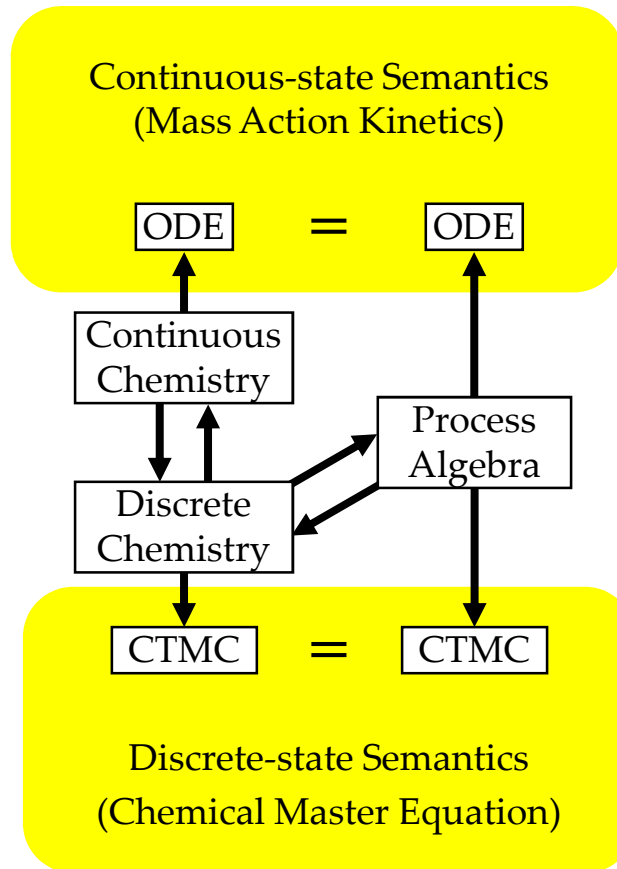


Strand Algebras for DNA Computing

Luca Cardelli
with Andrew Phillips

Microsoft Research
Cambridge UK

Edinburgh 2008-10-31
<http://LucaCardelli.name>

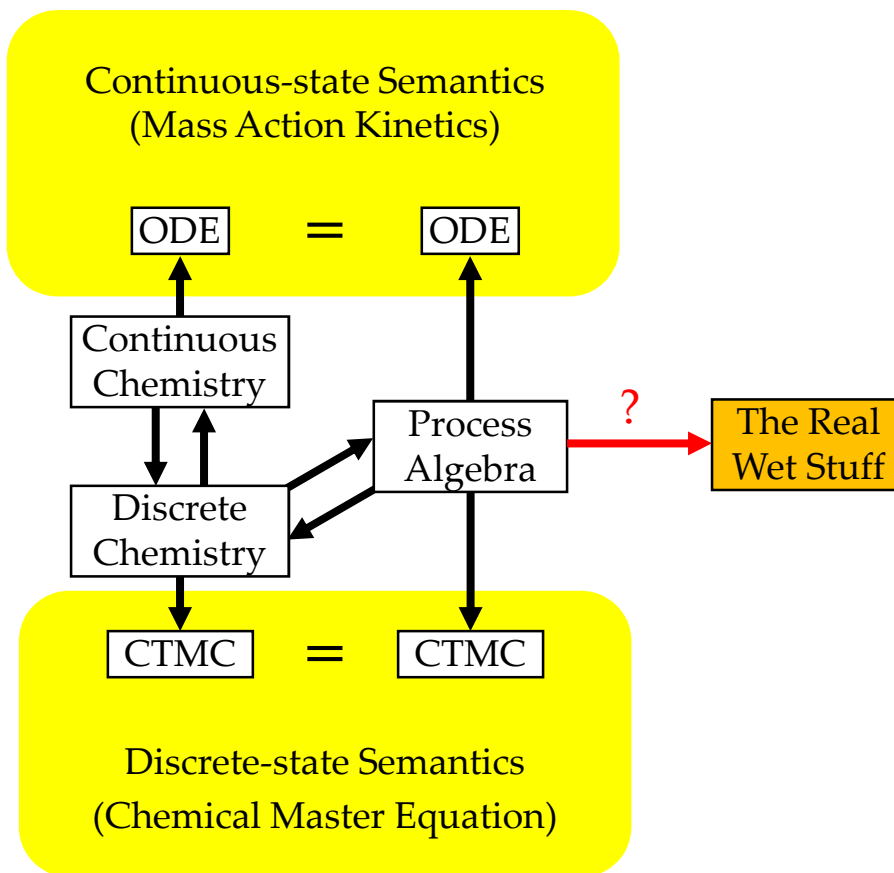


These diagrams commute via appropriate maps.

L. Cardelli: "On Process Rate Semantics" (TCS)

L. Cardelli: "A Process Algebra Master Equation" (QEST'07)

Motivation



How do we implement an arbitrary process?

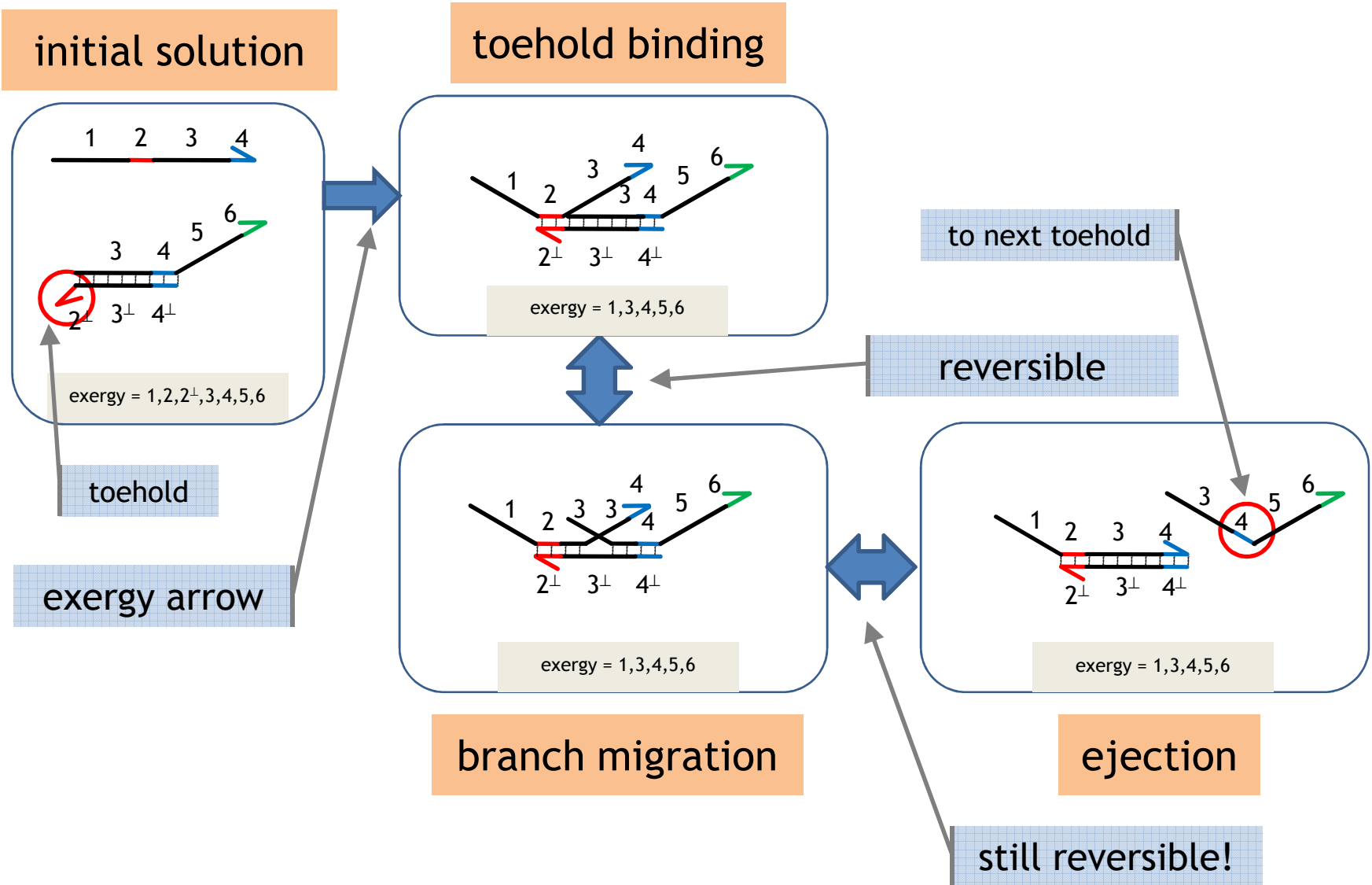
Chemistry does not necessarily help
(how do we then implement the chemical species?)

