Molecular Programming

Luca Cardelli

Microsoft Research

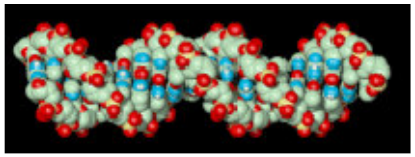
Redmond, 2009-06-29

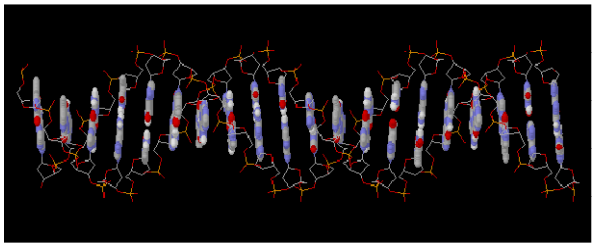
http://LucaCardelli.name

DNA Basics

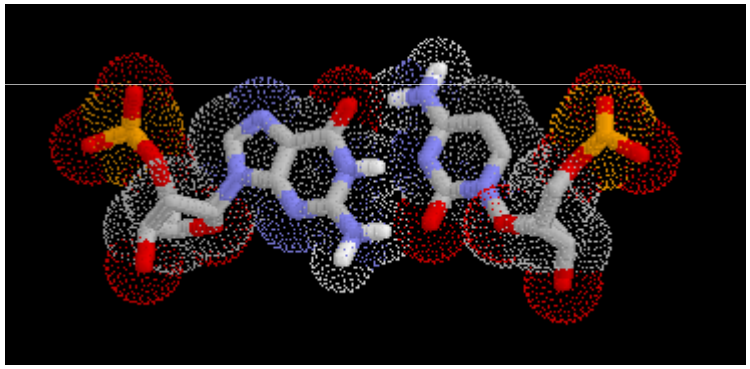
ACGT

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

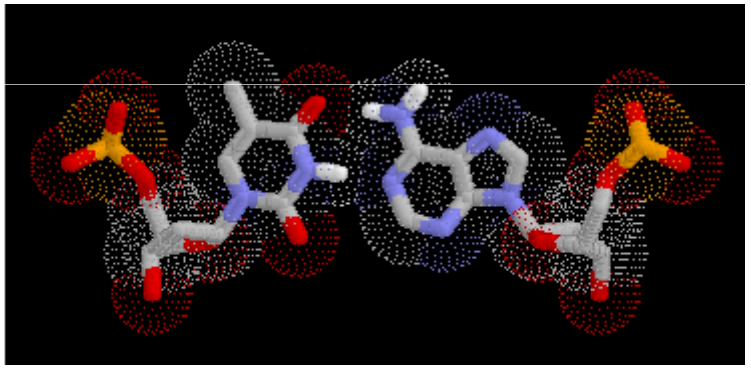




Sequence of Base Pairs



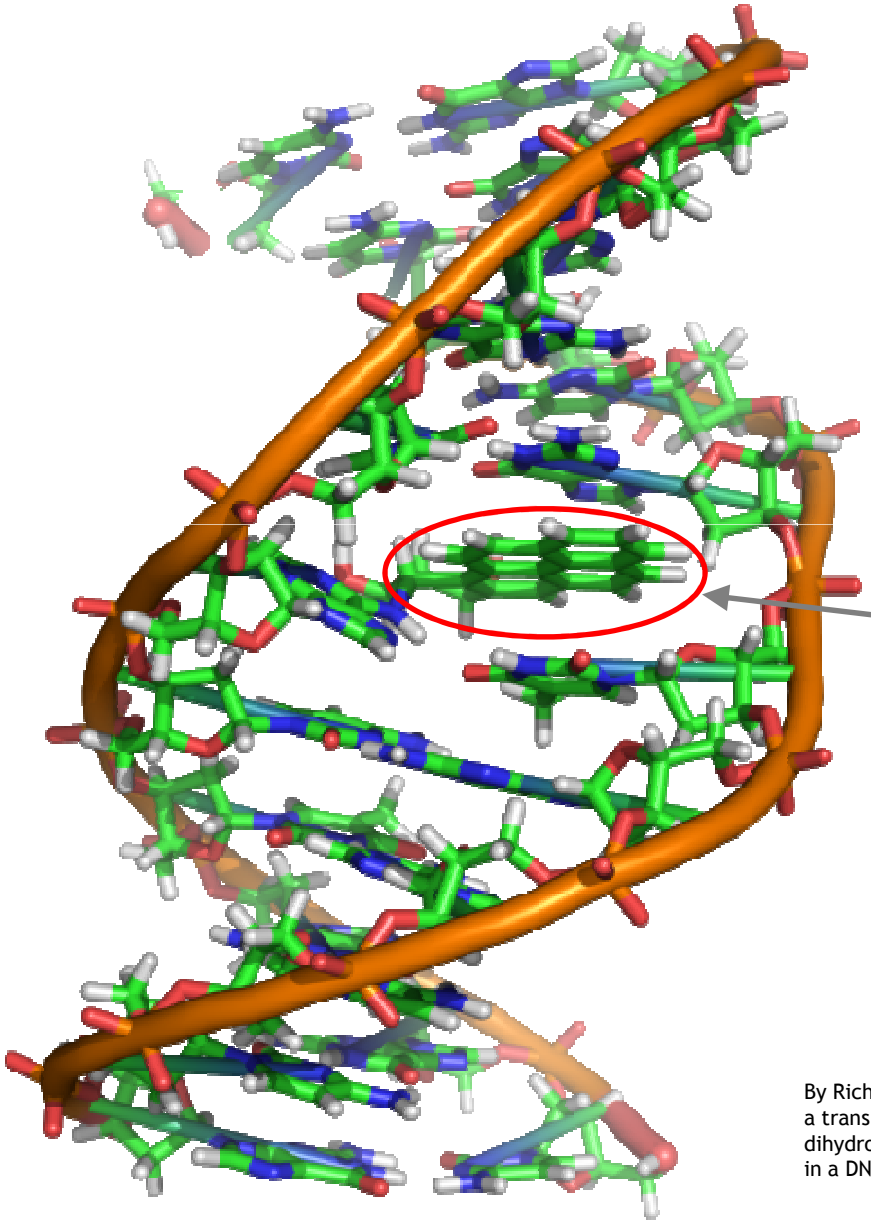
GC Base Pair
Guanine-Cytosine



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

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

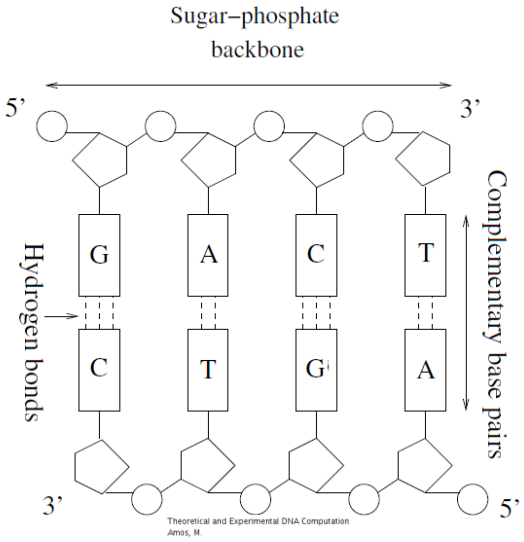
3' end

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



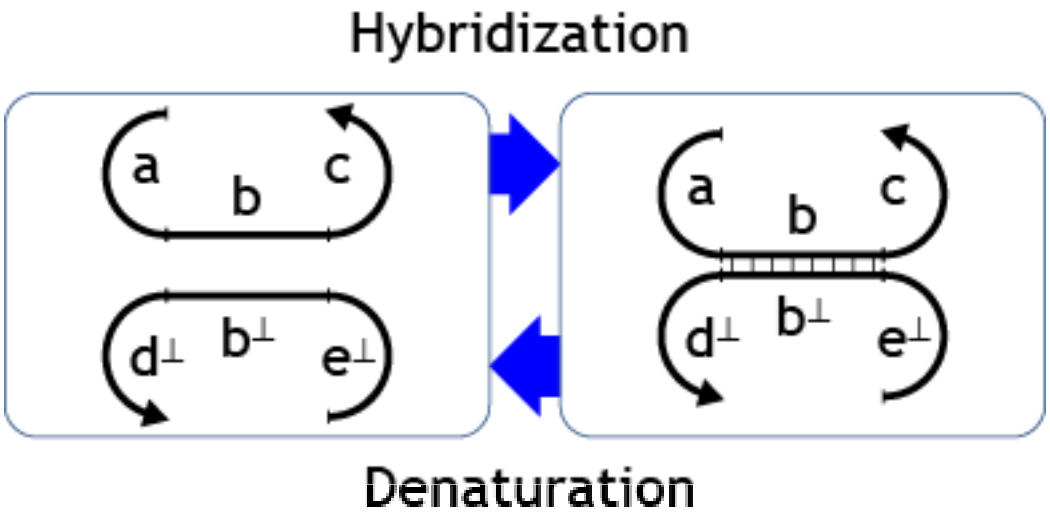
Hence $(G:A:C:T)^{\perp} = A:G:T:C = T^{\perp}:C^{\perp}:A^{\perp}:G^{\perp}$

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.

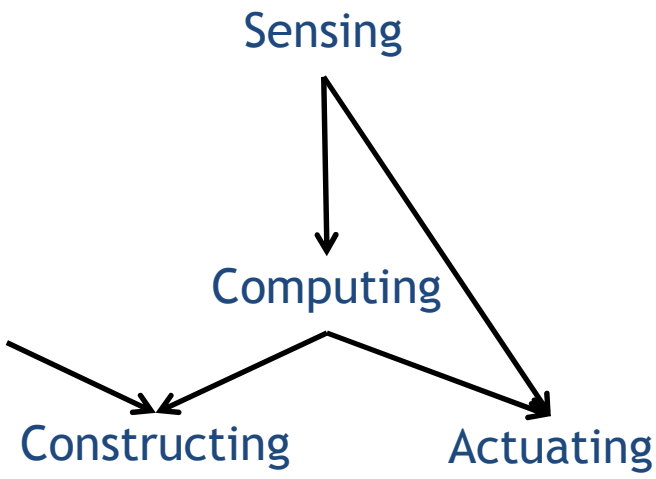
The direction of the reaction (or in general the equilibrium between the two states) is determined by a number of factors, e.g. temperature.

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

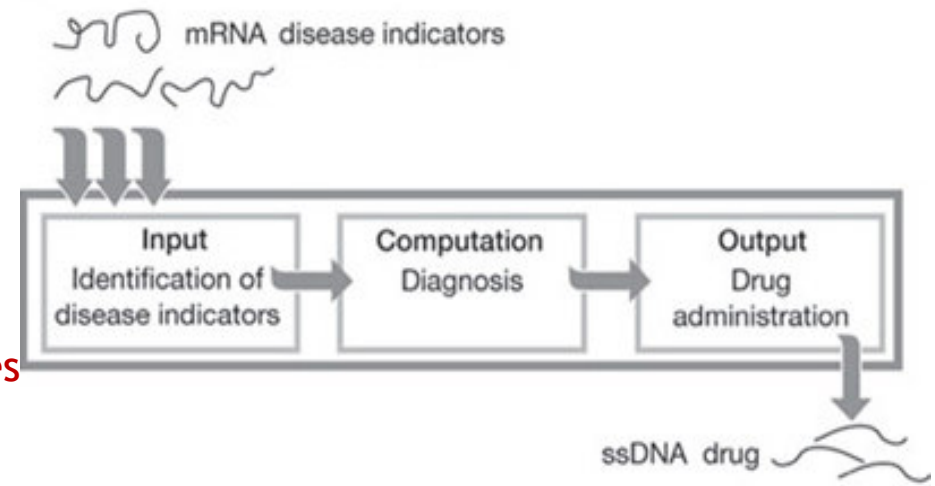
DNA Nanotechonology

Nano Tasks

- Sensing
  - Binding to specific molecules
- Computing
  - Analog: Signal Filtering or Amplification
  - Digital: Logical gates
- Actuating
  - Releasing molecules
  - Producing forces
- Constructing
  - By self-assembly
  - Or under 'program' control



- Nucleic Acids (DNA/RNA)
  - Probably the only materials that can perform all these functions.
  - Technology relatively well developed.
  - They can interface to biological entities

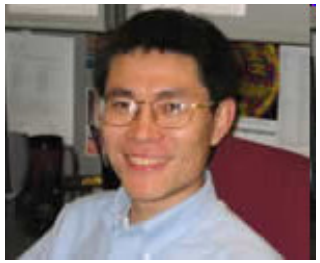


Y. Benenson *et al.*, *An autonomous molecular computer for logical control of gene expressionNature* 429, 423-429 (2004)

