Programming Molecules

Luca Cardelli
Microsoft Research

ISAT Workshop, SRI, 2012-03-07
http://lucacardelli.name

Outline

- Part I: Analyzing molecular networks
  - We try do discover the function of the network.
  - We try to understand how the structure is dictated by the function (and other natural constraints).

- Part II: Engineering molecular networks
  - We know the function we want to implement.
  - We use the structures we have available to implement the function. But we want to do this *in general* (programmatically).

Part I

Systems Biology

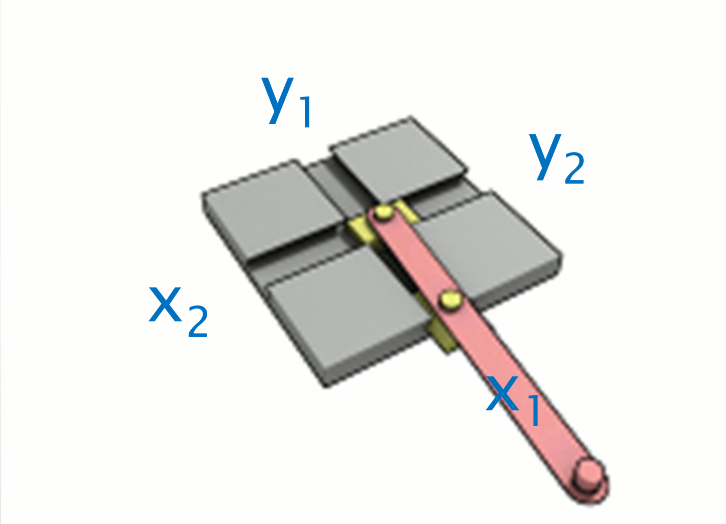
- *or* -

How Does Nature Build Molecular Oscillators?

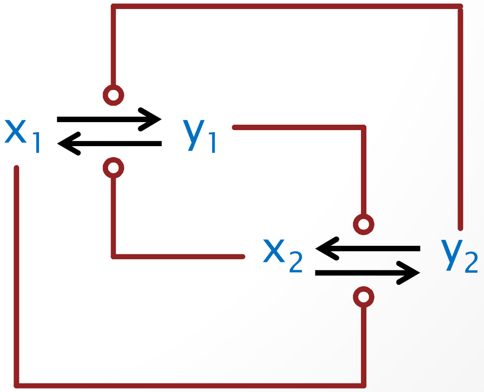
The Trammel of Archimedes

- A device to draw ellipses
  - Two interconnected **switches.**
  - When one switch is on (off) it flips the other switch on (off). When the other switch is on (off) it flips the first switch off (on).
  - The amplitude is kept constant by mechanical constraints.

The function



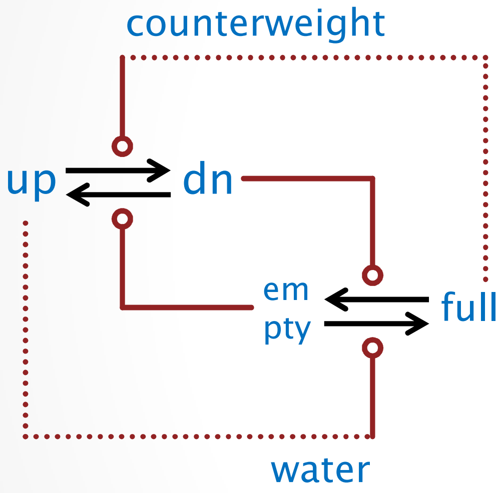
The network



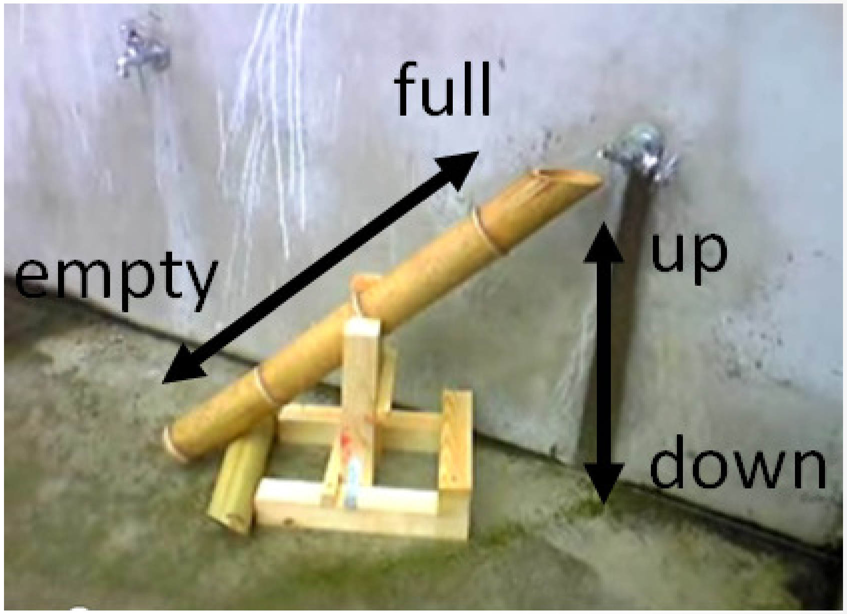
en.wikipedia.org/wiki/Trammel_of_Archimedes

The Shishi Odoshi

- A Japanese scarecrow (lit. scare-deer)
  - Used by Bela Novak to illustrate the cell cycle switch.



empty + up → up + full
up + full → full + dn
full + dn → dn + empty
dn + empty → empty + up

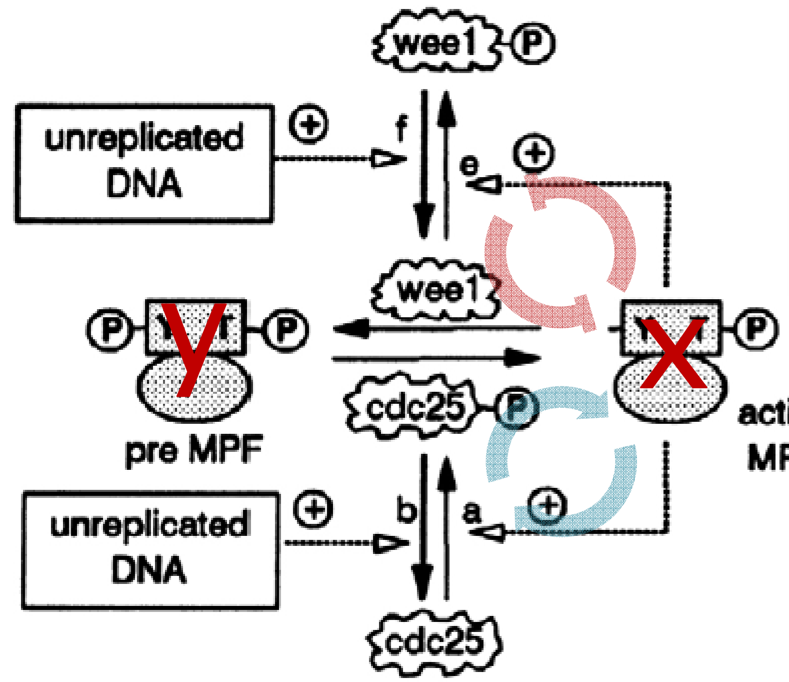


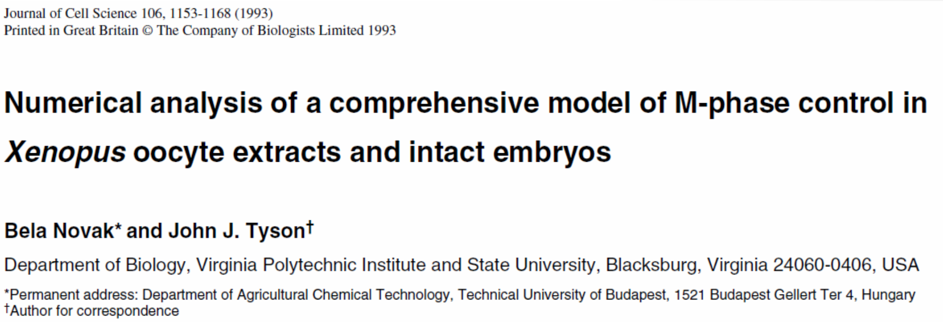
http://www.youtube.com/watch?v=VbvecTIftcE&NR=1&feature=fvwp

Outer switched connections replaced by constant influxes: tap water and gravity.

The Cell Cycle Switch

- At the core of the cell-cycled oscillator.
  - This network is universal in all Eukaryotes [P. Nurse].




Journal of Cell Science 106, 1153-1168 (1993)
Printed in Great Britain © The Company of Biologists Limited 1993

Numerical analysis of a comprehensive model of M-phase control in *Xenopus* oocyte extracts and intact embryos

Bela Novak* and John J. Tyson†

Department of Biology, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24060-0406, USA

*Permanent address: Department of Agricultural Chemical Technology, Technical University of Budapest, 1521 Budapest Gellert Ter 4, Hungary
†Author for correspondence

- Double positive feedback on x
- Double negative feedback on x
- No feedback on y
  ???

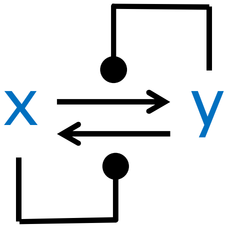
  - Well studied. But why this structure?

How to Build a Switch

- What is a "good" switch?
  - We need first a *bistable* system: one that has two *distinct* and *stable* states. I.e., given *any* initial state the system must *settle* into one of two states.
  - The settling must be *fast* (not get stuck in the middle for too long) and *robust* (must not spontaneously switch back).
  - Finally, we need to be able to *flip* the switch: drive the transitions by external inputs.

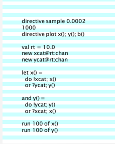
A Bad Algorithm

- Direct x-y competition
  - x catalyzes the transformation of y into x
  - y catalyzes the transformation of x into y

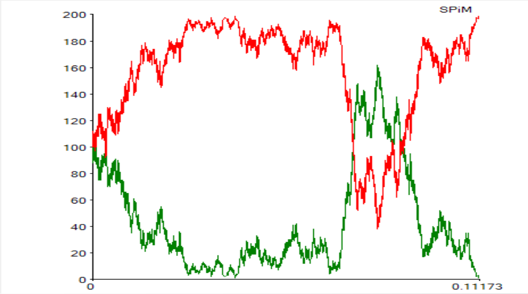


$$x + y \to x + x$$
$$y + x \to y + y$$

- This system is bistable, but
  - Convergence to a stable state is *slow* (a random wa
  - *Any* perturbation of a stable state can initiate a random walk to the other stable state

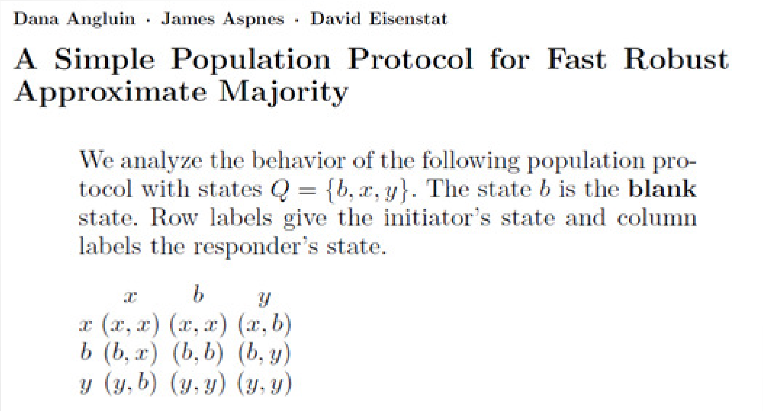


```
directive sample 0.0002
1000
directive plot x(); y(); b()

val rt = 10.0
new xcat@rt:chan
new ycat@rt:chan

let x() =
  do !xcat; x()
  or ?ycat; y()

and y() =
  do !ycat; y()
  or ?xcat; x()

run 100 of x()
run 100 of y()
```



A Very Good Algorithm

- Approximate Majority
  - Decide which of two populations is in majority
- A fundamental ‘population protocol’
  - Agents in a population start in state x or state y.
  - A pair of agents is chosen randomly at each step, they interact ("collide") and change state.
  - The whole population must eventually agree on a majority value (all x or all y) with probability 1.

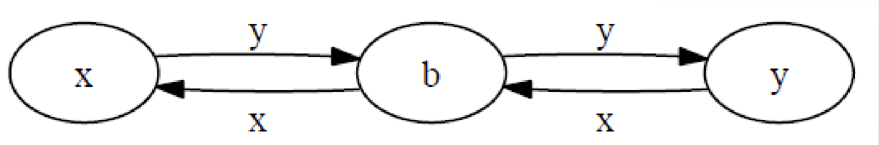


Dana Angluin · James Aspnes · David Eisenstat

A Simple Population Protocol for Fast Robust Approximate Majority

We analyze the behavior of the following population protocol with states $Q = \{b, x, y\}$. The state b is the **blank** state. Row labels give the initiator's state and column labels the responder's state.

	x	b	y
x	(x, x)	(x, x)	(x, b)
b	(b, x)	(b, b)	(b, y)
y	(y, b)	(y, y)	(y, y)



Third ‘undecided’ state.

Properties

- With high probability, for n agents
  - The number of state changes before converging is O(n log n)
  - The total number of interactions before converging is O(n log n)
  - The final outcome is correct if the initial disparity is ω(sqrt(n log n))

[Angluin et al.
http://www.cs.yale.edu/homes/aspnes/papers/disc2007-eisenstat-slides.pdf]

- The algorithm is the fastest possible
  - Must wait Ω(n log n) steps in expectation for all agents to interact

- Logarithmic time bound
  - Parallel time is the number of steps divided by the number of agents.
  - In parallel time the algorithm converges with high probability in O(log n).
  - That is true for any initial conditions, even x=y!

