Automata as Molecules

Implementing Interacting Automata as Biochemical Systems

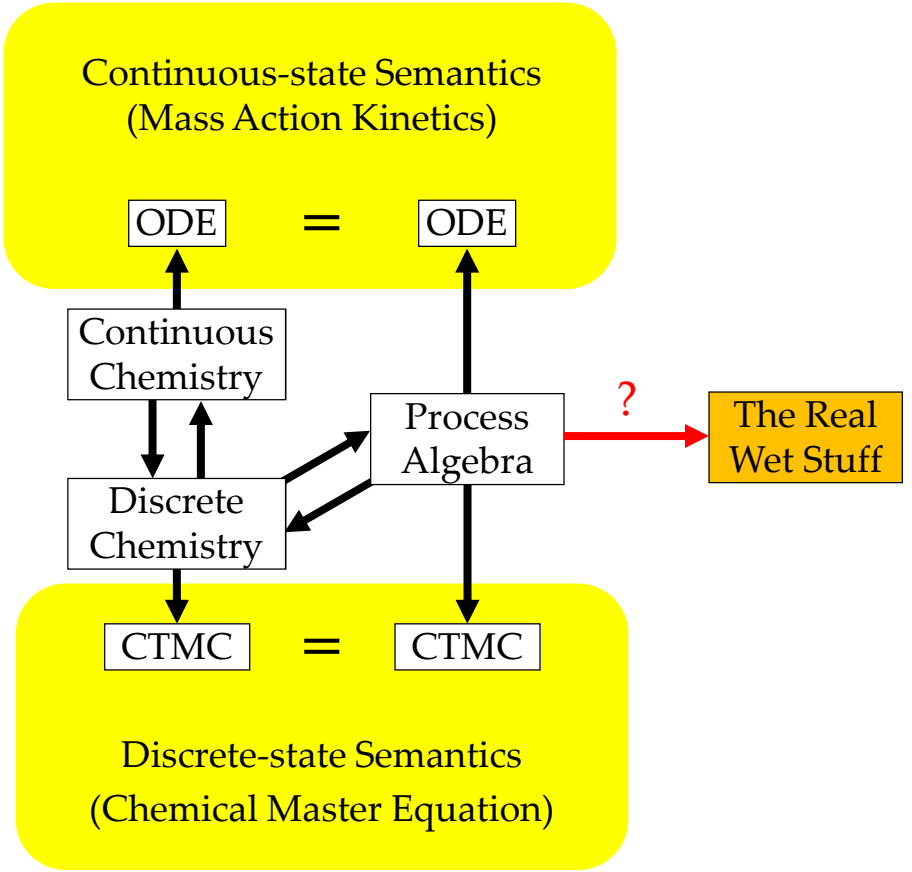
Luca Cardelli

Microsoft Research

Open Lectures for PhD Students in Computer Science
Warsaw 2009-05-07..08

http://lucacardelli.name

Motivation



How do we implement an arbitrary process?

How do we implement an arbitrary chemical system?
(how do we then implement the chemical species?)

Automata to Molecules

- 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 (means 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.
- Direct Compilation of Interacting Automata to DNA.
  - We can more simply go directly from Interacting Automata to DNA.
  - In doing so, we want to preserve the stochastic semantics (rates).

DNA Computing

- Early DNA Computing
  - Demonstrated computation by DNA hybridization [Adelman].
  - Why DNA? Widely available mature technology.
  - Massively concurrent (but still not enough for NP-complete problems).
  - Slow *and* awkward (manual cycling).

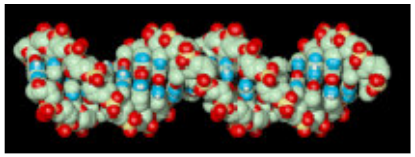
- New Focus
  - Not going to compete with Intel in speed (hours … days).
  - But can interface with biological systems!
  - For detection and intervention in live organisms.

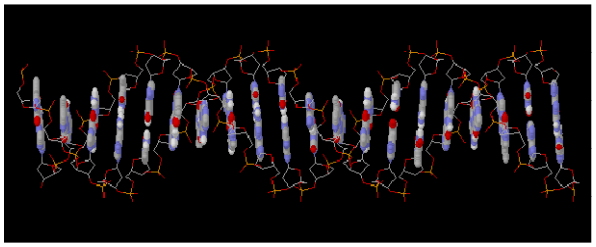
- New Paradigm
  - Autonomous DNA computation (mix-and-go) [Yurke&Mills].
  - Output readout by fluorescence or atomic microscopy, in vitro.
  - Or by influencing cellular mechanisms in vivo [Shapiro survey].

Computation by DNA Strand Displacement

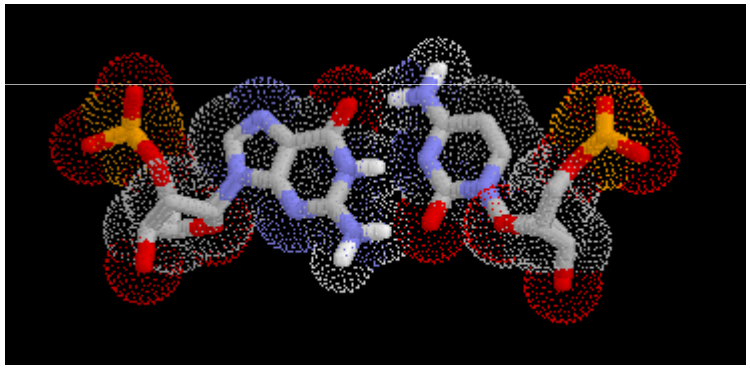
ACGT

Interactive DNA Tutorial
(http://www.biosciences.bham.ac.uk/labs/minchin/tutorials/dna.html)

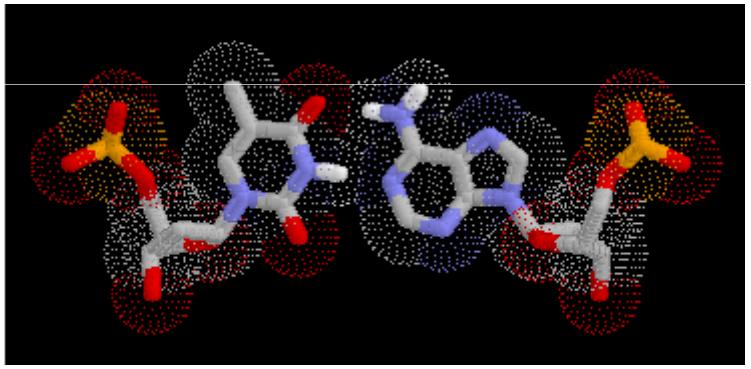




Sequence of Base Pairs



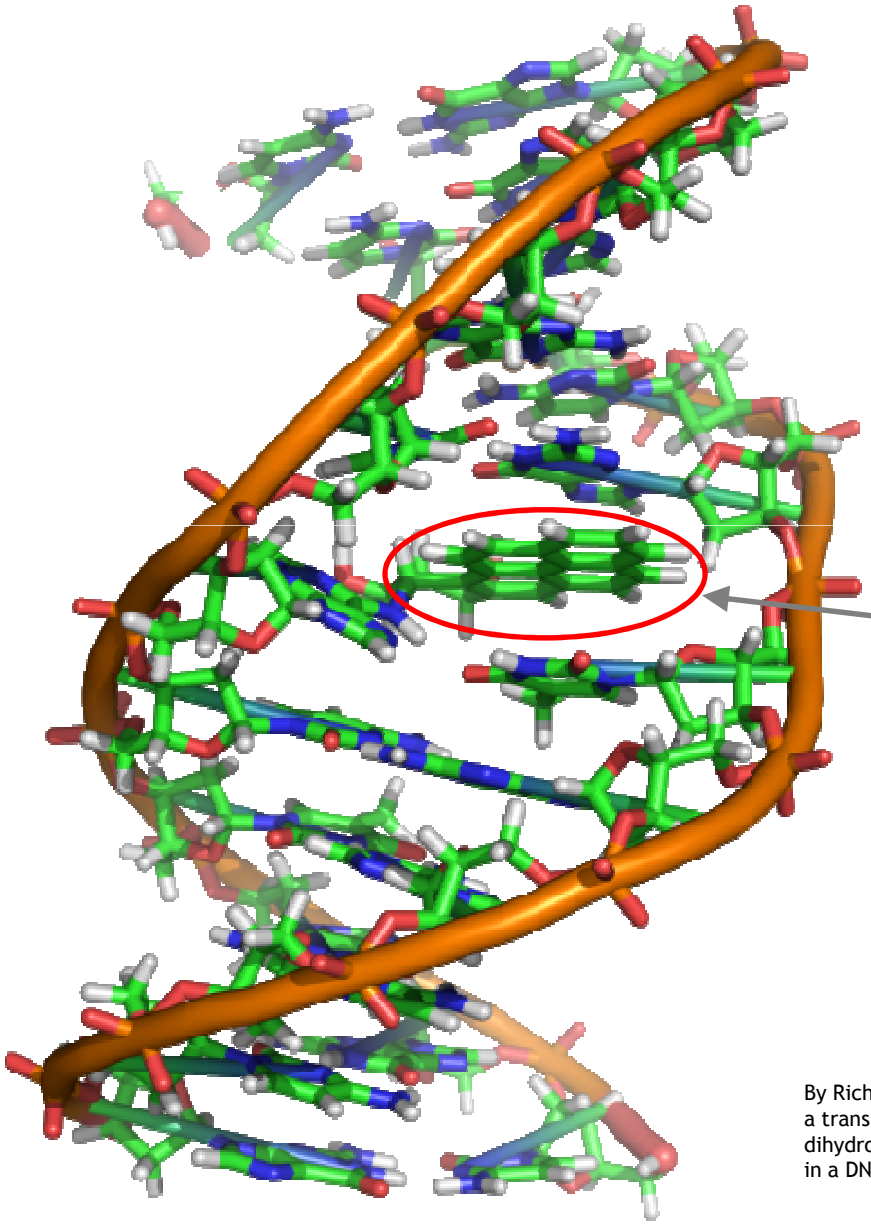
GC Base Pair
Guanine-Cytosine



TA Base Pair
Thymine-Adenine

Hence DNA is a string over a 4-letter ACGT alphabet
Human genome : ~3 billion base pairs
= 750 Megabytes (since 1 byte encodes 4 base pairs)
= 1 movie download!

DNA Double Helix



Benzopyrene

(What happens when you smoke)

By Richard Wheeler (Zephyris) 2007. Solution structure of a trans-opened (10S)-dA adduct of +)-(7S,8R,9S,10R)-7,8-dihydroxy-9,10-epoxy-7,8,9,10-tetrahydrobenzo[a]pyrene in a DNA duplex. Wikimedia Commons.

