Discovery through Automation

Luca Cardelli, University of Oxford

QAVAS Oxford, 2021-09-28

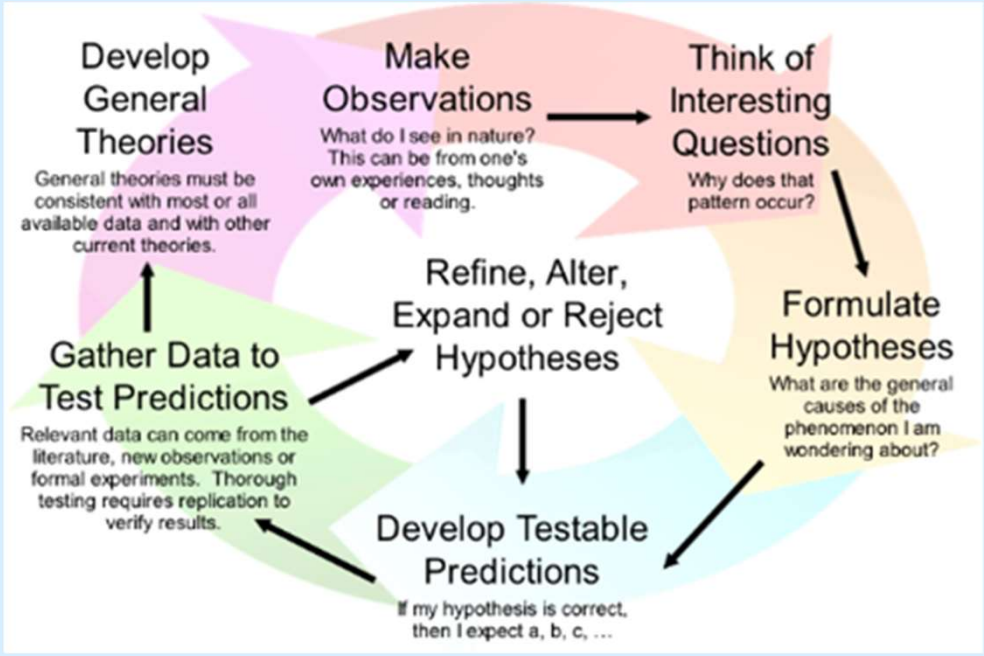
Outline

- 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

Ḥasan 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();
}
```

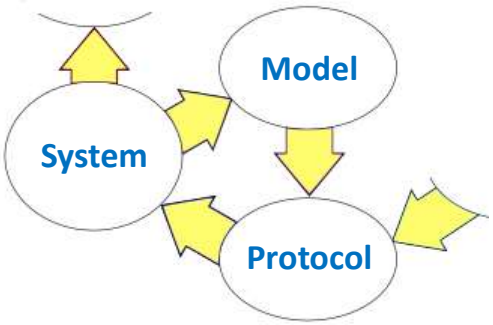
Robot scientist becomes first machine to discover new scientific knowledge

Ross King

Garland, Jr., Theodore. "The Scientific Method as an Ongoing Process". U C Riverside.

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

Models

(those things that know nothing about protocols)

Chemical Reaction Networks (CRN)

$$X + Y \xrightarrow{r} Z + W$$

- 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*

Programming Examples

spec	*program*
Y := 2X	X -> Y + Y
Y := ⌊X/2⌋	X + X -> Y
Y := X1 + X2	X1 -> Y X2 -> Y
Y := min(X1, X2)	X1 + X2 -> Y

Advanced Programming Examples

spec

Y := max(X1, X2)

program

X1 -> L1 + Y
X2 -> L2 + Y
L1 + L2 -> K
Y + K -> 0

max(X1,X2)=
(X1+X2)-min(X1,X2)

(but is not computed "sequentially")

Approximate Majority

(X,Y) :=
if X≥Y then (X+Y, 0)
if Y≥X then (0, X+Y)

X + Y -> Y + B
Y + X -> X + B
B + X -> X + X
B + Y -> Y + Y

Programming *any* algorithm ("approximately") 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.

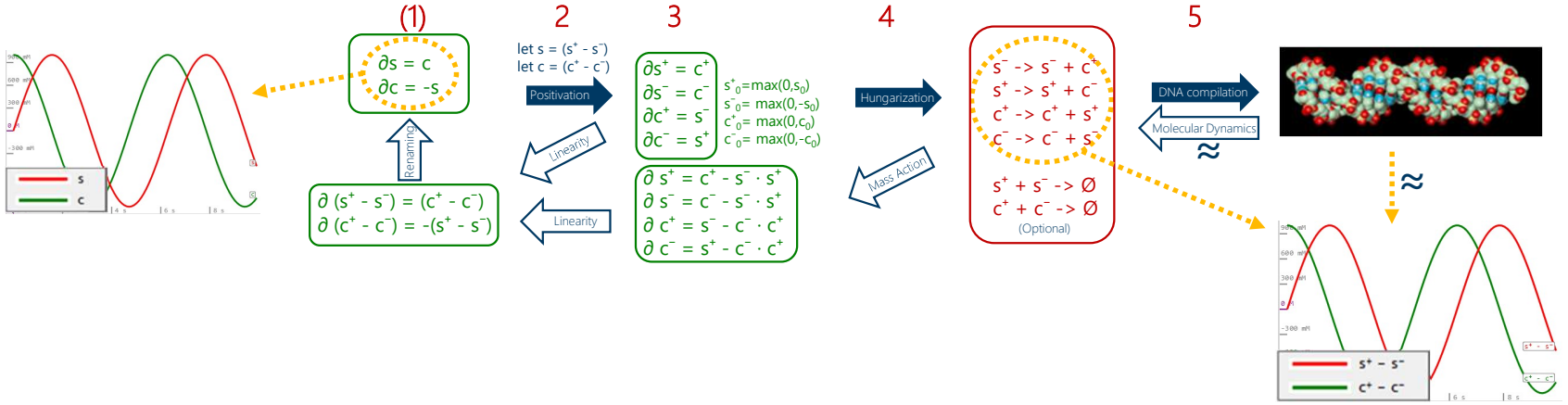
Adding polymerization to the model makes it fully Turing complete

Programming *any* "elementary" dynamical system as a CRN

$\partial^2\theta = -g/l \sin\theta$

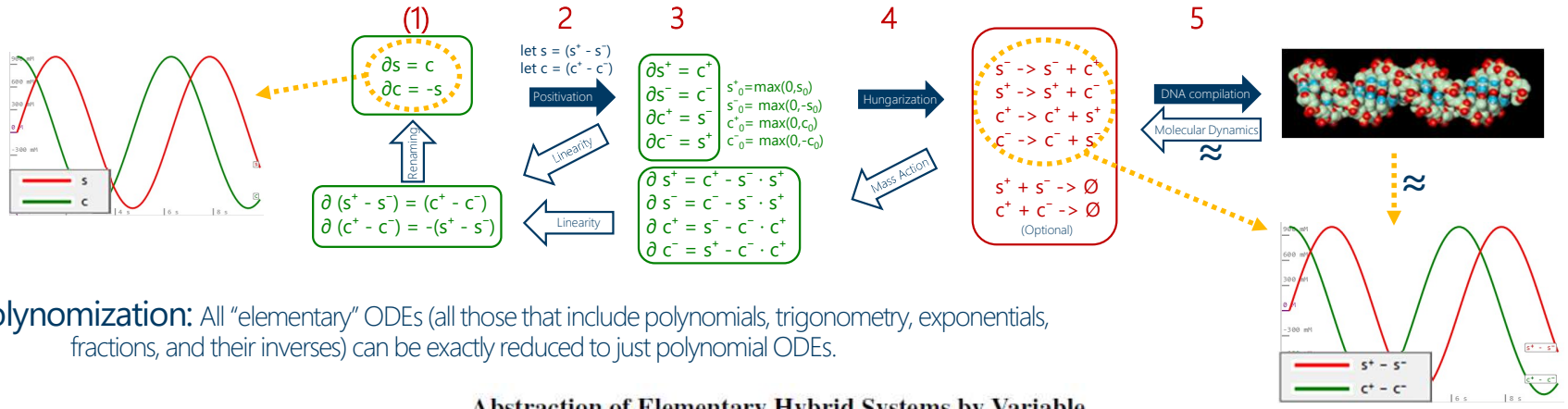
Equation of motion of a simple pendulum

For example, take *the* canonical oscillator: sine/cosine



Programming *any* "elementary" dynamical system as a CRN

For example, take *the* canonical oscillator: sine/cosine



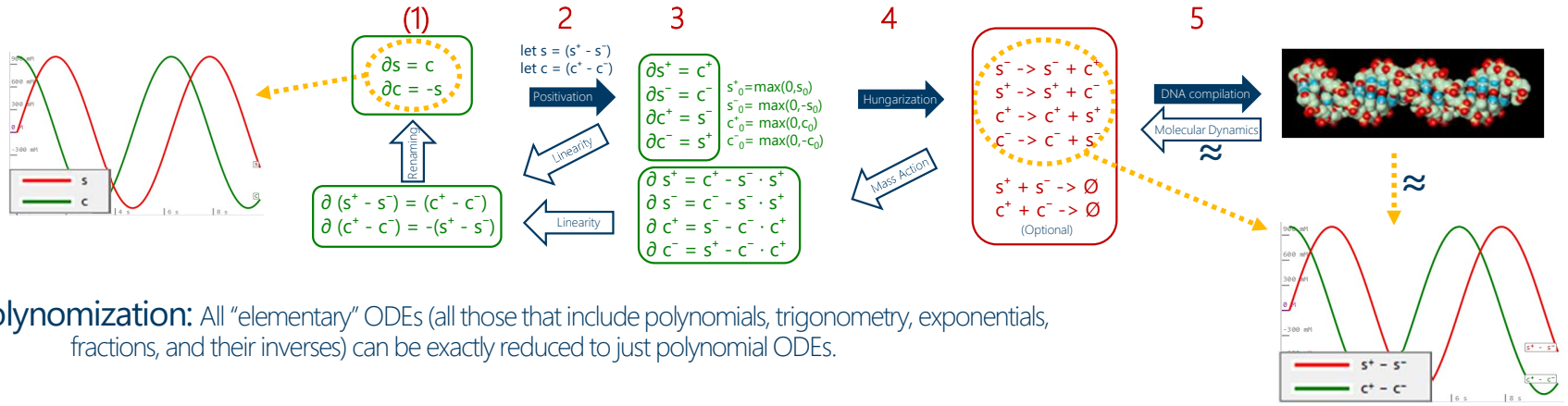
1. Polynomization: All "elementary" ODEs (all those that include polynomials, trigonometry, exponentials, fractions, and their inverses) can be exactly reduced to just polynomial ODEs.

Abstraction of Elementary Hybrid Systems by Variable Transformation

Jiang Liu[1], Naijun Zhan[2], Hengjun Zhao[1], and Liang Zou[2]

Programming *any* "elementary" 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.

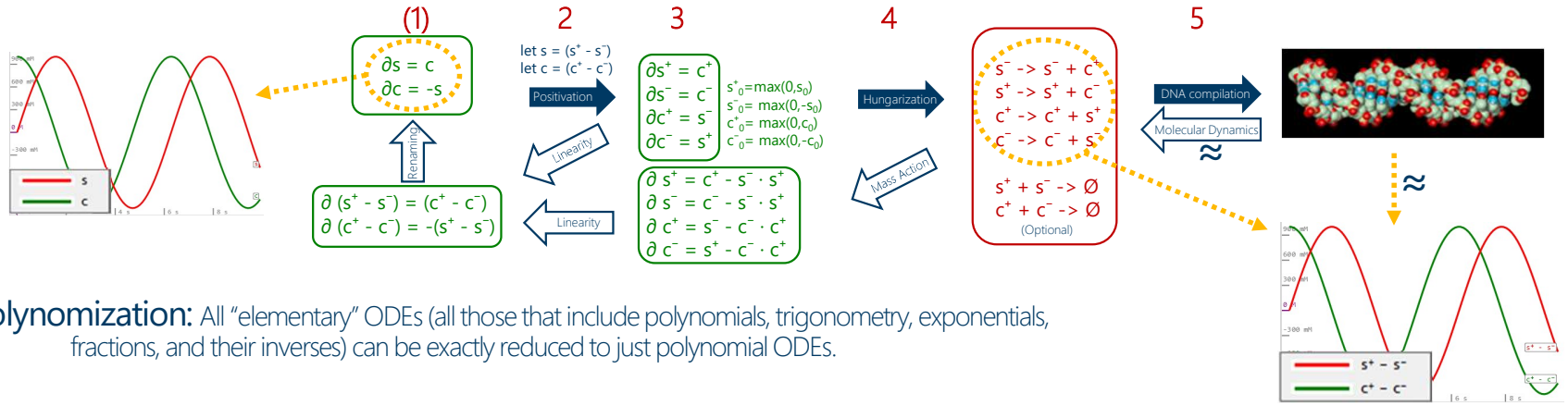