"Although we have described the population protocol model in a sequential light, in which each step is a single pairwise interaction, interactions between pairs involving different agents are independent and may be thought of as occurring in parallel. In measuring the speed of population protocols, then, we define 1 unit of parallel time to be jV j steps. The rationale is that in expectation, each agent initiates 1 interaction per parallel time unit; this corresponds to the chemists' idealized assumption of a well-mixed solution."
Distributed Computing 21(2):87-102.

Chemical Implementation

$$x + y \rightarrow y + b$$
$$y + x \rightarrow x + b$$
$$b + x \rightarrow x + x$$
$$b + y \rightarrow y + y$$

Bistable

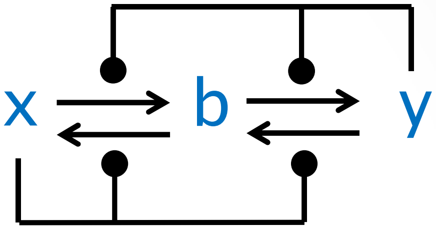
Even when x=y! (stochastically)

Fast

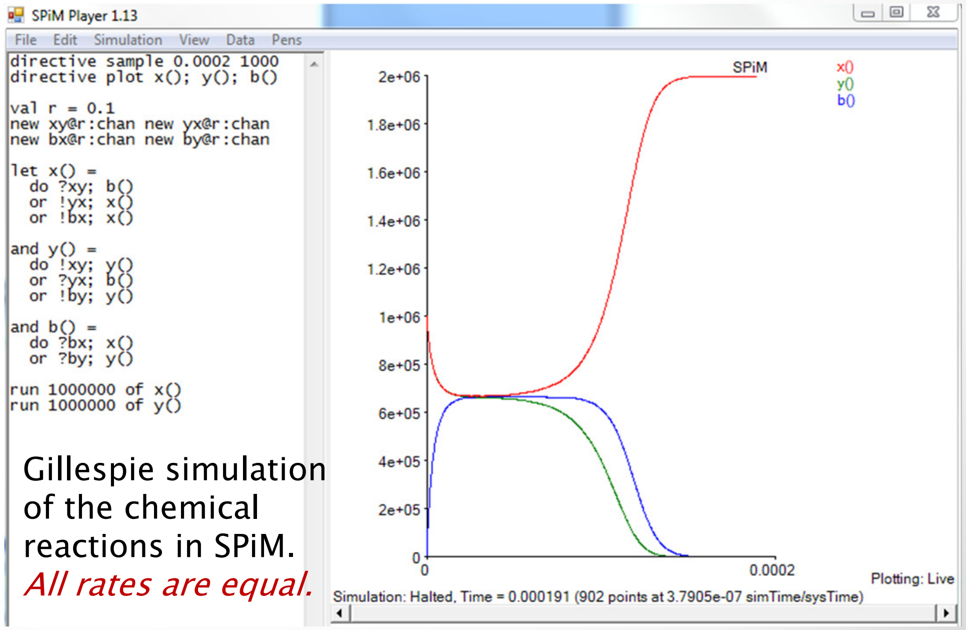
O(log n) convergence time

Robust

$\omega(\sqrt{n} \log n)$ majority wins whp



Worse case test: start with x=y.



```
directive sample 0.0002 1000
directive plot x(); y(); b()

val r = 0.1
new xy@r:chan new yx@r:chan
new bx@r:chan new by@r:chan

let x() =
  do ?xy; b()
  or !yx; x()
  or !bx; x()

and y() =
  do !xy; y()
  or ?yx; b()
  or !by; y()

and b() =
  do ?bx; x()
  or ?by; y()

run 1000000 of x()
run 1000000 of y()
```

Simulation: Halted, Time = 0.000191 (902 points at 3.7905e-07 simTime/sysTime)

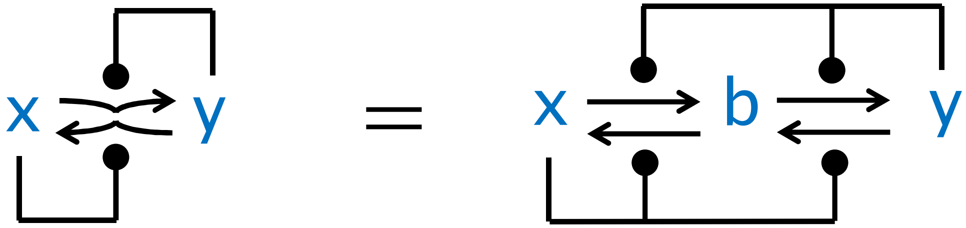
Gillespie simulation of the chemical reactions in SPiM. *All rates are equal.*

Back to the Cell Cycle

- The AM algorithm has great properties for settling a population into one of two states.
- But that is not what the cell cycle uses to switch its populations of molecules.
- Or is it?

Step 1: the AM Network

Abbreviated notation:

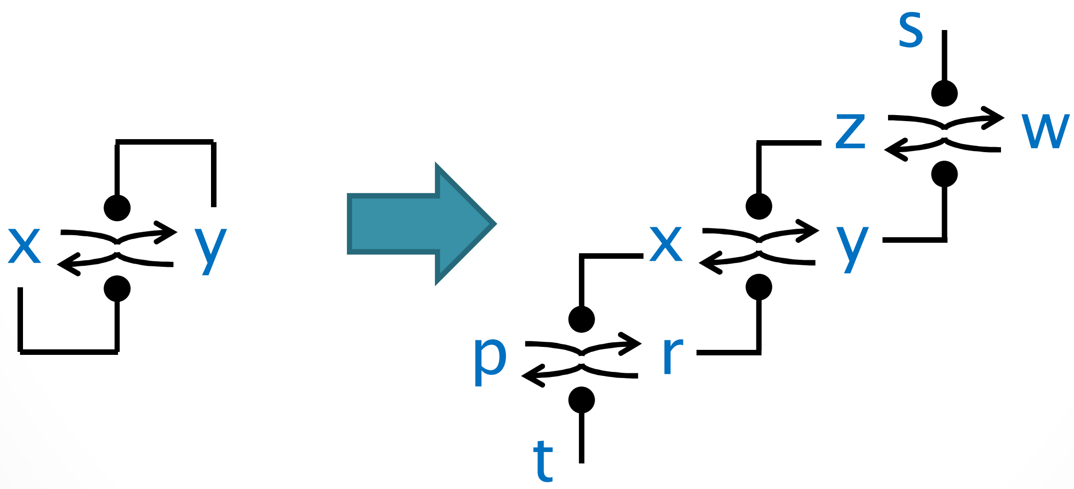


- Autocatalysis, and especially intricate autocatalysis, is not commonly seen in nature. Presumably, it's hard:

$$b + x \rightarrow x + x$$
$$b + y \rightarrow y + y$$

Step 2: remove auto-catalysis

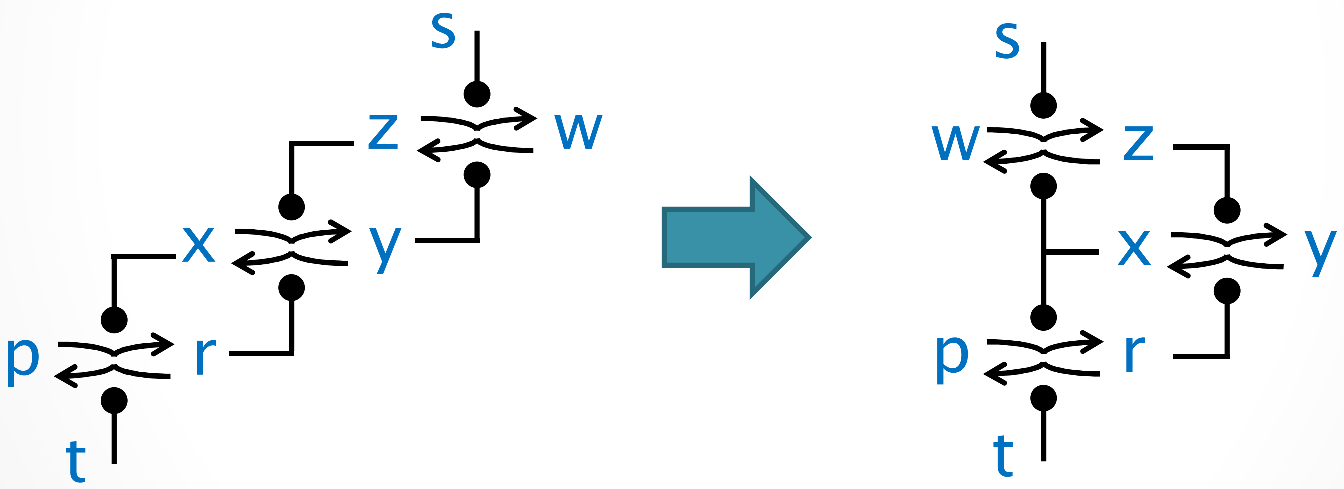
- Replace autocatalysis by mutual (simple) catalysis, introducing intermediate species z, r.
  - Here z breaks the y auto-catalysis, and r breaks the x auto-catalysis, while preserving the feedbacks.
  - z and r need to ‘relax back’ (to w and p) when they are not catalyzed: s and t provide the back pressure.



- Still, x and y (two states of the same molecule) are distinct active catalysts: that is not common in nature either.

Step 3: only one active state

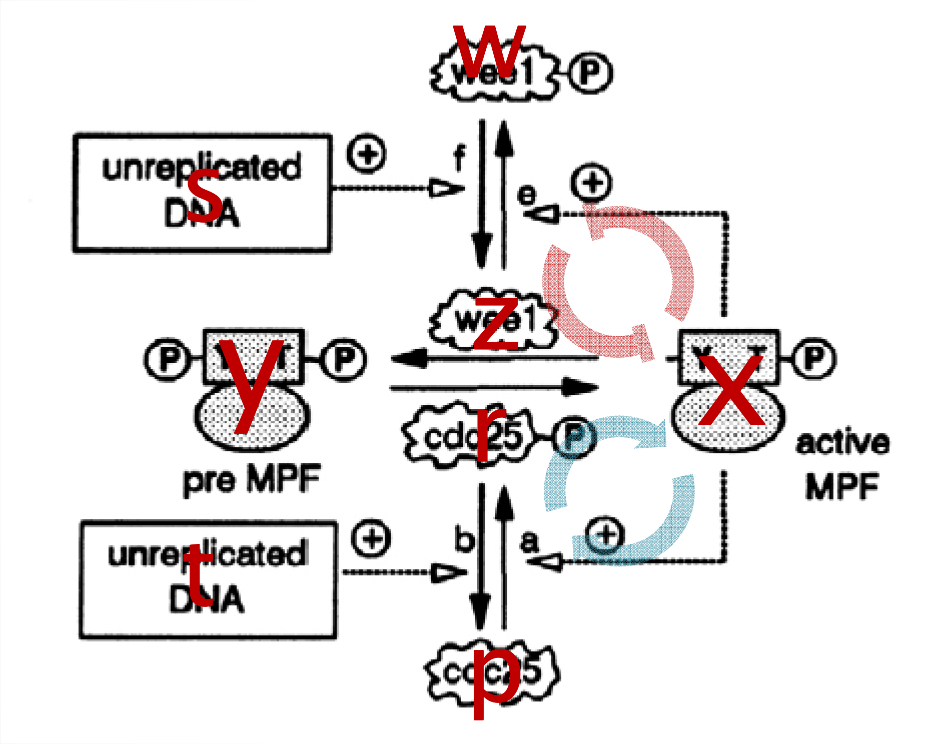
- Remove the catalytic activity of y.
  - Instead of y activating itself through z, we are left with z activating y (which remains passive). Hence, to deactivate y we now need to deactivate z. Since x ‘wants’ to deactivate y, we make x deactivate z.

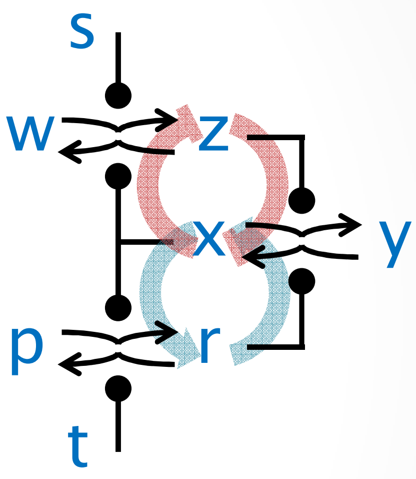


- All species now have one active (x,z,r) and one inactive (y,w,p) form. This is ‘normal’.

Network Structure

- … and that *is* the cell-cycle switch!





