Stochastic Strand Algebra

Stochastic Strand Algebra

- Unbounded populations P^* are meaningless because one cannot compute their stochastic impact. Hence stochastic strand algebra $\mathcal{P}^r$ drops P^*:

$$P \quad ::= \quad x \;\vdots\; [x_1,..,x_n].[x'_1,..,x'_m] \;\vdots\; 0 \;\vdots\; P|P \qquad n\geq 1,\ m\geq 0$$

- Instead of unbounded populations P^* should think of populations of size k, P^k, which of course we already have from iterated parallel composition.

- Moreover, each gate with n inputs has a fixed rate g_n (collapsing all gate parameters into one).

Propensity

- Given a global state P what is the *propensity* ('speed') of each possible next reaction? It is:

  $(P \text{ choose } (x_1 \mid \dots \mid x_n \mid [x_1,\dots,x_n].[y_1,\dots,y_m])) \times g_n$

- That is, it is the number of ways of choosing from P an n-ary gate and its n inputs, multiplied by the gate rate g_n.

  - If $P = x^n | y^m | ([x,y].z)^p$ with $x \neq y$, the propensity of the next $x|y|[x,y].z \rightarrow z$ reaction is $n \times m \times p \times g_2$.

  - If $P = x^n | ([x,x].z)^p$, the propensity of the next $x|x|[x,x].z \rightarrow z$ reaction is (n choose 2)$\times p \times g_2$ = $n \times (n-1)/2 \times p \times g_2$.

  - N.B. we have the factor $r = n \times (n-1)/2$ here. For large n we have $n \times (n-1) \sim n^2$ in terms of reaction order. In terms of reaction rate, note that in chemistry the (measured, macroscopic) *mass action rate* k = 2r is always twice the underlying (actual, microscopic) *stochastic rate* k for reactions of the form $x+x \rightarrow \dots$. Hence we have an overall *macroscopic* reaction speed of $\sim(k \times g_2) \times n^2 \times p$ which, further converting the molecule counts to concentrations in a volume, gives us back the law of mass action.

- For (unary) curried gates, the propensity is:

  $(P \text{ choose } (x_0 \mid x_0.[x_1,\dots,x_n].[y_1,\dots,y_m])) \times g_1$

Global Transition

- A *global transition* $P \rightarrow^{r} P'$ of a global state P to a next global state P' with propensity r is then defined as:

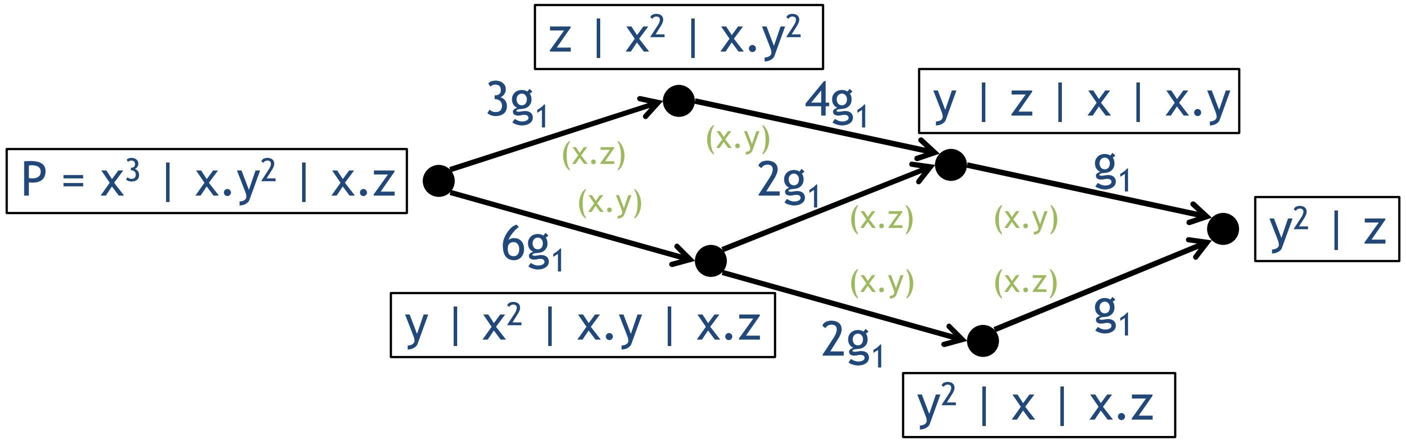
$$P \quad \rightarrow^{(P \text{ choose } (x_1 \mid \ldots \mid x_n \mid [x_1,\ldots,x_n].[y_1,\ldots,y_m])) \times g_n}$$

$$(P \setminus (x_1 \mid \ldots \mid x_n \mid [x_1,\ldots,x_n].[y_1,\ldots,y_m])) \mid y_1 \mid \ldots \mid y_m$$

where $\setminus$ is multiset difference. (Similarly for curried gates.)