DNA as a Building Material

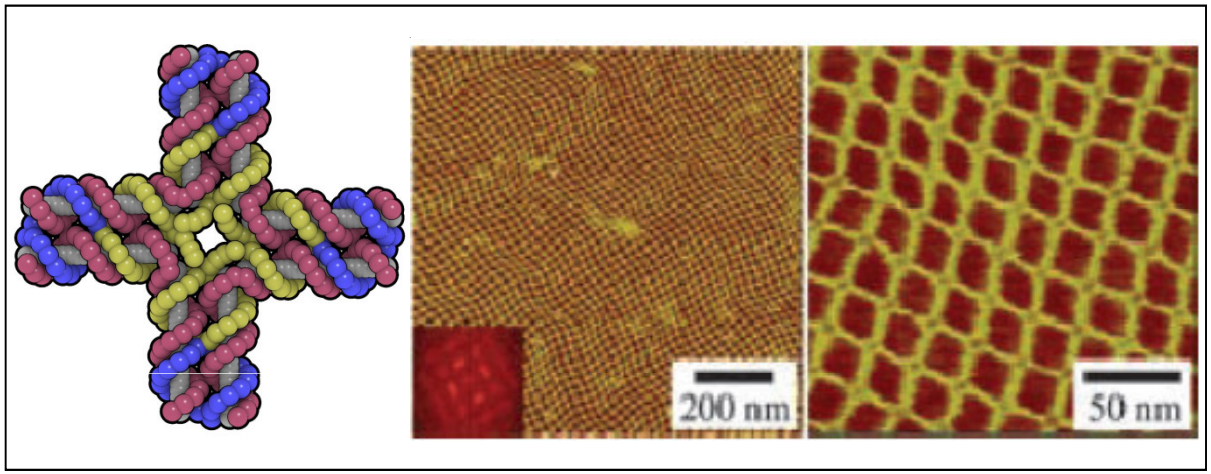
Slides by John Reif

2D DNA Lattices

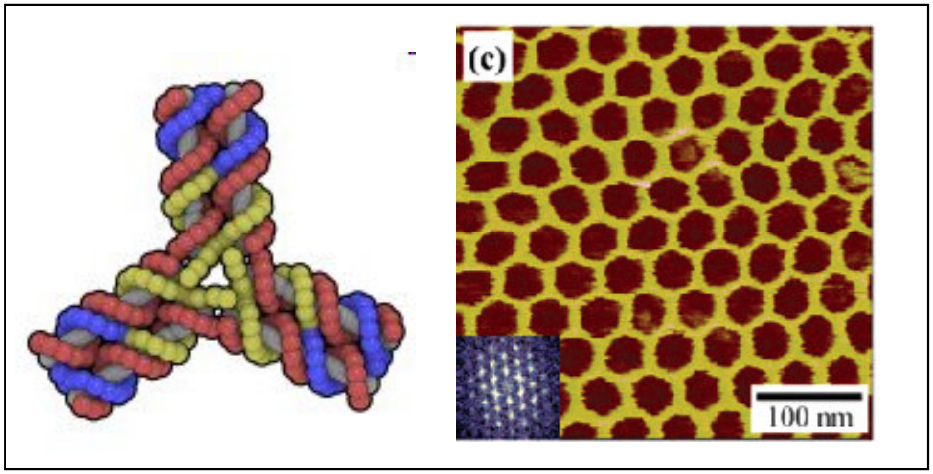


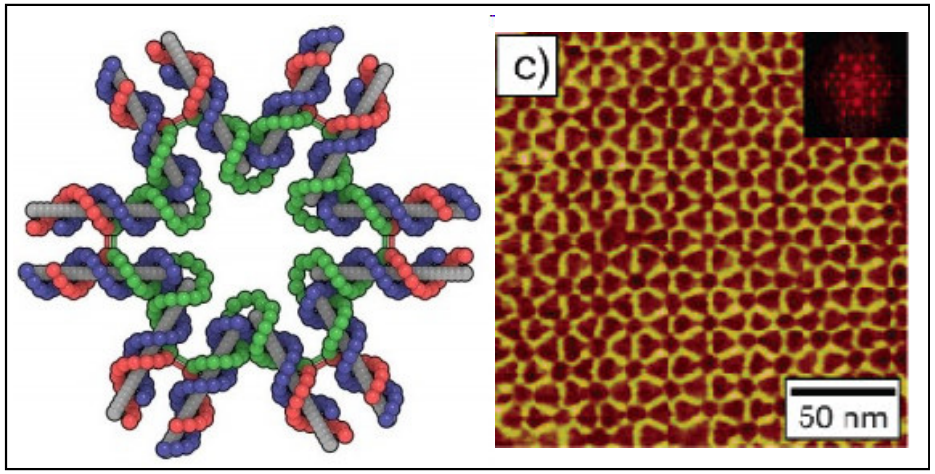
Chengde Mao

Purdue University, USA



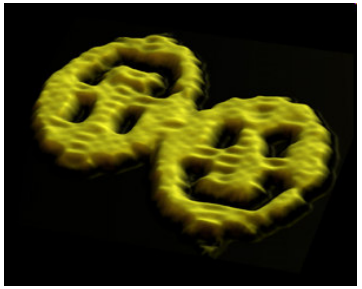
N-point Stars

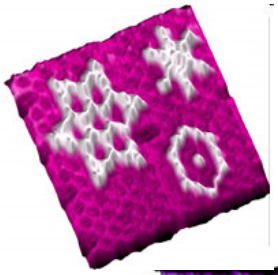


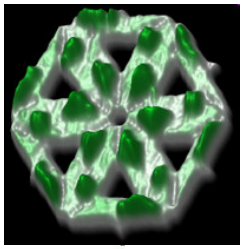


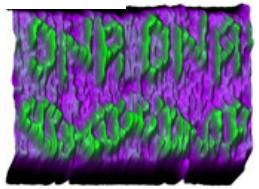
DNA Origami

Nature, 2006



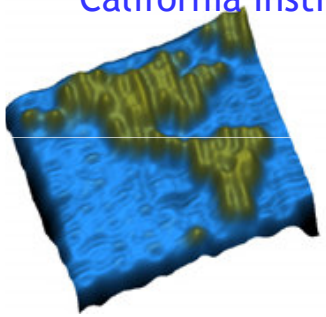


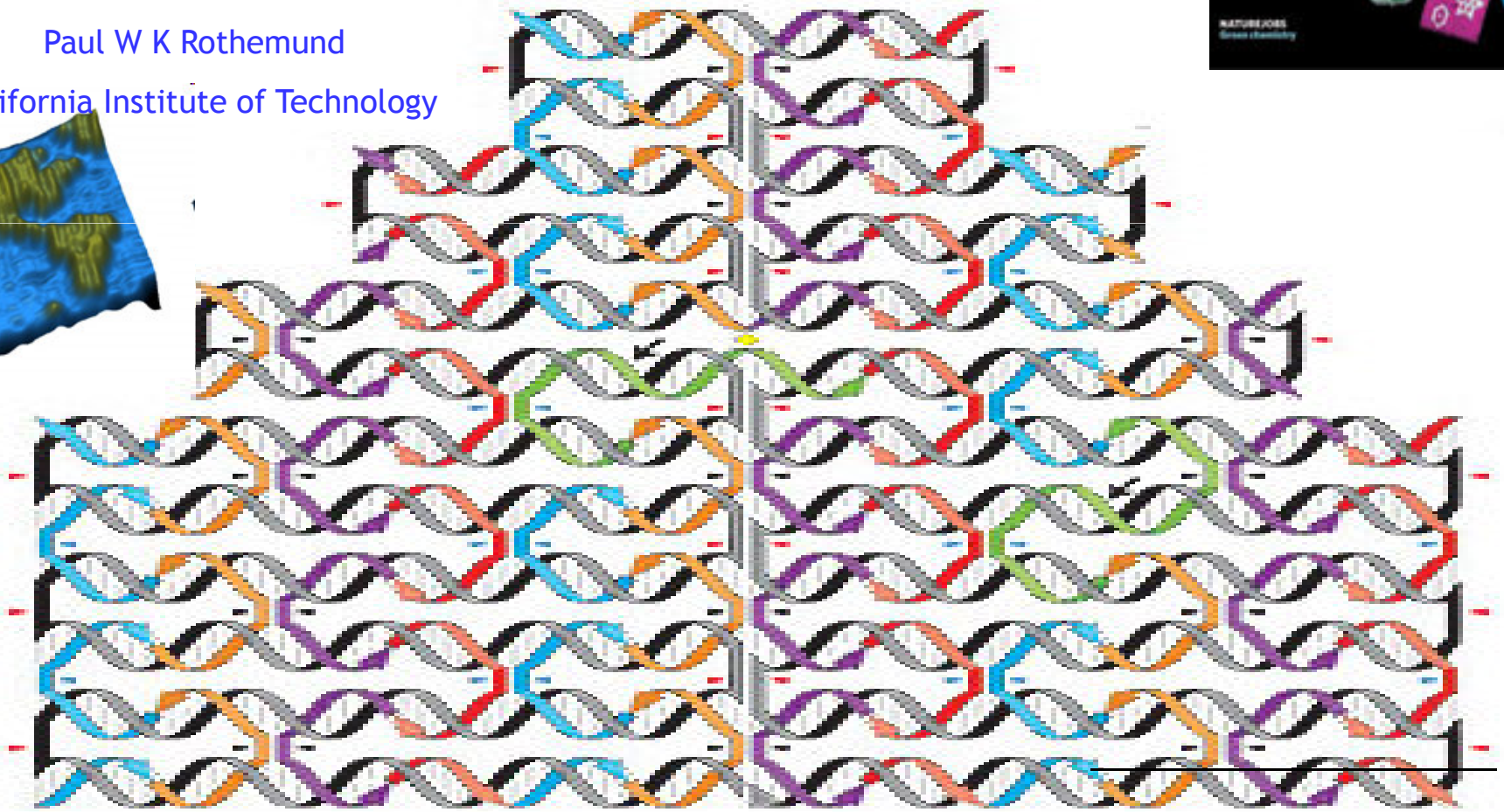




Paul W K Rothemund

California Institute of Technology





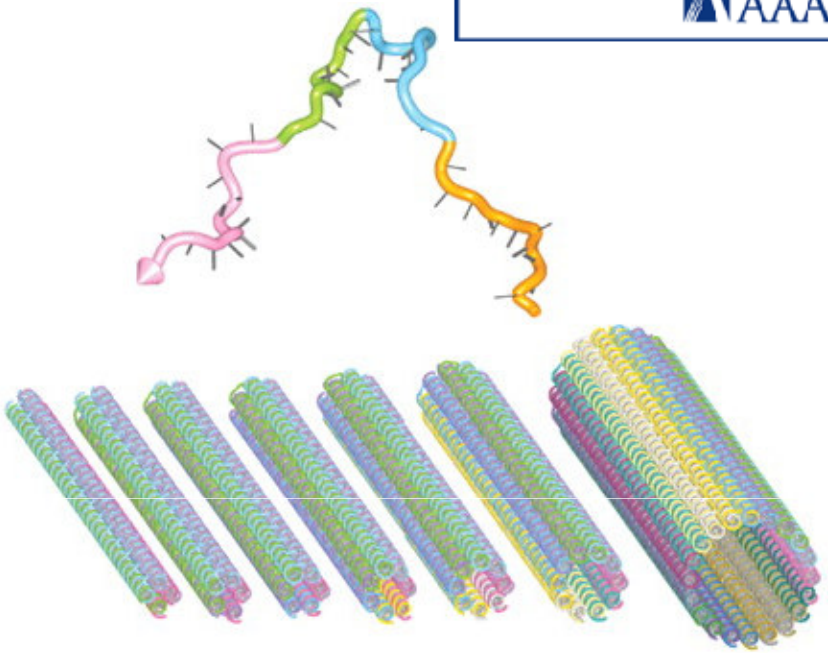
PWK Rothemund, *Nature* 440, 297 (2006)

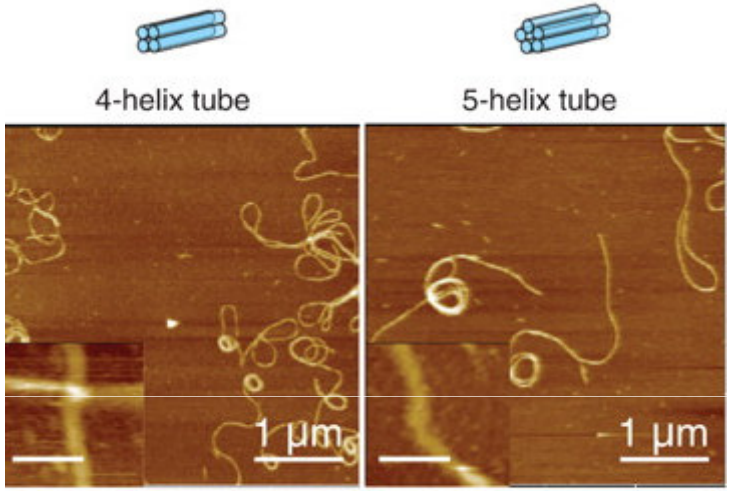
Science
AAAS

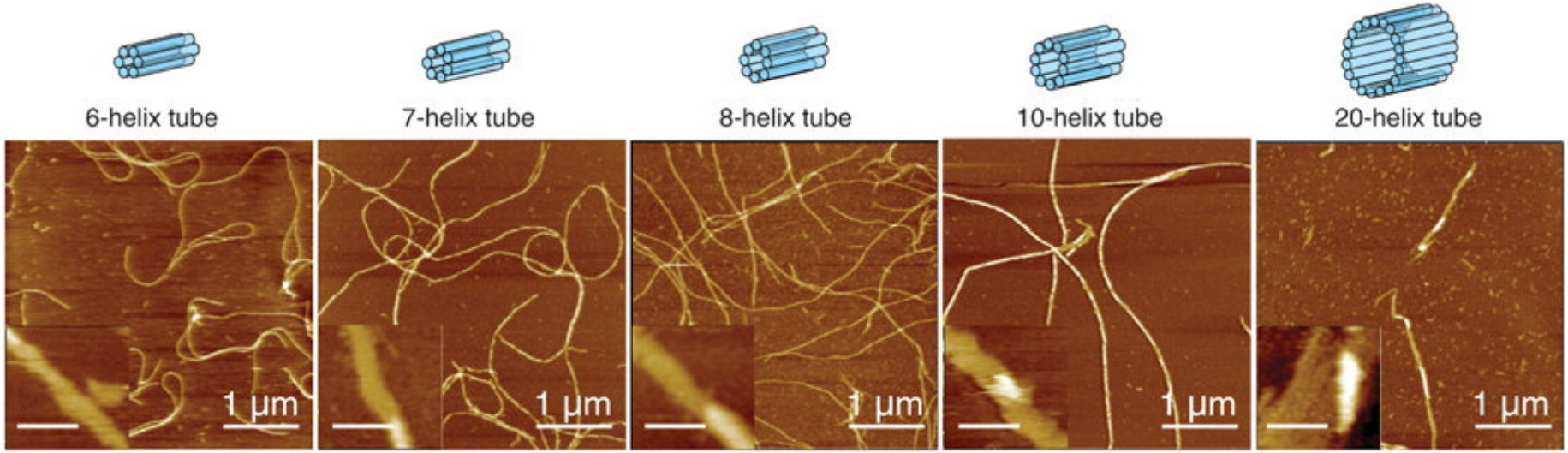
Programming DNA Tube Circumferences

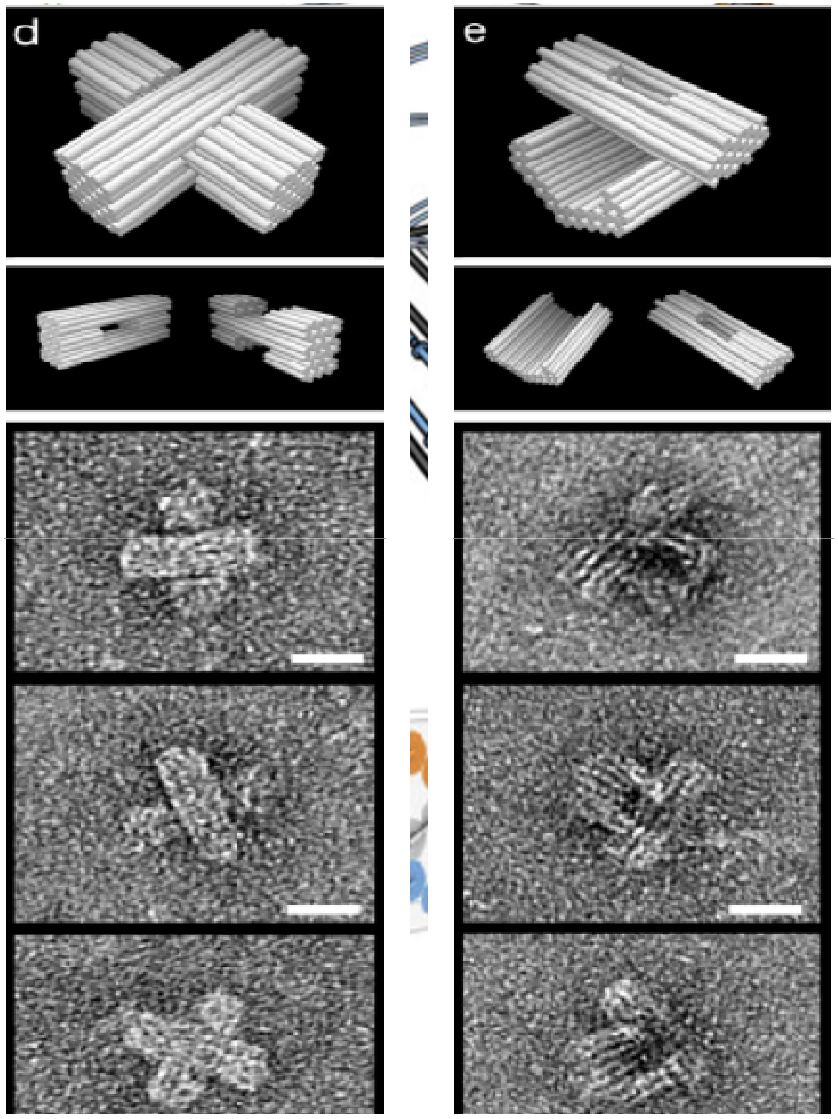
Peng Yin, *et al.*

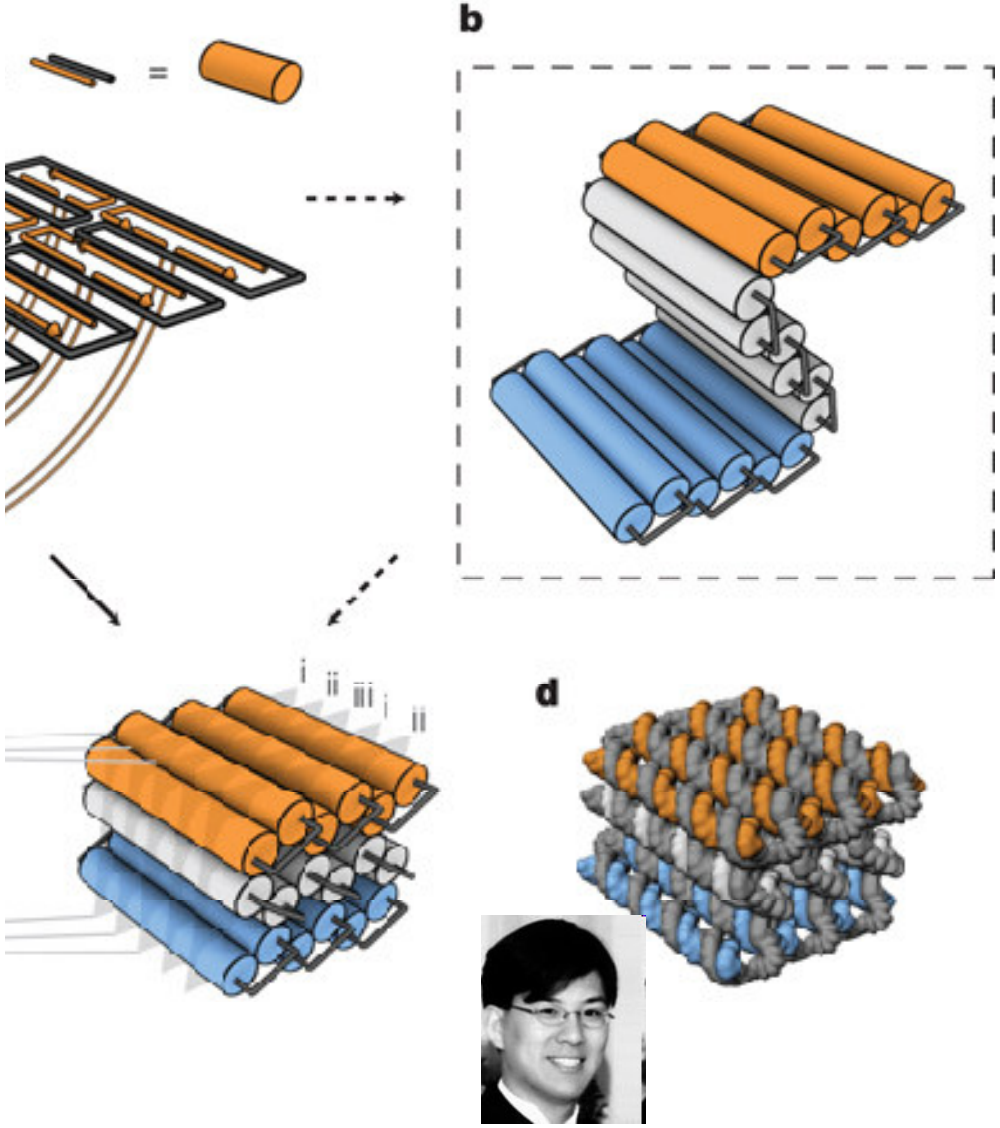
Science **321**, 824 (2008);
DOI: 10.1126/science.1157312









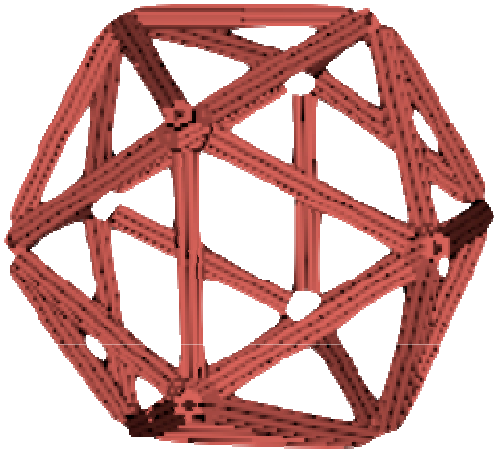


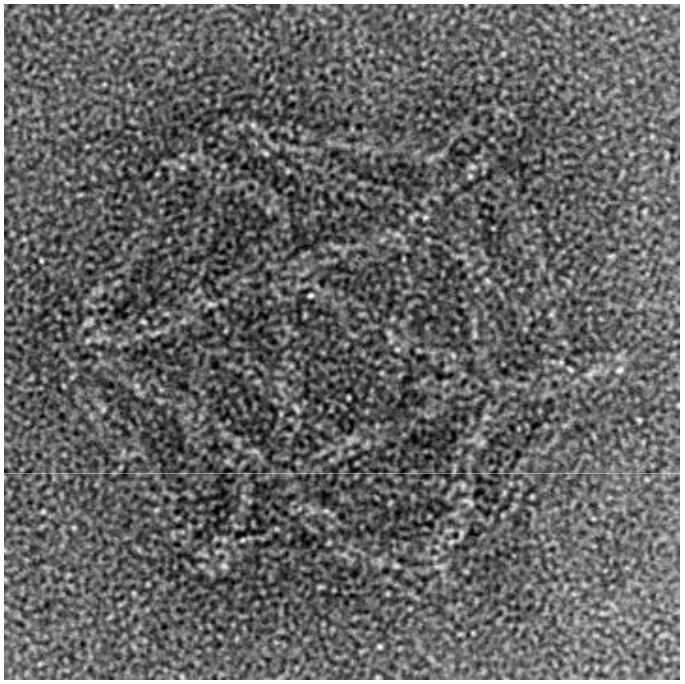
William Shi

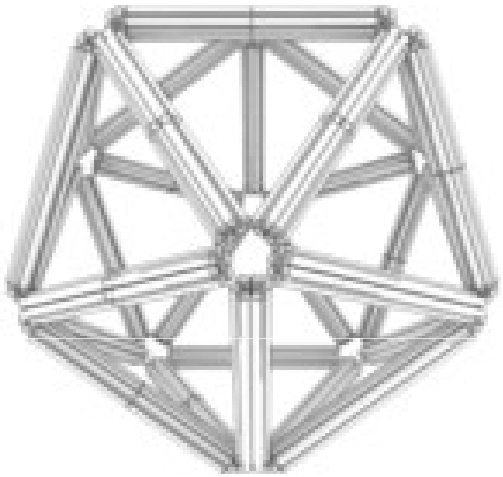
Harvard

S.M. Douglas, H. Dietz, T. Liedl, B. Högberg, F. Graf and W. M. Shih
Self-assembly of DNA into nanoscale three-dimensional shapes, Nature (2009)