Watson-Crick Duality

G - C
T - A
Affinity

G:A:C:T
C:T:G:A
Double Strand

$G^{\perp} = C$
$T^{\perp} = A$
Complementarity

$X^{\perp\perp} = X$

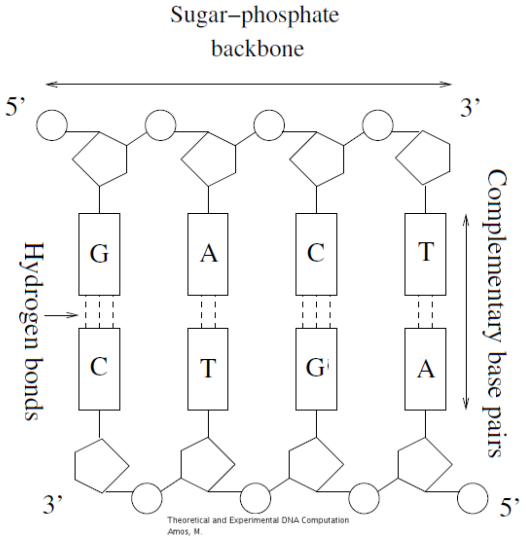
3' end

C:T:G:A = A:G:T:C

Equal Single Strands

G:A:C:T
$\perp$
C:T:G:A = A:G:T:C

Complementary Single Strands



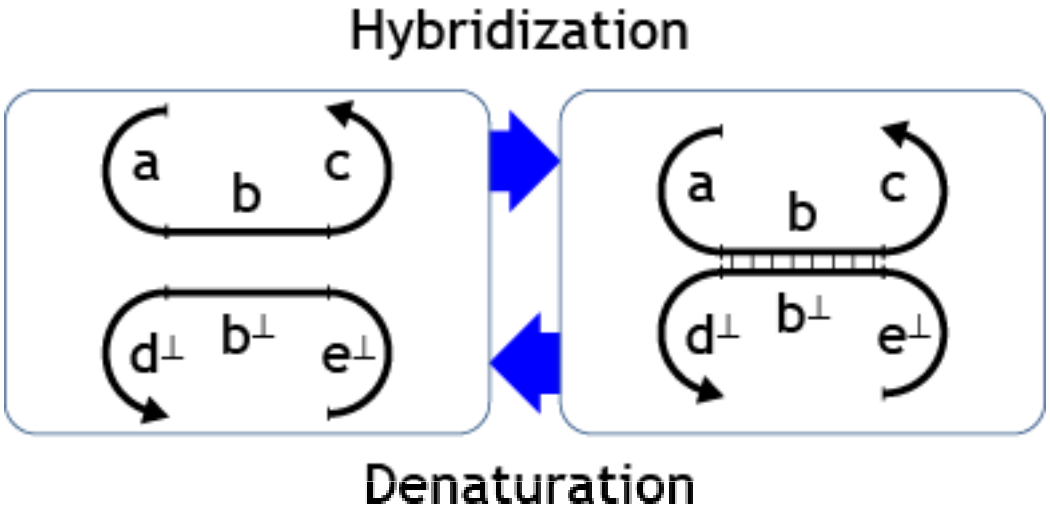
Hence $(G:A:C:T)^{\perp} = A:G:T:C = T^{\perp}:C^{\perp}:A^{\perp}:G^{\perp}$

all written from 5' to 3'

$(X:Y)^{\perp} = Y^{\perp}:X^{\perp}$

Watson-Crick duality
(for any sequences of bases X,Y)

Hybridization

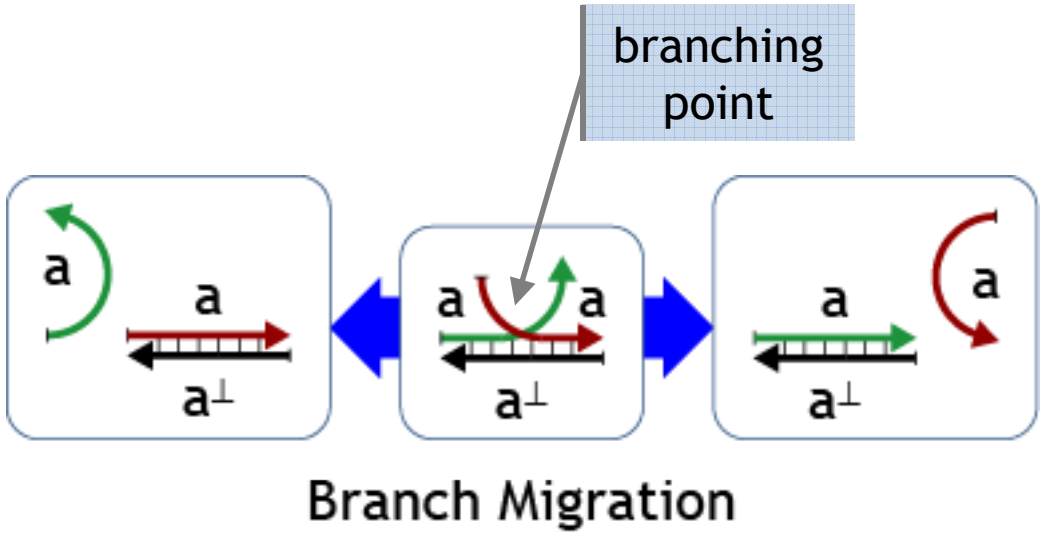


Hybridization is also called annealing; denaturation is also called melting.

The direction of the reaction (or in general the equilibrium between the two states) is determined by a number of factors, e.g. temperature.

We assume we are in conditions that favor hybridization beyond a certain length of matching region.

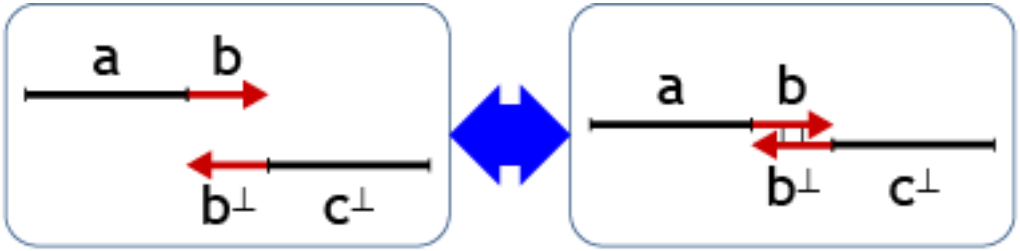
Branch Migration



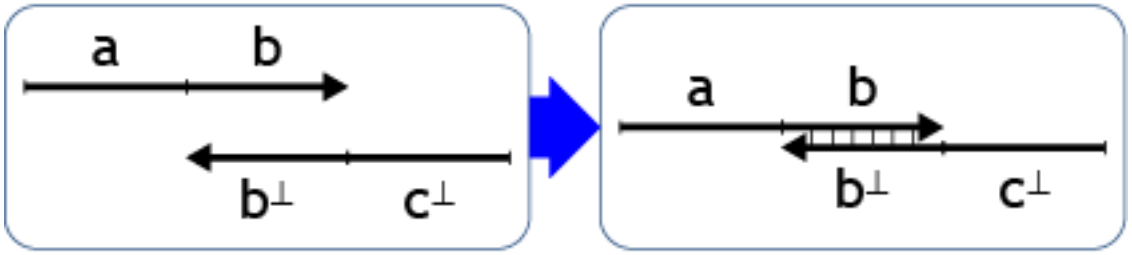
Branch Migration

The branching point moves left and right by a random walk. Until it reaches an end point.

Short and Long Segments

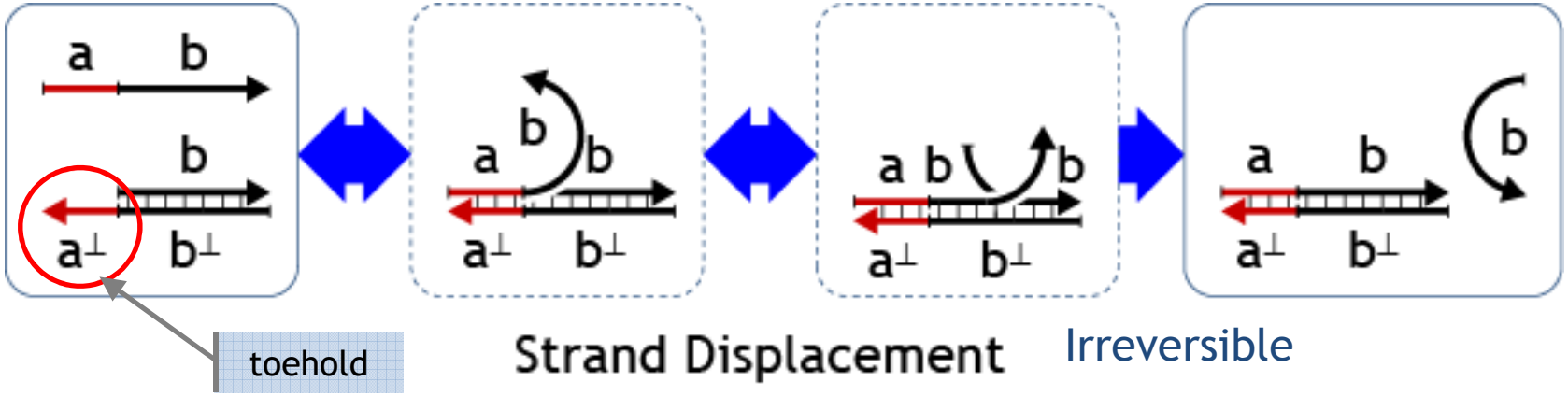


Reversible Binding

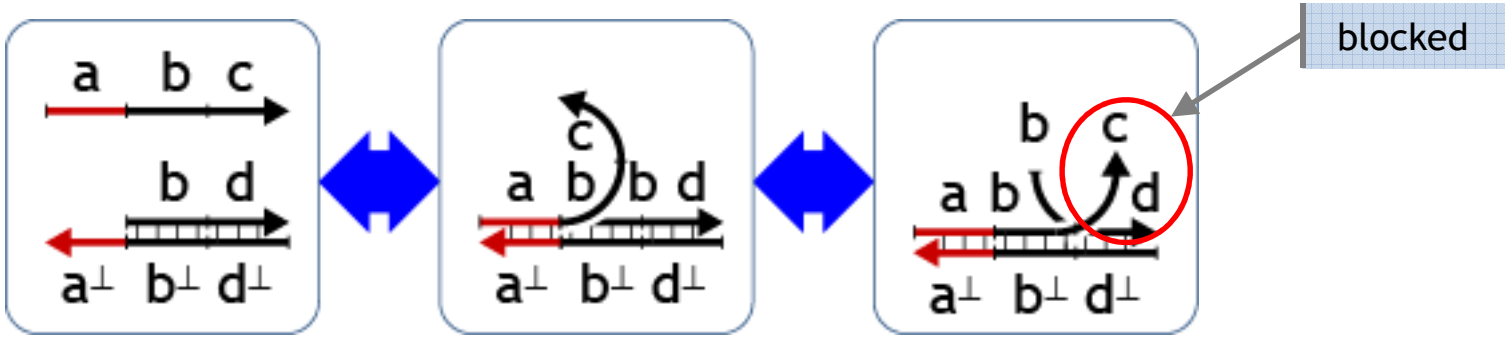


Irreversible Binding

Strand Displacement Reaction



Partial Match

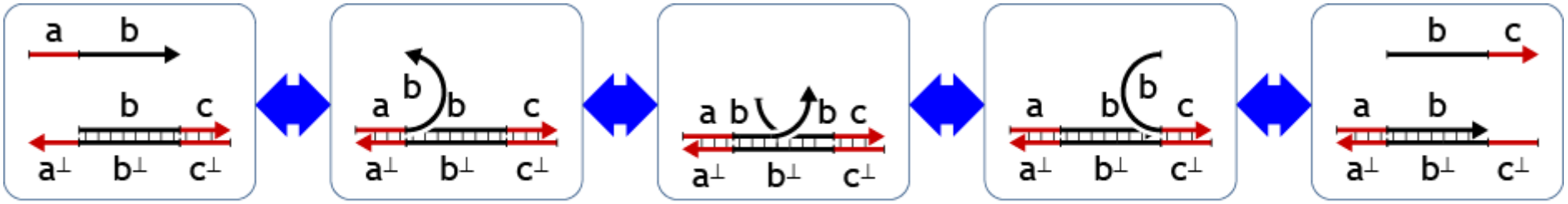


Reversible!

because the random walk is 'reflected' by the blockage

Irreversible match is determined by the toehold plus the branch migration region. That is, the toehold is a *cache* for the full address. The toehold must be short enough to guarantee reversible binding, but the branch migration region is practically unlimited. This means that the address space is unlimited.

