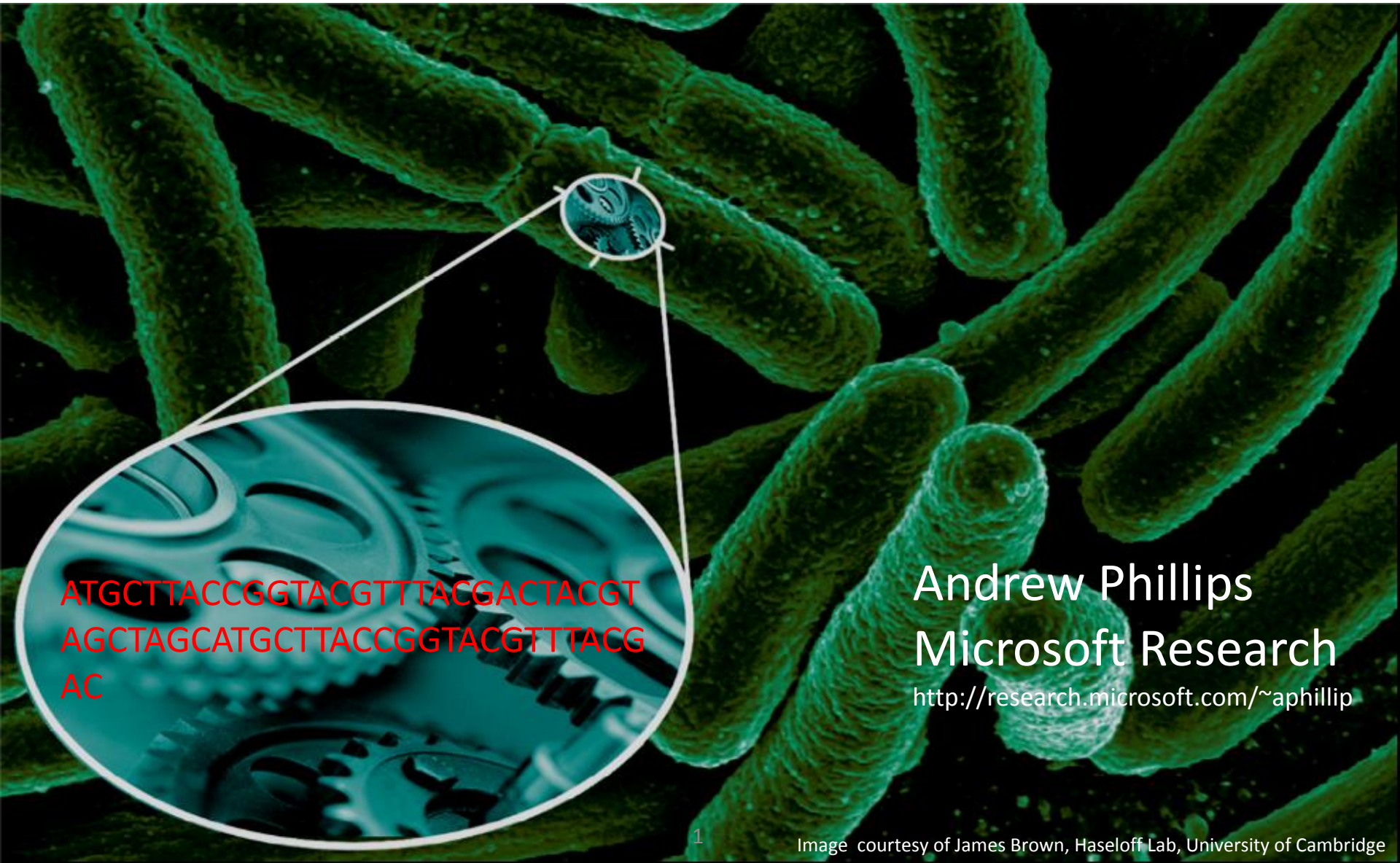
Programming Life



Andrew Phillips
Microsoft Research
http://research.microsoft.com/~aphillip

Image courtesy of James Brown, Haseloff Lab, University of Cambridge

Programming Cells in the 21st Century

Medicine

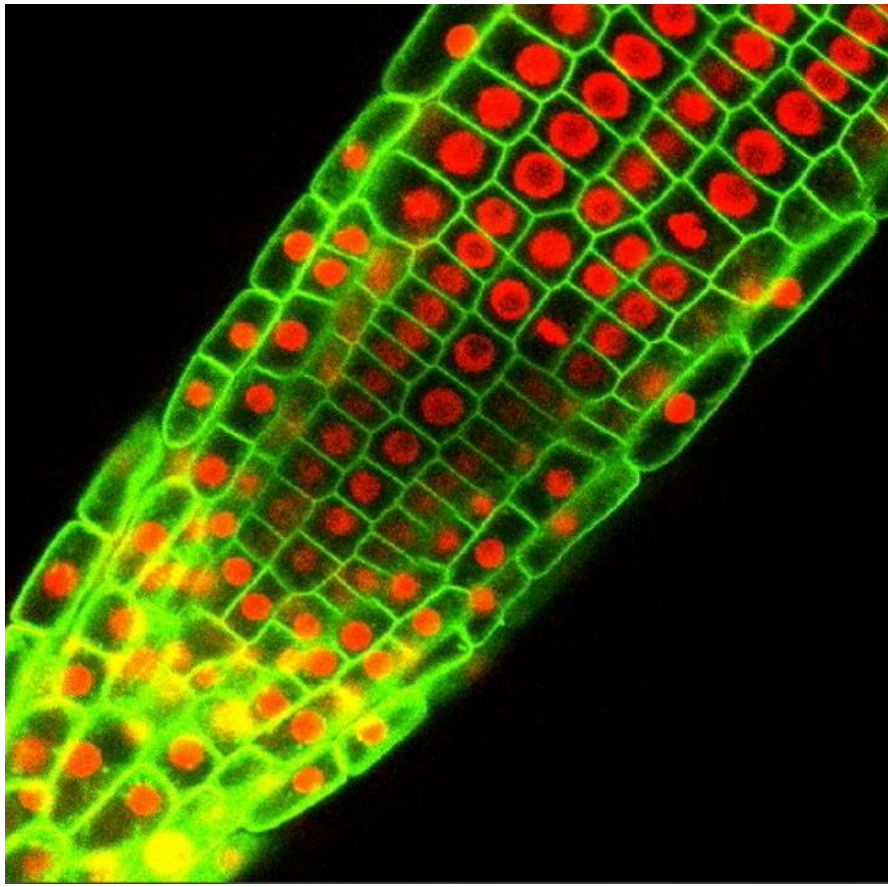
- Programming bacteria to fight tumours and viruses
- Programming yeast to synthesise novel vaccines
- Programming immune cells to improve immune response

Food

- Programming bacteria to fix nitrogen for plants
- Programming plant cells to improve crop yields

Energy & Environment

- Programming bacteria to convert CO_2 from the atmosphere into fuel



Video courtesy of Fernan Federici, University of Cambridge

Outline

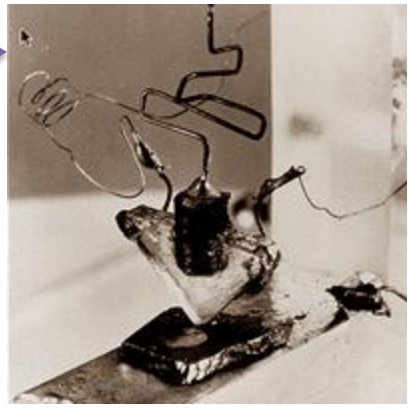
- Programming DNA Circuits
- Programming Genetic Devices
- Programming the Immune System

Programming DNA Circuits

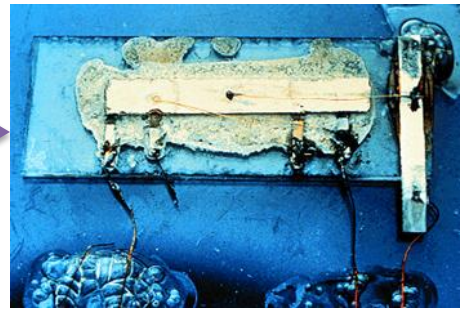
Luca Cardelli, Matthew Lakin,
Simon Youssef & Andrew Phillips

Smaller and Smaller

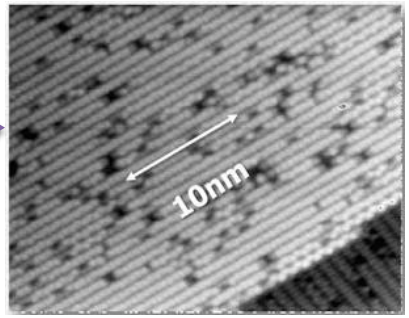
Dec. 23, 1947. John Bardeen, William Shockley, Walter Brattain show the first working transistor



Sep 1958. Jack Kilby builds the first integrated circuit.



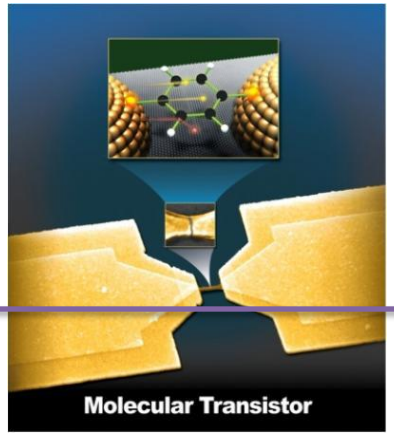
Jan 30, 2010. Intel and Micron announce 25nm NAND flash.



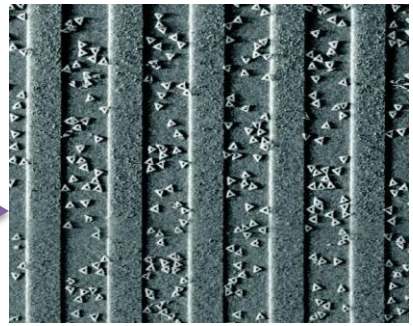
Scanning tunneling microscope image of a silicon surface showing 10nm is ~20 atoms across

Dec 24, 2009. Working transistor made of a single molecule.

Observation of molecular orbital gating. *Nature*, 2009; 462 (7276): 1039



The race is on for *molecular scale integrated circuits*.



Placement and orientation of individual DNA shapes on lithographically patterned surfaces. Nature Nanotechnology 4, 557 - 561 (2009).

DNA Storage

DNA in each human cell

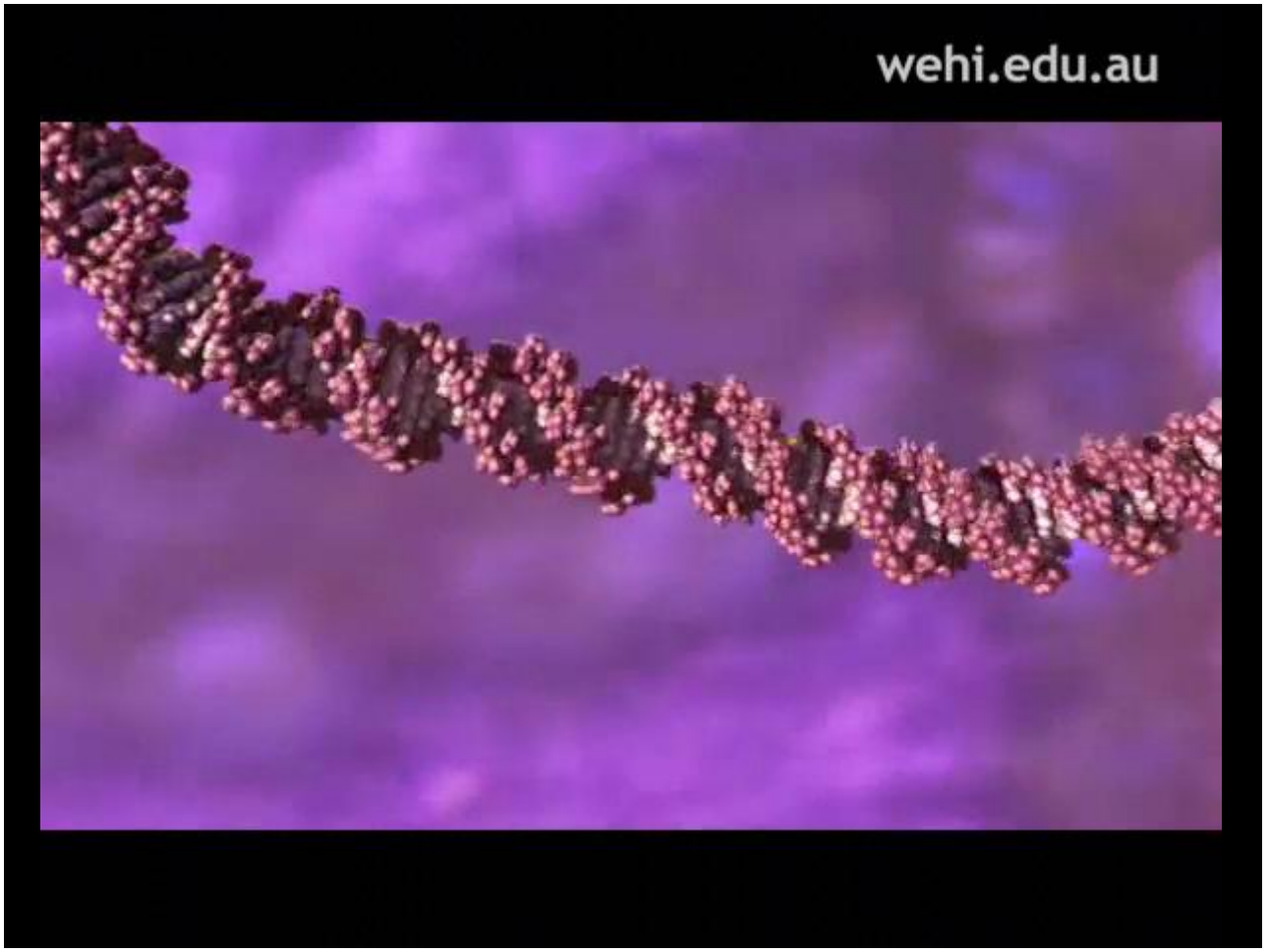
- 3 billion base pairs
- 2 meters long, 2nm thick
- folded into a 6nm ball
- 750 MegaBytes

DNA in human body

- 10 trillion cells
- 133 Astronomical Units
- 7.5 OctaBytes

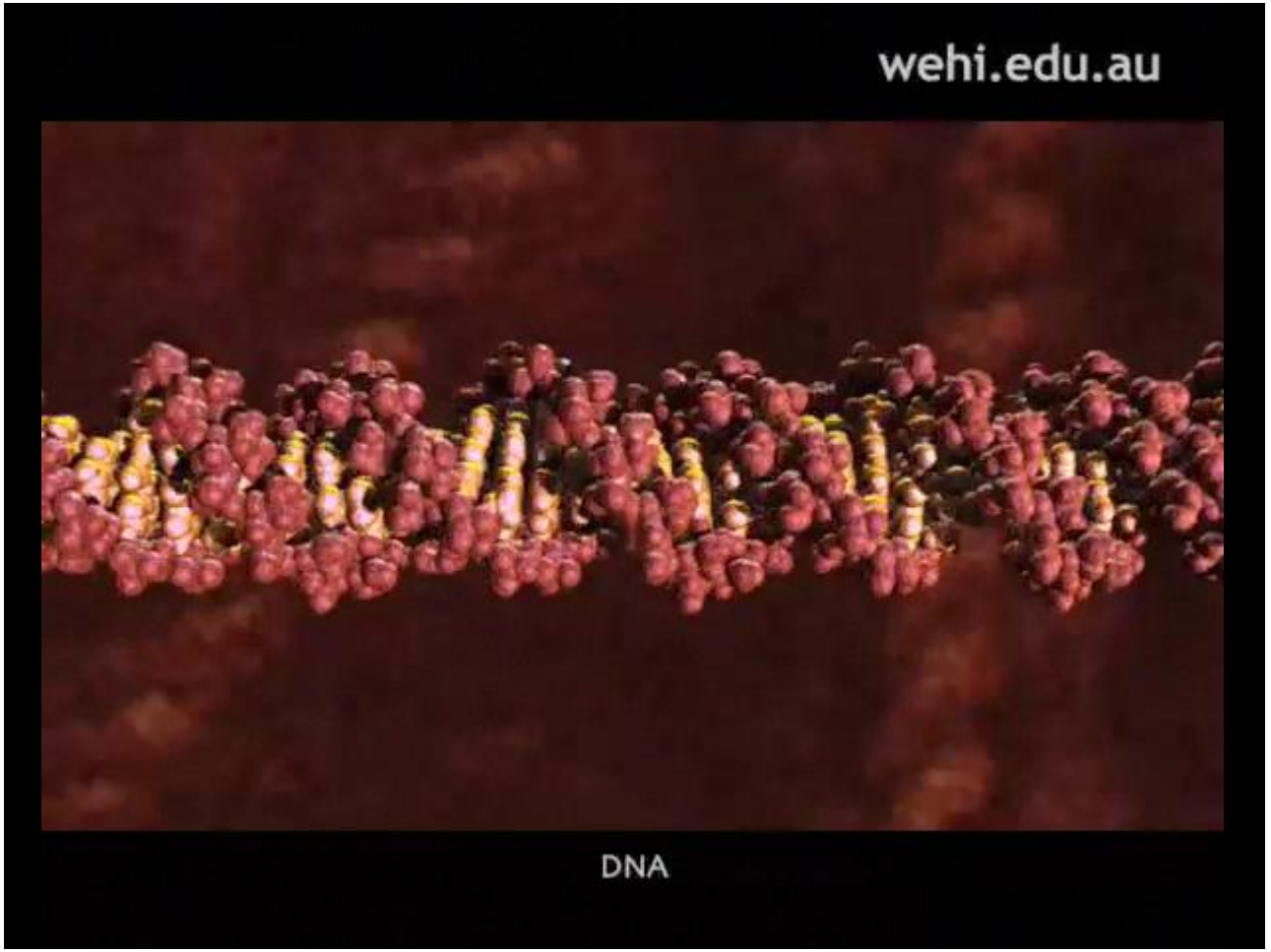
DNA in human population

- 20 million light years



Video by Drew Berry http://www.wehi.edu.au/wehi-tv

DNA Structure

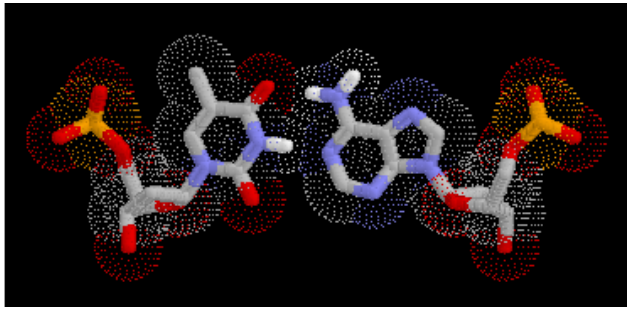


Video by Drew Berry http://www.wehi.edu.au/wehi-tv

DNA Sequence (T,A,G,C)

