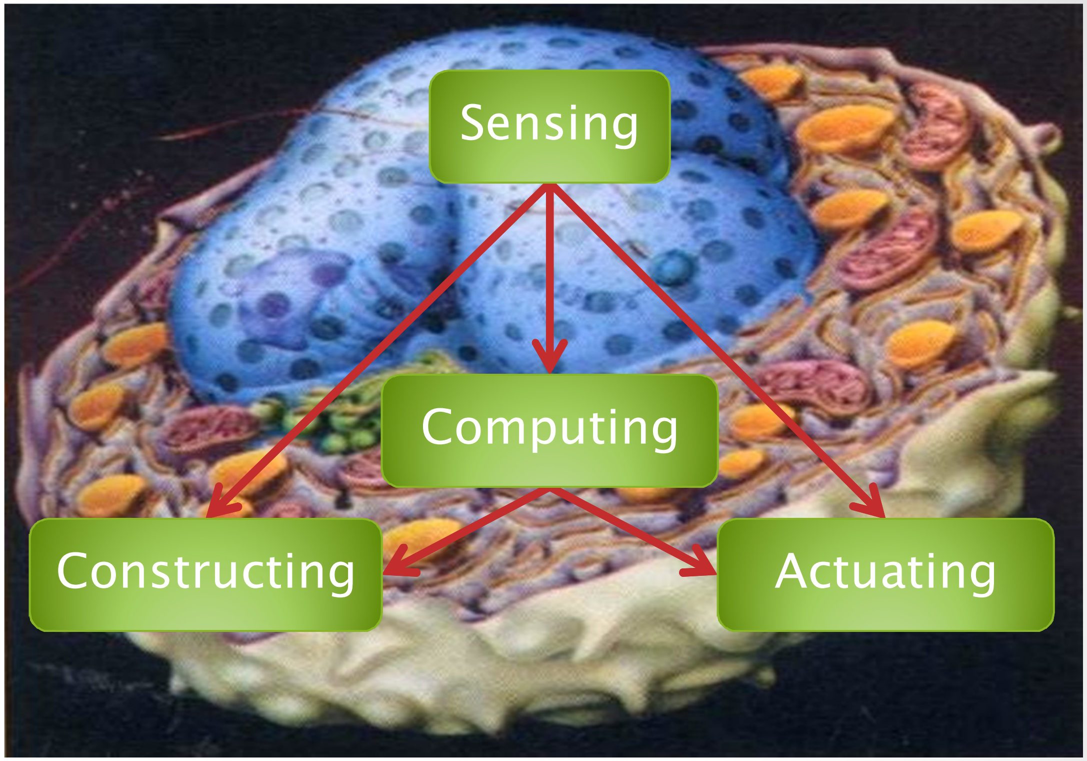
Algebras and Languages for Molecular Programming

Luca Cardelli
Microsoft Research

MergingKnowledge, Trento, 2010-12-01
http://lucacardelli.name

Nanoscale Engineering

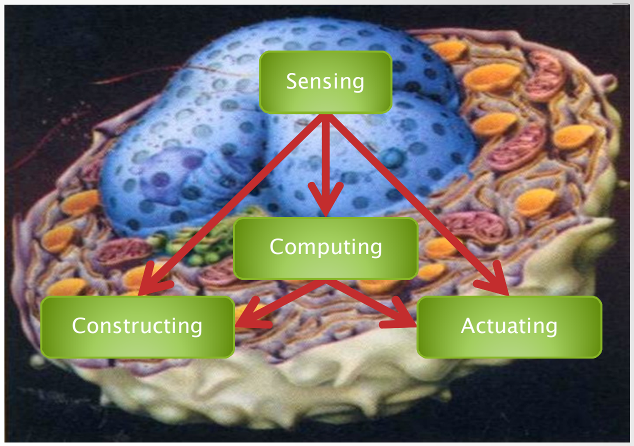
- Sensing
  - Reacting to forces
  - Binding to molecules
- Actuating
  - Releasing molecules
  - Producing forces
- Constructing
  - Chassis
  - Growth
- Computing
  - Signal Processing
  - Decision Making



Nucleic Acids can do all this.

And interface to biology.
And are programmable.

Curing



A doctor in each cell

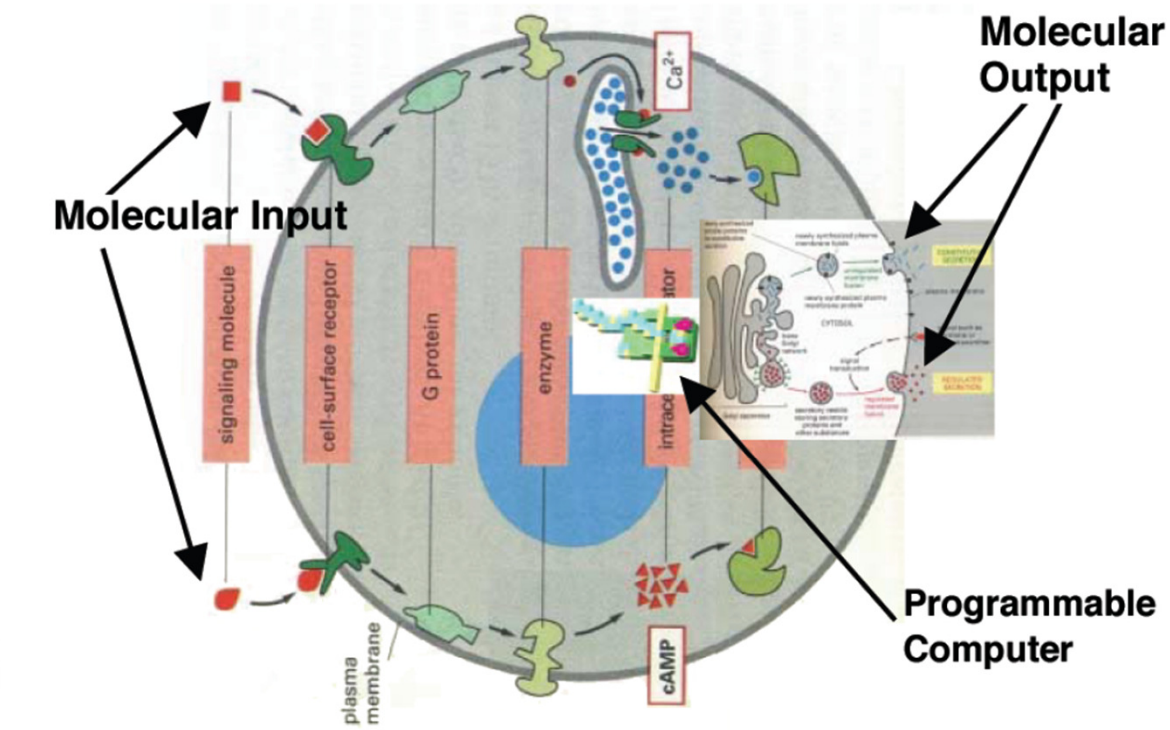


Fig. 1 Medicine in 2050: "Doctor in a Cell"

Ehud Shapiro
Rivka Adar
Kobi Benenson
Gregory Linshitz
Aviv Regev
William Silverman

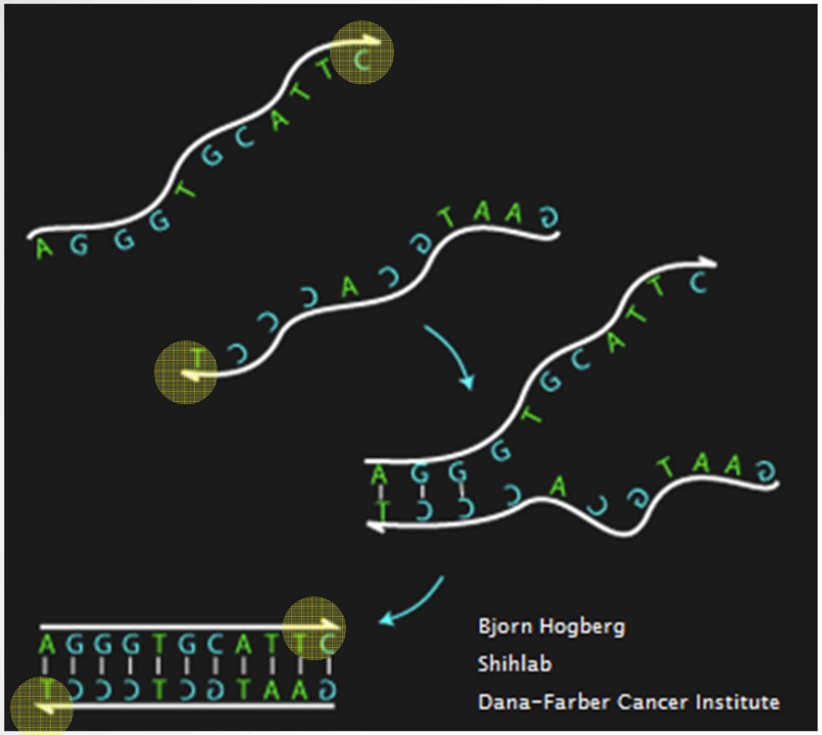
Molecules and computation

Execution?

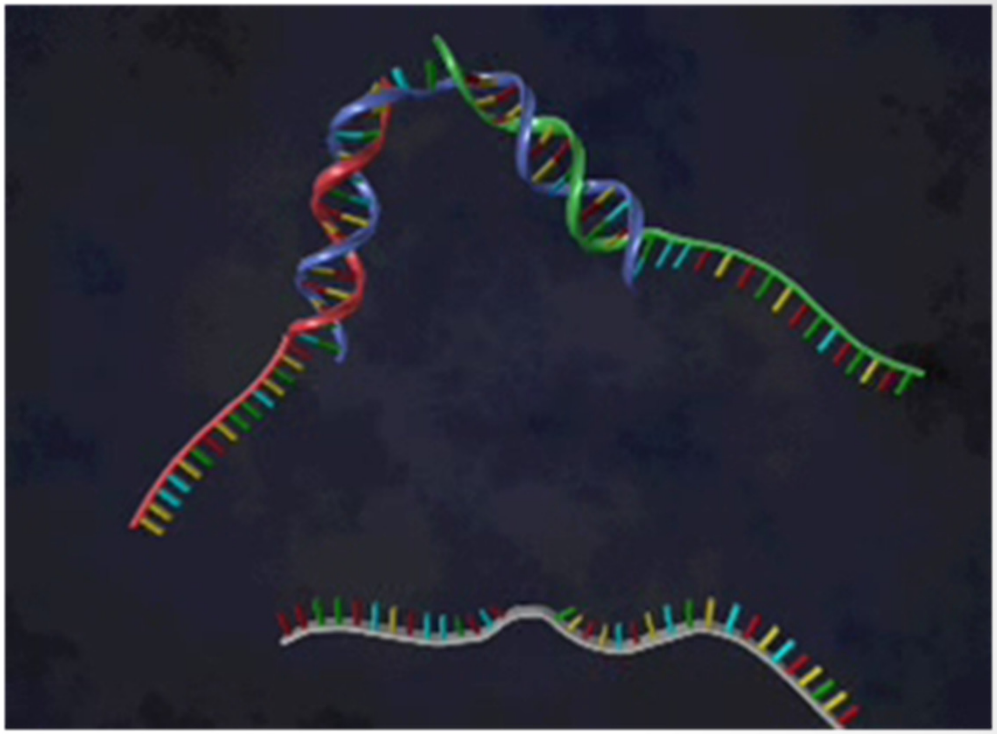
- Chemistry is not easily executable
  - Is chemistry a programming language?
  - Please Mr Chemist, execute me these reactions I just made up!
- Proteins are not easily programmable
- Most molecular-scale notations are descriptive (modeling) languages
- How can we actually execute molecular languages? With real molecules?