2. Positivation: All polynomial ODEs can be exactly reduced to polynomial ODEs in the positive quadrant (as differences).

Biomolecular implementation of linear I/O systems

K. Oishi E. Klavins

Programming *any* "elementary" dynamical system as a CRN

For example, take *the* canonical oscillator: sine/cosine



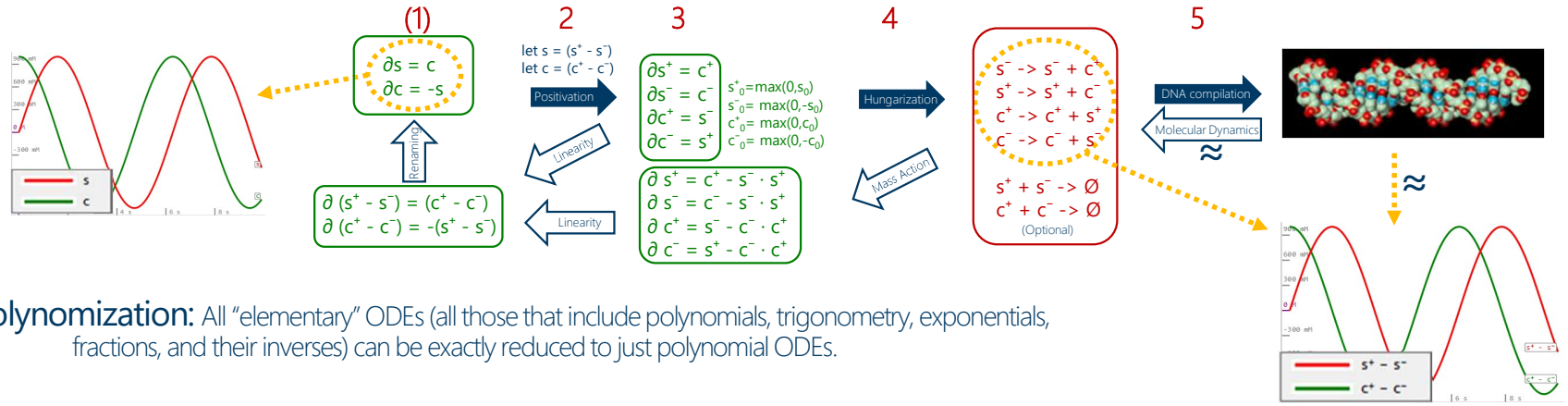
1. **Polynomization:** All "elementary" ODEs (all those that include polynomials, trigonometry, exponentials, fractions, and their inverses) can be exactly reduced to just polynomial ODEs.

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* "elementary" 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.

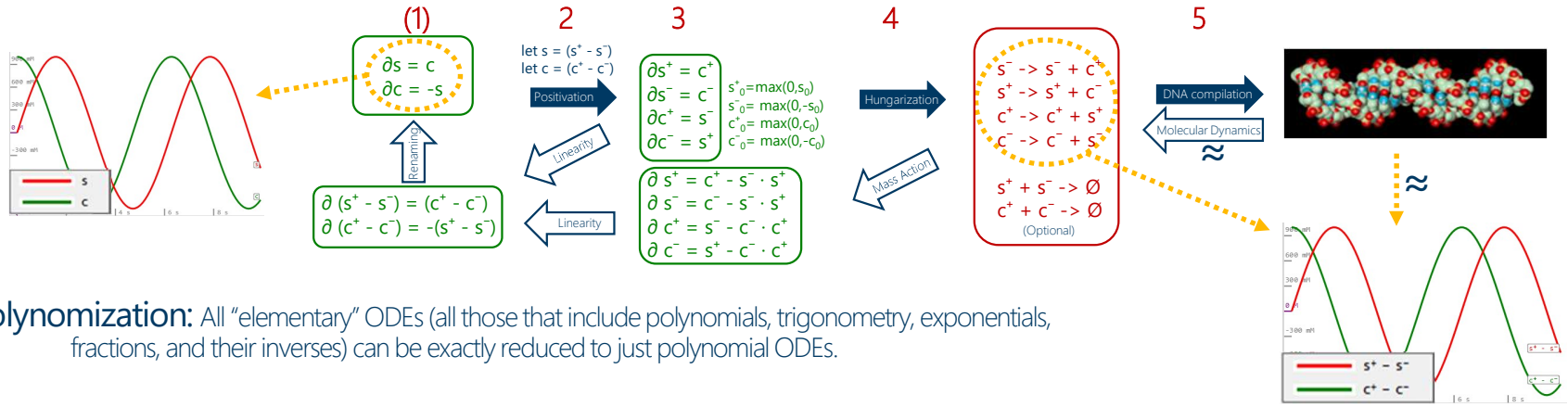
4. **Hungarization:** All Hungarian ODEs can be exactly reduced to mass action CRNs.

ON THE INVERSE PROBLEM OF REACTION KINETICS

V. HÁRS - J. TÓTH

Programming *any* "elementary" 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.

DNA as a universal substrate for chemical kinetics

David Soloveichik, Georg Seelig, and Erik Winfree

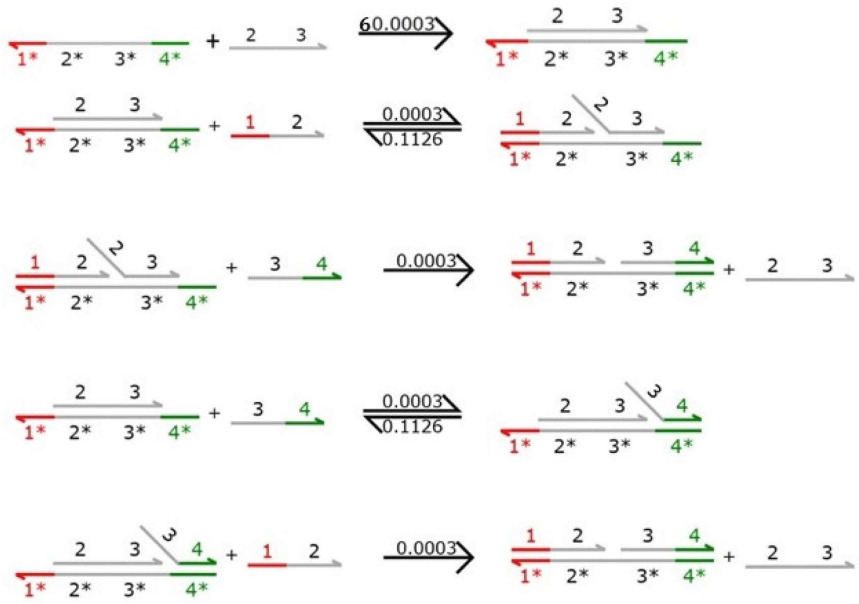
PNAS March 23, 2010 107 (12) 5393-5398; https://doi.org/10.1073/pnas.0909380107

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 $\mathcal{C} = (\mathcal{A}, \mathcal{R})$ (species $\mathcal{A}$, reactions $\mathcal{R}$)
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*):

$$[\![((\mathcal{A},\mathcal{R}, x_0), V, T)]\!](H)(t) = (G(t), V, T)$$

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

Summarizing

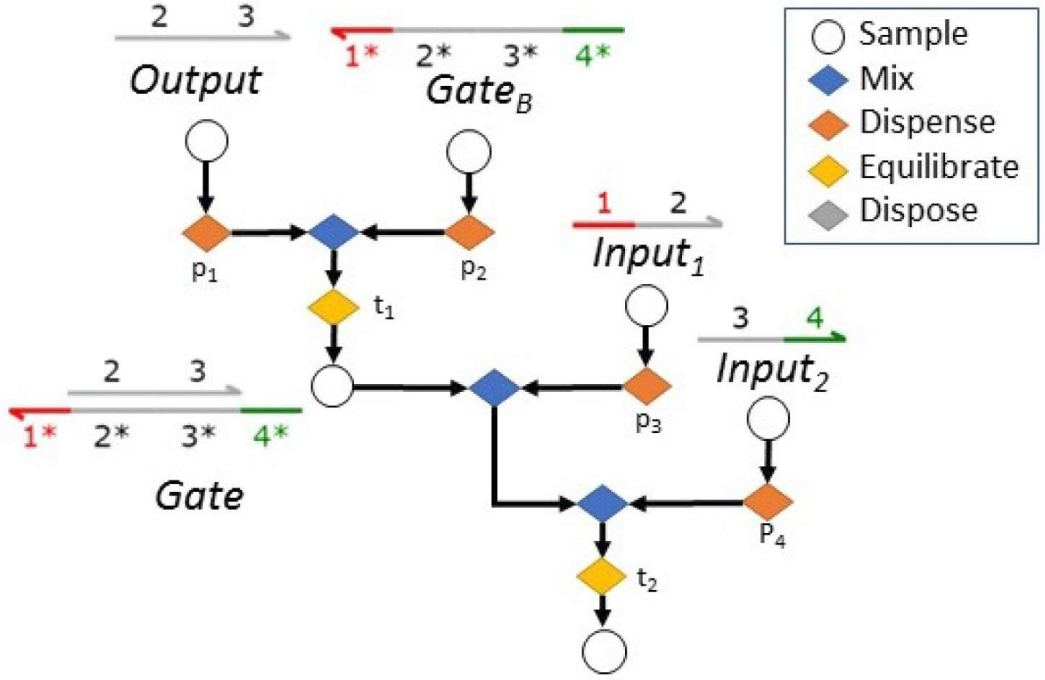
- Our models are (chemical) programs
- We can compute their behavior (their final state)
- We can (virtually) run them by integration of the ODEs
- We can (physically) run them by DNA nanotech

Protocols

(those things that know nothing about models)

A Protocol

For DNA gate assembly and activation in vitro



Protocol steps (liquid handing)

Digital Microfluidics

OpenDrop

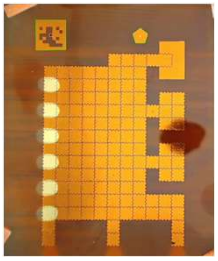
https://www.youtube.com/watch?v=ncfZWqPm7-4

OpenDrop speed test

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

Purple Drop (UW)

https://misl.cs.washington.edu/projects/fluidics.html



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

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