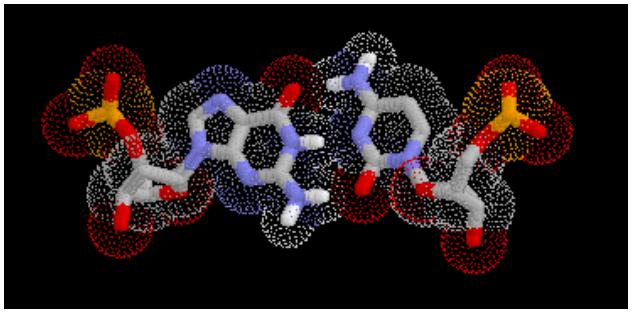
T-A Base Pair

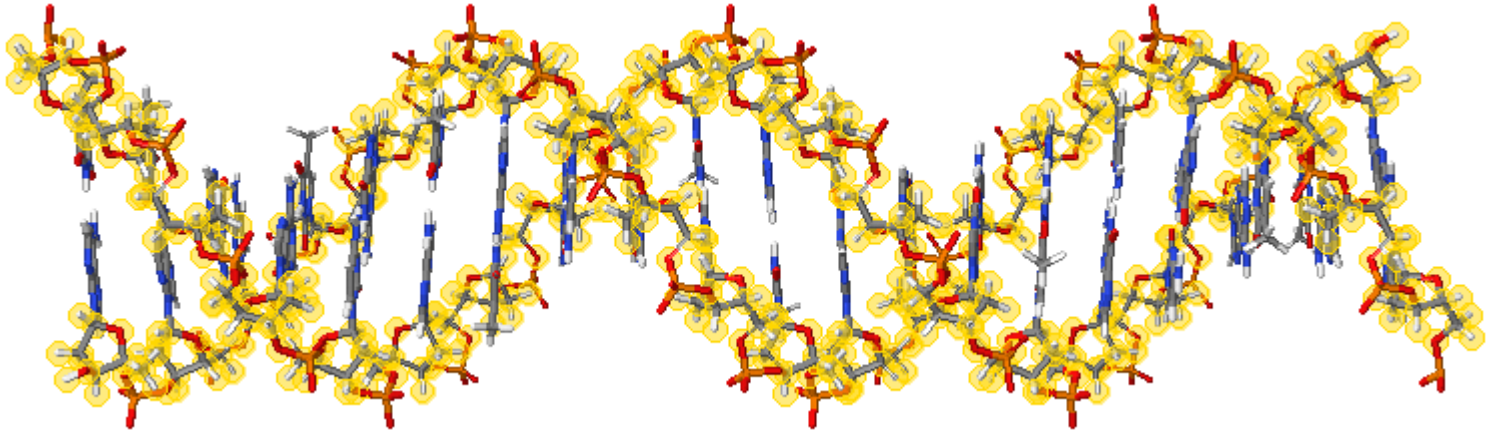
Thymine-Adenine



G-C Base Pair

Guanine-Cytosine



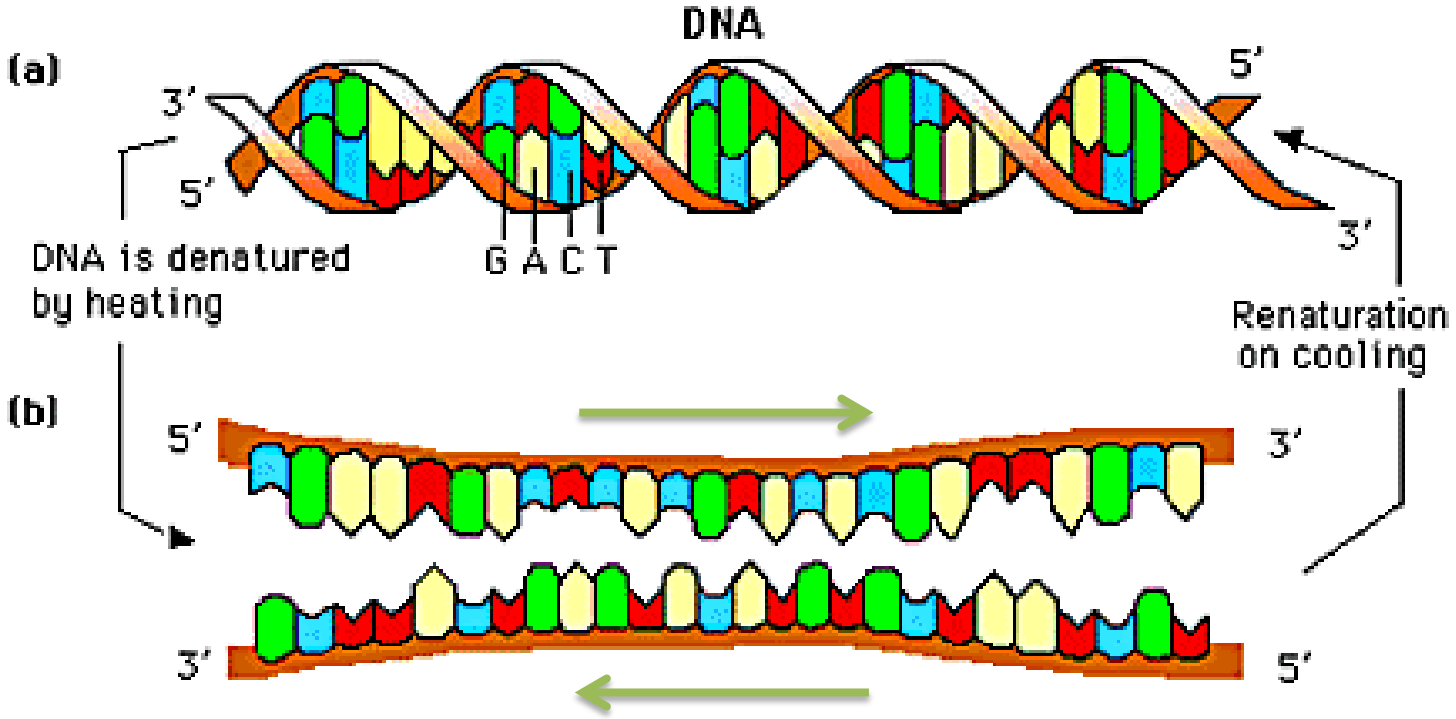


Sequence of Base Pairs (GACT alphabet)

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

DNA strands

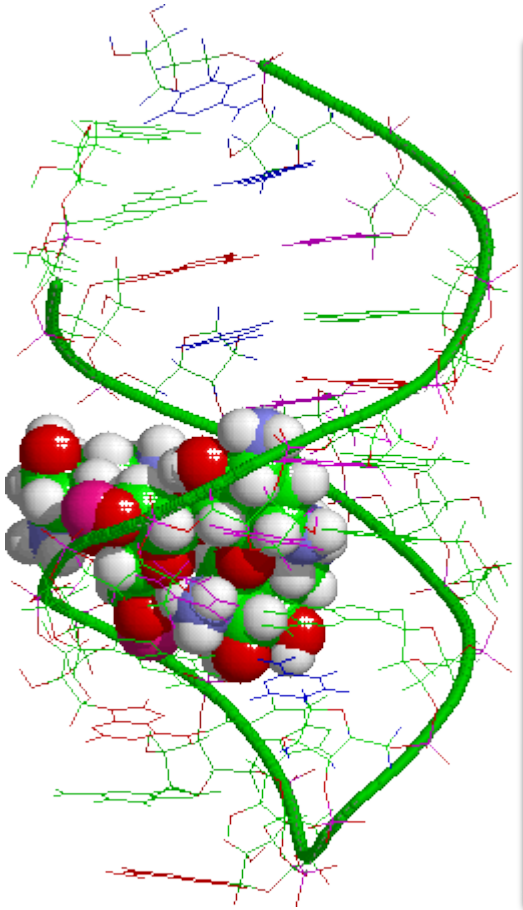
Double-stranded DNA



Single-stranded DNA has an orientation
Each strand spells a GACT sequence
The two strands have *opposite* orientations

DNA Aptamers

Artificially evolved DNA molecules that stick to anything you like (highly selectively).



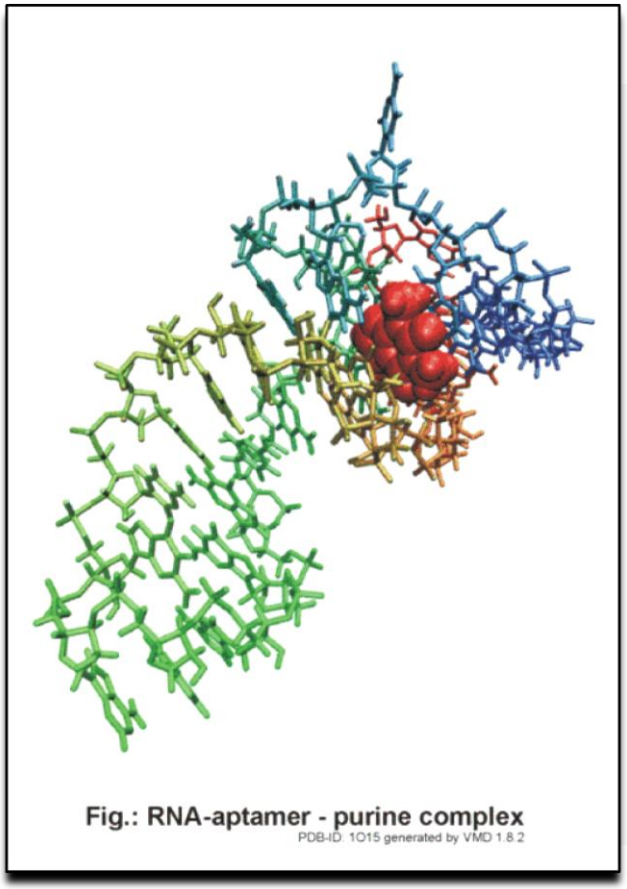
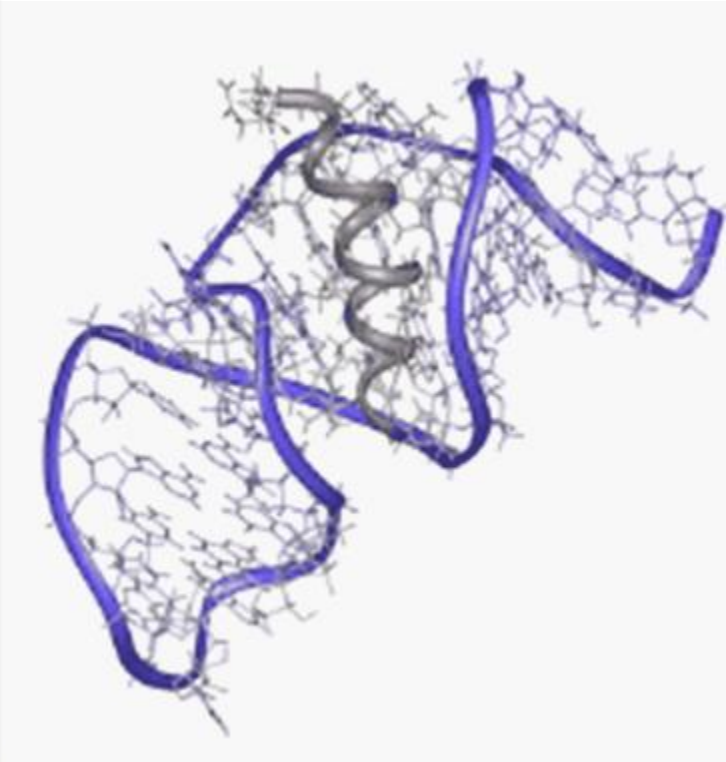


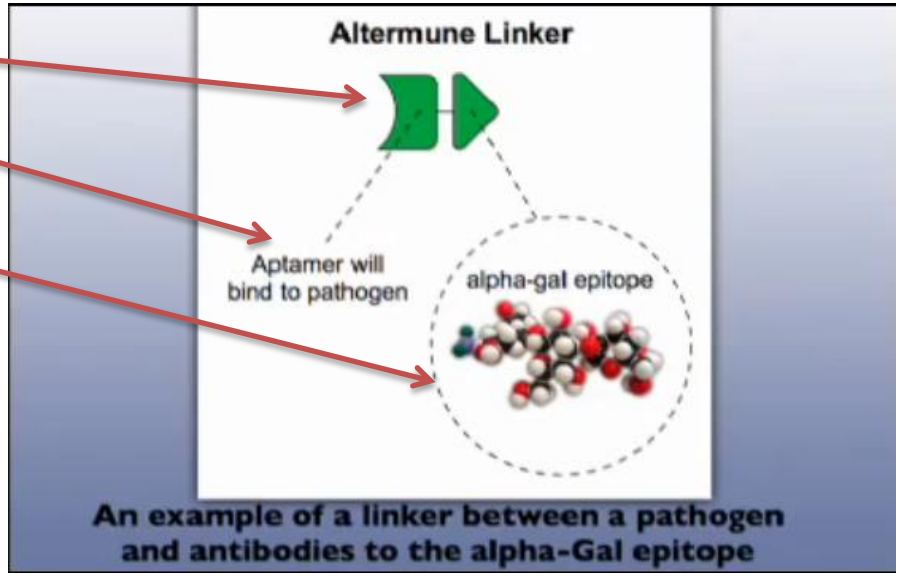
Fig.: RNA-aptamer - purine complex

PDB-ID: 1O15 generated by VMD 1.8.2

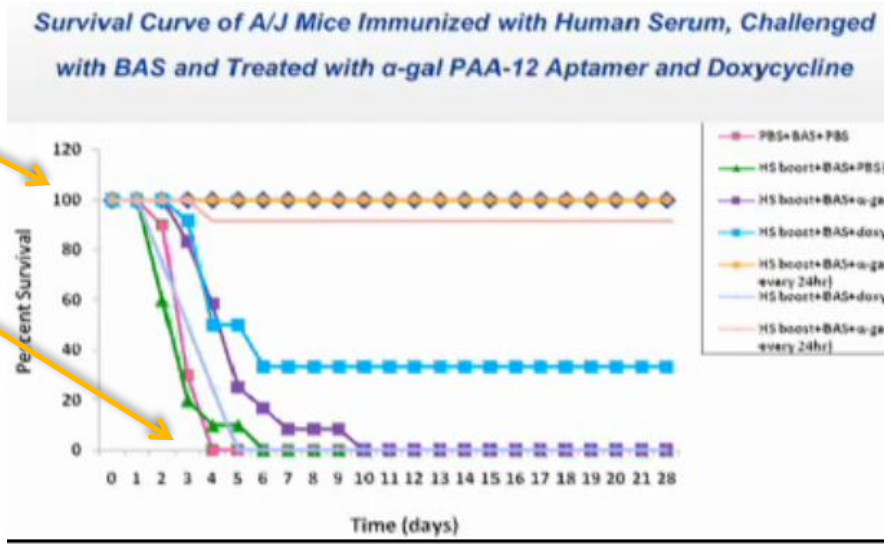


Aptamer DNA Drugs

- DNA aptamer binds to:
  - A) a pathogen
  - B) a molecule our immune system already hates and immediately removes (eats) along with anything attached to it



- Result: instant immunity
  - Mice poisoned with Anthrax plus aptamer (100% survival)
  - Mice poisoned with Anthrax (not so good)

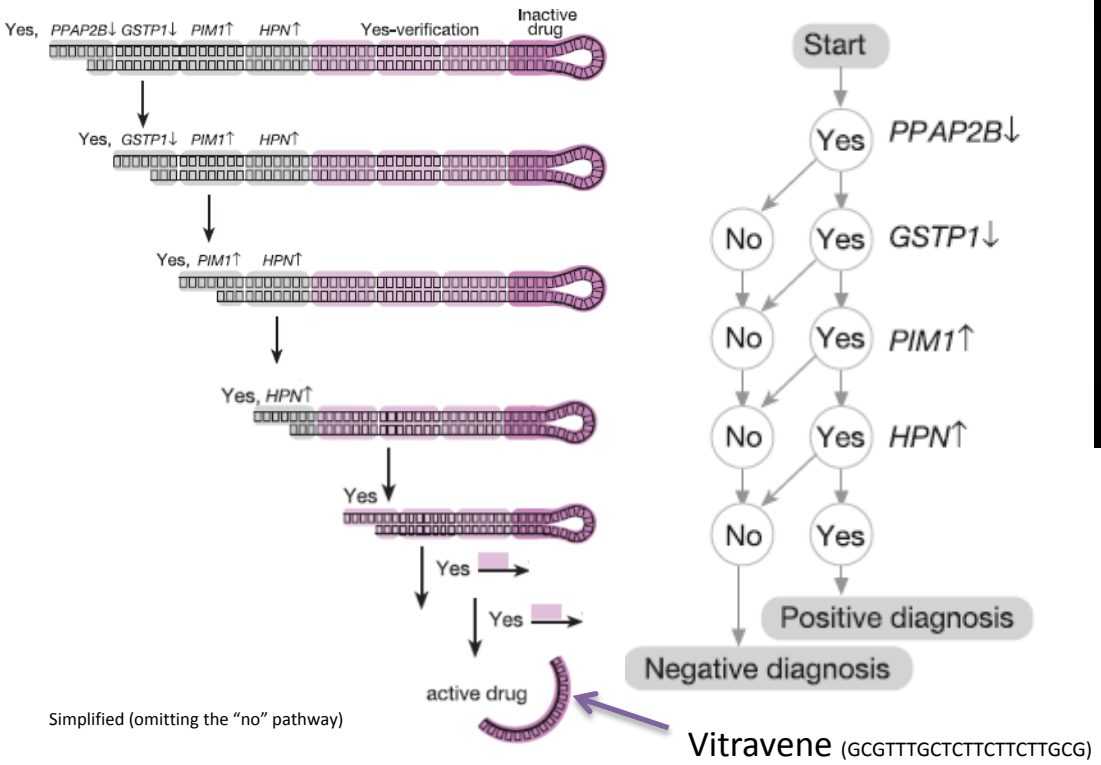


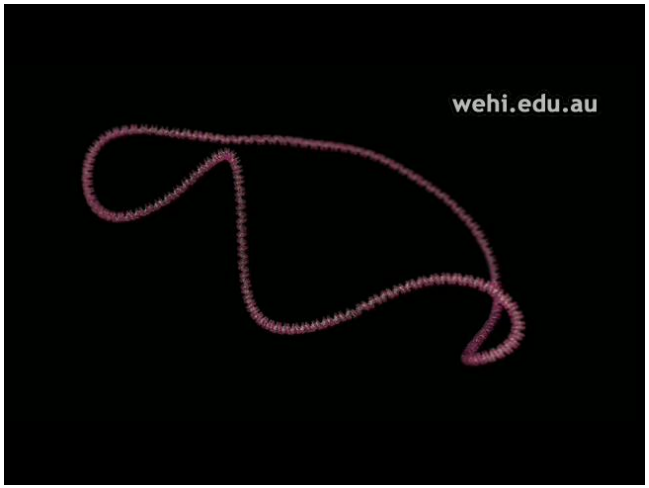
Kary Mullis (incidentally, also Nobel prize for inventing the Polymerase Chain Reaction)

Computational DNA Drugs

Perform logical computation before releasing drug

Uses restriction enzymes





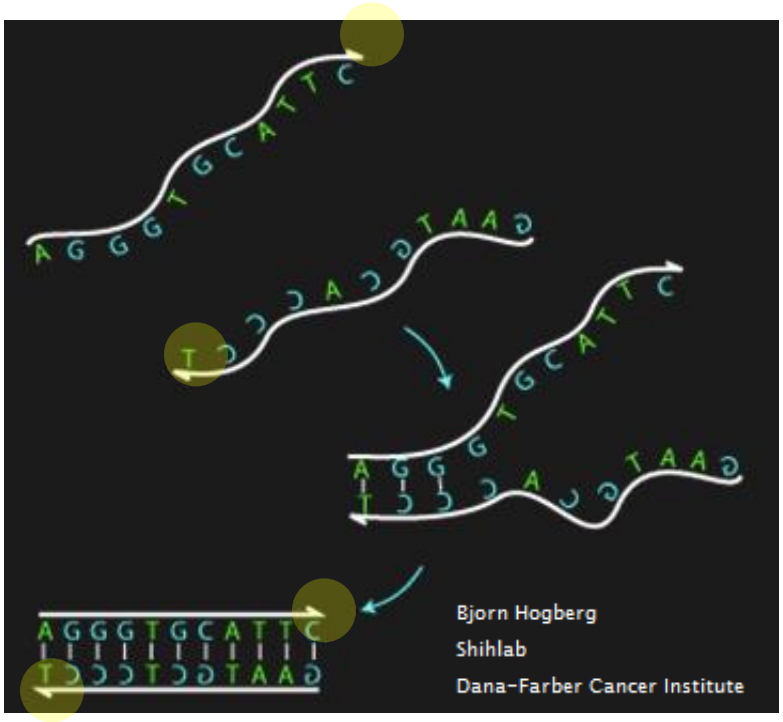
Vitravene (GCGTTTGCTCTTCTTCTTGCG)

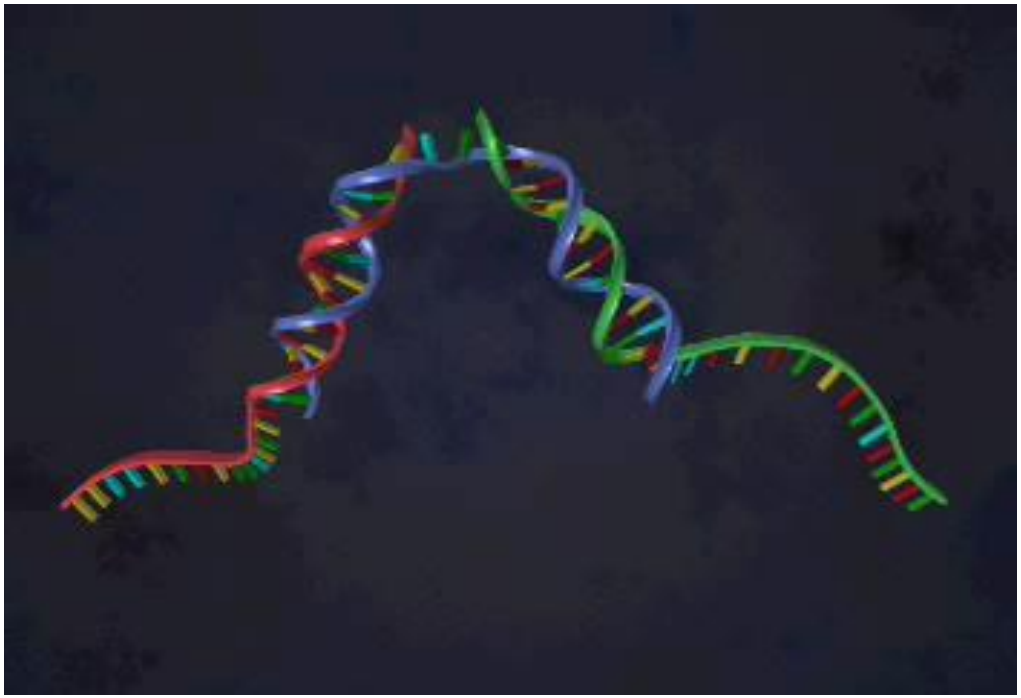
An automaton sequentially reading the string PPAP2B, GSTP1, PIM1, HPS (known cancer indicators) and sequentially cutting the DNA hairpin until a ssDNA drug (Vitravene) is released.