Strand Displacement Basics

DNA Hybridization



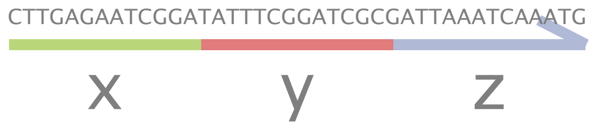
Bjorn Hogberg
Shihlab
Dana-Farber Cancer Institute



- Strands with **opposite orientation** and **complementary base pairs** stick to each other (Watson-Crick duality).
- This is all we are going to use
  - We are not going to exploit DNA replication, transcription, translation, **restriction and ligation enzymes**, etc., which enable other classes of tricks.

Domains

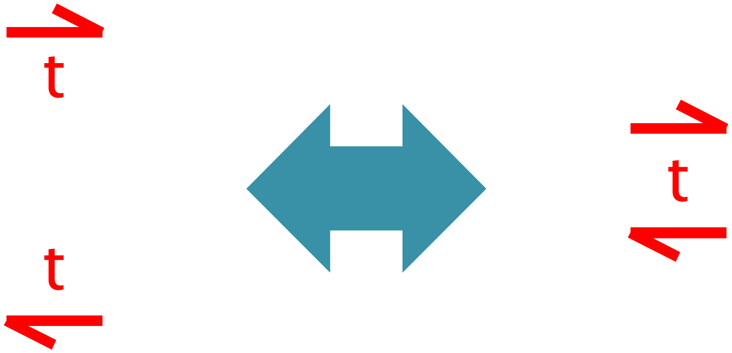
- Subsequences on a DNA strand are called domains.
- PROVIDED they are “independent” of each other.



- I.e., differently named domains must not hybridize:
  - With each other
  - With each other’s complement
  - With subsequences of each other
  - With concatenations of other domains (or their complements)
  - Etc.

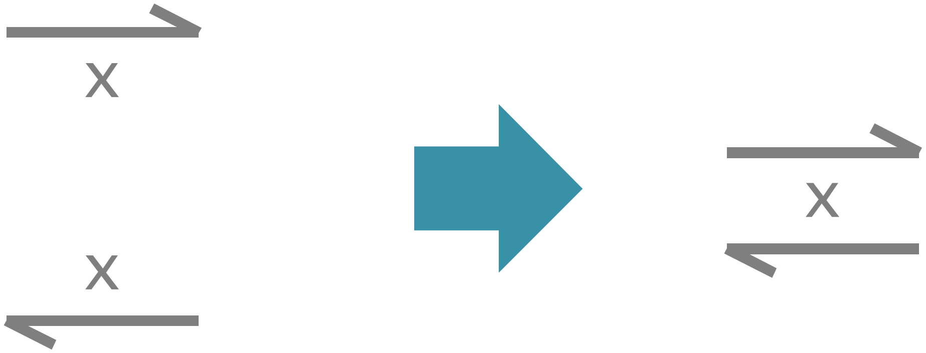
- Choosing domains (subsequences) that are suitably independent is a tricky issue that is still somewhat of an open problem (with a vast literature). But it can work in practice.

Short Domains



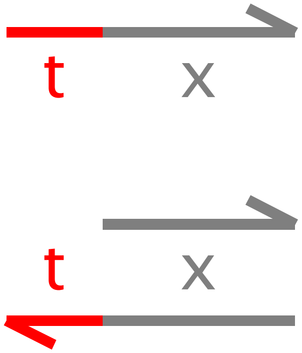
Reversible Hybridization

Long Domains



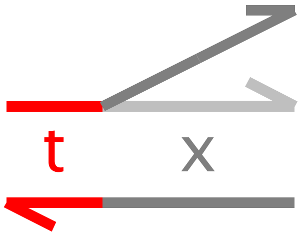
Irreversible Hybridization

Strand Displacement



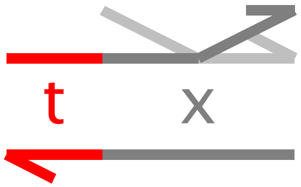
“Toehold Mediated”

Strand Displacement



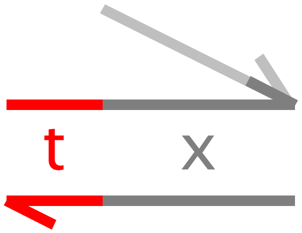
Toehold Binding

Strand Displacement



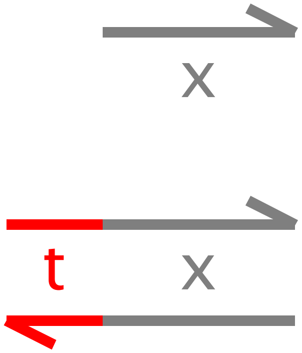
Branch Migration

Strand Displacement



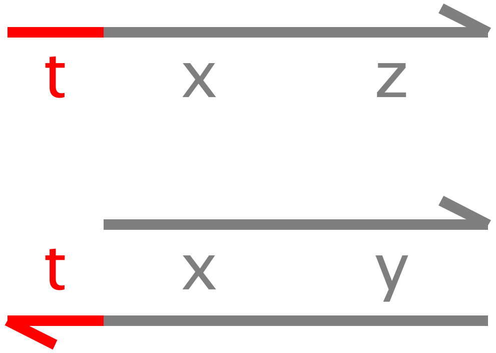
Displacement

Strand Displacement

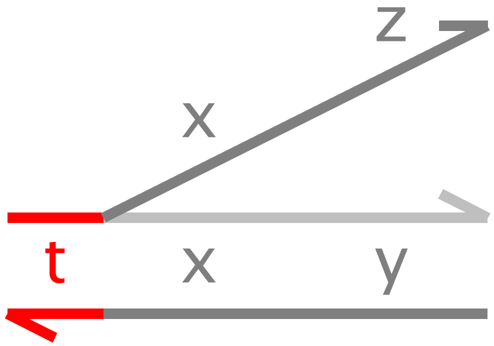


Irreversible release

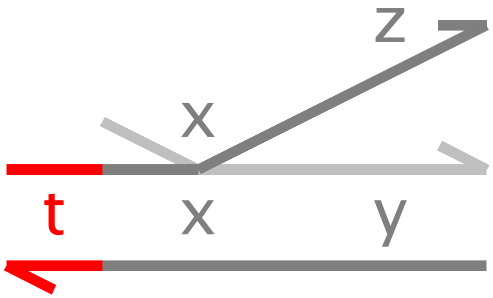
Bad Match



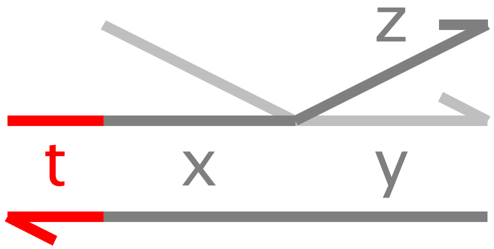
Bad Match



Bad Match



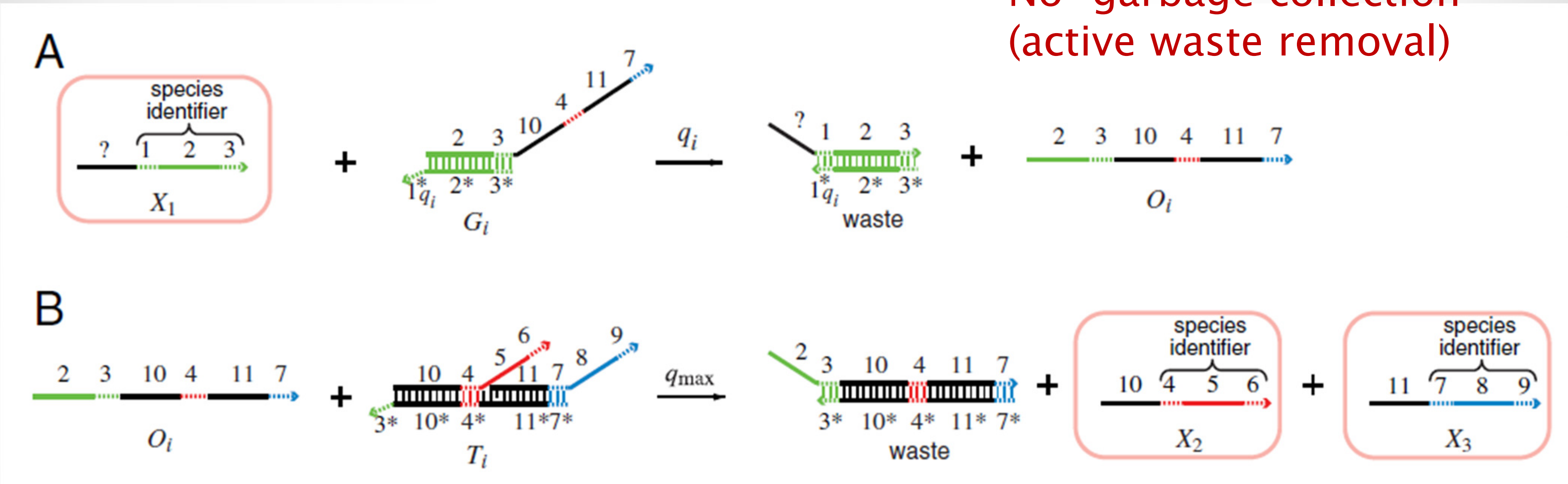
Bad Match



Cannot proceed
Hence will undo

Signals & Gates

Four–Domain Architecture



No "garbage collection"
(active waste removal)