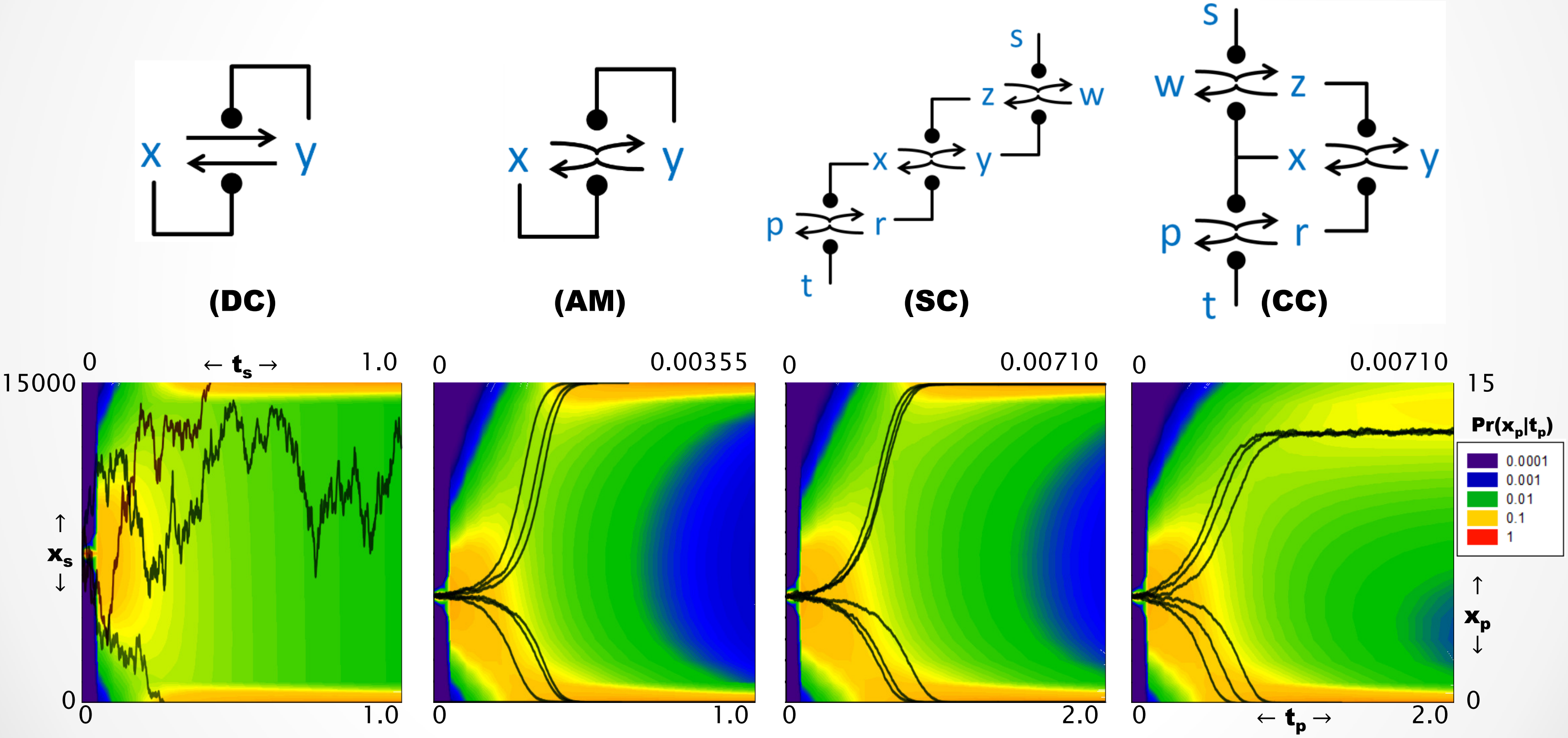
(Some of the bistable states can be enzymatic rather than multi-site phosporylations as in AM.)

- The question is: did we preserve enough *function* through our *network transformations*?

Quantitative Analysis

Switches as Computational Systems – Convergence

Techniques: Stochastic Simulation and Probabilistic Modelchecking

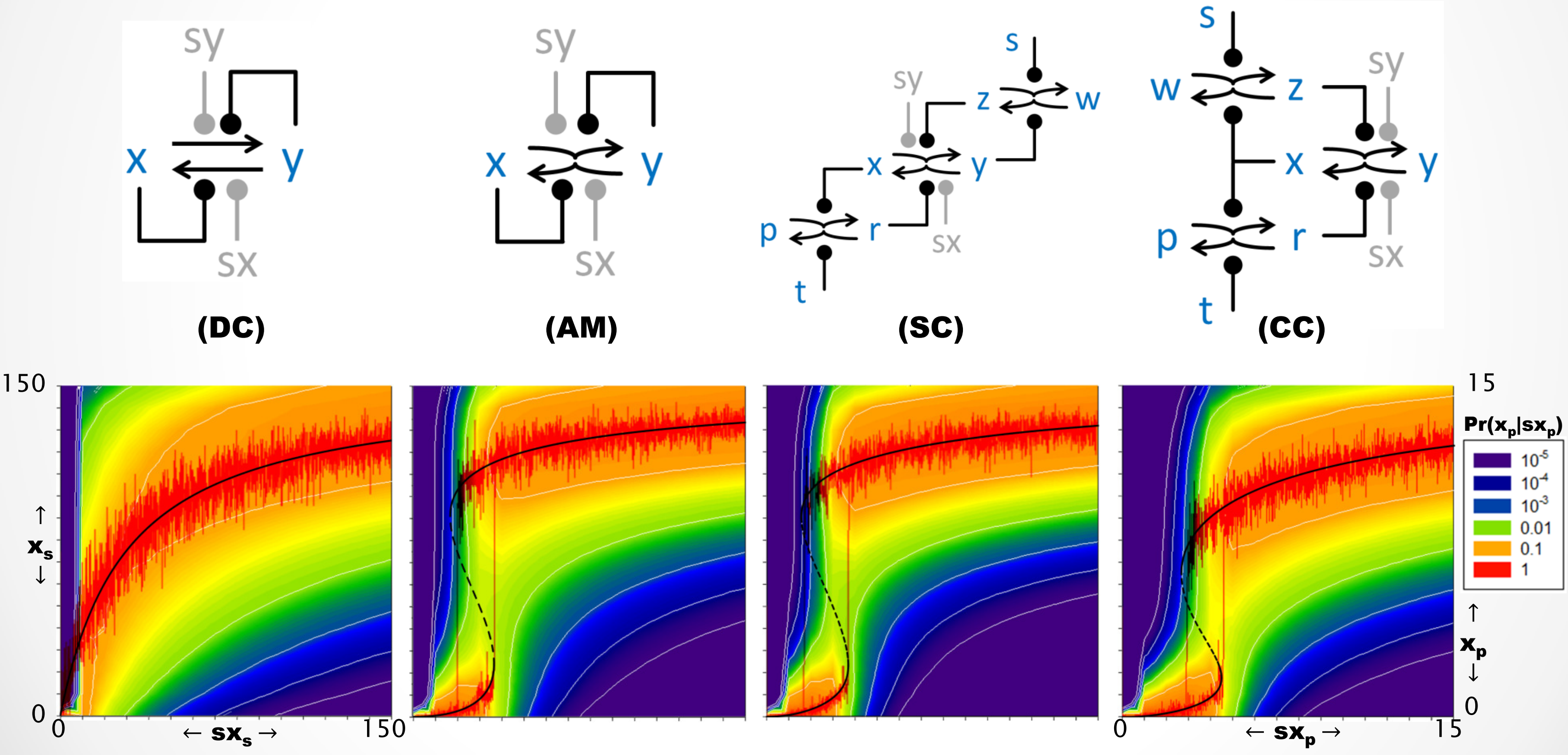


Joint work with Attila Csikász-Nagy

Quantitative Analysis

Switches as Dynamical Systems – Steady State Response

Techniques: as above, plus Dynamical Systems Theory



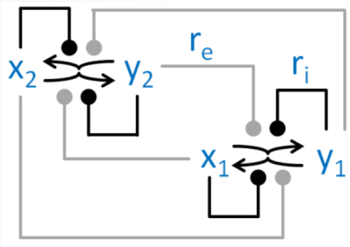
Joint work with Attila Csikász-Nagy

Quantitative Analysis

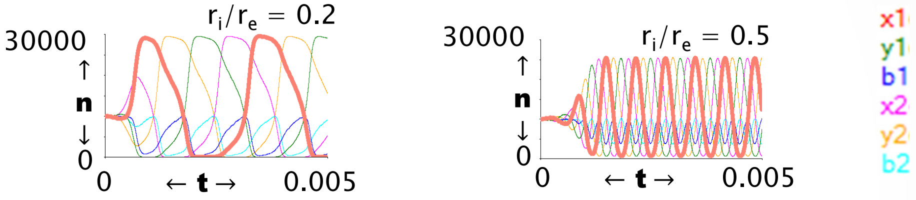
Switches in the context of larger networks

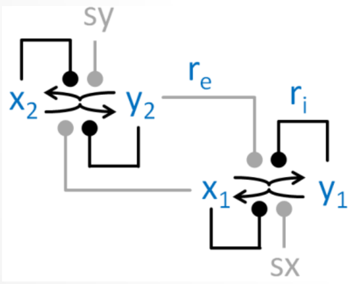
Techniques: testing

(We have better techniques for non-quantitative systems.)

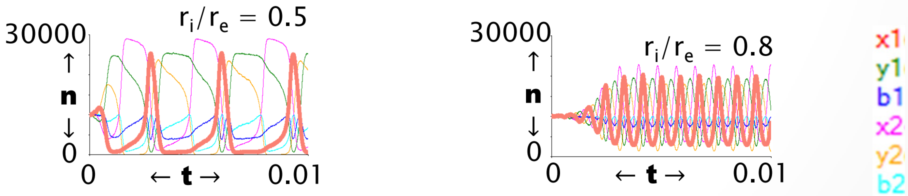


Trammel





Shishi Odoshi



Joint work with Attila Csikász-Nagy

Summary

- Q (traditional): What kind of dynamical system is the cell–cycle switch?
- A (traditional): Bistability – ultrasensitivity – hysteresis …
  Focused on how unstructured sub–populations change over time.

- Q: What kind of algorithmic system is the cell–cylce switch?
- A: Interaction – complexity – convergence ...
  Focused on individual molecules as programmable, structured, algorithmic entities.

Part II

Synthetic Biology
- *or* -
How Can *We* Build Molecular Oscillators? (or any other network?)

Molecular Programming Languages

- Reaction-Based (A + B → C + D) (Chemistry)
  - Limited to finite set of species (no polymerization)
  - Practically limited to small number of species (no run-away complexation)

- Interaction-Based (A = !r; C) (Process Algebra)
  - Reduces combinatorial complexity of models by combining independent submodels connected by interactions.

- Rule-Based (A{-}:B{p} → A{p}:B{-}) (Logic, Graph Rewriting)
  - Further reduces model complexity by describing molecular state, and by allowing one to 'ignore the context': a *rule* is a reaction in an unspecified (complexation/phosphorylation) context.
  - Similar to informal descriptions of biochemical events ("narratives").

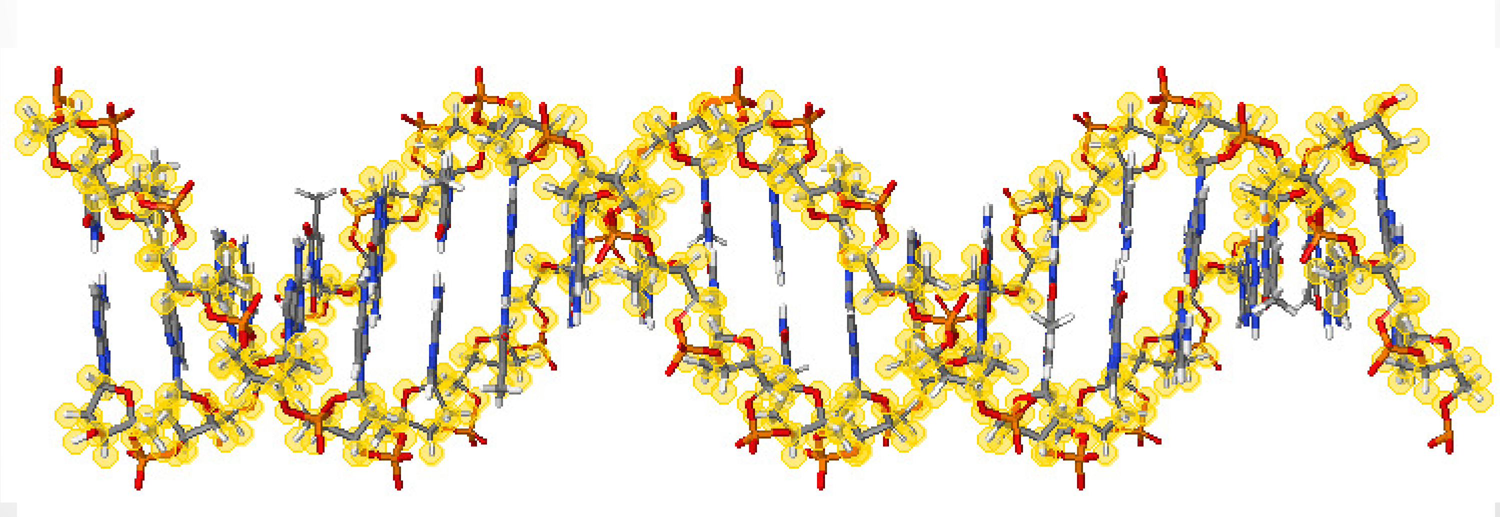
- Different levels of representation efficiency
  - The latter two can be translated (to each other and) to the first, but doing so may introduce an infinite, or anyway *extremely large*, number of species.

But what about Execution?

- Chemistry is not easily executable
  - Please Mr Chemist, execute me these reactions that I just made up.
- Description
  - Molecular languages used in systems biology are descriptive (modeling) languages
- Compilation
  - How can we compile *arbitrary* molecular programs?
- Execution
  - How can we actually execute molecular languages? With real molecules?

DNA as an Engineering Material

- This is why DNA/RNA is important: it is programmable matter.
- Not the only one, in principle, but the only one for which we have a well-developed manufacturing technology.