Designed by Benenson...Shapiro Nature 2004

DNA Computing Without Enzymes

Strands with opposite orientation and complementary base pairs stick to each other (Watson-Crick pairing)



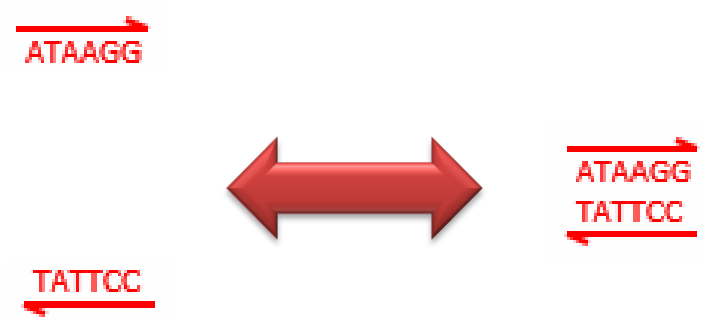


Bernard Yurke

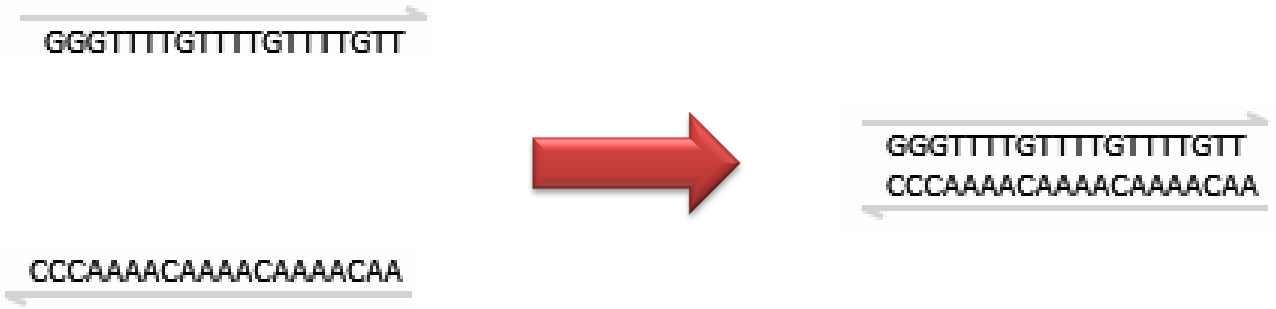
This simple principle can be used to compute with DNA

DNA Strand Displacement

Short complementary segments bind *reversibly*

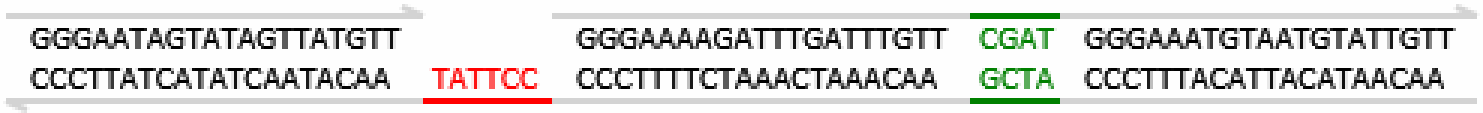


Long complementary segments bind *irreversibly*

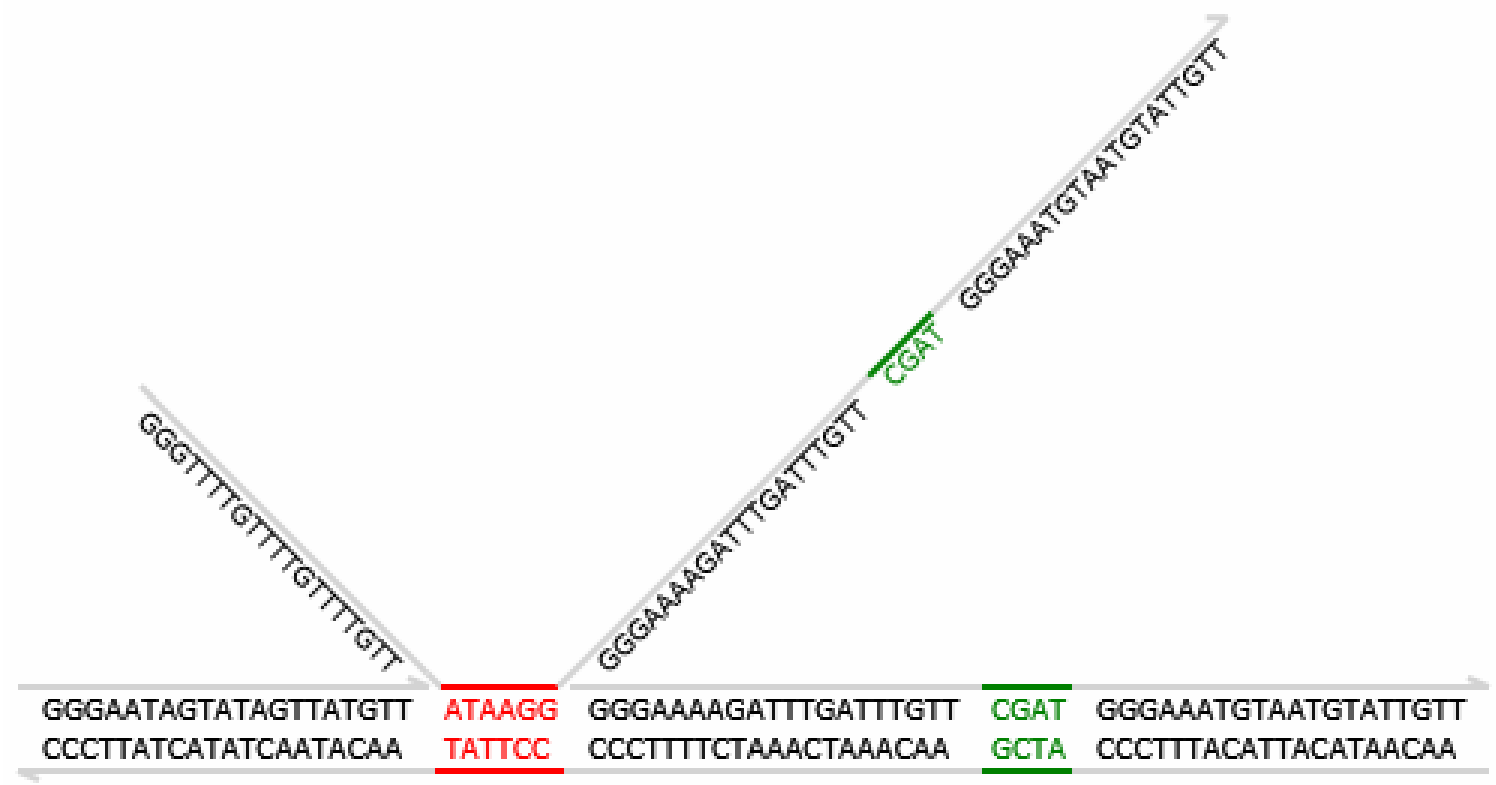


Proposed by Zhang et al., Science 2007.