Experimental Biological Protocols with Formal Semantics

Alessandro Abate[2], Luca Cardelli[1,2], Marta Kwiatkowska[2], Luca Laurenti[2], and Boyan Yordanov[1]

[1] Microsoft Research Cambridge
[2] Department of Computer Science, University of Oxford

Protocol Semantics (deterministic)

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

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

$$[\![x]\!]^\rho = \rho(x)$$
$$[\![x_0, V, T]\!]^\rho = (x_0, V, T)$$
$$[\![Mix(P_1, P_2)]\!]^\rho =$$
$$\quad let\ (x_0^1, V_1, T_1) = [\![P_1]\!]^\rho$$
$$\quad let\ (x_0^2, V_2, T_2) = [\![P_2]\!]^\rho$$
$$\quad (\frac{x_0^1 V_1 + x_0^2 V_2}{V_1 + V_2}, V_1 + V_2, \frac{T_1 V_1 + T_2 V_2}{V_1 + V_2})$$
$$[\![let\ x = P_1\ in\ P_2]\!]^\rho =$$
$$\quad let\ (x_0, V, T) = [\![P_1]\!]^\rho$$
$$\quad let\ \rho_1 = \rho\{x \leftarrow (x_0, V, T)\}$$
$$\quad [\![P_2]\!]^{\rho_1}$$

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

(*Equilibrate semantics*)

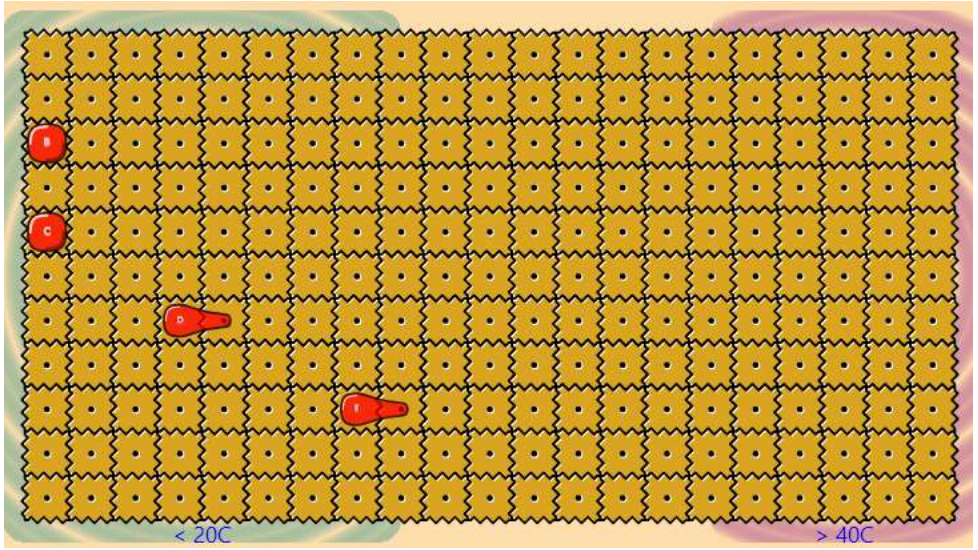
$$[\![Dispose(P)]\!]^\rho = (0^{|\Lambda|}, 0, 0),$$

(*CRN semantics*)

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

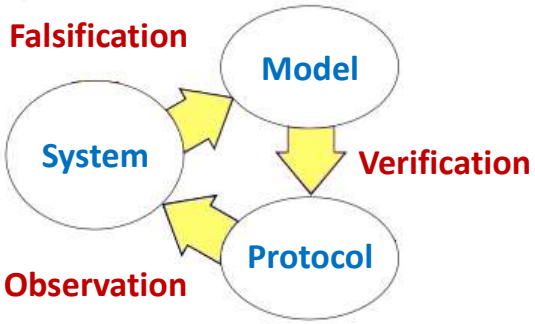
- Our protocols are (liquid handling) programs
- We can compute their behavior (their final state)
- We can (virtually) run them (by simulation)
- We can (physically) run them (by digital microfluidics)

Models *together with* Protocols

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

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

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

$\mathcal{C} = (\mathcal{A}, \mathcal{R})$ (species, reactions)

Experimental Biological Protocols with Formal Semantics

Alessandro Abate[2], Luca Cardelli[1,2], Marta Kwiatkowska[2], Luca Laurenti[2], and Boyan Yordanov[1]

[1] Microsoft Research Cambridge
[2] Department of Computer Science, University of Oxford

Protocol + Model

=

Joint script

$Input_1 =< 1^*\ 2 >\quad Output =< 2\ 3 >$
$Input_2 =< 3\ 4^* >\quad Gate = \{1^*\}[2\ 3]\{4^*\}$

$$
\begin{aligned}
P_1 = & let\, In1 = ((Input1, 100.0nM), 0.1mL, 25.0^oC)\, in\\
& let\, In2 = ((Input2, 100.0nM), 0.1mL, 25.0^oC)\, in\\
& let\, GA = ((Output, 100.0nM), 0.1mL, 25.0^oC)\, in\\
& let\, GB = ((Gate_B, 100.0nM), 0.1mL, 25.0^oC)\, in\\
& let\, sGA, = Dispense(GA, p_1)\, in\\