Toehold Exchange Reaction



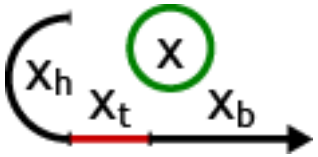
Toehold Exchange

Reversible

Signal Strand

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

(We work with a simpler version of their signal stands.)



x

x_h = history
x_t = toehold
x_b = binding

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

The combination x_t, x_b identifies the signal x.

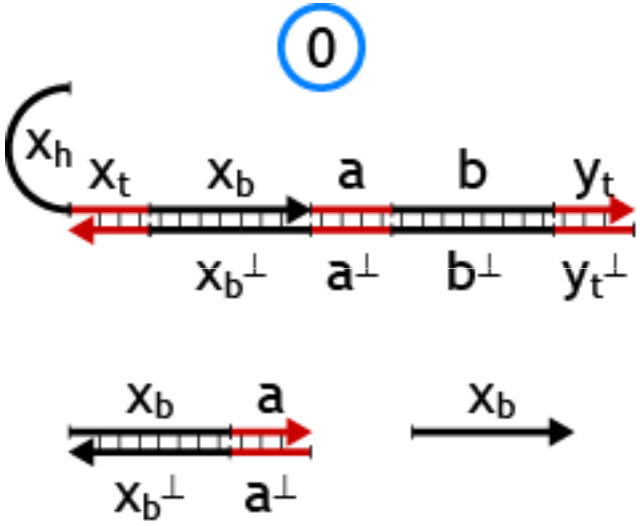
If $x \neq y$ then x and $y^{\perp}$ are not supposed to hybridize.

Signals and Gates

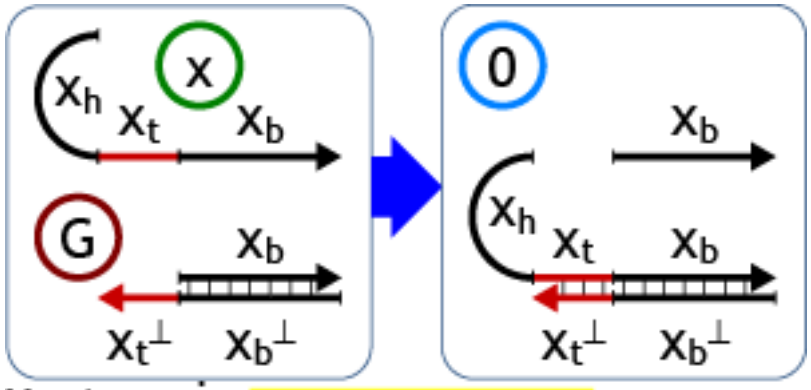
- Signals “x” are always positive strands
- Gates “x.y” always have a negative strand toehold and backbone.
  - that is, the input “x” is implicitly perp’ed
  - and the output “y” is another positive signal
- This separation helps the DNA realization, as one can use 3-letter alphabets (ATC/ATG) for each strand, minimizing secondary structure and entanglement.
- This way, by the way, we appear to give up Turing completeness, which is possible by freely using positive and negative strands. (Turing completeness has been demonstrated in DNA tiling systems.)
- (It is not clear how to achieve Turing completeness based on this signal/gate structure.)

Inert Systems

A system is considered *inert* (terminated) if it has no free toeholds.



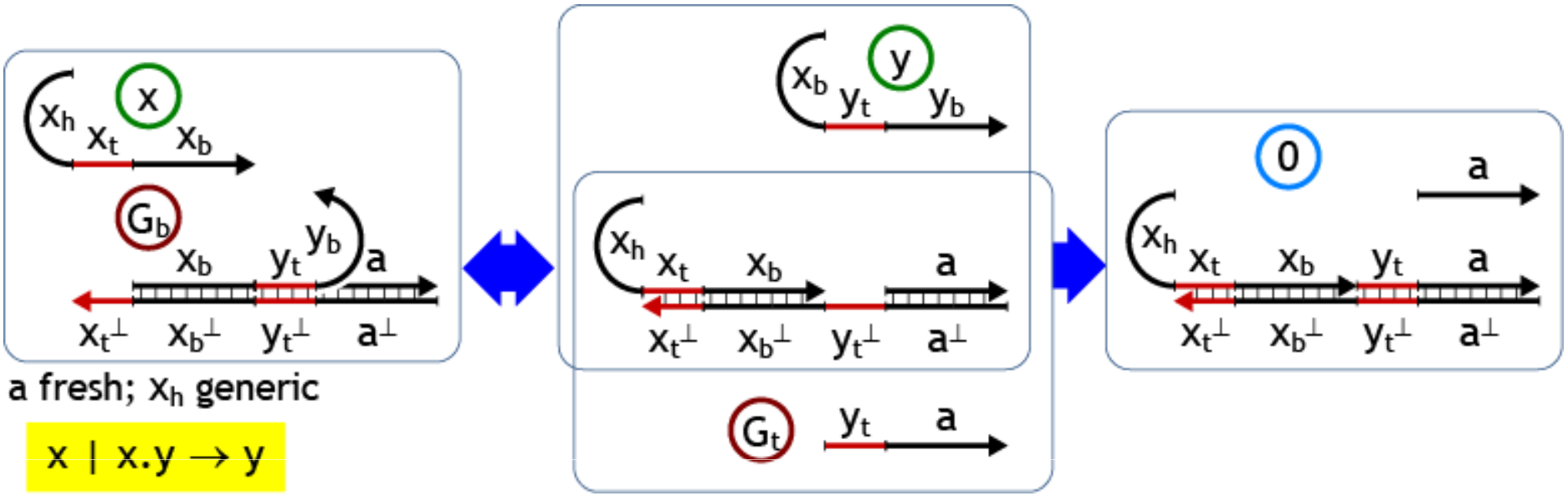
x.[] Annihilator Gate



This is just the strand displacement reaction, but seen as a gate absorbing a signal x and producing nothing (0 = inert).

Any history segment that is not determined by the gate structure is said to be ‘generic’ (can be anything).

x.y Transducer Gate

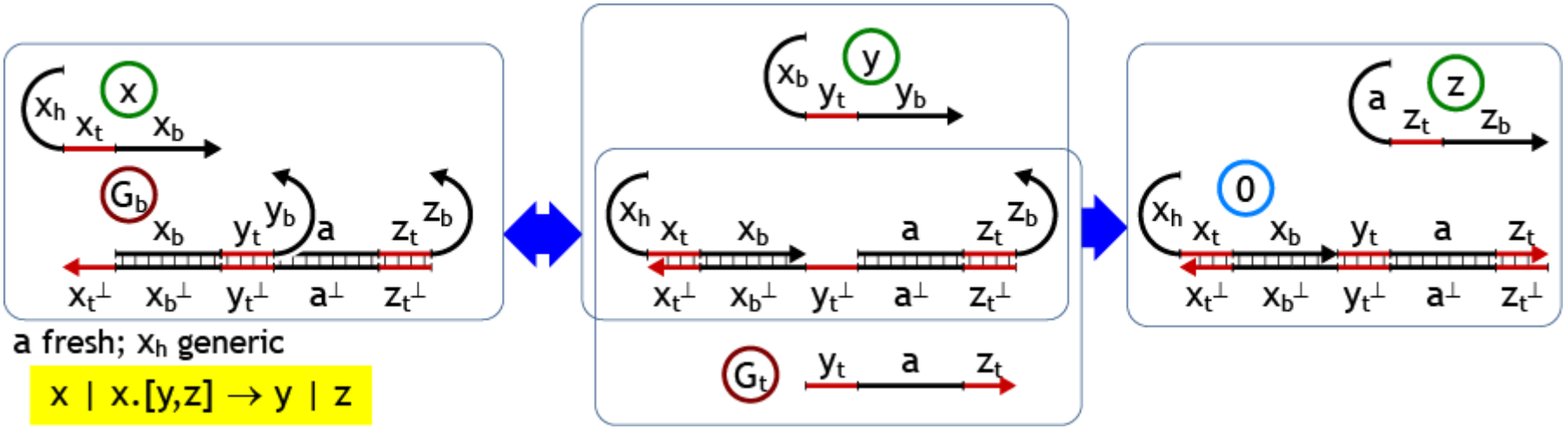


G_b,G_t (gate backbone and trigger) form the transducer.

Any history segment that is not determined by the gate structure is said to be ‘generic’ (can be anything).

Any gate segment that is not a non-history segment of an input or output signal is taken to be ‘fresh’ (globally unique for the gate), to avoid possible interferences.

x.[y,z] Fork Gate

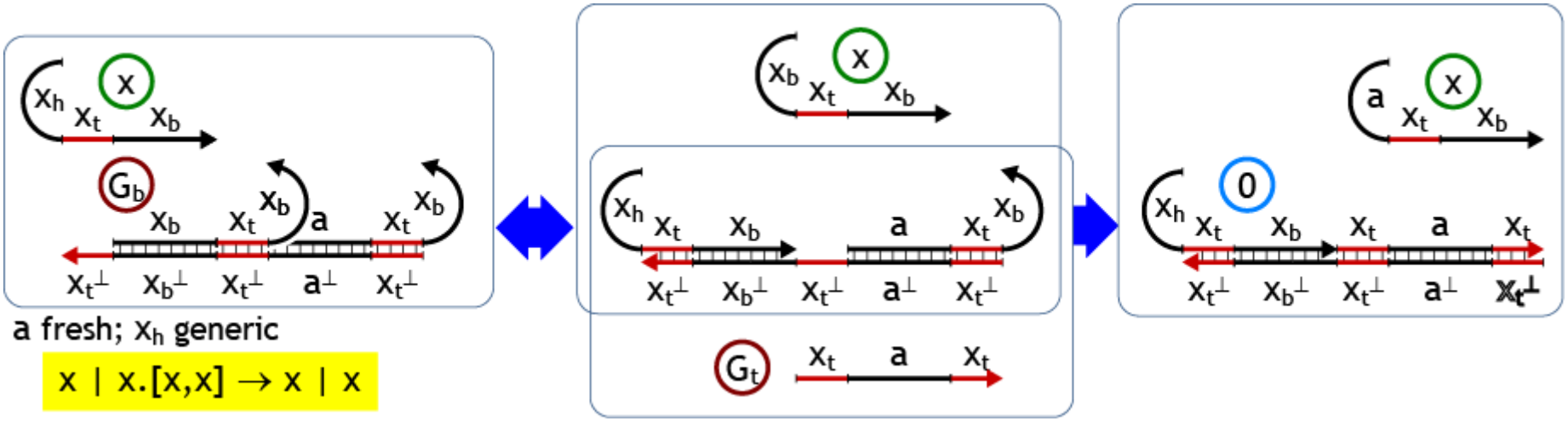


G_b,G_t (gate backbone and trigger) form the transducer.