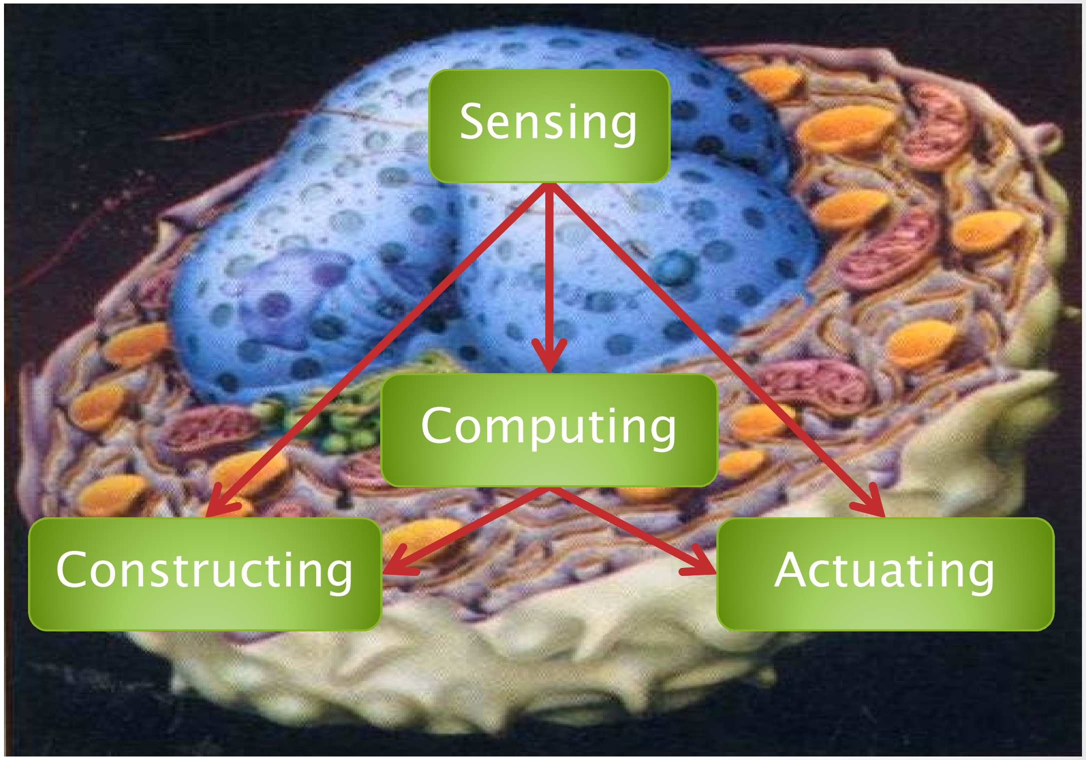


Sequence of Base Pairs (GACT alphabet)

Molecular Control Systems

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

Control Systems



Nucleic Acids can do all this.

And interface to biology.

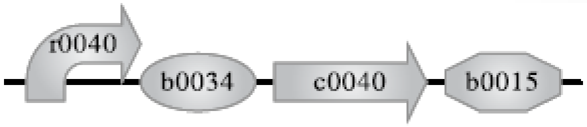
“Embedded” DNA Computing

(Synthetic Biology)

- Using bacterial machinery (e.g.) as the hardware. Using embedded gene networks as the software.

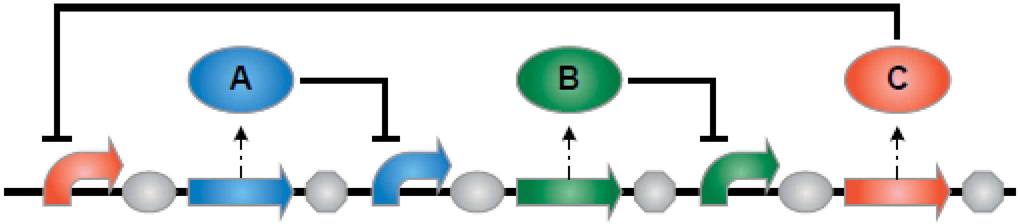
- MIT Registry of Standard Biological Parts

- GenoCAD
  - ○ Meaningful sequences [Cai et al.]



```
r0040:prom; b0034:rbs; c0040:pcr; b0015:ter
```

- GEC
  - ○ [Pedersen & Phillips]



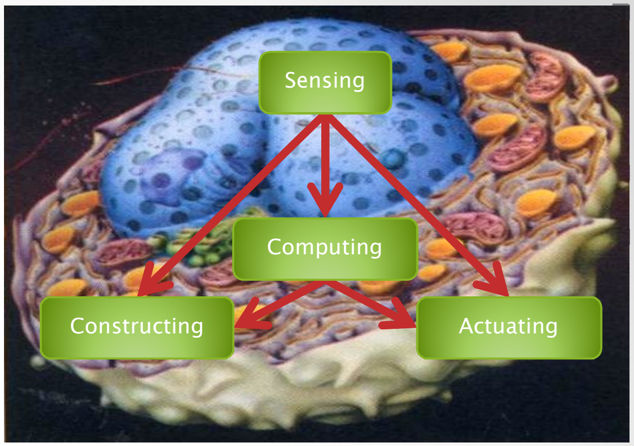
```
prom<neg(C)>; rbs; pcr<codes(A)>; ter;
prom<neg(A)>; rbs; pcr<codes(B)>; ter;
prom<neg(B)>; rbs; pcr<codes(C)>; ter
```

“Autonomous” DNA Computing

(Nano-engineering with biological materials)

- Mix & go
  - All (or most) parts are synthesized
  - No manual cycling (cf. early DNA computing)
  - In some cases, all parts are made of DNA (no enzyme/proteins)
- Self-assembled and self-powered
  - Can run on its own (e.g. environmental sensing)
  - Or be embedded into organisms (in the future)

Curing



A doctor in each cell

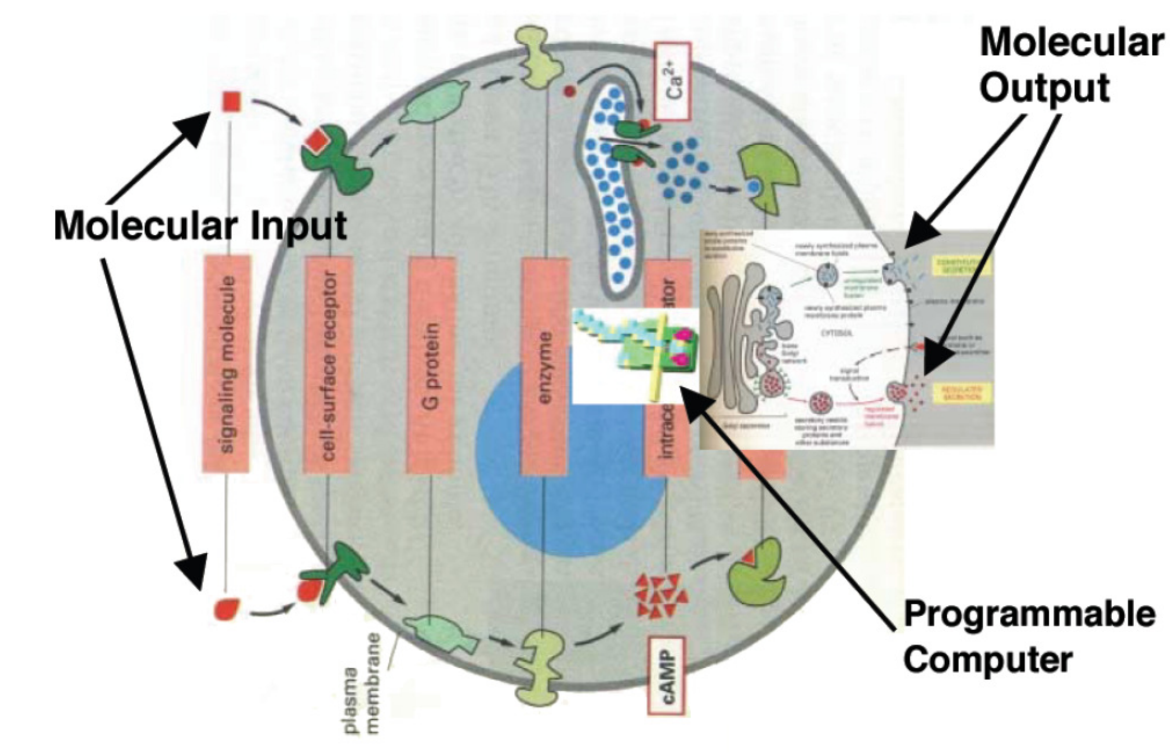


Fig. 1 Medicine in 2050: "Doctor in a Cell"

Ehud Shapiro

Rivka Adar
Kobi Benenson
Gregory Linshitz
Aviv Regev
William Silverman

Molecules and computation

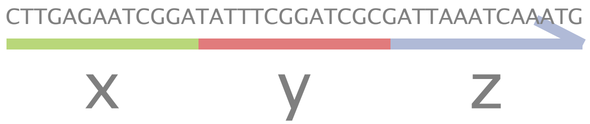
Modern DNA Computing

- Non-goals
  - Not to solve NP-complete problems.
  - Not to replace electronic computers.
  - Not necessarily using genes or to producing proteins.

- For general ‘molecular programming’
  - To precisely control the organization and dynamics of matter and information at the molecular level.
  - To interact algorithmically with biological entities.

Domains

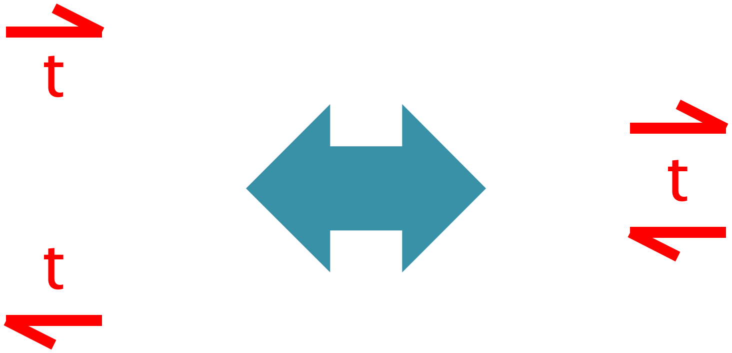
- Subsequences on a DNA strand are called domains. *PROVIDED* they are "independent" of each other.



- 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.

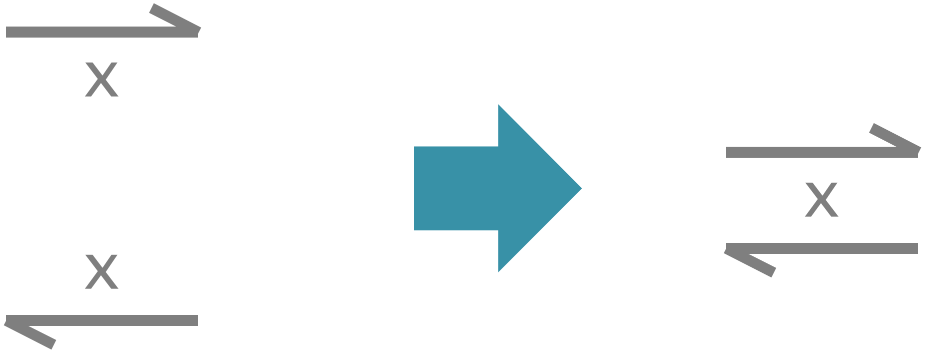
- Choosing domains (subsequences) that are suitably independent is a tricky issue that is still somewhat of an open problem (with a vast literature). But it can work in practice.

Short Domains



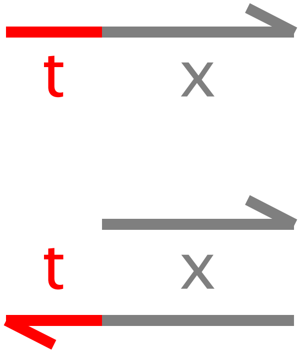
Reversible Hybridization

Long Domains



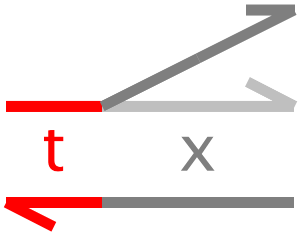
Irreversible Hybridization

Strand Displacement



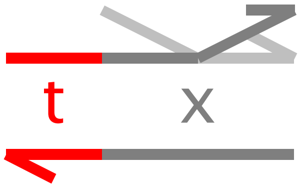
“Toehold Mediated”