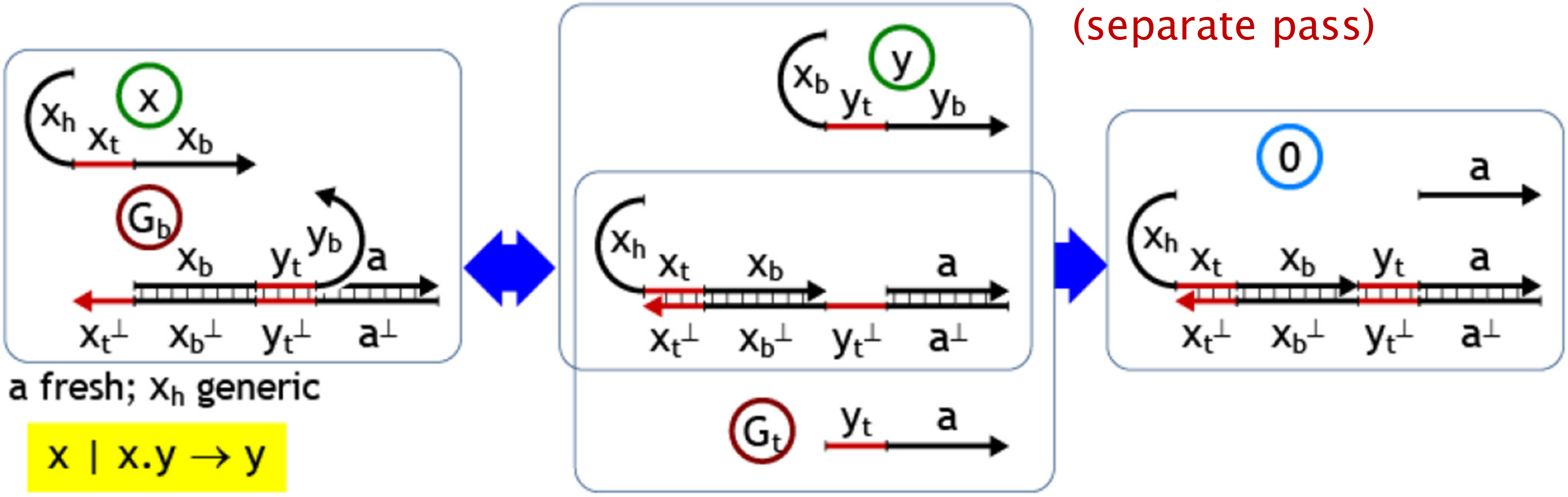
DNA as a universal substrate for chemical kinetics

David Soloveichik[a,1], Georg Seelig[a,b,1], and Erik Winfree[c,1]

PNAS | March 23, 2010 | vol. 107 | no. 12 | 5393–5398

Three-Domain Architecture

With garbage collection (separate pass)



a fresh; x_h generic

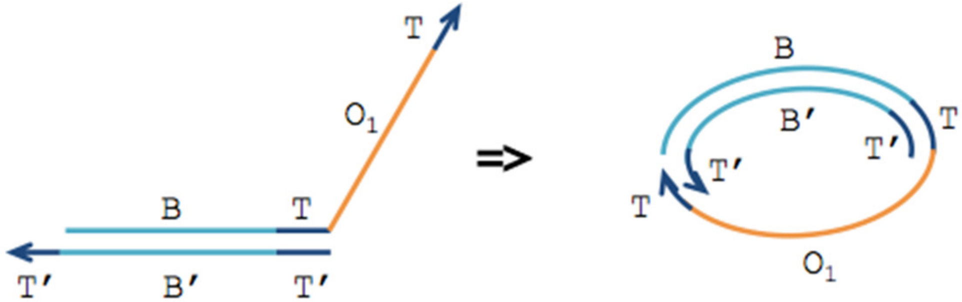
$x \mid x.y \rightarrow y$

Strand Algebras for DNA Computing

Luca Cardelli

DNA Computing and Molecular Programming.
15th International Conference, DNA 15, LNCS 5877, Springer 2009, pp 12-24.

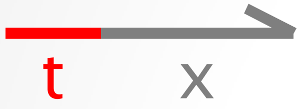
“Lulu’s Trouble”



(from D.Soloveichik)

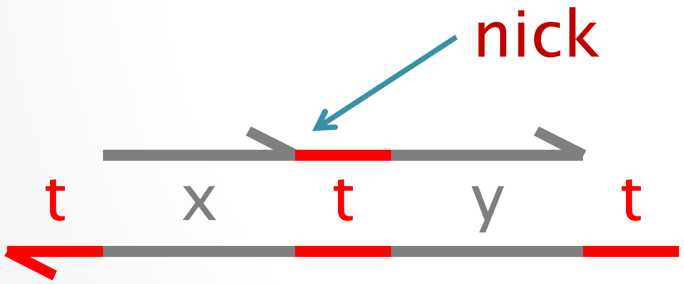
Two–Domain Architecture

- Signals: 1 toehold + 1 recognition region



Garbage collection
“built into” the gates

- Gates: “top–nicked double strands”
(or equivalently double strands with open toeholds)

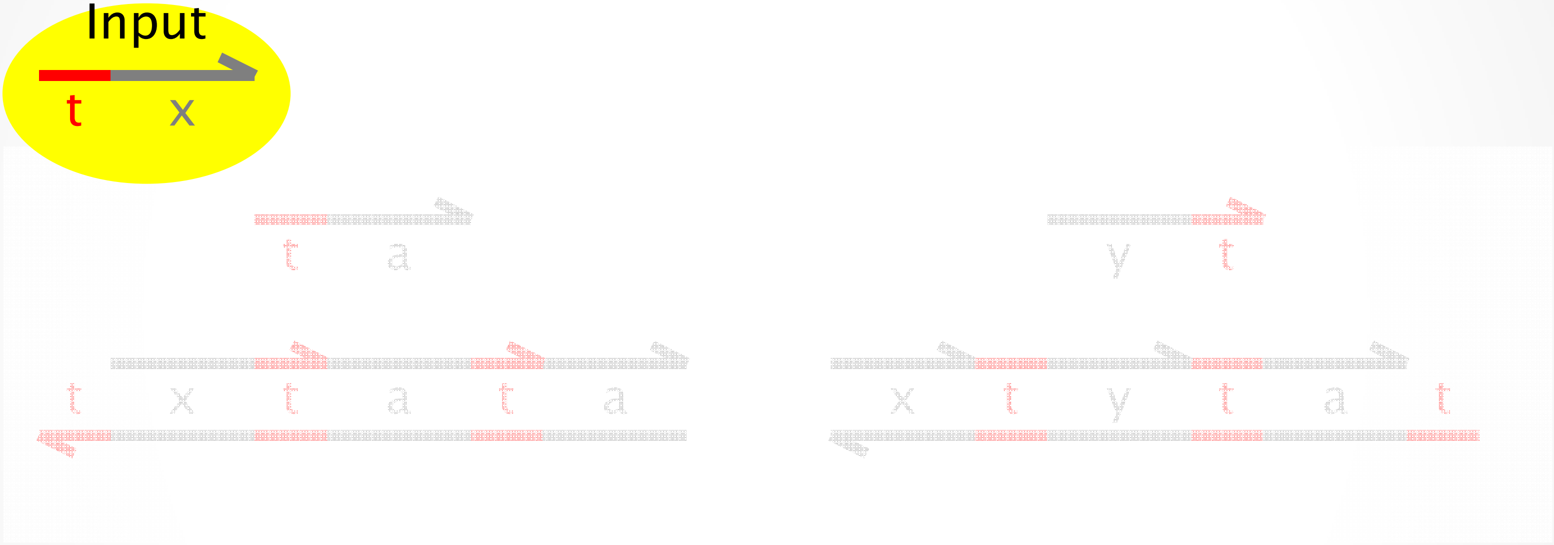


Two-Domain DNA Strand Displacement

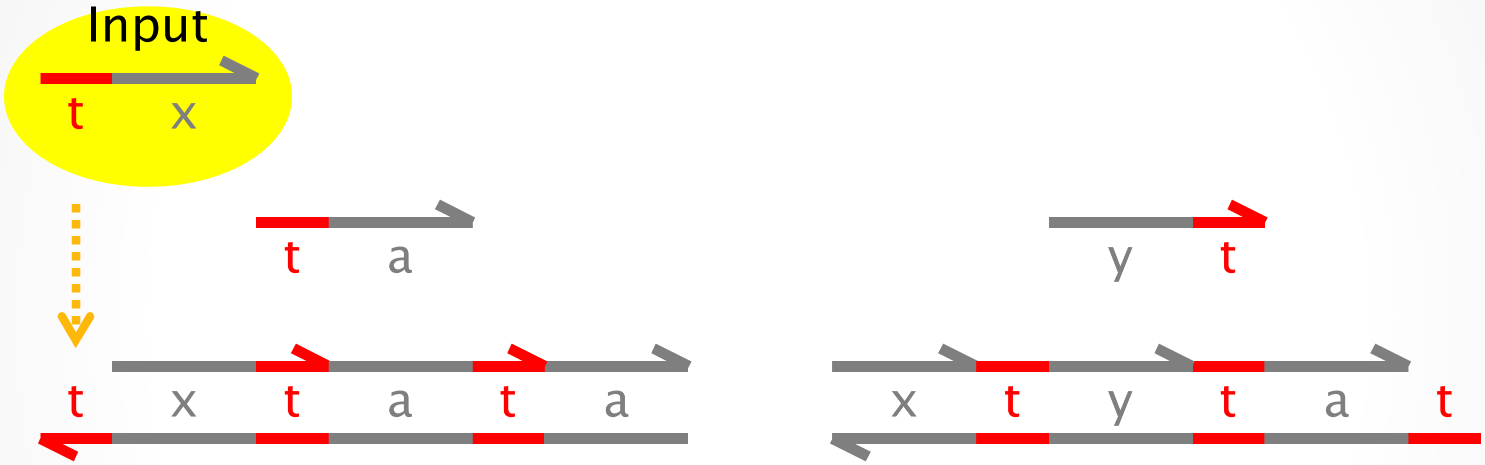
Luca Cardelli

In S. B. Cooper, E. Kashefi, P. Panangaden (Eds.):
Developments in Computational Models (DCM 2010).
EPTCS 25, 2010, pp. 33–47. May 2010.