3D Wireframe Icosahedron







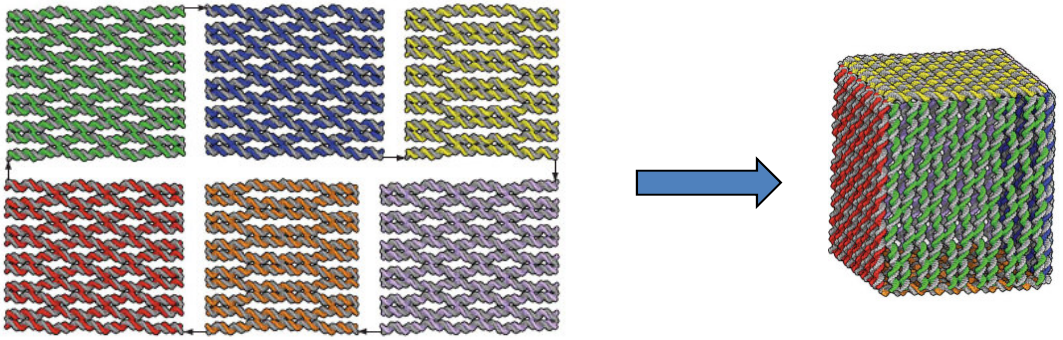
William Shi

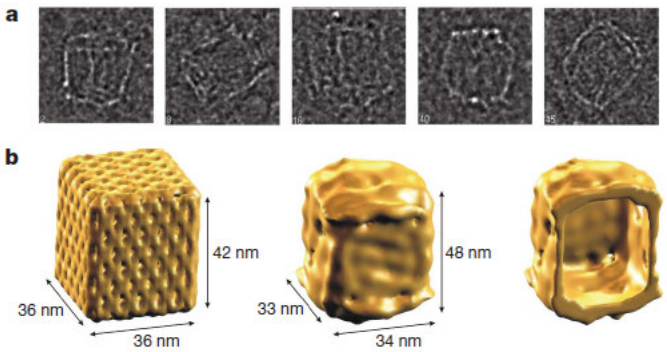
Harvard

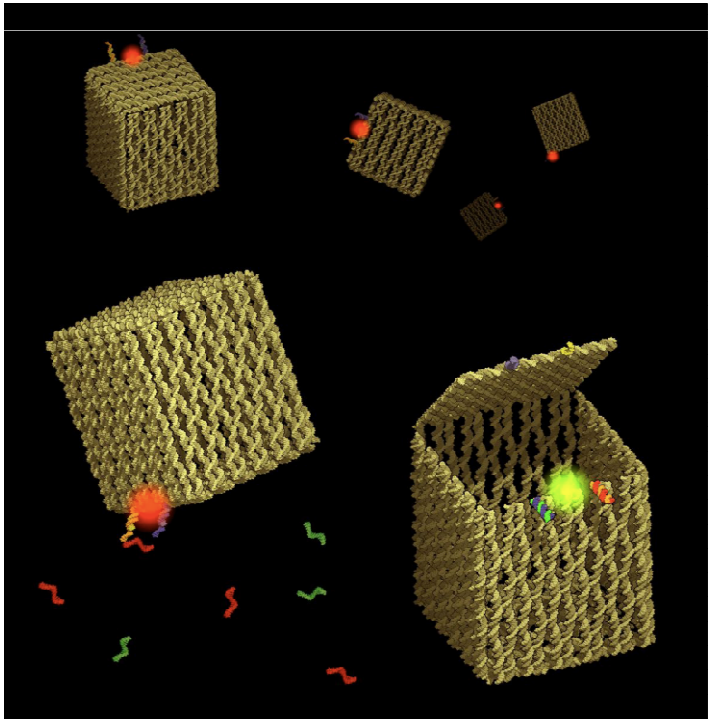
S.M. Douglas, H. Dietz, T. Liedl, B. Högberg, F. Graf and W. M. Shih
Self-assembly of DNA into nanoscale three-dimensional shapes, Nature (2009)

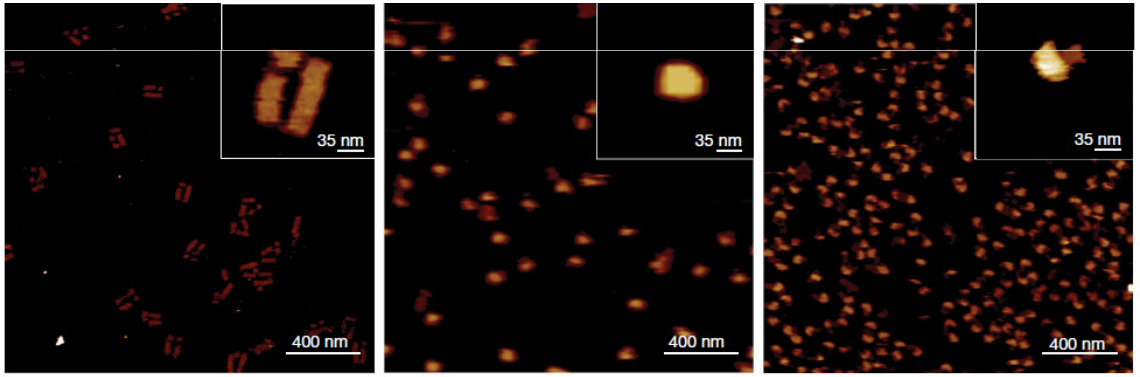
Self-assembly of a DNA origami box

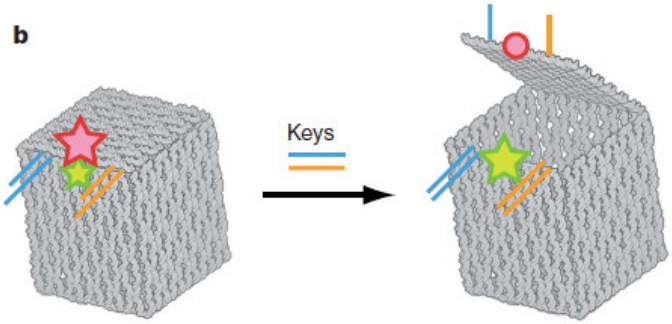
Andersen et al *Nature* **2009**, *459*, 73







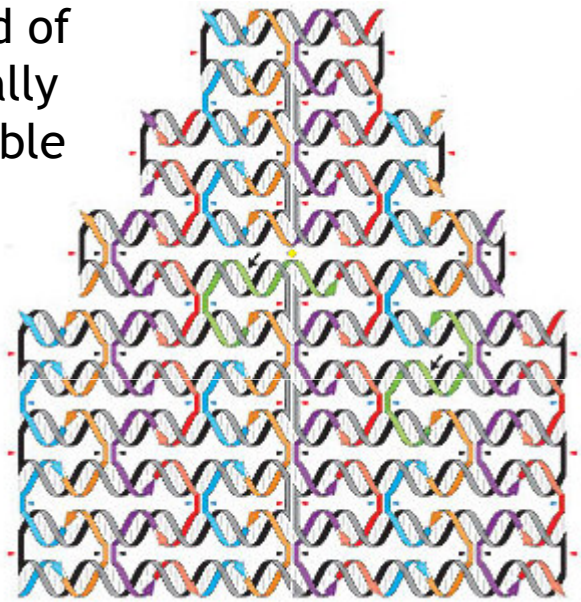




Aurhus Univ, Denmark

DNA circuit boards (IBM)

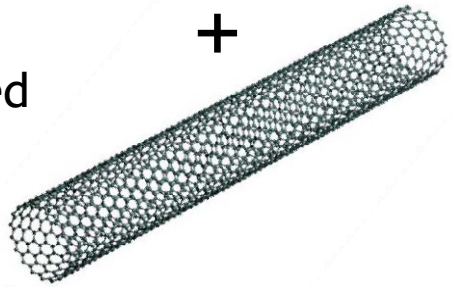
6 nm grid of individually addressable pixels



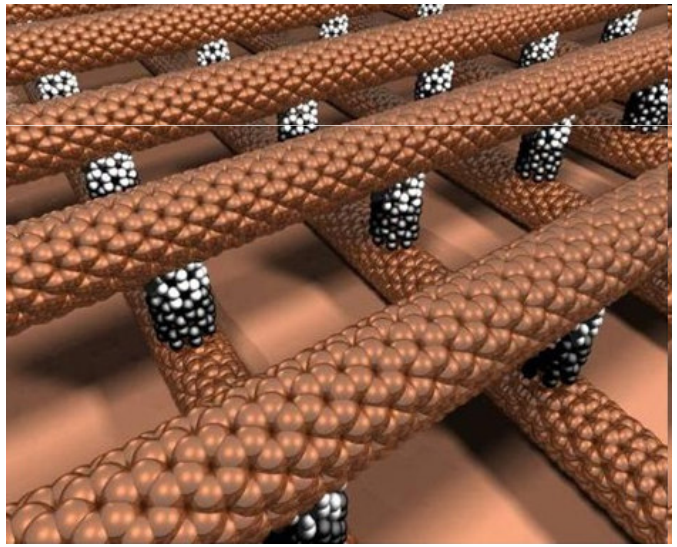
PWK Rothemund, *Nature* 440, 297 (2006)

+

DNA-wrapped nanotubes



"What we are really making are tiny DNA circuit boards that will be used to assemble other components."
--Greg Wallraff, IBM

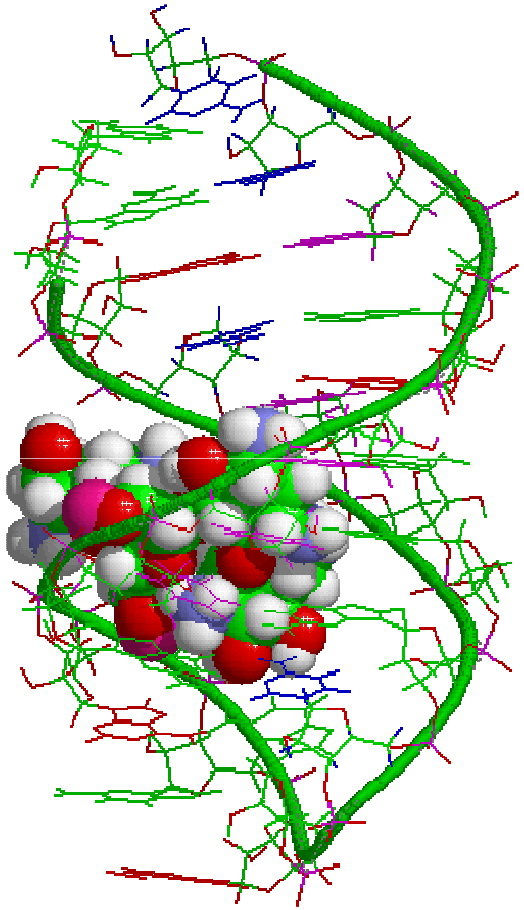


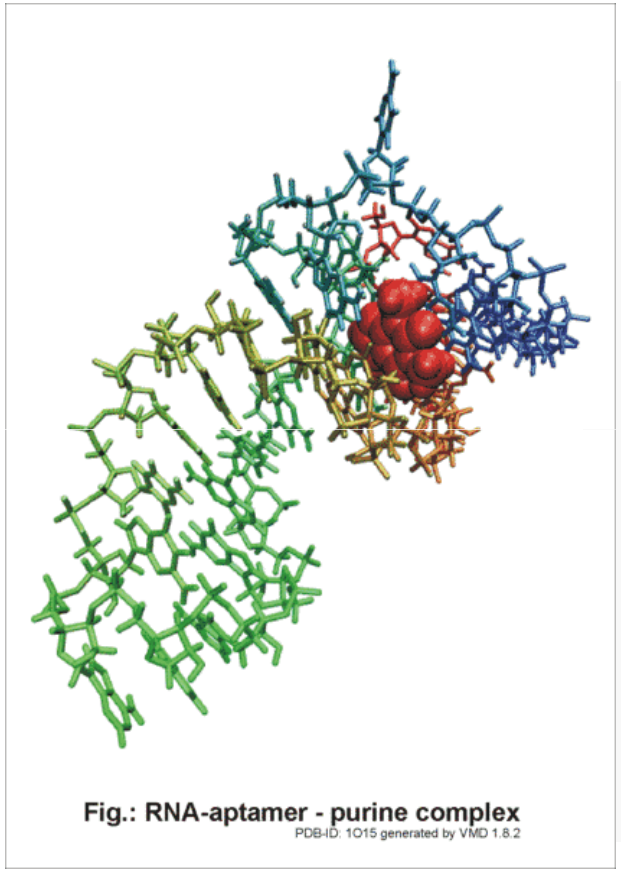
European Nanoelectronics Initiative Advisory Council

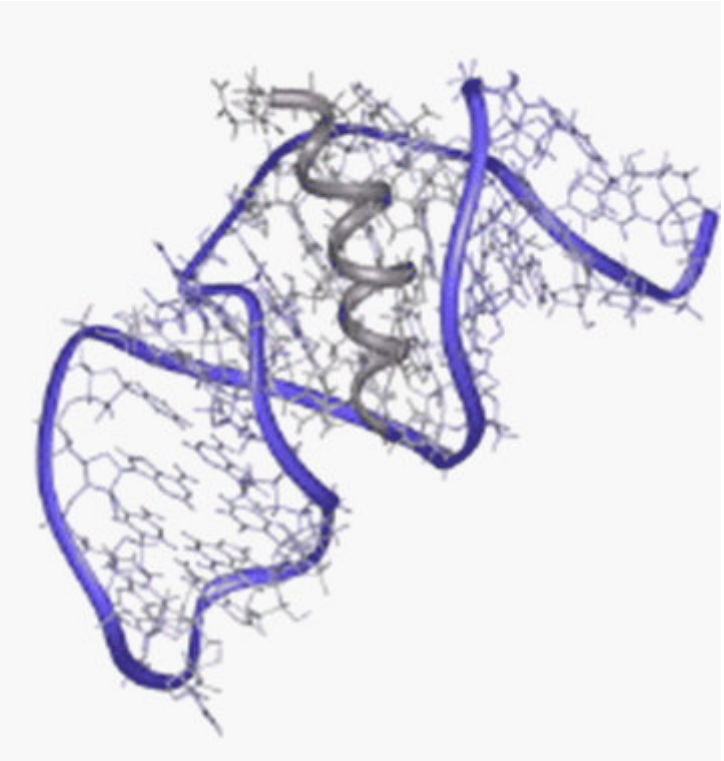
-self-assembly
-6 nm feature spacing
-versatile template / etch mask

DNA as a Computational Material

Aptamers (Sensors)







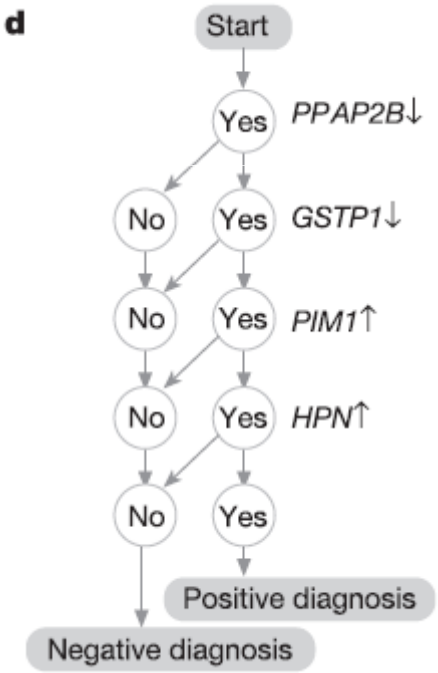
Computation: Curing Cancer with one AND Gate

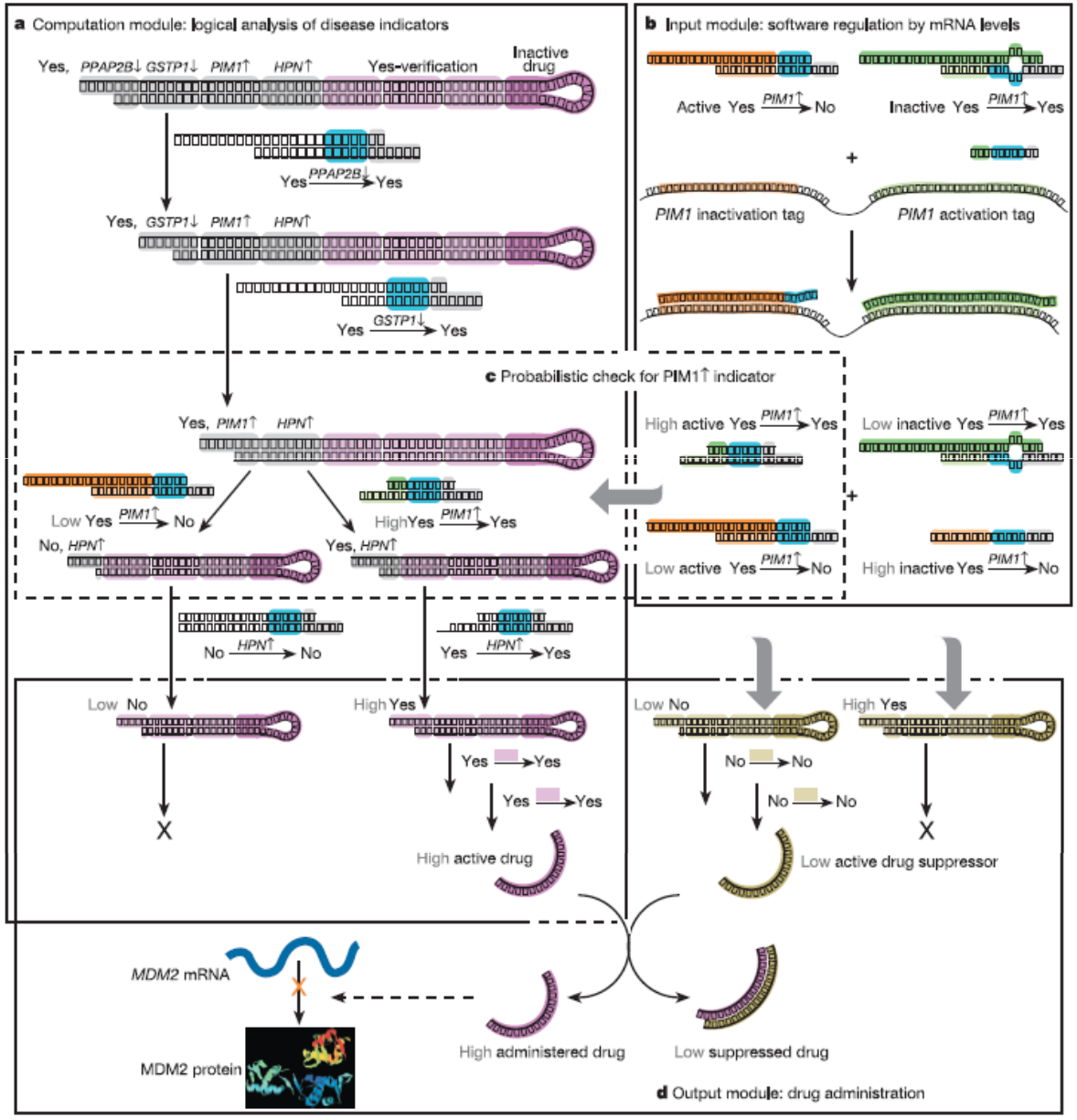
letters to nature

An autonomous molecular computer for logical control of gene expression

Yaakov Benenson[1,2], Binyamin Gil[2], Uri Ben-Dor[1], Rivka Adar[2] & Ehud Shapiro[1,2]

[1]Department of Computer Science and Applied Mathematics and [2]Department of Biological Chemistry, Weizmann Institute of Science, Rehovot 76100, Israel

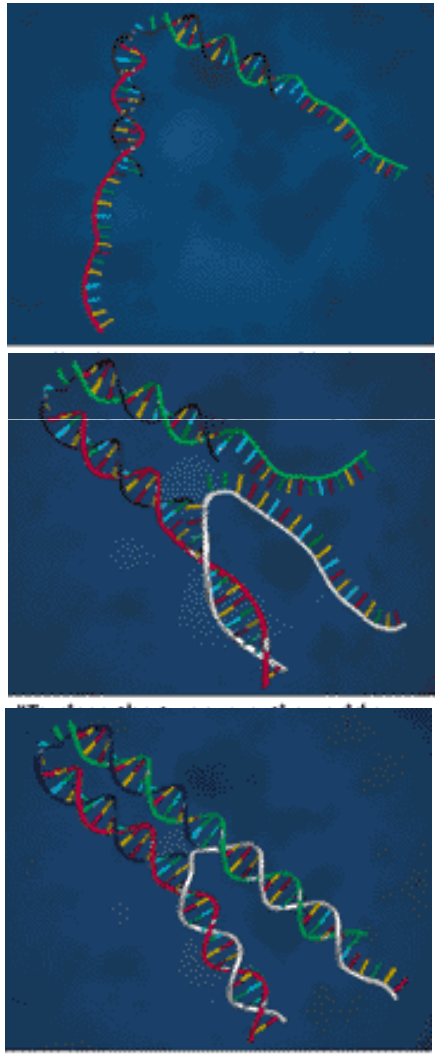




Actuators

DNA Tweezers

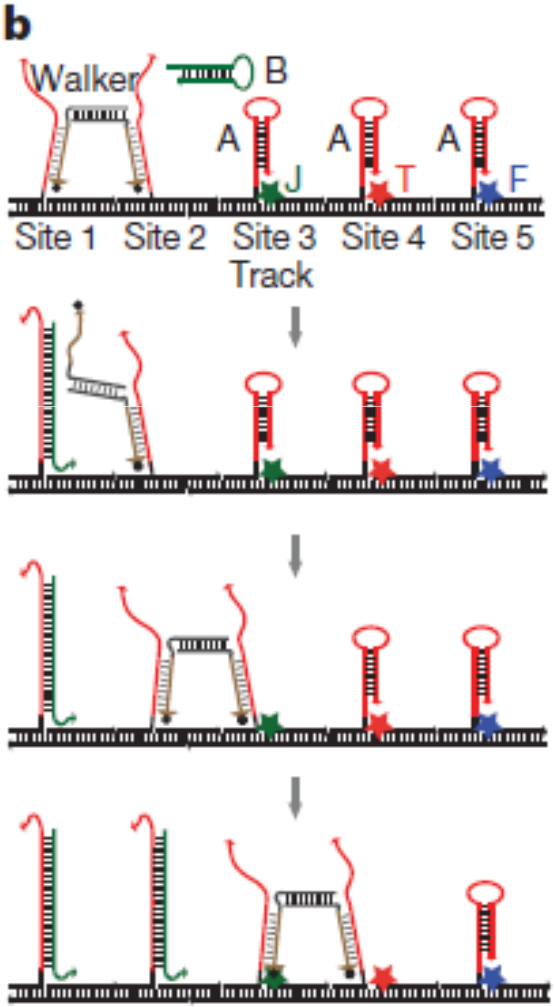
(Yurke & Turberfield, Nature 2000)



"The fuel strand attaches to the handles and draws the two arms of the tweezers together."

