Molecular Programming

Luca Cardelli

Microsoft Research

Southampton, 2009-07-10

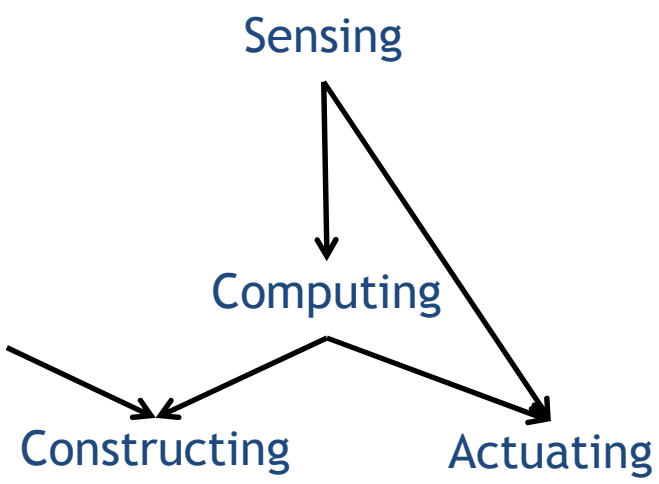
http://lucacardelli.name

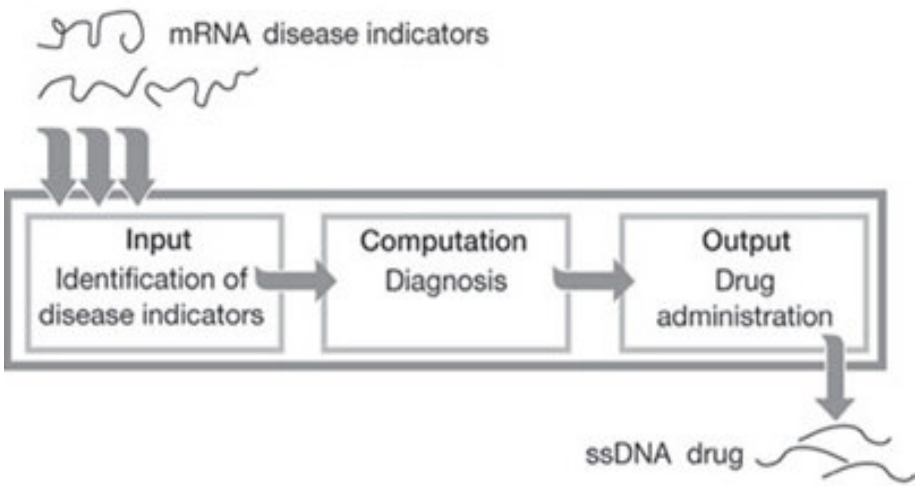
DNA Nanotechonology

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

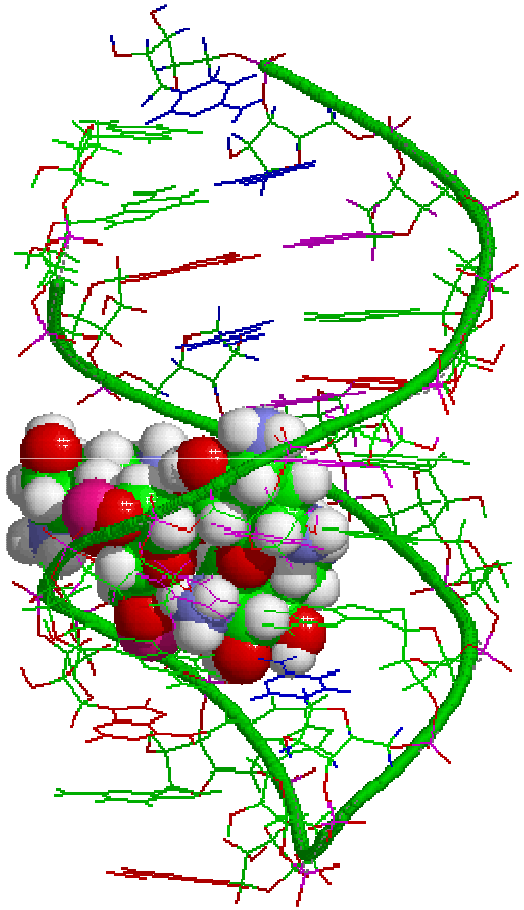
- Nucleic Acids (DNA/RNA)
  - Probably the only materials that can perform all these functions.
  - Technology relatively well developed.
  - Can interface to biological entities.

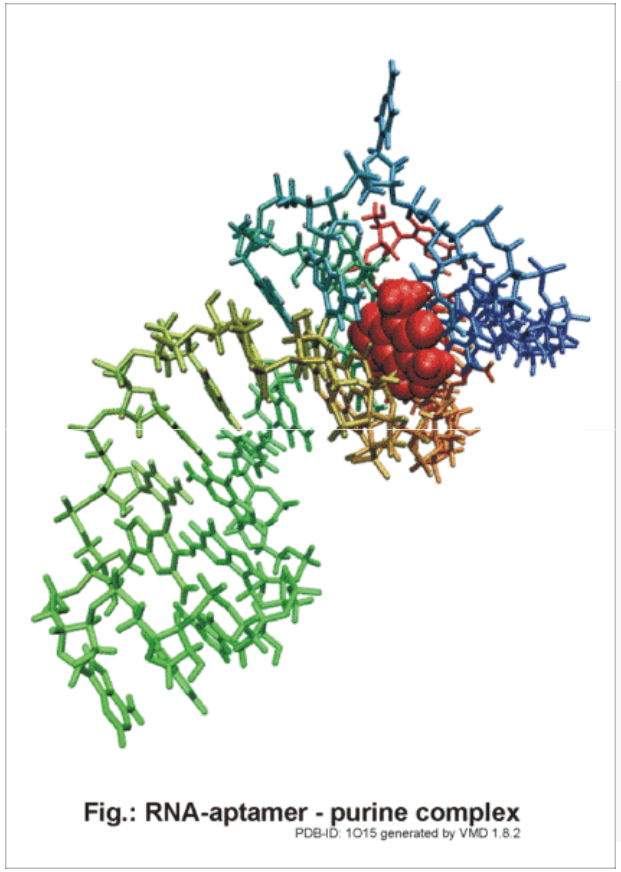


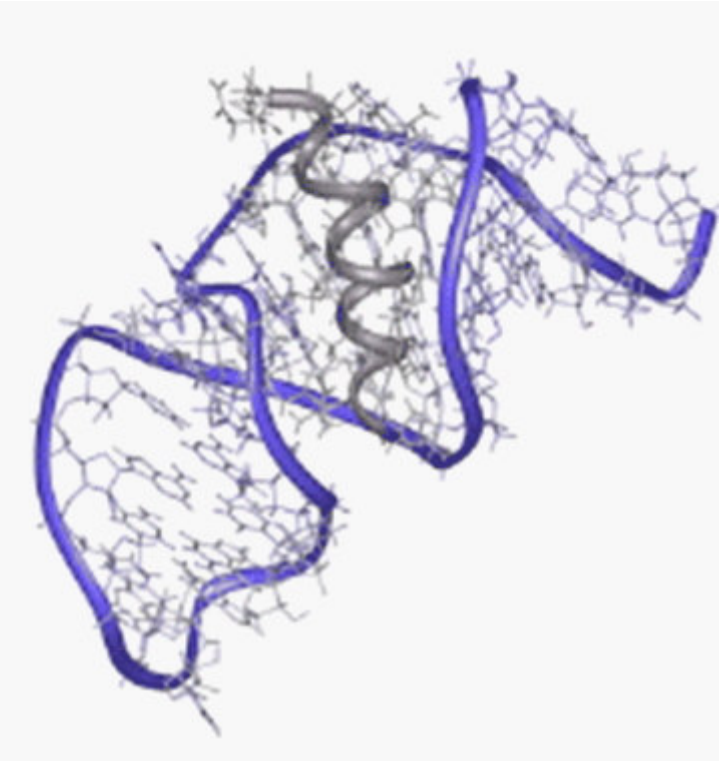


Y. Benenson *et al.*, *An autonomous molecular computer for logical control of gene expressionNature* 429, 423-429 (2004)

Aptamers (Sensors)







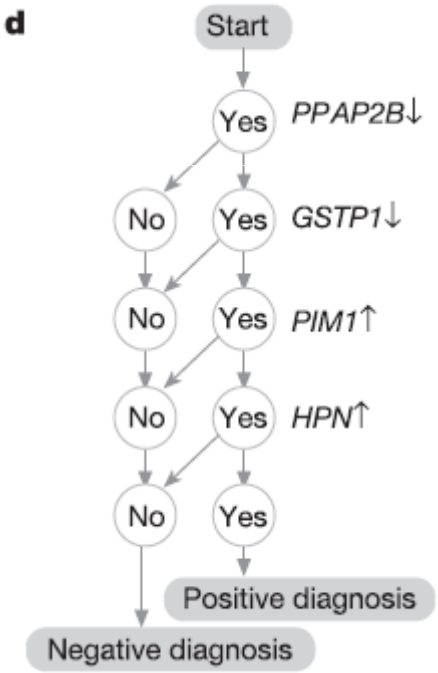
Computation: Curing Cancer with one AND Gate