Transducer x→y



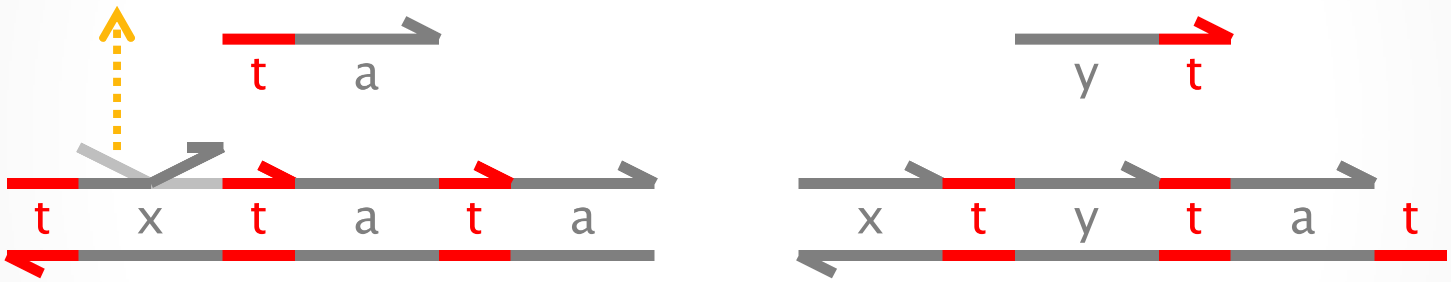
Transducer x→y



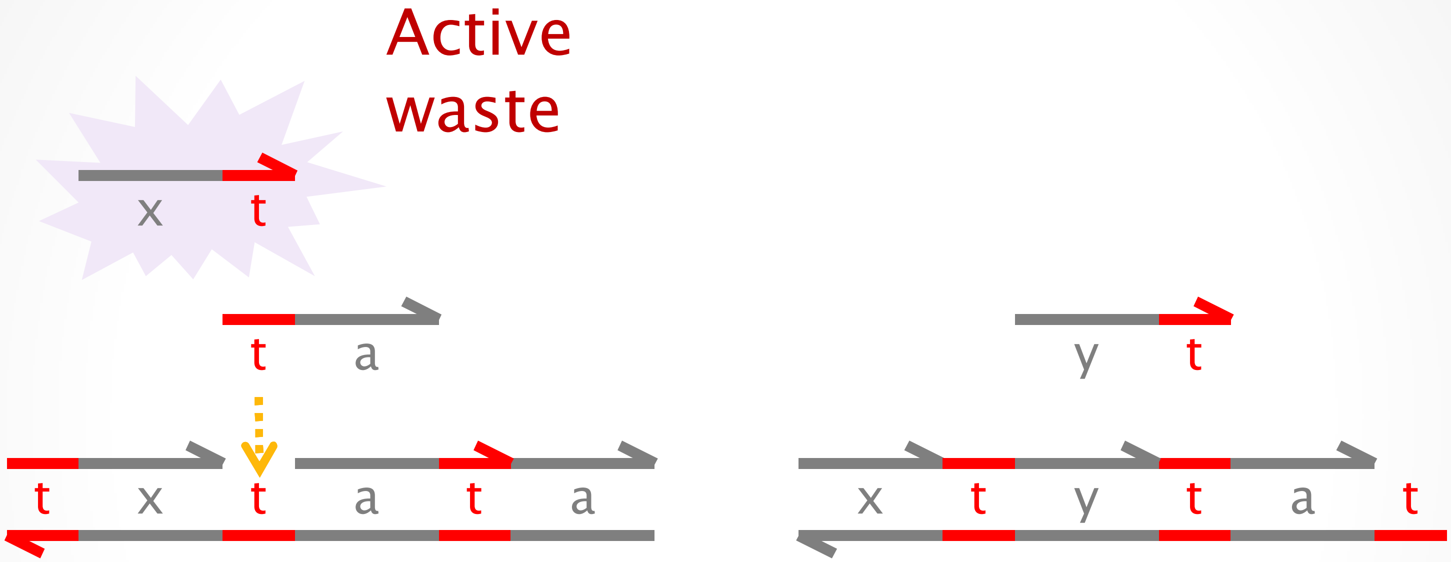
Built by self-assembly!

ta is a *private* signal (a different 'a' for each xy pair)

