNA Synthesis) → Protocol execution → Readout (e.g., DNA sequencing) → Data analysis

DNA compilation → Digital microfluidics compilation → Falsification+Optimization
Simulating Reaction Networks together with Digital Protocols
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 \( x_1 \_0 \) <- uniform(0,1) // random \( x_1 \_0 \)
number \( x_2 \_0 \) <- uniform(0,1) // random \( x_2 \_0 \)

species \( x_1 \) @ \( x_1 \_0 \) M // prey
species \( x_2 \) @ \( x_2 \_0 \) M // predator

\( x_1 \rightarrow x_1 + x_1 \) \{1\} // prey reproduces
\( x_1 + x_2 \rightarrow x_2 + x_2 \) \{1\} // predator eats prey
\( x_2 \rightarrow \emptyset \) \{1\} // predator dies

equilibrate for 40

<= Demo: LotkaVolterra
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”)
Ex: Predatorial (recursive model)

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
Ex: Serial Dilution (recursive protocol)

```plaintext
network SerialDilution(number count, sample s, network f) {
    if count > 0 then
        sample solvent {9*observe(volume,s) L, observe(kelvin,s) K}
        mix s = s, solvent
        split s, dilution = s by 0.1, 0.9
        f(dilution)
        SerialDilution(count-1, s, f)
    end
}

//initial sample to be diluted:
sample init {1mL, 25C}
species A @ 1M in init
species B @ 1M in init
A + B ->{20} A
A -> Ø

//apply this network to each dilution;
//note that this invokes a simulation
//each time in each solution
network test(sample s) {
    equilibrate s for 10
    dispose s
}

//dilute 4 times
SerialDilution(4, init, test)
```

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, 298.2K} {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}
Extracting the Model and the Protocol

From the script

**species** \{c\}

**sample** A
species a @ 1M in A
amount c @ 0.1M in A
\(a + c \rightarrow a + a\)
equilibrate \(A_1 = A\) for 1

**sample** B
species b @ 1M in B
amount c @ 0.1M in B
\(b + c \rightarrow c + c\)
equilibrare \(B_1 = B\) for 1

split C,D = \(A_1\) by 0.5
dispose C

mix E = D with \(B_1\)
\(a + b \rightarrow b + b\)
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
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 full story (Hybrid system)

From the script
Extra features

- General kinetic rates
  - Fractions, rational powers, exponentials, trigonometry. E.g., $x \rightarrow y \{1/x\}$
  - Work with both deterministic and stochastic simulation and equation-extraction
  - Event 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 $\emptyset \rightarrow x \{s \cdot y - s \cdot x\}$ 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
  • A new representation of directed hypergraphs with catalysis
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

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

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