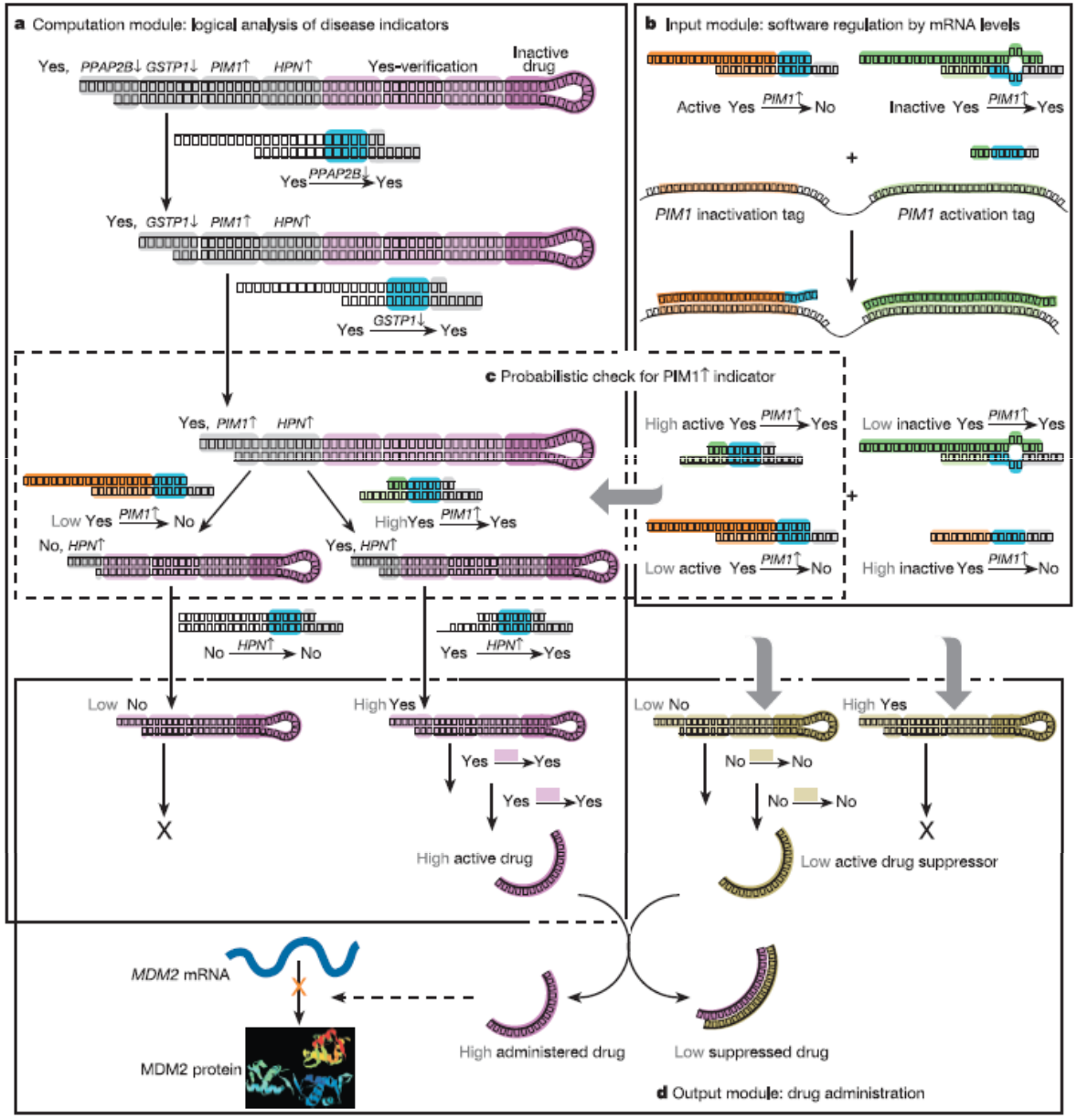
letters to nature

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

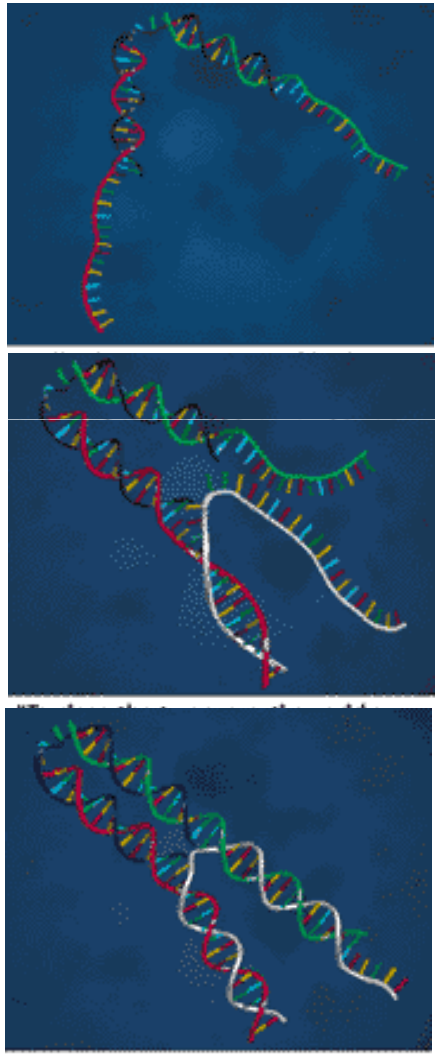




Actuators

DNA Tweezers

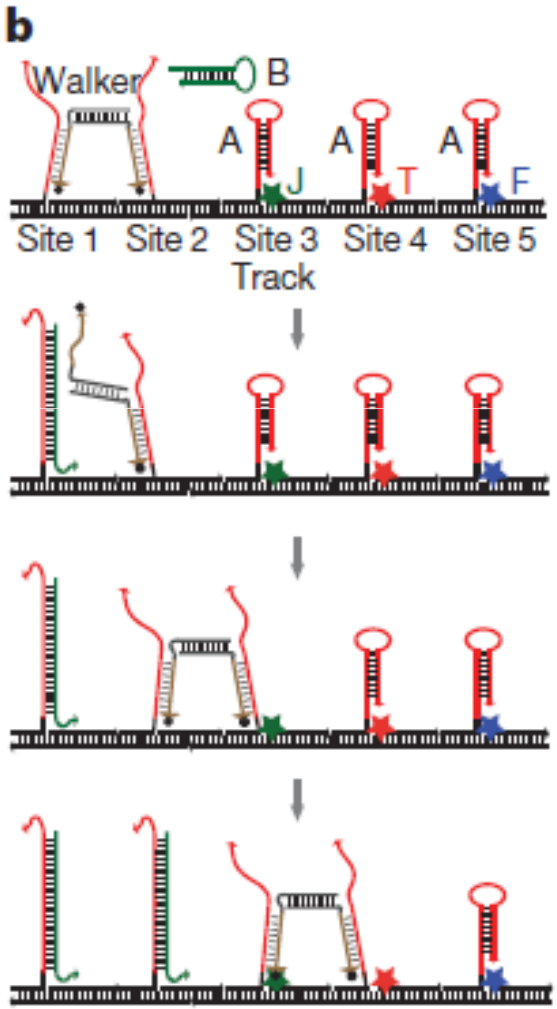
(Yurke & Turberfield, Nature 2000)



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

DNA Walkers

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



Compositionality

- Sensors and Actuators at the 'edge' of the system
  - They can use disparate kinds of inputs (sensors) and outputs (actuators)

- The 'kernel' of the system computes
  - Must use uniform inputs and outputs

- Compositionality in the kernel
  - Supporting 'arbitrary' computing complexity
  - The output of each computing components must be the same kind of 'signal' as the input

  - If the inputs are voltages, the outputs must be voltages
  - If the inputs are proteins, the outputs must be proteins
  - If the outputs are photons the inputs must be photons
  - If the inputs are DNA, the outputs must be DNA

  - What should our nano-signals be?

What does DNA Compute?

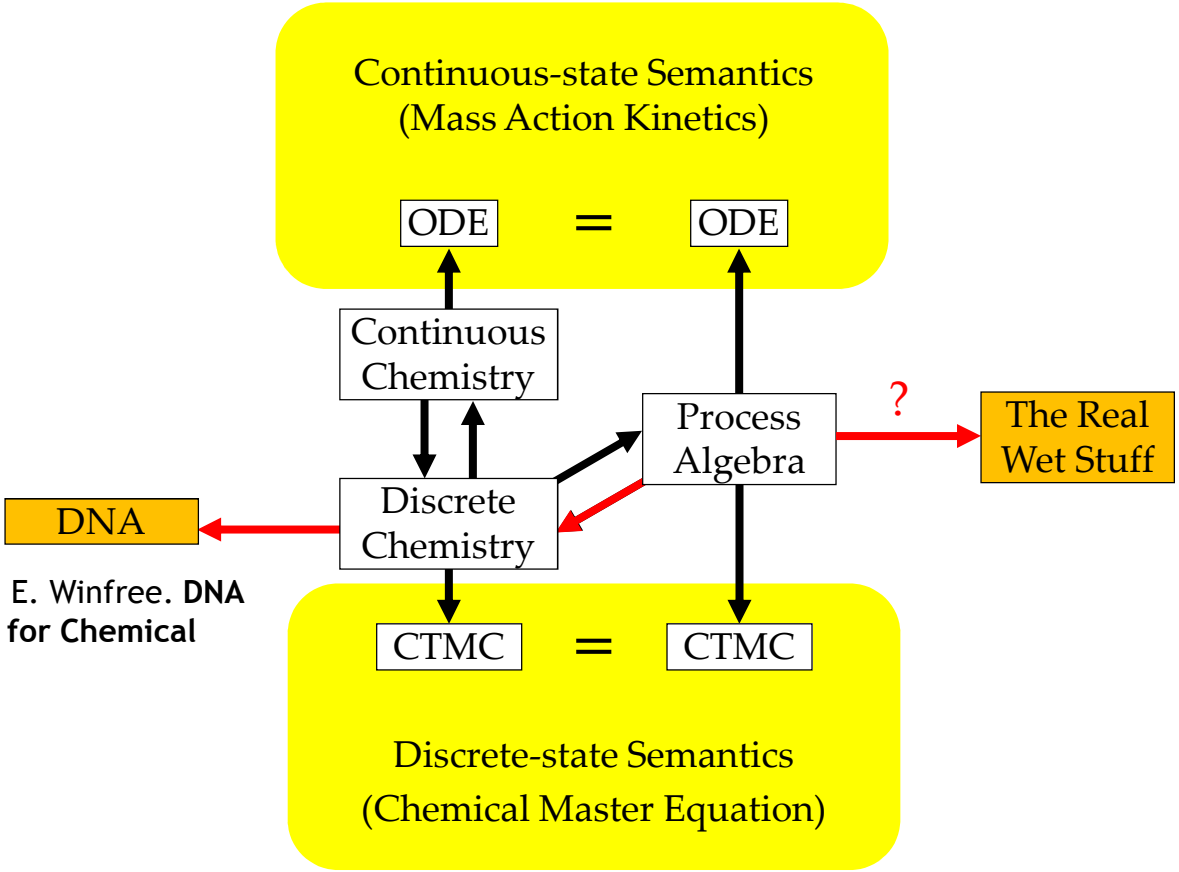
- Electronics has *electrons*
  - All electrons are the same
  - All you can do is see if you have *few* ('False') or *lots* ('True') of electrons
  - Hence Boolean logic is at the basis of digital circuit design
  - Symbolic and numeric computation has to be encoded above that
  - But mostly we want to compute with symbols and numbers, not with Booleans

- DNA computing has *symbols* (DNA words)
  - DNA words are not all the same
  - Symbolic computation can be done *directly*
  - We can also directly use molecular concurrency

- Process Algebra as the 'Boolean Algebra' of DNA Computing
  - What are the 'gates' of symbolic concurrent computation?
  - That's what Process Algebra is about
  - (Process Algebra comes from the theory of concurrent systems)

Implementing "Arbitrary" Computing Functions

Molecules as Automata (DNA14 Invited Talk)



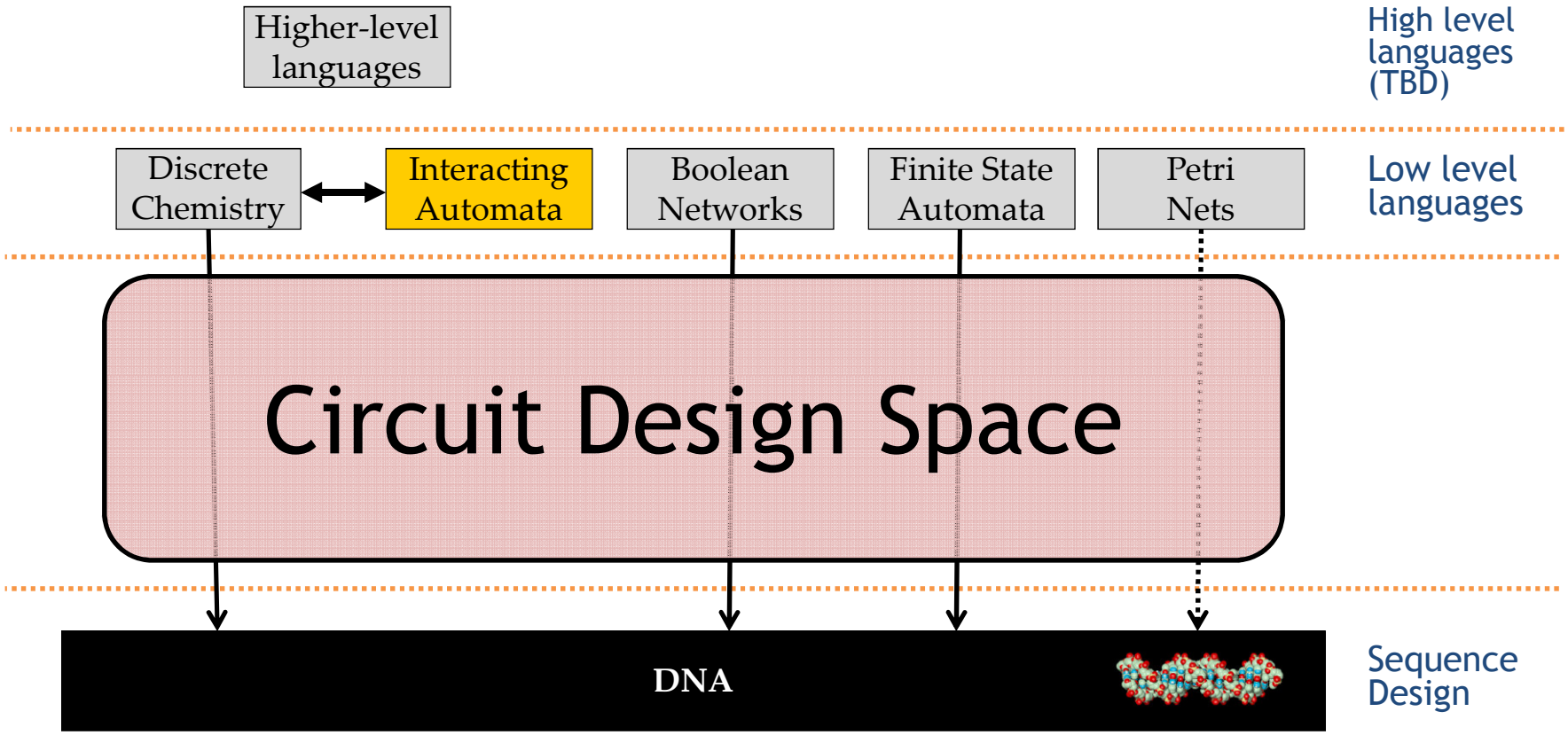
D. Soloveichik, G. Seelig, E. Winfree. **DNA as a Universal Substrate for Chemical Kinetics.** Proc. DNA14.