DNA Semantics

Naïve Thermodynamics

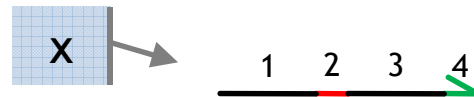
- Exergy (available energy) of the system:
 - Sum of the lengths of the unpaired strands.
- Exergy must ultimately decrease (second law)
 - Eventually exergy reaches zero, or the minimum possible.
- Local changes in exergy:
 - Downhill changes: “fast”
 - but locally not necessarily irreversible
 - Neutral changes
 - possibly down and then back up
 - reversible
 - Uphill changes
 - possible but temporary.
 - Bump changes: “slow”
 - they go up and lower down through exergy hills
 - possible, but less likely the higher the hill

Branch Migration



Signal Strand

D. Soloveichik, G. Seelig, E. Winfree. *DNA as a Universal Substrate for Chemical Kinetics*. Proc. DNA14.

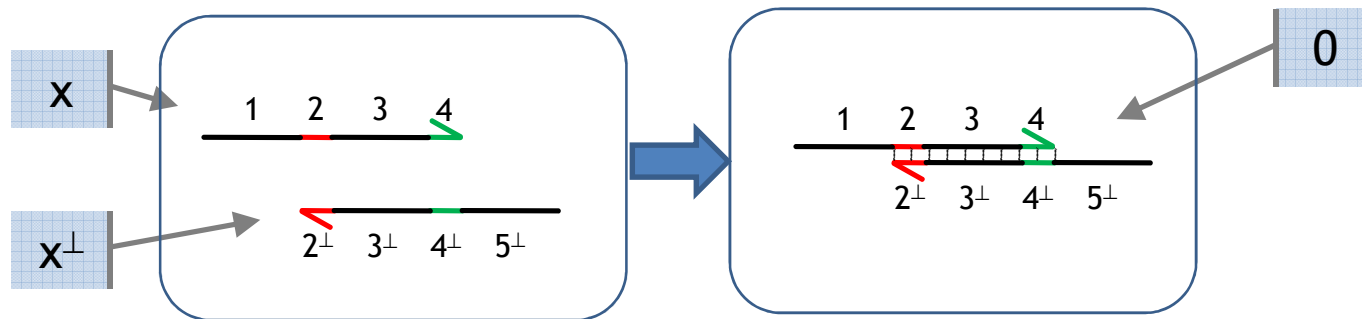


- 1 = history
- 2 = toehold
- 3 = body
- 4 = next toehold

The history is not part of signal recognition: strands with different histories should behave the same. Hence, x denotes an equivalence class of strand with different histories.

Strands x and x^\perp are supposed to hybridize (except for the history region). If $x \neq y$ then x and y^\perp are not supposed to hybridize (except for the history region).

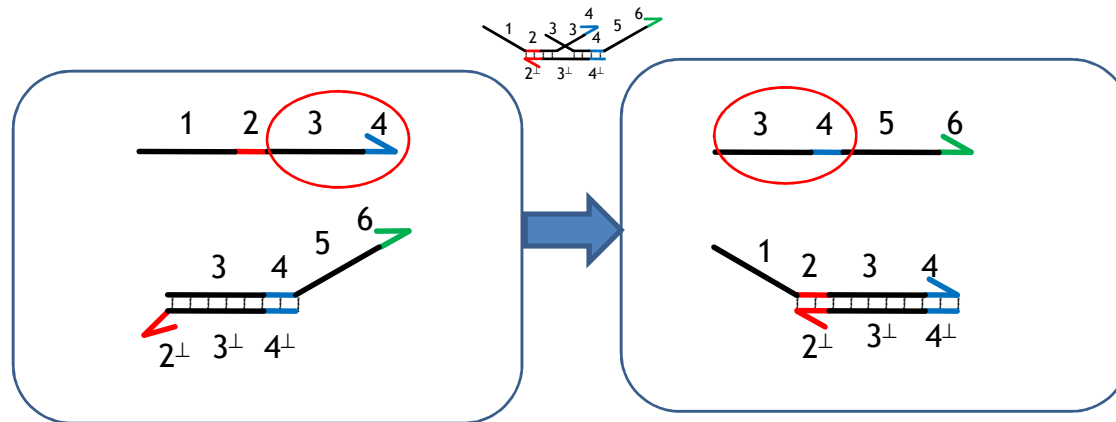
Hybridization




Watson-Crick complementarity:

$$(2,3,4)^\perp = (4^\perp,3^\perp,2^\perp)$$

Transducer



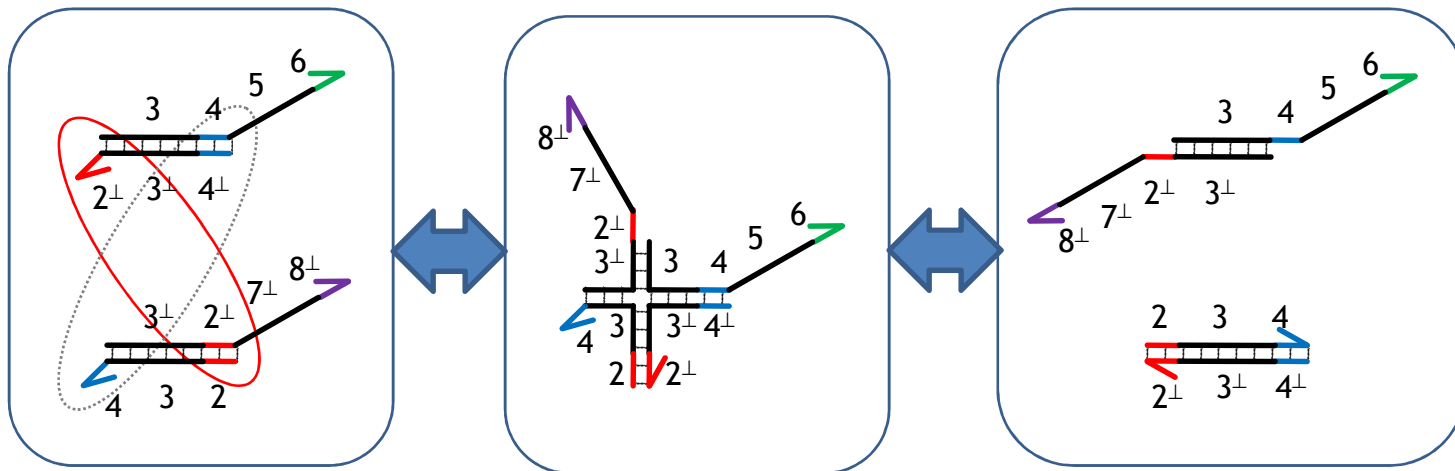
The output is not independent of the input. But two transducers can be composed to fix that (see Sequence).

Every reaction is reversible, but (at the right temperature) the equilibrium favors configurations with stronger (longer) hybridizations. Because of this, some reactions can be considered irreversible: this is the meaning of  . Instead, if two configurations have the same hybridization strength, they are fully reversible (possibly at a low rate).

DNA Spaghetti

These are two “complementary” transducers, one supposed to react to a (2,3,4) signal, the other to a $(2,3,4)^\perp = (4^\perp, 3^\perp, 2^\perp)$ signal.

But something is wrong: the transducers interact in absence of any signal!



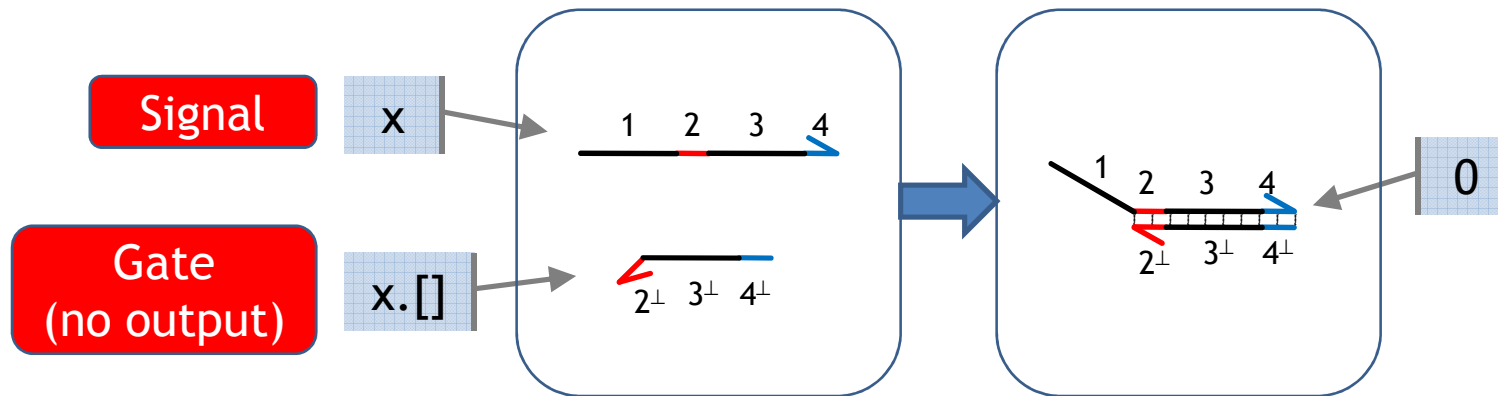
This reaction is neutral, **but the 4 and 2^\perp toeholds have been exposed**, and those are supposed to be exposed only in response to input signals. Hence, downstream, all hell breaks loose.

Imposing some discipline

- A precautionary (and possibly temporary) decision to avoid spaghetti: avoid complementary signals and complementary gates.
- Signals “x” are always positive
- Gates “x.y” are always negative
 - that is, the input “x” is implicitly perp’ed
 - and the output “y” is another positive signal
- This way, by the way, we give up Turing completeness, which is probably possible by using both positive and negative signals and gates.