& let\, sGB, = Dispense(GB, p_2)\, in\\
& let\, sIn1, = Dispense(In1, p_3)\, in\\
& let\, sIn2, = Dispense(In1, p_4)\, in\\
& Observe(Equilibrate(Mix(Mix(Equilibrate(\\
& \quad Mix(sGA, sGB), t_1), sIn1), sIn2), t_2), idn).
\end{aligned}
$$

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 $\mathcal{C} = (\mathcal{A}, \mathcal{R})$:

$$[\![x]\!]^\rho = \rho(x)$$

$$[\![x_0, V, T]\!]^\rho = (x_0, V, T)$$

$$[\![Mix(P_1, P_2)]\!]^\rho =$$
$$let\,(x_0^1, V_1, T_1) = [\![P_1]\!]^\rho$$
$$let\,(x_0^2, V_2, T_2) = [\![P_2]\!]^\rho$$
$$(\frac{x_0^1 V_1 + x_0^2 V_2}{V_1 + V_2}, V_1 + V_2, \frac{T_1 V_1 + T_2 V_2}{V_1 + V_2})$$

$$[\![let\,x = P_1\,in\,P_2]\!]^\rho =$$
$$let\,(x_0, V, T) = [\![P_1]\!]^\rho$$
$$let\,\rho_1 = \rho\{x \leftarrow (x_0, V, T)\}$$
$$[\![P_2]\!]^{\rho_1}$$

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

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

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

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

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

A Joint Semantics

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

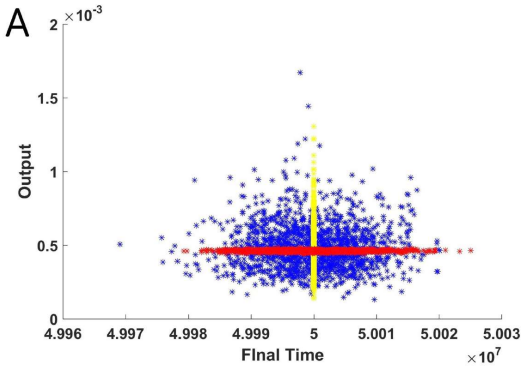
In this presentation everything is *deterministic*. The state of the protocol is passed to the chemical simulator, which computes a new state that it passes to the protocol simulator, and so on.

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.

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?
- 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

Kaemika

/'kimika/

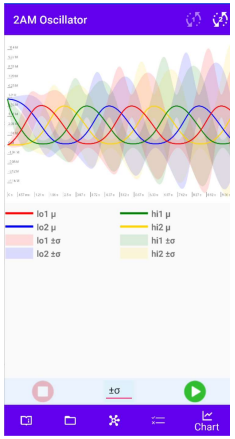
macOS

iOS

Android

Windows

GitHub



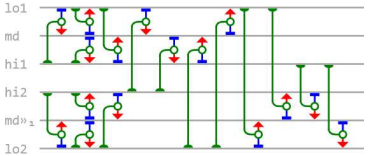
Search "Kaemika" in the app stores
http://lucacardelli.name/kaemika.html

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

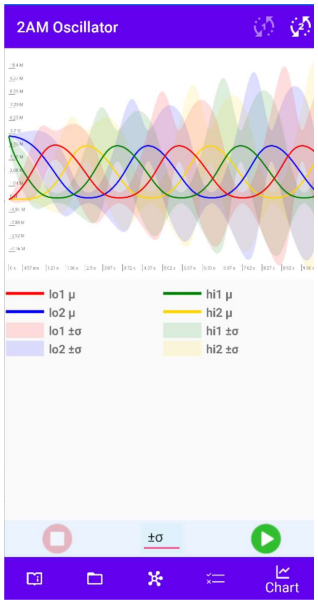
Kaemika

- A prototype language for chemical models & protocols

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

- Search "Kaemika" in the App stores

macOS
iOS
Android
Windows
GitHub



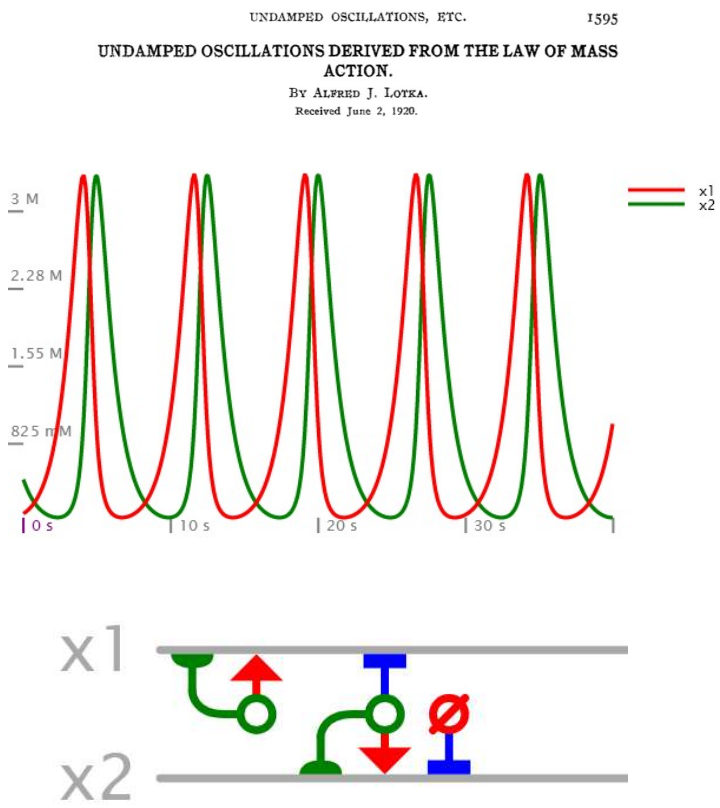
- CRN simulation
- Microfluidics simulation
- Reaction graphs
- ODE equations
- Stochastic noise (LNA)

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
```