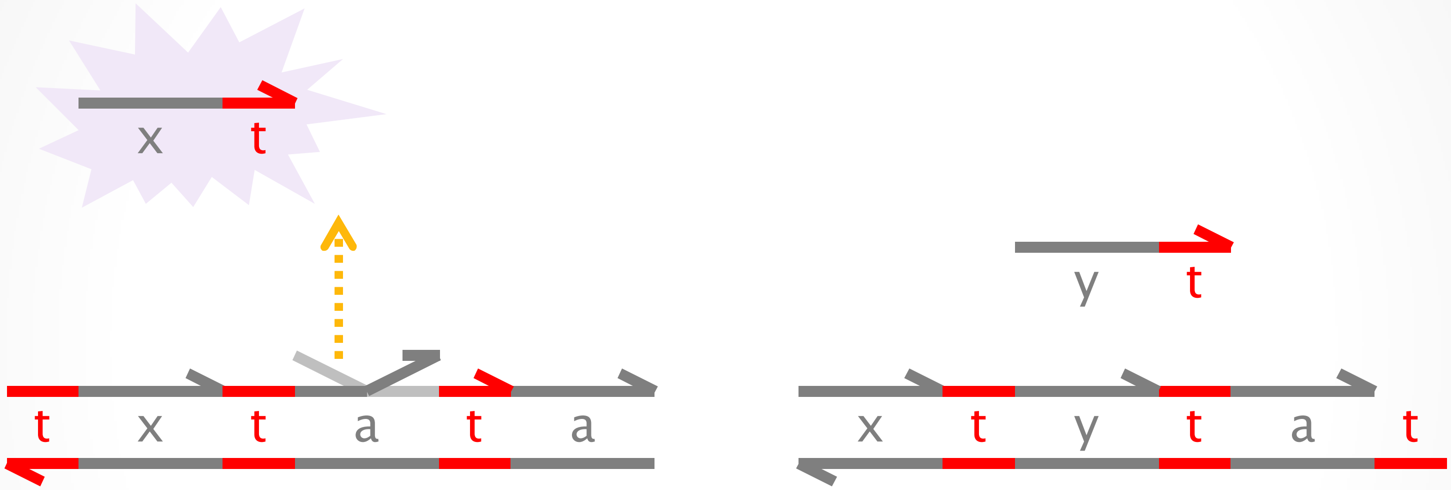
Transducer x→y



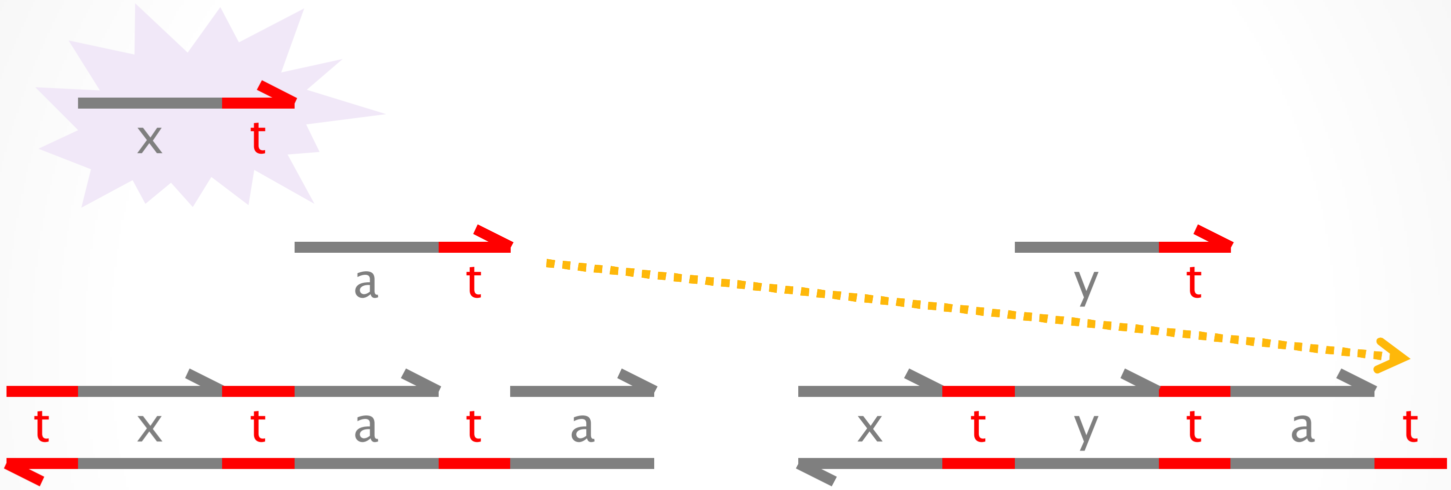
Transducer x→y



Transducer x→y

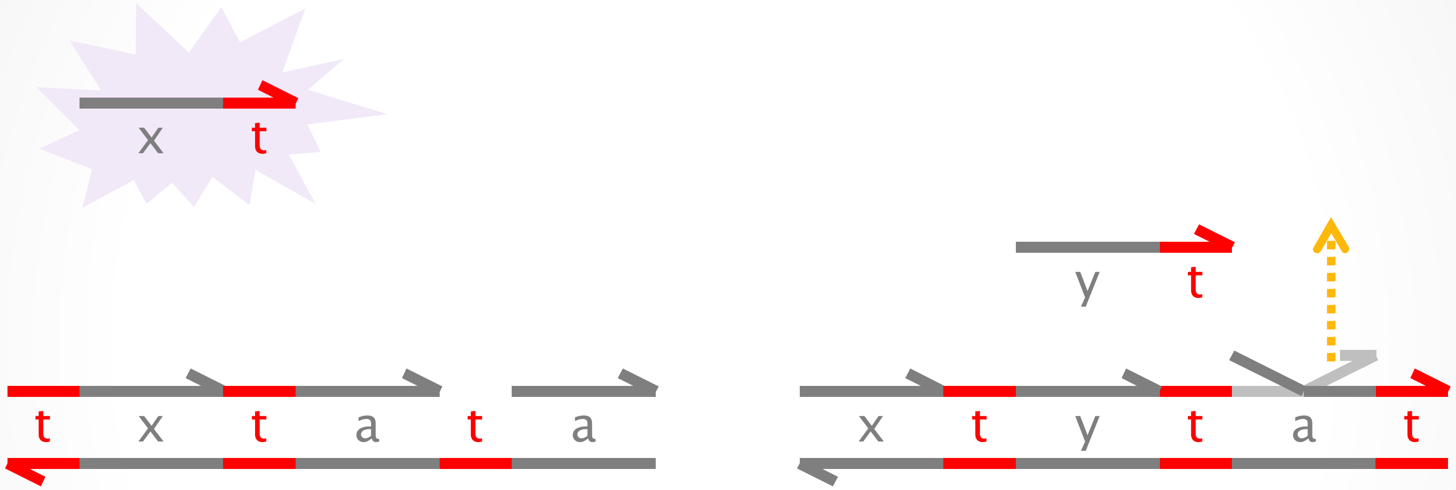


Transducer x→y

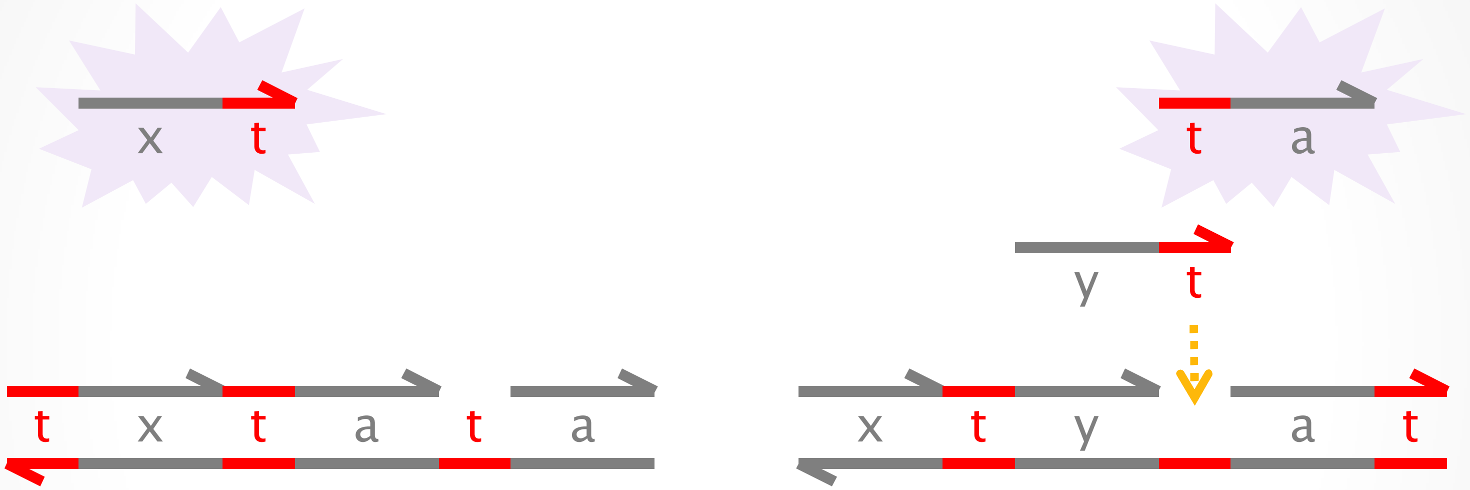


So far, a tx *signal* has produced an at *cosignal.*
But we want signals as output, not cosignals.

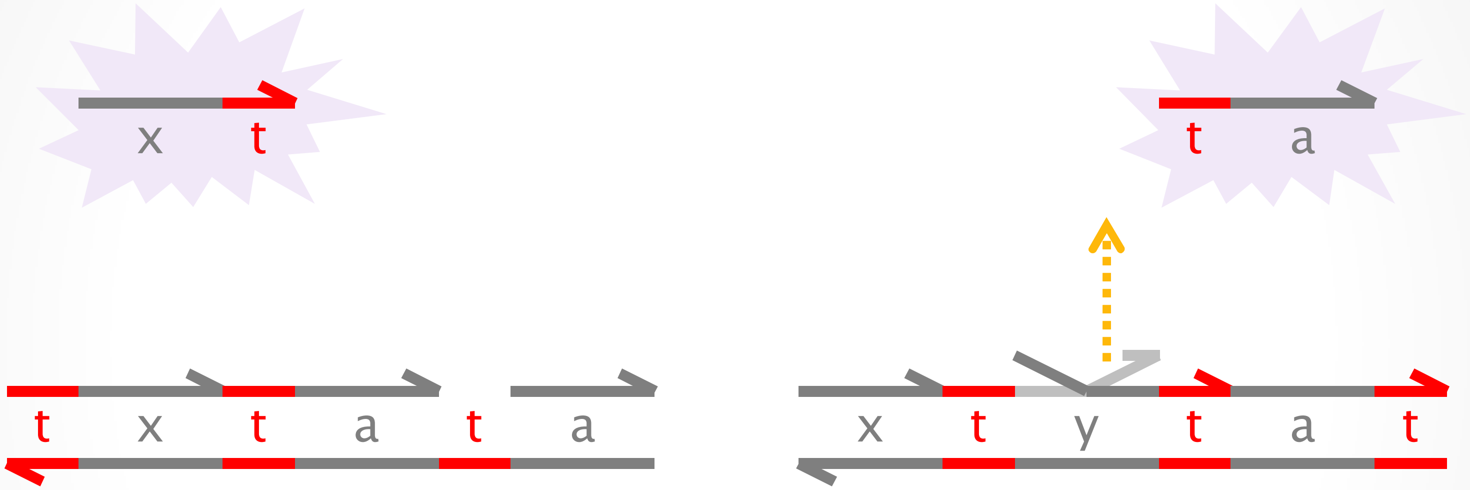
Transducer x→y



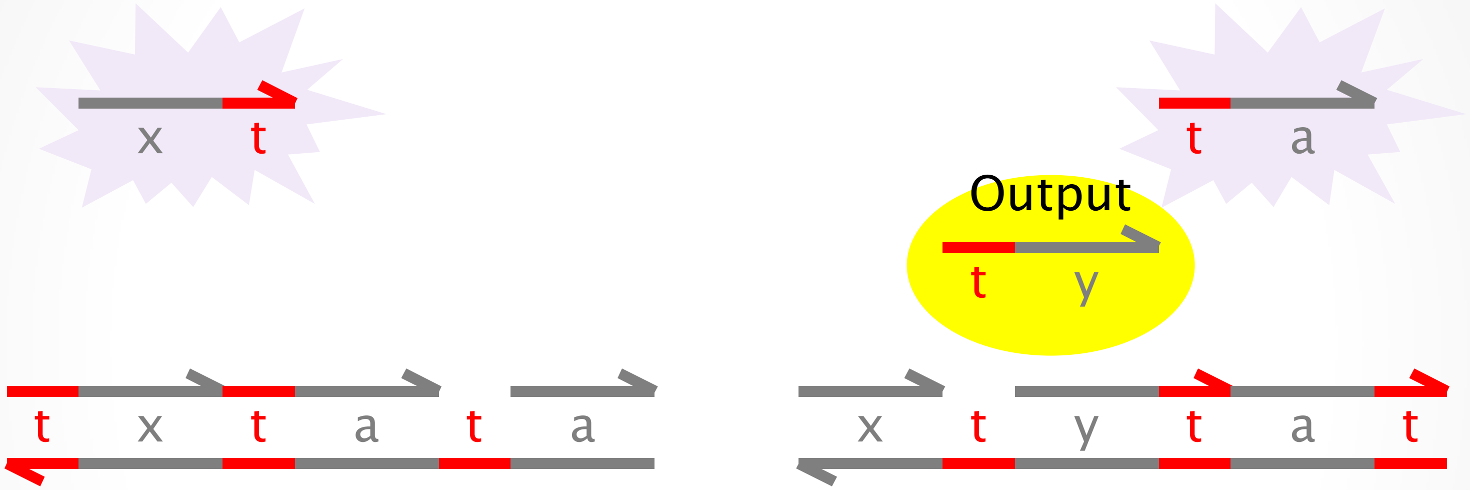
Transducer x→y



Transducer x→y



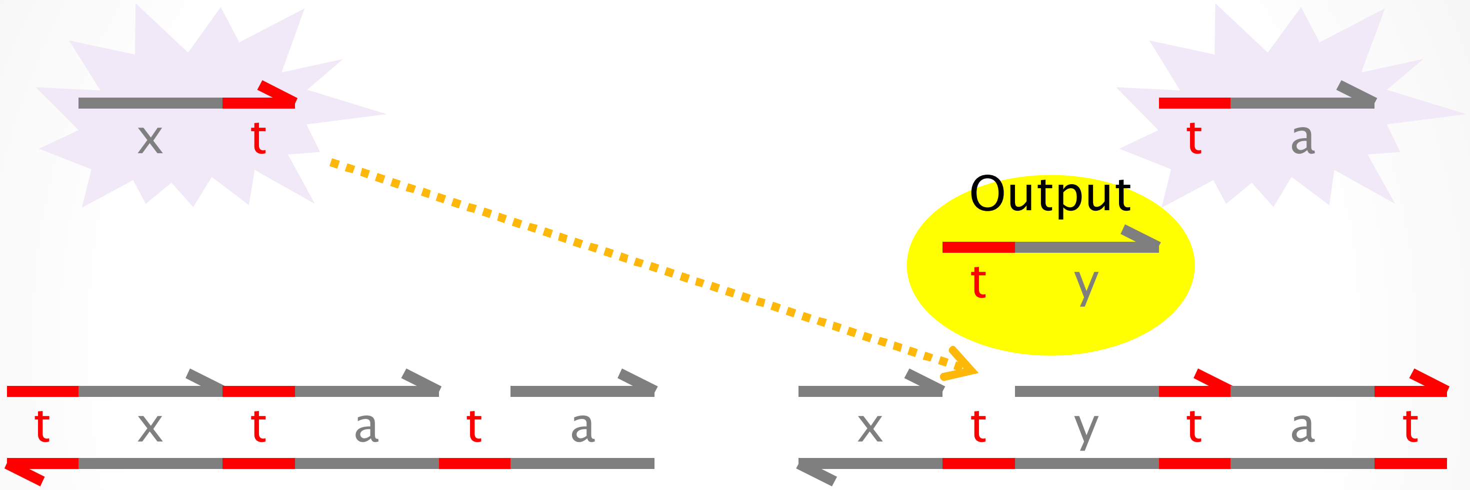
Transducer x→y



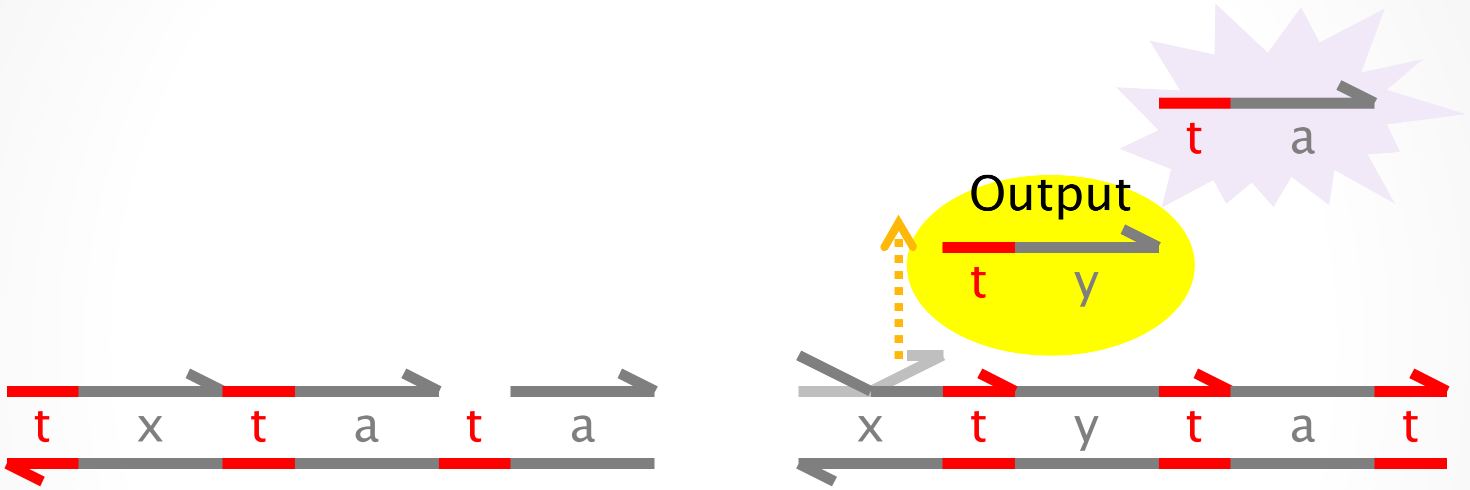
Here is our output ty *signal*.