Any history segment that is not determined by the gate structure is said to be ‘generic’ (can be anything).

Any gate segment that is not a non-history segment of an input or output signal is taken to be ‘fresh’ (globally unique for the gate), to avoid possible interferences.

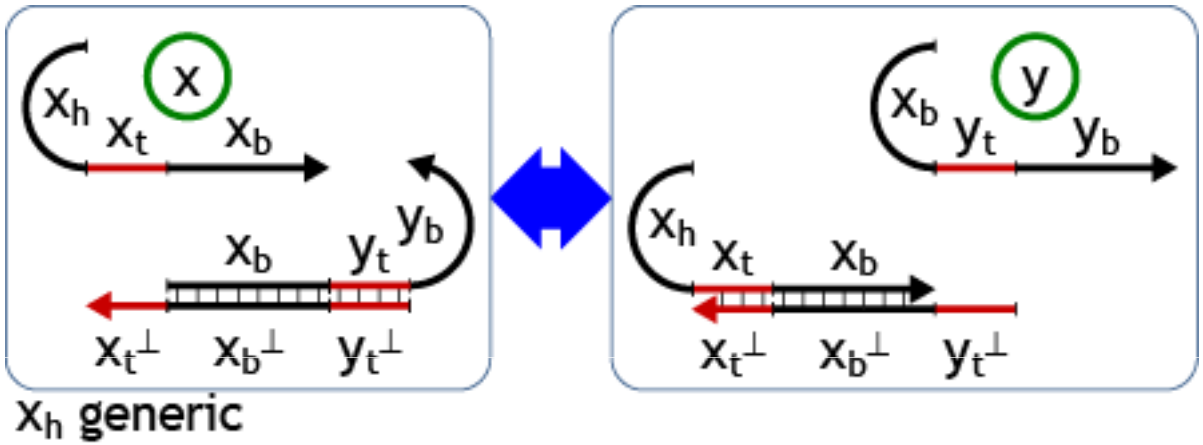
x.[x,x] Fork Gate



Autocatalyst

Adapter (a non-Gate)

Consider the reversible ‘first half’ of the transducer,
which works by toehold exchange:



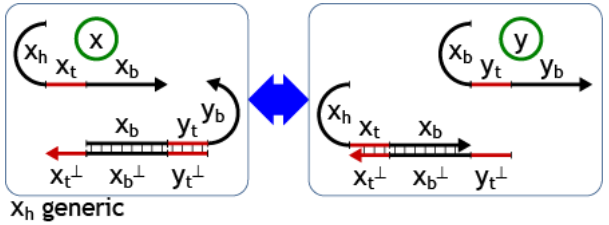
This has an interesting function: it adapts an x signal to a y signal:

if x is present then both x and y are available by reversibility

if y is consumed, then x is consumed.

If a gate produces an x and another gate expects a y, then we can (perhaps) use an adapter to connect them. Note that a full x to y transducer would not work as well as an adapter, because it would always remove x even if nobody wants y.

Non-gates



x_h generic

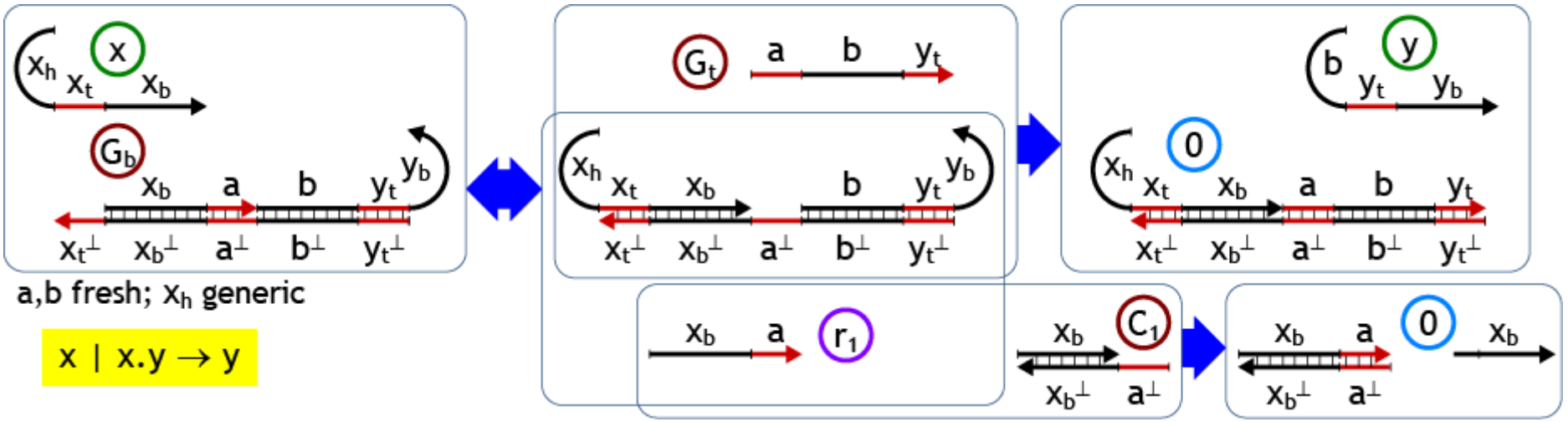
- However, the adapter is *not* a gate:
  - The inverse reaction works only for y's with x_b history. Gates must work on equivalence classes of signals, for any history. There is in fact no way to write the adapter as a gate reaction: x | adapt(x,y) → y | ?.
  - When y is consumed it leaves behind a non-inert components which eventually reduces the availability of x by speeding up the reverse reaction. These non-inert components should be removed.
  - Since both x and y are public signals, there is a possibility that some other part of the system may produce a y signal with x_b history, interfering with this adapter (slowing it down, and removing y's from somewhere else).
  - A proper adapter gate is instead (x.y | y.x), assuming a population of them.

- Although not a gate, an adapter can be used as part of a larger proper gate, like the Transducer, which:
  - works on equivalence classes of signals
  - does not leave active garbage around
  - but still admits interference on y (a alternative transducer is coming up).

Another Architecture

- We now start working with a slightly different gate structure.
  - The order of Trigger and Output is swapped.
- This is slightly more complex.
  - It requires a ‘garbage collection’ step.
- But it generalizes better to more complex gates.
  - Removes the worry about interference on x_b:y_t.
  - Join gates require garbage collection anyway.
- This results in a uniform structure for *all* gates.

x.y Transducer Gate

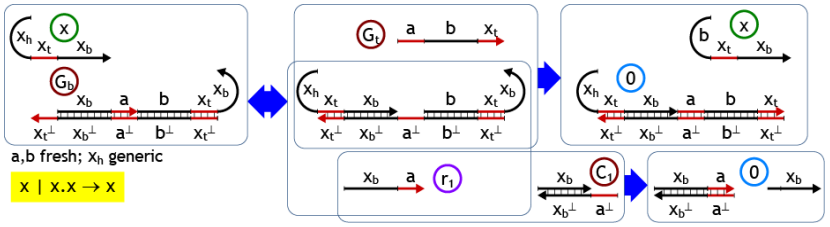


G_b,G_t,C_1 (gate backbone, trigger, collector) form the transducer.

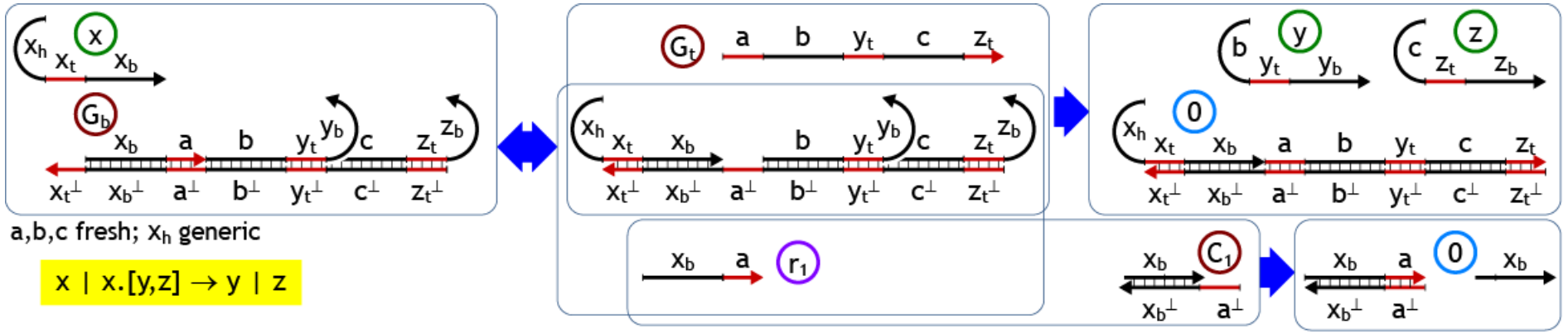
We need to collect the x_b:a strand to end up with an inert system.

If we do not collect it, it accumulates and *slows down* further transductions by pushing the reversible reaction to the left.

No problem with x.x:



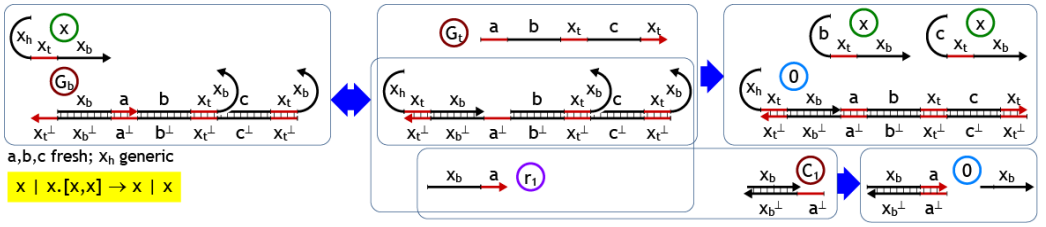
x.[y,z] Fork Gate



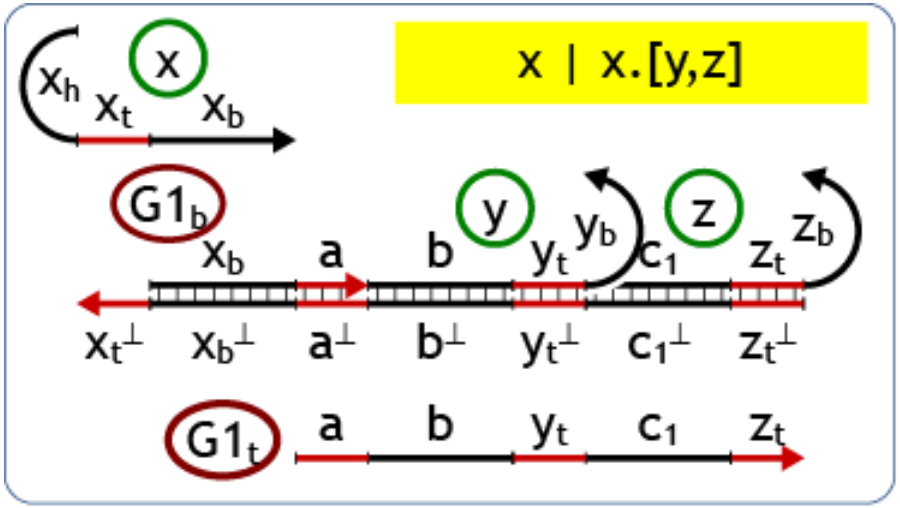
The triggering is now more uniform: all the outputs are released together.