Strand Displacement



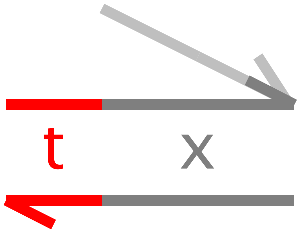
Toehold Binding

Strand Displacement



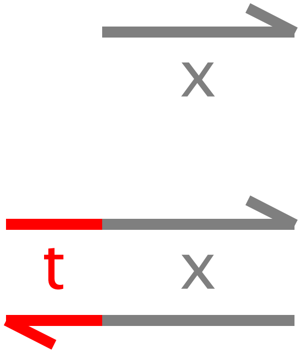
Branch Migration

Strand Displacement



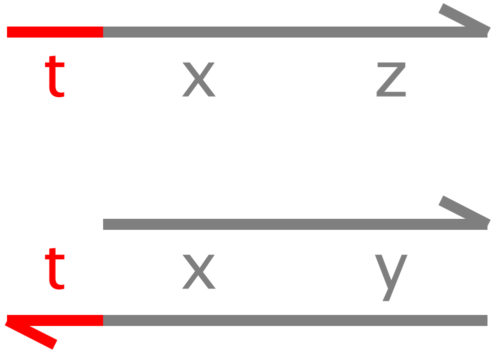
Displacement

Strand Displacement

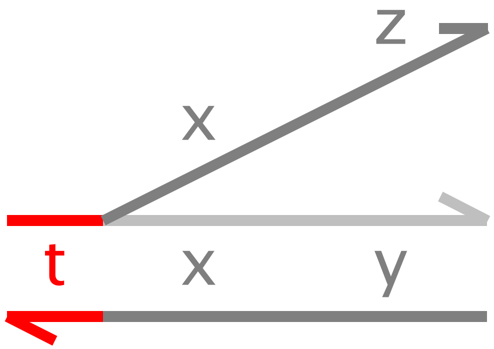


Irreversible release

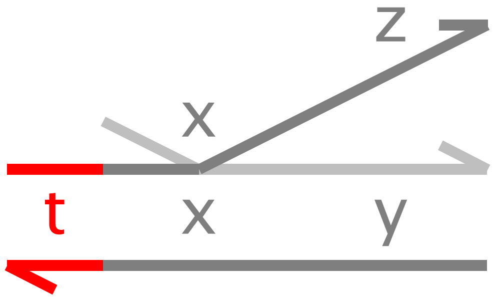
Bad Match



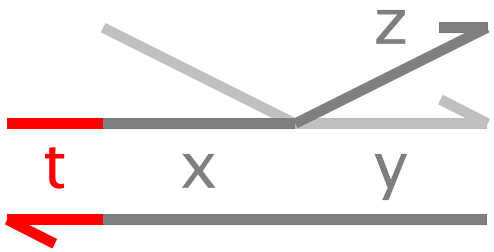
Bad Match



Bad Match



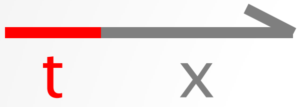
Bad Match



Cannot proceed
Hence will undo

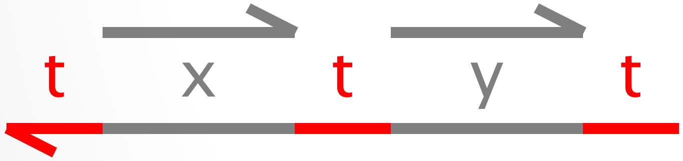
Two-Domain Architecture

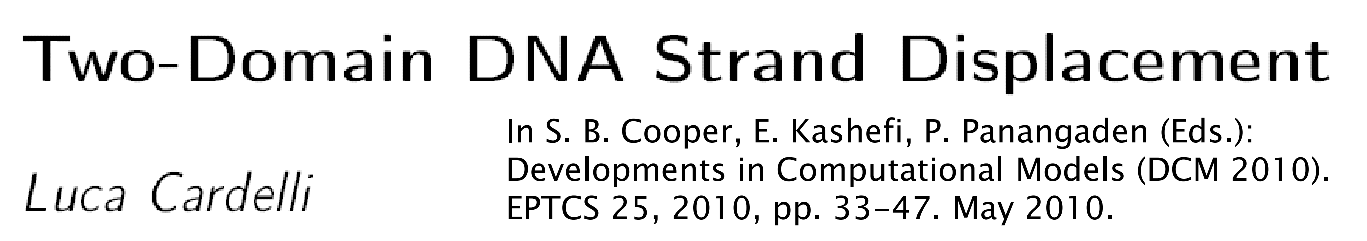
- Signals: 1 toehold + 1 recognition region



Garbage collection
“built into” the gates

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



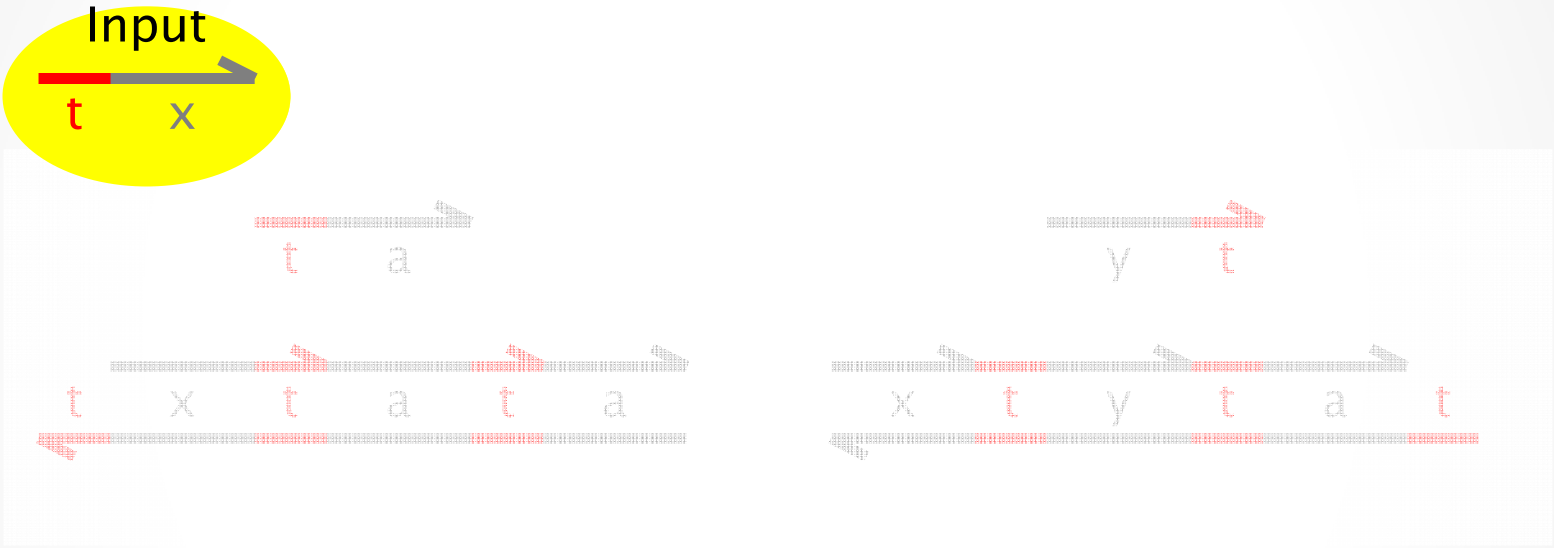


Two-Domain DNA Strand Displacement

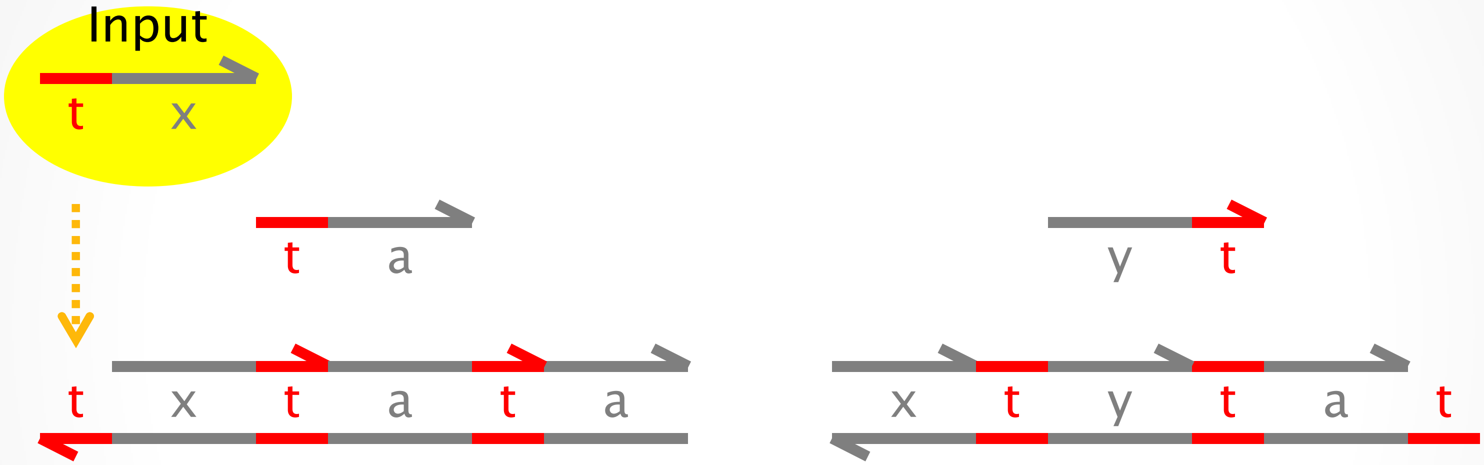
Luca Cardelli

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



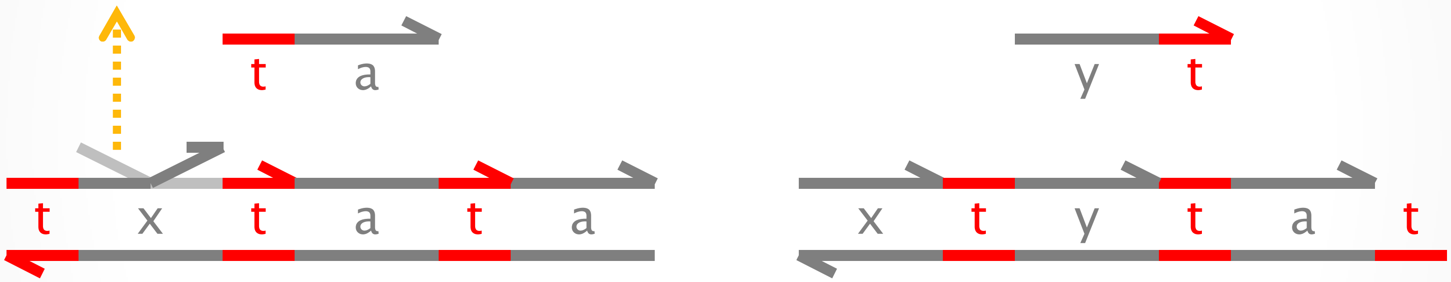
Transducer x→y



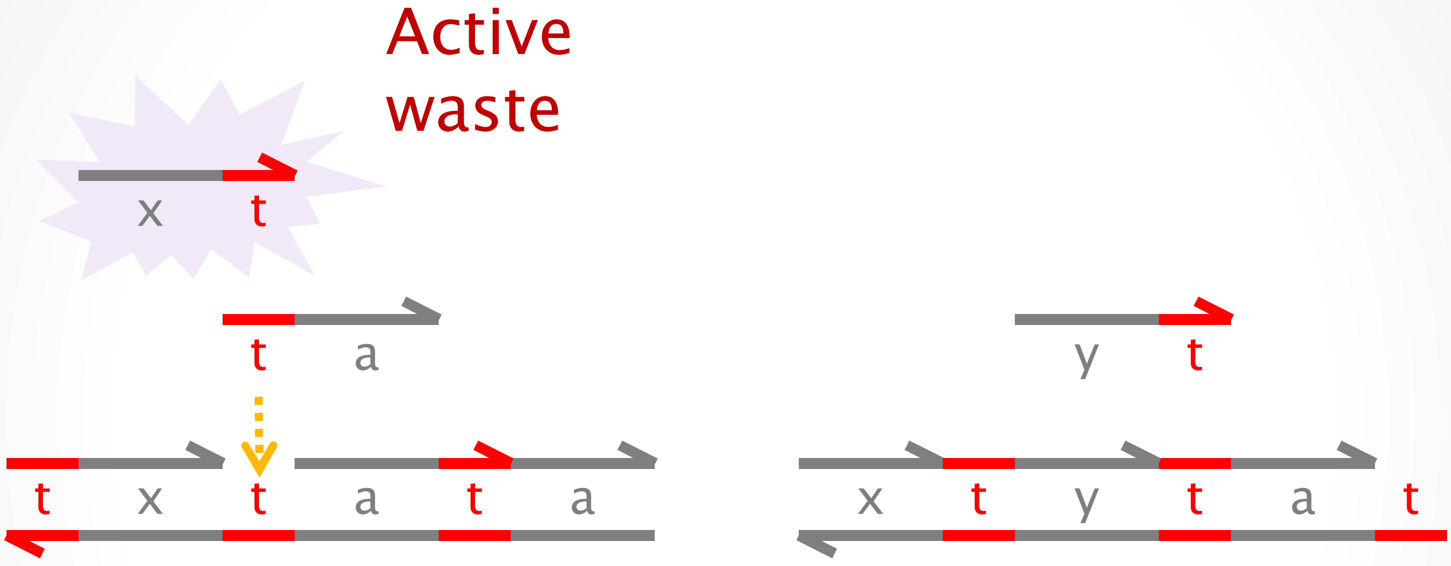
Built by self-assembly!