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

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

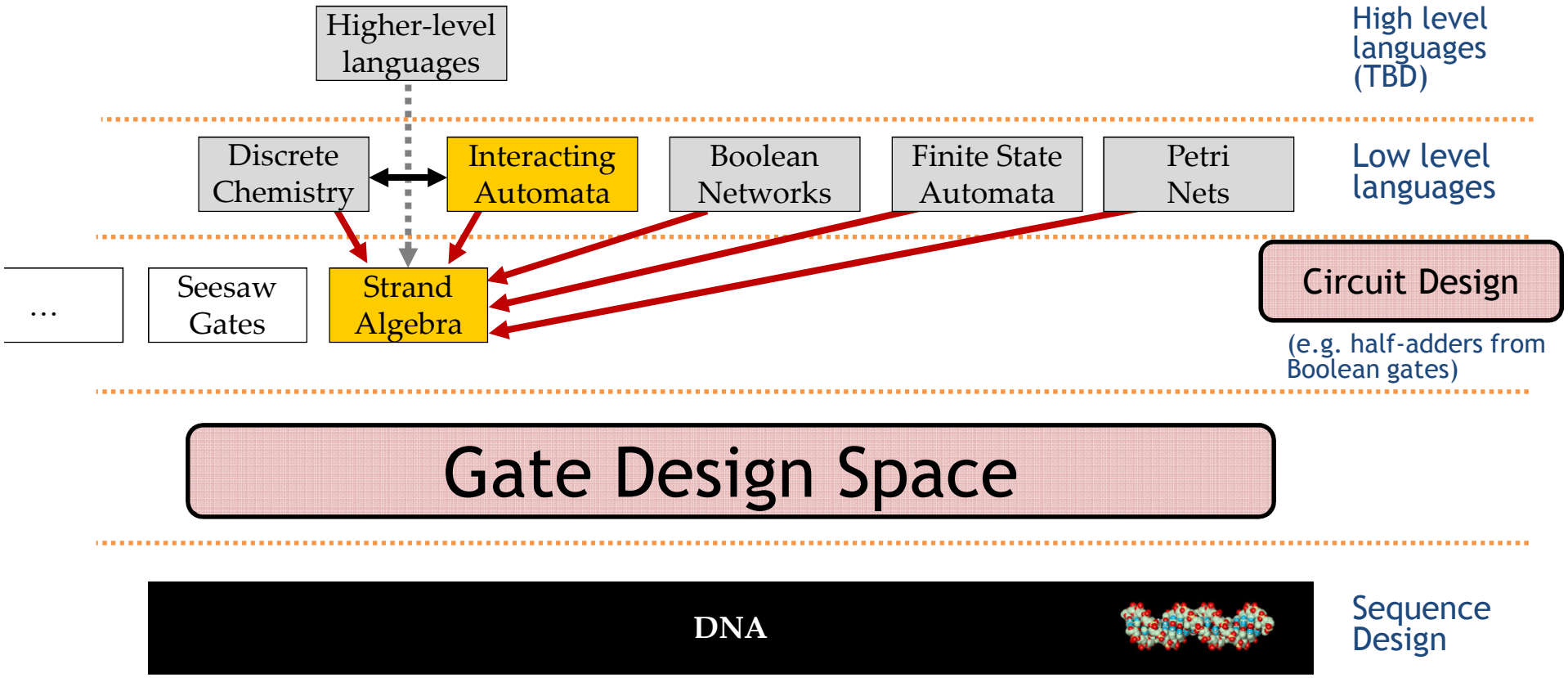
DNA Compilation

Separating Circuit Design from Gate Design



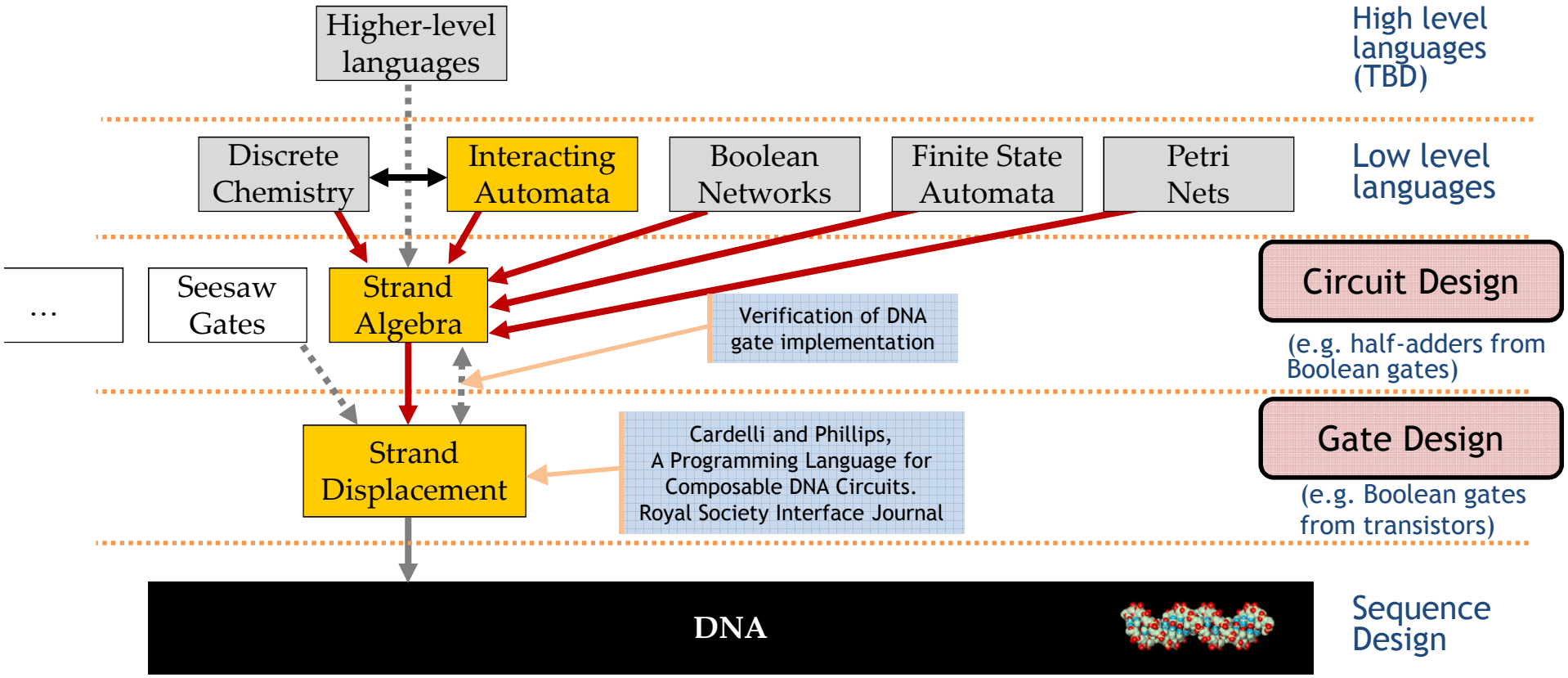
DNA Compilation

Separating Circuit Design from Gate Design



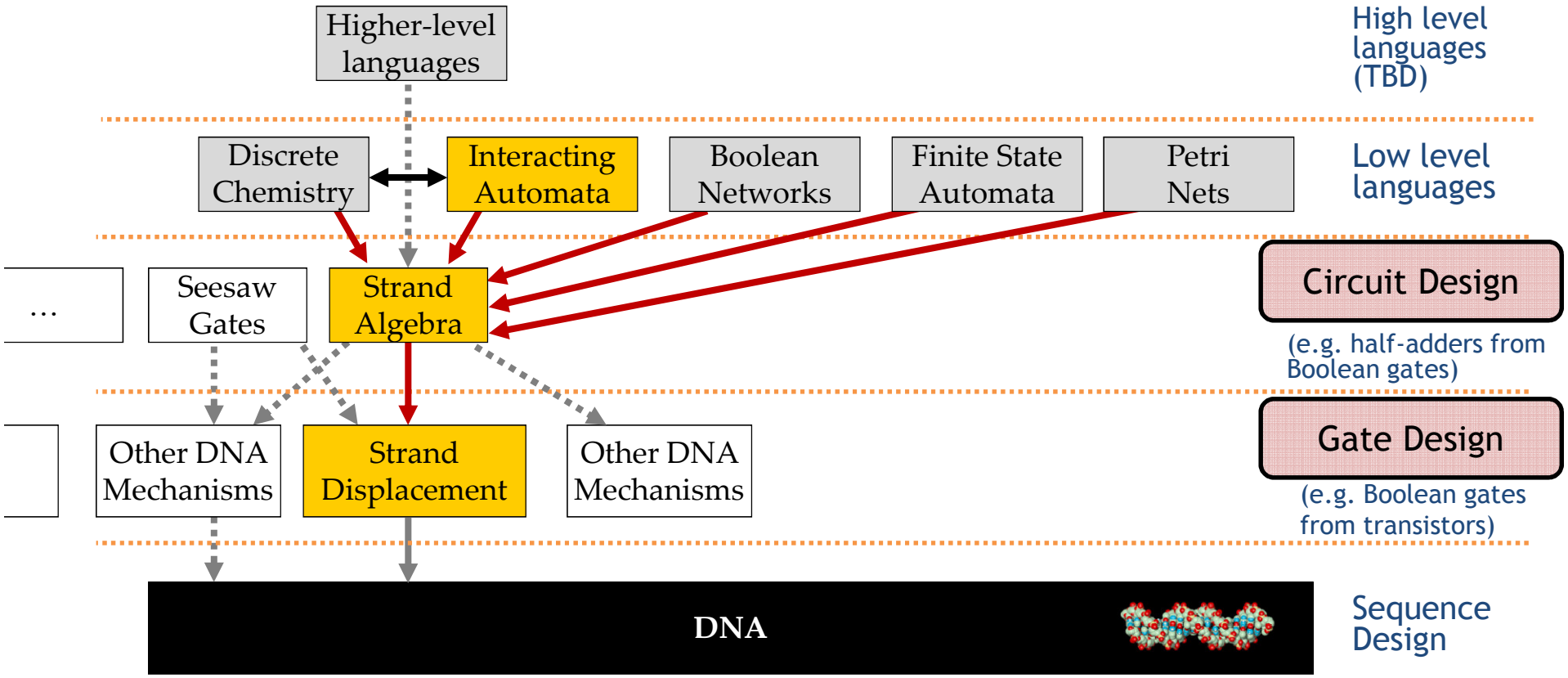
DNA Compilation

Separating Circuit Design from Gate Design



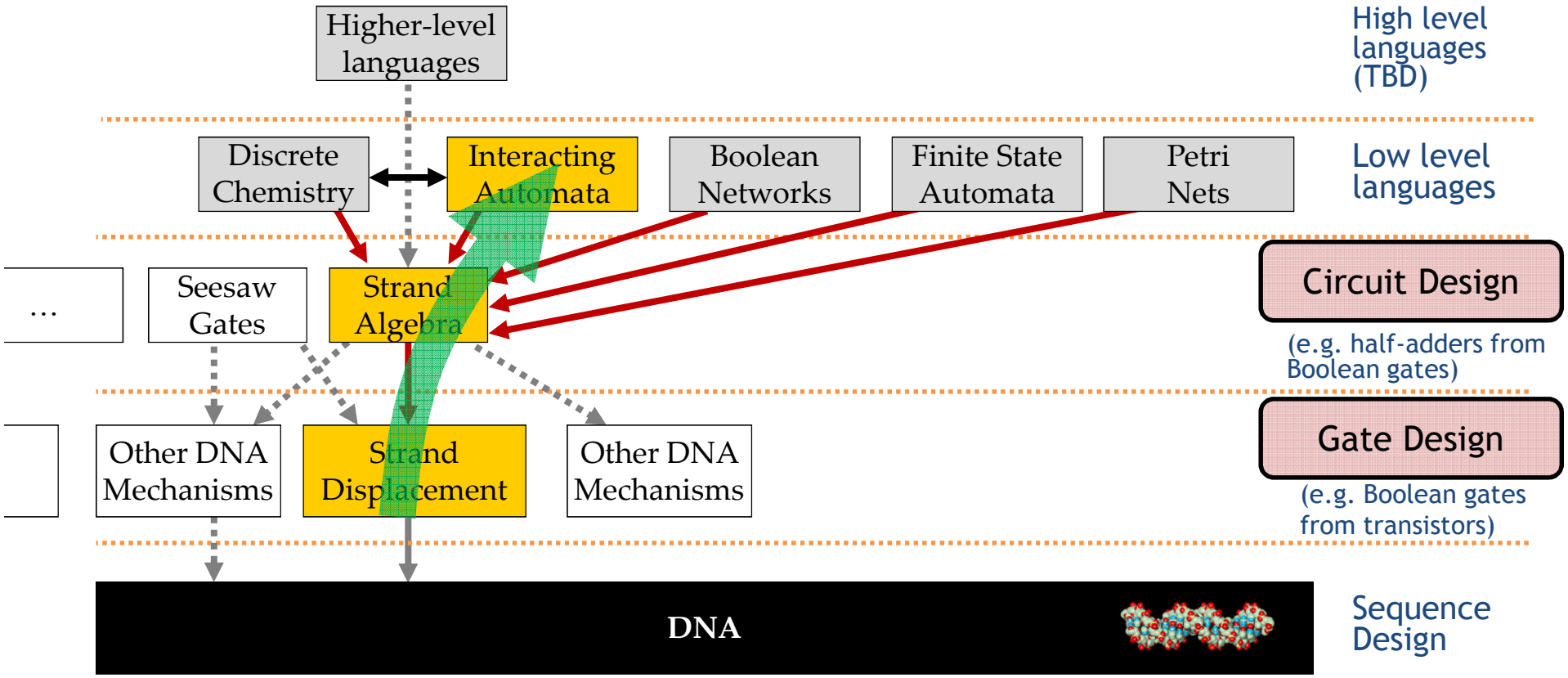
DNA Compilation

Separating Circuit Design from Gate Design



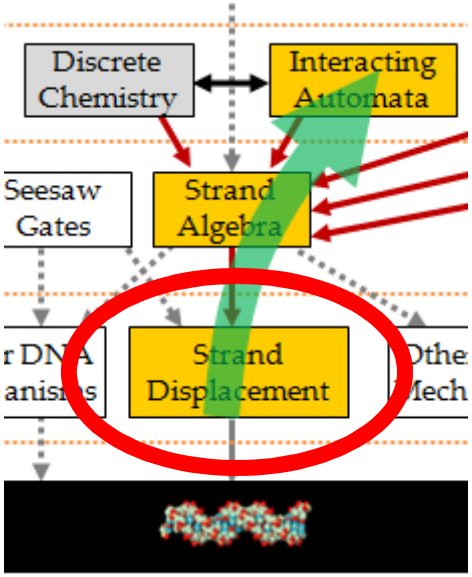
DNA Compilation

Separating Circuit Design from Gate Design



Rest of the talk: bottom up

Toehold Mediated Strand Displacement



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

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

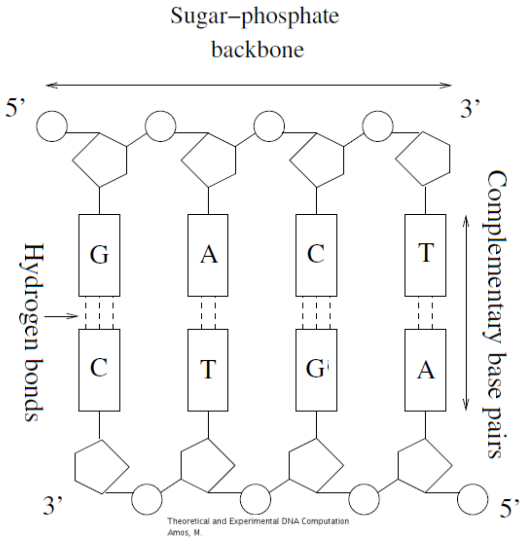
3' end

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

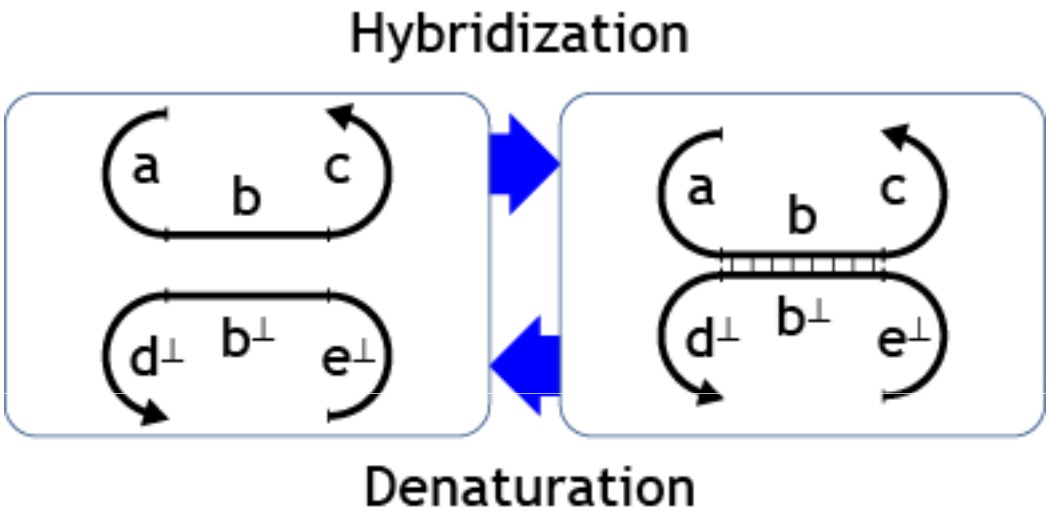
all written from 5' to 3'

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

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

Hybridization

a,b,c, etc. denote DNA (sub)sequences
with Watson-Crick complements $a^{\perp}$,$b^{\perp}$,$c^{\perp}$, etc.



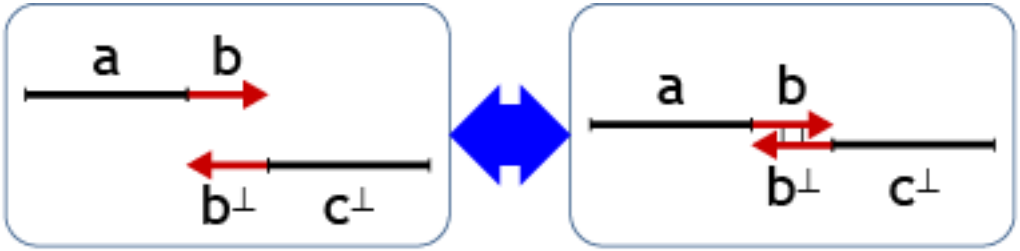