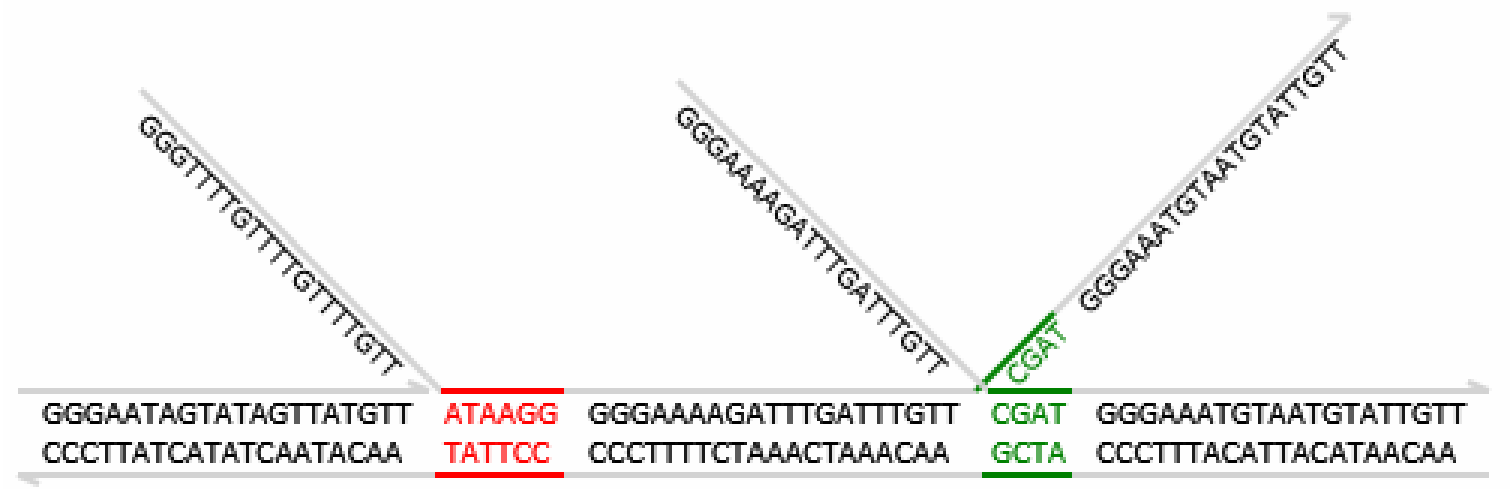
Bind, Migrate, Displace



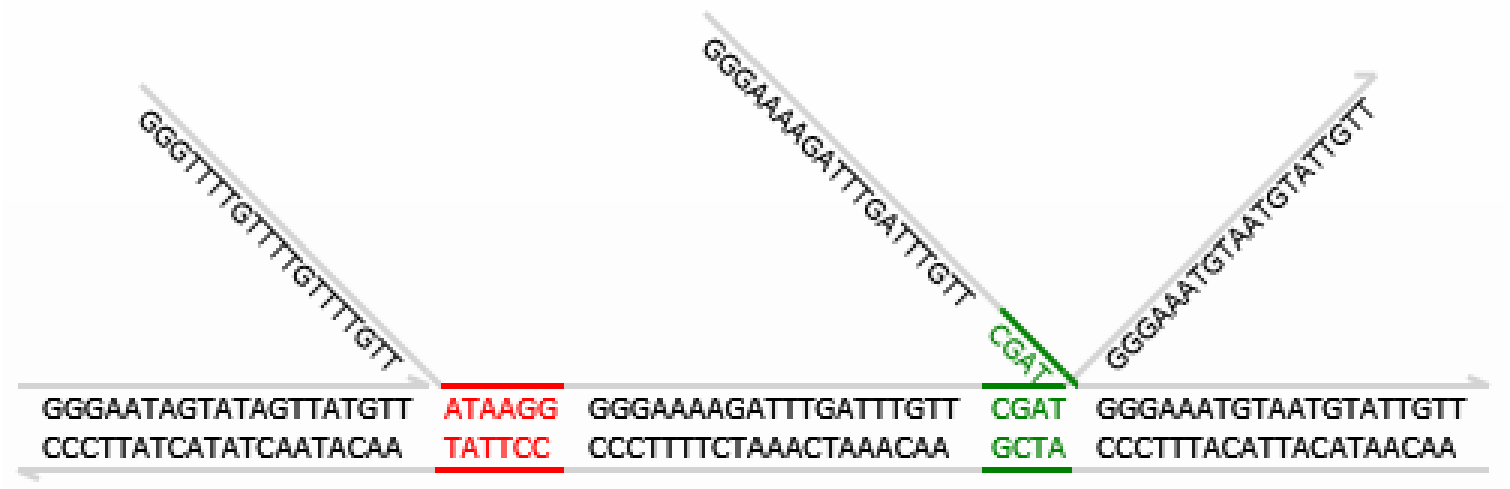
Bind, Migrate, Displace



Bind, Migrate, Displace



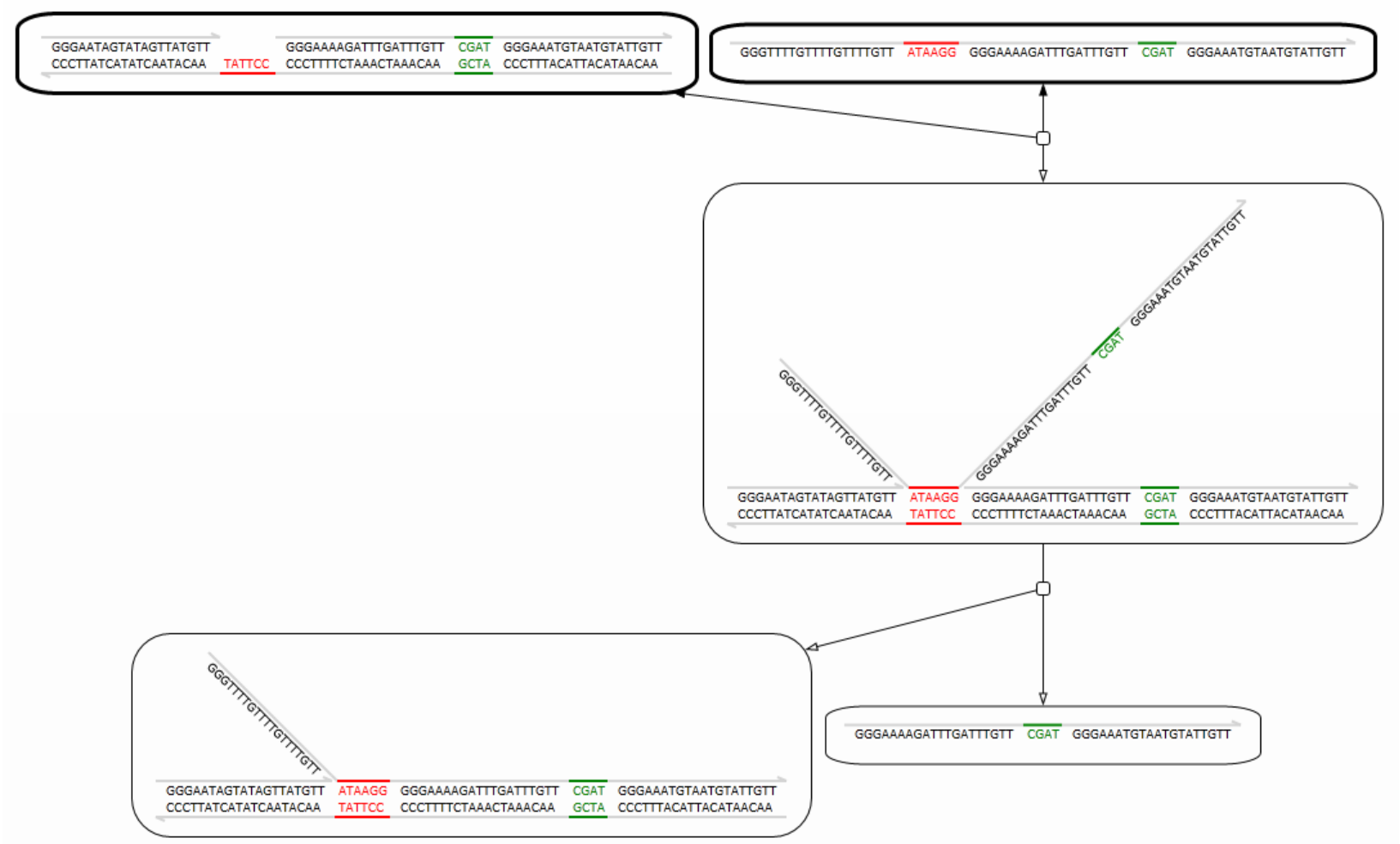
Bind, Migrate, Displace



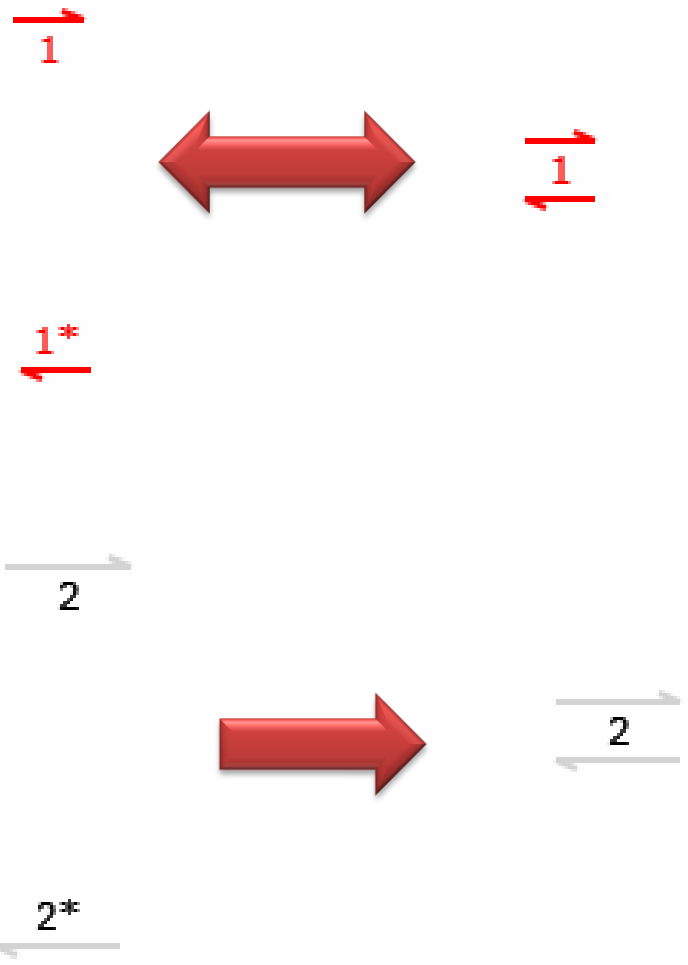
Bind, Migrate, Displace

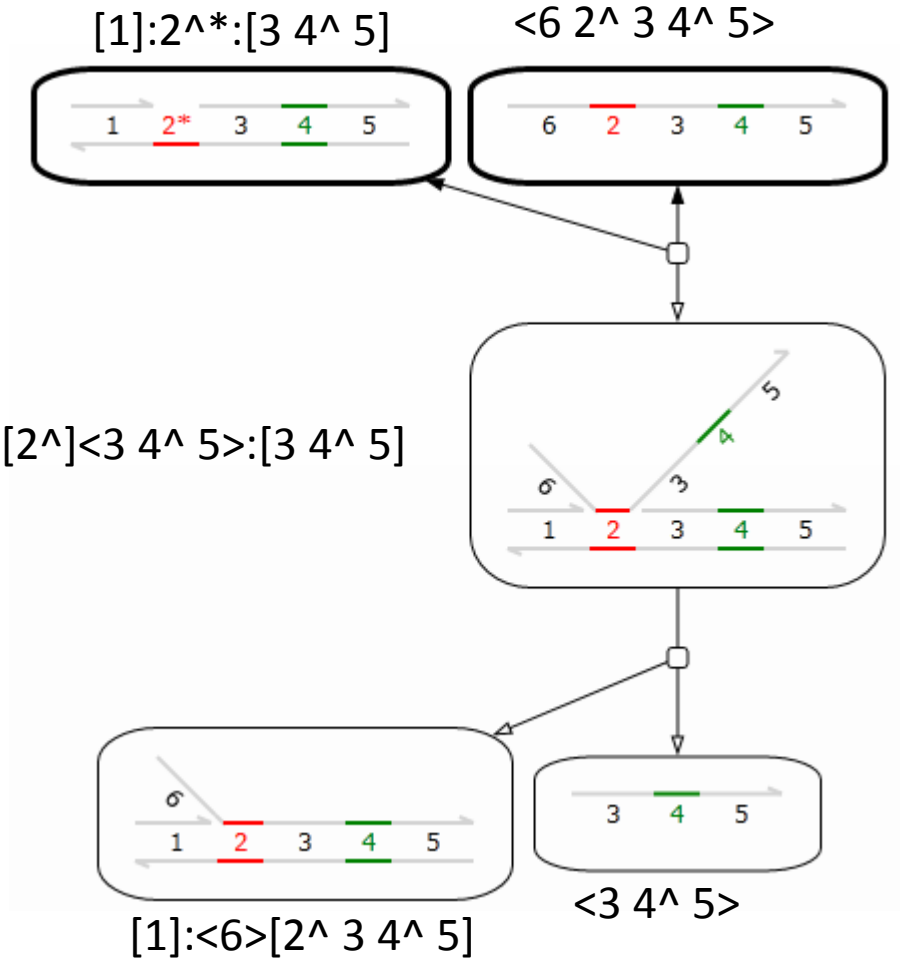
Reaction Graph

- Merge migrations into a single displacement



Simplified Notation

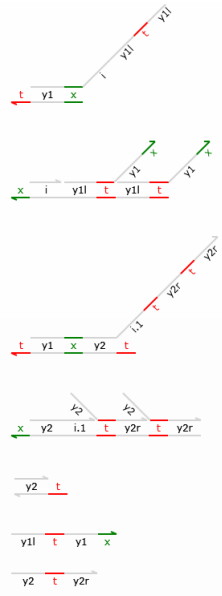




DNA Strand Displacement (DSD)

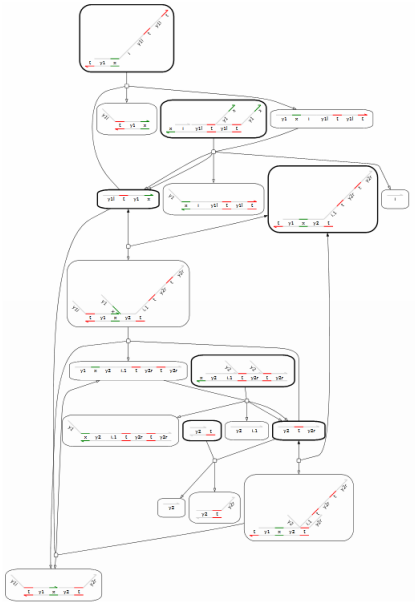
Designing DNA circuits

Step 1: Program circuit design

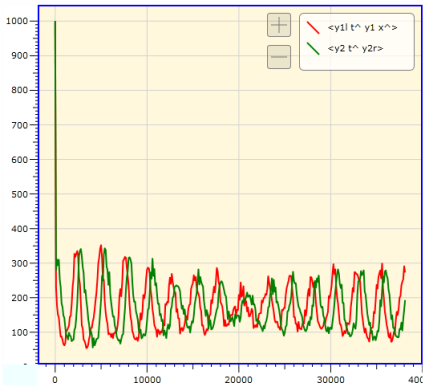


```
directive sample 200000.0 2000
directive plot <y1l t^ y1 x^>; <y2 t^ y2r>
directive leak 1.0E-10 (*1.0E-12*)

def Scale = 1
def Excess = 1000
def bind = 0.00001
def unbind = 0.1

new x@ bind,unbind
new t@ bind,unbind

def SpeciesL(N,al,a) = N * Scale * <al t^ a x^>
def SpeciesR(N,a,ar) = N * Scale * <a t^ ar>
def BinaryLRxRR(N,al,a,b,br,c,cr,d,dr) =
 new i
 ( constant N * Scale * t^:[a x^ b]<i t^ cr t^ dr>:t^
 | constant N * Excess * x^:[b i]:<c>[t^ cr]:<d>[t^ dr]
 )
def UnaryLxLL(N,al,a,cl,c,dl,d) =
 new i
 ( constant N * Scale * t^:[a x^]<i cl t^ dl t^>
 | constant N * Excess * x^:[i]:[cl t^]<c x^>:[dl t^]<d x^>
 )
def UnaryRx(N,a,ar) =
 constant N * Scale * [a]:t^

( UnaryLxLL(1000,y1l,y1,y1l,y1,y1l,y1)
| BinaryLRxRR(30000,y1l,y1,y2,y2r,y2,y2r,y2,y2r)
| UnaryRx(1000,y2,y2r)
| SpeciesL(1000,y1l,y1)
| SpeciesR(1000,y2,y2r)
)
```

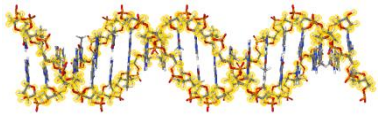
Step 2: Compile circuit behaviour



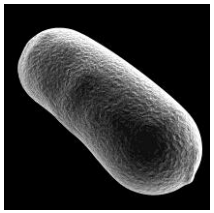
Step 3: Simulate circuit



Step 4: Compile circuit to DNA

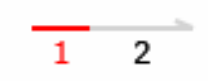


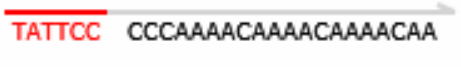
Step 5: Insert DNA into cells

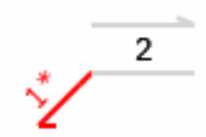


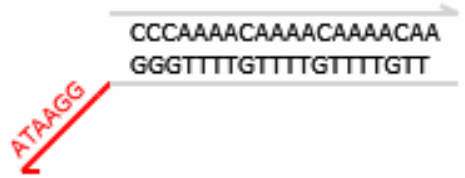
Phillips & Cardelli. Royal Society Interface, 2009

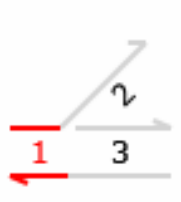
Basic Representation

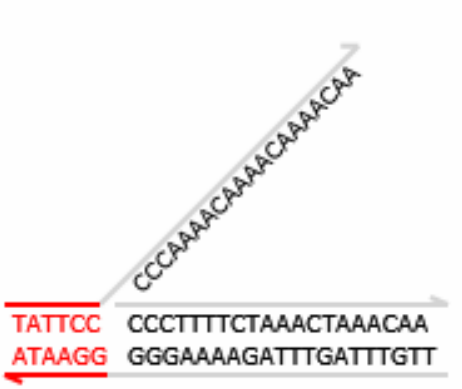








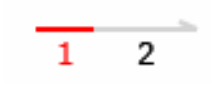




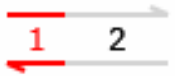
Basic Molecules

- Strands and Gates

{1^* 2*}



<1^ 2>



[1^ 2]

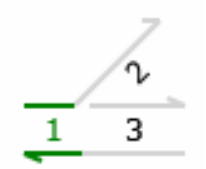
[1^ 2]:[3^ 4]

[1^ 2]:3^*:[4 5^]

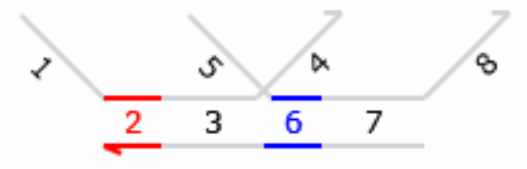
- Overhangs

<1>[2^ 3]<4>

1^*:[2]



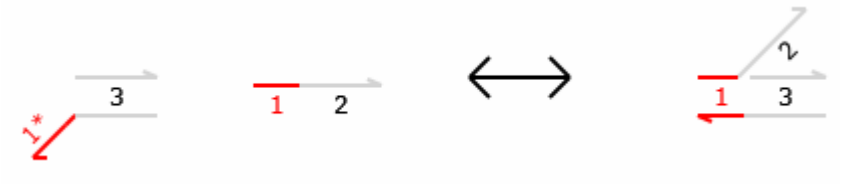
[1^]<2>:[3]



<1>[2^ 3]<4>:<5>[6^ 7]<8>

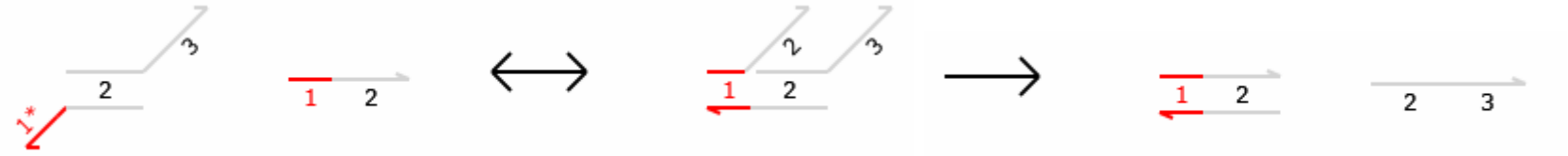
Basic Reactions

- Binding



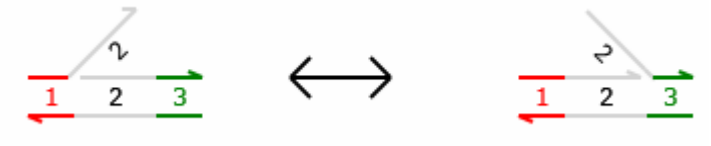
<1^ 2> | 1^*:[3] [1^]<2>:[3]

- Displacement



1^*:[2]<3> | <1^ 2> [1^]<2>:[2]<3> [1^ 2] | <2 3>

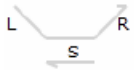
- Migration

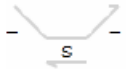
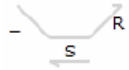
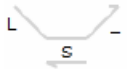
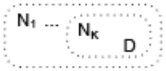
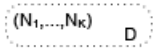


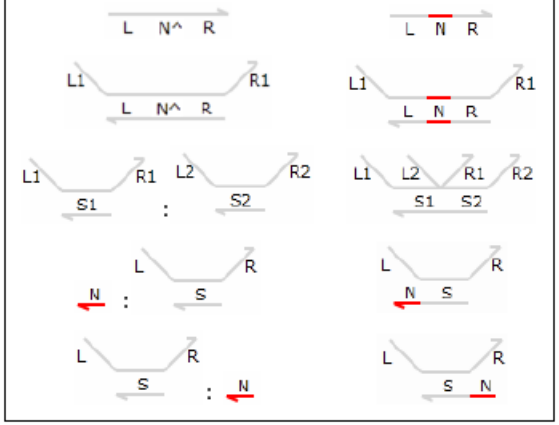
[1^]<2>:[2 3^] [1^ 2]:<2>[3^]

DSD Syntax

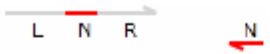
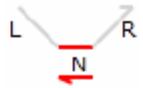
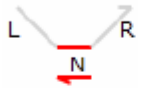
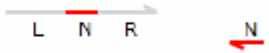
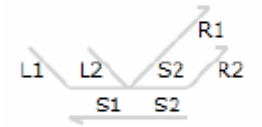
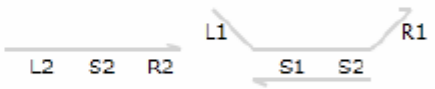
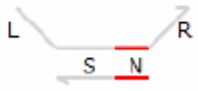
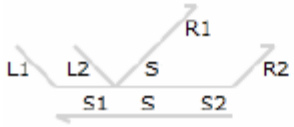
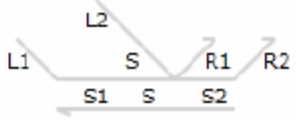
dsd	syntax	description
S	N	Domain
	N^	Toehold domain
	S1 S2	Concatenation of S1 and S2
L,R	_	Empty sequence
	S	Domain sequence

dsd	syntax	description
G	N^ 	Lower toehold domain N^
	<L>[S]<R> 	Double strand [S] with overhanging strands <L>, <R>
	G1:G2	Concatenation of gates G1,G2
D	<S>	Strand with sequence complementary to S
	G	Gate G
	D1 | D2 D1 D2	Parallel molecules D1, D2
	new N D 	Molecules D with private domain N
	X(n)	Module X with parameters n

syntax	abbreviation
<_>[S]<_> 	[S]
<_>[S]<R> 	[S]<R> 
<L>[S]<_> 	<L>[S] 
new N1 ... new NK D 	new (N1,...,NK) D 
D | ... | D (K) D ... D (K)	K*D K*D

syntax	abbreviation
	

DSD Semantics

#	before	red	after
RB	<L N^ R> | N^ 	$\xrightarrow{N+}$	<L>[N^]<R> 
RU	<L>[N^]<R> 	$\xrightarrow{N-}$	<L N^ R> | N^ 
RD	<L1>[S1]<S2 R1>:<L2>[S2]<R2> 	$\xrightarrow{S2\tilde{}}$	<L1>[S1 S2]<R1> | <L2 S2 R2> 
RC	<L>[S]<N^ R>:N^ 	$\xrightarrow{N\tilde{}}$	<L>[S N^]<R> 
RM	<L1>[S1]<S R1>:<L2>[S S2]<R2> 	$\xrightarrow{S\tilde{}}$	<L1>[S1 S]<R1>:<L2 S>[S2]<R2> 

DSD Compilation Algorithm

Table 7: Syntax of the DSD compiler, where a term T consists of a set of local domains N, strands S, gates G and a multiset of reactions R.

term	syntax	description
T	(N, S, G, R)	Local domains N, upper strands S, gates G, reactions R
S	$\{\texttt{<S}_1\texttt{>}, ..., \texttt{<S}_N\texttt{>}\}$	Set of N strands
G	$\{\texttt{G}_1, ..., \texttt{G}_N\}$	Set of N gates
R	$\{\theta_1, ..., \theta_N\}$	Multiset of N reactions
θ	$(\texttt{<S>}, \texttt{G}, r, \texttt{G'}, S')$	Binary reaction
	$(\texttt{G}, r, \texttt{G'}, S')$	Unary reaction

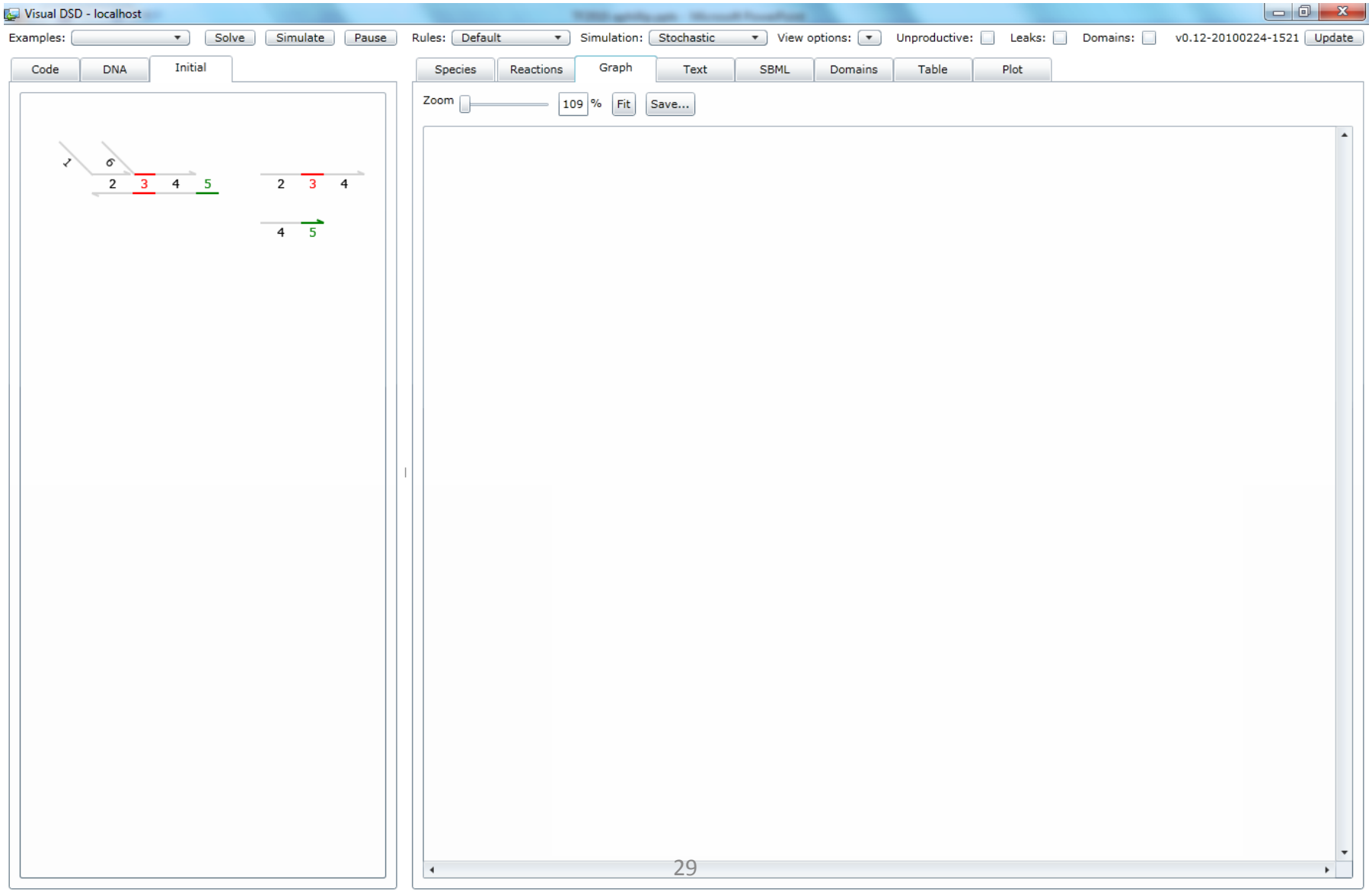
Table 8: Computing multisets of reactions and species

before	def	after
unary(G)	$\triangleq$	$\{(\texttt{G}, r, \texttt{G'}, \{\texttt{S}_1, ..., \texttt{S}_N\}) \mid \texttt{G} \xrightarrow{r} \texttt{G'} \mid \texttt{S}_1 \mid ... \mid \texttt{S}_N\}$
binary(G, S)	$\triangleq$	$\{(\texttt{<S>}, \texttt{G}, r, \texttt{G'}, \{\texttt{S}_1, ..., \texttt{S}_N\}) \mid \texttt{<S>} \in S \wedge \texttt{G'} \mid \texttt{<S>} \xrightarrow{r} \texttt{G'} \mid \texttt{S}_1 \mid ... \mid \texttt{S}_N\}$
binary(<S>, G)	$\triangleq$	$\{(\texttt{<S>}, \texttt{G}, r, \texttt{G'}, \{\texttt{S}_1, ..., \texttt{S}_N\}) \mid \texttt{G} \in G \wedge \texttt{G'} \mid \texttt{<S>} \xrightarrow{r} \texttt{G'} \mid \texttt{S}_1 \mid ... \mid \texttt{S}_N\}$
react(<S>, G, r, G', S')	$\triangleq$	$\{\texttt{G}, \texttt{<S>}\}$
react(G, r, G', S')	$\triangleq$	$\{\texttt{G}\}$
prod(<S>, G, r, G', S')	$\triangleq$	$\{\texttt{G'}\} \uplus S'$
prod(G, r, G', S')	$\triangleq$	$\{\texttt{G'}\} \uplus S'$
merge($\theta_1, \ldots, \theta_N$)	$\triangleq$	merge(θ_1) $\uplus \ldots \uplus$ merge(θ_N)
merge(G, r, G', S')	$\triangleq$	$\{(\texttt{G}, r, \texttt{G''}, S' \uplus \{\texttt{S}_1, ..., \texttt{S}_N\}) \mid \texttt{G'} \twoheadrightarrow \texttt{G''} \mid \texttt{S}_1 \mid ... \mid \texttt{S}_N\}$
merge(G, <S>, r, G', S')	$\triangleq$	$\{(\texttt{G}, \texttt{<S>}, r, \texttt{G''}, S' \uplus \{\texttt{S}_1, ..., \texttt{S}_N\}) \mid \texttt{G'} \twoheadrightarrow \texttt{G''} \mid \texttt{S}_1 \mid ... \mid \texttt{S}_N\}$