DNA Walkers

(Yin, Choi, Calvert & Pierce, Nature 2008)



Summary

- DNA technology is making great progress
  - Developing sensor, actuators, and building materials.
  - All thanks to the programmable nature of DNA.

- DNA computation has also been investigate deeply.
  - DNA tiling systems are Turing complete. They can be used to build 'carpets' with predetermined size and organization.
  - Automata and Turing machines have been demonstrated or designed.

- But there is still space for creativity
  - What is the 'best' way to write algorithms with DNA?

- What is DNA nanotech for? Ultimately:
  - To construct 'arbitrary' nanomaterials.
  - To compute 'in vivo'.

Computation by DNA Strand Displacement

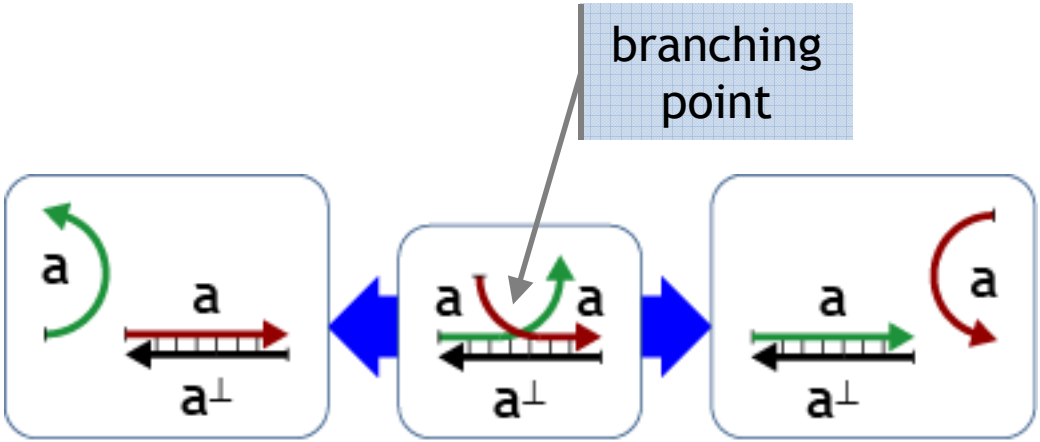
DNA Computing

- 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].

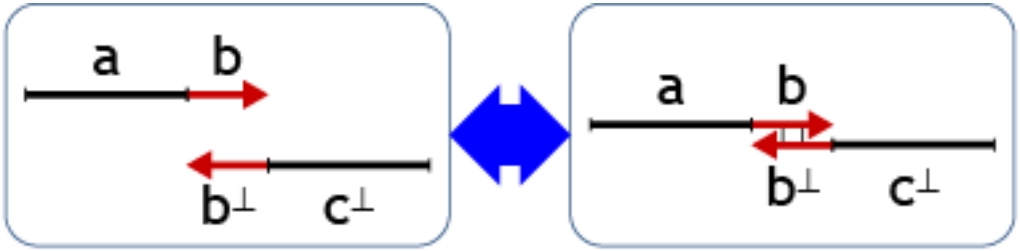
Branch Migration



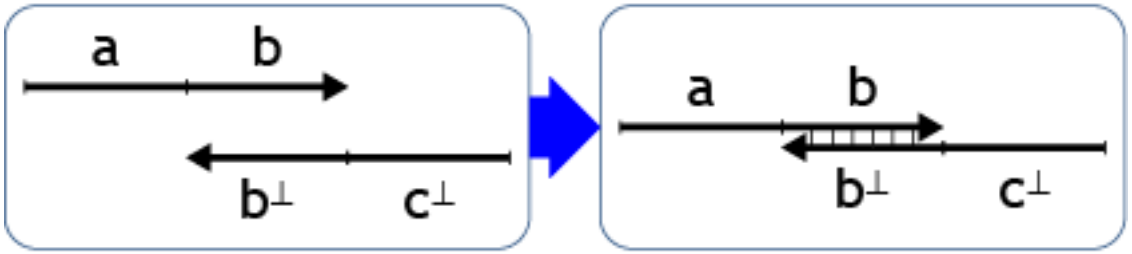
Branch Migration

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

Short and Long Segments

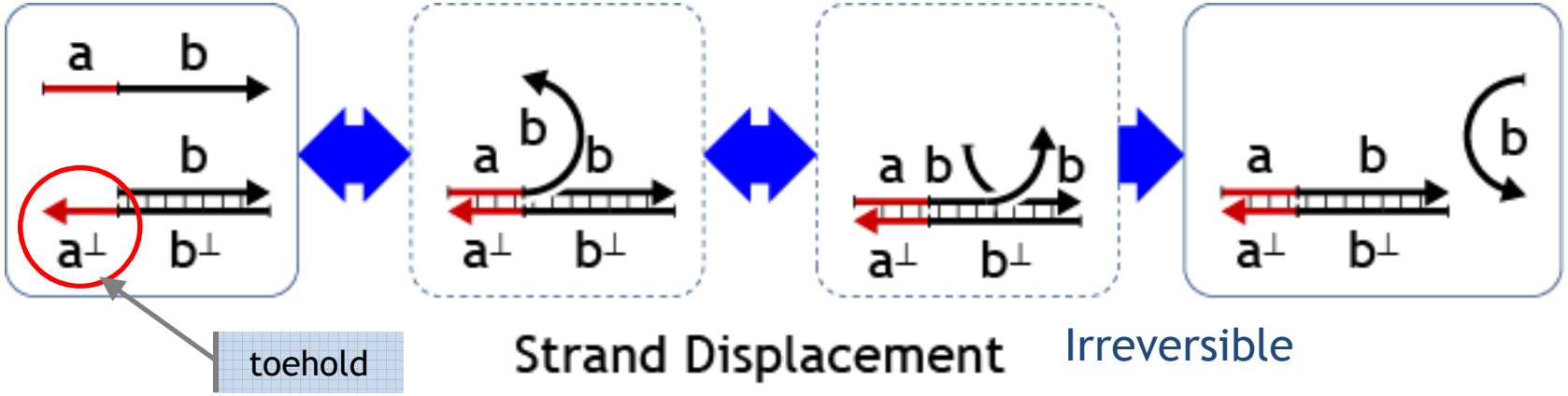


Reversible Binding



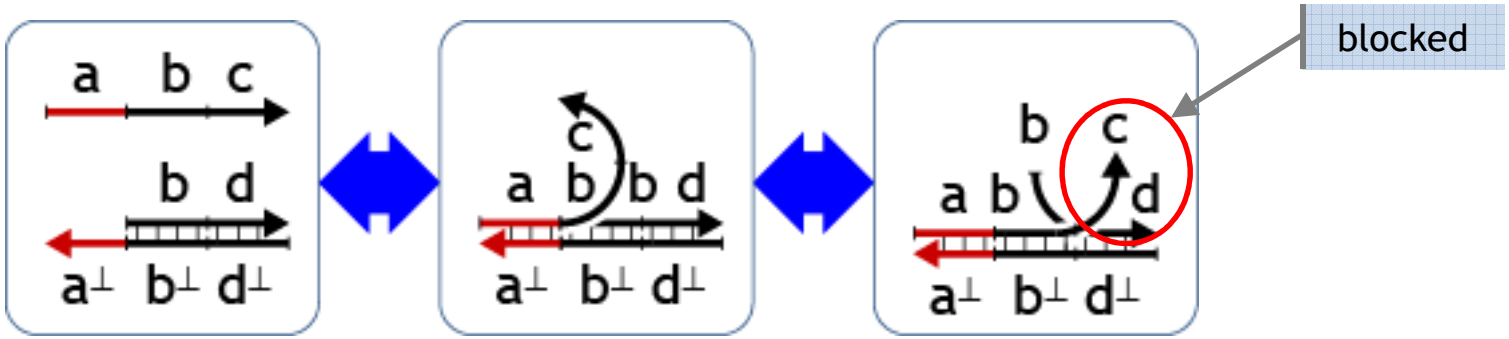
Irreversible Binding

Strand Displacement Reaction



Strand Displacement Irreversible

Partial Match

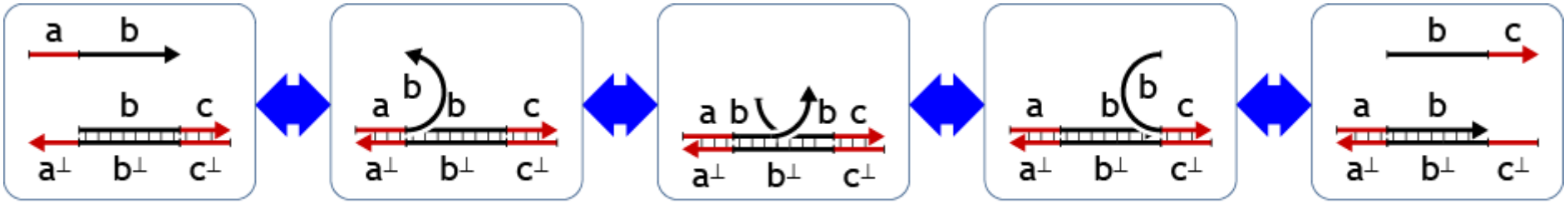


Reversible!

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



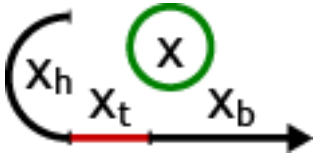
Toehold Exchange

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

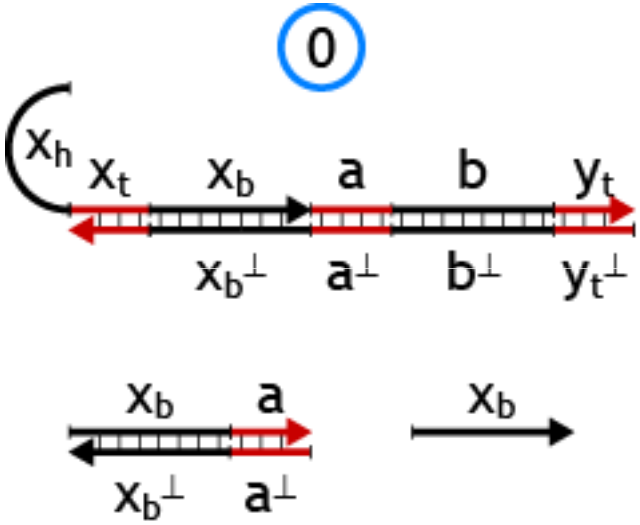
- Signals “x” are always positive strands

- 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

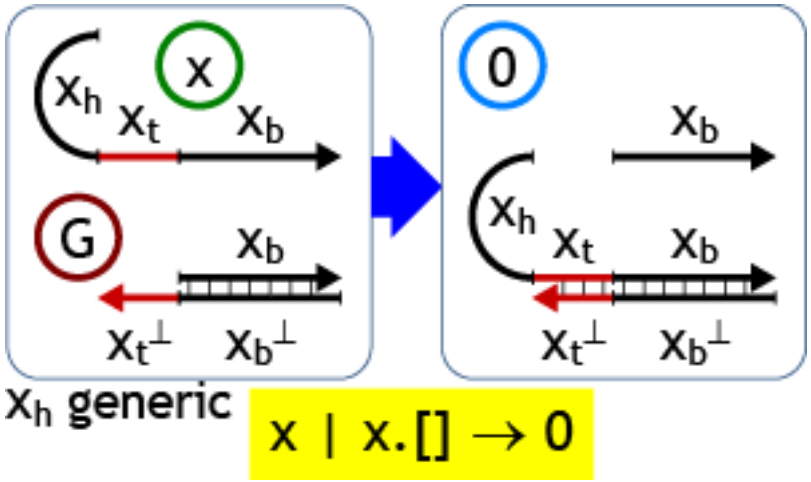
- This separation helps the DNA realization, as one can use 3-letter alphabets (ATC/ATG) for each strand, minimizing secondary structure and entanglement.

Inert Systems

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



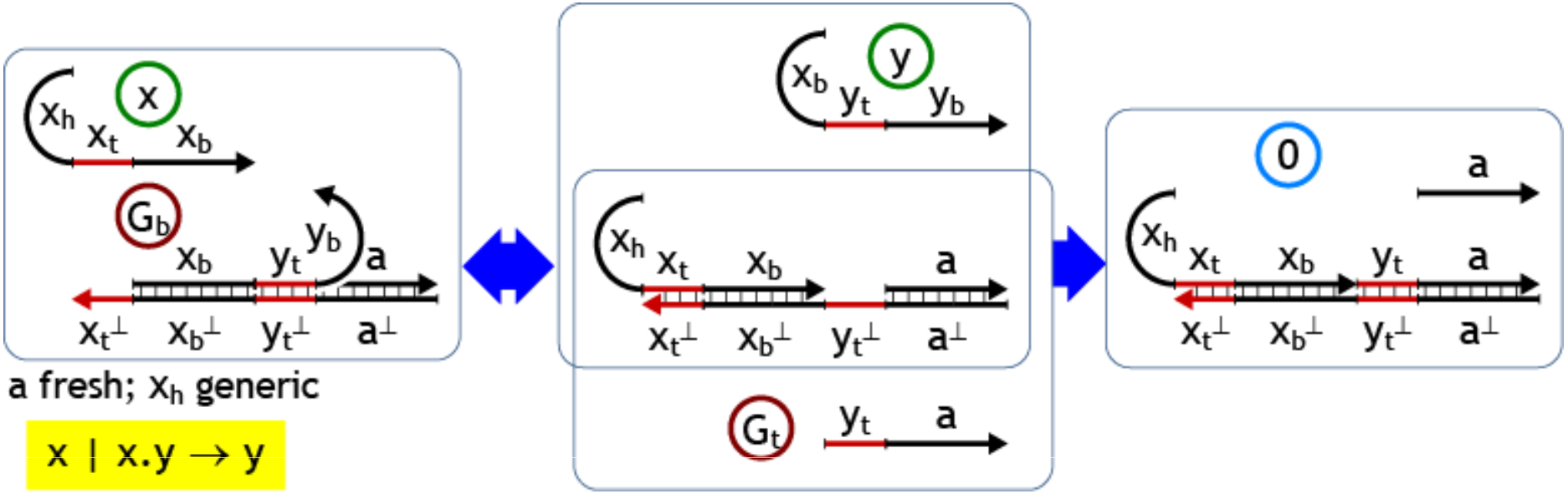
x.[] Annihilator Gate



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

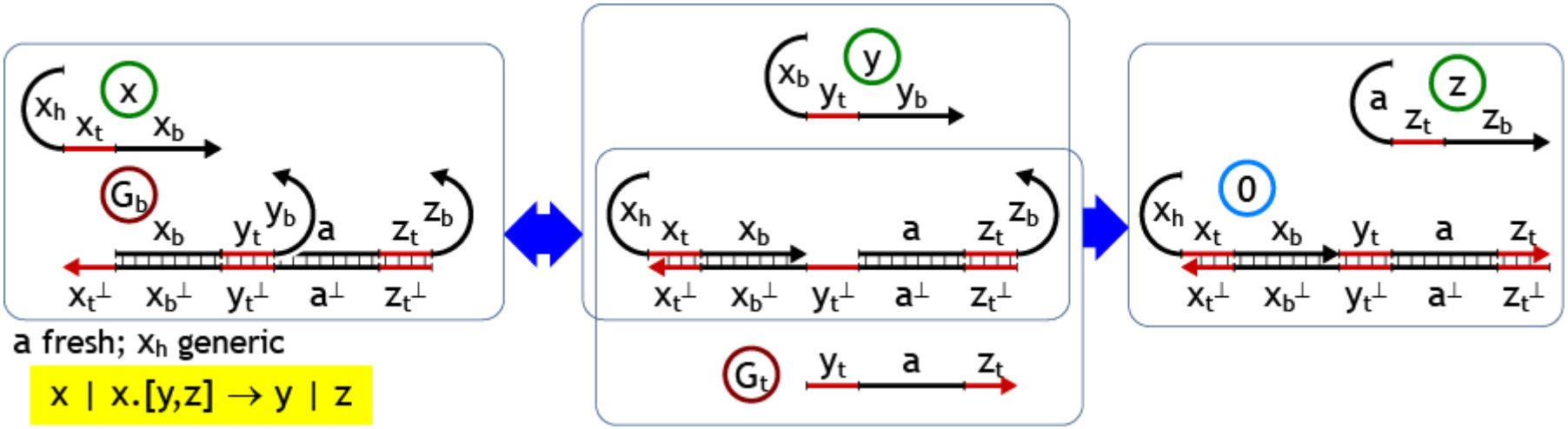


G_b,G_t (gate backbone and trigger) form the transducer.