Hybridization is also called annealing; denaturation is also called melting.

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.

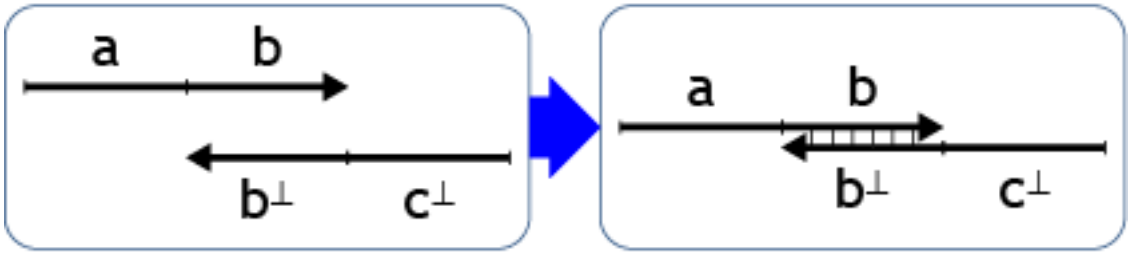
Gate Elements: Short and Long DNA Segments

Short (red) segments



Reversible Binding

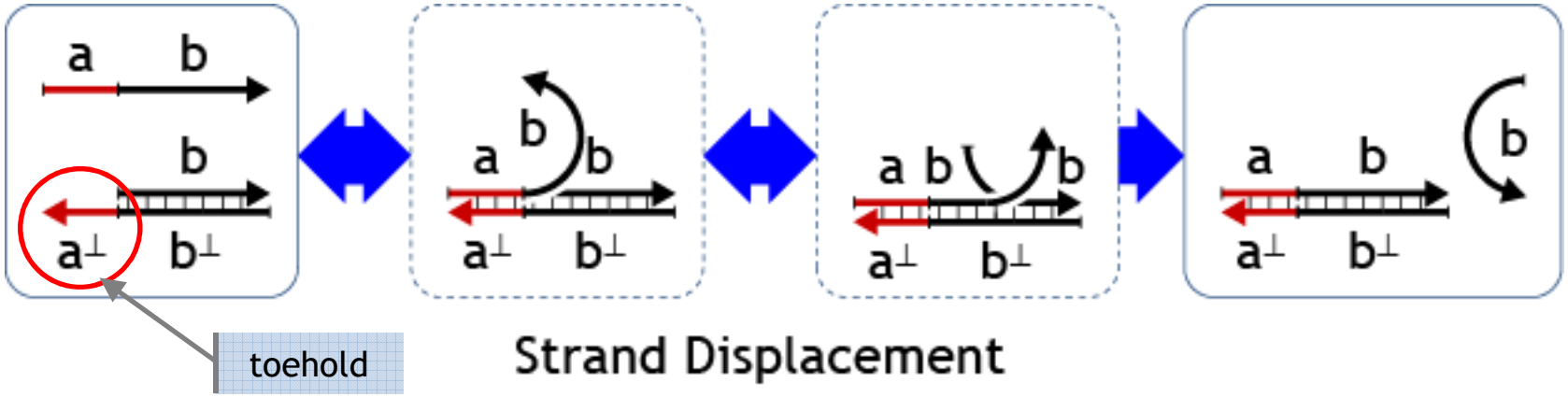
Long (black) segments



Irreversible Binding

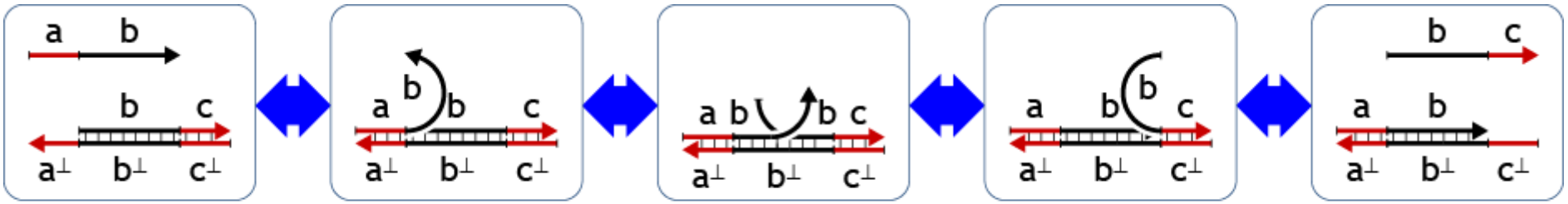
Gate Elements: Basic Mechanisms

Irreversible



Strand Displacement

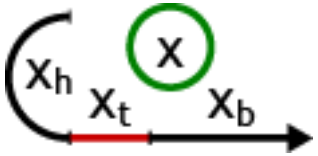
Reversible



Toehold Exchange

Gate Elements: Signals and Gates

- Signals "x" are single-stranded and 'positive'

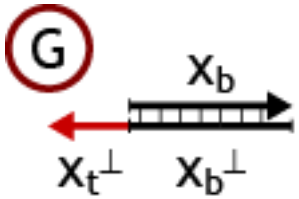


x_h = history
x_t = toehold
x_b = binding

x_t,x_b = signal identity for x

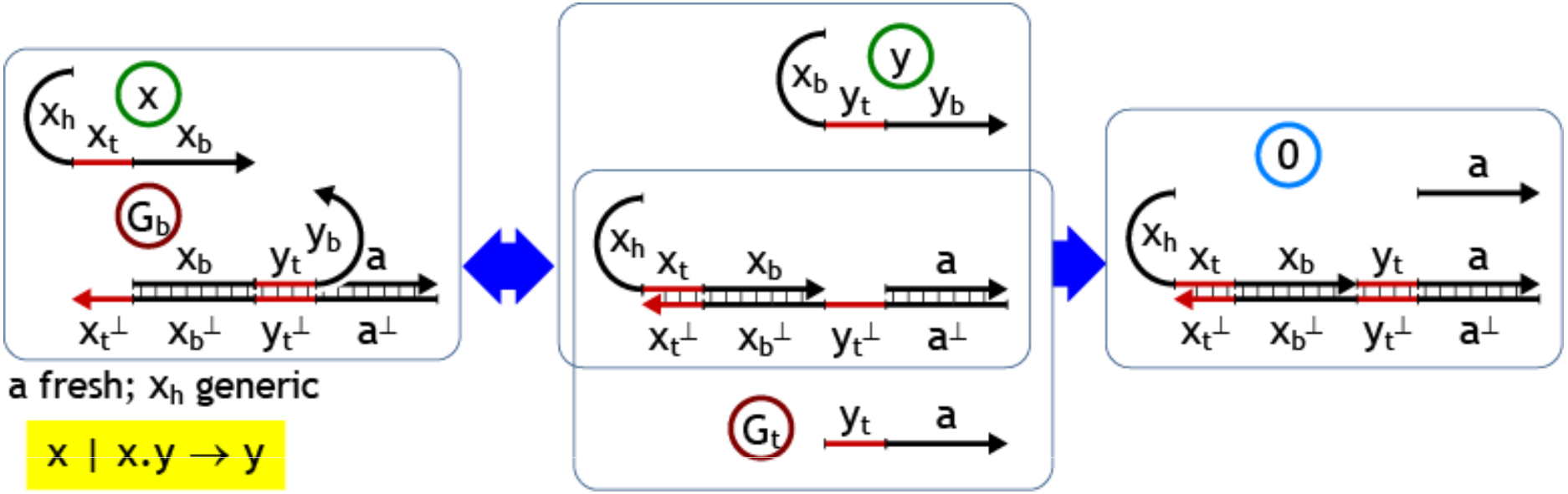
- This 3-segment signal representation is original to this work, it is based on the 4-segment signals of D. Soloveichik, G. Seelig, E. Winfree. Proc. DNA14, but leads to simpler and more regular gate structures

- Gate backbones are double-stranded, except for 'negative' toeholds.



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

Circuit Elements: x.y Transducer Gate



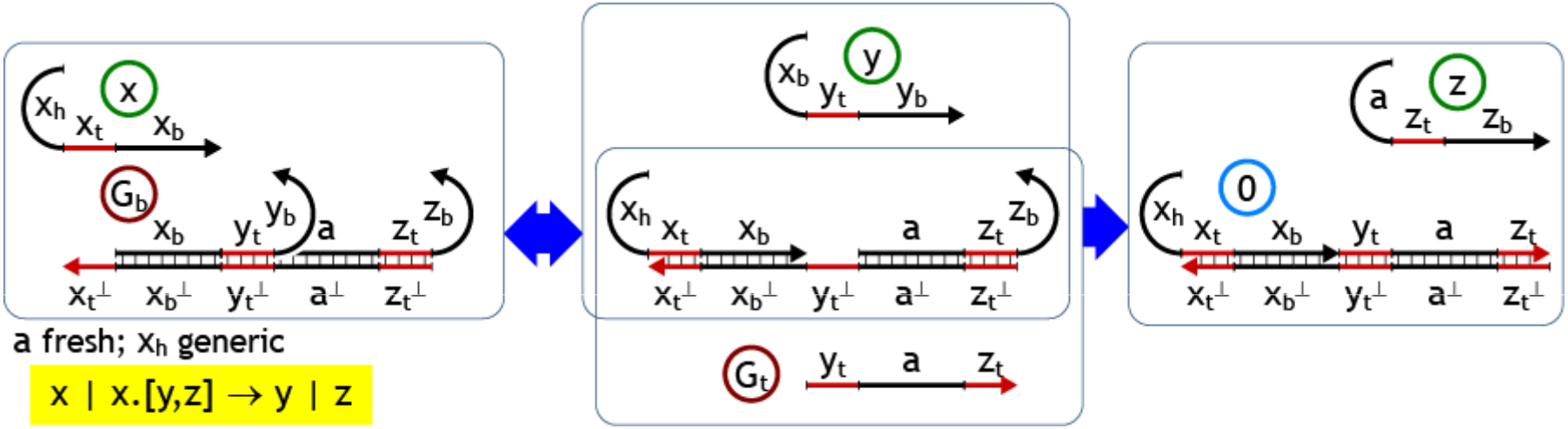
G_b,G_t (gate backbone and trigger) form the transducer.