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

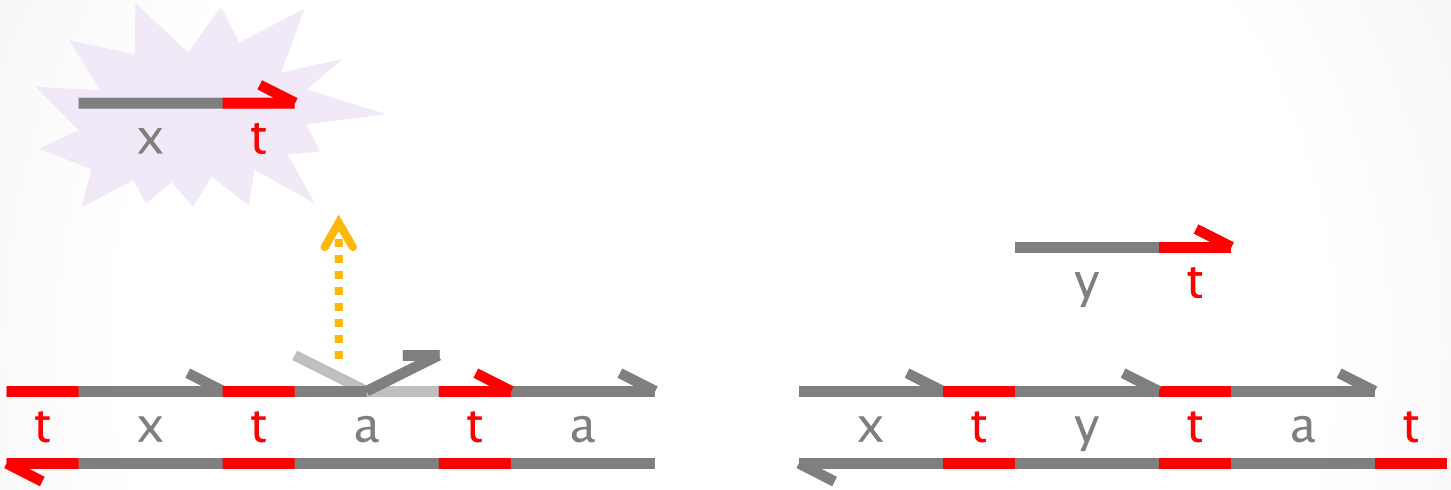
Transducer x→y



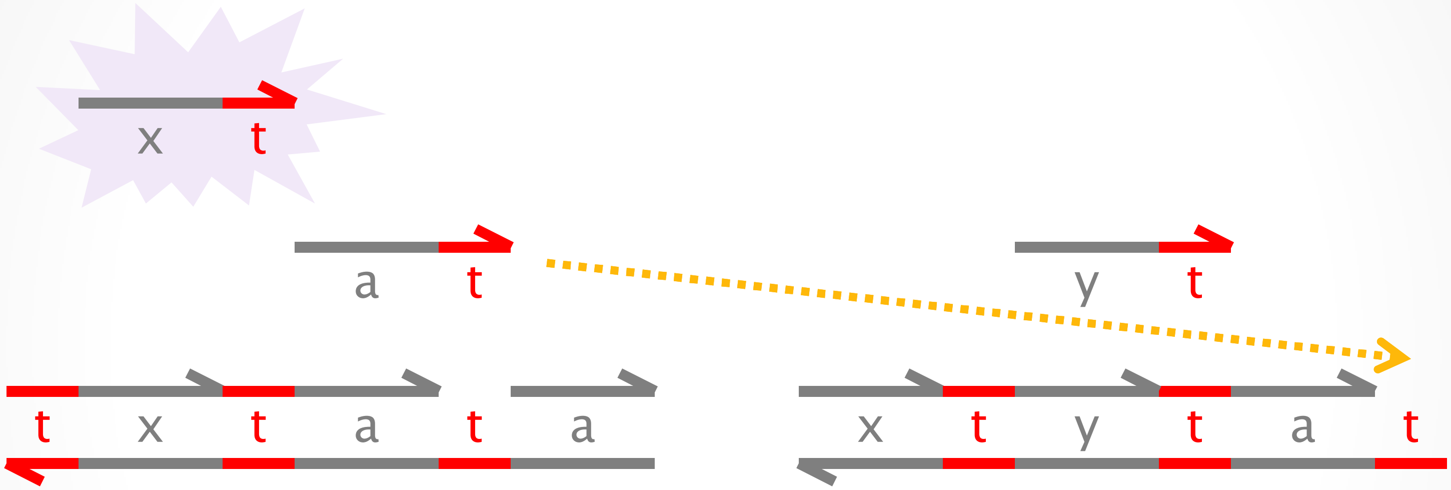
Transducer x→y



Transducer x→y

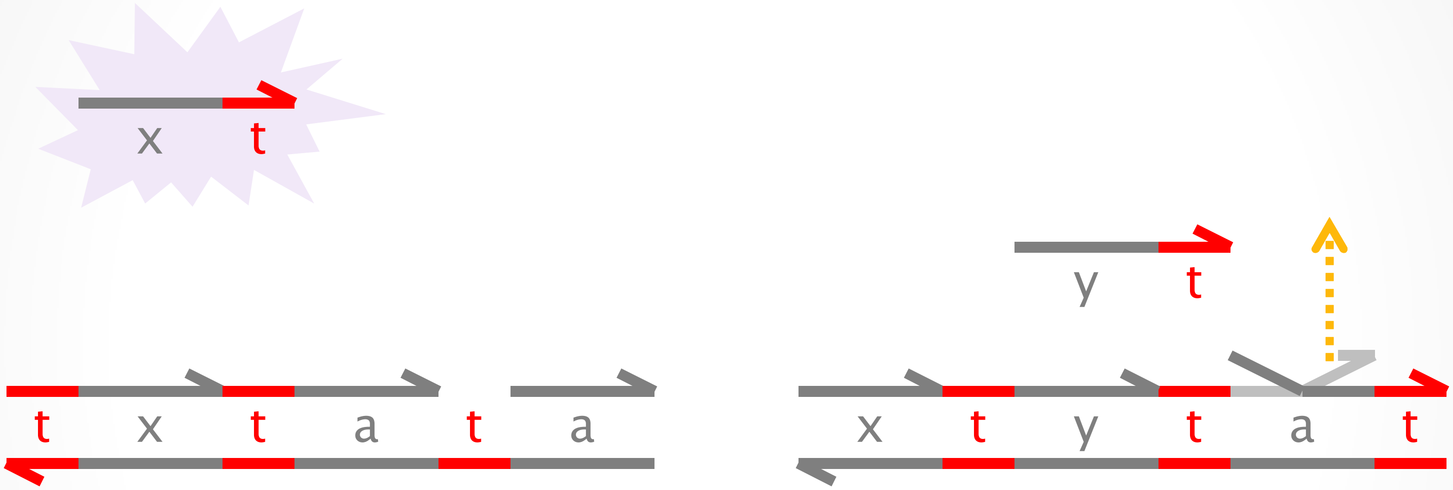


Transducer x→y

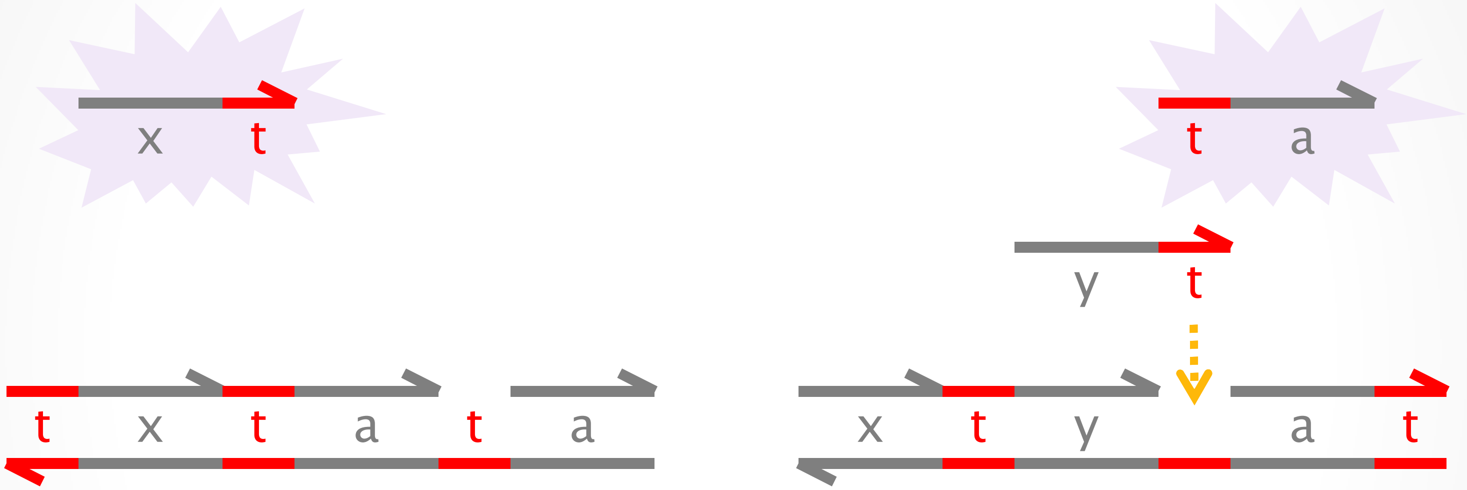


So far, a tx *signal* has produced an at *cosignal.*
But we want signals as output, not cosignals.

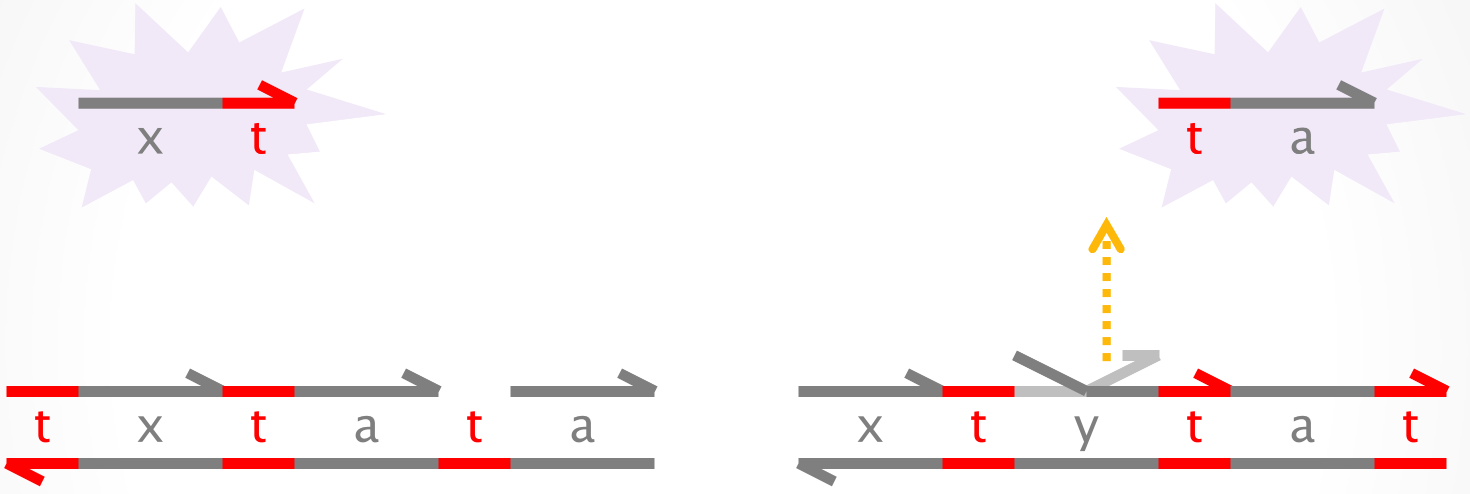
Transducer x→y



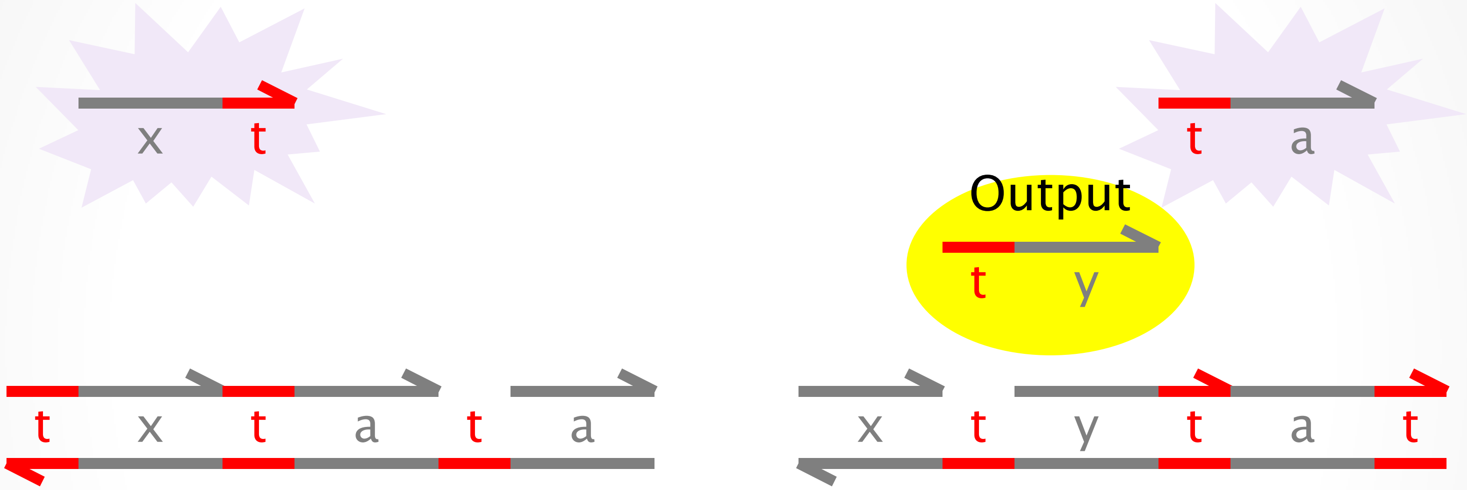
Transducer x→y



Transducer x→y



Transducer x→y



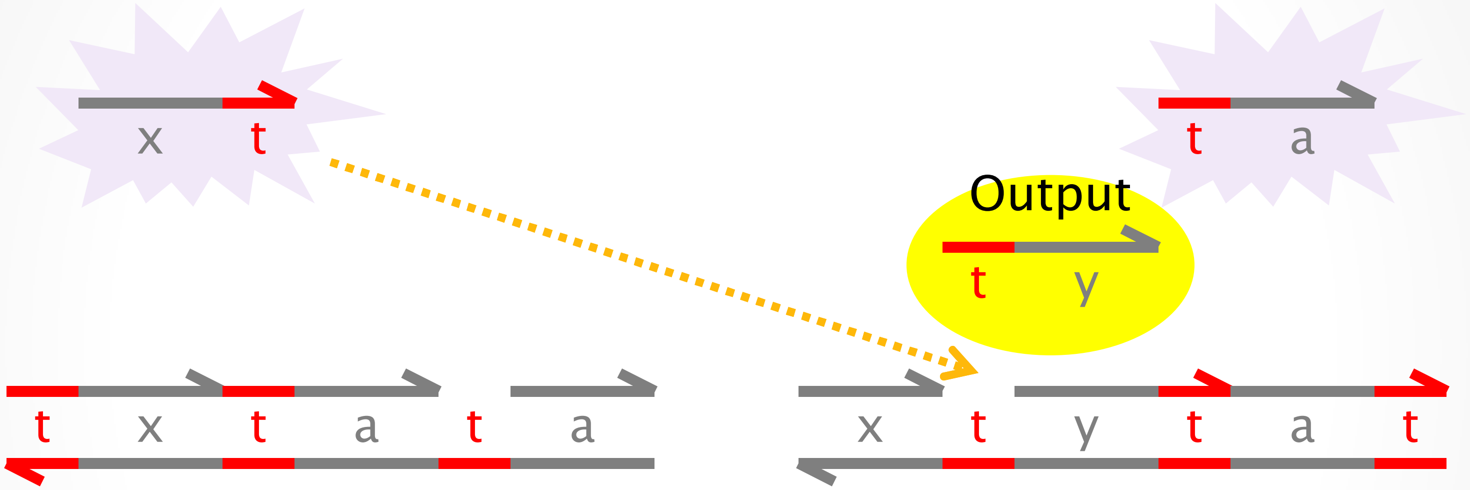
Here is our output ty *signal*.