Continuous Time Markov Chain

- From the global transitions of a global state P, and of all its successive states, we can build a CTMC for any P.



- From this we can extract the standard matrix representation of CTMCs.
- We can also extract a DTMC (state transition probabilities), but the CMTC has more information: the expected holding time in each state (1/Σexit-propensities), i.e. the *kinetics* of the system.

- This is the semantics of Stochastic Strand Algebra.

Buffered Populations

- We have given up P*, so how do we do recursion?
- Consider instead populations of *constant size* k, $P^{=k}$.
- Take for example $P = x.y$; we have that

$$x \mid P^{k} \rightarrow^{k\times g_1} y \mid P^{k-1}$$ (a global transition)

- We want to find a system $P^{=k}$ such that:

$$x \mid P^{=k} \rightarrow^{k\times g_1} y \mid P^{=k}$$

  - it should evolve at rate k ($\times g_1$) like P^{k}
  - but it should not get depleted in the process

The Buffer Population

- These are definable (to arbitrary approximation) by using a bigger *buffer population* to replenish the (pseudo-)constant population.

- For P = x.y (for example) define:

$$P^{=k} \stackrel{\text{def}}{=} (x.[y.Z])^k \mid (Z.x.[y.Z])^{BIG}$$

for a fresh Z

e.g. BIG = 10000×k

Here BIG is an example of a *large-enough* buffer: it ensures that reactions on Z are much faster than reactions on x by mass action. We can make Z reactions arbitrarily fast, without affecting x reactions.

Z.x.[y,Z] is a curried gates. The construction can be done also without curried gates, but then it requires balancing the rates of gates with different numbers of inputs.

Buffered Populations in Action

- Now we provide an input:

$$x \mid P^{=k} \;=\; x \mid (x.[y.Z])^{k} \mid (Z.x.[y.Z])^{BIG}$$
$$\rightarrow^{k\times g_1}\; y \mid Z \mid (x.[y.Z])^{k-1} \mid (Z.x.[y.Z])^{BIG}$$
$$\rightarrow^{BIG\times g_1}\; y \mid x.[y.Z] \mid (x.[y.Z])^{k-1} \mid (Z.x.[y.Z])^{BIG-1}$$
$$=\; y \mid (x.[y.Z])^{k} \mid (Z.x.[y.Z])^{BIG-1}$$

with $\rightarrow^{k\times g_1}\rightarrow^{BIG\times g_1} \quad \sim \quad \rightarrow^{k\times g_1}$

and $(x.[y.Z])^{k} \mid (Z.x.[y.Z])^{BIG-1} \quad \sim \quad P^{=k}$

- Hence, the propensities in $P^{=k}$ are (approximately) the same as in P^{k}:

$$x \mid P^{=k} \;\rightarrow^{\sim k\times g_1}\; y \mid \sim P^{=k}$$

but $P^{=k}$ does not get depleted (approximately).

Properties of Buffered Populations

- Hence, $P^{=k}$ is the stochastic equivalent of P^*:

$$P^{=k} \sim P \mid P^{=k}$$

  - We can make the approximation as good as we want by increasing the buffer.
  - We can 'top up' the buffers periodically, *without affecting the rest of the system* (only the arbitrarily fast reaction on Z).
  - By topping-up the buffers, we can support *arbitrarily long computations* and *recursion*.
  - Afterthought: it seems it may be true that $(P^{=k})^{=k} \sim P^{=k}$; I have not checked the details. This would be nice, but on the other hand it would mean that there is nothing to gain in building buffers of buffers. Maybe there are also different encodings of $P^{=k}$ with different properties.

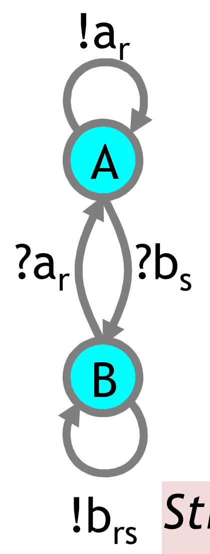
Chemistry (FSRN) to Strand Algebra

- With the help of $P^{=k}$, we can now encode stochastic formalisms, like Finite Stochastic Reaction Networks (finite sets of chemical reactions with stochastic rates) in Strand Algebra (and DNA).
  - [Soloveichik et al.] **DNA as a Universal Substrate for Chemical Kinetics:** how to implement an *arbitrary set of chemical reactions* by engineering chemical species (as DNA strands) that obey the reactions.