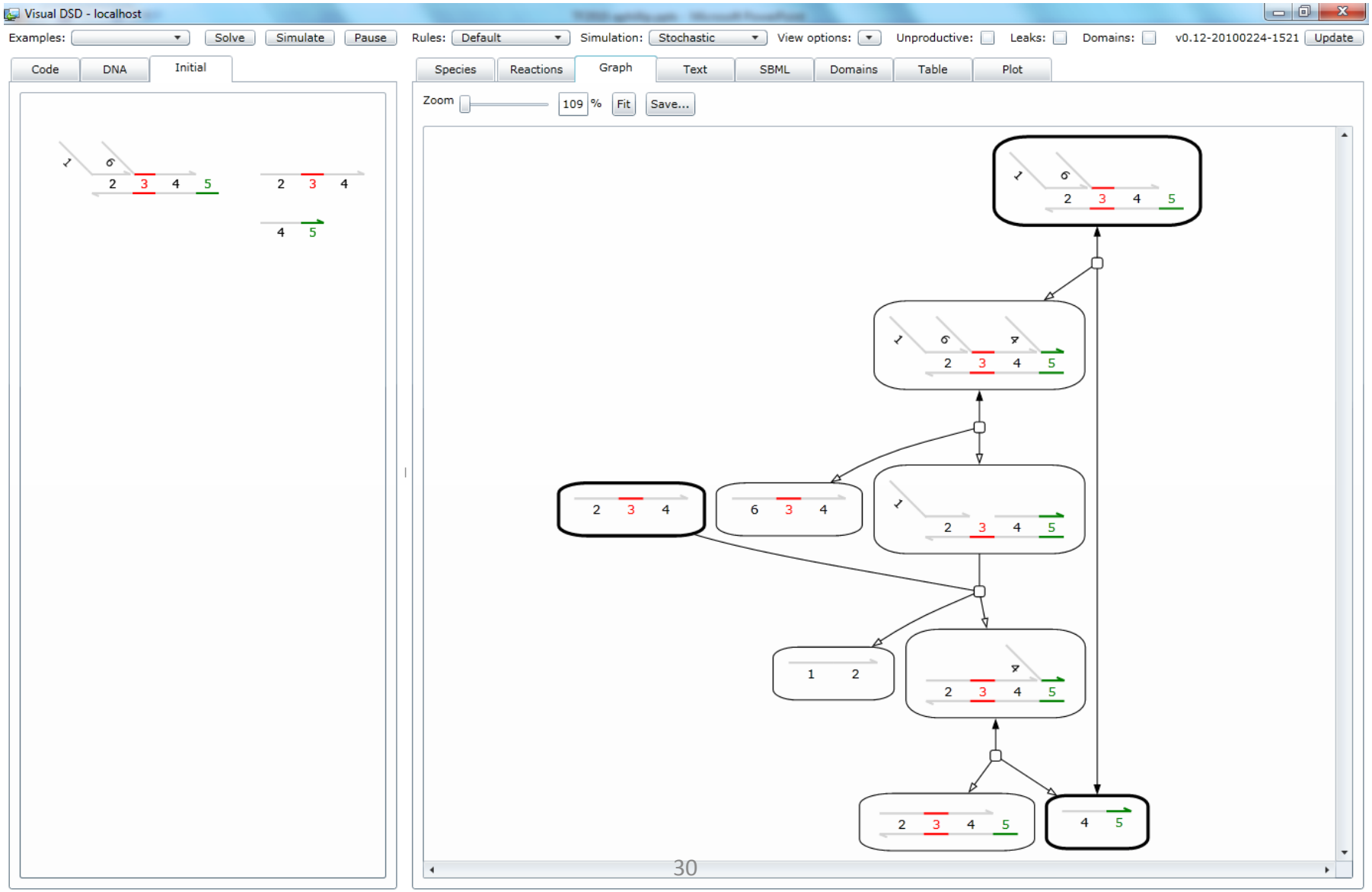
Table 9: Adding molecules to a term of the DSD compiler, where all molecules D are assumed to be in standard form. For the merged semantics, all gates G are also assumed to be in standard form. If S is a multiset $\{S_1, ..., S_N\}$ we write $S \oplus T$ as short for $S_1 \oplus \cdots \oplus S_N \oplus T$.

rule	conditions	before	def	after
CN	$\texttt{N} \notin N$	(new N D) $\oplus (N, S, G, R)$	$\triangleq$	$\texttt{D} \oplus (\{\texttt{N}\} \cup N, S, G, R)$
CP		$(\texttt{D}_1 \mid \texttt{D}_2) \oplus T$	$\triangleq$	$\texttt{D}_1 \oplus \texttt{D}_2 \oplus T$
CSZ	$\texttt{<S>} \in S$	<S> $\oplus (N, S, G, R)$	$\triangleq$	(N, S, G, R)
CS	$\texttt{<S>} \notin S \quad S' = \text{prod}(R')$	<S> $\oplus (N, S, G, R)$	$\triangleq$	$S' \oplus (N, \{\texttt{<S>}\} \cup S, G, R \uplus R')$
	$R' = \text{merge}(\text{binary}(\texttt{<S>}, G))$			
CGZ	$\texttt{G} \in G$	G $\oplus (N, S, G, R)$	$\triangleq$	(N, S, G, R)
CG	$\texttt{G} \notin G \quad S' = \text{prod}(R')$	G $\oplus (N, S, G, R)$	$\triangleq$	$S' \oplus (N, S, \{\texttt{G}\} \cup G, R \uplus R')$
	$R' = \text{merge}(\text{unary}(\texttt{G}) \uplus \text{binary}(\texttt{G}, S))$			