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.

Circuit Elements: x.[y,z] Fork Gate

- A Fork signal-processing gate takes a signal x and produces two signals y,z according to the reaction x | x.[y,z] → y | z



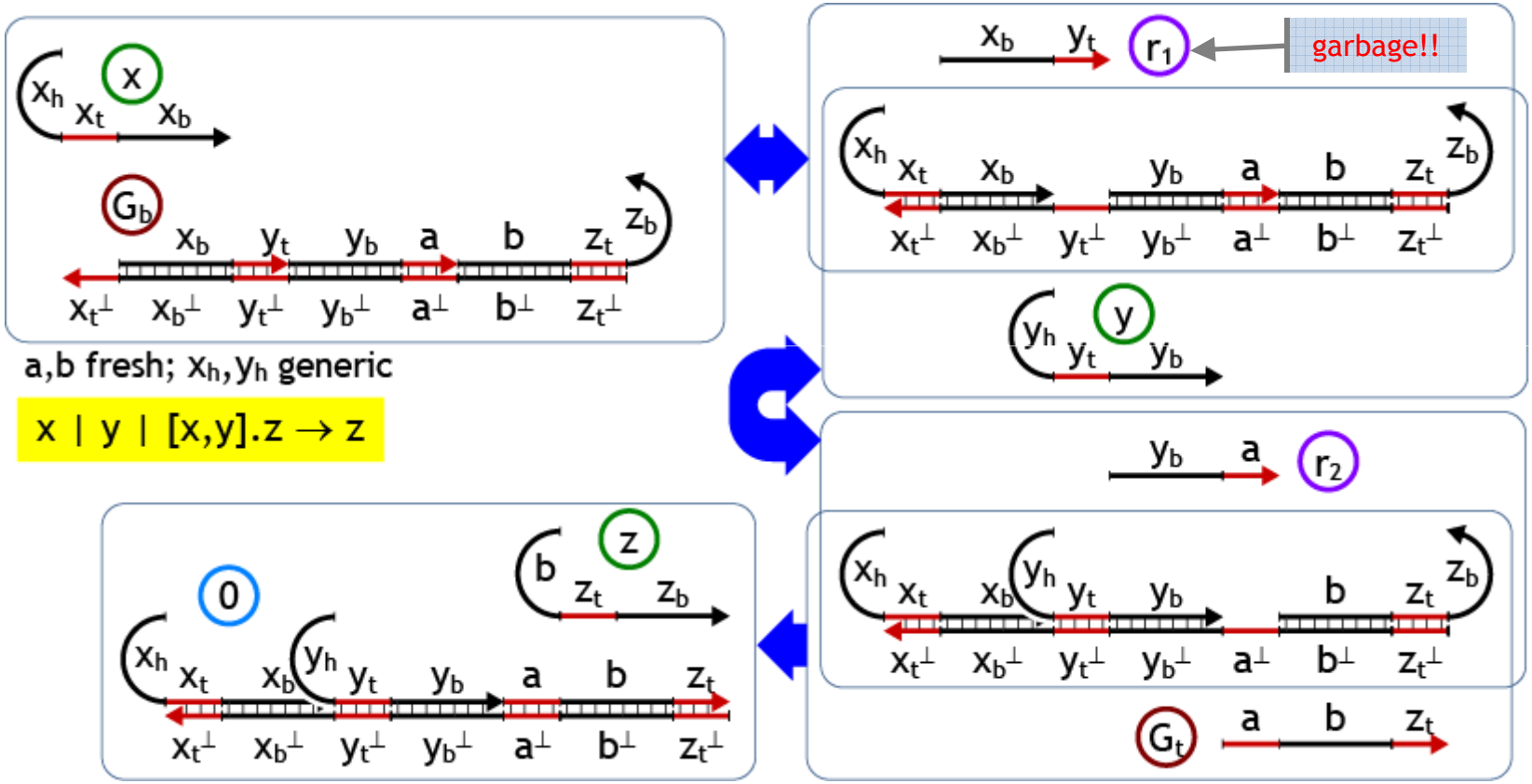
G_b,G_t (gate backbone and trigger) form the gate.

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.

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

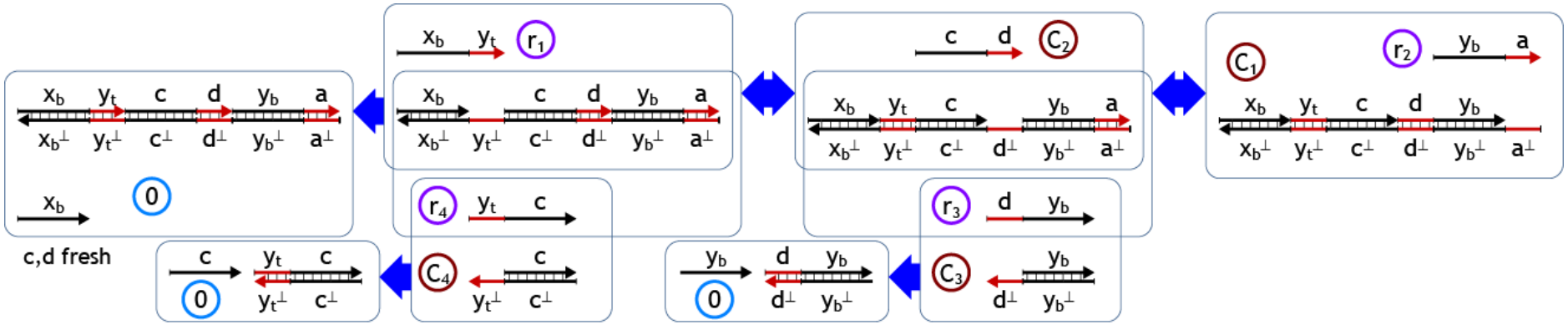
- A Join signal-processing gate takes *both* signals x,y and produces a signal z according to the reaction $x \mid y \mid [x,y].z \rightarrow z$



The garbage r_1 and r_2 must be collected (*after* the gate has fired) to avoid accumulation. This can be achieved by a similar scheme taking r_1,r_2 as input signals.

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

Garbage Collection

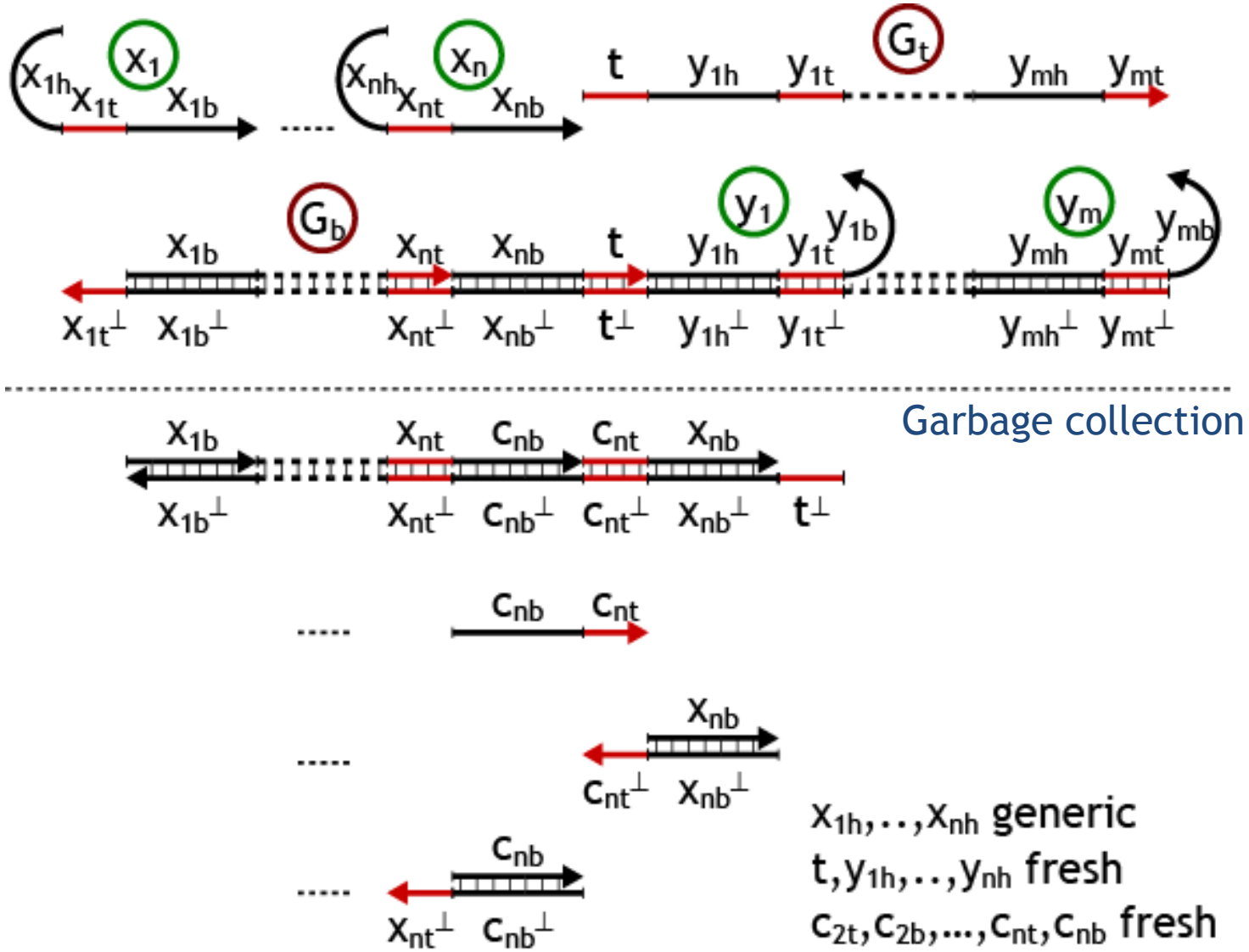


Garbage collection of r_1 is needed for join to work well. This is done by another reversible-AND between r_1 and r_2, triggered by the release of r_2. This second reversible-AND leaves garbage too (r_3, r_4), but this can be collected immediately, as we know by construction that both inputs r_1,r_2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

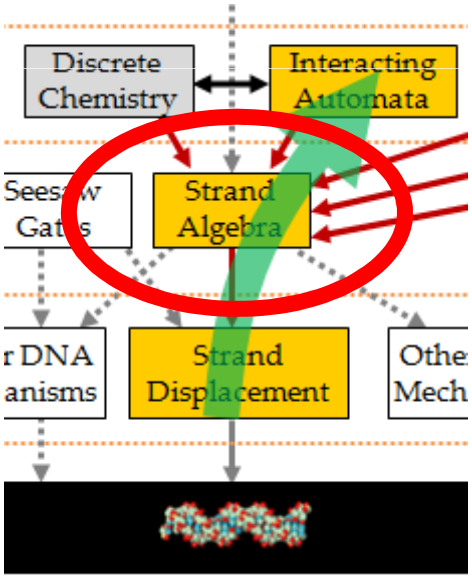
$[x_1,..,x_n].[y_1,..,y_m]$ General Join/Fork Gate

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



$x_{1h},..,x_{nh}$ generic
$t, y_{1h},..,y_{nh}$ fresh
$c_{2t}, c_{2b},...,c_{nt}, c_{nb}$ fresh

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)

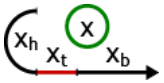
$$P \rightarrow P' \quad \Rightarrow \quad P \mid P'' \rightarrow P' \mid P'' \qquad \text{Dilution}$$

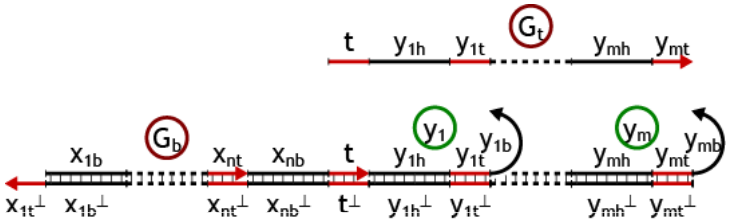
$$P \equiv P_1,\ P_1 \rightarrow P_2,\ P_2 \equiv P' \quad \Rightarrow \quad P \rightarrow P' \qquad \text{Well Mixing}$$