- For a stochastic reaction rate r of an n-ary reaction we use a constant-size population of size r/g_n (assume $r>>g_n$; otherwise scale up the rates to obtain large-enough populations):
  1) $A \to^r B_1 + \ldots + B_m \quad \Rightarrow \quad (A.[B_1,\ldots,B_m])^{=r/g_1}$
  2) $A_1+A_2 \to^r B_1 + \ldots + B_m \quad \Rightarrow \quad ([A_1,A_2].[B_1,\ldots,B_m])^{=r/g_2}$
  3) $A+A \to^r B_1 + \ldots + B_m \quad \Rightarrow \quad ([A,A].[B_1,\ldots,B_m])^{=r/g_2}$

- The propensities match:
  1) $P = A^n$: $(P \text{ choose } A)\times r = n\times r = (P \text{ choose } A)\times r/g_1\times g_1$
  2) $P = A_1^n+A_2^m$: $(P \text{ choose } A_1,A_2)\times r = n\times m\times r = (P \text{ choose } A_1,A_2)\times r/g_2\times g_2$
  3) $P = A^n$: $(P \text{ choose } A,A)\times r = n\times(n-1)/2\times r = (P \text{ choose } A,A)\times r/g_2\times g_2$

Interacting Automata to Strand Algebra



Groupies

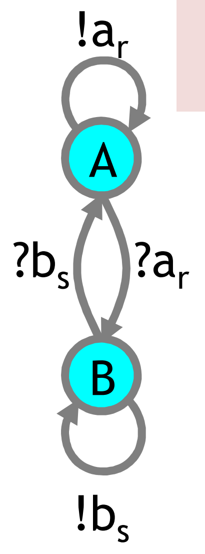
$A = !a_r.A \oplus ?b_s.B$
$B = !b_s.B \oplus ?a_r.A$

Strand(Groupies)

on a_r: $([B,A].[A,A])^{=r/g_2}$ |
on b_s: $([A,B].[B,B])^{=s/g_2}$

Map each possible interaction to a Join

Strand(E) = *Parallel*(⟪...⟫ means multisets
⟪ $(X.[P])^{=r/g_1}$ *s.t.* $\exists$i. E.X.i = τ;P ⟫ $\cup$
⟪ $([X,Y].[P,Q])^{=r/g_2}$ *s.t.* $X \neq Y$ and $\exists$i,j,c. E.X.i = $?c_r$;P and E.Y.j = !c;Q ⟫ $\cup$
⟪ $([X,X].[P,Q])^{=2r/g_2}$ *s.t.* $\exists$i,j,c. E.X.i = $?c_r$;P and E.X.j = !c;Q ⟫)



Celebrities

$A = !a_r.A \oplus ?a_r.B$
$B = !b_s.B \oplus ?b_s.A$

Strand(Celebrities)

on a_r: $([A,A].[B,A])^{=2r/g_2}$ |
on b_s: $([B,B].[A,B])^{=2s/g_2}$

Global Transitions and Propensities Match

Groupies

$A = !a_r.A \oplus ?b_s.B$	(on a_r)	$A^n \| B^m \rightarrow^{n\times m\times r} A^{n+1} \| B^{m-1}$
$B = !b_s.B \oplus ?a_r.A$	(on b_s)	$A^n \| B^m \rightarrow^{n\times m\times s} A^{n-1} \| B^{m+1}$

Strand(Groupies)

$P = ([B,A].[A,A])^{=r/g_2} \;\|$	$A^n \| B^m \| P \rightarrow^{\sim n\times m\times r/g_2\times g_2} A^{n+1} \| B^{m-1} \| \sim P$
$([A,B].[B,B])^{=s/g_2}$	$A^n \| B^m \| P \rightarrow^{\sim n\times m\times s/g_2\times g_2} A^{n-1} \| B^{m+1} \| \sim P$