Hybridization

$$x \mid x.[] \rightarrow 0$$

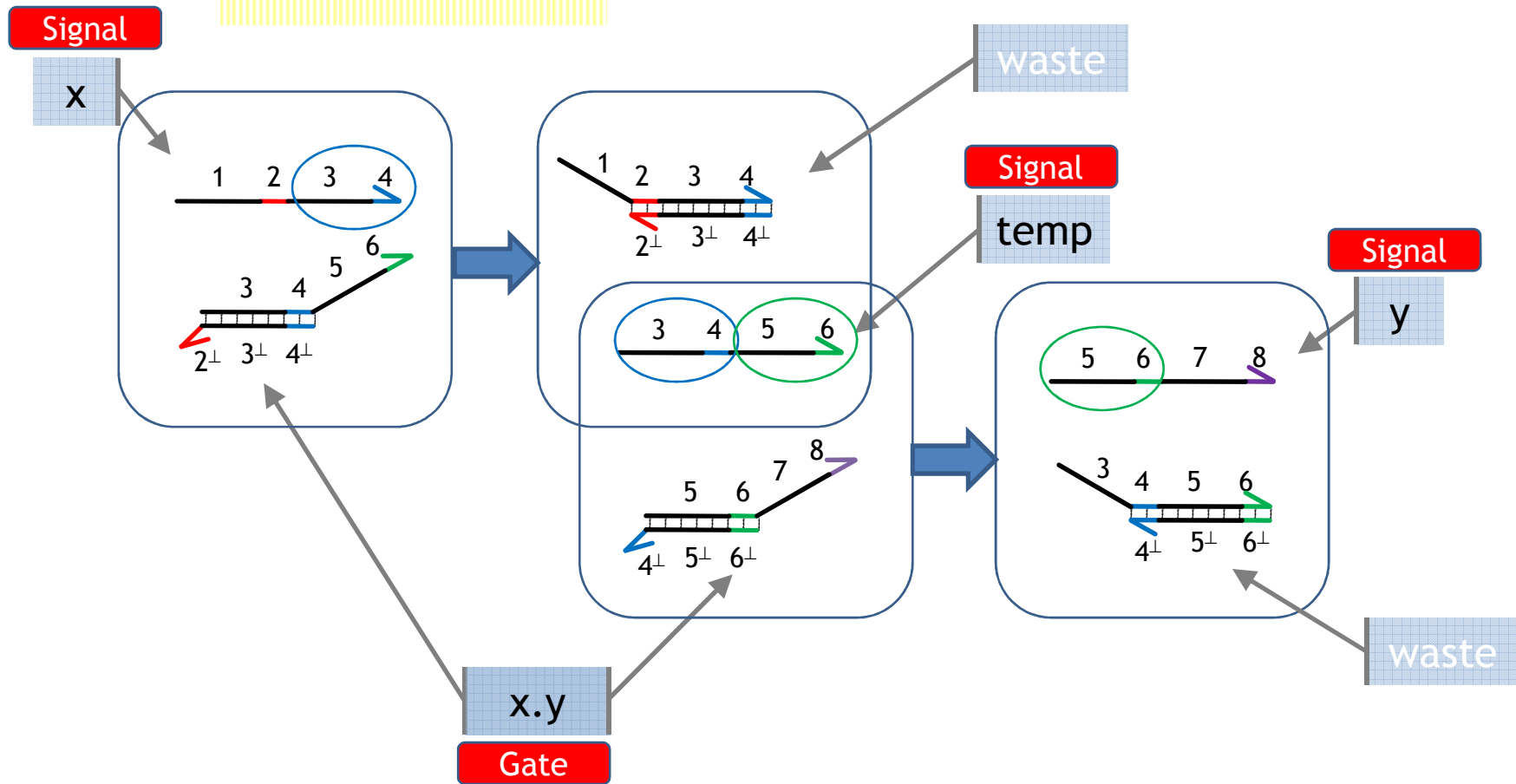


- A positive toehold indicates a signal.
- (A positive strand is a signal.)
- A negative toehold indicates a gate.
- (A negative strand is a gate with no output.)

Sequence

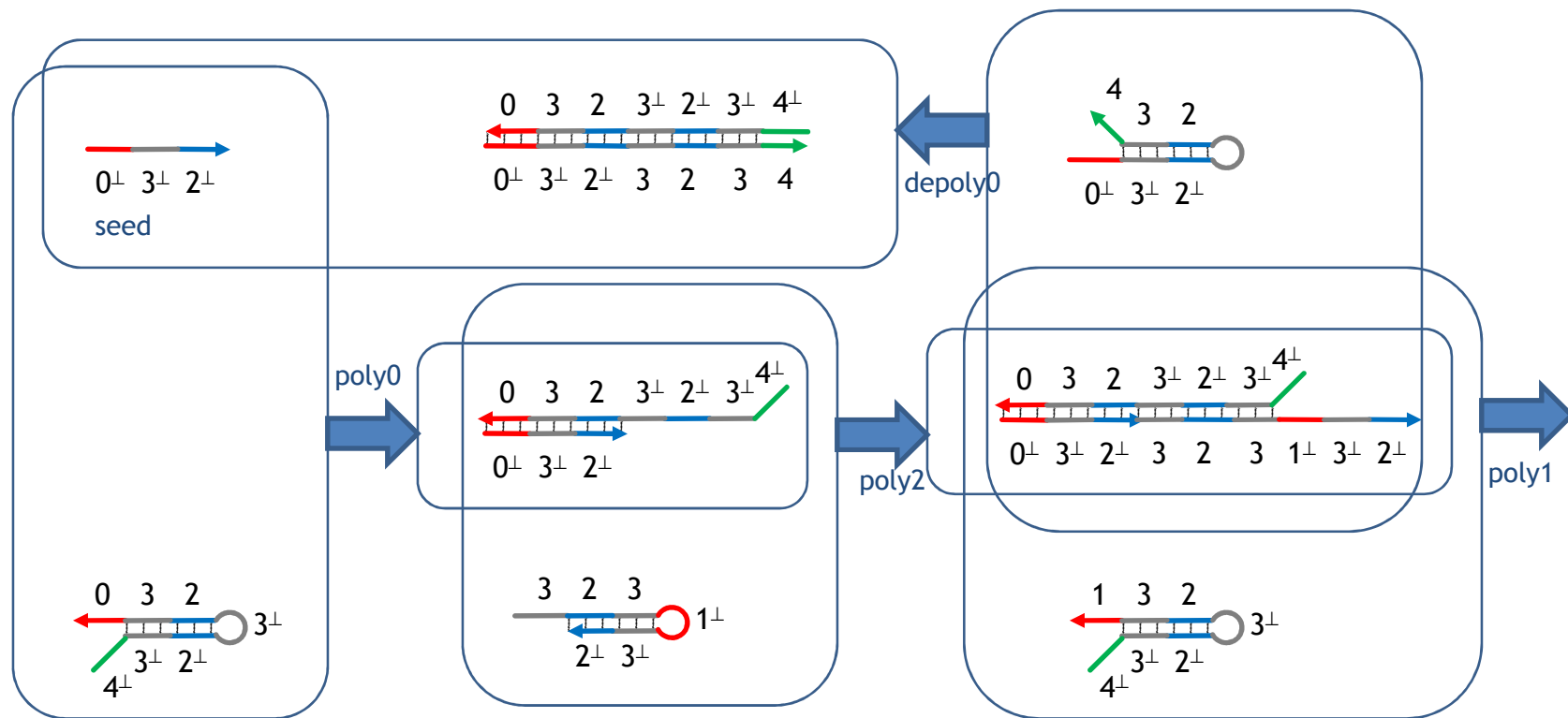
Composition of two transducers.