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



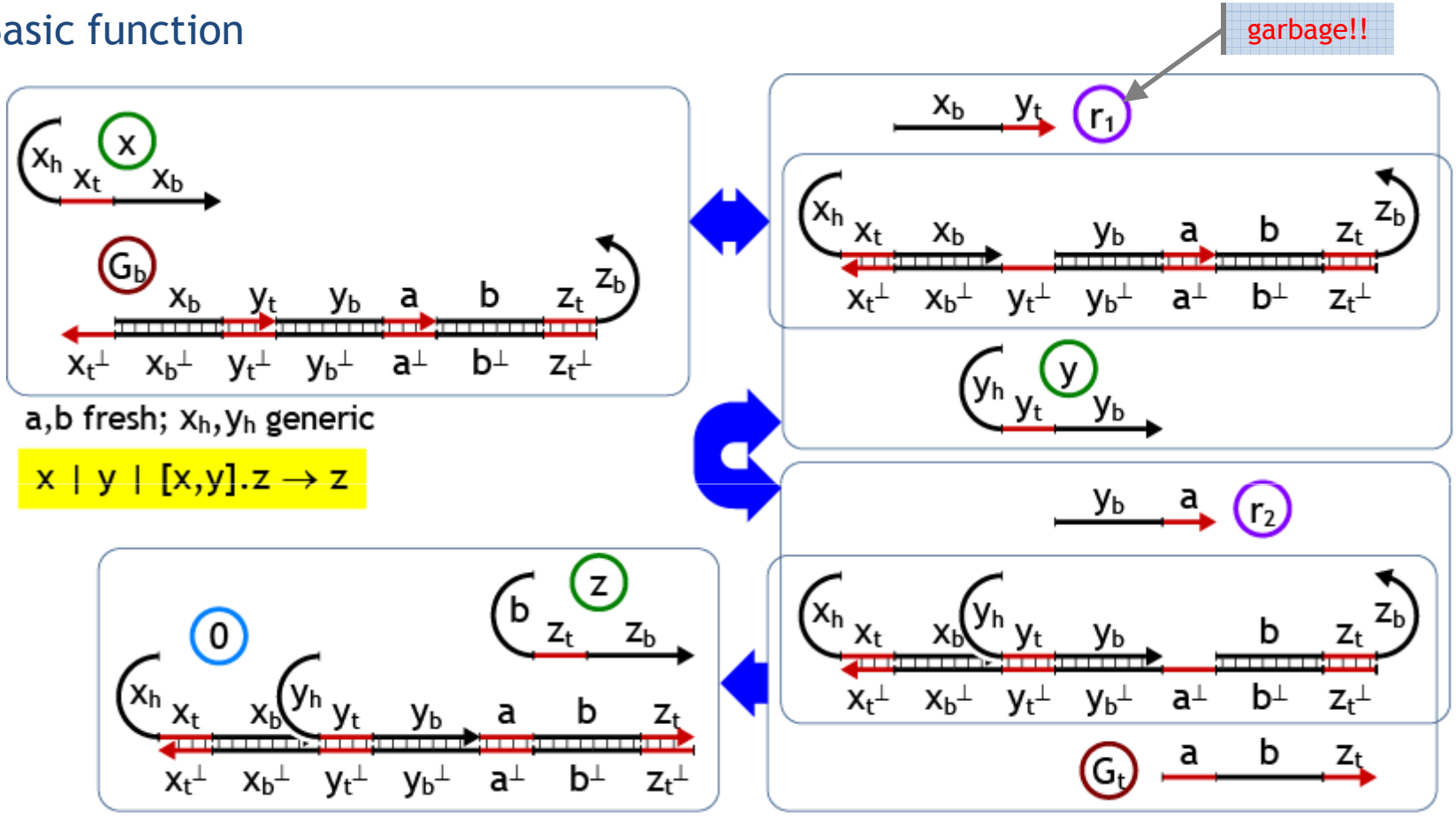
G_b,G_t (gate backbone and trigger) form the transducer.

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,y].z Join Gate (function)

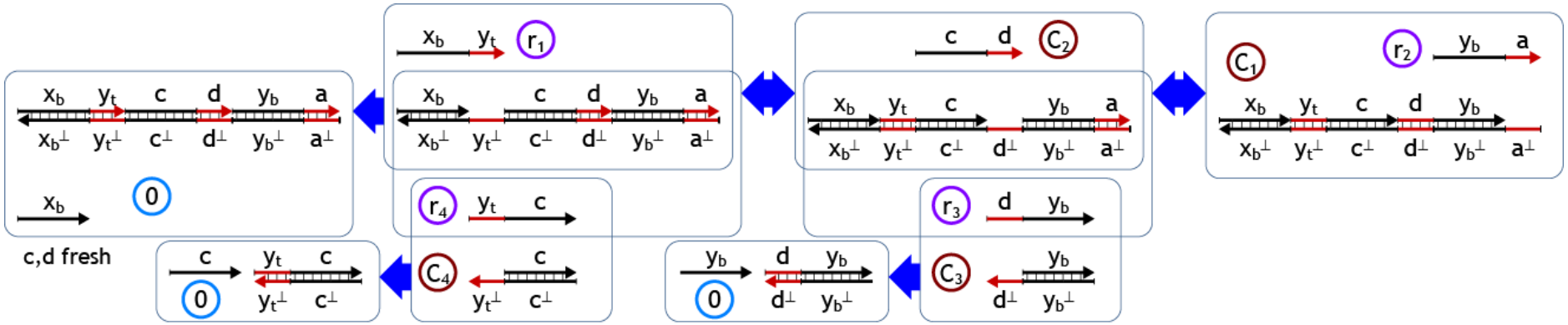
Basic function



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

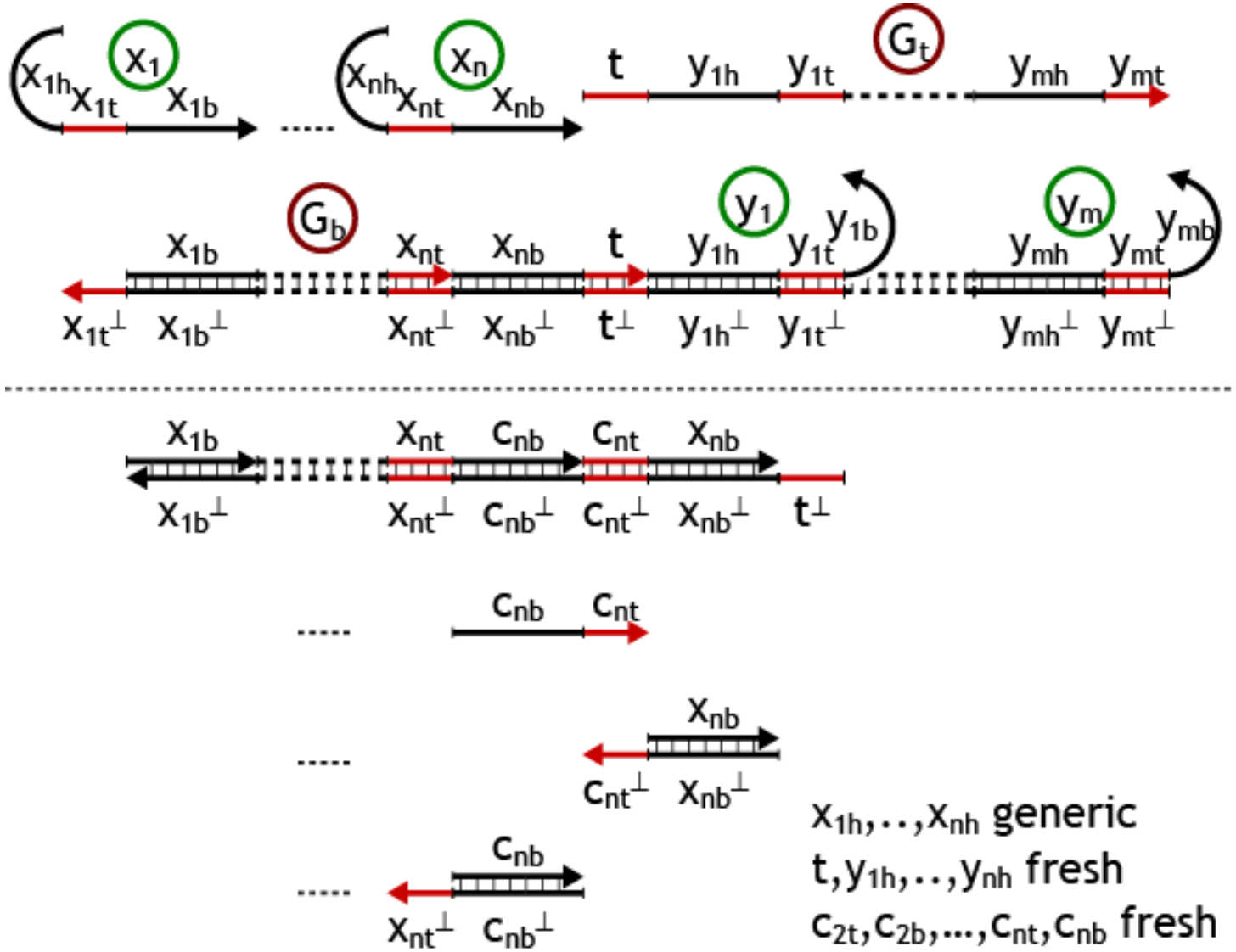


Garbage collection of r_1 is needed for join to work well. This is done by another reversible-AND between r_1 and r_2, triggered by the release of r_2. This second reversible-AND leaves garbage too (r_3, r_4), but this can be collected immediately, as we know by construction that both inputs r_1,r_2 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

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



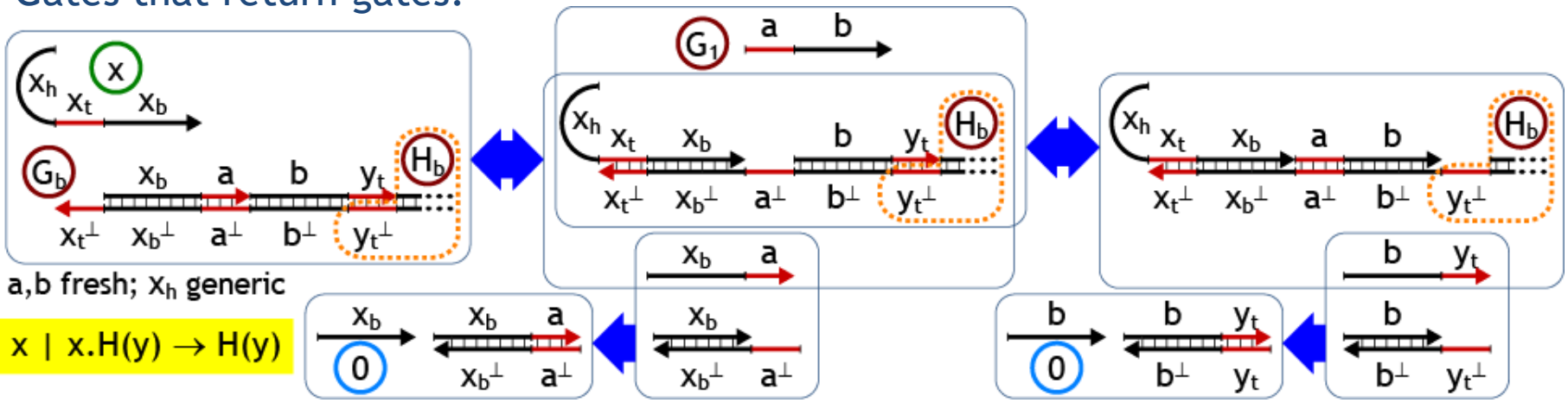
$x_{1h},..,x_{nh}$ generic

$t, y_{1h},..,y_{nh}$ fresh

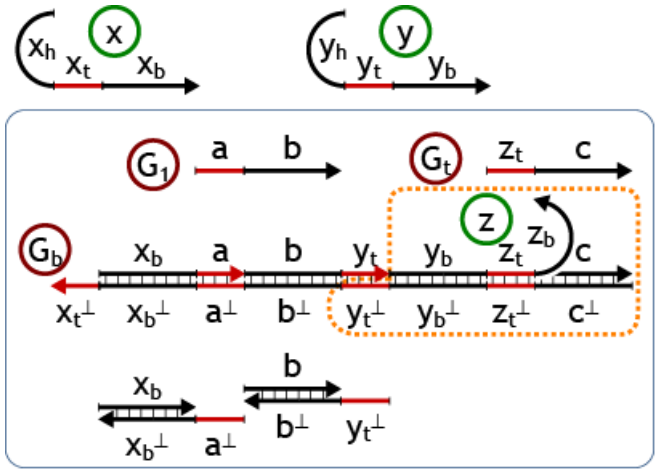
$c_{2t}, c_{2b},...,c_{nt}, c_{nb}$ fresh

x.H(y) Curried Gates

Gates that return gates:



For example, x.y.z:



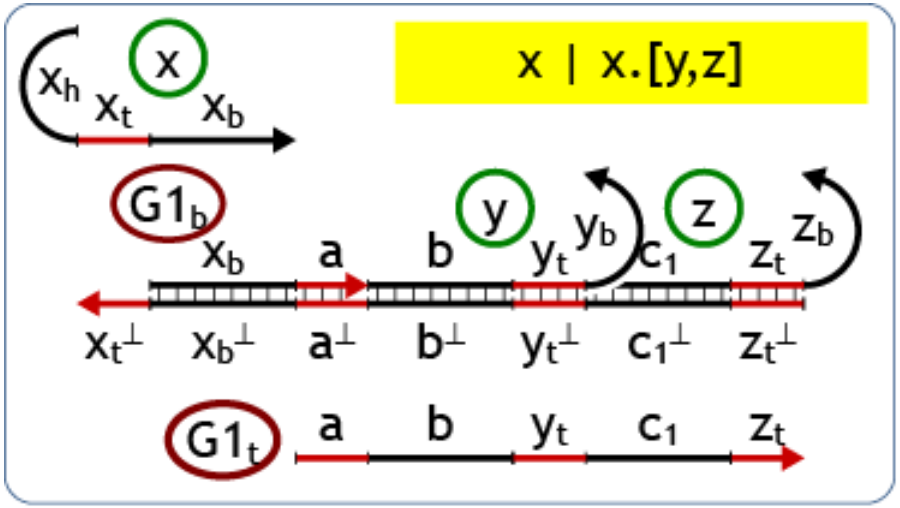
This means we can have gates of the form:

$$G \quad ::= \quad [x_1,..,x_n].[x'_1,..,x'_m] \quad \vdots$$
$$\qquad\quad [x_1,..,x_n].G$$

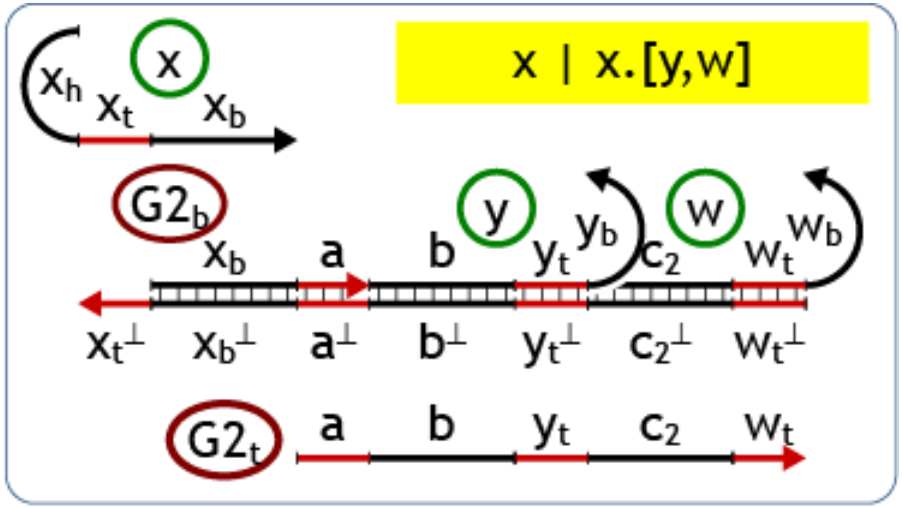
$n \geq 1$, $m \geq 0$

Design, Compilation and Verification Challenges

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:

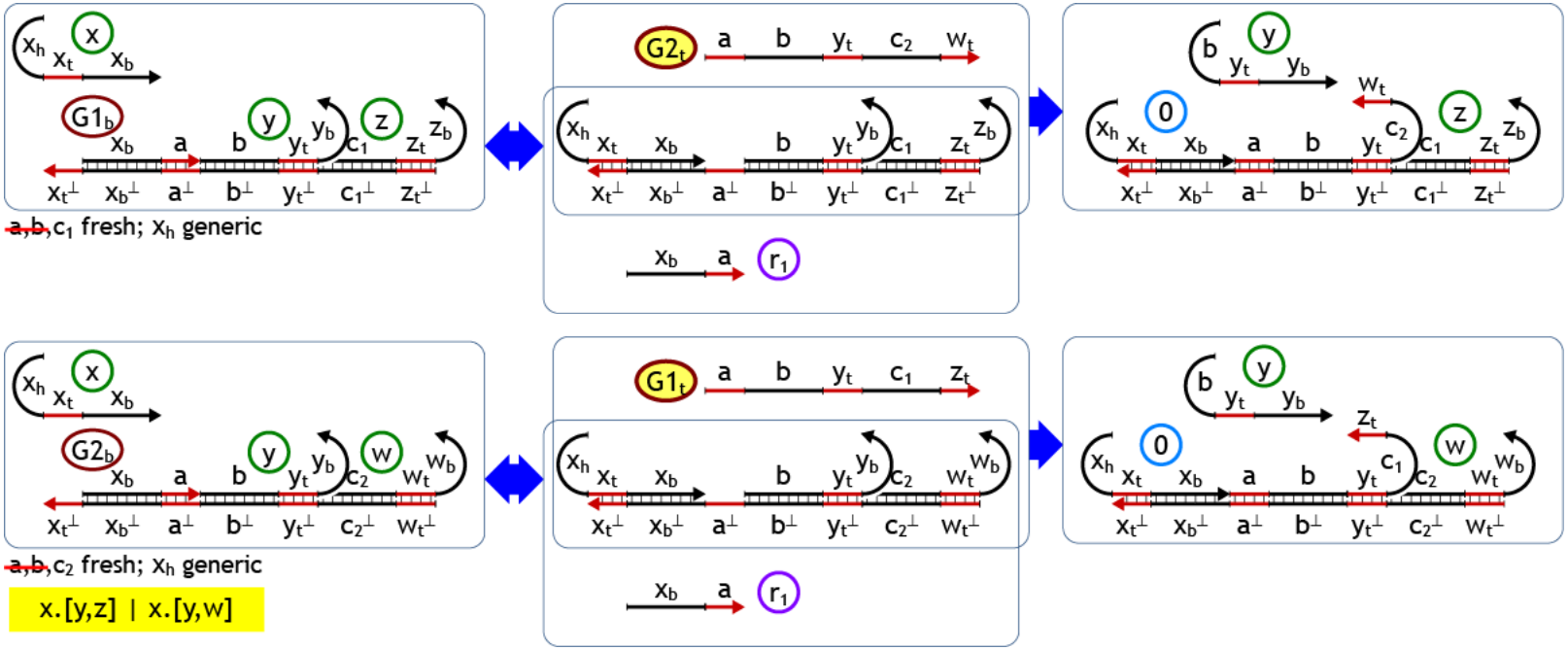
  x | x.[y,z] | x | x.[y,w]

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

- What goes wrong?

Exercise 3 Solution

Deadlocks! Consider x | x | x.[y,z] | x.[y,w], and suppose we had taken c fresh (hence different c_1,c_2), but did *not* used gate-unique segments for a,b:



The $G2_t$ trigger can bind to the wrong $G1_b$ backbone and get stuck there, and vice versa, without ever releasing z or w.

This is just a made-up problem, but one must watch out for all kinds of possible interferences.

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

[David Soloveichik]

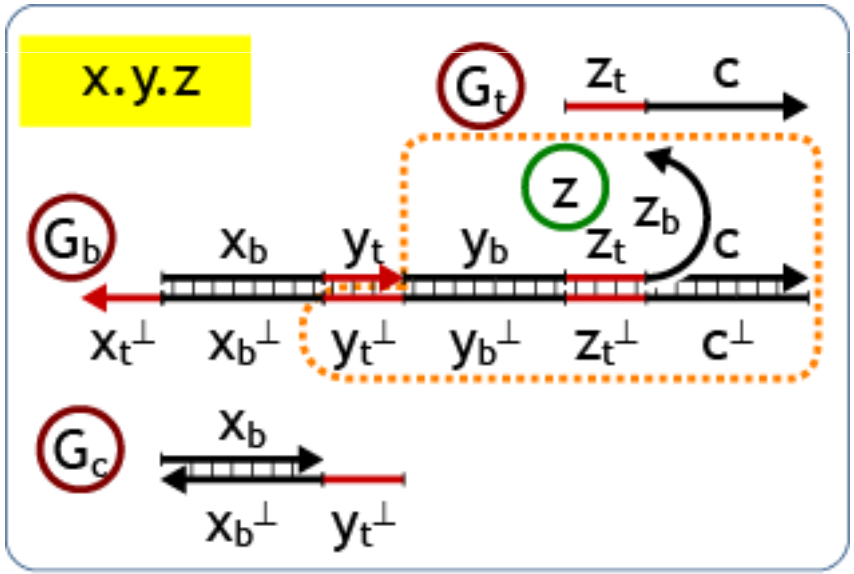
Consider curried gates without the a,b segments (example below): instead of releasing x_b,a and b,y_t segments, they would release x_b,y_t.