Celebrities

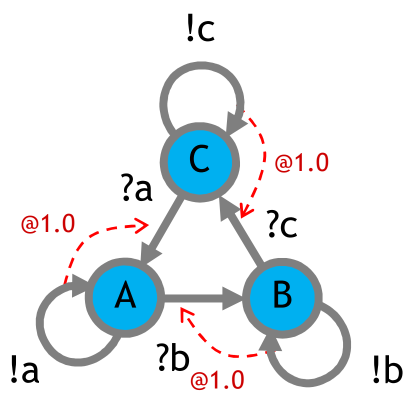
$A = !a_r.A \oplus ?a_r.B$	(on a_r)	$A^n \| B^m \rightarrow^{n\times(n-1)\times r} A^{n-1} \| B^{m+1}$
$B = !b_s.B \oplus ?b_s.A$	(on b_s)	$A^n \| B^m \rightarrow^{m\times(m-1)\times s} A^{n+1} \| B^{m-1}$

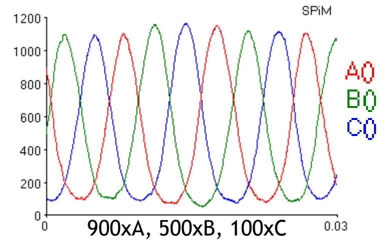
where there are two symmetric ?/! ways for A to interact with A, hence the propensity is 2×(n choose 2)×r = n×(n-1)×r.

Strand(Celebrities)

$P = ([A,A].[B,A])^{=2r/g_2} \;\|$	$A^n \| B^m \| P \rightarrow^{\sim n\times(n-1)/2\times 2r/g_2\times g_2} A^{n-1} \| B^{m+1} \| \sim P$
$([B,B].[A,B])^{=2s/g_2}$	$A^n \| B^m \| P \rightarrow^{\sim m\times(m-1)/2\times 2s/g_2\times g_2} A^{n+1} \| B^{m-1} \| \sim P$

Oscillator





```
directive sample 0.03 1000
directive plot A(); B(); C()

new a@1.0:chan new b@1.0:chan new
c@1.0:chan
let A() = do !a;A() or ?b; B()
and B() = do !b;B() or ?c; C()
and C() = do !c;C() or ?a; A()

run (900 of A() | 500 of B() | 100 of C())
```

All translations preserve stochastic semantics (to some arbitrary approximation).

$A = !a_{(s)};A \oplus ?b_{(s)};B$

$B = !b_{(s)};B \oplus ?c_{(s)};C$

$C = !c_{(s)};C \oplus ?a_{(s)};A$

$([A,B].[B,B])^{=s/g_2}$ |

$([B,C].[C,C])^{=s/g_2}$ |

$([C,A].[A,A])^{=s/g_2}$

$\longrightarrow$ DNA

$A+B \rightarrow^{s} B+B$

$B+C \rightarrow^{s} C+C$

$C+A \rightarrow^{s} A+A$

Summary

- Stochastic Strand Algebra can be obtained by restricting to finite populations and adding gate rates.
  - CTMC semantics.
  - Based on *propensities* of *global transitions*.

- A notion of buffered (constant) populations can be defined (to arbitrary approximation).

- That can be used to embed stochastic formalisms into Strand Algebra (and DNA), while preserving the stochastic semantics:
  - Stochastic Chemistry
  - (Stochastic) Interacting Automata

Nested Strand Algebra

Motivation

- Strand Algebra is pretty low-level: it is combinatorial (like assembly language).
- We want to demonstrate compilation of high(er) level languages to DNA.
- We consider an expression-based language, and we compile it to Strand Algebra, seen now as an intermediate (assembly) language.

Nested Expressions

- A sequence $x_1.x_2.x_3$ is *not* in the syntax of the combinatorial algebra.

- Still, it can be defined as:
  - $x_1.x_2.x_3 \;=\; x_1.x_0 \mid [x_0,x_2].x_3$
  - where x_0 can be chosen, e.g., as a fixed function of x_1,x_2

- The *nested* strand algebra generalizes this idea
  - Operations can be nested.
  - The main change is allowing arbitrary terms after a gate input.

Nested Strand Algebra n$\mathcal{P}$

$P ::= x \ \vdots \ [x_1,..,x_n].P \ \vdots \ 0 \ \vdots \ P|P \ \vdots \ P^* \qquad n \geq 1$

We now allow free cascading of operations: $x_1.[x_2,x_3].(x_4|x_5)$

And we also allow triggering whole populations: $x.P^*$

This syntax is a bit odd though: $x_1.x_2.x_3$ has x_2 is an input, while in $x_1.x_2$ has x_2 as an output. This gets confusing.