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

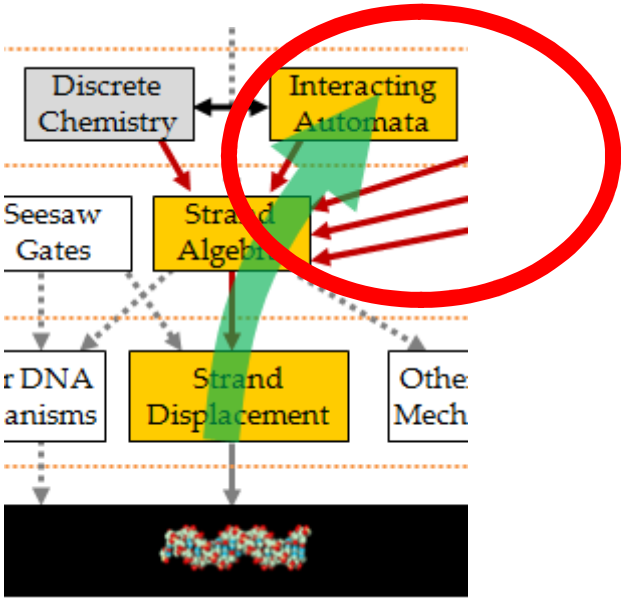
- compile(0) = empty solution
- compile(P | P') = mix(compile(P), compile(P'))
- compile(P*) = population(compile(P))

More in the Paper

- Stochastic strand algebra
  - Matches the stochastic semantics of interacting automata
  - Uses a technique for implementing constant buffered populations, to replace P* with finite populations

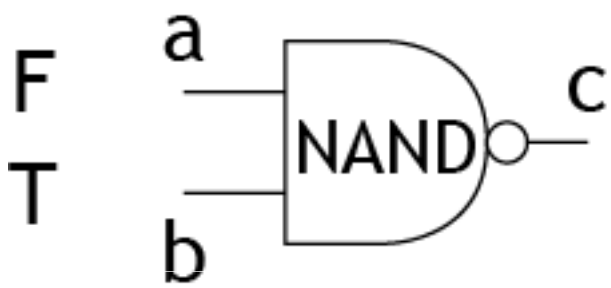
- Nested strand algebra
  - An higher-level language (with nested expressions)
  - A compilation algorithm into the basic strand algebra

Computational Abstractions ("Low-Level" Languages)



Boolean Networks

Boolean Networks to Strand Algebra



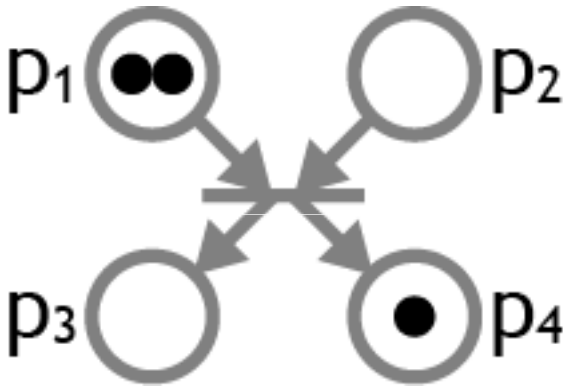
$$
\begin{array}{l}
([a_F,b_F].c_T)^* \mid \\
([a_F,b_T].c_T)^* \mid \\
([a_T,b_F].c_T)^* \mid \\
([a_T,b_T].c_F)^* \mid \\
a_F \mid b_T
\end{array}
$$

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

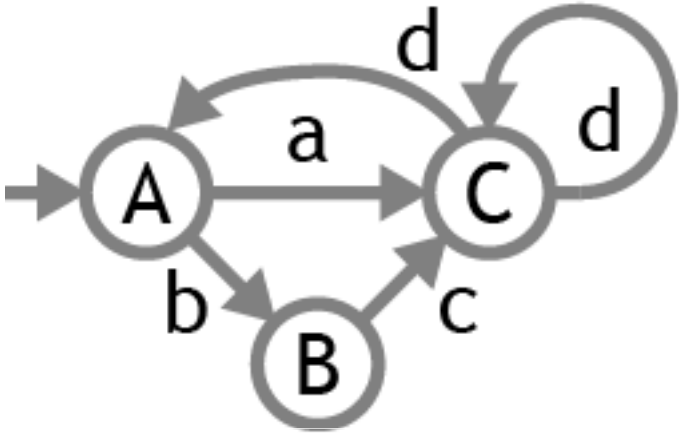


$$([p_1,p_2].[p_3,p_4])^* \mid p_1 \mid p_1 \mid p_4$$

Finite State Automata

Assuming ONE automaton and ONE input string.

FSA to Strand Algebra



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

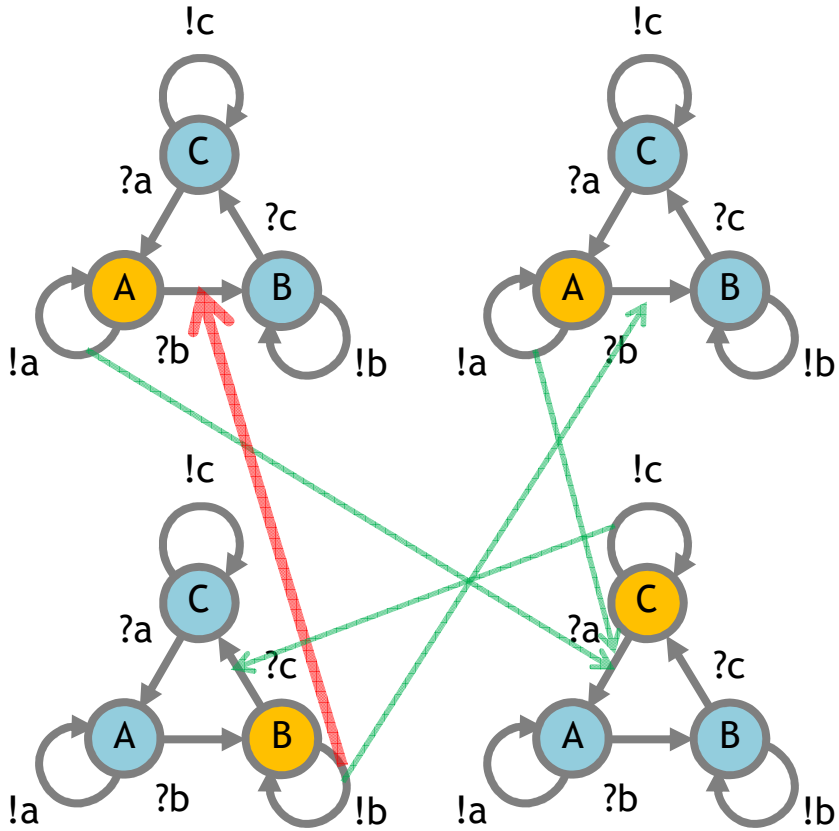
Input strings

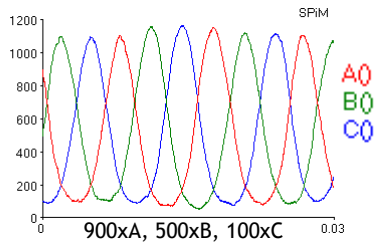
a,b,c,d

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

Automata *populations* are a more natural model...