$$x \mid x.y \rightarrow y$$



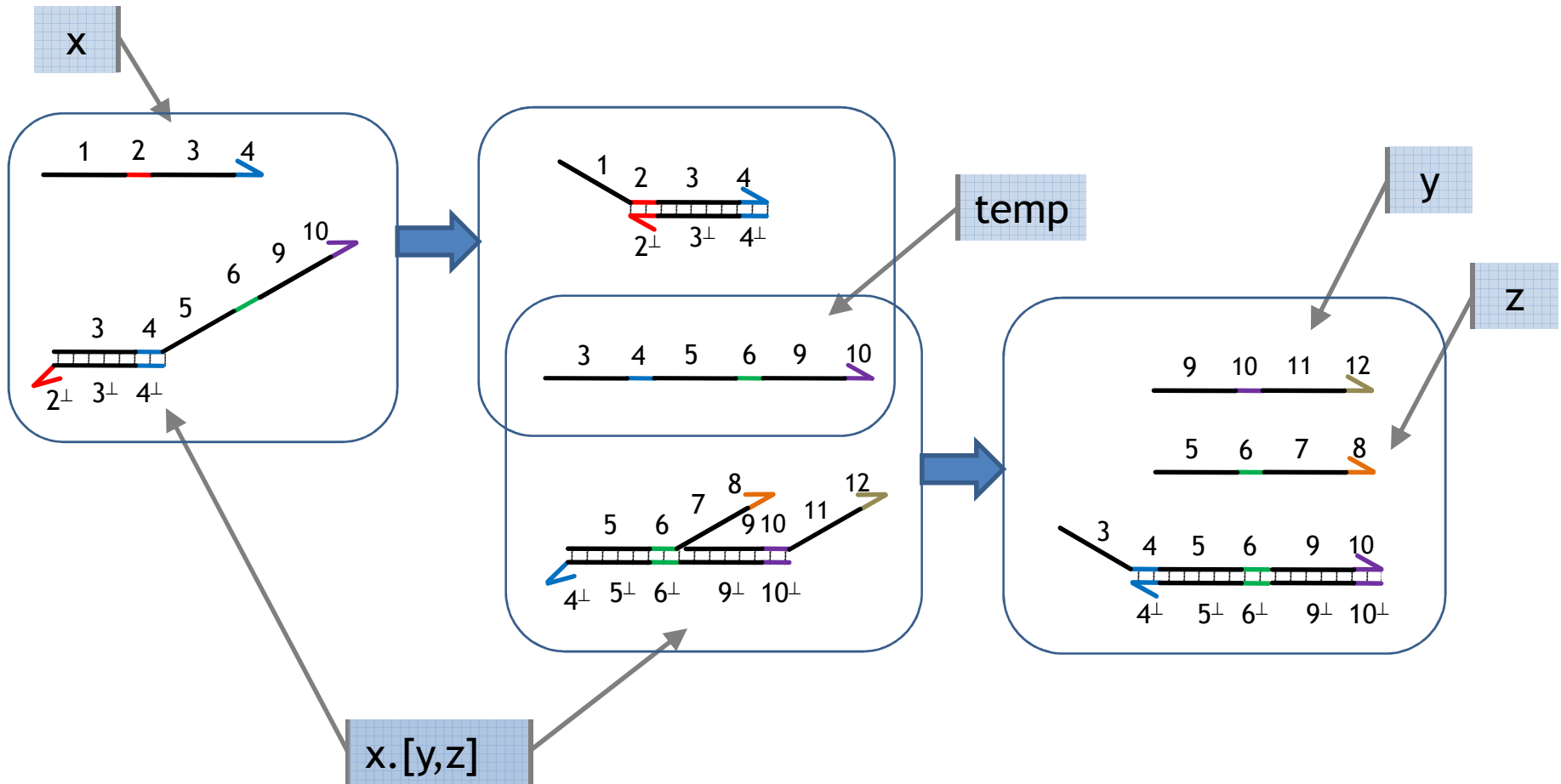
Venn Dnagrams™

By the way, this is a useful notation for complicated situations:



Fork

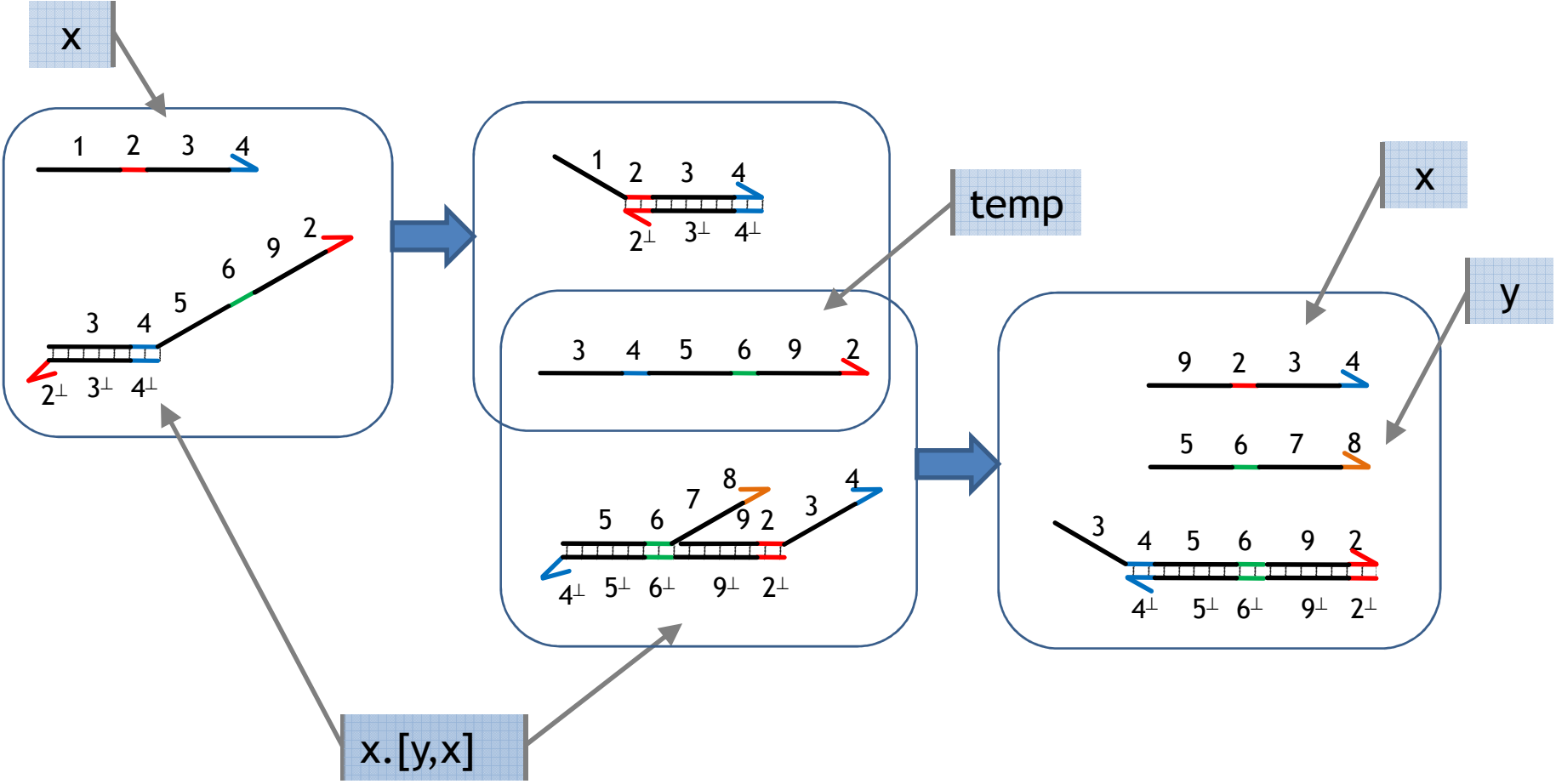
$$x \mid x.[y,z] \rightarrow y \mid z$$



Similarly to Sequence, with two output strands in second step (Soloveichik Fig 2)