We are going to better distinguish inputs form output, further generalizing the nested algebra:

$P ::= x \ \vdots \ ?[x_1,..,x_n].P \ \vdots \ ![x_1,..,x_n].P \ \vdots \ 0 \ \vdots \ P|P \ \vdots \ P^* \qquad n \geq 1$

Embedding of $\mathcal{P}$ in n$\mathcal{P}$:

$[x_1|..|x_n].[y_1|..|y_m]$ becomes $?[x_1|..|x_n].![y_1|..|y_m].0$

Reduction Relation for $\mathrm{n}\mathcal{P}$

The structural congruence relation is exactly the same (we shall not bother with congruence under prefix).

The reduction relation changes only in the Gate rule:

$?[x_1,..,x_n].P \mid x_1 \mid .. \mid x_n \;\rightarrow\; P$	Input Gate
$![x_1,..,x_n].P \;\rightarrow\; x_1 \mid .. \mid x_n \mid P$	Output Gate

$P \rightarrow P' \quad \Rightarrow \quad P \mid P'' \rightarrow P' \mid P''$	Diffusion
$P \equiv P_1,\ P_1 \rightarrow P_2,\ P_2 \equiv P' \quad \Rightarrow \quad P \rightarrow P'$	Well Mixing

n$\mathcal{P}$ to $\mathcal{P}$ Unnest Algorithm

U(P) =	X | U(X,P)	for fresh X
U(X, x) =	X.x	
U(X, ?[x_1,..,x_n].P) =	[X,x_1,..,x_n].Y | U(Y,P)	for fresh Y
U(X, ![x_1,..,x_n].P) =	X.[x_1,..,x_n,Y]| U(Y,P)	for fresh Y
U(X, 0) =	X.[]	
U(X, P|P') =	X.[Y,Z] | U(Y,P) | U(Z,P')	for fresh Y,Z
U(X, P*) =	(X.[Y,X] | U(Y,P))*	for fresh Y

In the inner loop U(X,P), the signal X triggers the activation of the translation of P.

The 'freshness' side conditions are formalized by letting U(P) have an additional parameter that is an infinite sequence of fresh signals (distinct signals not occurring in P), which are consumed during the translation.

$\mathrm{n}\mathcal{P}$ to $\mathcal{P}$ Unnest Algorithm (more formal)

Let $\mathcal{X}$ be an infinite lists of distinct strands,
and $\mathfrak{F}$ be the set of such $\mathcal{X}$'s.

$\mathcal{X}_i$	isthe i-th strand in the list,
$\mathcal{X}_{\geq I}$	is the list starting at the i-th position of $\mathcal{X}$,
evn($\mathcal{X}$)	is the even elements of $\mathcal{X}$,
odd($\mathcal{X}$)	is the odd elements.

Let $\mathfrak{F}_P$ be the set of those $\mathcal{X} \in \mathfrak{F}$ that
do not contain any strand that occurs in P.

Let $P \in {}_n\mathcal{P}$ and $\mathcal{X} \in \mathfrak{F}_P$,
let X indicate strands in $\mathcal{X}$

$U(\mathcal{X}_0, P)_{\mathcal{X}_{\geq 1}}$ produces a gate that is triggered by $\mathcal{X}_0$.

$$U(P)_{\mathcal{X}} = \mathcal{X}_0 \mid U(\mathcal{X}_0,P)_{\mathcal{X}_{\geq 1}}$$

$$U(X, x)_{\mathcal{X}} = X.x$$
$$U(X, ?[x_1,..,x_n].P)_{\mathcal{X}} = [X,x_1,..,x_n].\mathcal{X}_0 \mid U(\mathcal{X}_0,P)_{\mathcal{X}_{\geq 1}}$$
$$U(X, ![x_1,..,x_n].P)_{\mathcal{X}} = X.[x_1,..,x_n,\mathcal{X}_0] \mid U(\mathcal{X}_0,P)_{\mathcal{X}_{\geq 1}}$$
$$U(X, 0) = X.[]$$
$$U(X, P'|P'') = X.[\mathcal{X}_0,\mathcal{X}_1] \mid U(\mathcal{X}_0,P')_{evn(\mathcal{X}_{\geq 2})} \mid U(\mathcal{X}_1,P'')_{odd(\mathcal{X}_{\geq 2})}$$
$$U(X, P^*) = (X.[\mathcal{X}_0,X] \mid U(\mathcal{X}_0,P)_{\mathcal{X}_{\geq 1}})^*$$

Solving Recursive Equations

In the nested algebra we can more easily solve recursive equations, because we can always "add one more prefix".