This fork structure (although slightly more complex than the earlier fork) generalizes smoothly to multiple *inputs* as well, because in that case we cannot avoid a garbage collection phase.

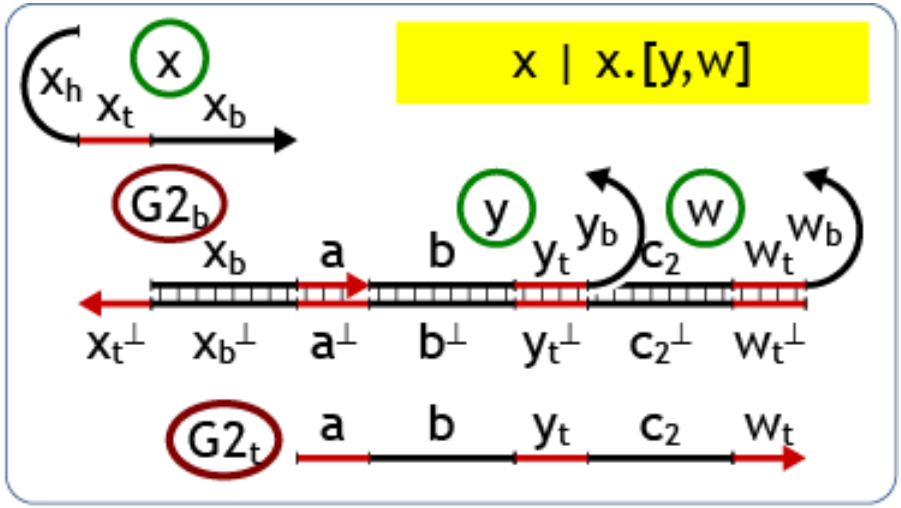
No problem with x.[x,x]:



Exercise 3: x.[y,z] | x.[y,w] Interference



~~a,b~~,c_1 fresh; x_h generic



~~a,b~~,c_2 fresh; x_h generic

- Suppose we 'forgot' to take a,b fresh, so they are shared by the two gates. Something goes horribly wrong from these initial conditions:

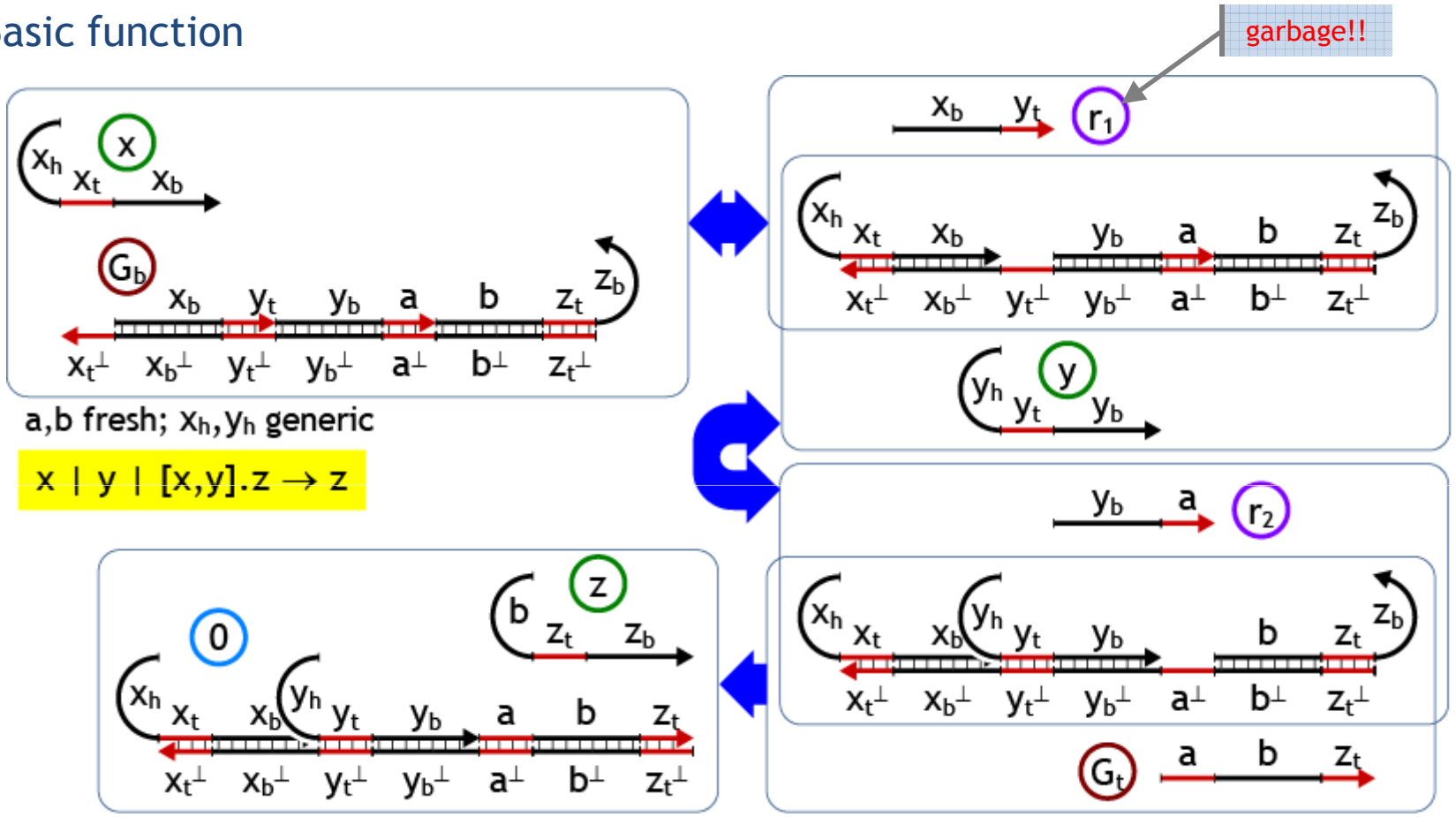
  x | x.[y,z] | x | x.[y,w]

  where x.[y,z] = $G1_b$,$G1_t$ and x.[y,w] = $G2_b$,$G2_t$

- What goes wrong?

[x,y].z Join Gate (function)

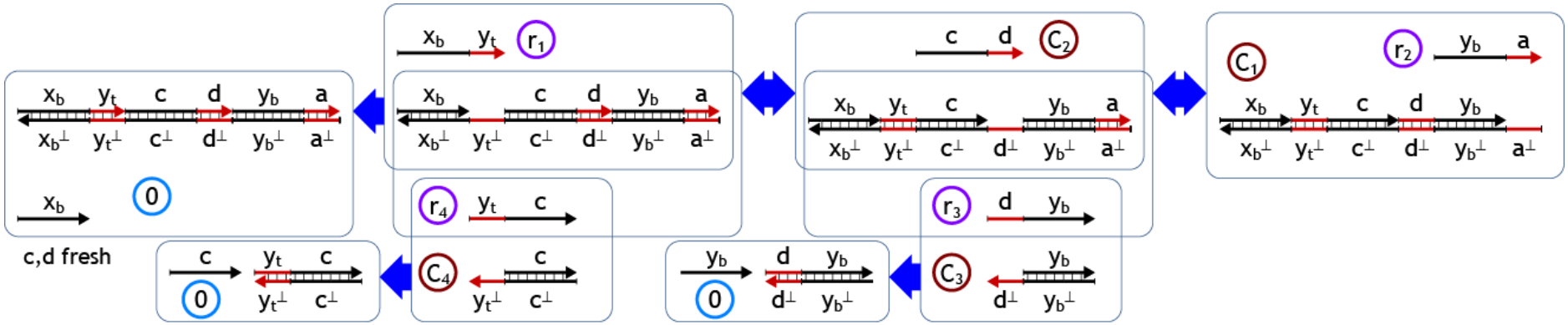
Basic function



Join can be implemented by a 'reversible-AND gate' taking two sequential inputs where the first one is reversible (Soloveichik Fig.3), so that x is not actually absorbed until y is found. The 'garbage' r_1 must not be collected until y is found: this is signaled by the release of r_2.

[x,y].z Join Gate (collection)

Garbage Collection

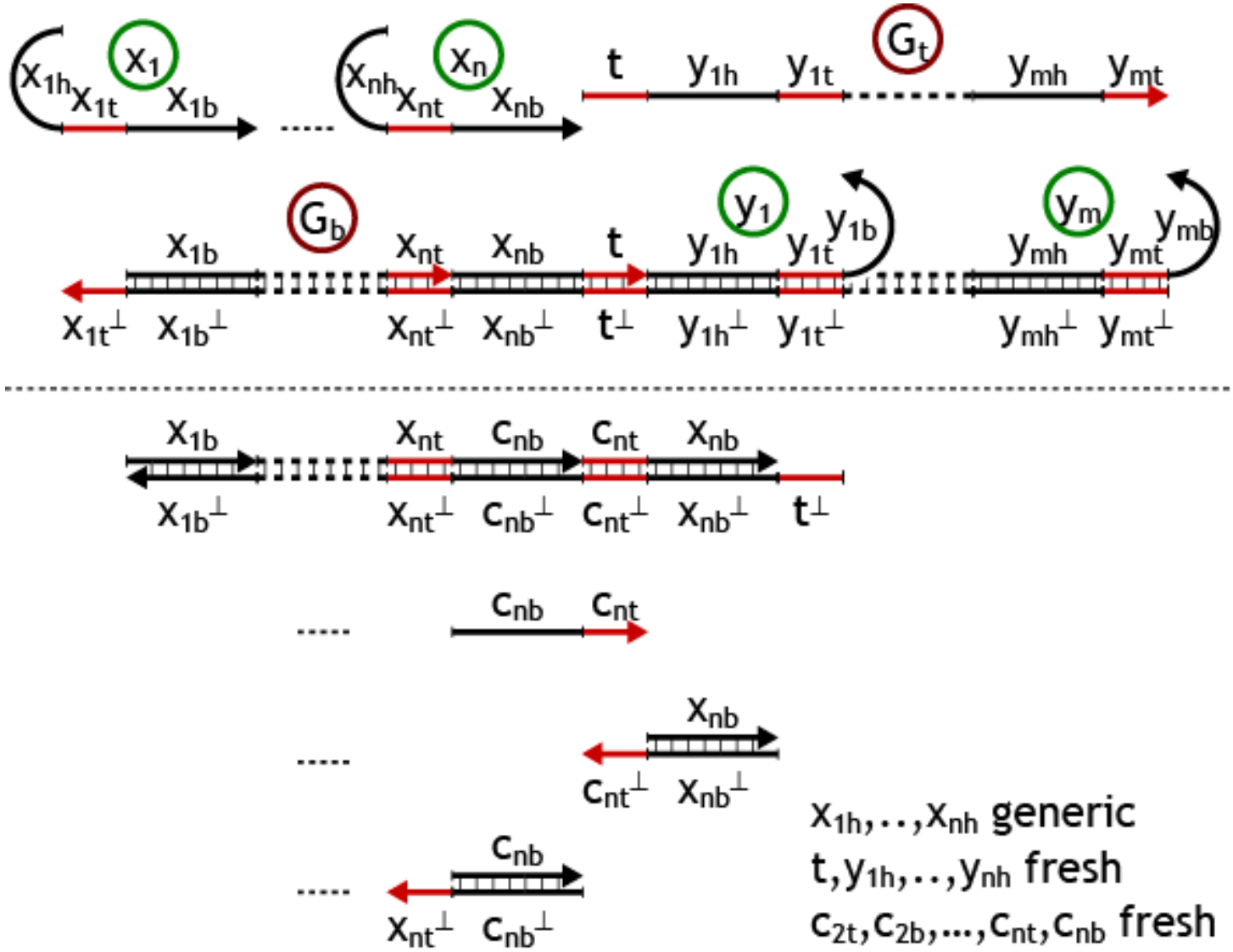


Garbage collection of r_1 is needed for join to work well. This is done by another reversible-AND between r_1 and r_2, triggered by the release of r_2. This second reversible-AND leaves garbage too (r_3, r_4), but this can be collected immediately, as we know by construction that both inputs r_1,r_2 are available and we need not wait to revert their bindings.