But we are not done yet:
1) We need to make the output irreversible.
2) We need to remove the garbage.
We can use (2) to achieve (1).

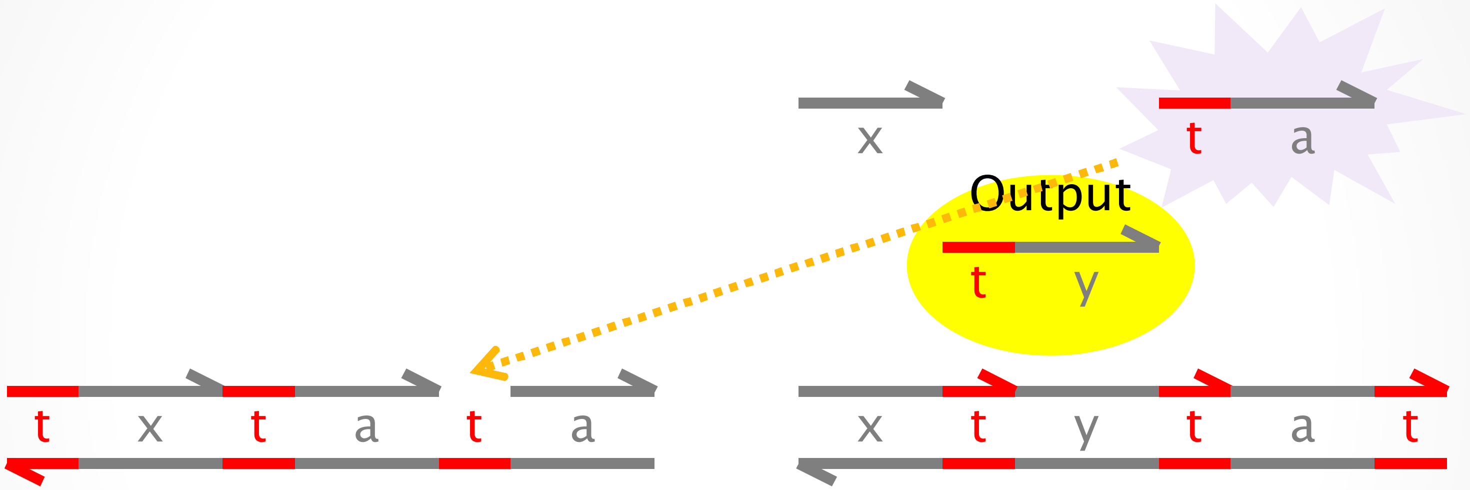
Transducer x→y



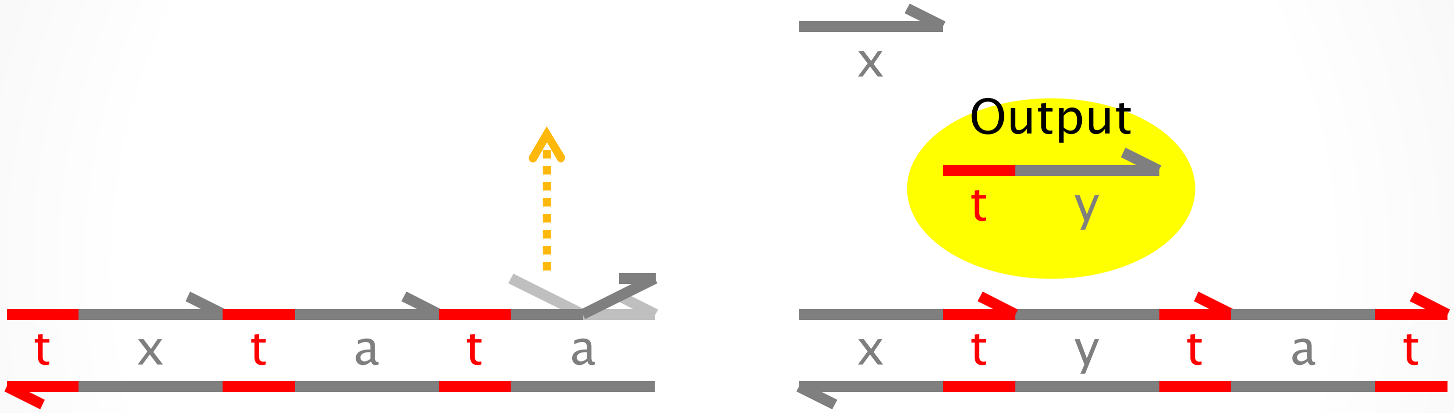
Transducer x→y



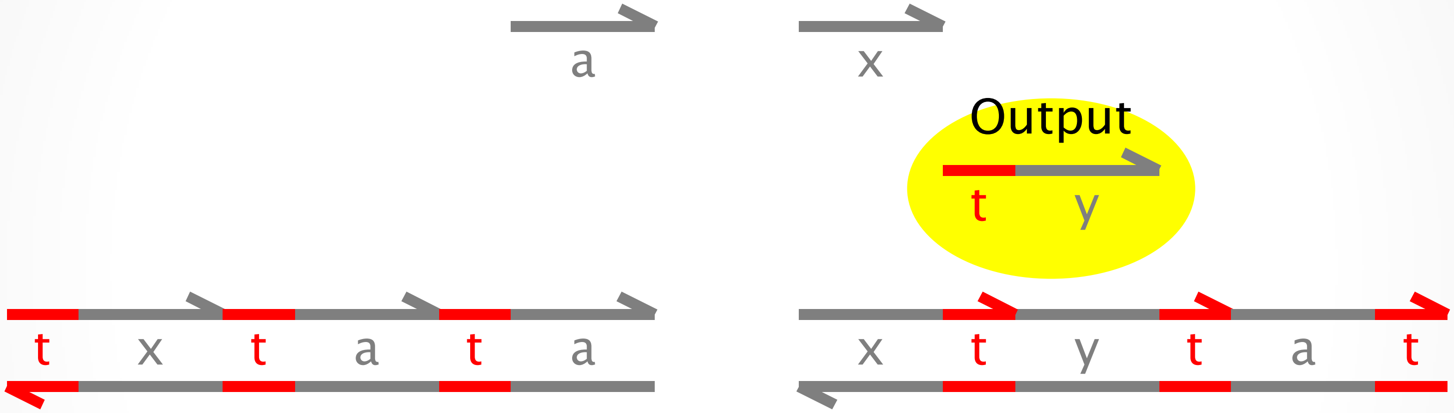
Transducer x→y



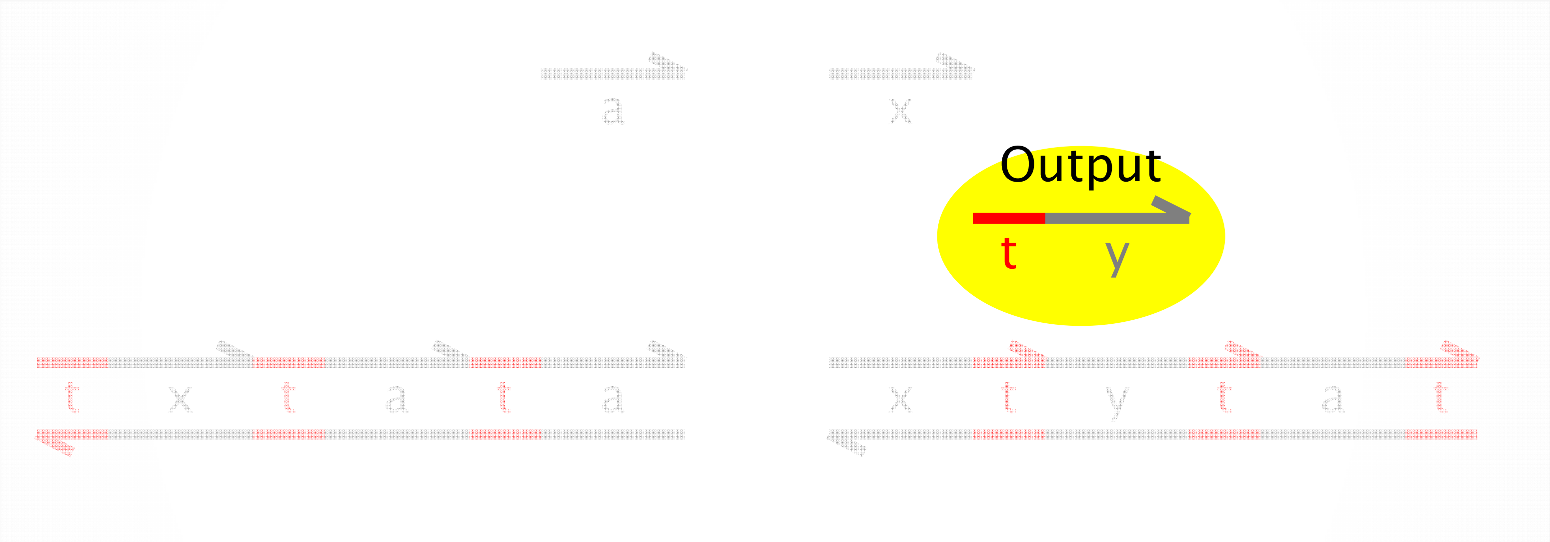
Transducer x→y



Transducer x→y



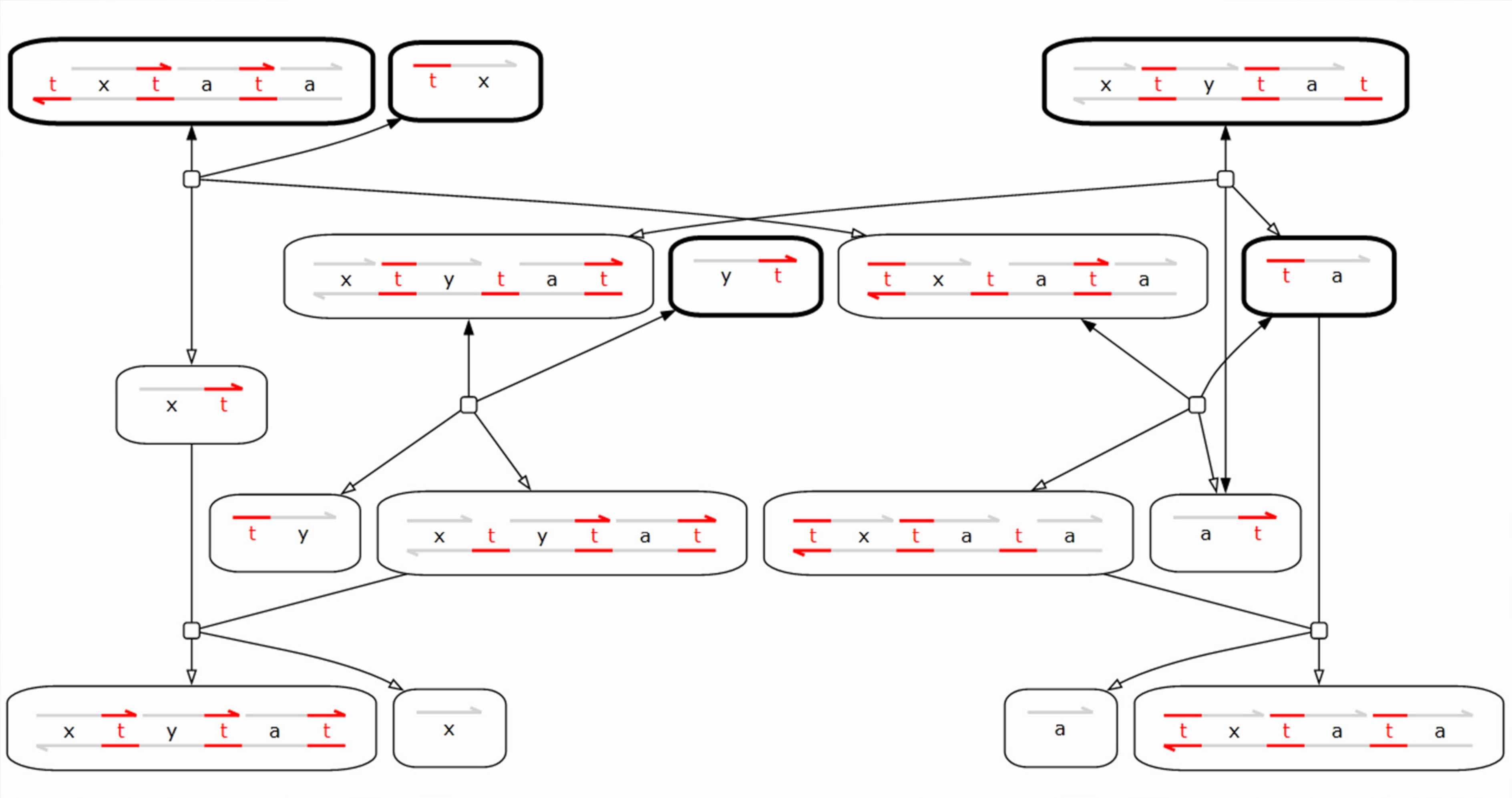
Transducer x→y



Done.

N.B. the gate is consumed: it is the energy source.

Reaction Graph for x→y



General n×m Join–Fork

- Easily generalized to 2+ inputs (with 1+ collectors).
- Easily generalized to 2+ outputs.

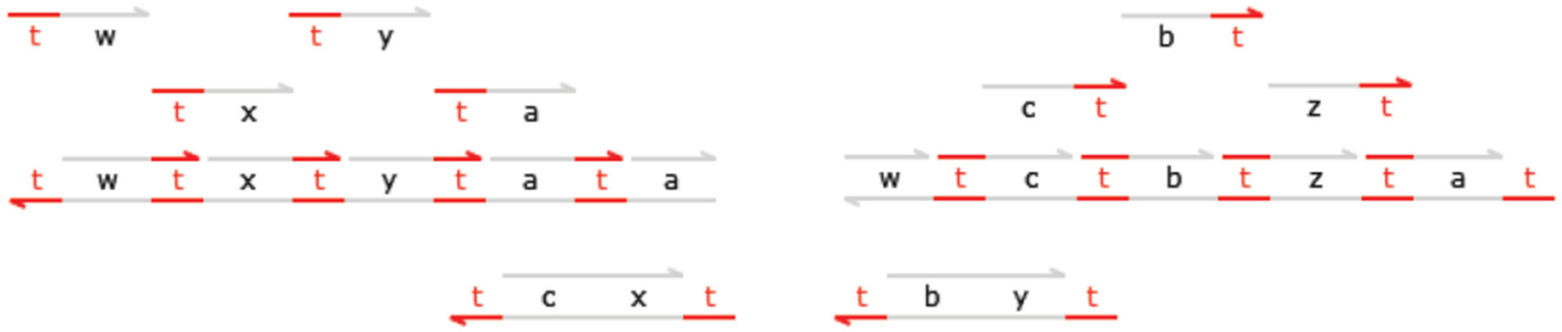
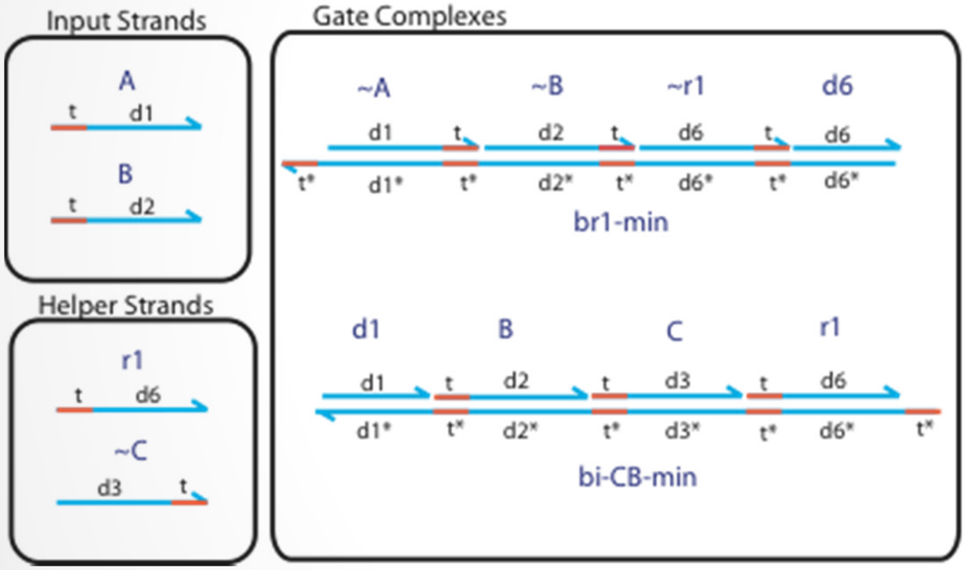