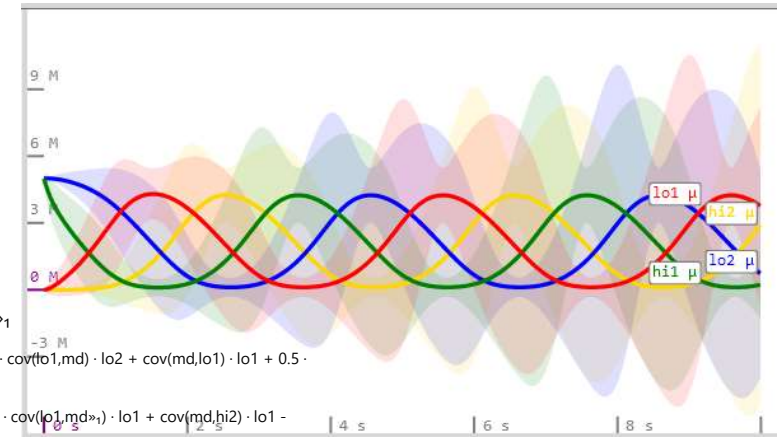


<= Demo: LotkaVolterra

Stochastic (LNA) simulation

- For *all* programs (any CRN, any Protocol)

2AM Oscillator



$\partial lo1 = - hi1 \cdot lo1 - 0.5 \cdot hi2 \cdot lo1 + lo1 \cdot md + 0.5 \cdot lo2 \cdot md$
$\partial hi2 = -0.5 \cdot hi1 \cdot hi2 - hi2 \cdot lo2 + hi2 \cdot md»_1 + 0.5 \cdot lo1 \cdot md»_1$
$\partial lo2 = 0.5 \cdot hi1 \cdot md»_1 - hi2 \cdot lo2 - 0.5 \cdot lo1 \cdot lo2 + lo2 \cdot md»_1$
$\partial hi1 = - hi1 \cdot lo1 - 0.5 \cdot hi1 \cdot lo2 + hi1 \cdot md + 0.5 \cdot hi2 \cdot md$
$\partial md = 2 \cdot hi1 \cdot lo1 + 0.5 \cdot hi1 \cdot lo2 + 0.5 \cdot hi2 \cdot lo1 - hi1 \cdot md - 0.5 \cdot hi2 \cdot md - lo1 \cdot md - 0.5 \cdot lo2 \cdot md$
$\partial md»_1 = 0.5 \cdot hi1 \cdot hi2 - 0.5 \cdot hi1 \cdot md»_1 + 2 \cdot hi2 \cdot lo2 + 0.5 \cdot lo1 \cdot lo2 - hi2 \cdot md»_1 - 0.5 \cdot lo1 \cdot md»_1 - lo2 \cdot md»_1$

$\partial var(lo1) = - cov(hi1,lo1) \cdot lo1 - 0.5 \cdot cov(hi2,lo1) \cdot lo1 - cov(lo1,hi1) \cdot lo1 - 0.5 \cdot cov(lo1,hi2) \cdot lo1 + cov(lo1,md) \cdot lo1 + hi1 \cdot lo1 + 0.5 \cdot hi2 \cdot lo1 + 0.5 \cdot cov(lo1,md) \cdot lo2 + cov(md,lo1) \cdot lo1 + 0.5 \cdot cov(md,lo1) \cdot lo2 + 0.5 \cdot cov(lo1,lo2) \cdot md + 0.5 \cdot cov(lo2,lo1) \cdot md + lo1 \cdot md + 0.5 \cdot lo2 \cdot md - 2 \cdot hi1 \cdot var(lo1) - hi2 \cdot var(lo1) + 2 \cdot md \cdot var(lo1)$

$\partial cov(lo1,hi2) = cov(lo1,md»_1) \cdot hi2 - 0.5 \cdot cov(lo1,hi1) \cdot hi2 - cov(hi1,hi2) \cdot lo1 - 1.5 \cdot cov(lo1,hi2) \cdot hi1 - 0.5 \cdot cov(lo1,hi2) \cdot hi2 - cov(lo1,lo2) \cdot hi2 + 0.5 \cdot cov(lo1,md»_1) \cdot lo1 + cov(md,hi2) \cdot lo1 - cov(lo1,hi2) \cdot lo2 + 0.5 \cdot cov(md,hi2) \cdot lo2 + cov(lo1,hi2) \cdot md + 0.5 \cdot cov(lo2,hi2) \cdot md + cov(lo1,hi2) \cdot md»_1 - 0.5 \cdot lo1 \cdot var(hi2) + 0.5 \cdot md»_1 \cdot var(lo1)$

$\partial cov(lo1,lo2) = 0.5 \cdot cov(lo1,md»_1) \cdot hi1 - cov(hi1,lo2) \cdot lo1 - 0.5 \cdot cov(hi2,lo2) \cdot lo1 + cov(lo1,md»_1) \cdot lo2 + cov(md,lo2) \cdot lo1 + 0.5 \cdot cov(md,lo2) \cdot lo2 + 0.5 \cdot cov(lo1,hi1) \cdot md»_1 - 0.5 \cdot cov(lo1,lo2) \cdot lo1 - cov(lo1,hi2) \cdot lo2 - cov(lo1,lo2) \cdot hi1 - 1.5 \cdot cov(lo1,lo2) \cdot hi2 + cov(lo1,lo2) \cdot md + cov(lo1,lo2) \cdot md»_1 - 0.5 \cdot lo2 \cdot var(lo1) + 0.5 \cdot md \cdot var(lo2)$