The extra intermediate c,d segments separate the r_1 binding from the r_2 binding. Without them, a segment y_t:y_b (instead of y_t:c and d:y_b) would be released: that is y!

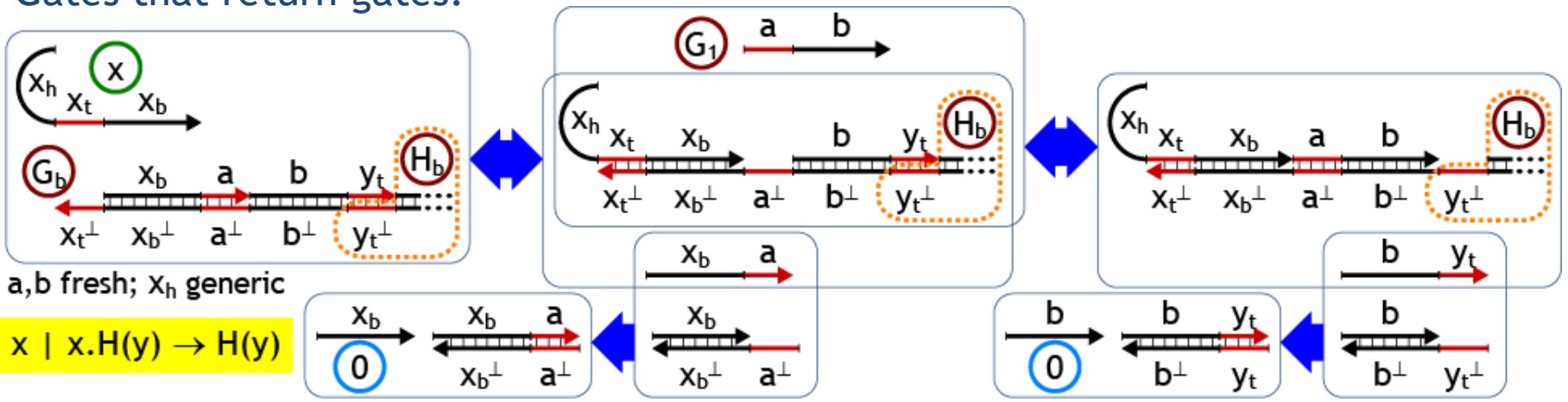
$[x_1,..,x_n].[y_1,..,y_m]$ General Join/Fork Gate

$x_1 \mid .. \mid x_n \mid [x_1,..,x_n].[y_1,..,y_m] \rightarrow y_1 \mid .. \mid y_m$

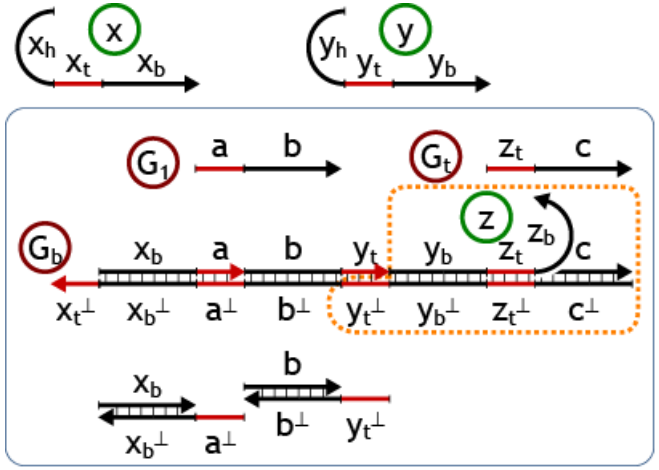


x.H(y) Curried Gates

Gates that return gates:



For example, x.y.z:



This means we can have gates of the form:

$$G ::= [x_1,..,x_n].[x'_1,..,x'_m] : [x_1,..,x_n].G$$

$n \geq 1,\ m \geq 0$

Exercise 4: x.y.z | [x,y].w Interference

[David Soloveichik]

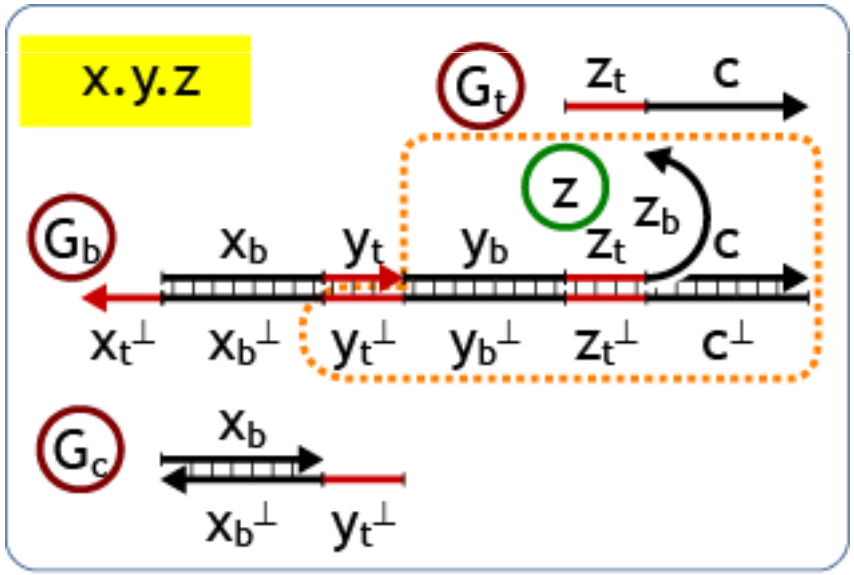
Consider curried gates without the a,b segments (example below): instead of releasing x_b,a and b,y_t segments, they would release x_b,y_t.

But that is exactly the strand r_1 of an [x,y].w gate: the strand that reverts the x input. This definitely causes an interference between x.y.z and [x,y].w.

Find a situation where the presence (x.y.z as below) or absence (x.y.z as in previous slide) of this interference causes different outcomes.

Hint: it changes outcome *probability*.

Note: the a,b segments prevent the interference.



c fresh; x_h,y_h generic
(without the a,b segments)

Summary

- DNA strand displacement technology provides a way of implementing abstract signal transducer networks.
- Fork gates and Join gates are the main components.
- How powerful it this style of computation?

Combinatorial Strand Algebra

Formalizing a Process Algebra

- A term syntax (almost always including parallel composition):
  - $P ::= \ldots \mid P \mid P \mid \ldots$
- A structural congruence relation (chemical mixing):
  - $P \equiv Q$ (e.g. commutative monoid rules of '|')
- A reduction relation (chemical reactions):
  - $P \rightarrow Q$
- Standard rules to connect the two, which have a 'chemical' meaning:
  - 'Dilution': $P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R$
  - 'Well-mixing': $P \equiv P', P' \rightarrow Q', Q' \equiv Q \Rightarrow P \rightarrow Q$
- Various equivalences (e.g. bisimulation) derived from the above.
- Algebraic laws proved (not taken as axioms) from the equivalences.

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 *signal*, and [..].[..] is a *gate*:

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

x		is a *strand*
$x_1.x_2$	$\stackrel{\text{def}}{=} [x_1].[x_2]$	is a *sequence gate*
$x.[x_1,..,x_m]$	$\stackrel{\text{def}}{=} [x].[x_1,..,x_n]$	is a *fork gate*
$[x_1,..,x_n].x$	$\stackrel{\text{def}}{=} [x_1,..,x_n].[x]$	is a *join gate*

0	is *inert*
P|P	is *parallel composition* of signals and gates
P*	is a *population* (multiset) of signals and gates

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

Structural Congruence for $\mathcal{P}$

$P \equiv P$	equivalence
$P \equiv P' \Rightarrow P' \equiv P$	
$P \equiv P', P' \equiv P'' \Rightarrow P \equiv P''$	
$P \equiv P' \Rightarrow P \mid P'' \equiv P' \mid P''$	congruence
$P \equiv P' \Rightarrow P^* \equiv P'^*$	
$P \mid 0 \equiv P$	diffusion
$P \mid P' \equiv P' \mid P$	
$P \mid (P' \mid P'') \equiv (P \mid P') \mid P''$	
$P^* \equiv P^* \mid P$	population
$0^* \equiv 0$	
$(P \mid P')^* \equiv P^* \mid P'^*$	
$P^{**} \equiv P^*$	

Reduction for $\mathcal{P}$

$$x_1 \mid .. \mid x_n \mid [x_1,..,x_n].[x_1',..,x'_m] \rightarrow x'_1 \mid .. \mid x'_m \qquad \text{Gate}$$

$$P \rightarrow P' \quad \Rightarrow \quad P \mid P'' \rightarrow P' \mid P'' \qquad \text{Dilution}$$

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

Technically, the fully parenthesized Gate rule is:

$x_1 \mid (.. \mid (x_n \mid [x_1,..,x_n].[x'_1,..,x'_m])..) \rightarrow x'_1 \mid (.. \mid (x'_m)..)$

but we have structural congruence to reassociate.

Examples

$x_1 \mid x_1.x_2 \rightarrow x_2$

$x_1 \mid x_1.x_2 \mid x_2.x_3 \rightarrow\!\!\rightarrow x_3$

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

$x_1 \mid x_1.x_2 \mid x_1.x_3 \rightarrow x_2 \mid x_1.x_3$

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

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

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

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

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

High(er)-Level Languages

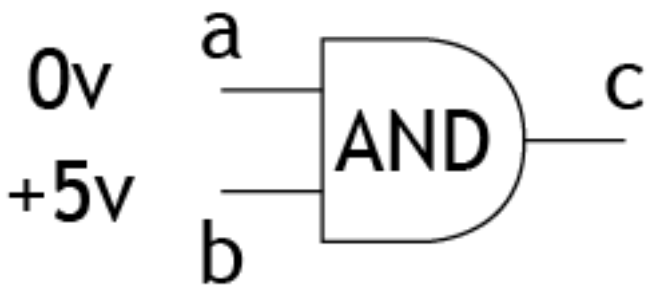
- We now have an intermediate language: the combinatorial strand algebra
  - It can be compiled "directly" to DNA following [Soloveichik et al.]

- But we really want to compile "high-level languages". Such as:
  - Boolean Networks
  - Petri Nets
  - Finite State Automata
  - Finite Stochastic Reaction Networks (Chemistry) [Soloveichik et al.]
  - Interacting Automata
  - π-calculus (no, not in the current strand algebra)

- And also
  - Higher-level strand algebras, which may form more convenient intermediate languages.
  - Such as the *Nested Strand Algebra* (with nested expressions).