Interacting Automata

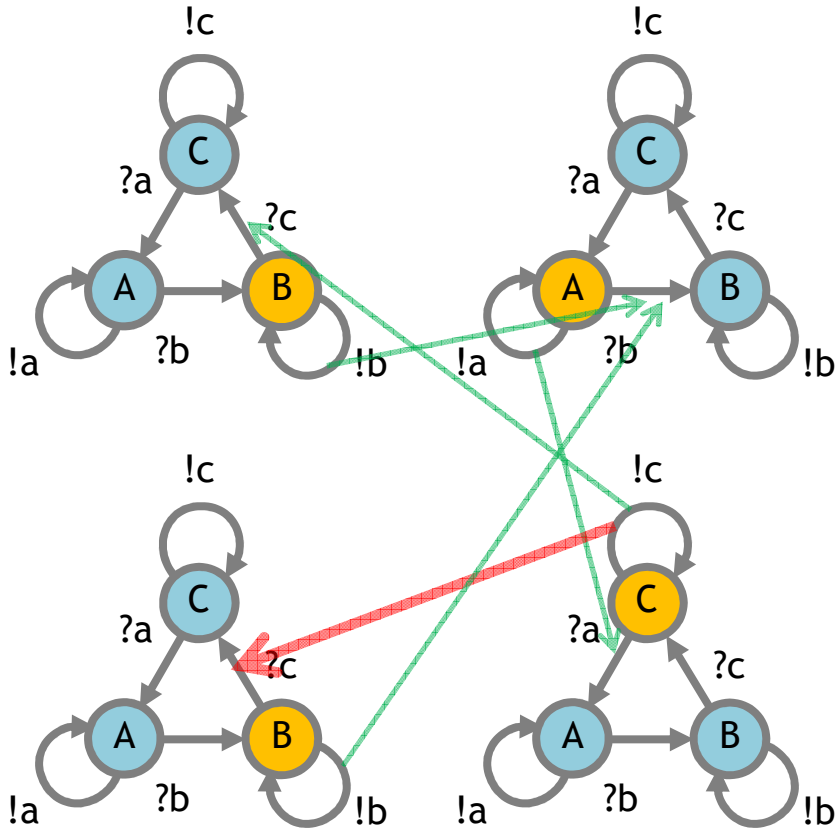


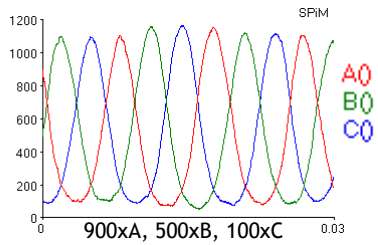


([A,B].[B,B])* |
([B,C].[C,C])* |
([C,A].[A,A])* |
A | A | B | C

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

Interacting Automata

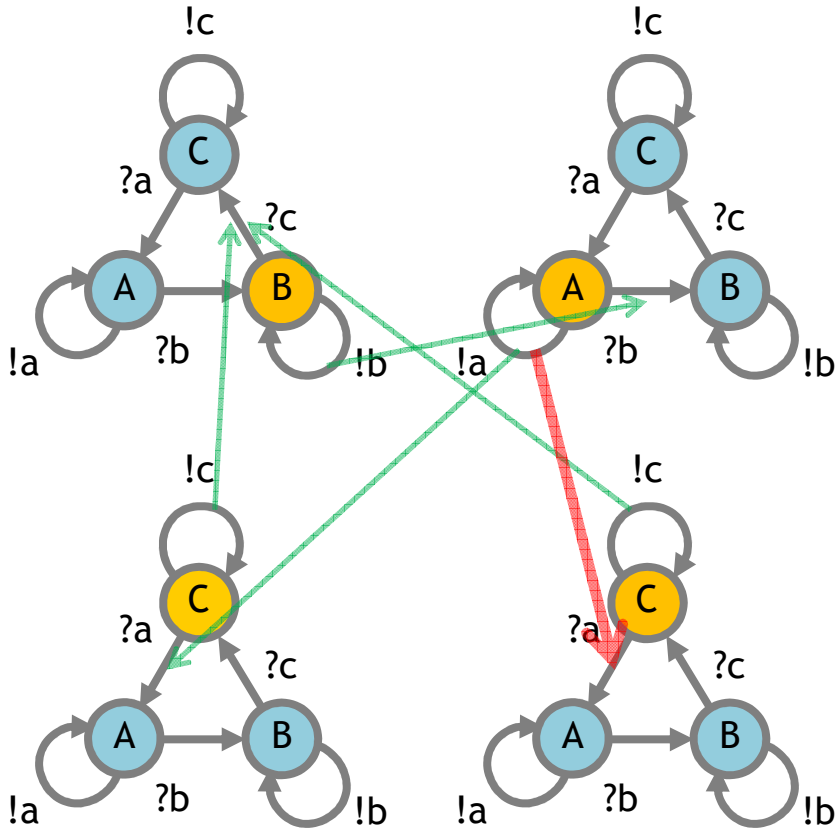


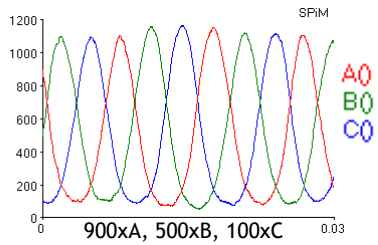


([A,B].[B,B])* |
([B,C].[C,C])* |
([C,A].[A,A])* |
A | B | B | C

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

Interacting Automata

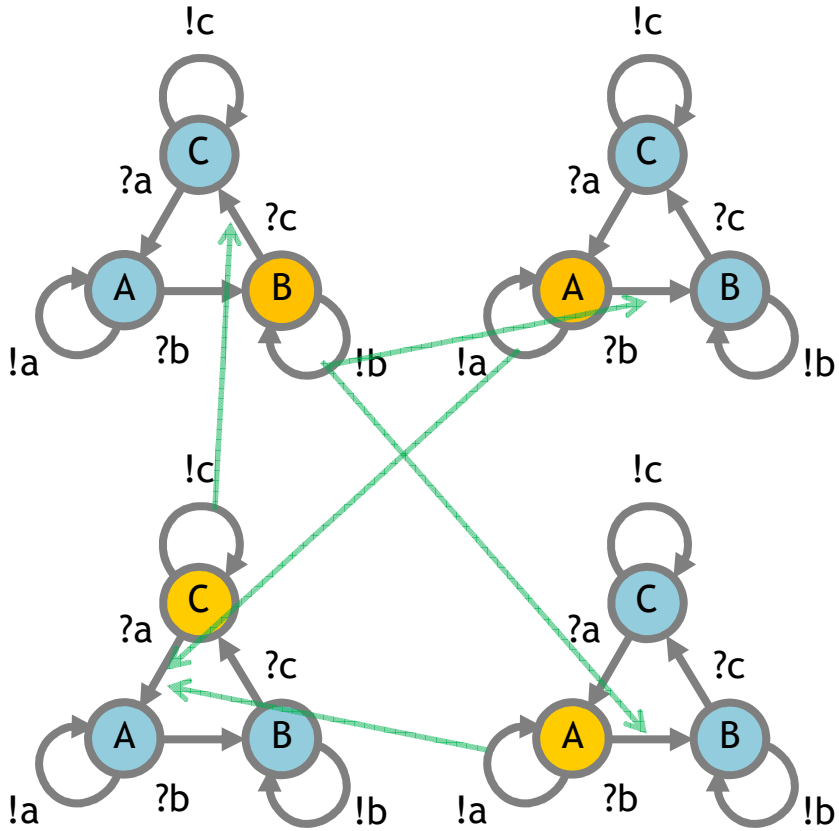


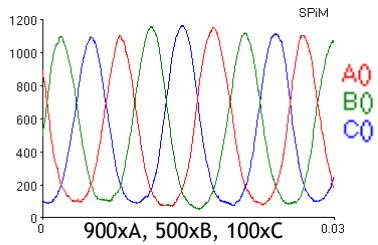


([A,B].[B,B])* |
([B,C].[C,C])* |
([C,A].[A,A])* |
A | B | C | C

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

Interacting Automata

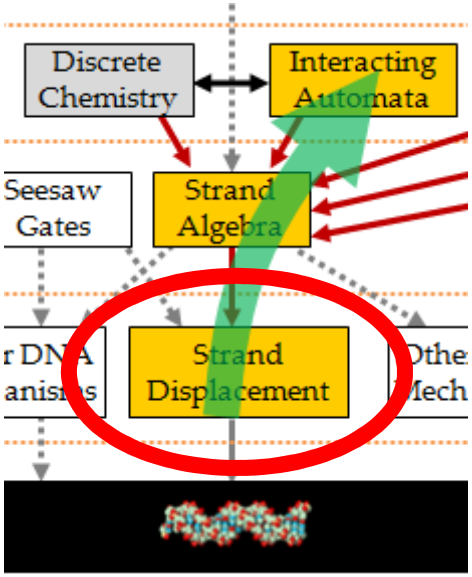




([A,B].[B,B])* |
([B,C].[C,C])* |
([C,A].[A,A])* |
A | A | B | C

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

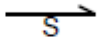
Strand Displacement Intermediate Language



Syntax

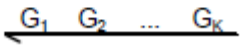
A. Syntax of DNA molecules *D*

Upper strand with sequence complementary to S



`<S>`

Molecule with segments $G_1,\ldots,G_K$



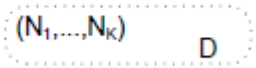
`G1:G2:...:GK`

Parallel molecules $D_1,\ldots,D_K$

D_1 D_2 ... D_K

`D1 | D2 | ... | DK`

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



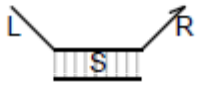
`new (N1,...,NK) D`

B. Syntax of DNA segments *G*

Lower strand with toehold N^c

`N^c`

Double strand with sequence S and overhangs L, R



`<L>[S]<R>`

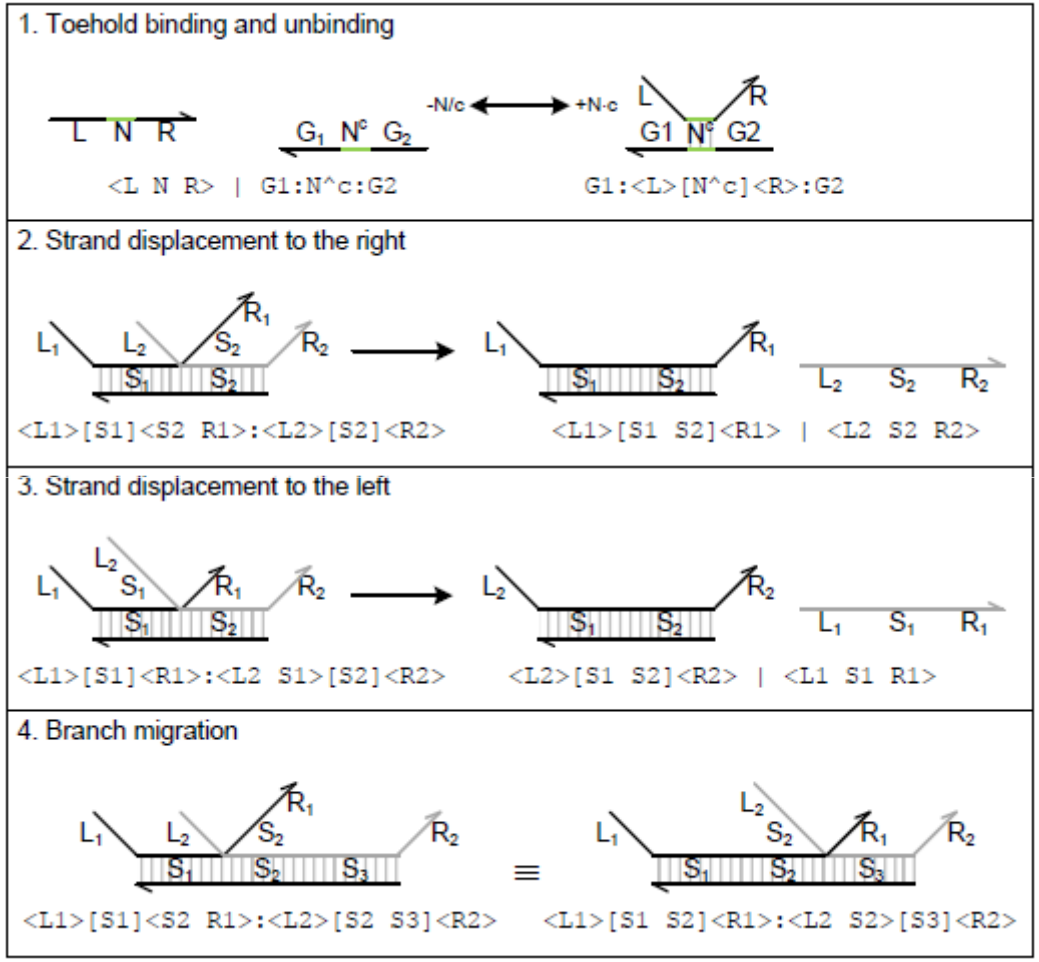
C. Syntax of DNA sequences *S,L,R*

Sequence of domains $O_1,\ldots,O_K$

O_1 O_2 ... O_K

`O1 O2 ... OK`

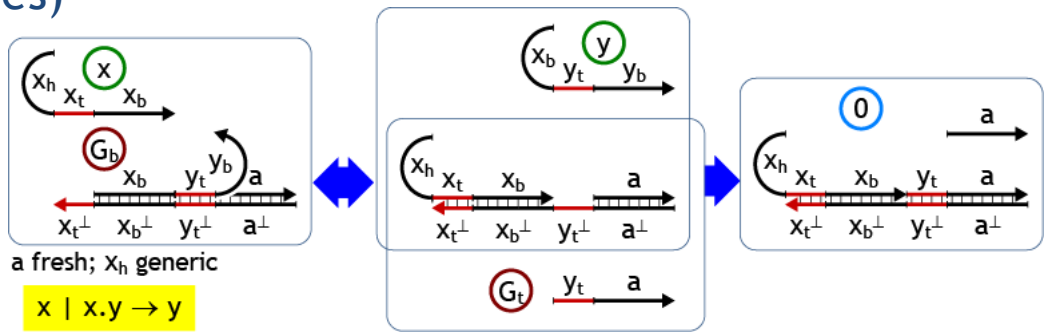
Dynamics

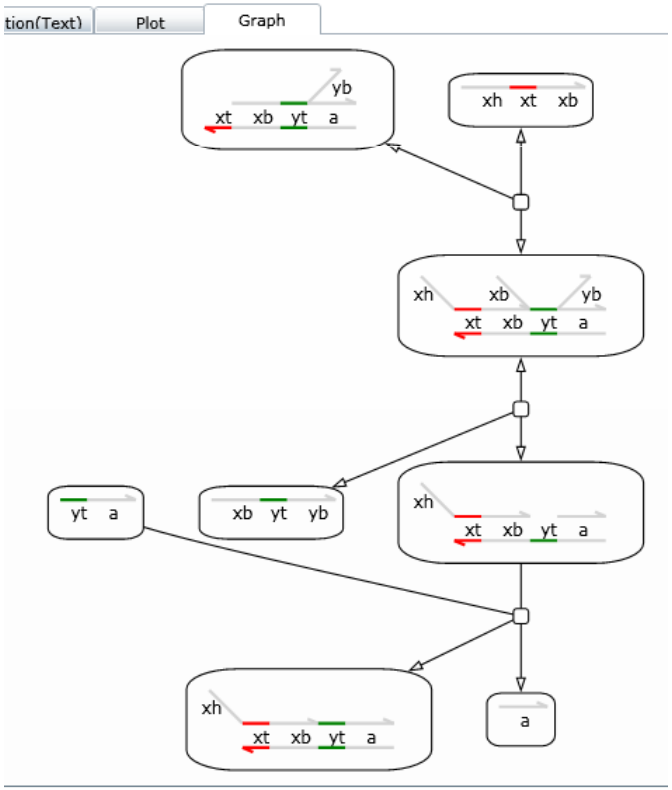

1. Toehold binding and unbinding
L N R
G1 N^c G2
-N/c
+N-c
L R
G1 N^c G2
<L N R> | G1:N^c:G2
G1:<L>[N^c]<R>:G2
2. Strand displacement to the right
L1 L2 S1 S2 R1 R2
L1 S1 S2 R1
L2 S2 R2
<L1>[S1]<S2 R1>:<L2>[S2]<R2>
<L1>[S1 S2]<R1> | <L2 S2 R2>
3. Strand displacement to the left
L1 L2 S1 S2 R1 R2
L2 S1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