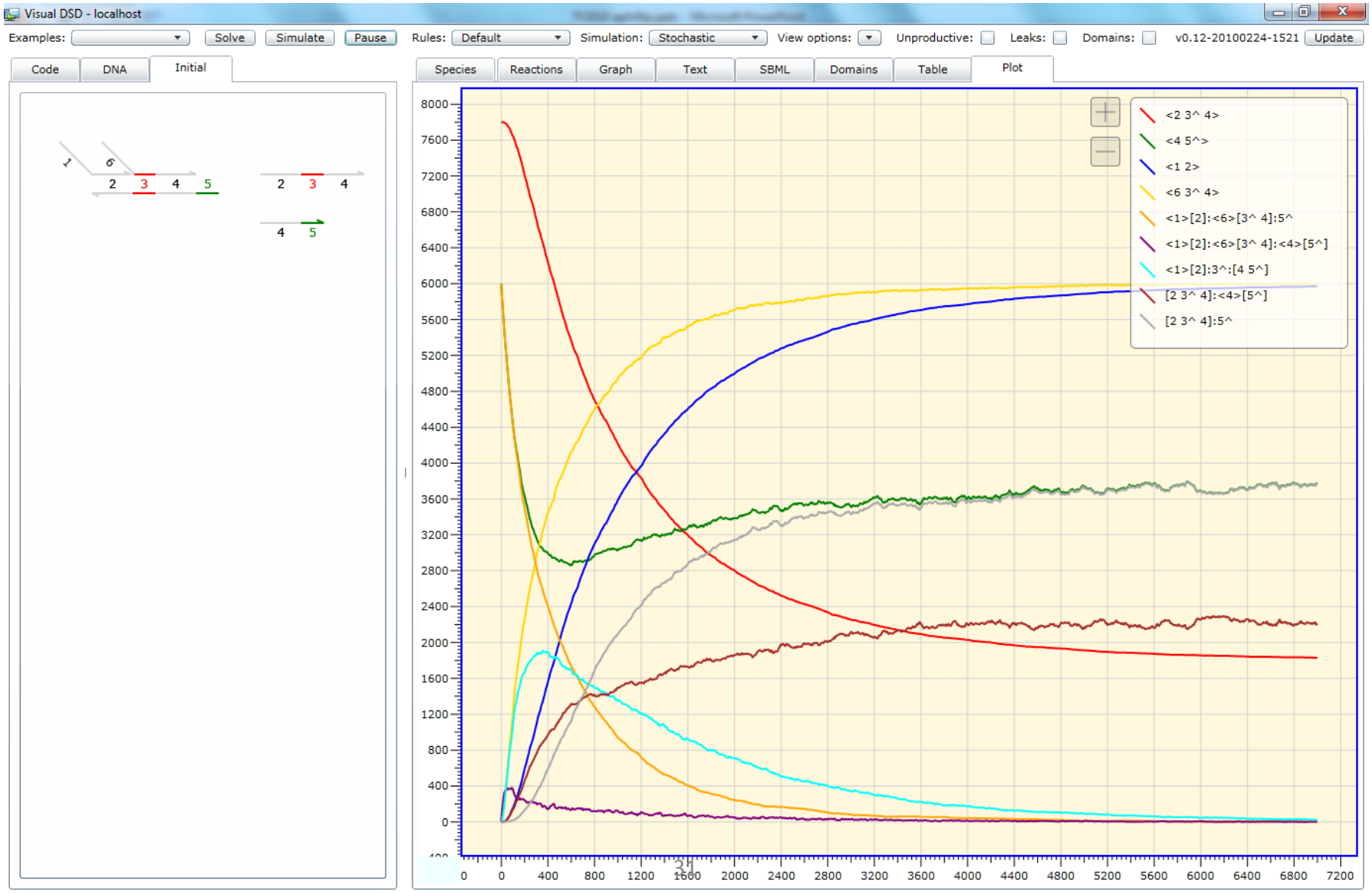
DNA Strand Displacement Tool



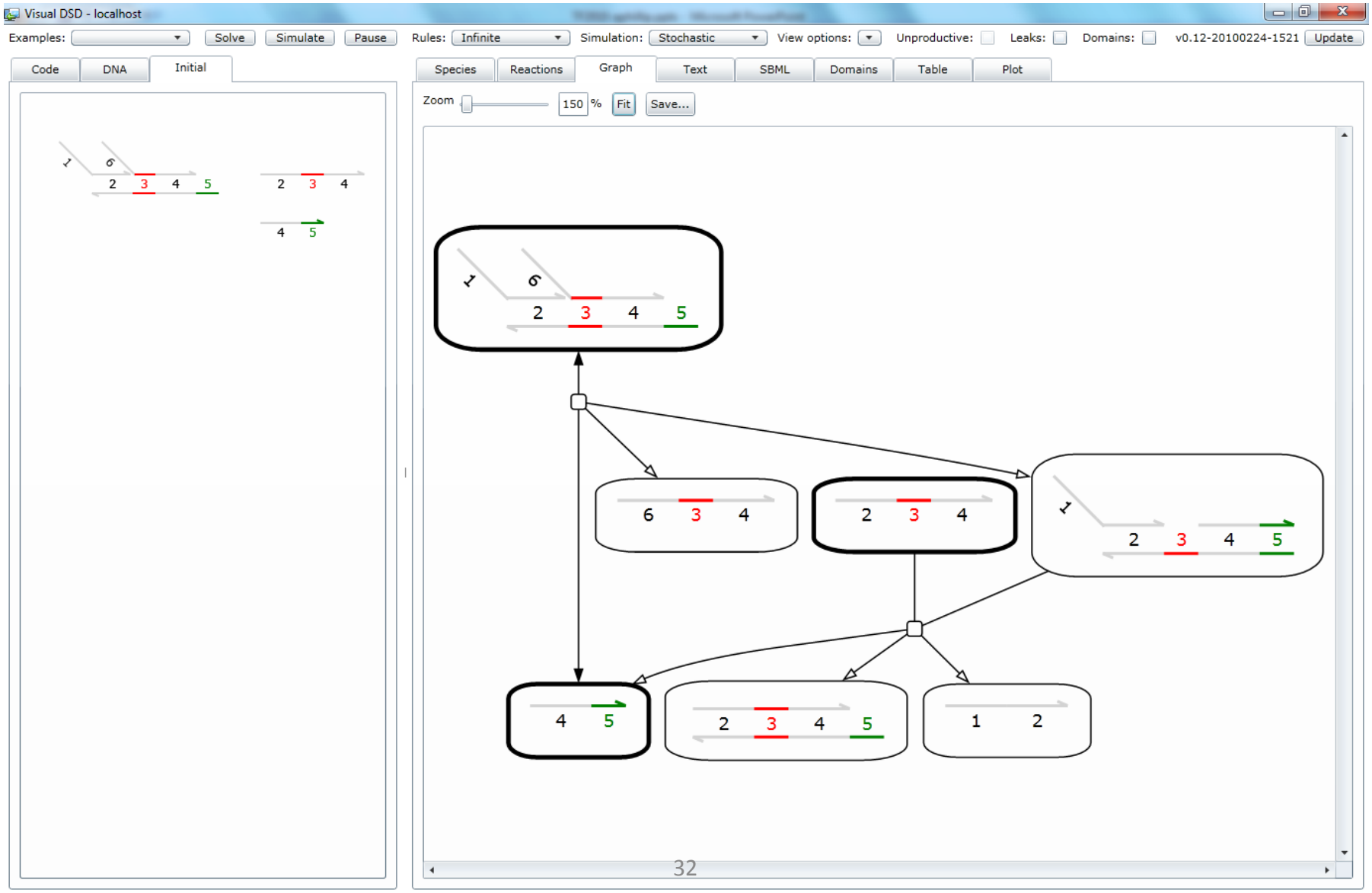
Computing Circuit Behaviour



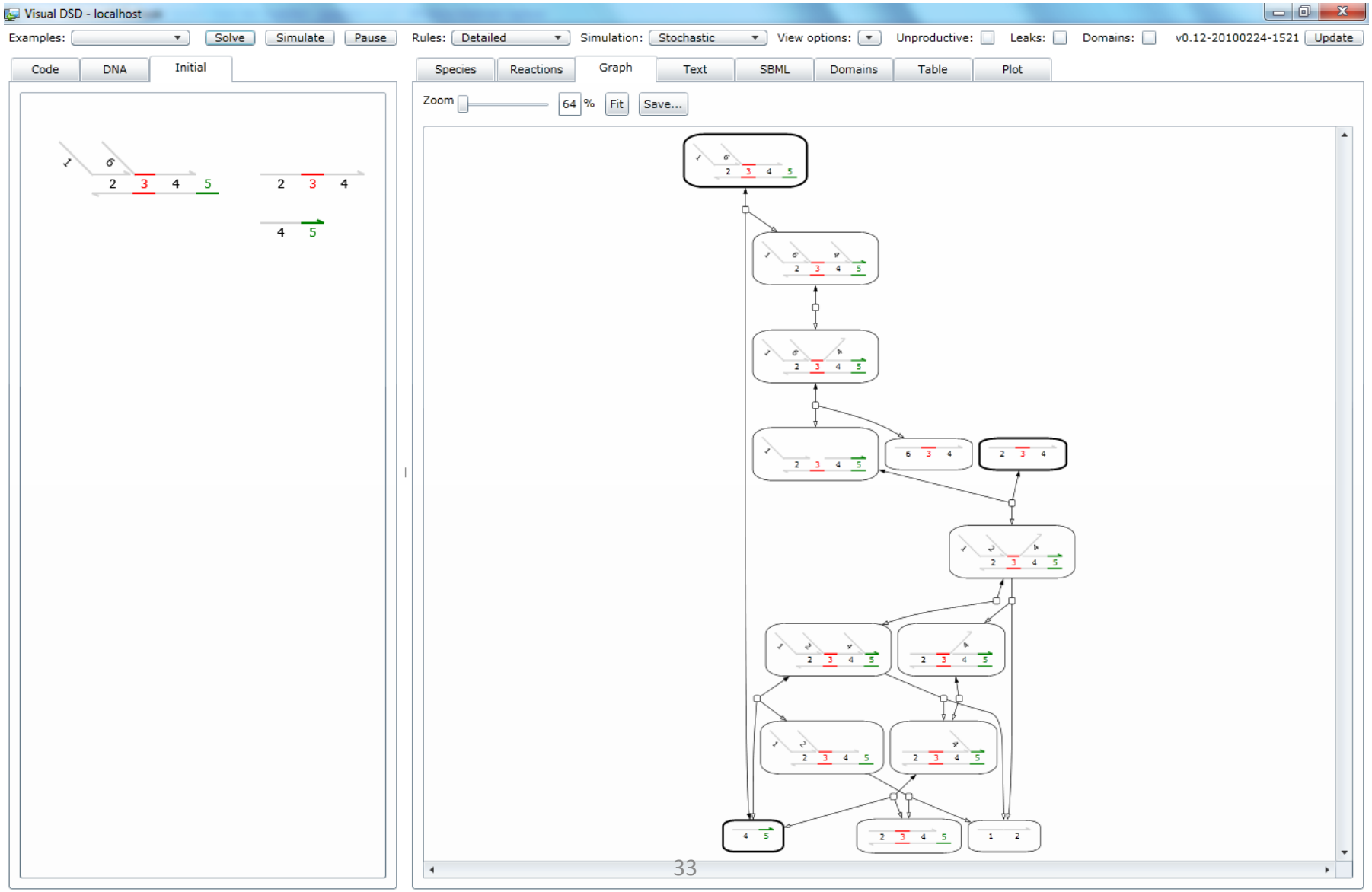
Simulating Circuit Behaviour



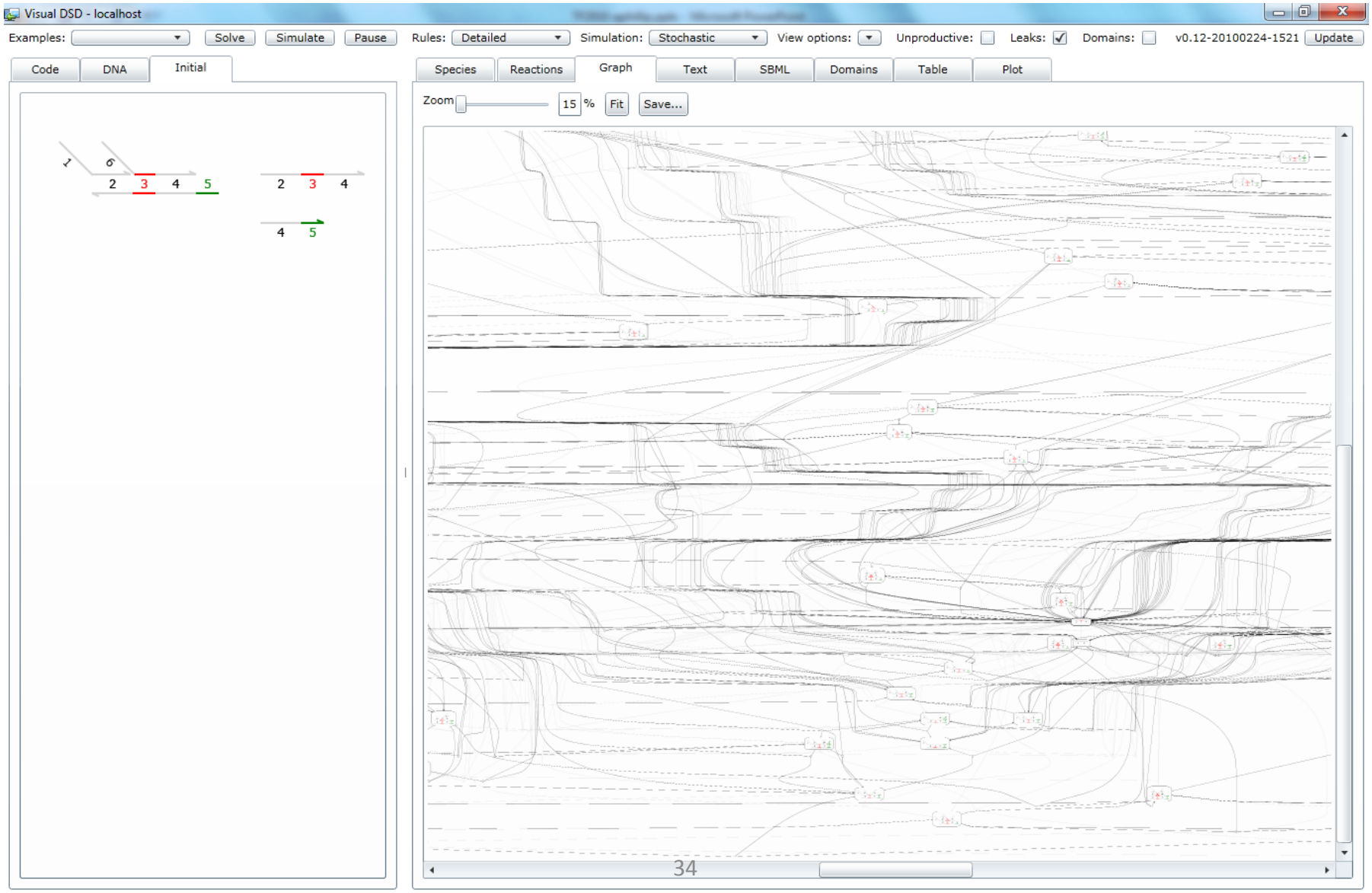
Abstract Reactions



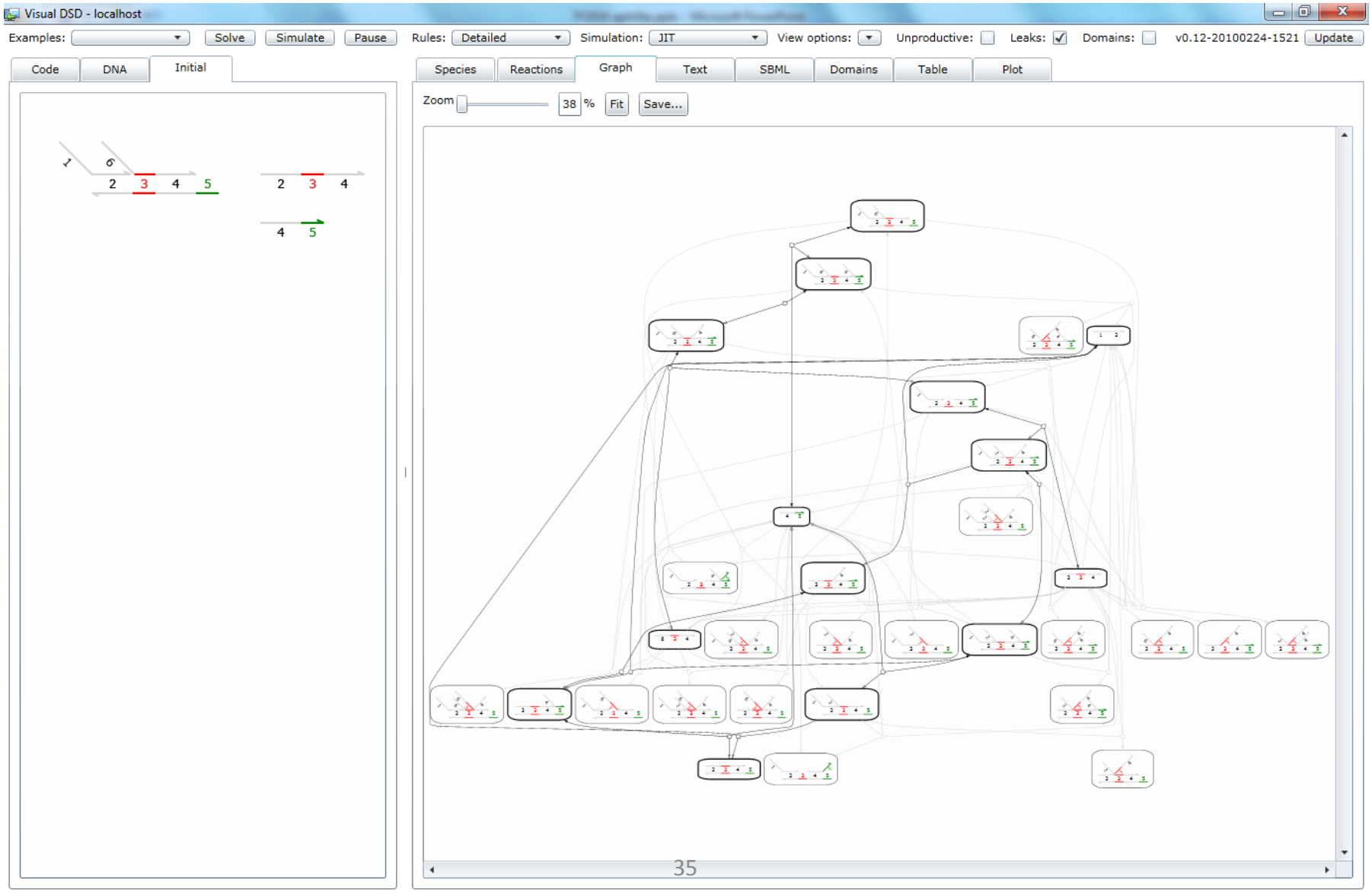
Detailed Reactions



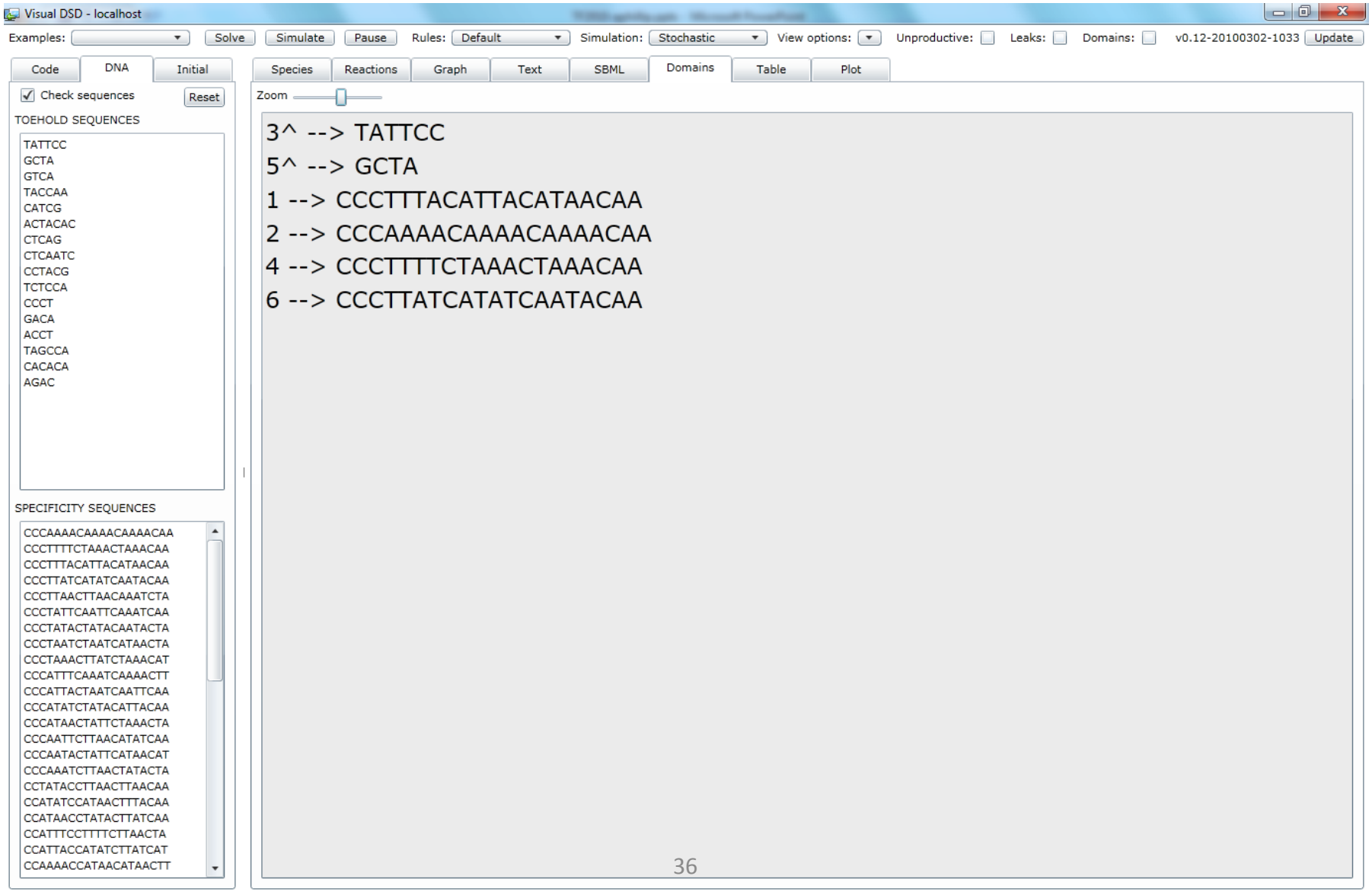
Leak Reactions!



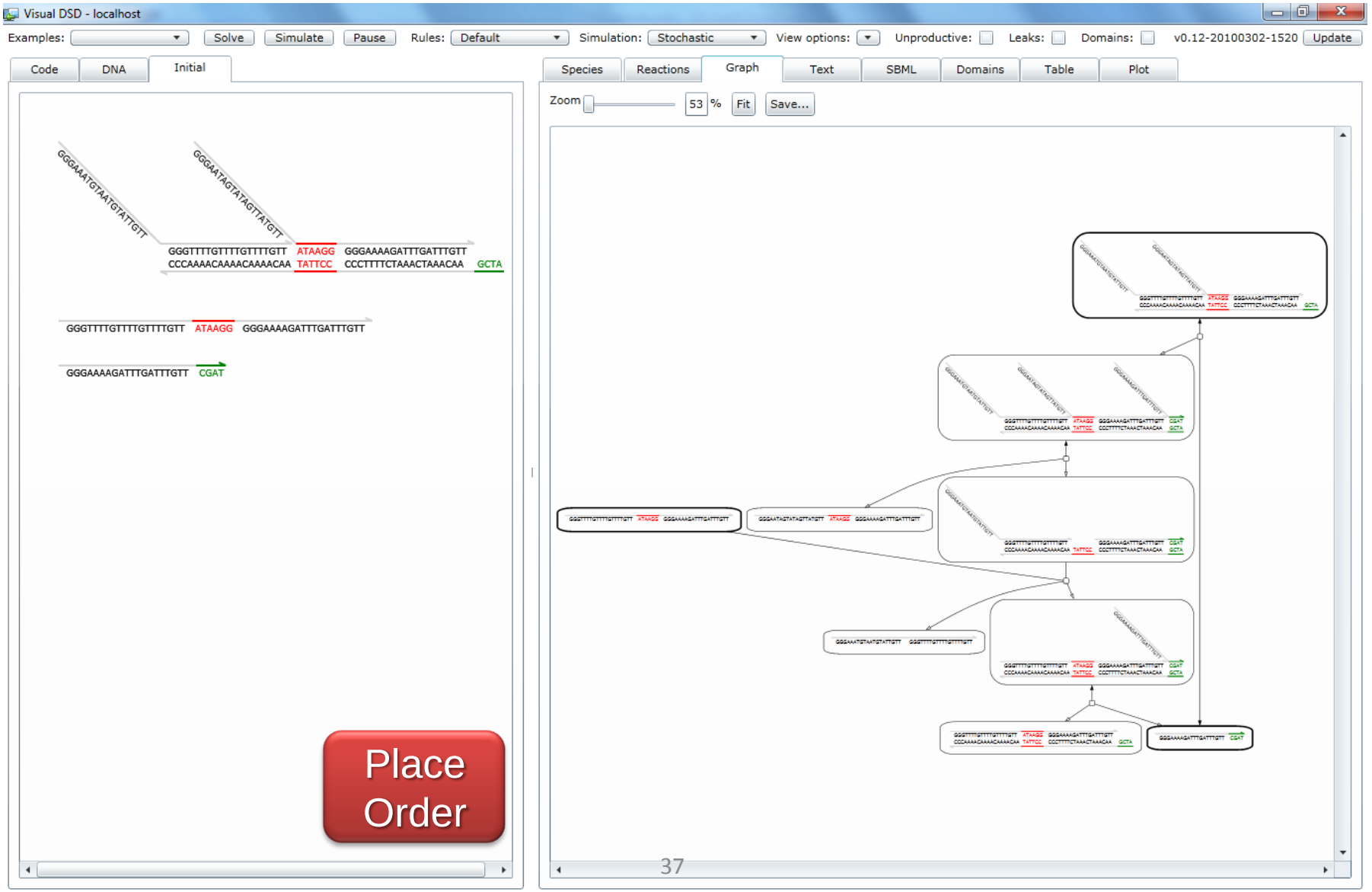
Just-In-Time Compilation



Compilation to DNA



Final DNA Circuit



Ordering DNA Online

Fastest tunaround time for less money!

Standard gene synthesis from 0.36 €/bp in just 8 days

Your Chance to Win a Nintendo Wii

SameDay® Oligo Service

Only £0.57 GBP / Base!

Base Pricing		
Synthesis Scale	**Price**	
25 nmole DNA Oligo	£0.25 GBP / Base	**Order**
100 nmole DNA oligo	£0.45 GBP / Base	**Order**
250 nmole DNA oligo	£0.80 GBP / Base	**Order**
1 µmole DNA oligo	£1.60 GBP / Base	**Order**
5 µmole DNA oligo	£7.50 GBP / Base	**Order**
10 µmole DNA oligo	£14.50 GBP / Base	**Order**

Find Out G-Reward

Earn rewards for every purchase!

Custom DNA/RNA

Pricing (USD)

DNA(mg)	Desalted	Purified
15	$700	$1,050
50	$1,200	$1,450
100	$1,500	$1,800
250	$2,000	$2,400
500	$2,900	$3,400
1000	$4,550	$5,400
5000	$9,000	$10,700

RNA(mg)	Desalted	Purified
5	$1,500	$1,925
15	$1,950	$2,490
50	$2,050	$2,625
100	$2,575	$3,575
250	$4,575	$5,725
500	$7,900	$9,190
1000	$13,900	$15,900
5000		$37,125

Please inquire for larger quantities

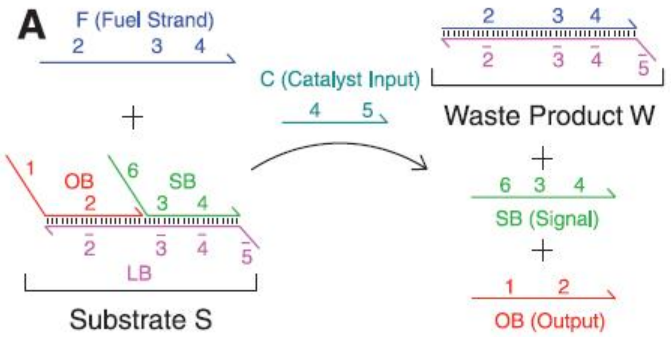
Gene Synthesis →

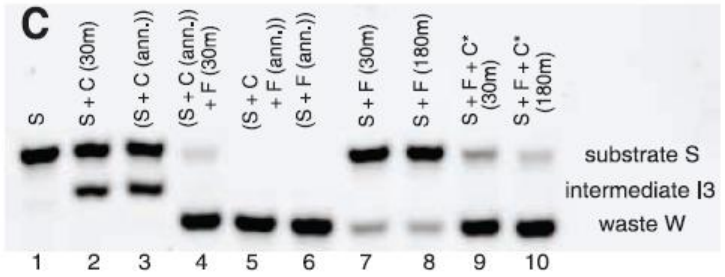
- Synthesize gene at $0.39/bp (till 3/31/2010)
- Guaranteed 100% sequence fidelity
- CloneEZ® seamless cloning technology

Struggling with cloning?

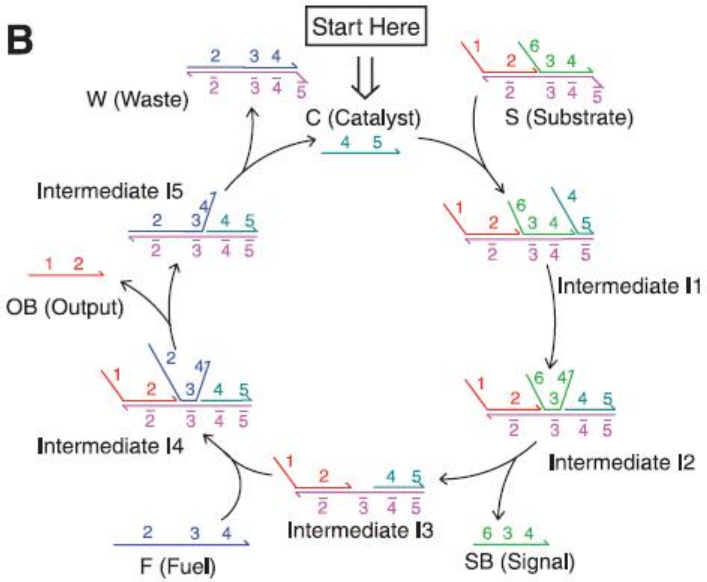
Try Gene-on-Demand® Service!