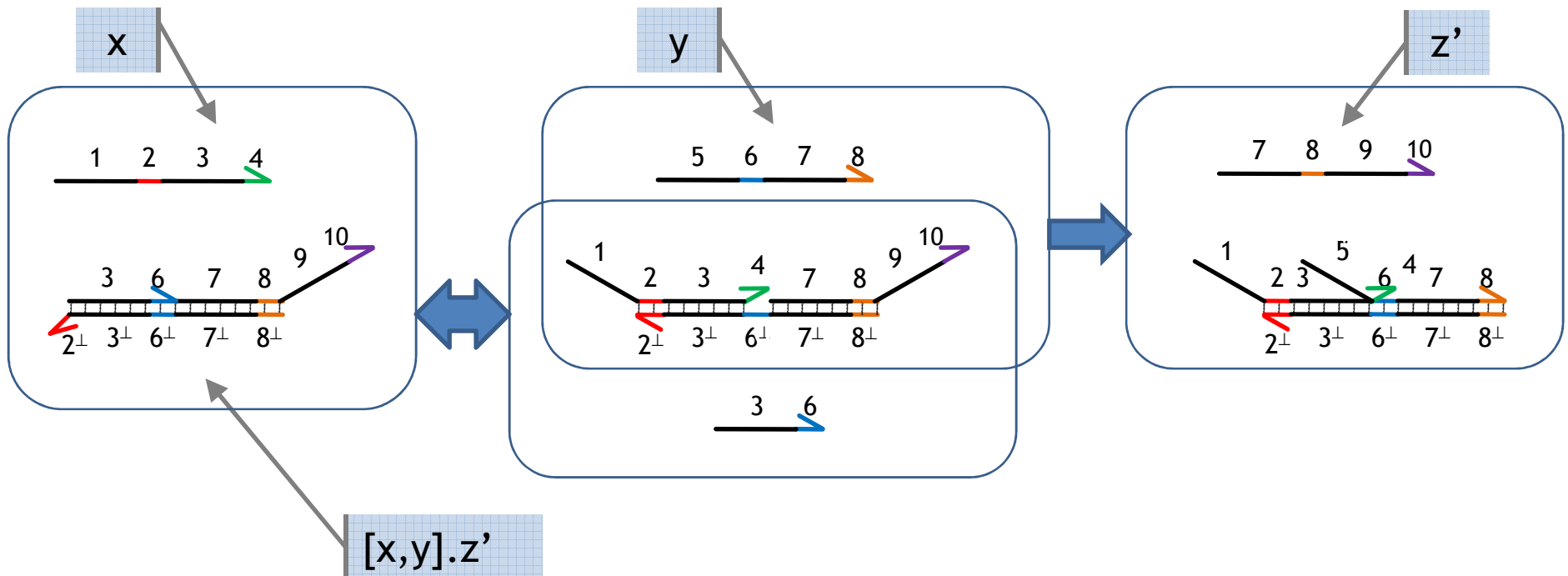
Fork-Catalyst

$$x \mid x.[y,x] \rightarrow y \mid x$$



2-way Join

$$x \mid y \mid [x,y].z \rightarrow z$$

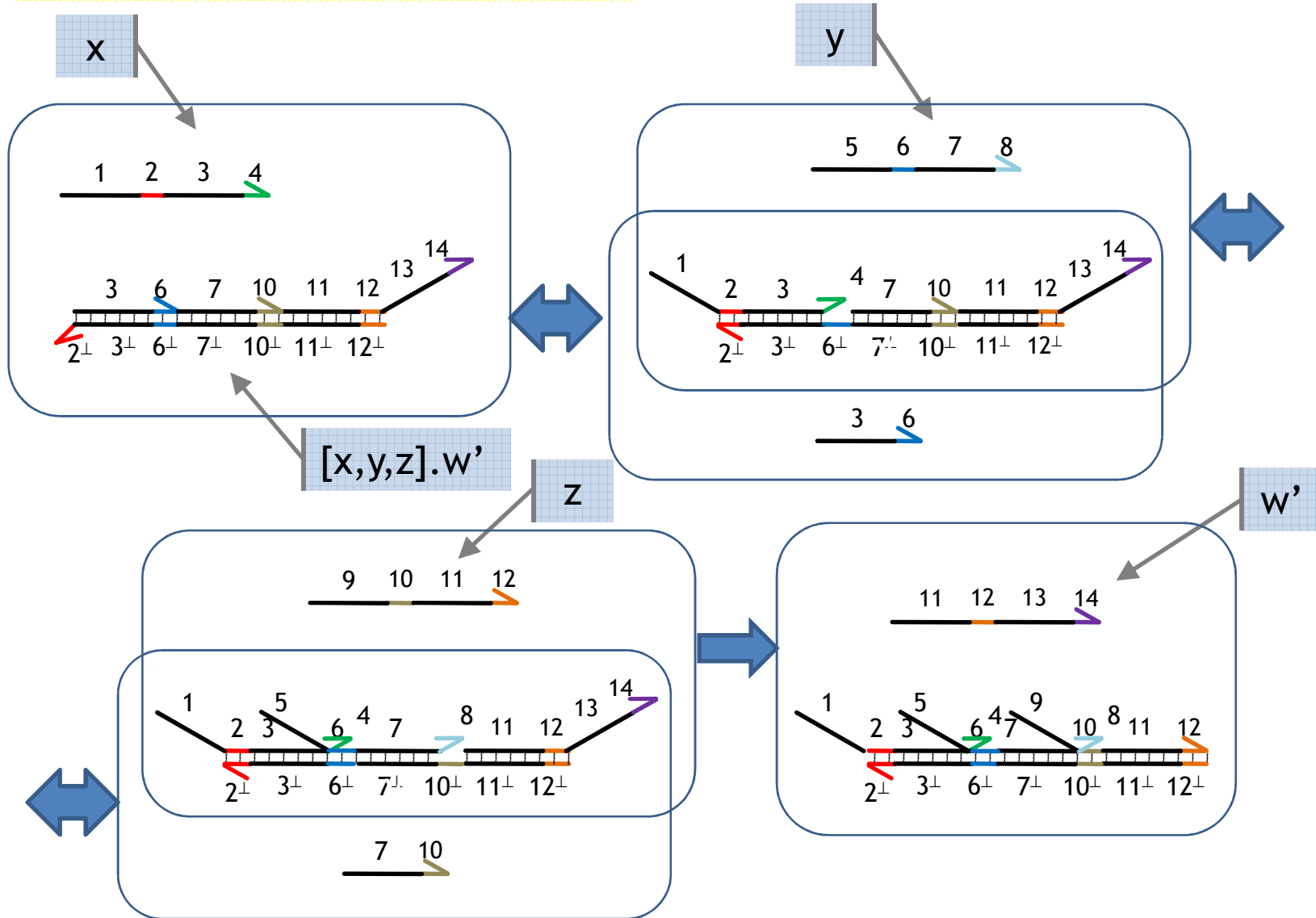


This can be implemented by two sequential inputs where the first one is reversible (Soloveichik Fig.3). An output transducer is then needed from z' to z , because z' overlaps with y .

Informally, this is implemented this way: $X = x \cdot ((x \mid X) + y \cdot z)$

3-way Join

$$x \mid y \mid z \mid [x,y,z].w \rightarrow w$$



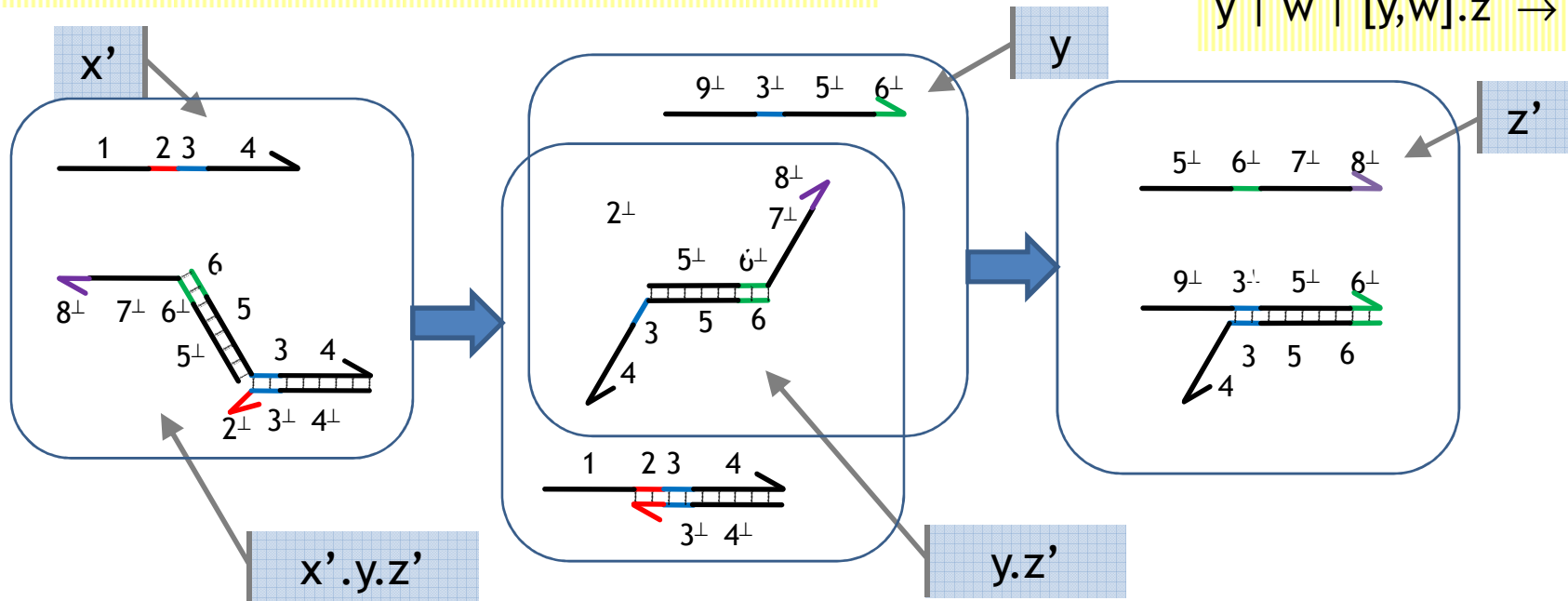
Ex.: Optimization of Cascades

$$x \mid y \mid x.y.z \rightarrow y \mid y.z \rightarrow z$$

instead of

$$x \mid x.w \rightarrow w$$

$$y \mid w \mid [y,w].z \rightarrow z$$

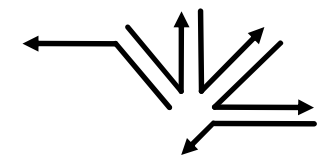


We need to prefix this with a transducer for x to x' , because x' is not in standard form and is not independent of y (3).

The signal y is non-standard as well, hence this optimization is context-dependent.

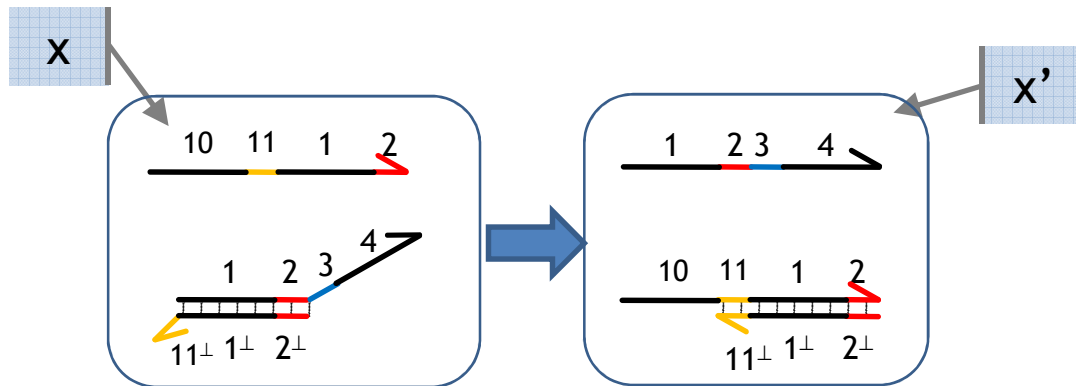
We need to postfix this with an output transducer for z' to z , because z' is not independent of y (5:6).

Note: This can be continued in a spoke pattern for any number of sequential inputs!!



Transducers for Optimized Cascades

Input transducer:



Output transducer:

(standard)

Combinatorial Strand Algebra

Strand Algebra \mathcal{P}

No compound expressions except for parallel composition $P|P$ and populations P^* .
Hence this is a combinator-based (“assembly”) language.

Here x is a *strand*, and $[..].[..]$ is a *gate*.

$$P ::= x \mid [x_1, \dots, x_n].[x_1', \dots, x_m'] \mid 0 \mid P|P \mid P^* \quad n \geq 1, m \geq 0$$

- x is a *strand*
- $x_1.x_2 \stackrel{\text{def}}{=} [x_1].[x_2]$ is a *sequence gate*
- $x.[x_1, \dots, x_m] \stackrel{\text{def}}{=} [x].[x_1, \dots, x_m]$ is a *fork gate*
- $[x_1, \dots, x_n].x \stackrel{\text{def}}{=} [x_1, \dots, x_n].[x]$ is a *join gate*
- 0 is *inert*
- $P|P$ is *parallel composition* of strands and gates
- P^* is a *population* of strands and gates

Note: $\sigma.P^*$ is not in the syntax: populations are only top-level.
C.f.: Petri net *tokens* (strands) and *transitions* (gates).
However, here both strands and gates are *consumed* by interaction.