Exercise 5: Boolean Networks

Boolean Networks to Strand Algebra



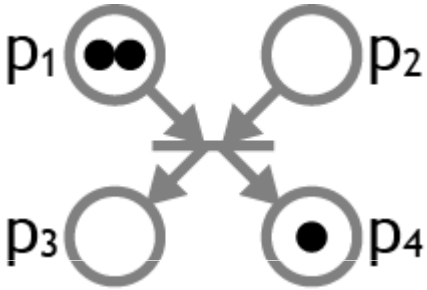
Find an encoding of Boolean networks in Strand Algebra.

It's enough to show how to encode and AND gate that takes Boolean signals on a,b wires and produces a Boolean signal on the c wire, and a NOT gate. (Their combination, a NAND gate, is a universal gate.)

Petri Nets

Petri Nets to Strand Algebra

Transitions as Gates
Markings as Signals

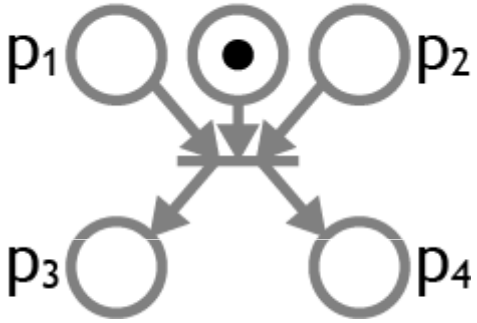


$([p_1,p_2].[p_3,p_4])^* \mid p_1 \mid p_1 \mid p_4$

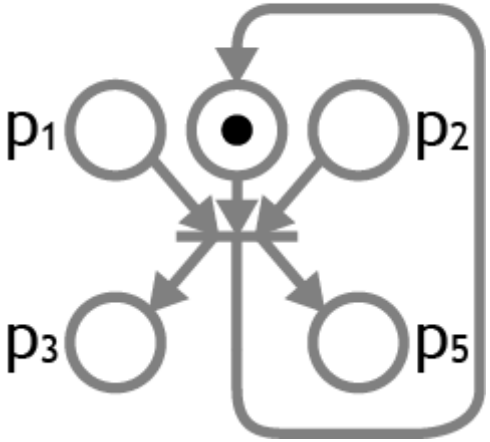
Strand Algebra to Petri Nets

Gates as Transitions

$[p_1,p_2].[p_3,p_4]$

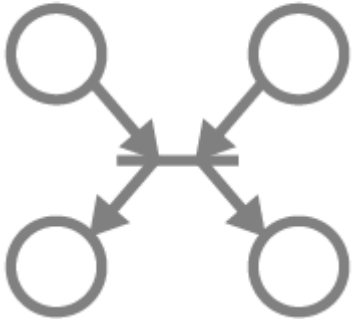


$([p_1,p_2].[p_3,p_4])^*$



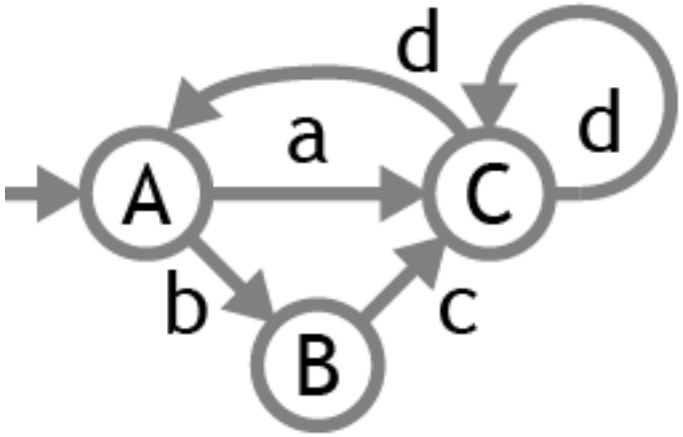
or

$([p_1,p_2].[p_3,p_4])^*$



Finite State Automata

FSA to Strand Algebra



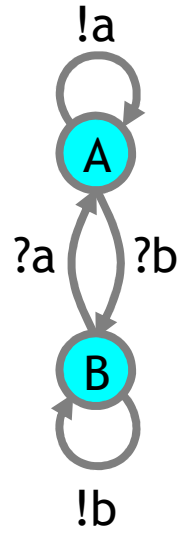
$$
\begin{aligned}
&([A,a].[C,\tau])^* \mid \\
&([A,b].[B,\tau])^* \mid \\
&([B,c].[C,\tau])^* \mid \\
&([C,d].[C,\tau])^* \mid \\
&([C,d].[A,\tau])^* \mid \\
&A \mid \tau
\end{aligned}
$$

Input strings

a,b,c,d

$$
\begin{aligned}
&\tau\ .[a,\sigma_1] \mid \\
&[\sigma_1,\tau].[b,\sigma_2] \mid \\
&[\sigma_2,\tau].[c,\sigma_3] \mid \\
&[\sigma_3,\tau].\ d
\end{aligned}
$$

Interacting Automata (first try)

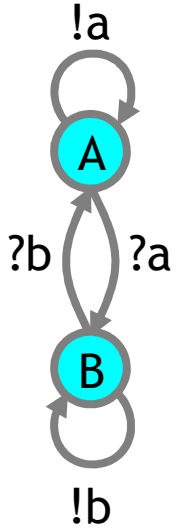


Interacting Automata to Strand Algebra

Groupies	*Strand*(Groupies)
A = !a;A ⊕ ?b;B	on a: ([B,A].[A,A])* |
B = !b;B ⊕ ?a;A	on b: ([A,B].[B,B])*

Map each possible interaction to a Join

Strand(E) = *Parallel*(⟪…⟫ means multisets
⟪ (X.[P])* *s.t.* ∃i. E.X.i = τ;P ⟫ ∪
⟪ ([X,Y].[P,Q])* *s.t.* ∃i,j,c. E.X.i = ?c;P and E.Y.j = !c;Q ⟫)



Celebrities	*Strand*(Celebrities)
A = !a;A ⊕ ?a;B	on a: ([A,A].[B,A])* |
B = !b;B ⊕ ?b;A	on b: ([B,B].[A,B])*

However, Interacting Automata are stochastic!

Summary

- DNA strand displacement gates can be formalizes as Strand Algebra.
- Strand Algebra is equivalent to place/transition Petri nets, but:
  - Has a compositional syntax.
  - Keeps track of the (DNA) resources that are consumed during computation.
- While not Turing complete, Strand algebra can embed many common and useful formalisms:
  - Boolean Networks
  - Petri Nets
  - Finite State Automata over multisets and over strings
  - Finite State Transducers over multisets, and from string to multisets (not shown) (not clear how to transduce to strings)
  - Interacting Automata, at least qualitatively.

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*:

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

- 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 choose $(x_1 \mid \ldots \mid x_n \mid [x_1,\ldots,x_n].[y_1,\ldots,y_m])) \times g_n$

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

  - If $P = x^n \mid ([x,x].z)^p$, the propensity of the next $x \mid x \mid [x,x].z \to z$ 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 $x+x \to \ldots$. 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 choose $(x_0 \mid x_0.[x_1,\ldots,x_n].[y_1,\ldots,y_m])) \times g_1$

Global Transition

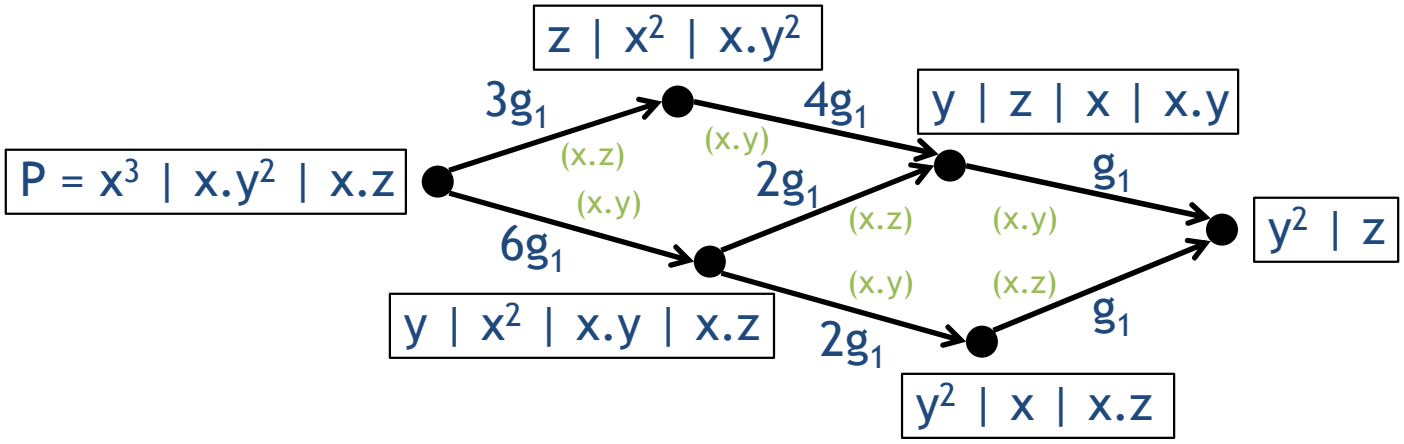
- A *global transition* $P \rightarrow^{r} P'$ of a global state P to a next global state P' with propensity r is then defined as:

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

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

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

- 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.

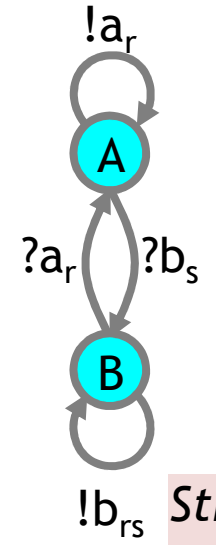
- For a stochastic reaction rate r of an n-ary reaction we use a constant-size population of size r/g_n (assume $r>>g_n$; otherwise scale up the rates to obtain large-enough populations):

  1) $A \rightarrow^r B_1 + \ldots + B_m \quad \Rightarrow \quad (A.[B_1,\ldots,B_m])^{=r/g_1}$
  2) $A_1+A_2 \rightarrow^r B_1 + \ldots + B_m \quad \Rightarrow \quad ([A_1,A_2].[B_1,\ldots,B_m])^{=r/g_2}$
  3) $A+A \rightarrow^r B_1 + \ldots + B_m \quad \Rightarrow \quad ([A,A].[B_1,\ldots,B_m])^{=r/g_2}$

- The propensities match:

  1) $P = A^n$: (P choose A)×r = n×r = (P choose A)×r/g_1×g_1
  2) $P = A_1^n+A_2^m$: (P choose A_1,A_2)×r = n×m×r = (P choose A_1,A_2)×r/g_2×g_2
  3) $P = A^n$: (P choose A,A)×r = n×(n-1)/2×r = (P choose A,A)×r/g_2×g_2

Interacting Automata to Strand Algebra



Groupies

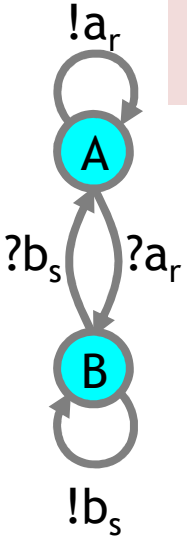
$A = !a_r.A \oplus ?b_s.B$
$B = !b_s.B \oplus ?a_r.A$