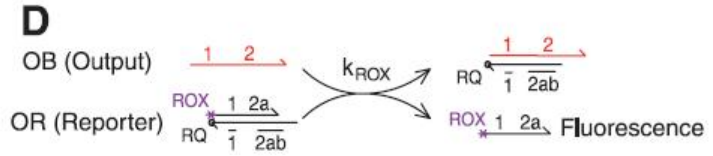
Catalytic DNA Circuit

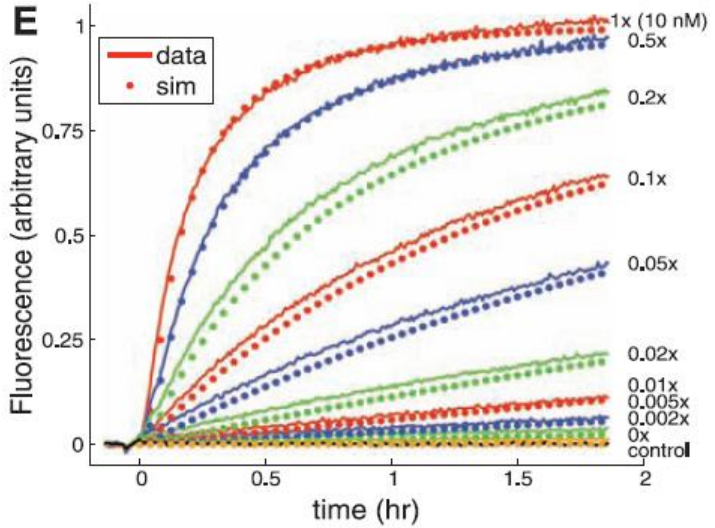




Sequence
5′-CTTTCCTACA-3′
5′-CCTACG-3′
5′-TCTCCA-3′
5′-ACTAACTTACGG-3′
5′-CCCT-3′
5′-CATTCAATACCCTACG-3′
5′-TCTCCA-3′
5′-CCACATACATCATATT-3′

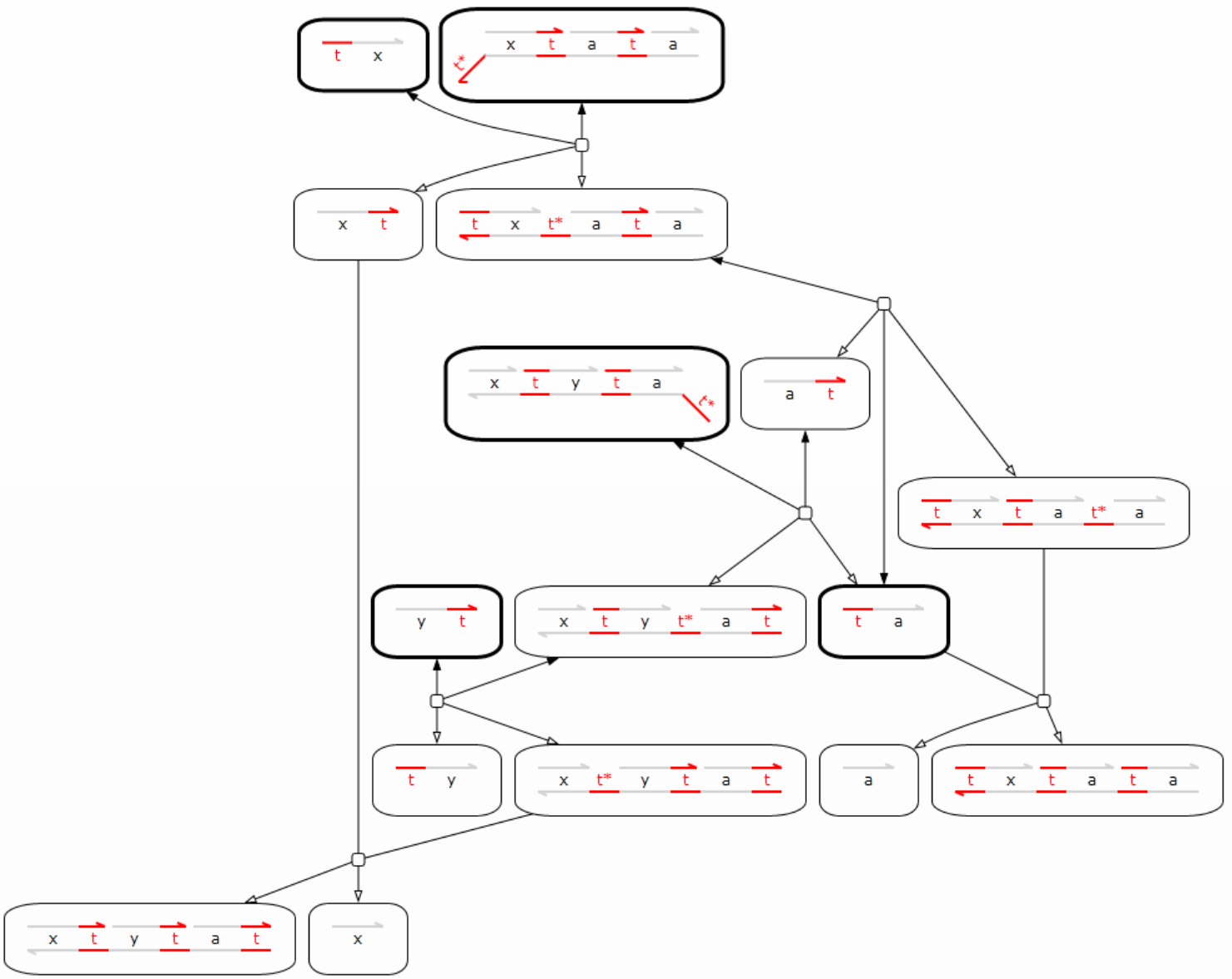




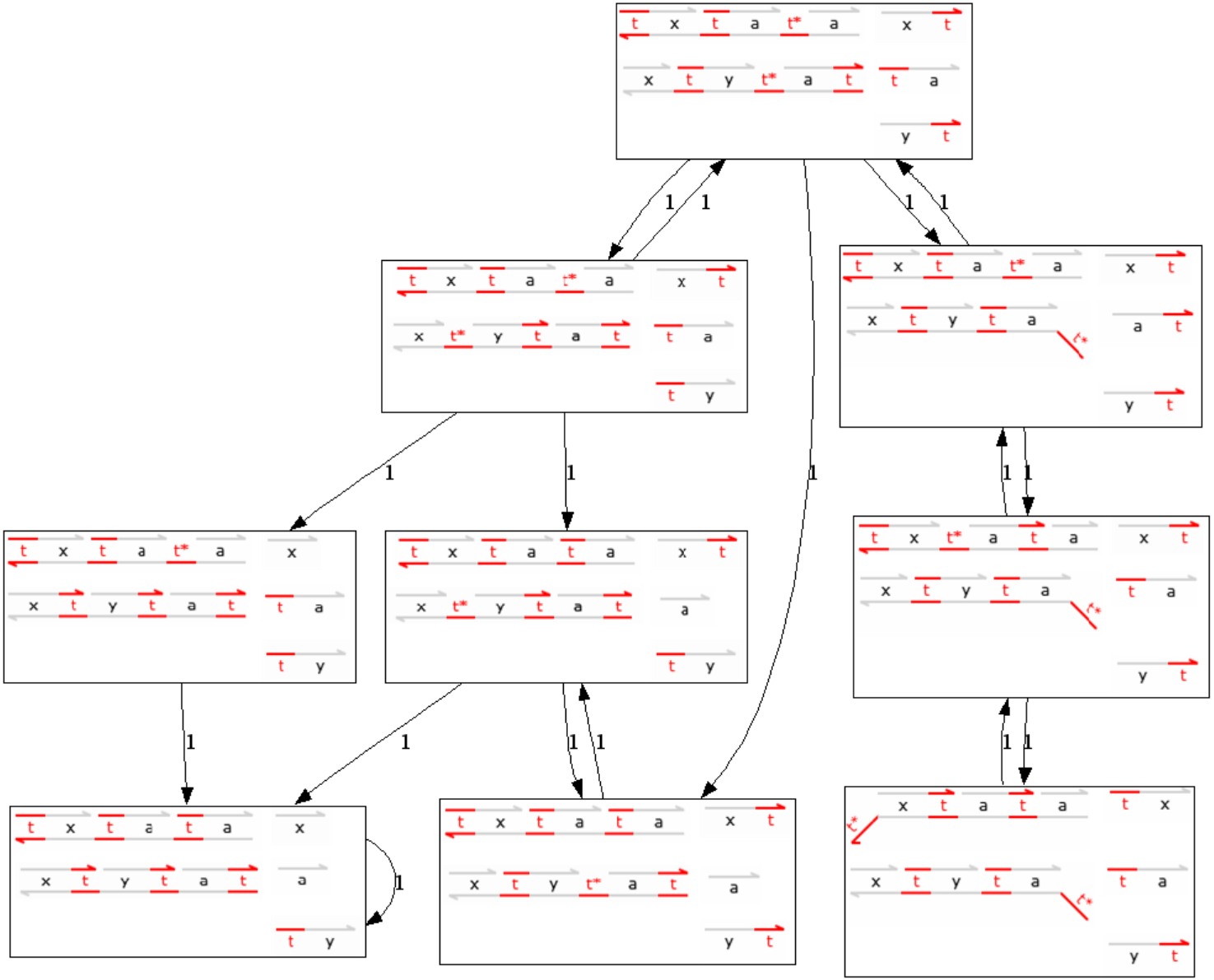


Engineered by Zhang et al., Science 2007.

Transducer

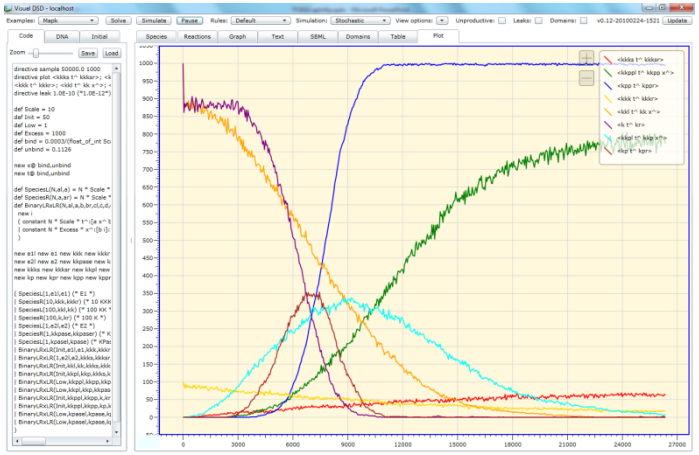

t x
x t a t a
t*
x t
t x t* a t a
x t y t a
t*
a t
t x t a t* a
y t
x t y t* a t
t a
t y
x t* y t a t
a
t x t a t a
x t y t a t
x

Transducer State Space

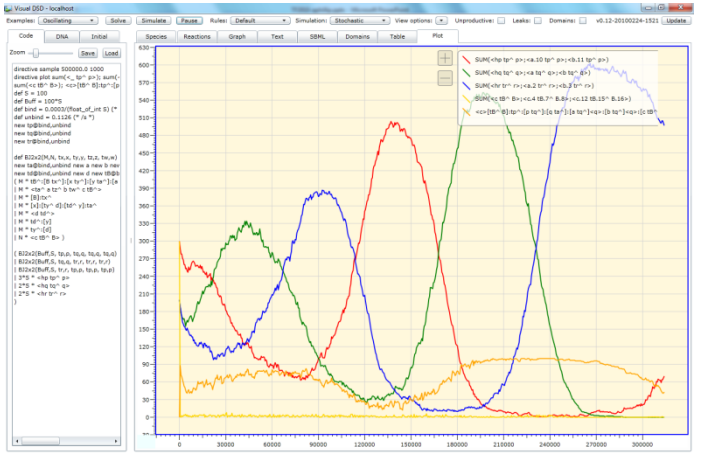


New Designs

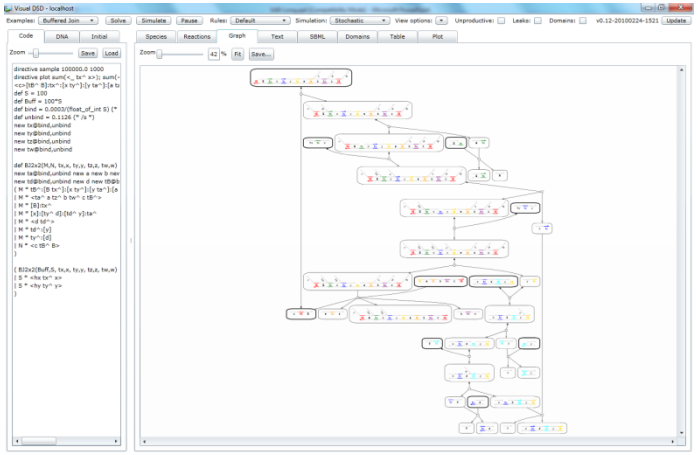
Ultrasensitive Switch



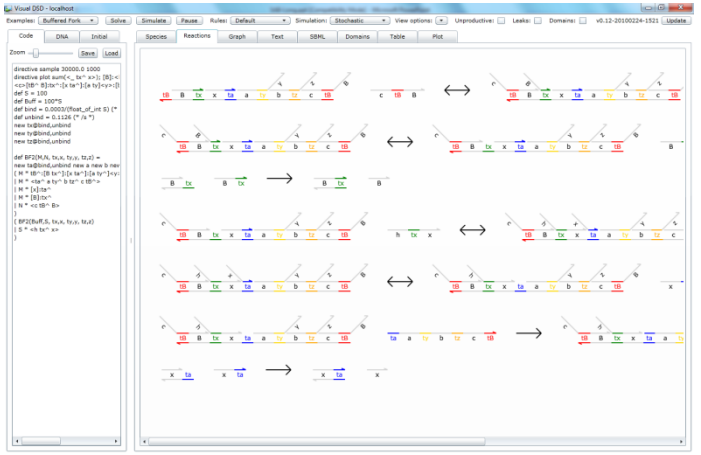
Timed Oscillator



Arbitrary Chemical System

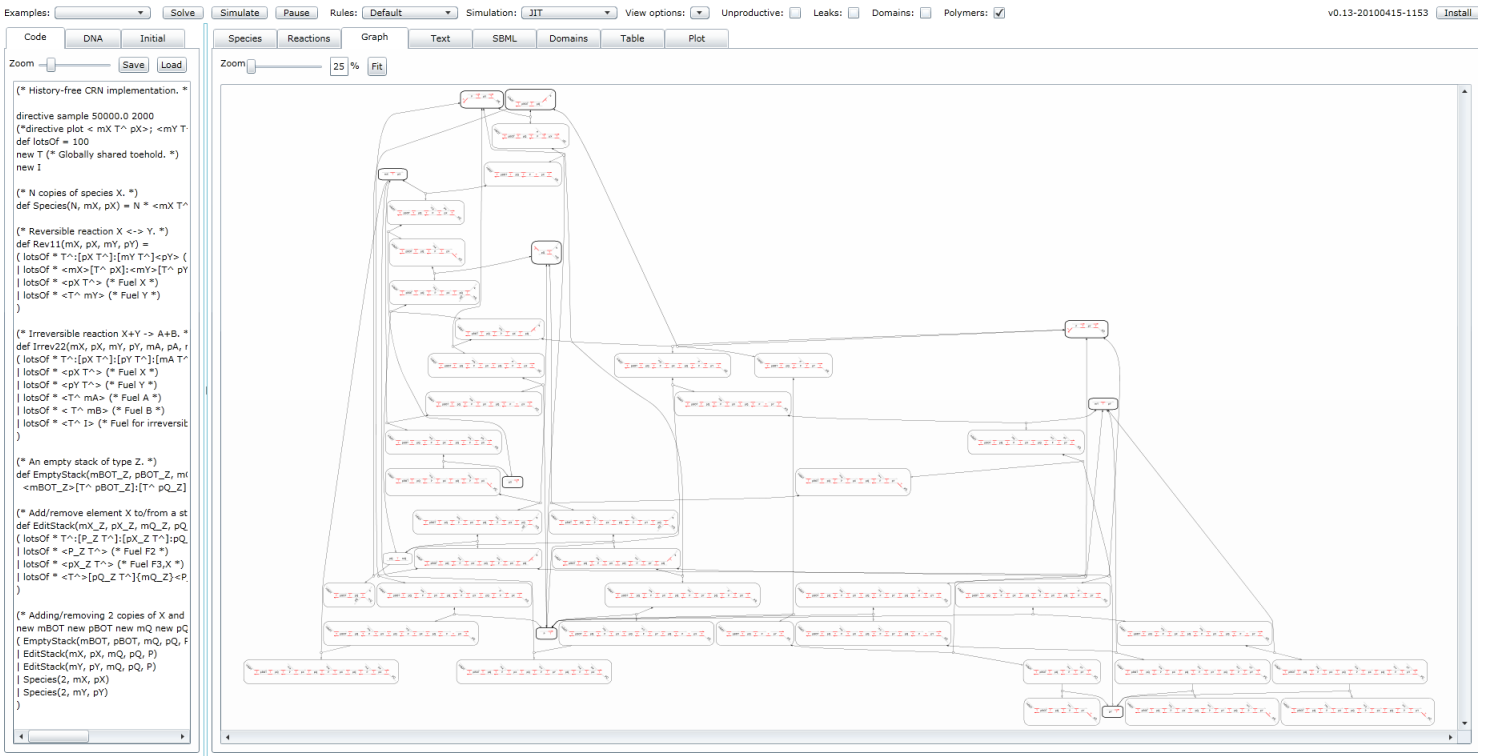


Petri Nets, Boolean Networks



Scientific Challenges

- Design a universal computer made of DNA



- Design smart drugs made of DNA

Programming Genetic Devices

Michael Pedersen, Neil Dalchau,
James Brown & Andrew Phillips

Environmentally Controlled Invasion of Cancer Cells by Engineered Bacteria

J. Christopher Anderson[1,3], Elizabeth J. Clarke[3], Adam P. Arkin[1,2*] and Christopher A. Voigt[2,3]

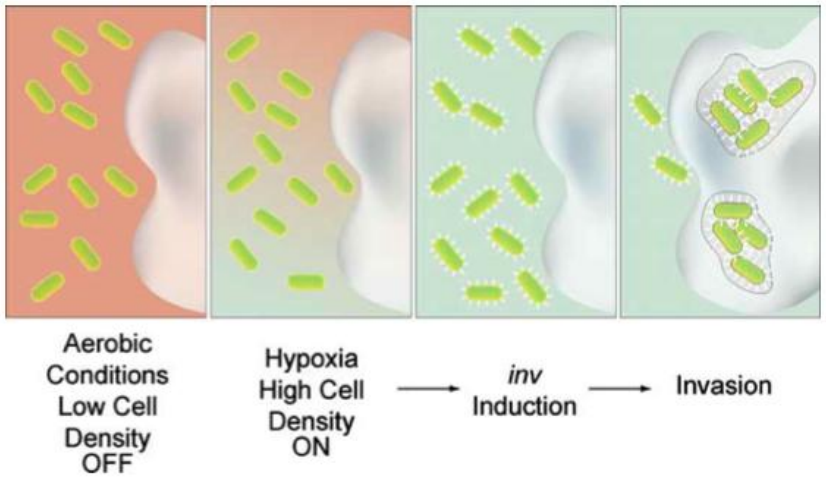
[1]*Howard Hughes Medical Institute, California Institute of Quantitative Biology Department of Bioengineering University of California, 717 Potter Street, Room 257 Berkeley, CA 94720, USA*

[2]*Physical Biosciences Division E.O. Lawrence Berkeley National Laboratory, 1 Cyclotron Road, MS 977-257 Berkeley, CA 94710, USA*

[3]*Biophysics Program Department of Pharmaceutical Chemistry, California Institute of Quantitative Biology The University of California San Francisco, 600 16th St. San Francisco, CA 94107 USA*

**Corresponding author*

Bacteria can sense their environment, distinguish between cell types, and deliver proteins to eukaryotic cells. Here, we engineer the interaction between bacteria and cancer cells to depend on heterologous environmental signals. We have characterized invasin from *Yersinia pseudotuburculosis* as an output module that enables *Escherichia coli* to invade cancer-derived cells, including HeLa, HepG2, and U2OS lines. To environmentally restrict invasion, we placed this module under the control of heterologous sensors. With the *Vibrio fischeri lux* quorum sensing circuit, the hypoxia-responsive *fdhF* promoter, or the arabinose-inducible *araBAD* promoter, the bacteria invade cells at densities greater than 10^8 bacteria/ml, after growth in an anaerobic growth chamber or in the presence of 0.02% arabinose, respectively. In the process, we developed a technique to tune the linkage between a sensor and output gene using ribosome binding site libraries and genetic selection. This approach could be used to engineer bacteria to sense the microenvironment of a tumor and respond by invading cancerous cells and releasing a cytotoxic agent.



nature Vol 440 13 April 2006 doi:10.1038/nature04640

LETTERS

Production of the antimalarial drug precursor artemisinic acid in engineered yeast

Dae-Kyun Ro[1]*, Eric M. Paradise[2]*, Mario Ouellet[1], Karl J. Fisher[6], Karyn L. Newman[1], John M. Ndungu[3], Kimberly A. Ho[1], Rachel A. Eachus[1], Timothy S. Ham[4], James Kirby[2], Michelle C. Y. Chang[1], Sydnor T. Withers[2], Yoichiro Shiba[2], Richmond Sarpong[3] & Jay D. Keasling[1,2,4,5]