Structural Congruence for \mathcal{P}

$$P \equiv P$$

equivalence

$$P \equiv P' \Rightarrow P' \equiv P$$

$$P \equiv P', P' \equiv P'' \Rightarrow P \equiv P''$$

$$P \equiv P' \Rightarrow P | P'' \equiv P' | P''$$

congruence

$$P \equiv P' \Rightarrow P^* \equiv P'^*$$

$$P | 0 \equiv P$$

diffusion

$$P | P' \equiv P' | P$$

$$P | (P' | P'') \equiv (P | P') | P''$$

$$P^* \equiv P^* | P$$

population

$$(0)^* \equiv 0$$

$$(P | P')^* \equiv P^* | P'^*$$

$$P^{**} \equiv P^*$$

Reduction for \mathcal{P}

$$x_1 \mid \dots \mid x_n \mid [x_1, \dots, x_n]. [x_1', \dots, x_m'] \rightarrow x_1' \mid \dots \mid x_m' \quad \text{Gate}$$

$$P \rightarrow P' \quad \Rightarrow \quad P \mid P'' \rightarrow P' \mid P'' \quad \text{Parallel}$$

$$P \equiv P_1, P_1 \rightarrow P_2, P_2 \equiv P' \quad \Rightarrow \quad P \rightarrow P' \quad \text{Mixing}$$

Technically, the Gate rule is:

$$x_1 \mid (\dots \mid (x_n \mid [x_1, \dots, x_n]. [x_1', \dots, x_m']) \dots) \rightarrow x_1' \mid (\dots \mid (x_m') \dots)$$

but we have structural congruence to rearrange.

Examples

$$x_1 \mid x_1 \cdot x_2 \rightarrow x_2$$

$$x_1 \mid x_1 \cdot x_2 \mid x_2 \cdot x_3 \rightarrow \rightarrow x_3$$

$$x_1 \mid x_2 \mid [x_1, x_2] \cdot x_3 \rightarrow x_3$$

$$x_1 \mid x_1 \cdot x_2 \mid x_1 \cdot x_3 \rightarrow x_2 \mid x_1 \cdot x_3$$

and also $\rightarrow x_3 \mid x_1 \cdot x_2$

$$x_1 \mid x_2 \mid x_3 \mid [x_1, x_2] \cdot x_4 \mid [x_1, x_3] \cdot x_5 \rightarrow x_3 \mid x_4 \mid [x_1, x_3] \cdot x_5$$

and also $\rightarrow x_2 \mid [x_1, x_2] \cdot x_4 \mid x_5$

$$X \mid ([X, x_1] \cdot [x_2, X])^*$$

a catalytic system ready to transform multiple x_1 to x_2 , with catalyst X

Emulation of \mathcal{P} in π

$$P ::= x : [x, \dots, x]. [x, \dots, x] : 0 : P \mid P : P^*$$
$$\llbracket 0 \rrbracket = 0$$
$$\llbracket P_1 \mid P_2 \rrbracket = \llbracket P_1 \rrbracket \mid \llbracket P_2 \rrbracket$$
$$\llbracket P^* \rrbracket = \llbracket P \rrbracket^*$$
$$\llbracket x \rrbracket = !x$$
$$\llbracket [x]. [x_1, \dots, x_n] \rrbracket = ?x. (!x_1 \mid \dots \mid !x_n)$$
$$\llbracket [x_1, x_2]. [x_1, \dots, x_n] \rrbracket = \text{rec } X. ?x_1. (!x_1.X + ?x_2. (!x_1 \mid \dots \mid !x_n))$$
$$\llbracket [x_1, x_2, x_3]. [x_1, \dots, x_n] \rrbracket = \text{etc.}$$

But this *does not preserve termination*, e.g. for $x_1 \mid [x_1, x_2]. []$.

DNA Machine Calculus

Syntax

We define a syntax of DNA D in terms of Gates \overleftarrow{G} and Strands \overrightarrow{S} . An arbitrary DNA sequence is represented by a number x , where a toehold sequence is represented in bold as \mathbf{x} .

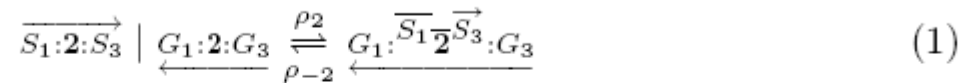
$D ::=$	\emptyset	Null	$G ::=$	S	Sequence	
	\vdots	\overrightarrow{S}	Strand	\vdots	$\overrightarrow{S_1} \overrightarrow{S_2} \overrightarrow{S_3}$	Bound Strand, $S_2 \neq \emptyset$
	\vdots	\overleftarrow{G}	Gate	\vdots	$G_1 : G_2$	Concatenate
	\vdots	$D_1 \mid D_2$	Parallel			
$S ::=$	\emptyset	Empty				
	\vdots	x	Code			
	\vdots	\mathbf{x}	Toehold			
	\vdots	$S_1 : S_2$	Concatenate			

Branch migration as Structural Congruence:

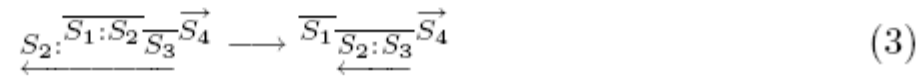
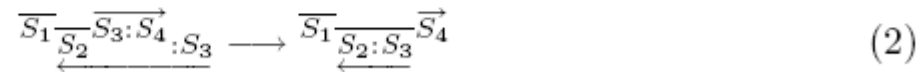
$$\overrightarrow{S_1} \overrightarrow{S_2} \overrightarrow{S_3} : \overrightarrow{S_4} \overrightarrow{S_5} \overrightarrow{S_3} \overrightarrow{S_6} \overrightarrow{S_7} \equiv \overrightarrow{S_1} \overrightarrow{S_2} \overrightarrow{S_3} \overrightarrow{S_4} : \overrightarrow{S_5} \overrightarrow{S_3} \overrightarrow{S_6} \overrightarrow{S_7} \quad (9)$$

Reduction Rules

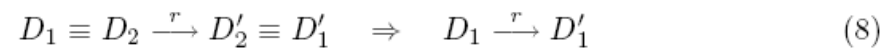
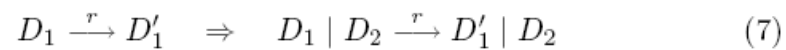
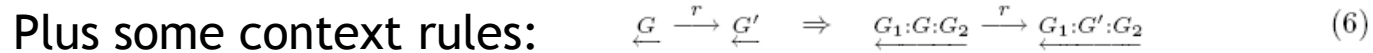
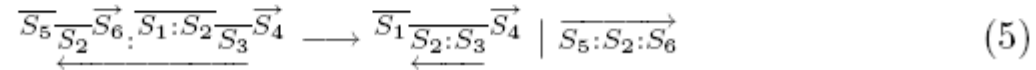
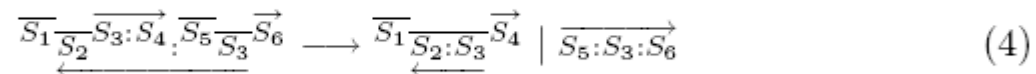
Toe-hold binding and unbinding:



Right and left strand binding to a gate



Right and left strand displacement from a gate



Examples

Annihilation

$$\overrightarrow{1:2:3:4} \mid \overleftarrow{2:3:4} \xrightarrow{\rho_2} \overleftarrow{\overrightarrow{2:3:4}} : \overrightarrow{3:4} \longrightarrow \overleftarrow{\overrightarrow{2:3:4}}$$

Sequence

$$\begin{aligned} \overrightarrow{1:2:3:4} \mid \overleftarrow{2:3:4} \overrightarrow{5:6} &\xrightarrow{\rho_2} \overleftarrow{\overrightarrow{2:3:4}} : \overrightarrow{3:4} \overrightarrow{5:6} \longrightarrow \overleftarrow{\overrightarrow{2:3:4}} \mid \overrightarrow{3:4:5:6} \\ \overrightarrow{3:4:5:6} \mid \overleftarrow{4:5:6} \overrightarrow{7:8} &\xrightarrow{\rho_4} \overleftarrow{\overrightarrow{4:5:6}} : \overrightarrow{5:6} \overrightarrow{7:8} \longrightarrow \overleftarrow{\overrightarrow{4:5:6}} \mid \overrightarrow{5:6:7:8} \end{aligned}$$

2-Way Fork

$$\overrightarrow{1:2:3:4} \mid \overleftarrow{2:3:4} \overrightarrow{5:6:9:10} \xrightarrow{\rho_2} \overleftarrow{\overrightarrow{2:3:4}} : \overrightarrow{3:4} \overrightarrow{5:6:9:10} \longrightarrow \overleftarrow{\overrightarrow{2:3:4}} \mid \overrightarrow{3:4:5:6:9:10}$$

$$\begin{aligned} \overrightarrow{3:4:5:6:9:10} \mid \overleftarrow{4:5:6} \overrightarrow{7:8} \overrightarrow{9:10} \overrightarrow{11:12} &\xrightarrow{\rho_4} \overleftarrow{\overrightarrow{4:5:6:9:10}} : \overrightarrow{5:6} \overrightarrow{7:8} \overrightarrow{9:10} \overrightarrow{11:12} \\ &\longrightarrow \overleftarrow{\overrightarrow{4:5:6} \overrightarrow{9:10}} : \overrightarrow{9:10} \overrightarrow{11:12} \mid \overrightarrow{5:6:7:8} \\ &\longrightarrow \overleftarrow{\overrightarrow{4:5:6:9:10}} \mid \overrightarrow{9:10:11:12} \mid \overrightarrow{5:6:7:8} \end{aligned}$$