But we are not done yet:

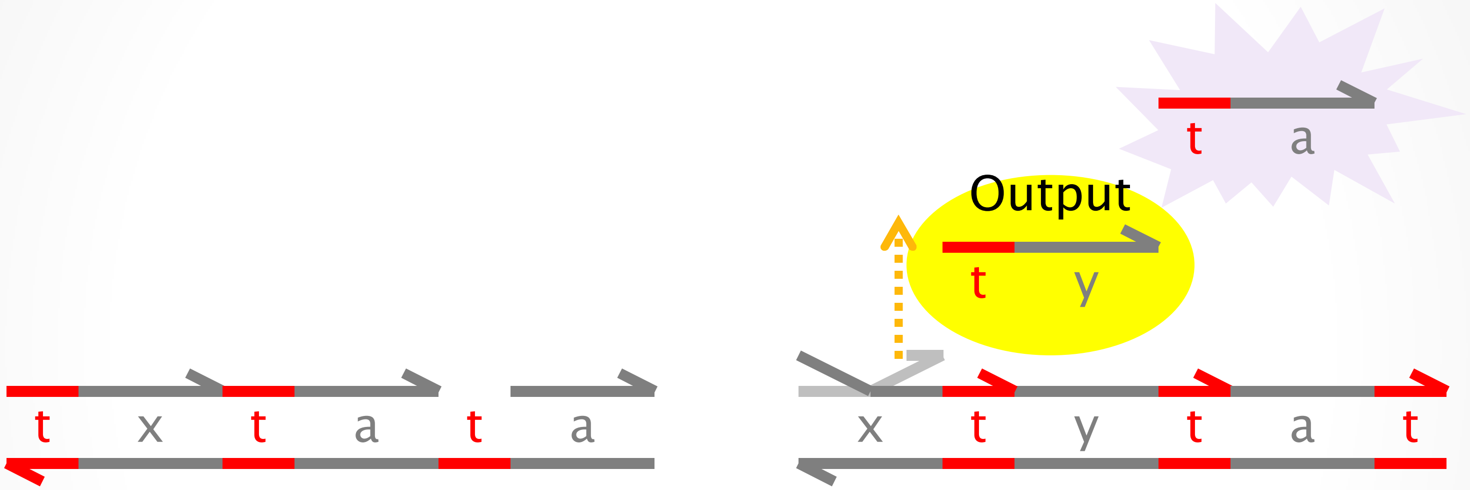
1) We need to make the output irreversible.
2) We need to remove the garbage.

We can use (2) to achieve (1).

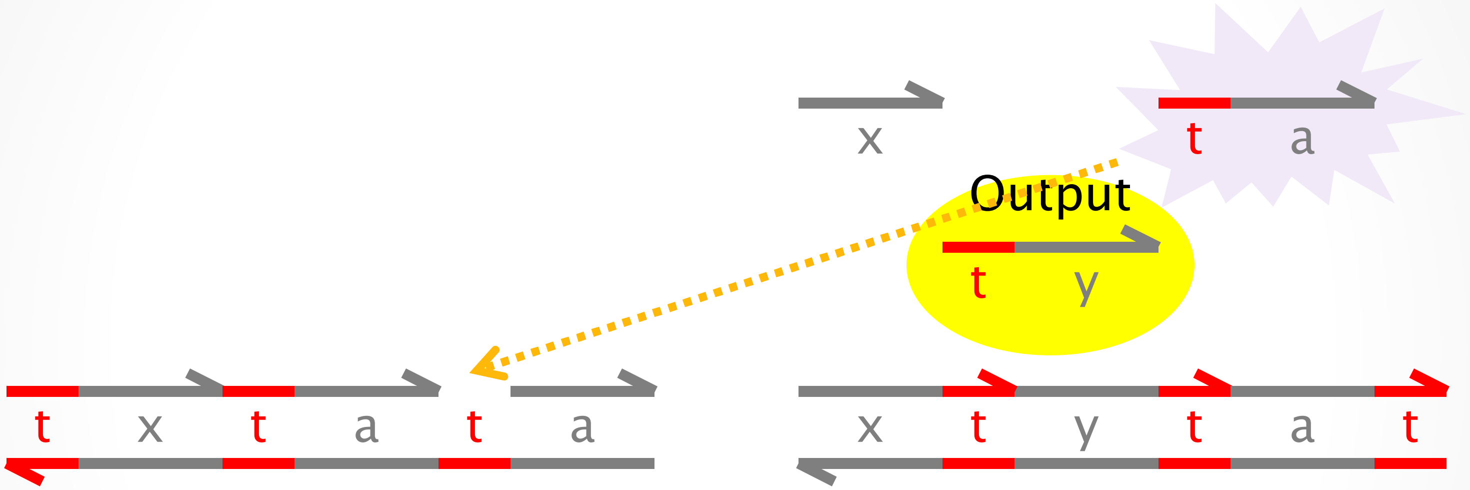
Transducer x→y



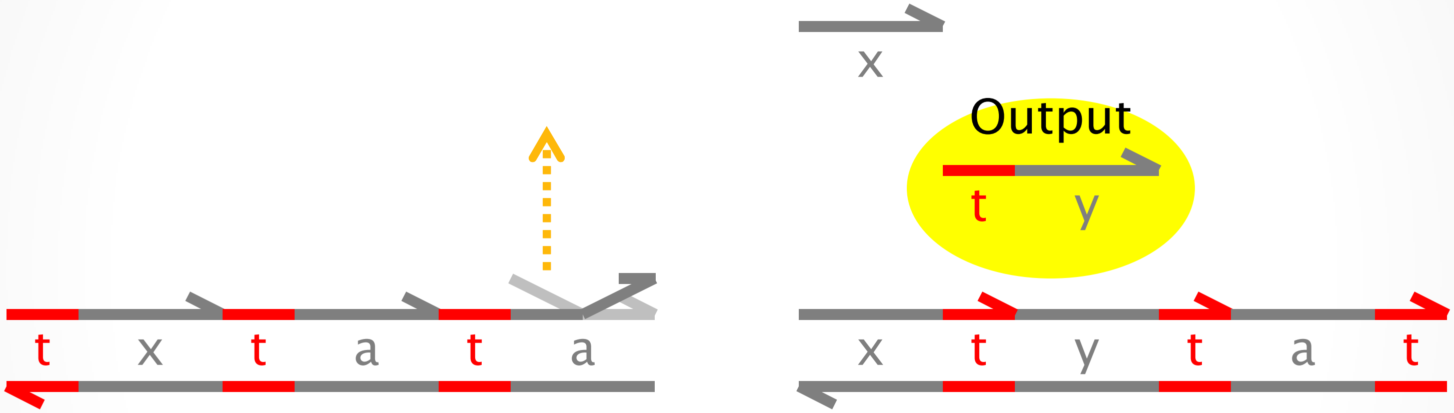
Transducer x→y



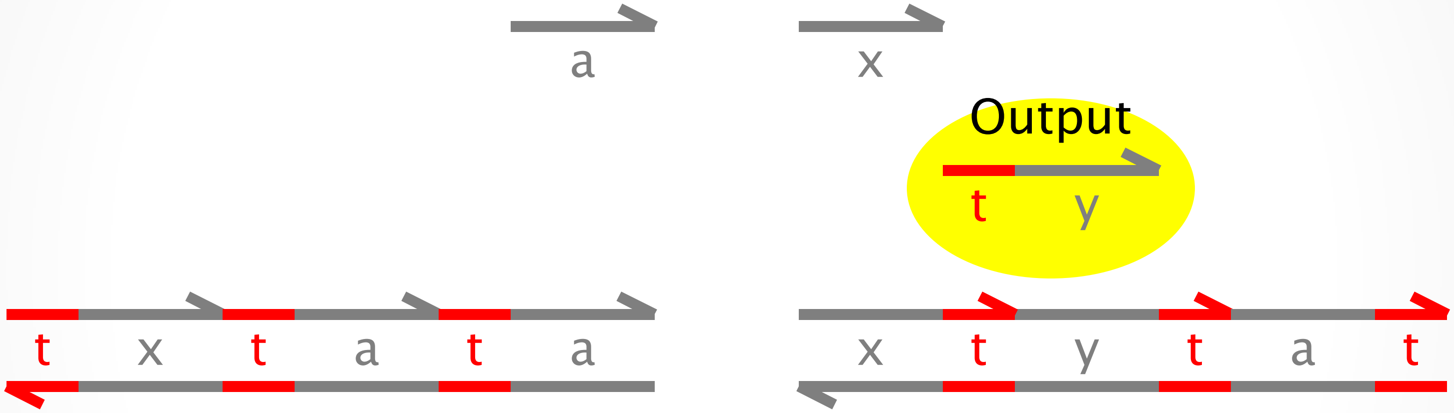
Transducer x→y



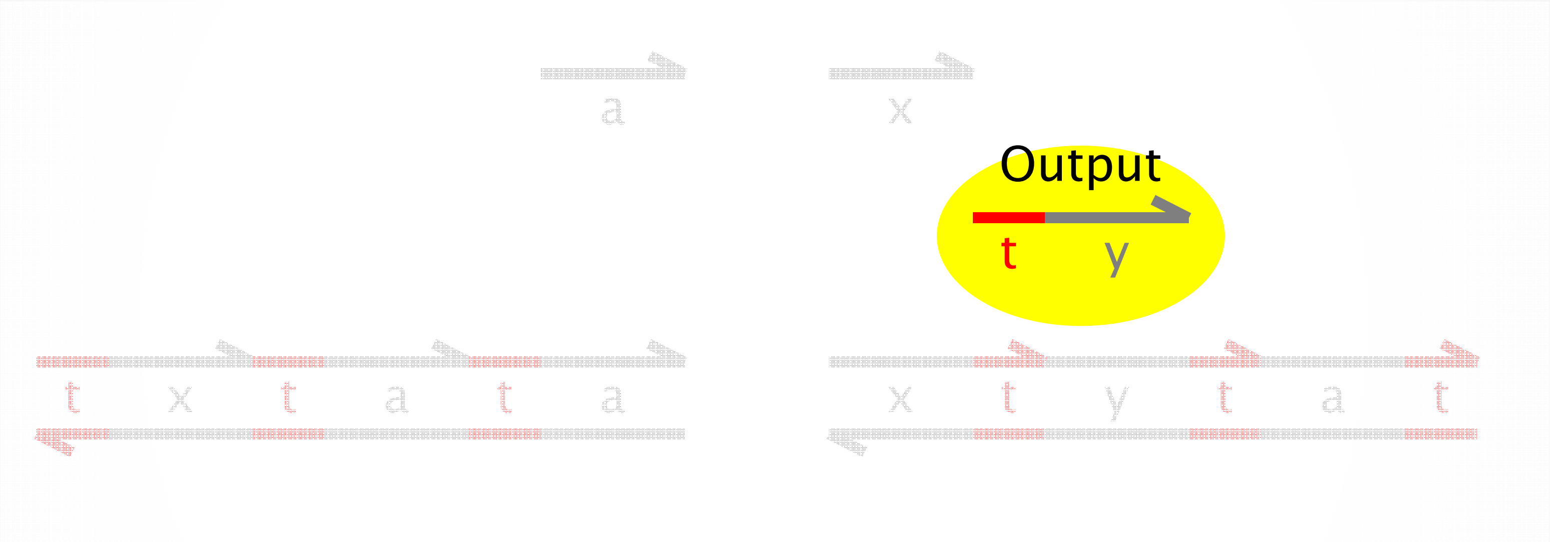
Transducer x→y



Transducer x→y



Transducer x→y



Done.