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.* $\exists i.\ E.X.i = \tau;P$ ⟫ ∪
⟪ $([X,Y].[P,Q])^{=r/g_2}$ *s.t.* $X \neq Y$ and $\exists i,j,c.\ E.X.i = ?c_r;P$ and $E.Y.j = !c;Q$ ⟫ ∪
⟪ $([X,X].[P,Q])^{=2r/g_2}$ *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) $A^n \mid B^m \rightarrow^{n \times m \times r} A^{n+1} \mid B^{m-1}$

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

Strand(Groupies)

$P = ([B,A].[A,A])^{=r/g_2} \mid ([A,B].[B,B])^{=s/g_2}$

$A^n \mid B^m \mid P \rightarrow^{\sim n \times m \times r/g_2 \times g_2} A^{n+1} \mid B^{m-1} \mid \sim P$

$A^n \mid B^m \mid P \rightarrow^{\sim n \times m \times s/g_2 \times g_2} A^{n-1} \mid B^{m+1} \mid \sim P$

Celebrities

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

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

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

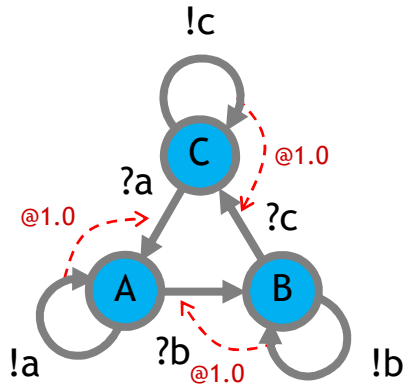
Strand(Celebrities)

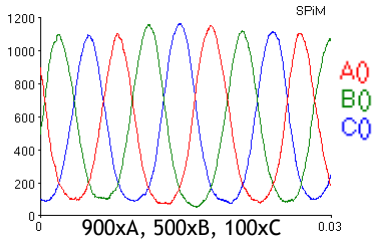
$P = ([A,A].[B,A])^{=2r/g_2} \mid ([B,B].[A,B])^{=2s/g_2}$

$A^n \mid B^m \mid P \rightarrow^{\sim n \times (n-1)/2 \times 2r/g_2 \times g_2} A^{n-1} \mid B^{m+1} \mid \sim P$

$A^n \mid B^m \mid P \rightarrow^{\sim m \times (m-1)/2 \times 2s/g_2 \times g_2} A^{n+1} \mid B^{m-1} \mid \sim P$

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())
```

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

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

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

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

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

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

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

DNA

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

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:
  - $x_1.x_2.x_3 \;=\; x_1.x_0 \;|\; [x_0,x_2].x_3$
  - 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 $n\mathcal{P}$

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.

$n\mathcal{P}$ 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,
$\mathcal{X}_{\geq l}$	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:

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

$x \mid U(?x.P^*) \rightarrow Z \mid (Z.[Y,Z] \mid U(Y,P))^*$
$\equiv Z \mid Z.[Y,Z] \mid U(Y,P) \mid (Z.[Y,Z] \mid U(Y,P))^*$
$\rightarrow Y \mid U(Y,P) \mid Z \mid (Z.[Y,Z] \mid U(Y,P))^*$
…

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

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

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.

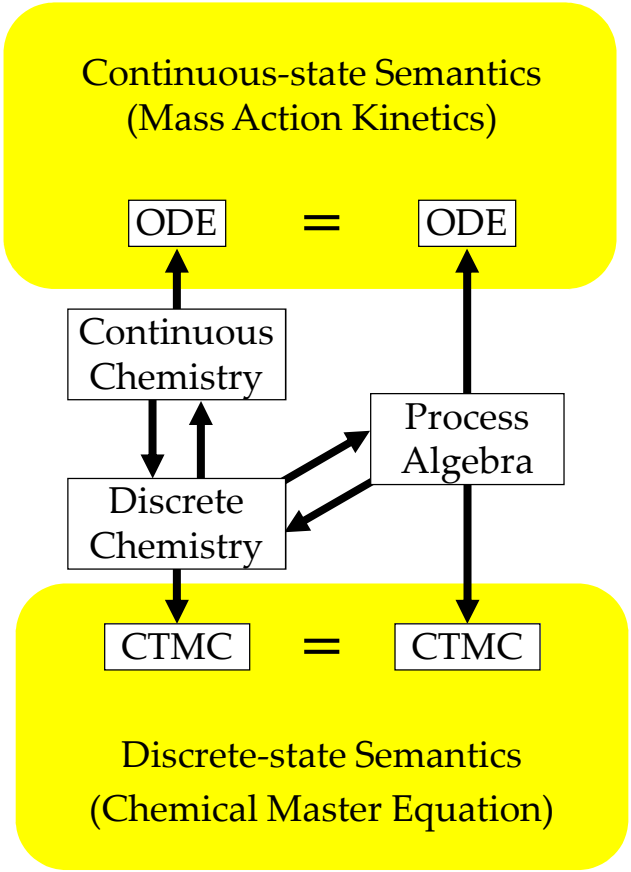
Standard questions can be asked: what happens if somebody inserts three coins very quickly? Or somebody presses refund twice? Etc.

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

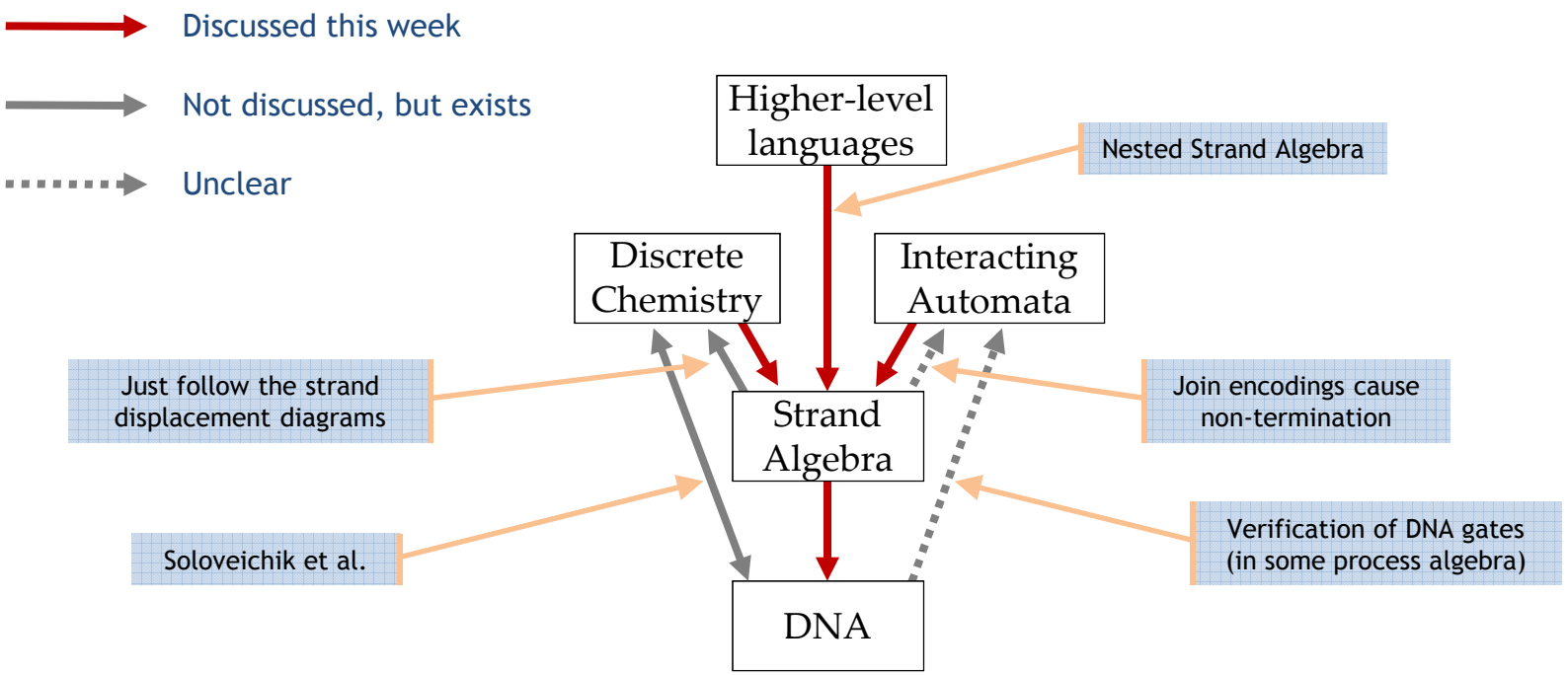


These diagrams commute via appropriate maps.

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

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

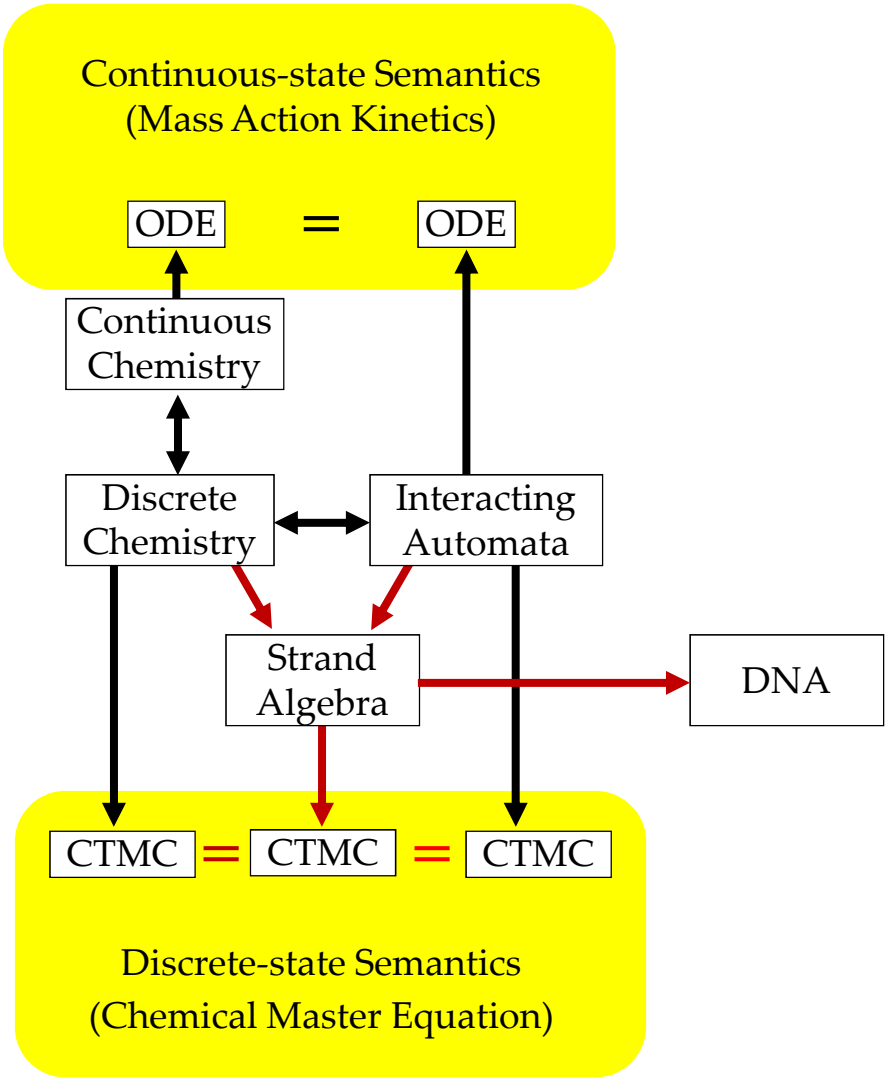
Automata as Molecules



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.