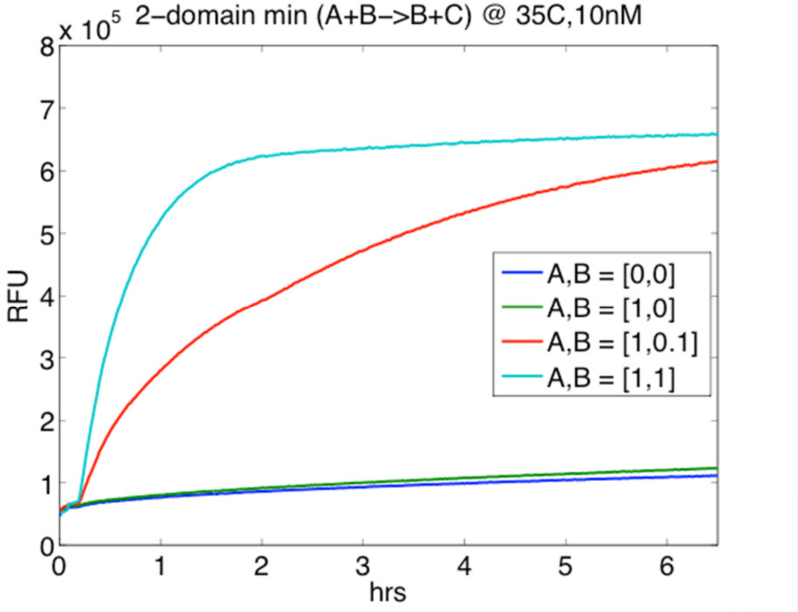
Figure 9: 3-Join $J_{wxyz} \mid tw \mid tx \mid ty \rightarrow tz$: initial state plus inputs tw, tx, ty.

Experiments

Georg Seelig, Matt Olson
(U.Washington)

$A + B \rightarrow B + C$

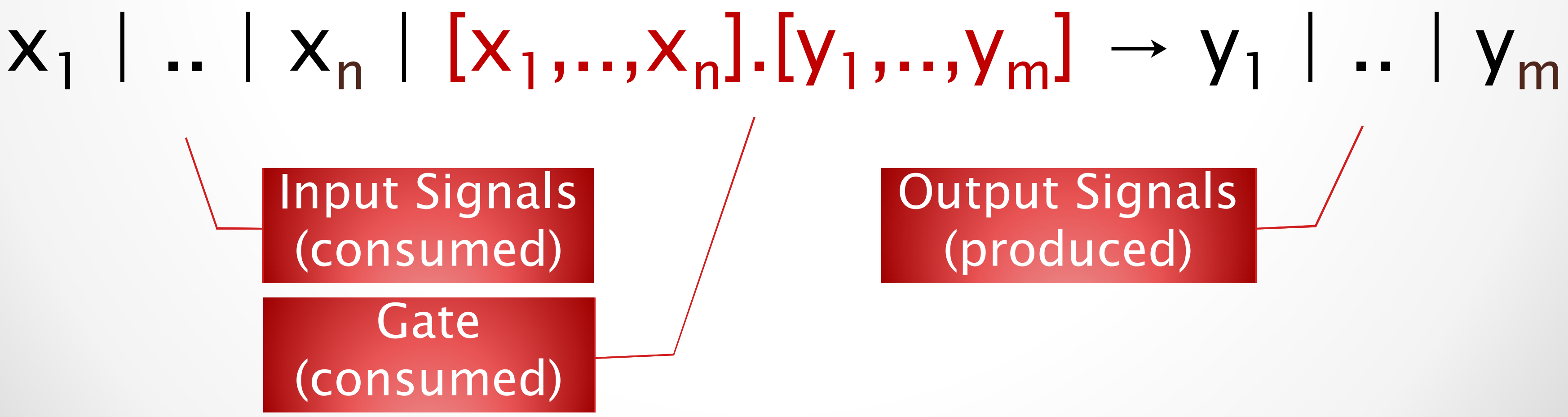




Compilation and Verification

Strand Algebra

- We have seen a (2-domain strand displacement) implementation of a class of computational gates
- More abstractly described as a *strand algebra*: an intermediate language for molecular computing
    - Signals: x
    - Gates: $[x_1,..,x_n].[y_1,..,y_m]$
    - Parallel composition: |
    - Populations: (…)*



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

Input Signals (consumed)