$\partial cov(lo1,hi1) = cov(lo1,md) \cdot hi1 + 0.5 \cdot cov(lo1,md) \cdot hi2 - cov(lo1,hi1) \cdot lo1 + cov(md,hi1) \cdot lo1 - 0.5 \cdot cov(lo1,hi1) \cdot lo2 - 0.5 \cdot cov(lo1,lo2) \cdot hi1 - 0.5 \cdot cov(hi2,hi1) \cdot lo1 - cov(lo1,hi1) \cdot hi1 - 0.5 \cdot cov(lo1,hi1) \cdot hi2 + 0.5 \cdot cov(md,hi1) \cdot lo2 + 2 \cdot cov(lo1,hi1) \cdot md + 0.5 \cdot cov(lo1,hi2) \cdot md + 0.5 \cdot cov(lo2,hi1) \cdot md - lo1 \cdot var(hi1) - hi1 \cdot var(lo1)$

$\partial cov(lo1,md) = 2 \cdot cov(lo1,hi1) \cdot lo1 - cov(hi1,md) \cdot lo1 - cov(lo1,md) \cdot lo1 - hi1 \cdot lo1 - 0.5 \cdot hi2 \cdot lo1 + 0.5 \cdot cov(lo1,hi1) \cdot lo2 - 0.5 \cdot cov(lo1,md) \cdot lo2 - cov(lo1,hi1) \cdot md + 0.5 \cdot cov(lo1,lo2) \cdot hi1 - 0.5 \cdot cov(hi2,md) \cdot lo1 + 0.5 \cdot cov(lo1,hi2) \cdot lo1 - 0.5 \cdot cov(lo1,hi2) \cdot md - 0.5 \cdot cov(lo1,lo2) \cdot md - 2 \cdot cov(lo1,md) \cdot hi1 - cov(lo1,md) \cdot hi2 + cov(lo1,md) \cdot md + 0.5 \cdot cov(lo2,md) \cdot md - lo1 \cdot md - 0.5 \cdot lo2 \cdot md + 2 \cdot hi1 \cdot var(lo1) + 0.5 \cdot hi2 \cdot var(lo1) + lo1 \cdot var(md) + 0.5 \cdot lo2 \cdot var(md) - md \cdot var(lo1)$

$\partial cov(lo1,md»_1) = 0.5 \cdot cov(lo1,hi1) \cdot hi2 - cov(hi1,md»_1) \cdot lo1 - 0.5 \cdot cov(lo1,md»_1) \cdot lo1 - cov(lo1,md»_1) \cdot lo2 + cov(md,md»_1) \cdot lo1 + 0.5 \cdot cov(md,md»_1) \cdot lo2 - 0.5 \cdot cov(lo1,hi1) \cdot md»_1 + 0.5 \cdot cov(lo1,hi2) \cdot hi1 + 2 \cdot cov(lo1,lo2) \cdot hi2 - 0.5 \cdot cov(hi2,md»_1) \cdot lo1 + 0.5 \cdot cov(lo1,lo2) \cdot lo1 + 2 \cdot cov(lo1,hi2) \cdot lo2 - cov(lo1,lo2) \cdot md»_1 + 0.5 \cdot cov(lo2,md»_1) \cdot md - cov(lo1,hi2) \cdot md»_1 - 1.5 \cdot cov(lo1,md»_1) \cdot hi1 - 1.5 \cdot cov(lo1,md»_1) \cdot hi2 + cov(lo1,md»_1) \cdot md + 0.5 \cdot lo2 \cdot var(lo1) - 0.5 \cdot md»_1 \cdot var(lo1)$

...

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***

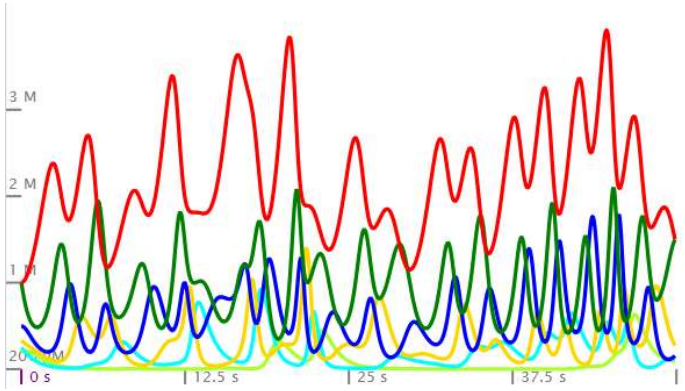
- Modern abstractions

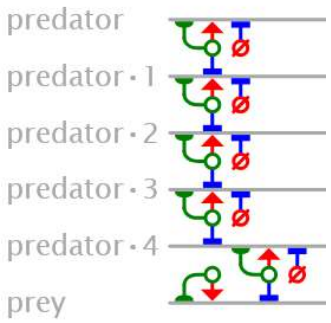
  Functional: programs take *data* as parameters and produce *data* as results
  Monadic: programs also produce *effects* (*species*, *reactions*, *liquid handling*)
  Nominal: *lexically scoped* chemical species (species are not "strings")

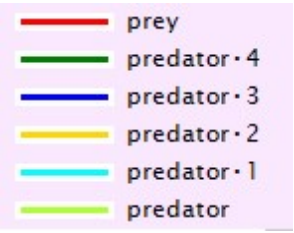
Ex: Predatorial

```
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 -> ∅ // 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 = mol/L.
- The default implicit sample is called the **vessel** {1 mL, 20 C}

```
species {c} // a species for multiple samples

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

<= Demo: MixAndSplit

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

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.

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

Ex: Serial Dilution (recursive protocol)

```
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)
```

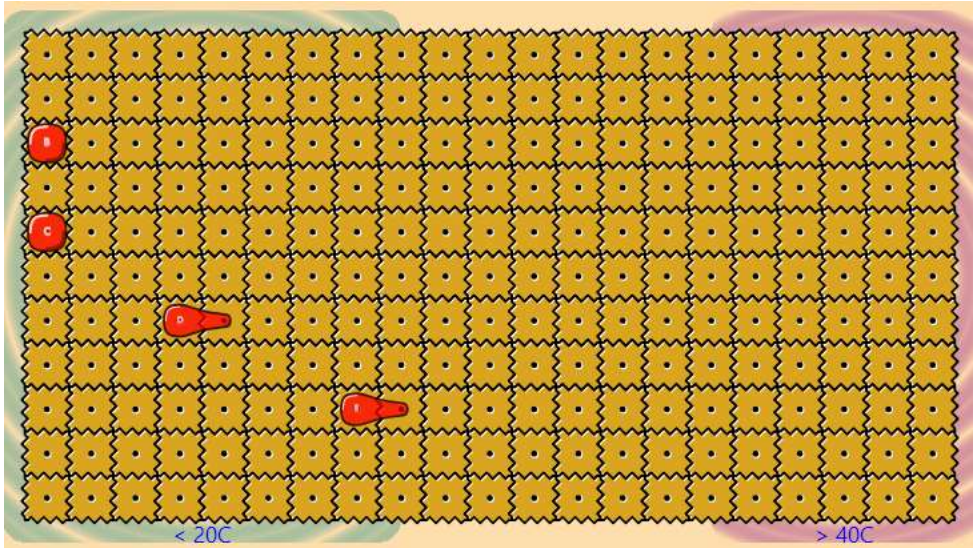
Prepare a series of increasingly diluted solutions and apply a network f to each (f can add species and reactions to the solutions)

```
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}
```