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)

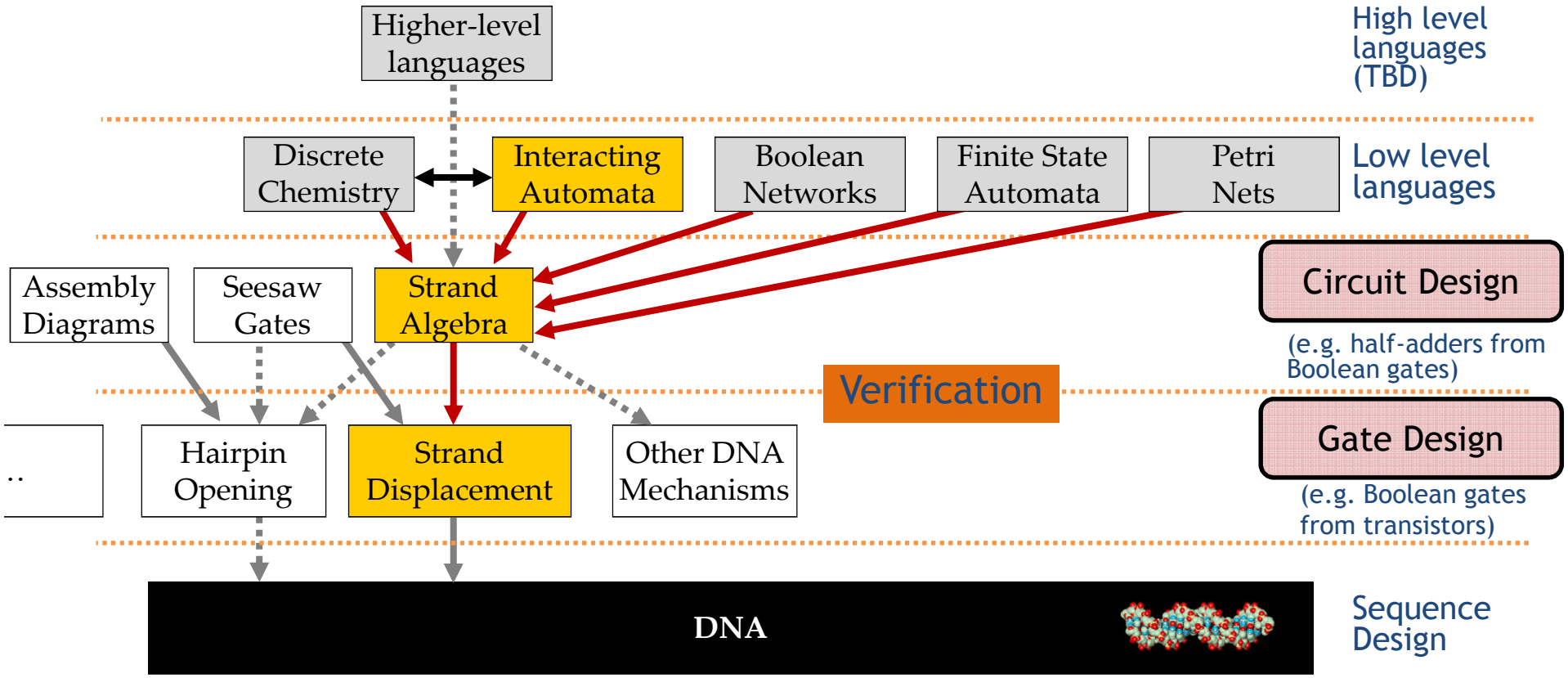
Addressing the Challenges

- We need to formally specify
  - The intended behavior of DNA gates.
  - Their implementation.

- We need to verify
  - That the implementation satisfies the specification.
  - In all possible 'soups' (contexts).
  - Possibly by modelchecking (the state space is highly combinatorial).

DNA Compilation

Separating Circuit Design from Gate Design



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?

- How do we verify its correctness?

Strand Algebra

Strand Algebra

$$P \ ::= \ x \ \vdots \ [x_1,..,x_n].[y_1,..,y_m] \ \vdots \ 0 \ \vdots \ P|P \ \vdots \ P^* \qquad n\geq 1,\ m\geq 0$$

x	is a *signal*
$[x_1,..,x_n].[y_1,..,y_m]$	is a *gate*
0	is an *inert solution*
$P\|P$	is *parallel composition* of signals and gates
P^*	is a *population* (multiset) of signals and gates

Reaction Rule

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

Auxiliary rules (axioms of diluted well-mixed solutions)

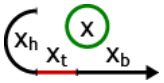
$$P \rightarrow P' \ \Rightarrow \ P \mid P'' \rightarrow P' \mid P'' \qquad \text{Dilution}$$

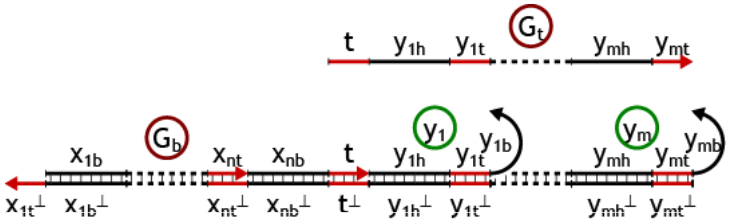
$$P \equiv P_1,\ P_1 \rightarrow P_2,\ P_2 \equiv P' \ \Rightarrow \ P \rightarrow P' \qquad \text{Well Mixing}$$

Where $\equiv$ is a congruence relation (syntactical 'chemical mixing')
with $P^* \equiv P \mid P^*$ for unbounded populations.

Compiling Strand Algebra to DNA

$P ::= x \vdots [x_1,..,x_n].[y_1,..,y_m] \vdots 0 \vdots P|P \vdots P^* \qquad n\geq 1,\ m\geq 0$

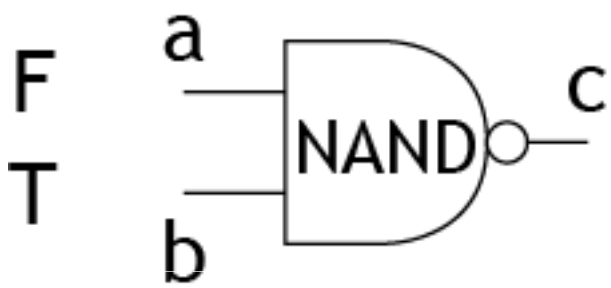
- compile(x) = 

- compile($[x_1,..,x_n].[y_1,..,y_m]$) = 

- compile(0) = empty solution
- compile(P | P') = mix(compile(P), compile(P'))
- compile(P*) = population(compile(P))

Boolean Networks

Boolean Networks to Strand Algebra



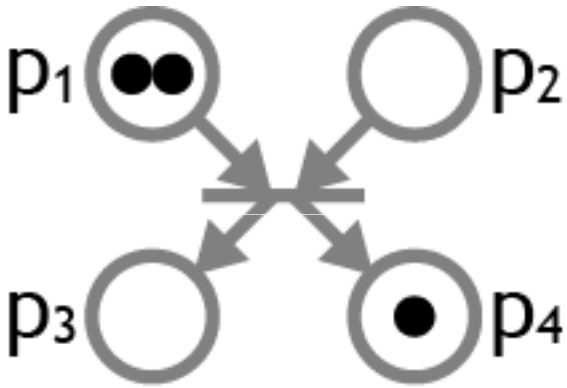
$$
\begin{array}{l}
([a_F,b_F].c_T)^* \mid \\
([a_F,b_T].c_T)^* \mid \\
([a_T,b_F].c_T)^* \mid \\
([a_T,b_T].c_F)^* \mid \\
a_F \mid b_T
\end{array}
$$

This encoding is *compositional*, and can encode *any* Boolean network:
- multi-stage networks can be assembled (combinatorial logic)
- network loops are allowed (sequential logic)

Petri Nets

Petri Nets to Strand Algebra

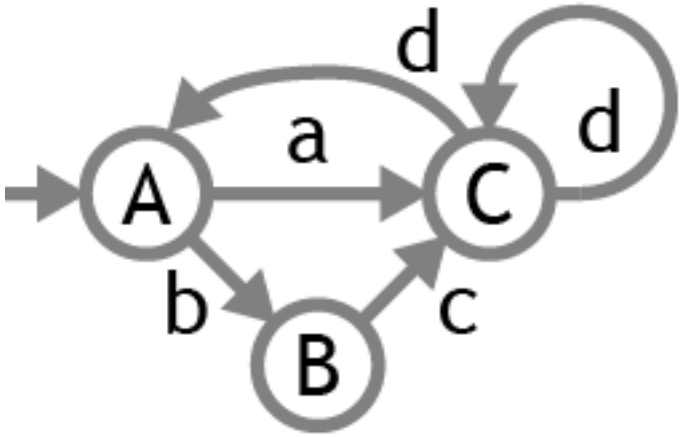
Transitions as Gates
Place markings as Signals



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

Finite State Automata

FSA to Strand Algebra



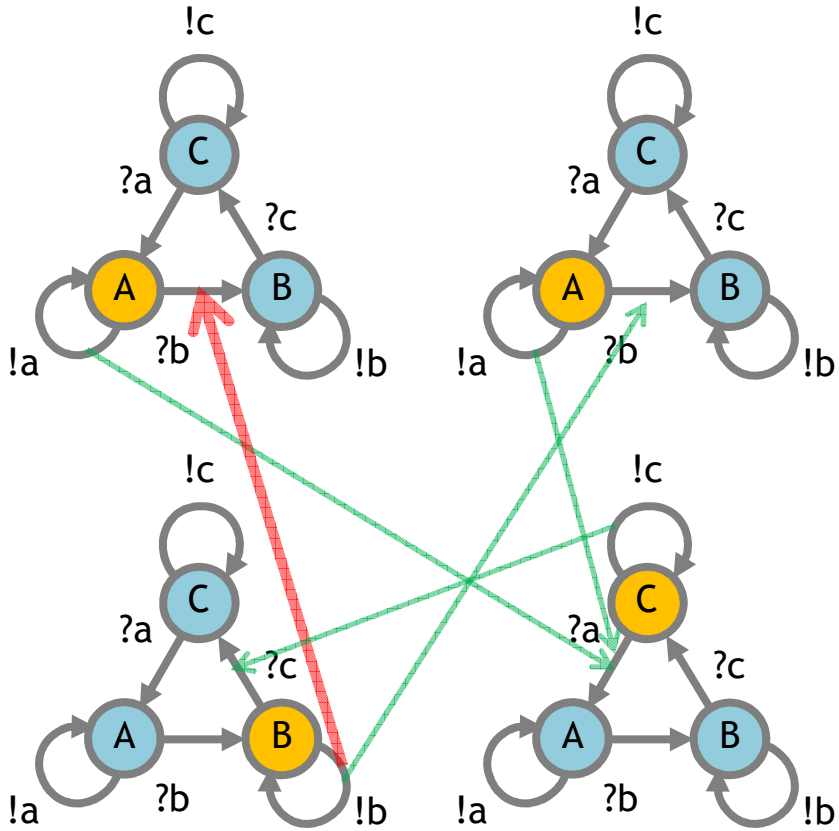
$$
\begin{aligned}
&([A,a].[C,\tau])^* \mid \\
&([A,b].[B,\tau])^* \mid \\
&([B,c].[C,\tau])^* \mid \\
&([C,d].[C,\tau])^* \mid \\
&([C,d].[A,\tau])^* \mid \\
&A \mid \tau
\end{aligned}
$$

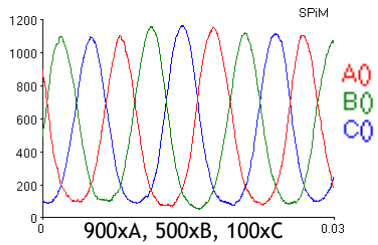
Input strings

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

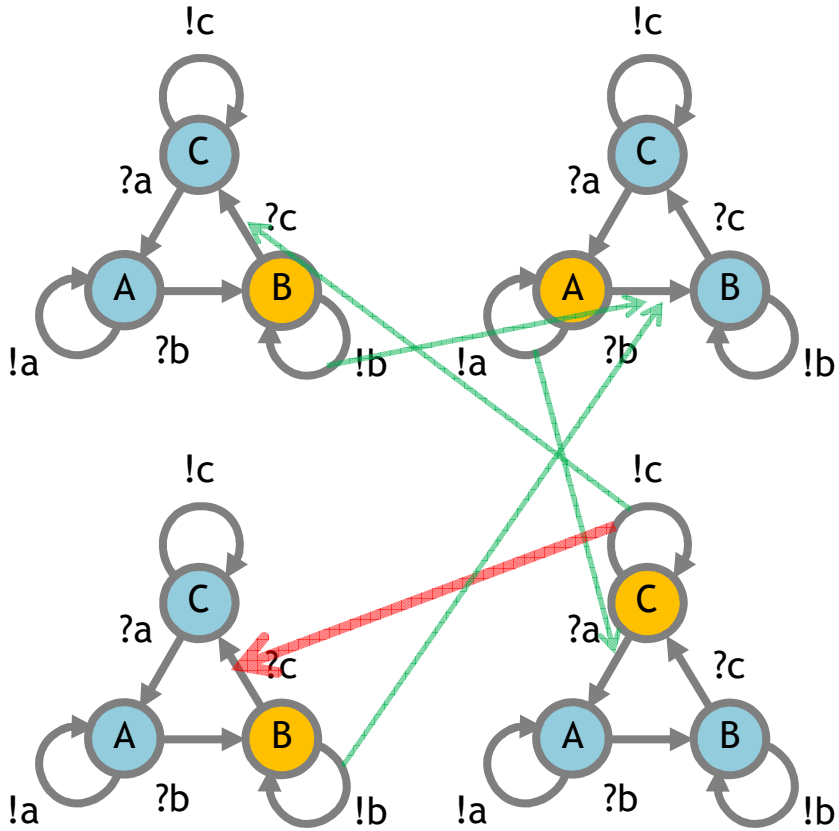


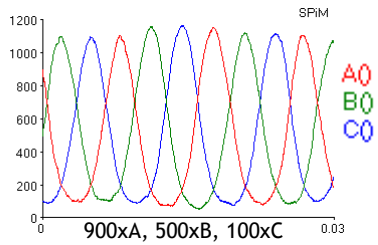


([A,B].[B,B])* |
([B,C].[C,C])* |
([C,A].[A,A])* |
A | A | B | C

This is a uniform population of identical automata,
but heterogeneous populations of interacting automata can be similarly handled.

Interacting Automata

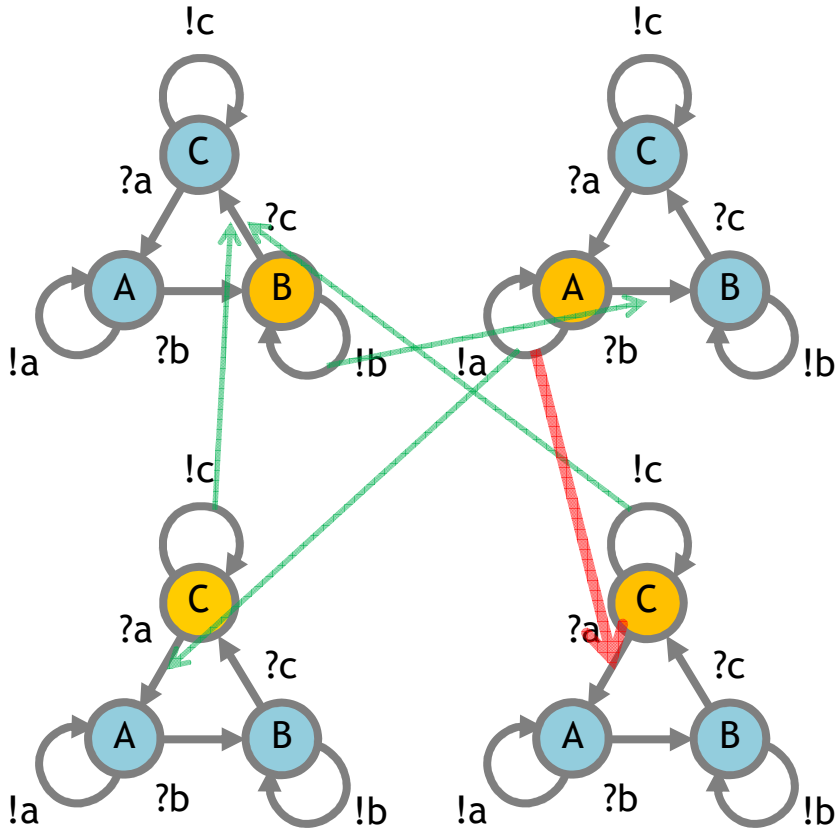


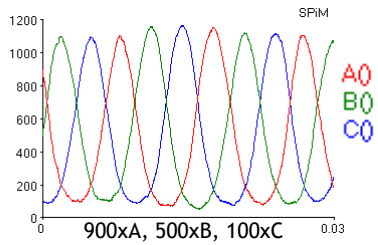


([A,B].[B,B])* |
([B,C].[C,C])* |
([C,A].[A,A])* |
A | B | B | C

This is a uniform population of identical automata,
but heterogeneous populations of interacting automata can be similarly handled.

Interacting Automata

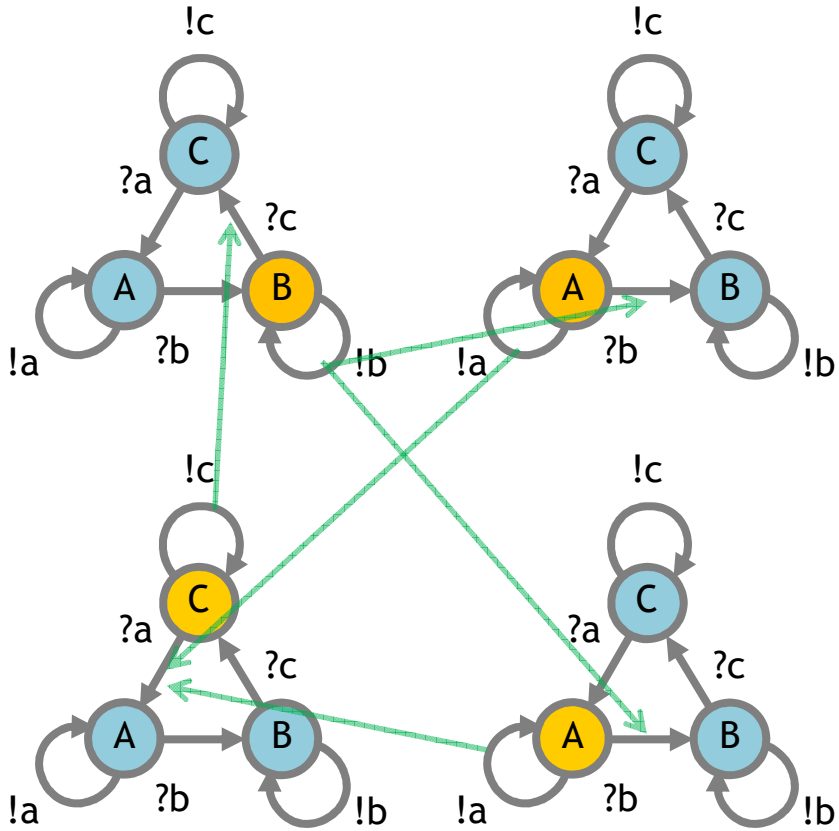


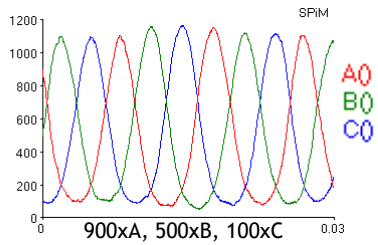


([A,B].[B,B])* |
([B,C].[C,C])* |
([C,A].[A,A])* |
A | B | C | C

This is a uniform population of identical automata,
but heterogeneous populations of interacting automata can be similarly handled.