2-Way Join

$$\overrightarrow{1:2:3:4} \mid \overleftarrow{2:3:6:7:8} \overrightarrow{9:10} \xrightleftharpoons[\rho_{-2}]{\rho_2} \overleftarrow{\overrightarrow{2:3:4}} : \overrightarrow{3:6:7:8} \overrightarrow{9:10} \xrightleftharpoons[\rho_6]{\rho_{-6}} \overrightarrow{3:6} \mid \overleftarrow{\overrightarrow{2:3} \overrightarrow{4}} : \overrightarrow{6:7:8} \overrightarrow{9:10}$$

$$\overrightarrow{5:6:7:8} \mid \overleftarrow{\overrightarrow{2:3} \overrightarrow{4}} : \overrightarrow{6:7:8} \overrightarrow{9:10} \xrightarrow{\rho_6} \overleftarrow{\overrightarrow{2:3} \overrightarrow{4}} : \overrightarrow{5:6} \overrightarrow{7:8} \overrightarrow{7:8} \overrightarrow{9:10} \longrightarrow \overleftarrow{\overrightarrow{2:3} \overrightarrow{4}} : \overrightarrow{5:6:7:8} \mid \overrightarrow{7:8:9:10}$$

Uses

- A formal translation from chemical reactions to DNA sequences.
- A formal translation from Strand Algebra to DNA sequences (instead of pictures).

Stochastic Strand Algebra

Stochastic Populations

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

$$P ::= x \cdot [x_1, \dots, x_n] \cdot_r [x_1', \dots, x_m'] \cdot 0 \cdot P \mid P \quad n \geq 1, m \geq 0$$

- Instead of unbounded populations P^* should think of populations of size k , P^k , which of course we already have from iterated parallel composition.
- Further, we can think of populations of *constant size* k , $P^{=k}$.
These too are in a sense definable, using a bigger *buffer population*:
 - Take for example $P = x.Y$
 - $P^{=k} \stackrel{\text{def}}{=} X^k \mid P\text{buf}^{k^3}$ for some fresh X
 - $P\text{buf}^n \stackrel{\text{def}}{=} ([X, x] \cdot [Y, X])^n$
 - Here k^3 is an example of a large enough buffer: it ensures that reactions on X are much faster than reactions on x by mass action, and that the join $[X, x]$ is always saturated in the X input.

Constant Populations

- We have, for e.g. $P=x.Y$:

$$\begin{aligned}
 \circ P^{=k} &= X^k \mid Pbuf^{k^3} \\
 &= (X \mid [X,x].[Y,X])^k \mid Pbuf^{k^3-k} \\
 &\rightarrow Y \mid X \mid (X \mid [X,x].[Y,X])^{k-1} \mid Pbuf^{k^3-k} \\
 &= Y \mid (X \mid [X,x].[Y,X])^k \mid Pbuf^{k^3-k-1}
 \end{aligned}$$

$$\begin{aligned}
 P &= x.Y \\
 P^{=k} &\stackrel{\text{def}}{=} X^k \mid Pbuf^{k^3} \\
 Pbuf^n &\stackrel{\text{def}}{=} ([X,x].[Y,X])^n
 \end{aligned}$$

- Hence there is always constant population:

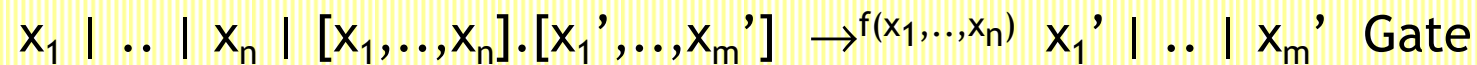
- $(X \mid [X,x].[Y,X])^k$
- with a constant k weight on the x input (or temporarily $\sim k-1$)
- constant up to a re-equilibration speed determined by $\sim k^3$

- Moreover, the buffer can be “topped up” from time to time, without disturbing the rates of the system!
And we can have “buffers of buffers” $(P^{=k})^{=k}$.

- This gives us a practical way to implement “unbounded recursion”, akin to P^* , in a stochastic system (up to topping up the buffers).
Otherwise all computations terminate.

Stochastic Gate Rates

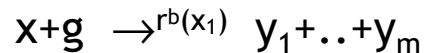
- A simple idea: associate a reaction rate r to each gate.
 - This is what one would do in stochastic Petri nets (gate = transition).
 - But this does not reflect the DNA implementation. Instead:
- Each x_i has a binding rate $r^b(x_i)$ and an unbinding rate $r^u(x_i)$. Then the rate of a gate reaction is a function of those rates (it is not an exponential distribution):



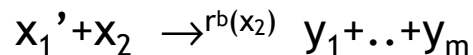
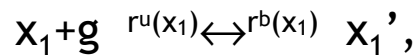
Semantics of \mathcal{P}^r in Chemistry(FSRN)

The (common) assumption here is that branch migration is fast w.r.t. binding rates. First we extract the system of reactions from a whole term P:

for each $g = x.[y_1, \dots, y_m]$ add the reaction:



for each $g = [x_1, x_2].[y_1, \dots, y_m]$ add the reactions:



etc. for higher joins.

These are *reactions*, i.e. they are independent of the number of strands x and gates g in the system. Then, we add all those as “initial conditions”, as a translation $\llbracket P \rrbracket$:

$$\llbracket x \rrbracket = x$$

$$\llbracket [x_1, \dots, x_n].[x_1', \dots, x_m'] \rrbracket = g \quad (\text{that } g \text{ associated to this gate above})$$

$$\llbracket 0 \rrbracket = 0$$

$$\llbracket P | P' \rrbracket = \llbracket P \rrbracket + \llbracket P' \rrbracket$$

So if there are n x and m gates $x.[..]$, the propensity of binding is $n*m*r^b(x)$.

Semantics of Chemistry(FSRN) in \mathcal{P}^r

- This is what [Soloveichik et al.] do: how to implement an *arbitrary set of chemical reactions* (before this, one would not have known in general how to engineer a set of chemical species that would obey those reactions).
- Given a reaction $A+B\rightarrow C$ make up some DNA with the same (approximate) kinetics.
- But we cannot just translate $A+B\rightarrow C$ to $[A,B].[C]$ or $([A,B].[C])^k$: gates are consumed, hence reactions would get “weaker and weaker”.
- We can solve that problem by constant populations $([A,B].[C])^k$ (which is what [Soloveichik et al.] do). For further details see that paper.

Compiling to Strand Algebra

High(er)-Level Languages

- We now have an intermediate language: the combinatorial strand algebra
 - It can be compiled “directly” to DNA following [Soloveichik et al.]
- But we really want to compile “high-level languages”. Such as:
 - Boolean Networks (fairly easy)
 - Finite Stochastic Reaction Networks (Chemistry) [Soloveichik et al.]
 - Petri Nets (easy)
 - Finite State Automata and Transducers (easy)
 - Interacting Automata (hard)
 - π -calculus (???)
- And also
 - Higher-level strand algebras, which may form more convenient intermediate languages.
 - Such as the *Nested Strand Algebra*.

Nested Strand Algebra

Motivation

- 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 only change is allowing arbitrary terms after a gate input.

Nested Strand Algebra $n\mathcal{P}$

$$\mathcal{P} ::= x \mid [x_1, \dots, x_n].\mathcal{P} \mid 0 \mid \mathcal{P} \mid \mathcal{P} \mid \mathcal{P}^* \quad n \geq 1$$

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

And we also allow triggering whole populations: $x.\mathcal{P}^*$

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

$[x_1 \mid \dots \mid x_n].[x_1' \mid \dots \mid x_m']$ becomes $[x_1 \mid \dots \mid x_n].(x_1' \mid (\dots \mid (x_m' \mid 0) \dots))$

Structural Congruence for $n\mathcal{P}$

$P \equiv P$ equivalence

$P \equiv P' \Rightarrow P' \equiv P$

$P \equiv P', P' \equiv P'' \Rightarrow P \equiv P''$

$P \equiv P' \Rightarrow [x_1, \dots, x_n].P \equiv [x_1, \dots, x_n].P'$ congruence

$P \equiv P' \Rightarrow P | P'' \equiv P' | P''$

$P \equiv P' \Rightarrow P^* \equiv P'^*$

$P | 0 \equiv P$ diffusion

$P | P' \equiv P' | P$

$P | (P' | P'') \equiv (P | P') | P''$

$P^* \equiv P^* | P$ population

$(0)^* \equiv 0$

$(P | P')^* \equiv P^* | P'^*$

$P^{**} \equiv P^*$

Reduction for $n\mathcal{P}$

$x_1 \mid \dots \mid x_n \mid [x_1, \dots, x_n].P \rightarrow P$

Gate

$P \rightarrow P' \Rightarrow P \mid P'' \rightarrow P' \mid P''$

Parallel

$P \equiv P_1, P_1 \rightarrow P_2, P_2 \equiv P' \Rightarrow P \rightarrow P'$

Mixing

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

$U(P) = X \mid U(X,P)$ for fresh X

$U(X, x) = X.x$

$U(X, [x_1, \dots, x_n].P) = [X, x_1, \dots, x_n].Y \mid U(Y,P)$ for fresh Y

$U(X, 0) = X.[]$

$U(X, P \mid P') = X.[Y,Z] \mid U(Y,P) \mid U(Z,P')$ for fresh Y, Z

$U(X, P^*) = (X.[Y,X] \mid U(Y,P))^*$ for fresh Y

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

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

\mathcal{X}_i is the 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{X}_p be the set of those $\mathcal{X} \in \mathfrak{X}$ that
do not contain any strand that occurs in P .