Digital 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
```



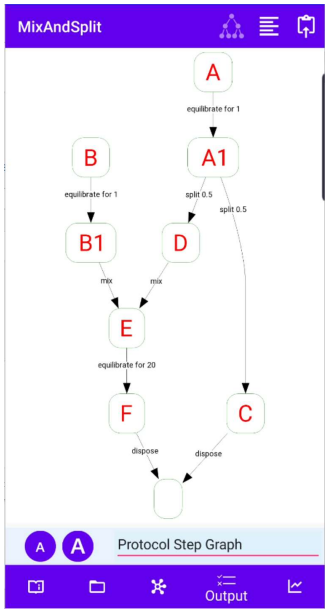
<= Demo: MixAndSplit

Extracting the Model and the Protocol

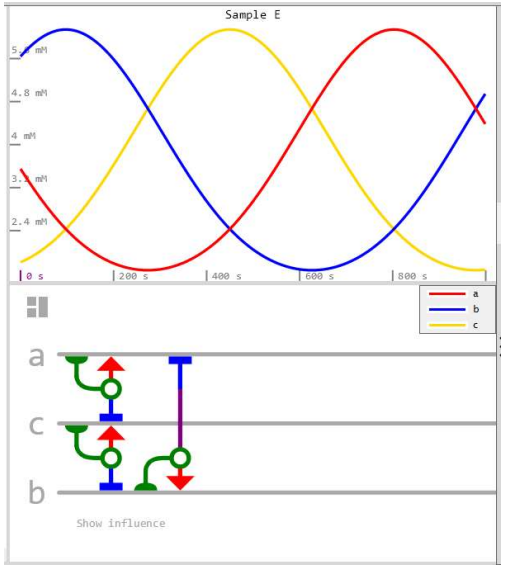
From the script

```
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)



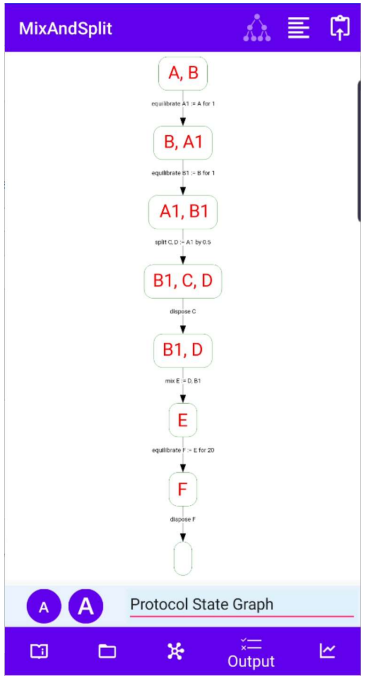

```
STATE_5
sample E {1.5mL, 293.2K} {
   a = 354.5mM
   c = 178mM
   b = 0.5674M
   consumed
   a + c -> a + a
   b + c -> c + c
   a + b -> b + b
}

KINETICS for STATE_5 (sample E) for 20 time units:
∂a = a * c - a * b
∂c = c * b - a * c
∂b = a * b - c * b
```

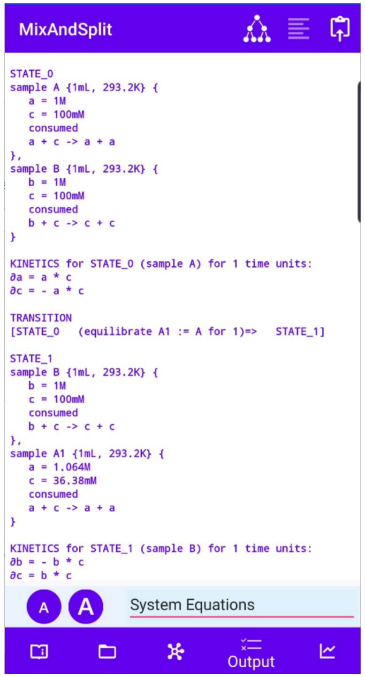
Extracting the Hybrid Transition System

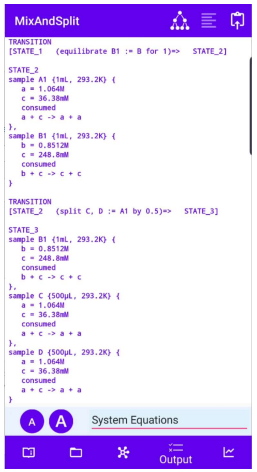
From the script

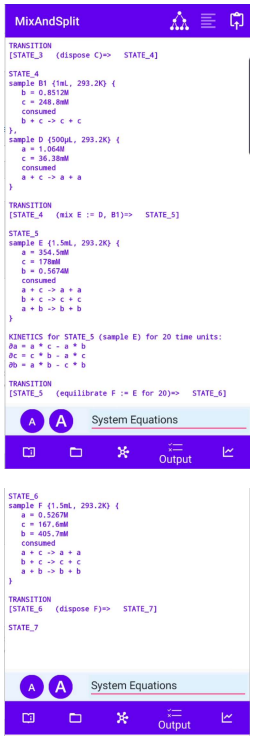
```
species {c}

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)









Kaemika: Extra features

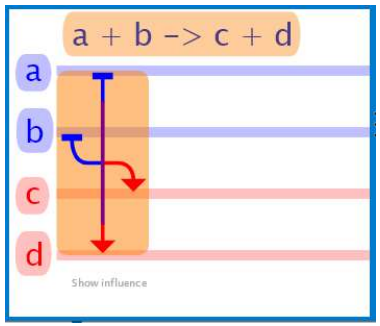
Extra features

- General kinetic rates
  - Fractions, rational powers, exponentials, trigonometry. E.g., x -> y {{ 1/x }}
  - 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 ∂x = s · y - s · x
  - This is translated to the reaction ∅ -> x {{s · y - s · x}} using general kinetic rates
- Timeflows (trajectories as first-class values)
  - Programmable plot reports (e.g., var(2 · a - 3 · 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

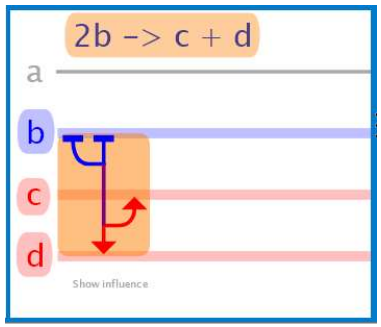
Reaction scores (graphical representation of reaction networks)

Horizonal lines: *species*. Vertical stripes: *reactions*. Blue: reagents. Red: products. Green: catalysts.

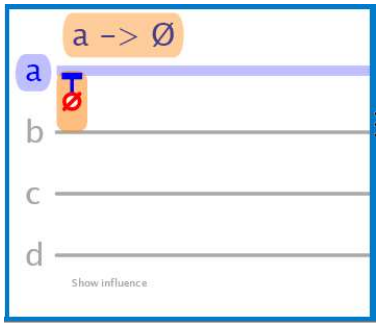
Reactants and products



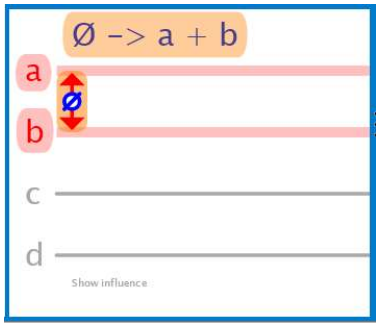
Repeated species



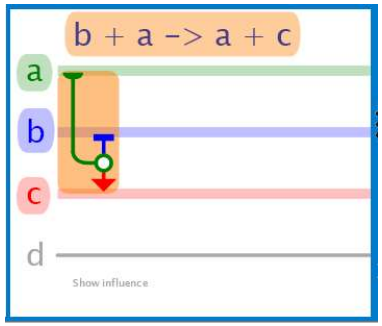
Reactants but no products



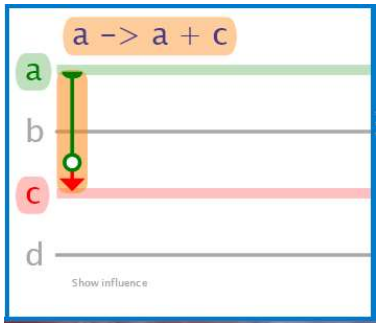
Products but no reactants



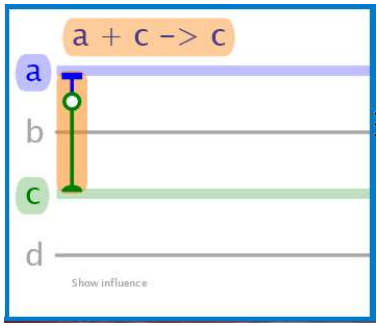
Catalyst



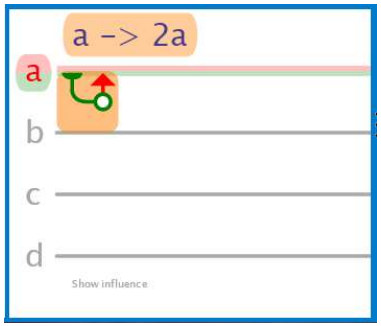
Catalyst but no reactants



Catalyst but no products



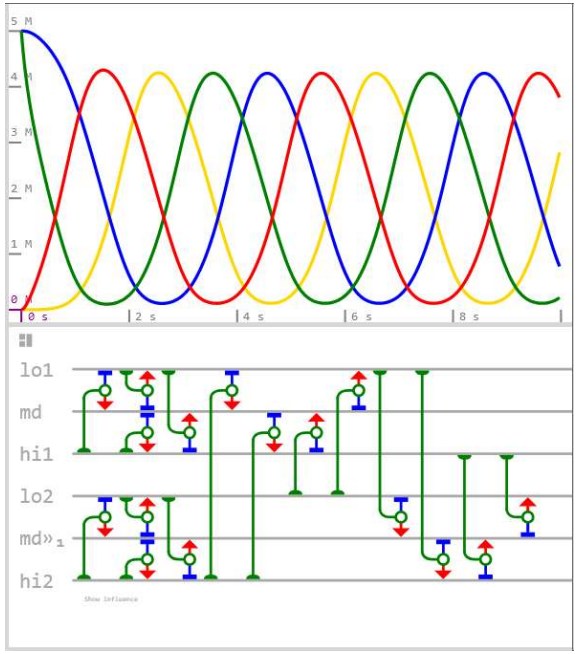
Autocatalyst

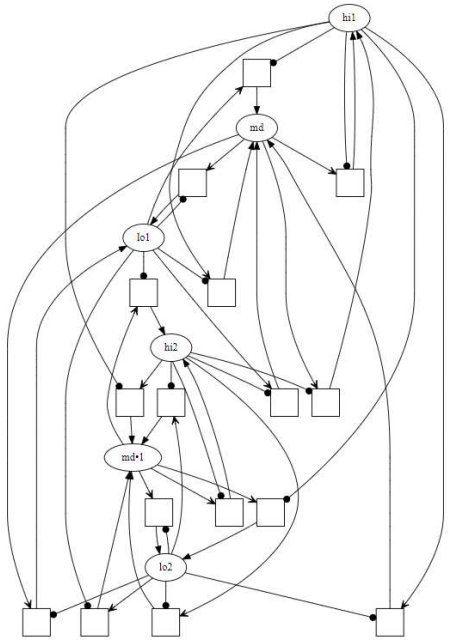


Reaction Scores vs. Reaction Graphs

- 2AM Oscillator