To solve the following equations:

$$X = ?x_1.X \mid !x_2.Y$$
$$Y = ?x_3.(X \mid Y)$$

write:

$$(?X.\ (?x_1.X \mid !x_2.Y))^* \mid$$
$$(?Y.\ ?x_3.(X \mid Y))^*$$

Triggering Populations

We can nest populations, and hence cause a single signal to release a whole population:

U(?x.P*) = X | [X,x].Z | (Z.[Y,Z] | U(Y,P))*

```
x | U(?x.P*)  →  Z | (Z.[Y,Z] | U(Y,P))*
          ≡  Z | Z.[Y,Z] | U(Y,P) | (Z.[Y,Z] | U(Y,P))*
          →  Y | U(Y,P) | Z | (Z.[Y,Z] | U(Y,P))*
          …
```

This causes a linear production of U(P); for an exponential production just change U(X, P*) = (X.[Y,X,X] | U(Y,P))*

U(P) =	X | U(X,P)	for fresh X
U(X, x) =	X.x	
U(X, ?[x_1,..,x_n].P) =	[X,x_1,..,x_n].Y | U(Y,P)	for fresh Y
U(X, ![x_1,..,x_n].P) =	X.[x_1,..,x_n,Y]| U(Y,P)	for fresh Y
U(X, 0) =	X.[]	
U(X, P|P') =	X.[Y,Z] | U(Y,P) | U(Z,P')	for fresh Y,Z
U(X, P*) =	(X.[Y,X] | U(Y,P))*	for fresh Y

Exercise 6: Wet Vending Machine Controller

A coffee vending machine controller, Vend, accepts two coins for coffee; an ok is given after the first coin and then either a second coin (for coffee) or an abort (for refund) is accepted:

```
Vend = ?coin. ![ok,mutex]. (Coffee | Refund)
Coffee = ?[mutex,coin]. !coffee. (Coffee | Vend)
Refund = ?[mutex,abort]. !refund. (Refund | Vend)
```

Exercise: compile that to the Combinatorial Strand Algebra; if you do it by the U(P) algorithm you can then heavily hand-optimize it.

Each Vend iteration spawns two branches, Coffee and Refund, waiting for either coin or abort. The branch not taken in the mutual exclusion is left behind; this could skew the system towards one population of branches. Therefore, when the Coffee branch is chosen and the system is reset to Vend, we also spawn another Coffee branch to dynamically balance the Refund branch that was not chosen; conversely for Refund.

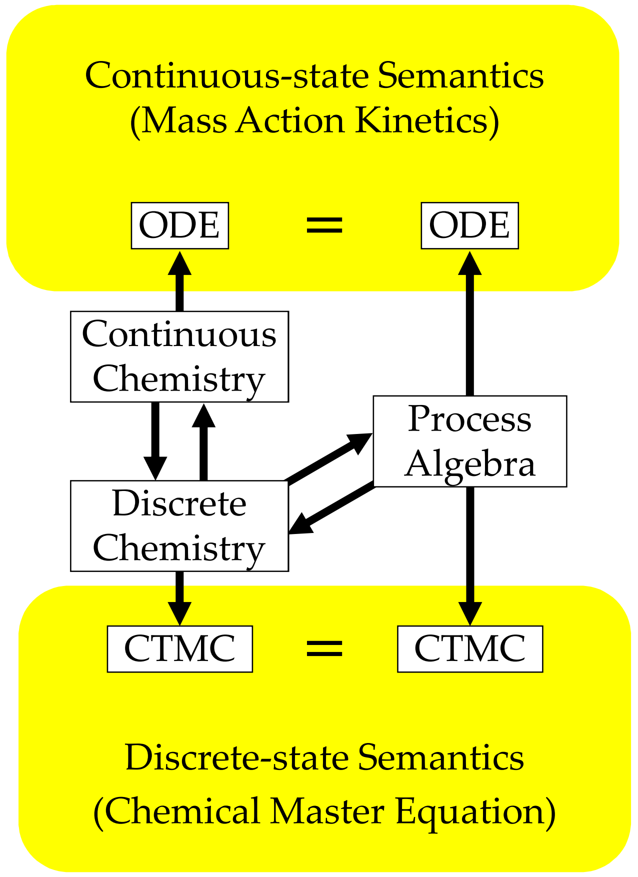
Standard questions can be asked: what happens if somebody inserts three coins very quickly? Or somebody presses refund twice? Etc.

Summary

- The Nested Strand Algebra is a ‘high level’ (expression based) language.
- It can be compiled to the basic Strand Algebra by a simple algorithm.
- It is expressive enough to program classical controllers fairly conveniently.
- And again, we can get DNA out of it.