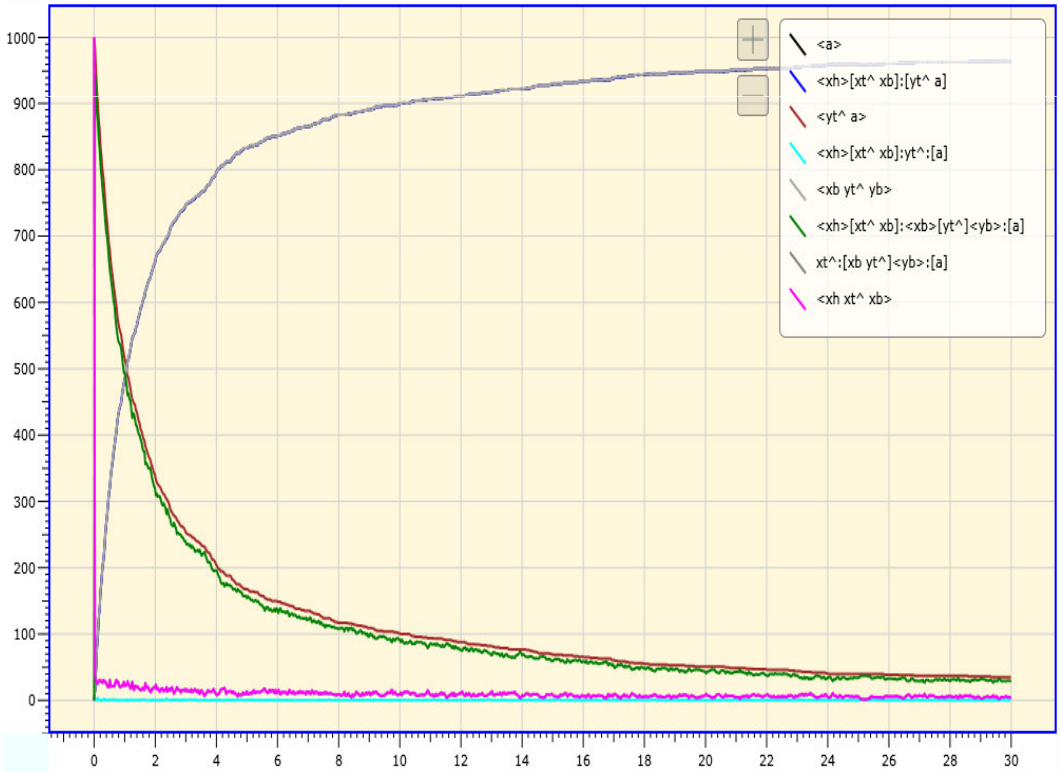
Strand Displacement Simulation Tool

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>
)
```



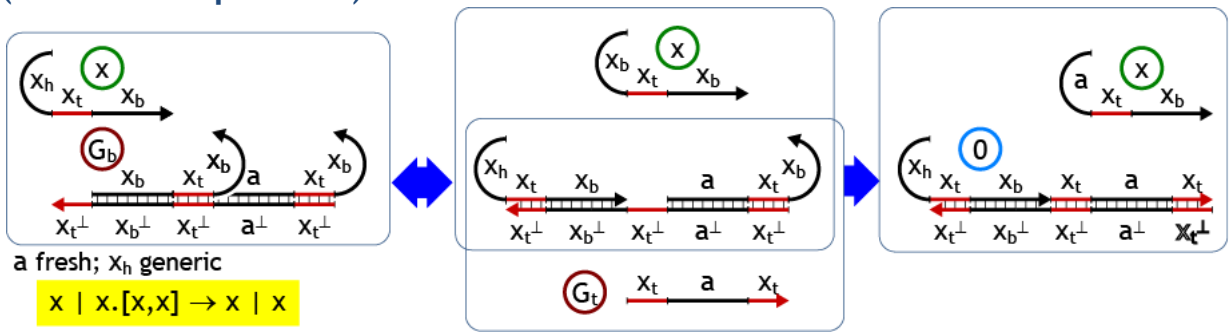




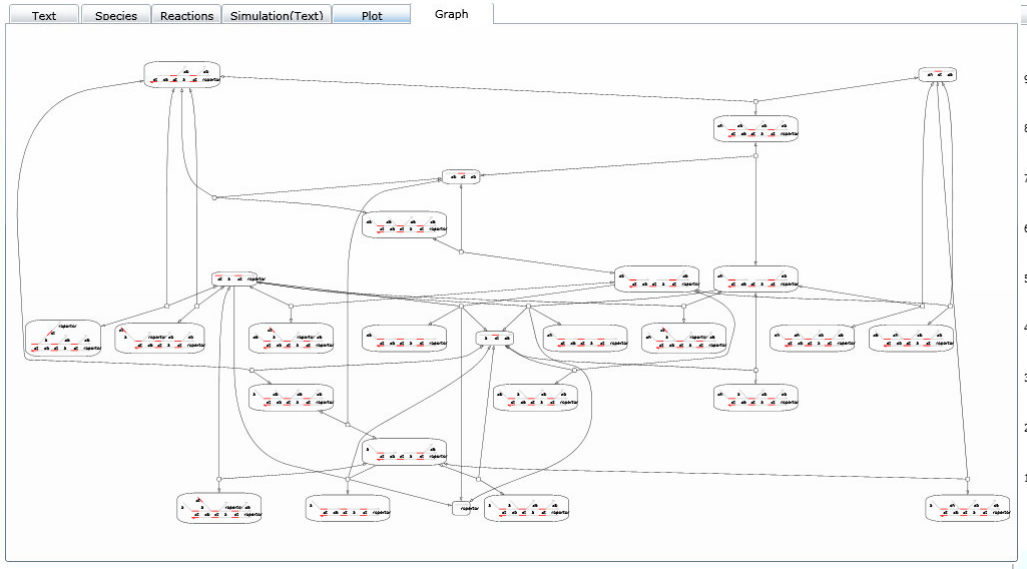
Strand Displacement Simulation Tool

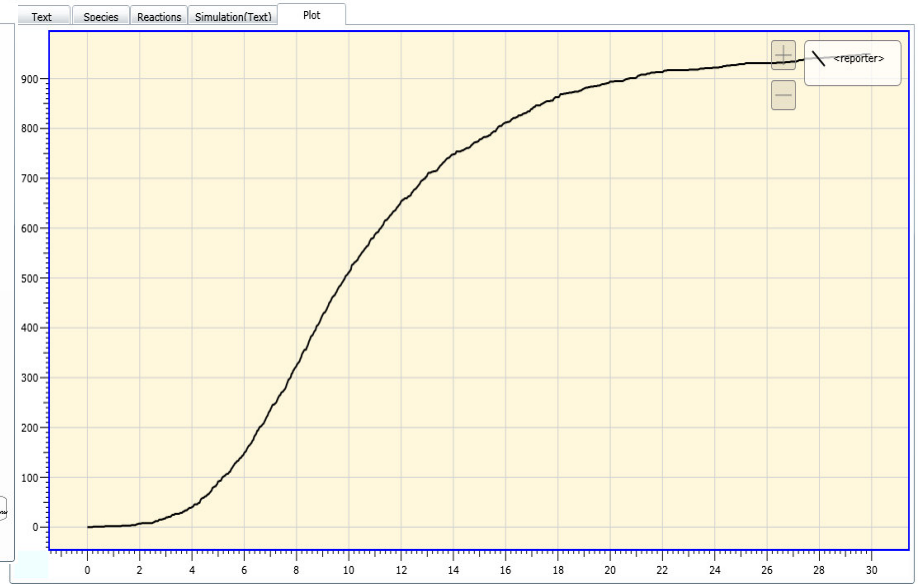
Fork Chain Reaction x.[x,x] (3 initial species)

```
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>
)
```



26 Species, 20 Reactions



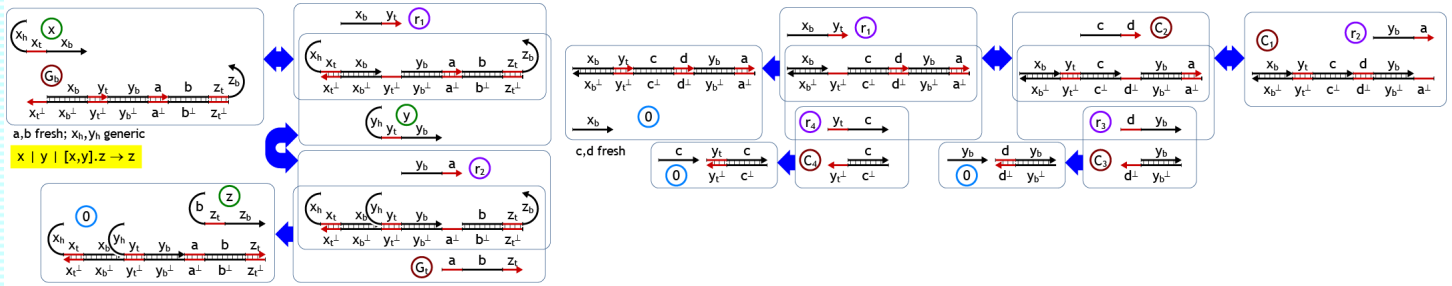


Strand Displacement Simulation Tool

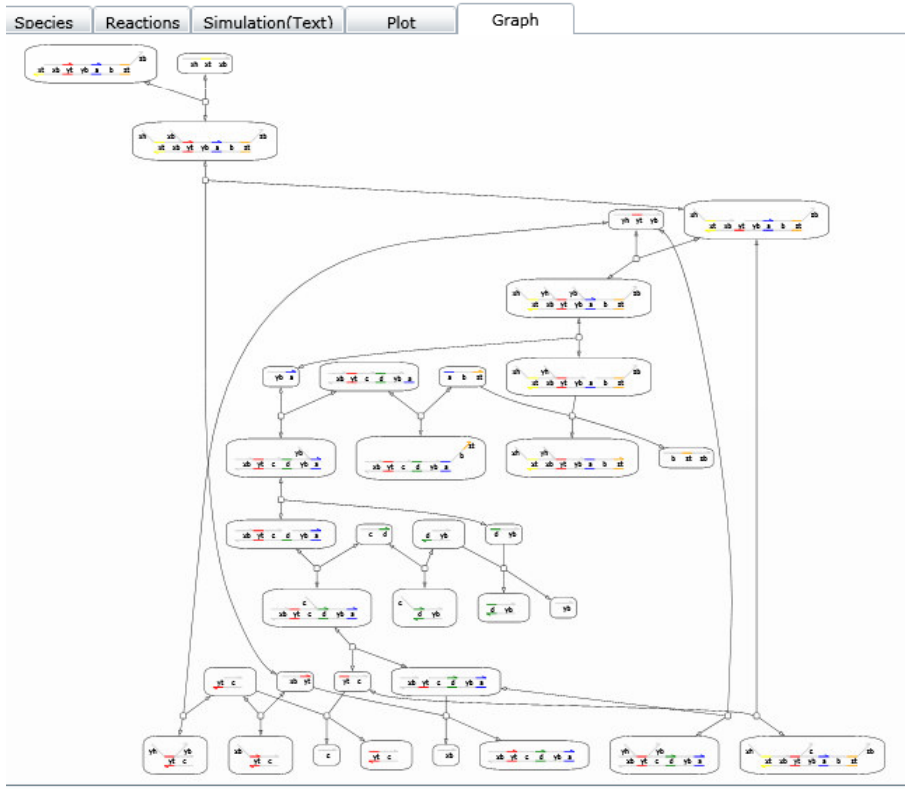
1 Join gate with garbage collection [x,y].z (8 initial species)

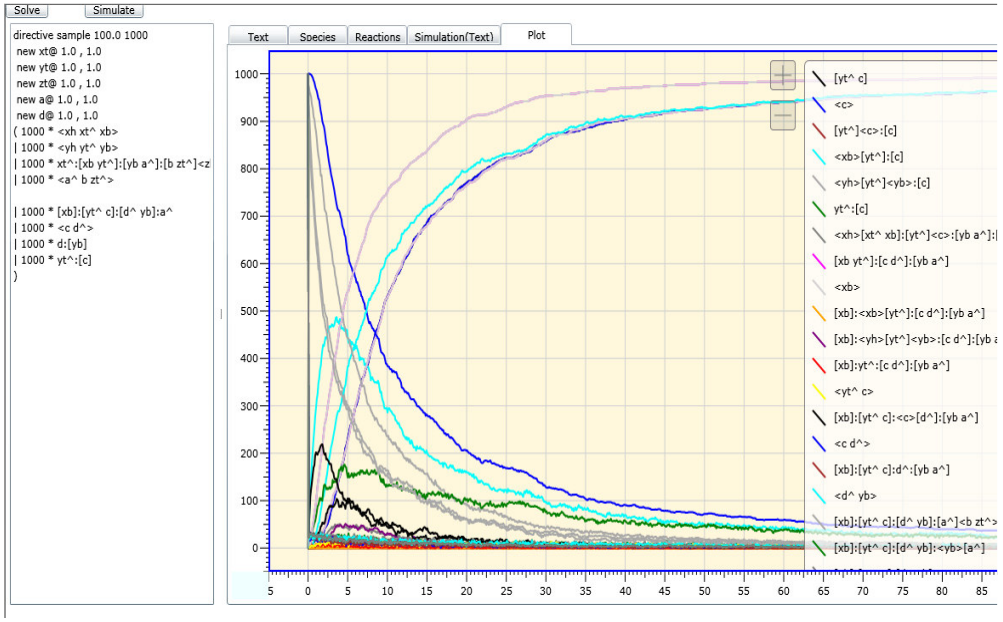
```
directive sample 1000.0 1000
 new xt@ 1.0 , 1.0
 new yt@ 1.0 , 1.0
 new zt@ 1.0 , 1.0
 new a@ 1.0 , 1.0
 new d@ 1.0 , 1.0
( 1000 * <xh xt^ xb>
| 1000 * <yh yt^ yb>
| 1000 * xt^:[xb yt^]:[yb a^]:[b zt^]<zb>
| 1000 * <a^ b zt^>

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

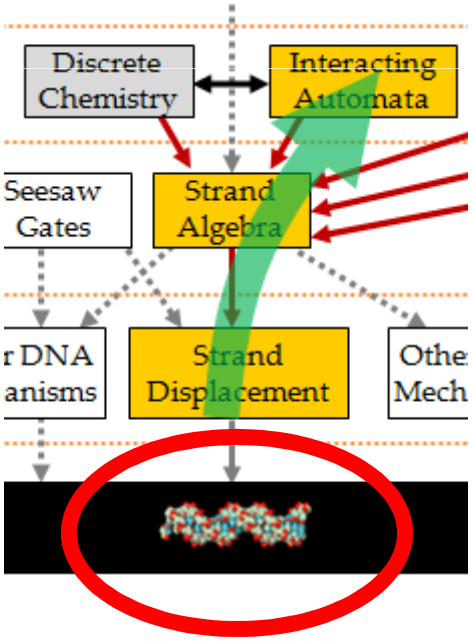


34 Species, 18 Reactions





Sequence Design



Sequence Design

NUPACK BETA
nucleic acid package

Analysis | Design | Downloads

Input | References | Demos | Help

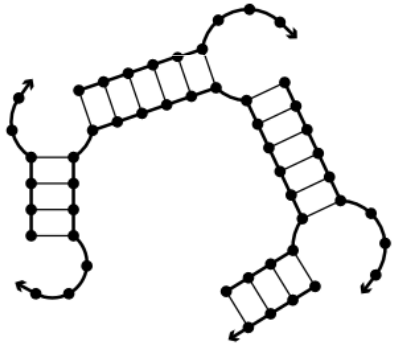
Nucleic acid type: RNA DNA

Number of designs: 1

Target structure: ((((..+((((((...+((((((...+((((+))))))))))))))))))...

Input

Preview:

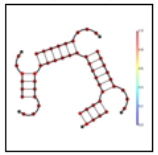


NUPACK BETA
nucleic acid package

Analysis | Design | Downloads

Input | Results | References | Demos | Help

Designability summary



Sequence designs

Average percentage of correct nucleotides	Average number of incorrect nucleotides	GC content	Sequence	
99.1%	0.475	74.5%	GGCCUC+GCAAGCACC+GCC AGCUUG+GCUC+GAGCGCUG GCGCUUGCGGCCGUG	Analyze

Output

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 (Strand Algebra, Strand Displacement Language).
  - Sequence generation.
- Tools
  - Thermodynamic analysis.
  - Simulation.
  - Verification (not yet).