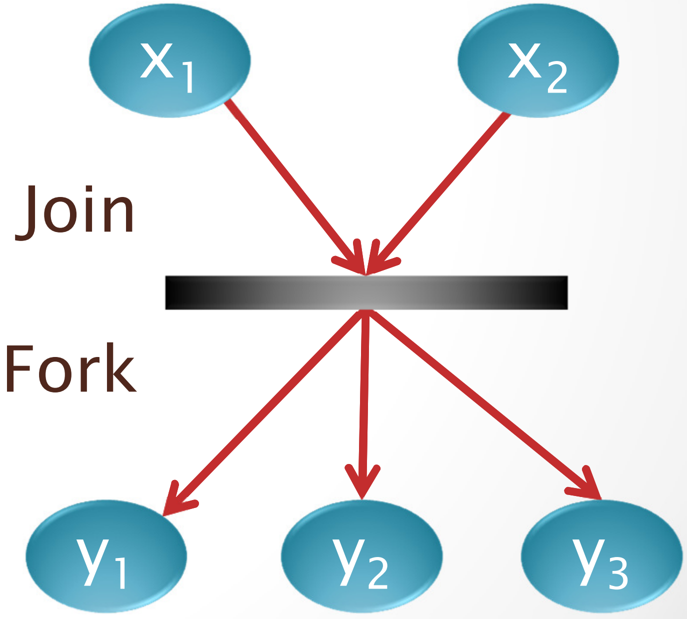
Gate (consumed)

Output Signals (produced)

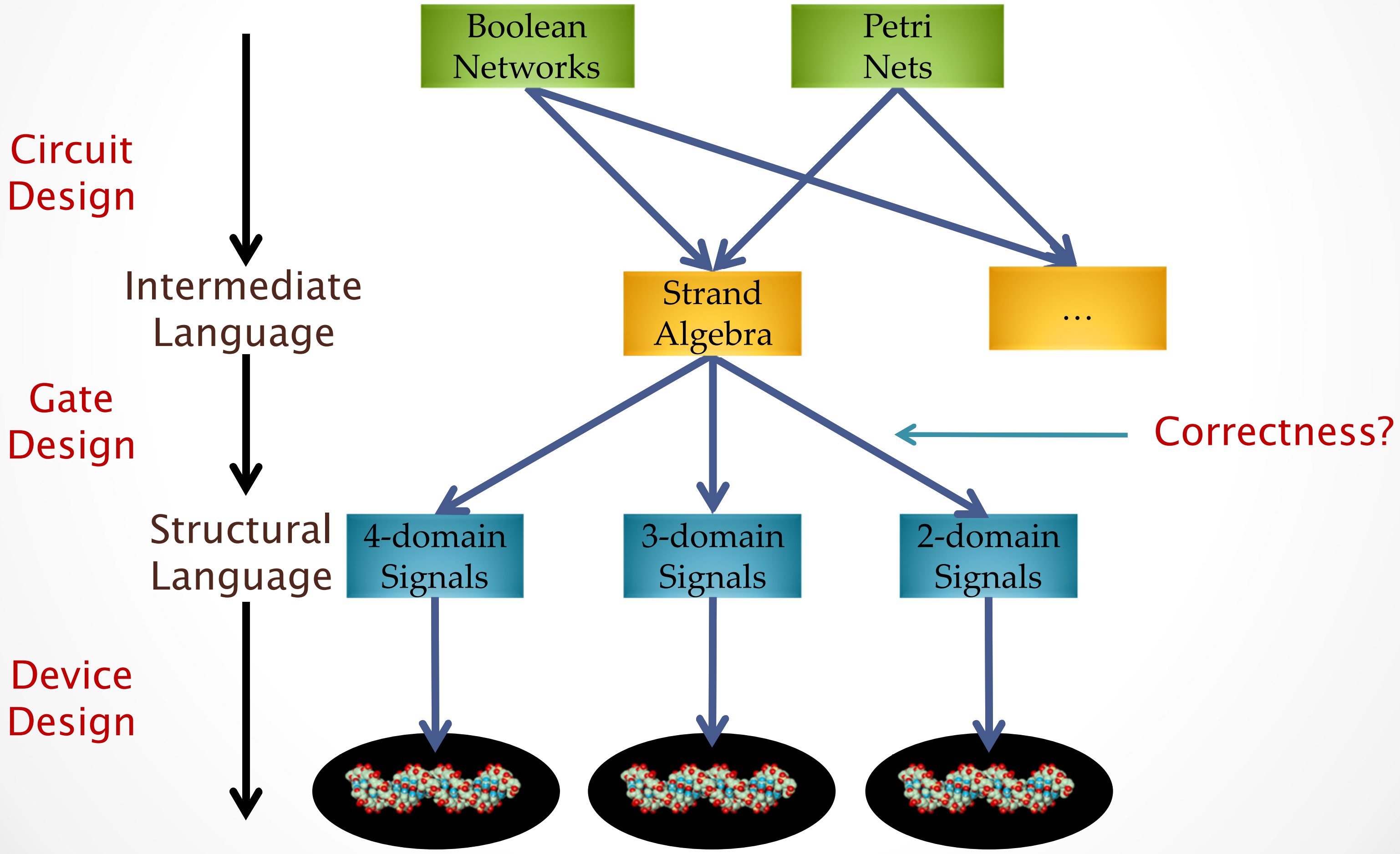
Computational Power

- Equivalent to Petri Nets
  - Not Turing complete, but a rich class nonetheless.
  - The correspondence is not completely trivial: gates are consumed by activation, hence a persistent Petri net transition requires a stable population of gates.

- Many other abstract machines are expressible
  - Boolean networks
  - Interacting Automata
  - Population Protocols
  - Chemistry itself



Molecular Compilation



Optimization Issues

- Reduce number of species
- Optimize kinetics
- Etc.

Verification Issues

- Environment
  - The nano-environment is messy (stochastic noise, failures, etc.)
  - But we should al least ensure our designs are *logically correct*

- Verifying Components
  - Reversible reactions (infinite traces)
  - Interferences (deadlocks etc.) between copies of the same gate
  - Interferences (deadlocks etc.) between copies of different gates
  - Removal of active byproducts (garbage collection) is tricky

- Verifying Populations
  - Gates come in (large) populations
  - Each population *shares private domains* (technologically unavoidable)
  - Correctness of populations means proofs with large state spaces

Correctness

- The spec of a transducer:

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

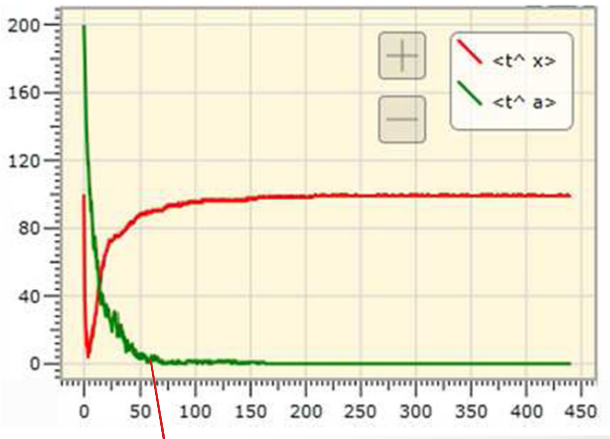
  - Is it true at all?
  - Is it true *possibly, necessarily,* or *probabilistically ?*
  - Is it true in the context of a *population of identical transducers?*
  - Is it true *in all possible contexts?*
  - Is it true (only) for *infinite populations*?

Interfering Transducers

- Let a be the private transducer domain, but let's share it between x.y and y.x

- Interference: $x._{a}y \mid y._{a}x \mid x \nrightarrow^{\forall} x$

- But still: $x._{a}y \mid y._{a}x \mid x \mid y \rightarrow^{\forall} x \mid y$



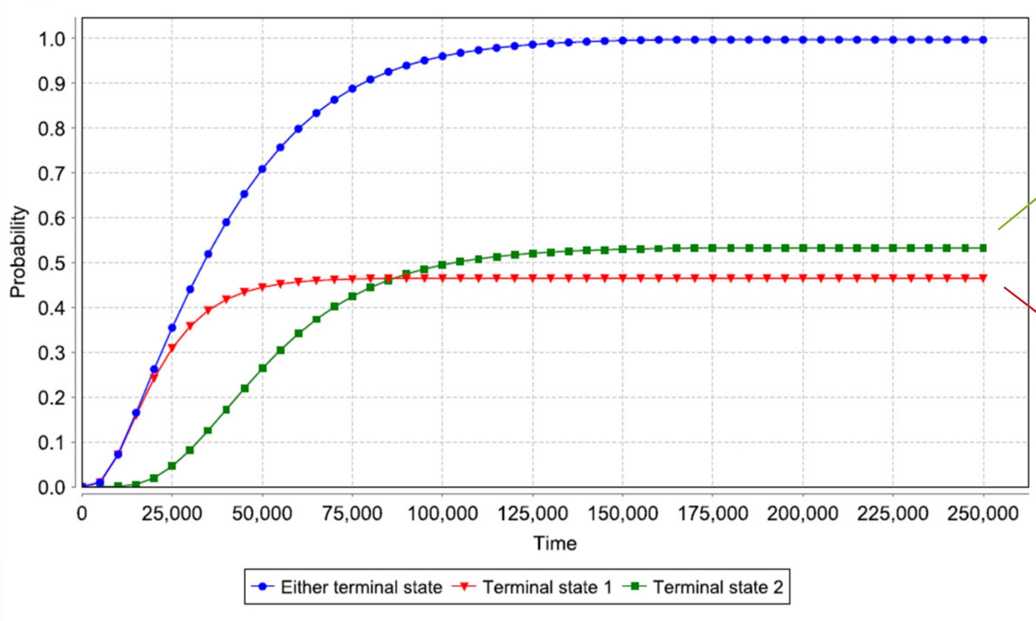
Stuck gates in a population of 200

- A large population of such gates in practice does not deadlock easily.

- The wisdom of crowds: individuals can be wrong, but the population is all right.

Modelchecking DNA Systems

- Using the PRISM stochastic modelchecker
  - Termination probability of interfering transducers
    x | $x._{a}y$ | $y._{a}z$



Correct Termination

Incorrect Termination

L. Cardelli, M. Kwiatkowska, M. Lakin, D. Parker and A. Phillips.
Design and Analysis of DNA Circuits using Probabilistic Model Checking.
http://qav.comlab.ox.ac.uk/papers/dna-pmc.pdf. September 2010

Conclusions

- A new architecture for molecular circuits
  - Simple signals, simple gate structures.
  - Self-cleaning: no garbage left by operation (except inert).
  - Enabling new ways of assembling gates.
  - Experimental evidence that it works.

- A correspondingly simple algebra
  - As an intermediate language for molecular compilers.
  - For verifying gate designs mechanically.

- Molecular Programming
  - Telling (some class of) molecules how to behave.
  - Controlling (biological) systems at the nano scale.

Acknowledgments

- Microsoft Research
  - Andrew Phillips
- Caltech
  - Winfree Lab
- U.Washington
  - Seelig Lab