Global Recap

Molecules as Automata

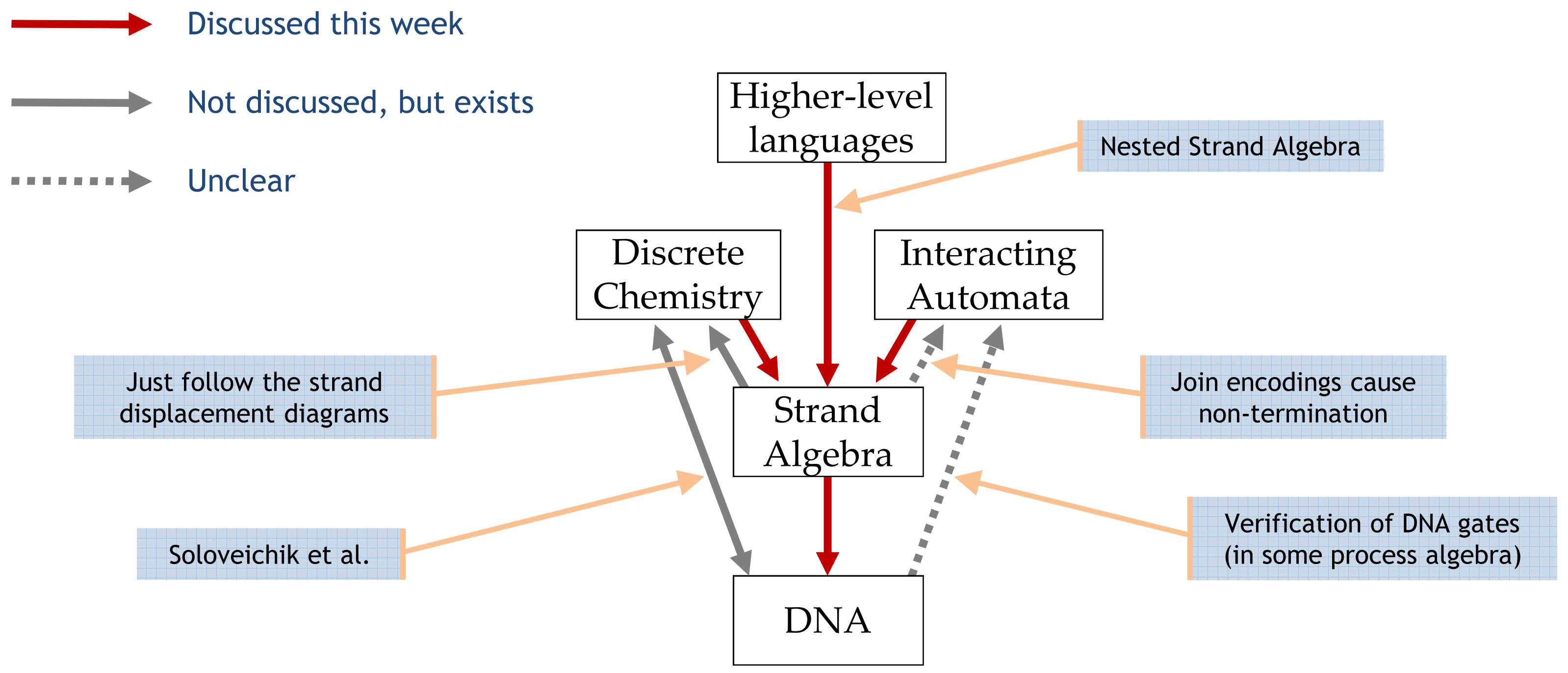


These diagrams commute via appropriate maps.