```
hi1 + md -> 2hi1
lo1 + hi1 -> lo1 + md
lo1 + md -> 2lo1
hi2 + lo1 -> hi2 + md {0.5}
hi2 + md -> hi2 + hi1 {0.5}
lo2 + hi1 -> lo2 + md {0.5}
lo2 + md -> lo2 + lo1 {0.5}
hi2 + lo2 -> hi2 + md»₁
hi2 + md»₁ -> 2hi2
lo2 + hi2 -> lo2 + md»₁
lo2 + md»₁ -> 2lo2
lo1 + lo2 -> lo1 + md»₁ {0.5}
lo1 + md»₁ -> lo1 + hi2 {0.5}
hi1 + hi2 -> hi1 + md»₁ {0.5}
hi1 + md»₁ -> hi1 + lo2 {0.5}
```



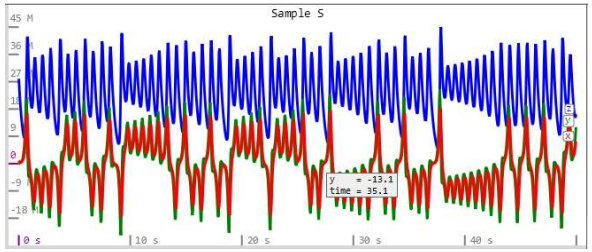


GraphViz

<= Demo: 2AM Oscillator

Mass Action Compiler

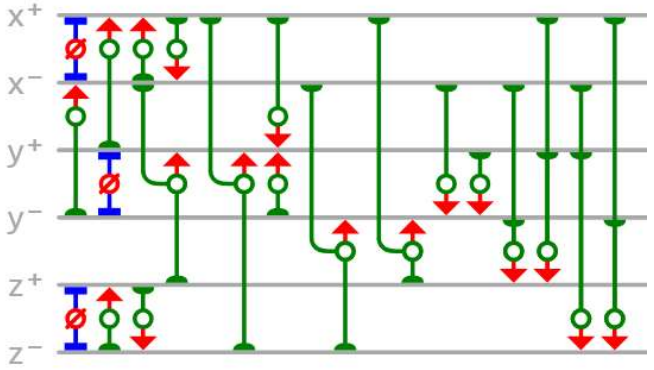
- Lorenz chaotic attractor



not mass action

$\partial x = s \cdot y - s \cdot x$
$\partial y = r \cdot x - x \cdot z - y$
$\partial z = x \cdot y - b \cdot z$

$s = 10$
$b = 8/3$
$r = 28$
$x_0 = 1$
$y_0 = 0$
$z_0 = 28$



$x^+ + x^- \rightarrow \emptyset$
$y^+ \rightarrow y^+ + x^+$ {10}
$x^- \rightarrow x^- + x^+$ {10}
$y^- \rightarrow y^- + x^-$ {10}
$x^+ \rightarrow x^+ + x^-$ {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)

Advanced Scripting

```
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)]
}
```

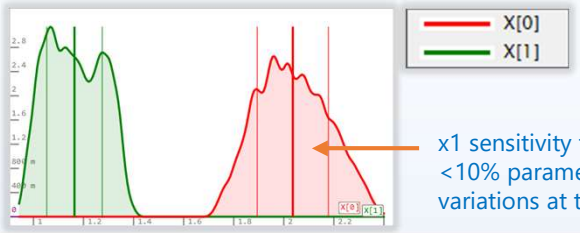
<- A function f to run one simulation (ri are the input parameters to be perturbed)
<- define *D* yield *E* returns the value of *E* after executing the statements *D*
<- Make a new sample S to contain species and reactions for simulation
<- Lotka-Volterra prey species x1 (initial conditions could be a parameter as well)
<- Lotka-Volterra predator species x2
<- Prey reproduces, with perturbed rate r1
<- Predator eats prey, with perturbed rate r2
<- Predator dies, with perturbed rate r3
<- Simulate the system up to time 2.5 (first peak of the oscillation)
<- Return the output concentrations of x1,x2 from S at time 2.5 as pairs

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

<- Create a bivariate random variable X over uniform[0..1) sample spaces w(i)
<- producing random instances $f(1+e_1, 1+e_2, 1+e_3) = [x1,x2]_{e1,e2,e3,t=2.5}$
with e_1, e_2, e_3 being 10% independent perturbations of the parameters

```
draw 2000 from X
```

<- Produce a density plot of 2000 instances drawn from X
i.e. a plot of the distributions of X[0]=x1 and X[1]=x2 at time 2.5
vertical bars are mean and standard deviation



x1 sensitivity to random <10% parameter variations at time 2.5

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

Conclusions

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
http://lucacardelli.name/Papers/Kaemika%20User%20Manual.pdf

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 (general obfuscator)
- App store certificates
- Dark mode support