Interacting Automata





([A,B].[B,B])* |
([B,C].[C,C])* |
([C,A].[A,A])* |
A | A | B | C

This is a uniform population of identical automata,
but heterogeneous populations of interacting automata can be similarly handled.

Strand Displacement Intermediate Language

Syntax

A. Syntax of DNA molecules *D*

Upper strand with sequence complementary to S

$\overline{S}$

`<S>`

Molecule with segments $G_1,\ldots,G_K$

$\overleftarrow{G_1 \quad G_2 \quad \ldots \quad G_K}$

`G1:G2:...:GK`

Parallel molecules $D_1,\ldots,D_K$

$D_1 \quad D_2 \quad \ldots \quad D_K$

`D1 | D2 | ... | DK`

Molecules *D* with private domains $N_1,\ldots,N_K$

$(N_1,\ldots,N_K)$ D

`new (N1,...,NK) D`

B. Syntax of DNA segments *G*

Lower strand with toehold N^c

$\underline{N^c}$

`N^c`

Double strand with sequence S and overhangs L, R

L R S

`<L>[S]<R>`

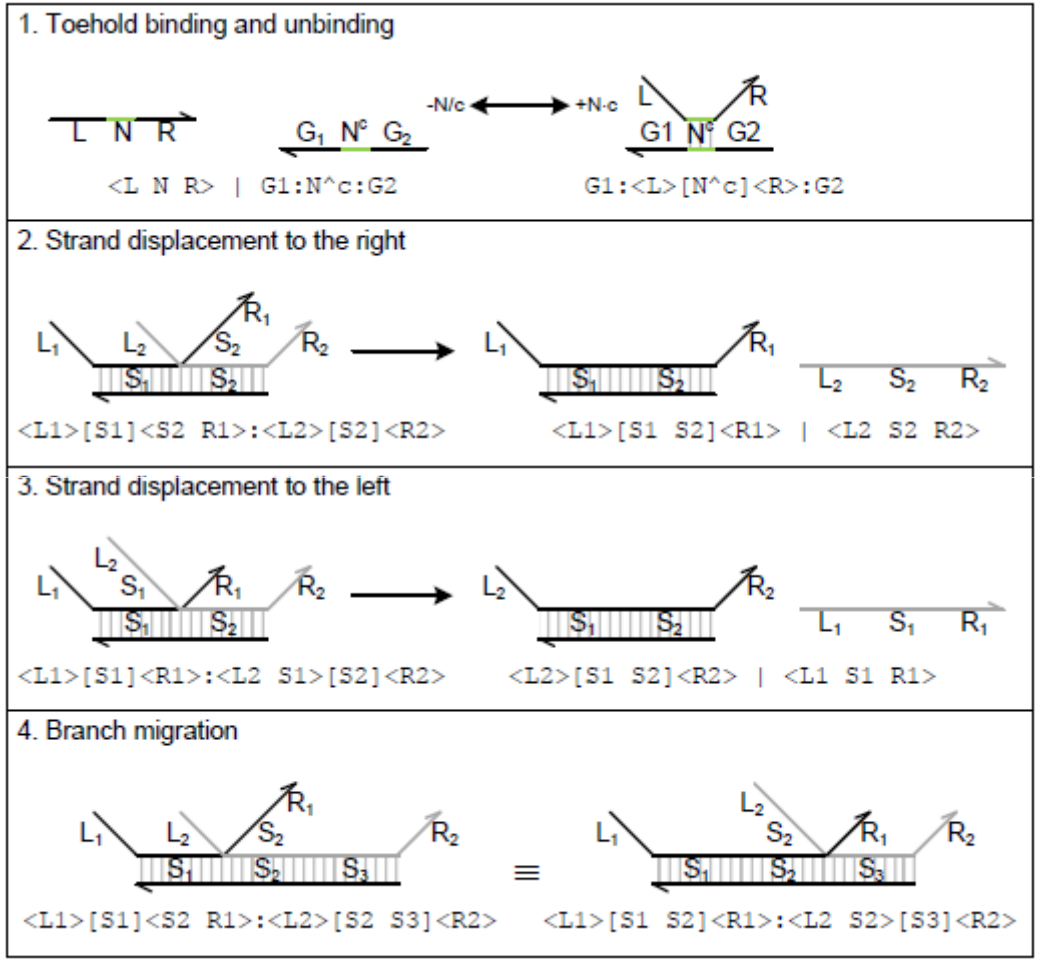
C. Syntax of DNA sequences *S,L,R*

Sequence of domains $O_1,\ldots,O_K$

$O_1 \quad O_2 \quad \ldots \quad O_K$

`O1 O2 ... OK`

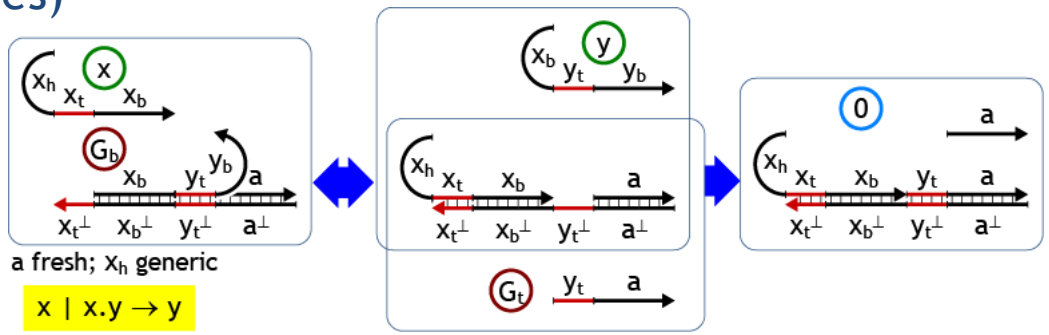
Dynamics

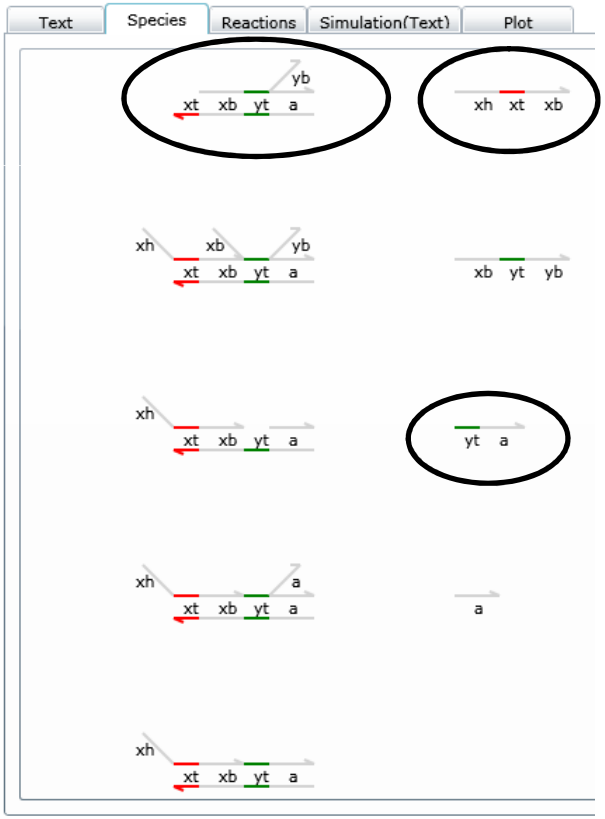

1. Toehold binding and unbinding
L N R
G1 N^c G2
-N/c
+N-c
L R
G1 N^c G2
<L N R> | G1:N^c:G2
G1:<L>[N^c]<R>:G2
2. Strand displacement to the right
L1 L2 S1 S2 R1 R2
L1 S1 S2 R1
L2 S2 R2
<L1>[S1]<S2 R1>:<L2>[S2]<R2>
<L1>[S1 S2]<R1> | <L2 S2 R2>
3. Strand displacement to the left
L1 L2 S1 S2 R1 R2
L2 S1 S2 R2
L1 S1 R1
<L1>[S1]<R1>:<L2 S1>[S2]<R2>
<L2>[S1 S2]<R2> | <L1 S1 R1>
4. Branch migration
L1 L2 S1 S2 S3 R1 R2
≡
L1 L2 S1 S2 S3 R1 R2
<L1>[S1]<S2 R1>:<L2>[S2 S3]<R2>
<L1>[S1 S2]<R1>:<L2 S2>[S3]<R2>

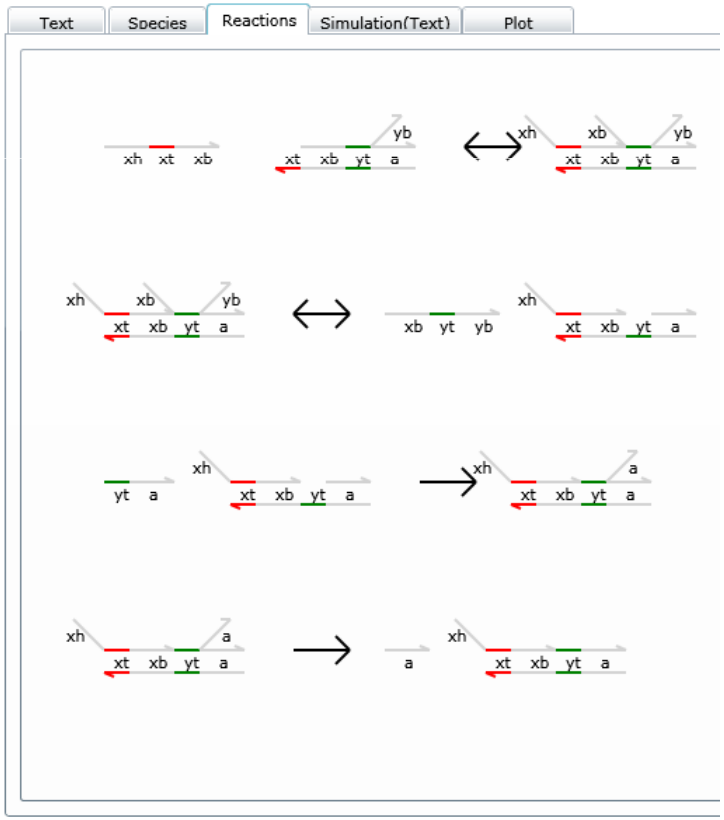
Strand Displacement Simulation Tool

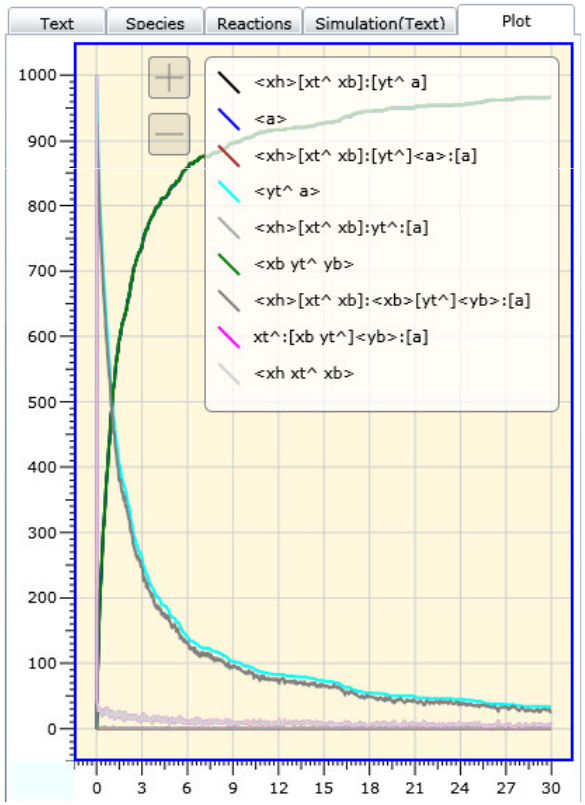
1 Transducer gate x.y (3 initial species)

```
directive sample 30.0 1000
new xt@1.0,1.0
new yt@1.0,1.0
( 1000 * <xh xt^ xb>
| 1000 * xt^:[xb yt^]<yb>:[a]
| 1000 * <yt^ a>
)
```





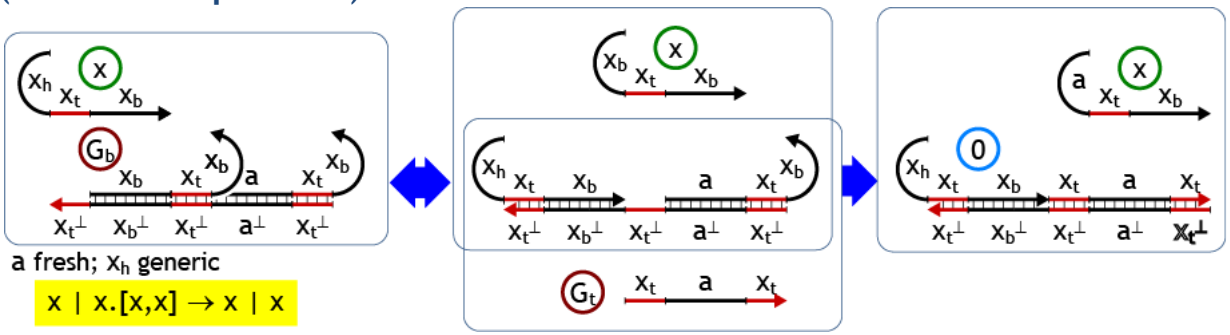




Strand Displacement Simulation Tool

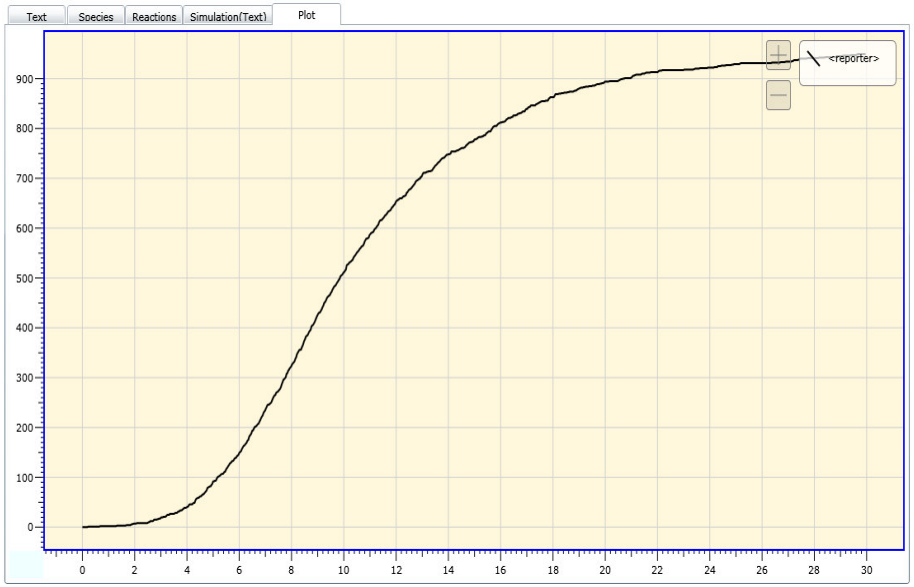
Fork Chain Reaction x.[x,x] (3 initial species)

```
directive sample 30.0 1000
directive plot "<reporter>"
 new xt@ 1.0 , 1.0
( 1 * <xh xt^ xb>
| 1000 * xt^:[xb xt^]<xb>:[a xt^]<xb>:[reporter]
| 1000 * <xt^ a xt^ reporter>
)
```