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

General n×m Join-Fork

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

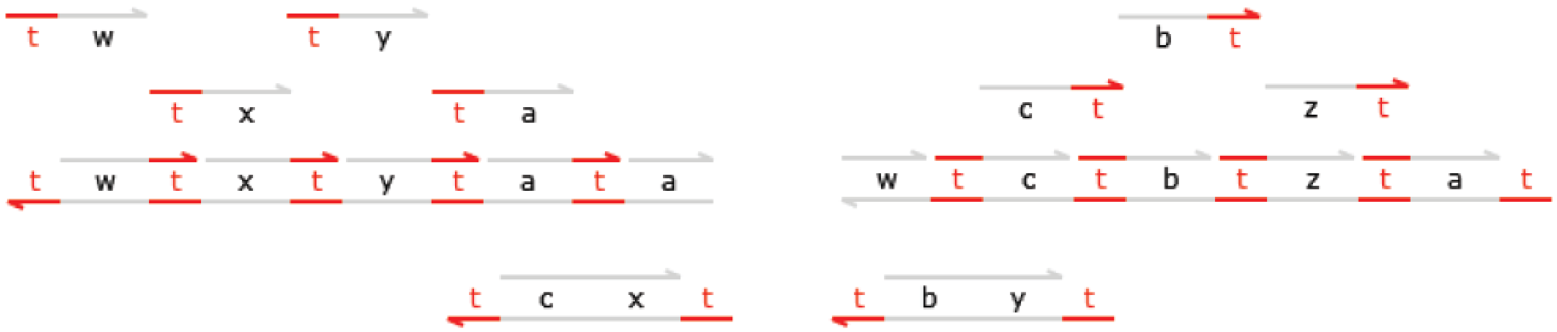
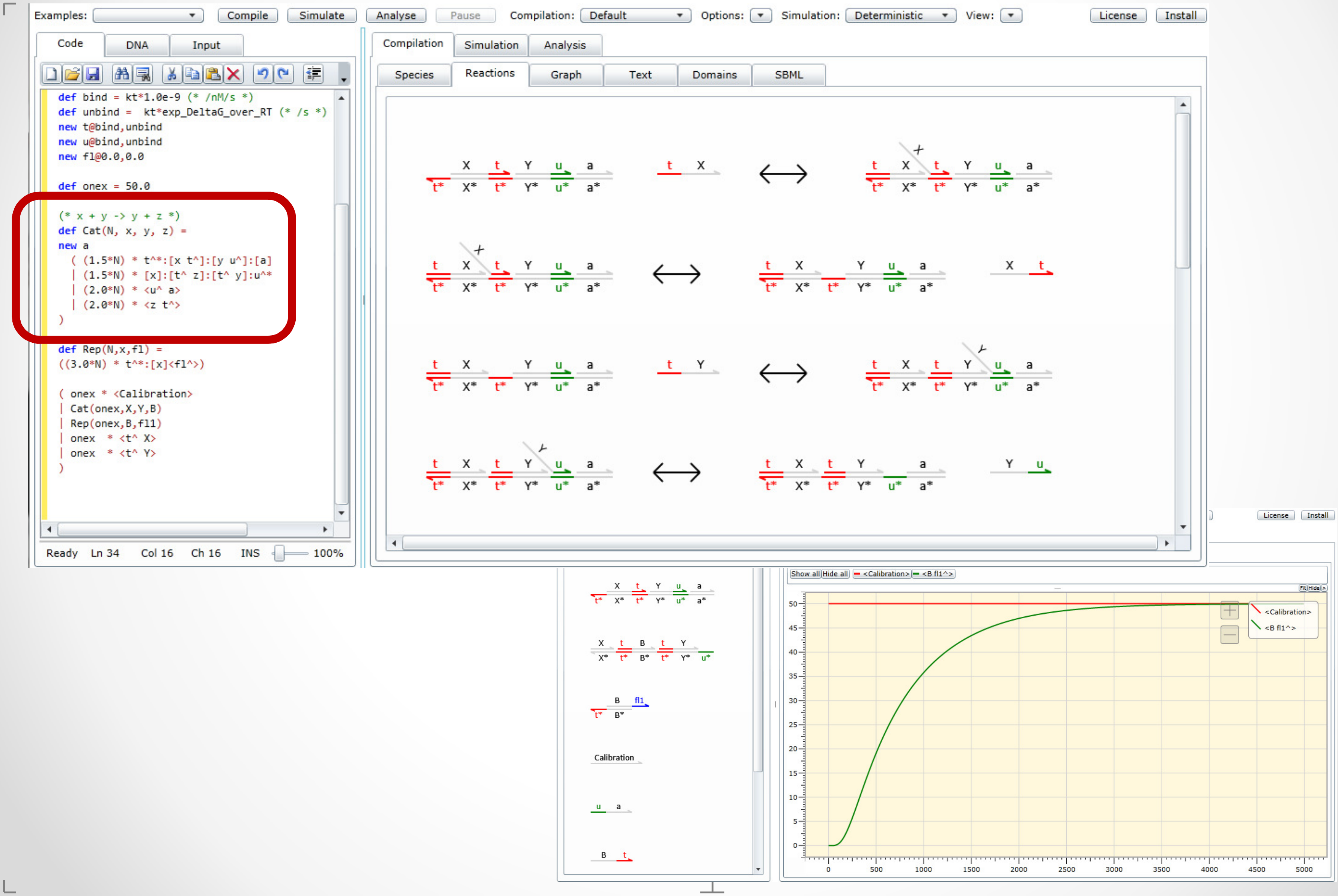


Figure 9: 3-Join $J_{wxyz} \mid tw \mid tx \mid ty \rightarrow tz$: initial state plus inputs tw, tx, ty.

DNA Programming



Compilation: Default

Simulation: Deterministic

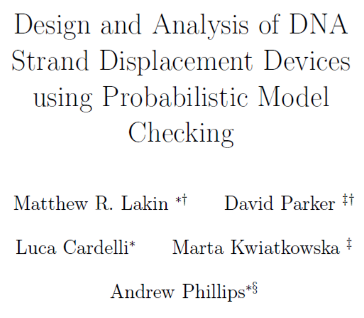
```
def bind = kt*1.0e-9 (* /nM/s *)
def unbind =  kt*exp_DeltaG_over_RT (* /s *)
new t@bind,unbind
new u@bind,unbind
new fl@0.0,0.0

def onex = 50.0

(* x + y -> y + z *)
def Cat(N, x, y, z) =
new a
  ( (1.5*N) * t^*:[x t^]:[y u^]:[a]
  | (1.5*N) * [x]:[t^ z]:[t^ y]:u^*
  | (2.0*N) * <u^ a>
  | (2.0*N) * <z t^>
)

def Rep(N,x,fl) =
((3.0*N) * t^*:[x]<fl^>)

( onex * <Calibration>
| Cat(onex,X,Y,B)
| Rep(onex,B,fl1)
| onex  * <t^ X>
| onex  * <t^ Y>
)
```

Ready   Ln 34   Col 16   Ch 16   INS   100%

Debugging

- Big Networks
  - Two-domain DNA gates for 1 Approximate Majority switch.
  - Initial species: 17
  - Total number of species: 85 (including run-time produced ones)
  - Total number of reactions: 104

- Analysis
  - Gate correctness
  - Circuit correctness
  - Compiler correctness
  - Currently, by simulation
  - Increasingly, by modelchecking:



Design and Analysis of DNA Strand Displacement Devices using Probabilistic Model Checking

Matthew R. Lakin *† David Parker ‡‡

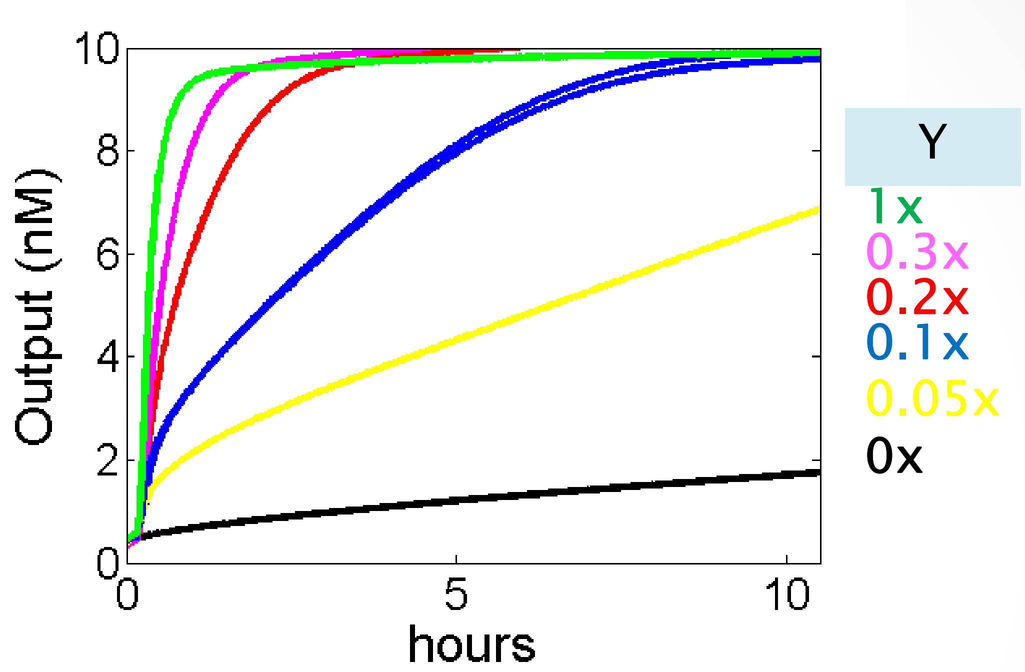
Luca Cardelli* Marta Kwiatkowska ‡

Andrew Phillips*§

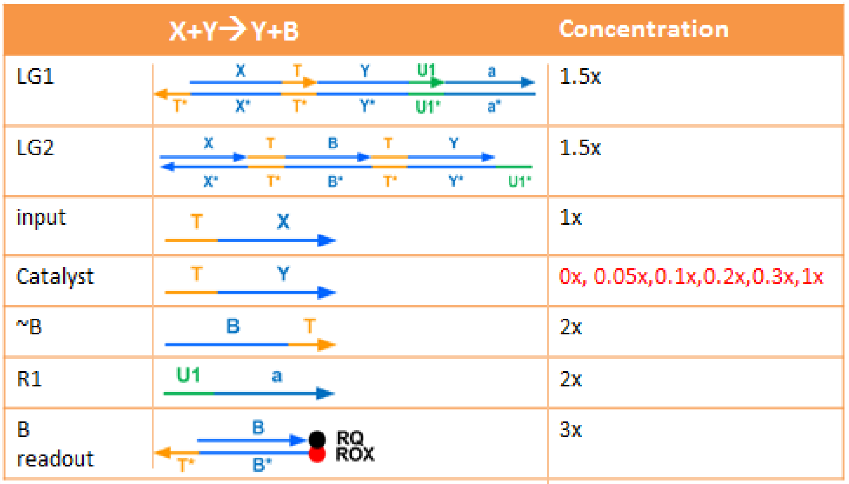
Experiments

Two-domain gate
for X+Y → Y+B

X+Y→Y+B
35C
1x = 50nM



Yuan-Jyue Chen and Georg Seelig
U.Washingon.



	X+Y→Y+B	Concentration
LG1	X T Y U1 a T* X* T* Y* U1* a*	1.5x
LG2	X T B T Y X* T* B* T* Y* U1*	1.5x
input	T X	1x
Catalyst	T Y	0x, 0.05x,0.1x,0.2x,0.3x,1x
~B	B T	2x
R1	U1 a	2x
B readout	B RQ T* B* ROX	3x

Summary

- Executable chemistry
  - Given an arbitrary finite chemical network, compile it systematically and execute it.
    [D. Soloveichik, G. Seelig, E. Winfree. DNA as a Universal Substrate for Chemical Kinetics. PNAS 107 no. 12, 5393-5398, 2010.]
  - Finite chemical networks have the computing power of (stochastic) Petri Nets. Population protocols (such as AM) are also well-characterized. [D.Angluin, J.Aspnes, D.Eisenstat, E.Ruppert: The Computational Power of Population Protocols].
- Executable bio-chemistry
  - In addition, DNA supports polymerization, which gives the computing power of Turing Machines.
  - Then the programming language cannot be just chemical reactions, but has to be something more like process algebra or term-rewriting systems.

Conclusions

Much to be done

- Systems Biology
  - Develop the algorithmic understanding of molecular networks that will allow us to understand their structure and function (and how to do it better).

- Synthetic Biology
  - Develop the materials and technology that will allow us to 'code-up' arbitrary molecular networks.
  - Develop the quantitative techniques that will allow us to 'debug' them.

Acknowledgments

- Microsoft Research
  - Andrew Phillips
- Caltech
  - Winfree Lab
- U.Washington
  - Seelig Lab
- CoSBi
  - Attila Csikász-Nagy

Challenges

Verification

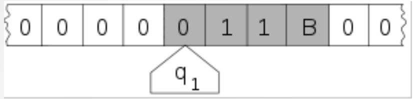
- Environment
  - The nano-environment is messy (stochastic noise, failures, etc.)
  - But we should al least ensure our designs are *logically correct*

- 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

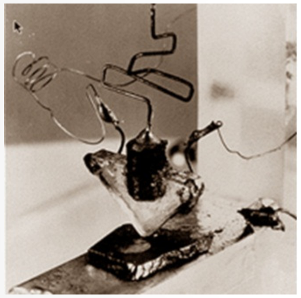
- Verifying Populations
  - Gates come in (large) populations
  - Each population *shares private domains* (technologically unavoidable)
  - Correctness of populations means proofs with large state spaces

A Brief History of DNA

Turing Machine, 1936

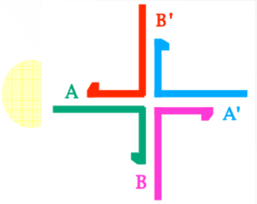


Transistor, 1947

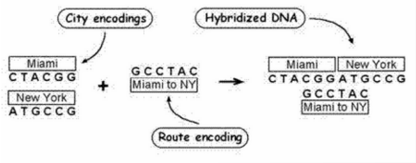


DNA, –3,800,000,000

Structural DNA, 1982



DNA Algorithm, 1994



~~Digital Computers~~

Computer programming

systematic manipulation of information

20th century

systematic manipulation of matter

21th century

~~DNA Computers~~

Molecular programming

Conclusions

Conclusions

- A vast literature on cell cycle switching
  - Ferrell et.al., Novak-Tyson et.al., etc.
    Mostly ODE based analysis, plus noise
  - Many bistable transitions have different implementations in different cell cycle phases and organisms (phosphorylation, enzymes, synthesis/degradation, etc.)
  - We focused on a mechanism that can only be seen stochastically (quick majority switching with x=y)
- A range of ‘network transformation’
  - Can explain the structure of some natural networks
  - From some non-trivial underlying algorithms
  - Discovering the transformation can elucidate the structure and function of the networks
  - But how can we say that these transformations ‘preserve (essential) behavior’?