L. Cardelli: “On Process Rate Semantics” (TCS)

L. Cardelli: “A Process Algebra Master Equation” (QEST’07)

Automata as Molecules

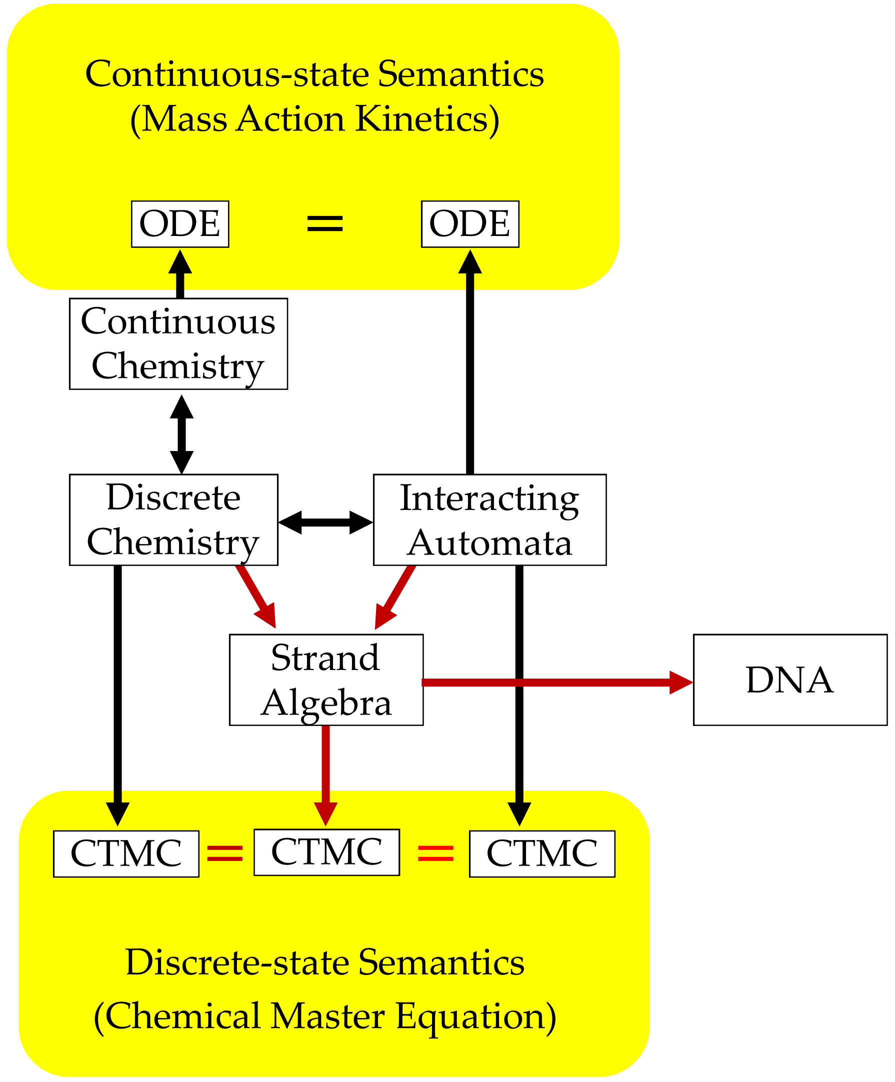


Discussed this week
Not discussed, but exists
Unclear
Higher-level languages
Nested Strand Algebra
Discrete Chemistry
Interacting Automata
Just follow the strand displacement diagrams
Strand Algebra
Join encodings cause non-termination
Verification of DNA gates (in some process algebra)
Soloveichik et al.
DNA

Verification of DNA gates:

prove that the DNA signal and a gate structures correctly implement the Strand Algebra reduction semantics in all possible contexts

Automata vs. Molecules



Open Problems/Questions

Implementing Choice in DNA

- This would allow compiling interacting automata to strand algebra *without* going through the n^2-expansion of the chemical translation.
- This is *hard*.
- Particularly because we don’t have a restriction operator (in strand algebra); otherwise there are some classical techniques to compile some π-calculus choice operators to parallel compositions.
- Note that there is no restriction operator in DNA, unless maybe one throws in the whole DNA transcription apparatus. Therefore, many encodings, particularly when replicated, tend to self-interfere.

Compiling Join to Choice

- I.e., compiling strand algebra to interacting automata.
- This *should* be just an exercise.
- Trivial if one admits divergence (by using the same “reversible binding” trick as in the DNA implementation of join).
- But how can one compile join to choice in a termination-preserving way?

Bib

For possible DNA implementations of the strand algebra see:

DNA as a Universal Substrate for Chemical Kinetics (Extended Abstract)
David Soloveichik, Georg Seelig, and Erik Winfree
http://www.dna.caltech.edu/Papers/DNA_for_CRNs_preprint_DNA14.pdf
(The primitives used here are x.y, x.[y,z], and [x,y].z).

and

Programming biomolecular self-assembly pathways. P. Yin, H.M.T. Choi, C.R. Calvert, N.A. Pierce Nature, 451:318-322, 2008.
(The primitives used here are x.y and x.[y,z], although “dissociation” is also used to great effect, and this is not easily expressible.)

and

Strand Algebras for DNA Computing. L. Cardelli. Proc. DNA Computing 15.