28 Species, 22 Reactions

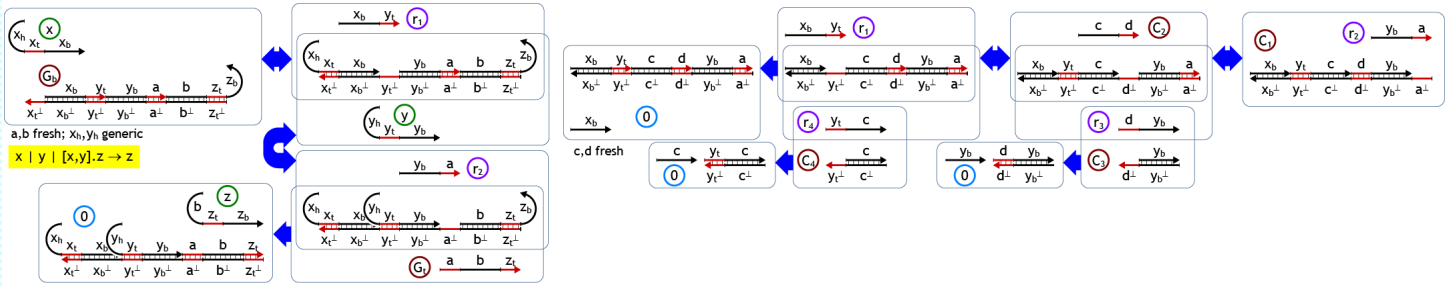
```
<xh xt^ xb>
<xb xt^ xb>
<xt^ a xt^ reporter>
<reporter>
<a xt^ xb>
xt^:[xb xt^]<xb>:[a xt^]<xb>:[reporter]
<xh>[xt^ xb]:<xb>[xt^]<xb>:[a xt^]<xb>:[reporter]
<xh>[xt^ xb]:xt^:[a xt^]<xb>:[reporter]
<xh>[xt^ xb]:<xh>[xt^]<xb>:[a xt^]<xb>:[reporter]
<xb>[xt^ xb]:<xb>[xt^]<xb>:[a xt^]<xb>:[reporter]
<xb>[xt^ xb]:xt^:[a xt^]<xb>:[reporter]
<xb>[xt^ xb]:<xh>[xt^]<xb>:[a xt^]<xb>:[reporter]
<xb>[xt^ xb]:[xt^ a]<xt^ reporter>:<a>[xt^]<xb>:[reporter]
<xb>[xt^ xb]:[xt^ a xt^]<reporter>:[reporter]
<xb>[xt^ xb]:[xt^ a xt^ reporter]
<xb>[xt^ xb]:<a>[xt^]<xb>:[a xt^]<xb>:[reporter]
<xh>[xt^ xb]:<a>[xt^]<xb>:[a xt^]<xb>:[reporter]
<a>[xt^ xb]:<xb>[xt^]<xb>:[a xt^]<xb>:[reporter]
<a>[xt^ xb]:xt^:[a xt^]<xb>:[reporter]
<a>[xt^ xb]:<a>[xt^]<xb>:[a xt^]<xb>:[reporter]
<a>[xt^ xb]:[xt^ a]<xt^ reporter>:<a>[xt^]<xb>:[reporter]
<a>[xt^ xb]:[xt^ a xt^]<reporter>:[reporter]
<a>[xt^ xb]:[xt^ a xt^ reporter]
<a>[xt^ xb]:<xh>[xt^]<xb>:[a xt^]<xb>:[reporter]
<xh>[xt^ xb]:[xt^ a]<xt^ reporter>:<a>[xt^]<xb>:[reporter]
<xh>[xt^ xb]:[xt^ a xt^]<reporter>:[reporter]
<xh>[xt^ xb]:[xt^ a xt^ reporter]
[xt^]<a xt^ reporter>:[xb xt^]<xb>:[a xt^]<xb>:[reporter]

<xh xt^ xb> + xt^:[xb xt^]<xb>:[a xt^]<xb>:[reporter]  {rmxt^}<->{rxt^}  <xh>[xt^ xb]:<xb>[xt^]<xb>:[a xt^]<xb>:[reporter]
<xh>[xt^ xb]:<xb>[xt^]<xb>:[a xt^]<xb>:[reporter]  {rxt^}<->{rmxt^}  <xb xt^ xb> + <xh>[xt^ xb]:xt^:[a xt^]<xb>:[reporter]
<xh xt^ xb> + <xh>[xt^ xb]:xt^:[a xt^]<xb>:[reporter]  {rmxt^}<->{rxt^}  <xh>[xt^ xb]:<xh>[xt^]<xb>:[a xt^]<xb>:[reporter]
<xb xt^ xb> + xt^:[xb xt^]<xb>:[a xt^]<xb>:[reporter]  {rmxt^}<->{rxt^}  <xb>[xt^ xb]:<xb>[xt^]<xb>:[a xt^]<xb>:[reporter]
<xb>[xt^ xb]:<xb>[xt^]<xb>:[a xt^]<xb>:[reporter]  {rxt^}<->{rmxt^}  <xb xt^ xb> + <xb>[xt^ xb]:xt^:[a xt^]<xb>:[reporter]
<xh xt^ xb> + <xb>[xt^ xb]:xt^:[a xt^]<xb>:[reporter]  {rmxt^}<->{rxt^}  <xh>[xt^ xb]:<xb>[xt^]<xb>:[a xt^]<xb>:[reporter]
<xt^ a xt^ reporter> + xt^:[xb xt^]<xb>:[a xt^]<xb>:[reporter]  {rmxt^}<->{rxt^}  [xt^]<a xt^ reporter>:[xb xt^]<xb>:[a xt^]<xb>:[reporter]
<xt^ a xt^ reporter> + <xh>[xt^ xb]:xt^:[a xt^]<xb>:[reporter]  ->{rxt^}  <xh>[xt^ xb]:[xt^ a]<xt^ reporter>:<a>[xt^]<xb>:[reporter]
<xt^ a xt^ reporter> + <xb>[xt^ xb]:xt^:[a xt^]<xb>:[reporter]  ->{rxt^}  <xb>[xt^ xb]:[xt^ a]<xt^ reporter>:<a>[xt^]<xb>:[reporter]
<xb>[xt^ xb]:[xt^ a]<xt^ reporter>:<a>[xt^]<xb>:[reporter]  ->{rm}  <a xt^ xb> + <xb>[xt^ xb]:[xt^ a xt^]<reporter>:[reporter]
<xb>[xt^ xb]:[xt^ a xt^]<reporter>:[reporter]  ->{rm}  <reporter> + <xb>[xt^ xb]:[xt^ a xt^ reporter]
<a xt^ xb> + xt^:[xb xt^]<xb>:[a xt^]<xb>:[reporter]  {rmxt^}<->{rxt^}  <a>[xt^ xb]:<xb>[xt^]<xb>:[a xt^]<xb>:[reporter]
<a xt^ xb> + <xh>[xt^ xb]:xt^:[a xt^]<xb>:[reporter]  {rmxt^}<->{rxt^}  <xh>[xt^ xb]:<a>[xt^]<xb>:[a xt^]<xb>:[reporter]
<a xt^ xb> + <xb>[xt^ xb]:xt^:[a xt^]<xb>:[reporter]  {rmxt^}<->{rxt^}  <xb>[xt^ xb]:<a>[xt^]<xb>:[a xt^]<xb>:[reporter]
<a>[xt^ xb]:<xb>[xt^]<xb>:[a xt^]<xb>:[reporter]  {rxt^}<->{rmxt^}  <xb xt^ xb> + <a>[xt^ xb]:xt^:[a xt^]<xb>:[reporter]
<xh xt^ xb> + <a>[xt^ xb]:xt^:[a xt^]<xb>:[reporter]  {rmxt^}<->{rxt^}  <a>[xt^ xb]:<xh>[xt^]<xb>:[a xt^]<xb>:[reporter]
<xt^ a xt^ reporter> + <a>[xt^ xb]:xt^:[a xt^]<xb>:[reporter]  ->{rxt^}  <a>[xt^ xb]:[xt^ a]<xt^ reporter>:<a>[xt^]<xb>:[reporter]
<a xt^ xb> + <a>[xt^ xb]:xt^:[a xt^]<xb>:[reporter]  {rmxt^}<->{rxt^}  <a>[xt^ xb]:<a>[xt^]<xb>:[a xt^]<xb>:[reporter]
<a>[xt^ xb]:[xt^ a]<xt^ reporter>:<a>[xt^]<xb>:[reporter]  ->{rm}  <a xt^ xb> + <a>[xt^ xb]:[xt^ a xt^]<reporter>:[reporter]
<a>[xt^ xb]:[xt^ a xt^]<reporter>:[reporter]  ->{rm}  <reporter> + <a>[xt^ xb]:[xt^ a xt^ reporter]
<xh>[xt^ xb]:[xt^ a]<xt^ reporter>:<a>[xt^]<xb>:[reporter]  ->{rm}  <a xt^ xb> + <xh>[xt^ xb]:[xt^ a xt^]<reporter>:[reporter]
<xh>[xt^ xb]:[xt^ a xt^]<reporter>:[reporter]  ->{rm}  <reporter> + <xh>[xt^ xb]:[xt^ a xt^ reporter]
```



Strand Displacement Simulation Tool

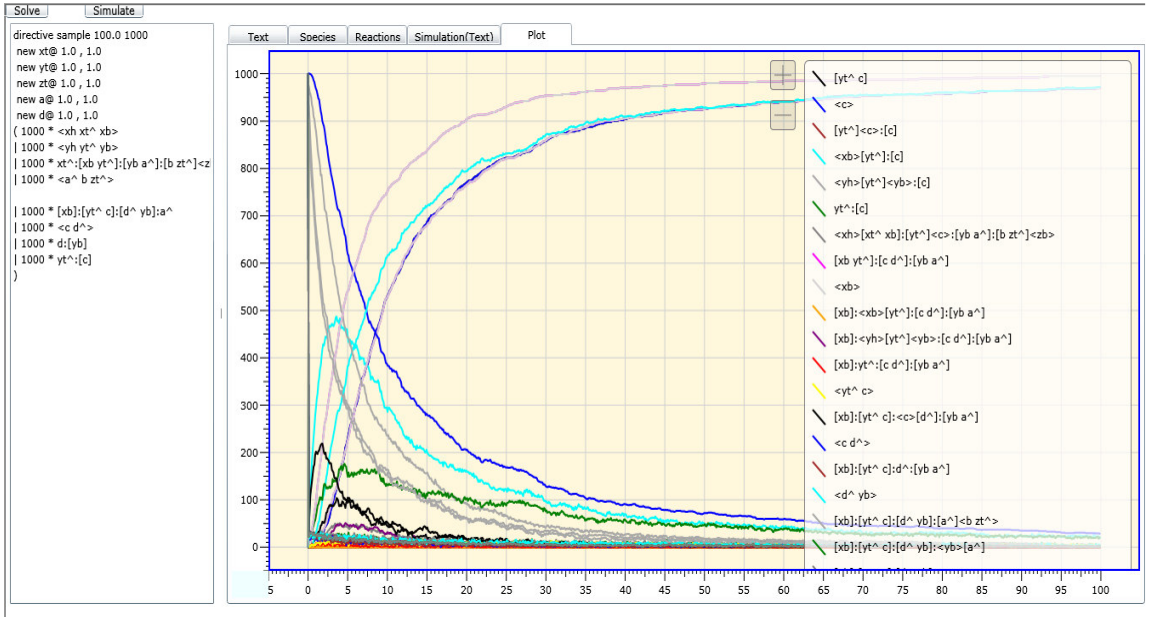
1 Join gate with garbage collection [x,y].z (8 initial species)

```
directive sample 1000.0 1000
 new xt@ 1.0 , 1.0
 new yt@ 1.0 , 1.0
 new zt@ 1.0 , 1.0
 new a@ 1.0 , 1.0
 new d@ 1.0 , 1.0
( 1000 * <xh xt^ xb>
| 1000 * <yh yt^ yb>
| 1000 * xt^:[xb yt^]:[yb a^]:[b zt^]<zb>
| 1000 * <a^ b zt^>

| 1000 * [xb]:[yt^ c]:[d^ yb]:a^
| 1000 * <c d^>
| 1000 * d:[yb]
| 1000 * yt^:[c]
)
```

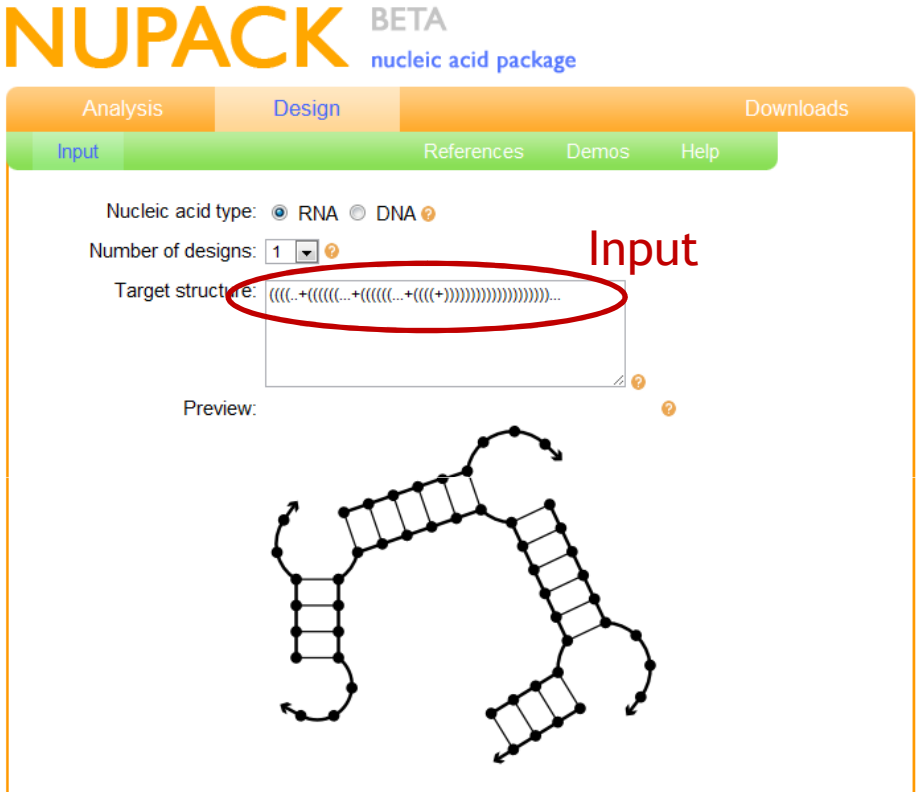


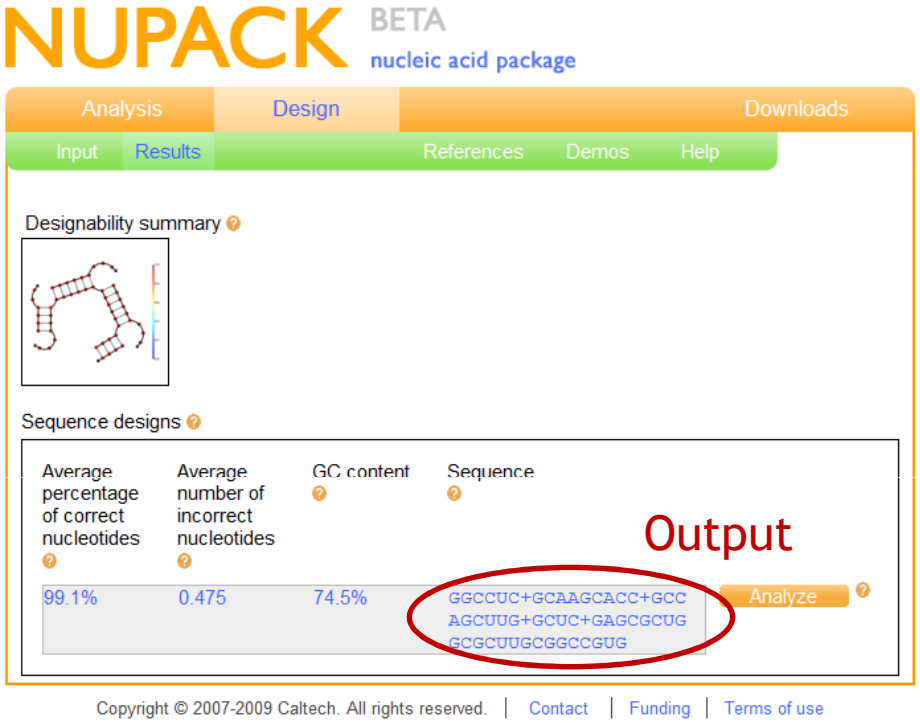
34 Species, 19 Reactions

```
<xh xt^ xb>
<yh yt^ yb>
<yb a^>
<xb yt^>
<a^ b zt^>
<b zt^ zb>
<d^ yb>
<c d^>
<xb>
<yt^ c>
<c>
xt^:[xb yt^]:[yb a^]:[b zt^]<zb>
<xh>[xt^ xb]:<xb>[yt^]:[yb a^]:[b zt^]<zb>
<xh>[xt^ xb]:yt^:[yb a^]:[b zt^]<zb>
<xh>[xt^ xb]:<yh>[yt^ yb]:<yb>[a^]:[b zt^]<zb>
<xh>[xt^ xb]:<yh>[yt^ yb]:a^:[b zt^]<zb>
<xh>[xt^ xb]:<yh>[yt^ yb]:[a^ b]<zt^>:<b>[zt^]<zb>
<xh>[xt^ xb]:<yh>[yt^ yb]:[a^ b zt^]
[xb]:[yt^ c]:[d^ yb]:a^
[xb]:[yt^ c]:[d^ yb]:[a^]<b zt^>
[xb]:[yt^ c]:[d^ yb]:<yb>[a^]
[xb]:[yt^ c]:d^:[yb a^]
[xb]:[yt^ c]:<c>[d^]:[yb a^]
[xb]:yt^:[c d^]:[yb a^]
[xb]:<xb>[yt^]:[c d^]:[yb a^]
[xb yt^]:[c d^]:[yb a^]
[xb]:<yh>[yt^]<yb>:[c d^]:[yb a^]
<xh>[xt^ xb]:[yt^]<c>:[yb a^]:[b zt^]<zb>
d:[yb]
yt^:[c]
[yt^]<c>:[c]
[yt^ c]
<xb>[yt^]:[c]
<yh>[yt^]<yb>:[c]

<xh xt^ xb> + xt^:[xb yt^]:[yb a^]:[b zt^]<zb>  {rmxt^}<->{rxt^}  <xh>[xt^ xb]:<xb>[yt^]:[yb a^]:[b zt^]<zb>
<xh>[xt^ xb]:<xb>[yt^]:[yb a^]:[b zt^]<zb>  {ryt^}<->{rmyt^}  <xb yt^> + <xh>[xt^ xb]:yt^:[yb a^]:[b zt^]<zb>
<yh yt^ yb> + <xh>[xt^ xb]:yt^:[yb a^]:[b zt^]<zb>  {rmyt^}<->{ryt^}  <xh>[xt^ xb]:<yh>[yt^ yb]:<yb>[a^]:[b zt^]<zb>
<xh>[xt^ xb]:<yh>[yt^ yb]:<yb>[a^]:[b zt^]<zb>  {ra^}<->{rma^}  <yb a^> + <xh>[xt^ xb]:<yh>[yt^ yb]:a^:[b zt^]<zb>
<a^ b zt^> + <xh>[xt^ xb]:<yh>[yt^ yb]:a^:[b zt^]<zb>  ->{ra^}  <xh>[xt^ xb]:<yh>[yt^ yb]:[a^ b]<zt^>:<b>[zt^]<zb>
<xh>[xt^ xb]:<yh>[yt^ yb]:[a^ b]<zt^>:<b>[zt^]<zb>  ->{rm}  <b zt^ zb> + <xh>[xt^ xb]:<yh>[yt^ yb]:[a^ b zt^]
<yb a^> + [xb]:[yt^ c]:[d^ yb]:a^  {rma^}<->{ra^}  [xb]:[yt^ c]:[d^ yb]:<yb>[a^]
<a^ b zt^> + [xb]:[yt^ c]:[d^ yb]:a^  {rma^}<->{ra^}  [xb]:[yt^ c]:[d^ yb]:[a^]<b zt^>
[xb]:[yt^ c]:[d^ yb]:<yb>[a^]  {rd^}<->{rmd^}  <d^ yb> + [xb]:[yt^ c]:d^:[yb a^]
<c d^> + [xb]:[yt^ c]:d^:[yb a^]  {rmd^}<->{rd^}  [xb]:[yt^ c]:<c>[d^]:[yb a^]
[xb]:[yt^ c]:<c>[d^]:[yb a^]  {ryt^}<->{rmyt^}  <yt^ c> + [xb]:yt^:[c d^]:[yb a^]
<yh yt^ yb> + [xb]:yt^:[c d^]:[yb a^]  {rmyt^}<->{ryt^}  [xb]:<yh>[yt^]<yb>:[c d^]:[yb a^]
<xb yt^> + [xb]:yt^:[c d^]:[yb a^]  ->{ryt^}  [xb]:<xb>[yt^]:[c d^]:[yb a^]
[xb]:<xb>[yt^]:[c d^]:[yb a^]  ->{rm}  <xb> + [xb yt^]:[c d^]:[yb a^]
<yt^ c> + <xh>[xt^ xb]:yt^:[yb a^]:[b zt^]<zb>  {rmyt^}<->{ryt^}  <xh>[xt^ xb]:[yt^]<c>:[yb a^]:[b zt^]<zb>
<yh yt^ yb> + yt^:[c]  {rmyt^}<->{ryt^}  <yh>[yt^]<yb>:[c]
<xb yt^> + yt^:[c]  {rmyt^}<->{ryt^}  <xb>[yt^]:[c]
<yt^ c> + yt^:[c]  ->{ryt^}  [yt^]<c>:[c]
[yt^]<c>:[c]  ->{rm}  <c> + [yt^ c]
```



Sequence Design





Conclusions

Conclusion

- Nucleic Acids
  - Programmable matter
- DNA Strand Displacement
  - A computational mechanism at the molecular level
- DNA Compilation
  - High-level languages (Boolean Networks, Petri Nets, Interacting Automata)
  - Intermediate languages.
  - Sequence generation.
- Tools
  - Thermodynamic analysis.
  - Simulation.
  - Verification (not yet).

Abstract

Nucleic acids (DNA/RNA) encode information digitally, and are currently the only truly 'user-programmable' entities at the molecular scale. They can be used to manufacture nano-scale structures, produce physical forces, act as sensors and actuators, and do computation in between. Eventually we will be able to interface then with biological machinery to detect and cure diseases at the cellular level under program control.

The technology to create and manipulate them has existed for many years, but the imagination necessary to exploit them has been evolving slowly. Recently, some very simple computational schemes have been developed that are autonomous (run on their own once started) and involve only short (easily synthesizable) DNA strands with no other complex molecules. To get this started, one emails some short character strings to a company to get the DNA strands built, mixes them up, and reads the output (fluorescence) with a camera. And yes, this can be done in your kitchen, more or less.

But of course we need programming tools. Molecular design is required to produce molecules that fold, or do not fold, or stick, or do not stick in the desired ways: this can be achieved fairly predictably only for DNA/RNA (e.g. not for proteins). On that basis one can design various kinds of 'logic gates' and 'computational architectures', which is where much of the imagination is currently needed.

Then one needs programming languages both at the level of gate implementation (where Andrew Phillips in Cambridge is building a strand-level language and simulator), and at the level of circuit implementation (where I will describe a Strand Algebra for implementing e.g. automata and Petri nets). Since DNA computation is massively concurrent, some tricky and yet familiar issues arise, like having to formally verify gate designs to avoid subtle deadlocks and race conditions, and having to design high-level languages that exploit concurrency and stochasticity.