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

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

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

$$U(X, P)_x =$$

- If $P = x$ return $X.x$
- If $P = [x_1, \dots, x_n].P'$ return $[X, x_1, \dots, x_n].\mathcal{X}_0 \mid U(\mathcal{X}_0, P')_{x_{\geq 1}}$
- If $P = 0$ return $X.[]$
- If $P = P' \mid P''$ return $X.[\mathcal{X}_0, \mathcal{X}_1] \mid U(\mathcal{X}_0, P')_{evn(\mathcal{X}_{\geq 2})} \mid U(\mathcal{X}_1, P'')_{odd(\mathcal{X}_{\geq 2})}$
- If $P = P'^*$ return $(X.[\mathcal{X}_0, X] \mid U(\mathcal{X}_0, P')_{x_{\geq 1}})^*$

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:

$$X = f(X, Y)$$

$$Y = g(X, Y)$$

in $P(X, Y)$

write:

$$P(X, Y) \mid (X.f(X, Y))^* \mid (Y.g(X, Y))^*$$

Triggering Populations

We can nest populations after all:

$$U(x.P^*) = X \mid [X,x].Z \mid (Z.[Y,Z] \mid U(Y,P))^*$$

$$\begin{aligned} U(P^{**}) &= X \mid (X.[Y,X] \mid U(Y,P^*))^* = \\ X \mid (X.[Y,X] \mid (Y.[Z,Y] \mid U(Z,P))^*)^* &= \\ X \mid X.[Y,X]^* \mid (Y.[Z,Y] \mid U(Z,P))^{**} &= \\ X \mid (X.[Y,X] \mid Y.[Z,Y] \mid U(Z,P))^* & \end{aligned}$$

$U(P) =$	$X \mid U(X,P)$	for fresh X
$U(X, x) =$	$X.x$	
$U(X, [x_1, \dots, x_n].P) =$	$[X, x_1, \dots, x_n].Y \mid U(Y,P)$	for fresh Y
$U(X, 0) =$	$X.[]$	
$U(X, P \mid P') =$	$X.[Y,Z] \mid U(Y,P) \mid U(Z,P')$	for fresh Y,Z
$U(X, P^*) =$	$(X.[Y,X] \mid U(Y,P))^*$	for fresh Y

Interacting Automata

Automata to DNA

- There are many schemas to compile automata to molecules
 - But most (all?) are about compiling a single automaton (e.g. an FSA).
- **Interacting Automata** can be compiled to chemical reactions [TCS'08].
 - Are concurrent and population based (a subset of CCS).
 - The translation has an n^2 blowup (mean automata are “more compact”).
 - But how does one engineer the necessary molecules?
- **Arbitrary chemistry can be compiled to DNA [Soloveichik et al.]**.
 - The translation is stochastically “almost” faithful.
 - Which can be seen as a defect of the translation, if you are a chemist.
- **Hence Interacting Automata can be compiled to DNA**.
 - Again, stochastically this is “almost” faithful as a single transition may need to be implemented with two transitions, which have a different distribution.
 - But for automata, we are probably not picky: we mostly want them to change states.

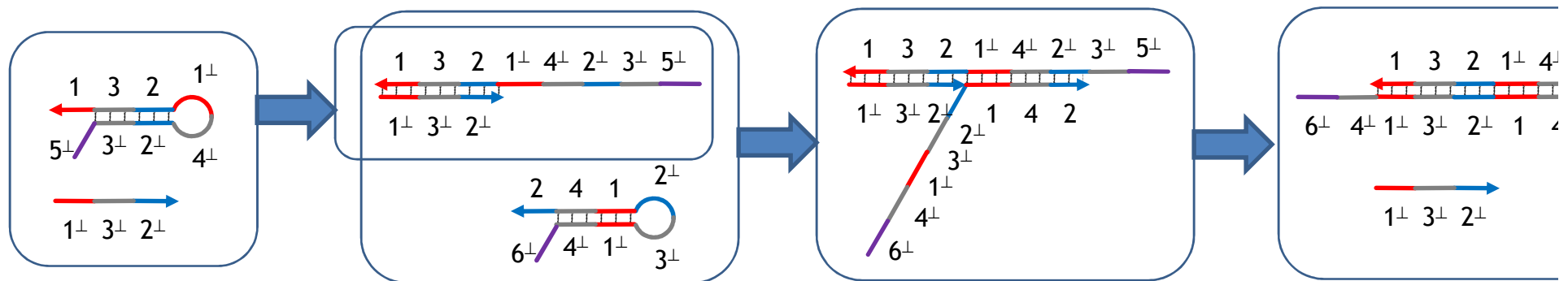
Alternative DNA Mechanisms

Hairpin Operators

Programming biomolecular self-assembly pathways

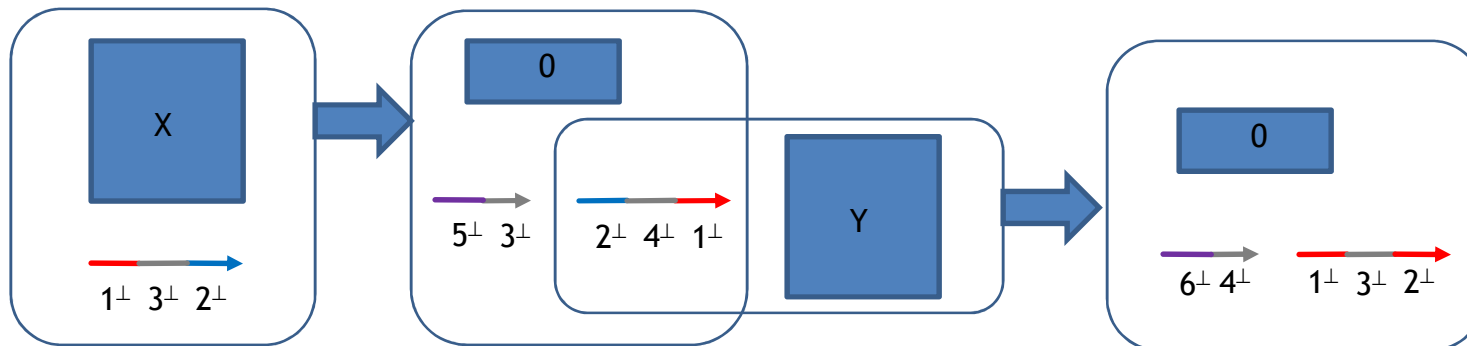
P. Yin, H.M.T. Choi, C.R. Calvert, N.A. Pierce

[Nature](#), 451:318-322, 2008.



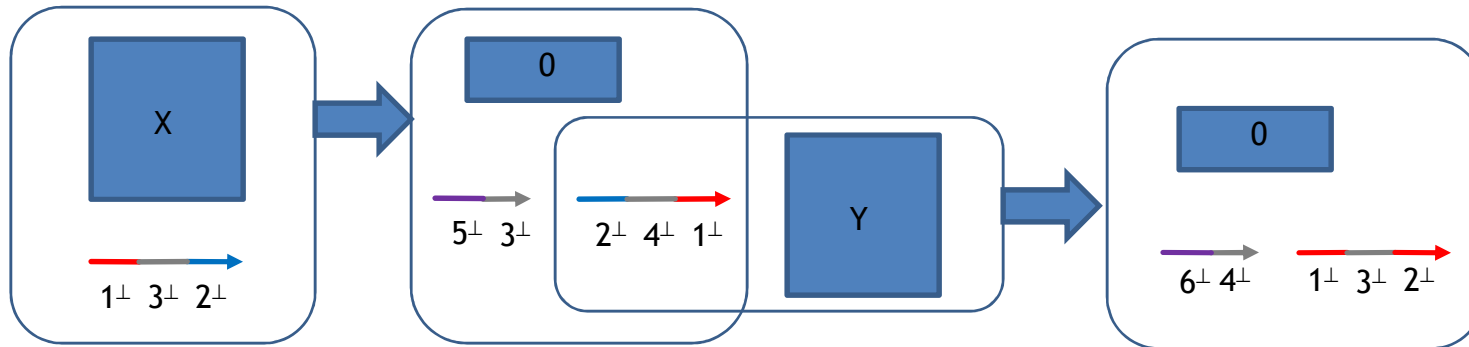
This is a quite different set of DNA primitives.

But it has this *function*:



Doing the same in Strand Algebra

If we just think of the *function* of that network:



Then we can implement that function in strand algebra:

$$X = (1^{\perp}:3^{\perp}:2^{\perp}).[(5^{\perp}:3^{\perp}), (2^{\perp}:4^{\perp}:1^{\perp})]$$

$$Y = (2^{\perp}:4^{\perp}:1^{\perp}).[(6^{\perp}:4^{\perp}), (1^{\perp}:3^{\perp}:2^{\perp})]$$

This leads to a different DNA implementation (according to the canonical DNA semantics of strand algebra).

But this is what algebra is good for: abstracting implementation models.

Open Problems/Questions

Implementing Choice in DNA

- I.e. compiling Choice to Join.
- 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 (or CCS).
- 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?

Compiling Choice to Join

- I.e., compiling interacting automata to strand algebra *without* going through the n^2 -expansion of the chemical translation.
- There is no known direct/compositional/linear translation of Interacting Automata (CGF) to strand algebra, because of the difficulties in translating choice.

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.)