Malaria is a global health problem that threatens 300–500 million people and kills more than one million people annually[1]. Disease control is hampered by the occurrence of multi-drug-resistant strains of the malaria parasite *Plasmodium falciparum*[2,3]. Synthetic antimalarial drugs and malarial vaccines are currently being developed, but their efficacy against malaria awaits rigorous clinical testing[4,5]. Artemisinin, a sesquiterpene lactone endoperoxide extracted from *Artemisia annua* L (family Asteraceae; commonly known as sweet wormwood), is highly effective against multi-drug-resistant *Plasmodium* spp., but is in short supply and unaffordable to most malaria sufferers[6]. Although total synthesis of artemisinin is difficult and costly[7], the semi-synthesis of artemisinin or any derivative from microbially sourced artemisinic acid, its immediate precursor, could be a cost-effective, environmentally friendly, high-quality and reliable source of artemisinin[8,9]. Here we report the engineering of *Saccharomyces cerevisiae* to produce high titres (up to 100 mg l^{-1}) of artemisinic acid using an engineered mevalonate pathway, amorphadiene synthase, and a novel cytochrome P450 monooxygenase (*CYP71AV1*) from *A. annua* that performs a three-step oxidation of amorpha-4,11-diene to artemisinic acid. The synthesized artemisinic acid is transported out and retained on the outside of the engineered yeast, meaning that a simple and inexpensive purification process can be used to obtain the desired product. Although the engineered yeast is already capable of producing artemisinic

To increase FPP production in *S. cerevisiae*, the expression of several genes responsible for FPP synthesis was upregulated, and one gene responsible for FPP conversion to sterols was downregulated. All of these modifications to the host strain were made by chromosomal integration to ensure the genetic stability of the host strain. Overexpression of a truncated, soluble form of 3-hydroxy-3-methylglutaryl-coenzyme A reductase (*tHMGR*)[12] improved amorphadiene production approximately fivefold (Fig. 2, strain EYP208). Downregulation of *ERG9*, which encodes squalene synthase (the first step after FPP in the sterol biosynthetic pathway), using a methionine-repressible promoter (P_{MET3})[13] increased amorphadiene production an additional twofold (Fig. 2, strain EPY225). Although *upc2-1*, a semi-dominant mutant allele that enhances the activity of *UPC2* (a global transcription factor regulating the biosynthesis of sterols in *S. cerevisiae*)[14], had only a modest effect on amorphadiene production when overexpressed in the EPY208 background (Fig. 2, strain EPY210), the combination of downregulating *ERG9* and overexpressing *upc2-1* increased amorphadiene production to 105 mg l^{-1} (Fig. 2, strain EPY213). Integration of an additional copy of *tHMGR* into the chromosome further increased amorphadiene production by 50% to 149 mg l^{-1} (Fig. 2, strain EPY219). Although overexpression of the gene encoding FPP synthase (*ERG20*) had little effect on total amorphadiene production (Fig. 2, strain EPY224), the specific production increased by about 10% owing to a decrease in cell density. Combining all of these modifi-

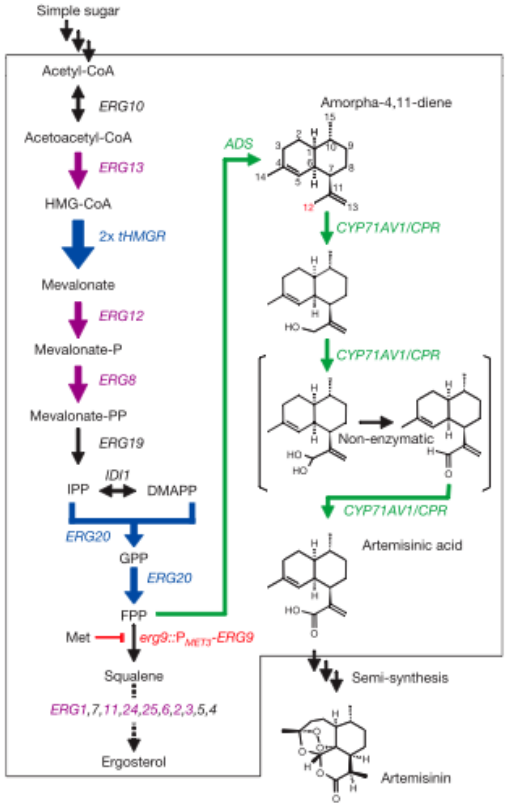
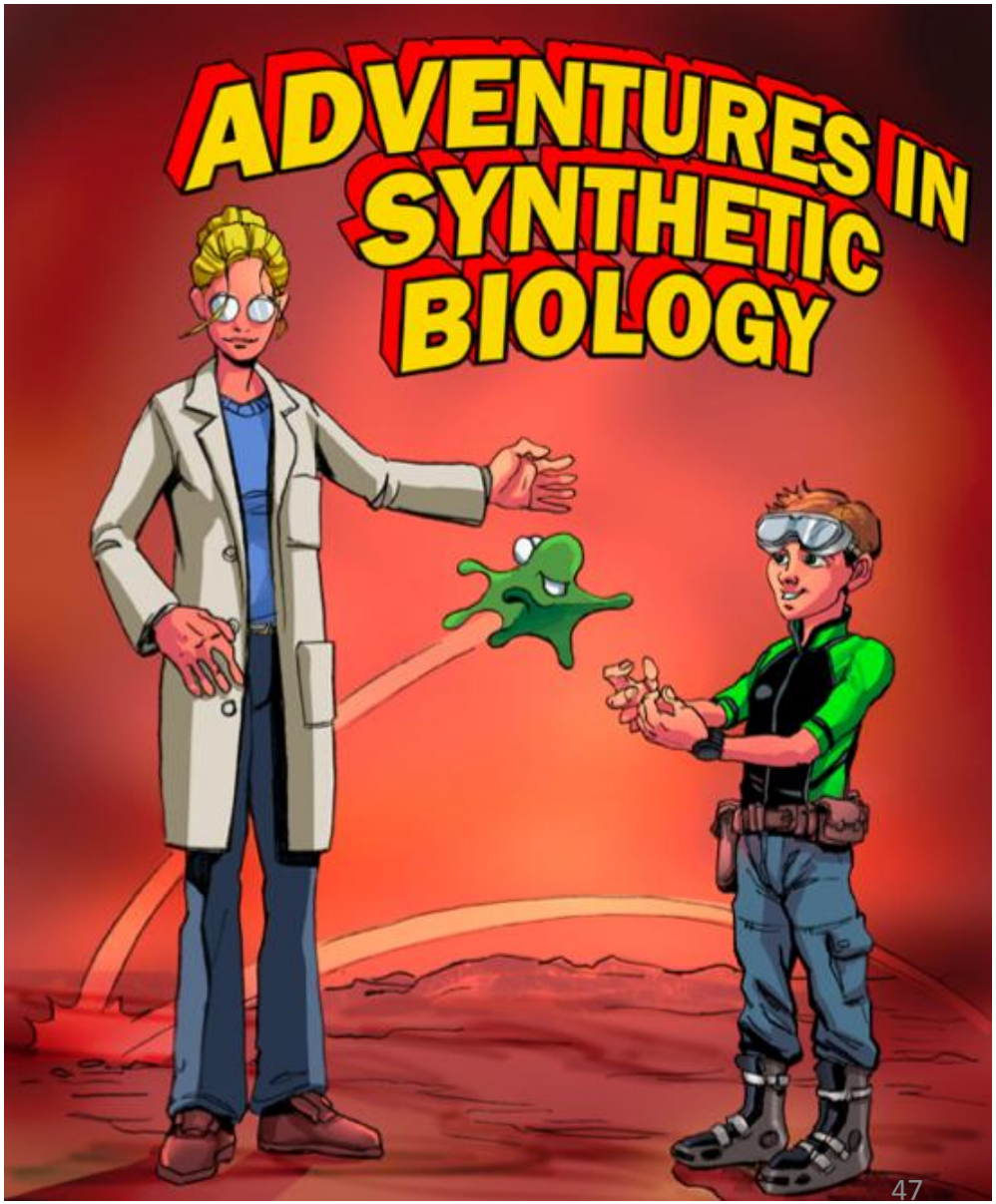


Figure 1 | Schematic representation of the engineered artemisinic acid biosynthetic pathway in *S. cerevisiae* strain EPY224 expressing *CYP71AV1*

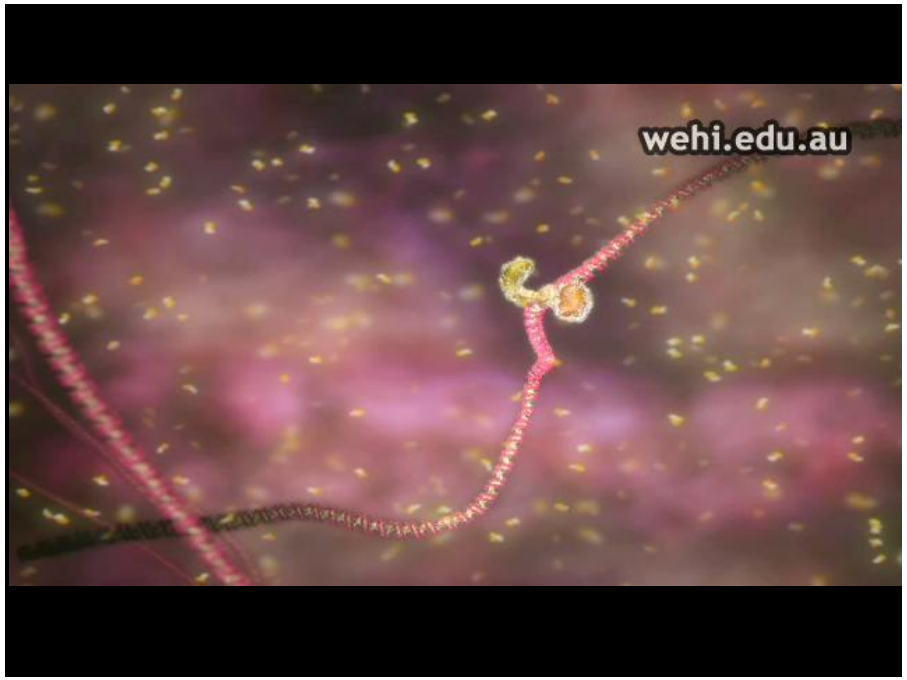
5. S. C. Eddy, J. D. Keasling, J. Castillon, and K. Cornish. 2008. "Initiator-independent and initiator-dependent"

The international Genetically Engineered Machine Competition



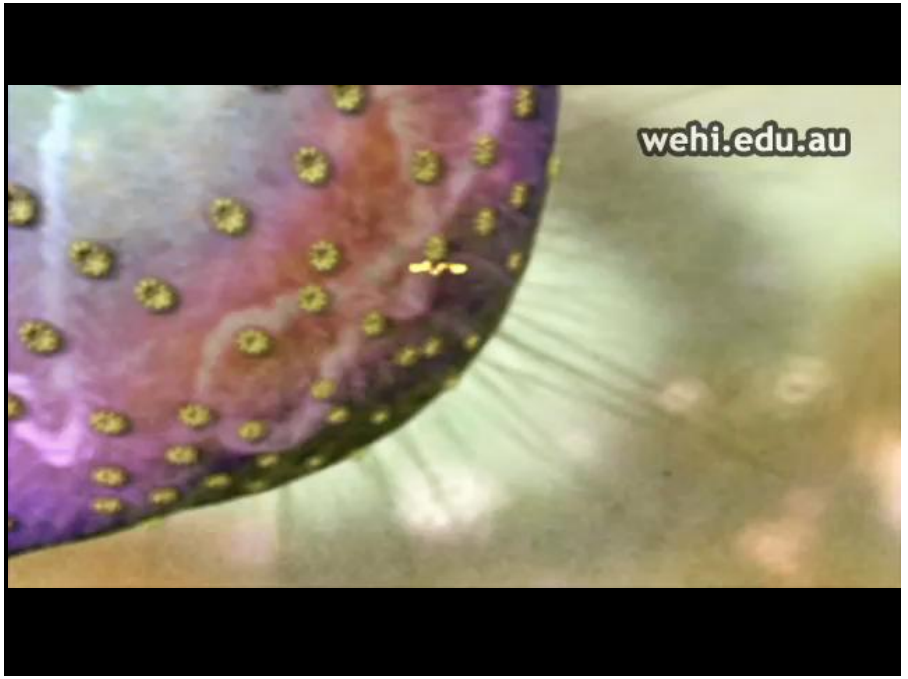
Programming Genetic Devices

DNA is a 4-letter digital code that tells a cell what proteins to make



DNA transcription in real time

RNA polymerase II: 15-30 base/second

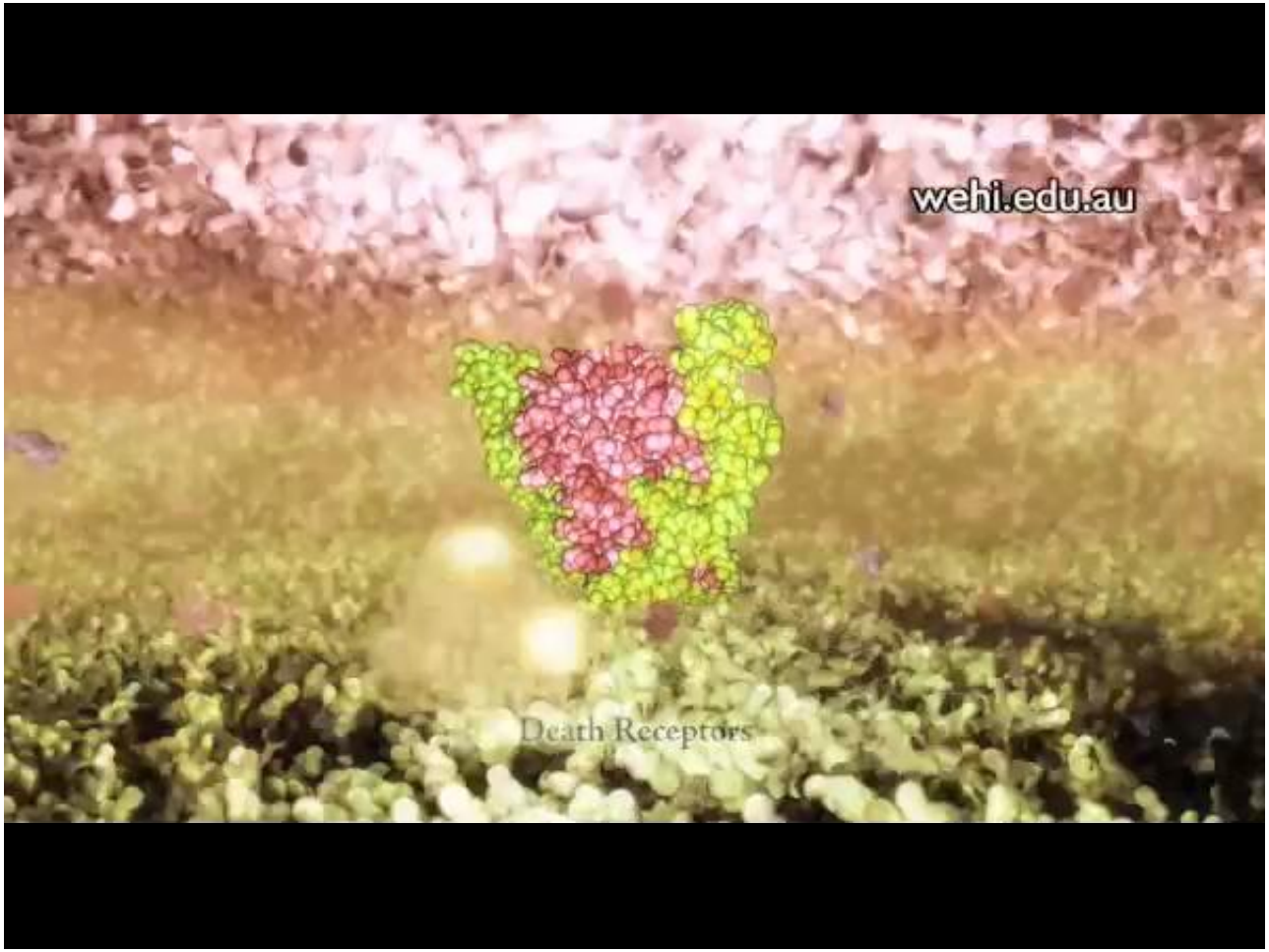


mRNA translation

Videos by Drew Berry http://www.wehi.edu.au/wehi-tv

Protein Information Processing

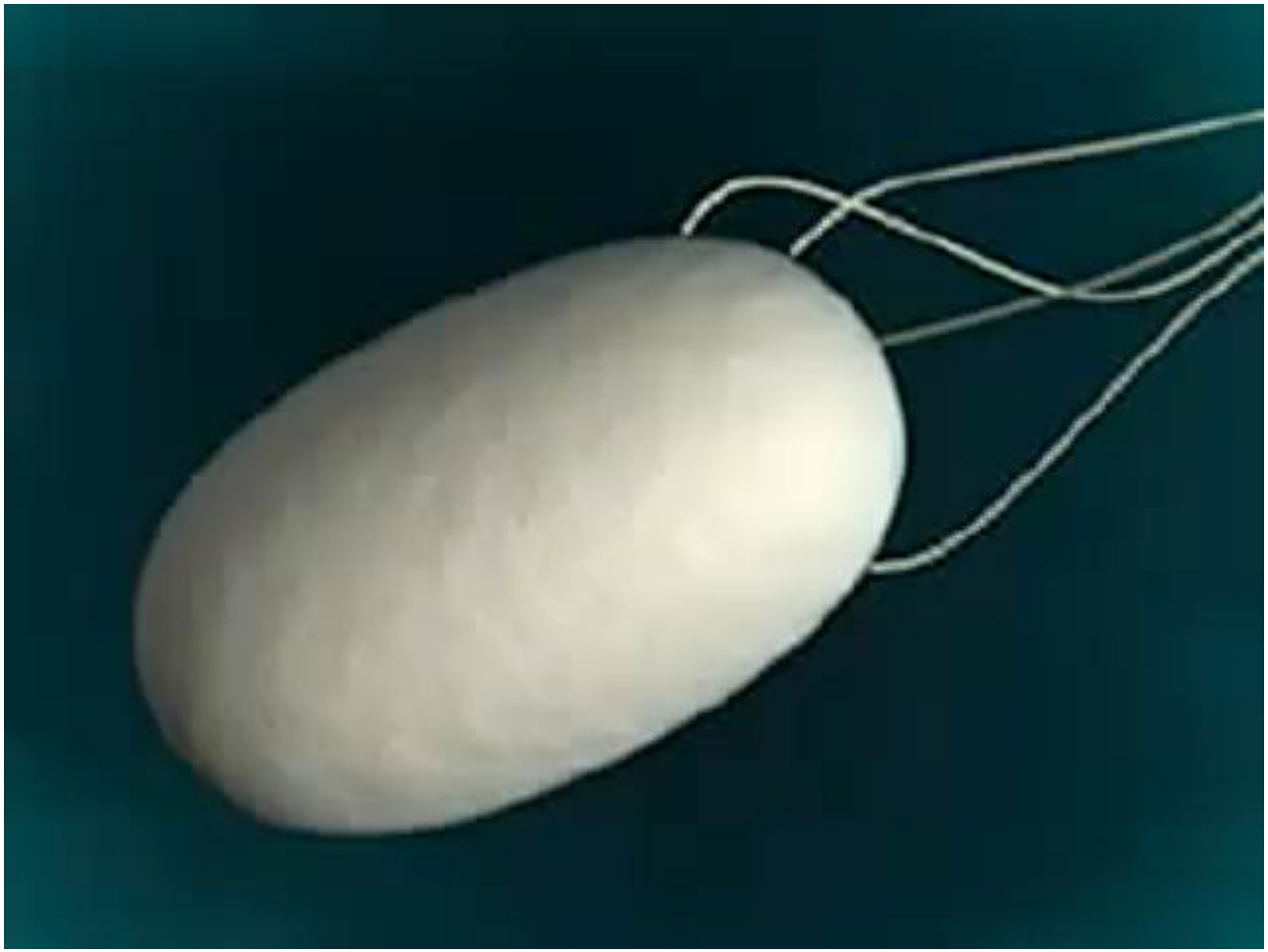
Proteins perform information processing for the cell



Videos by Drew Berry http://www.wehi.edu.au/wehi-tv

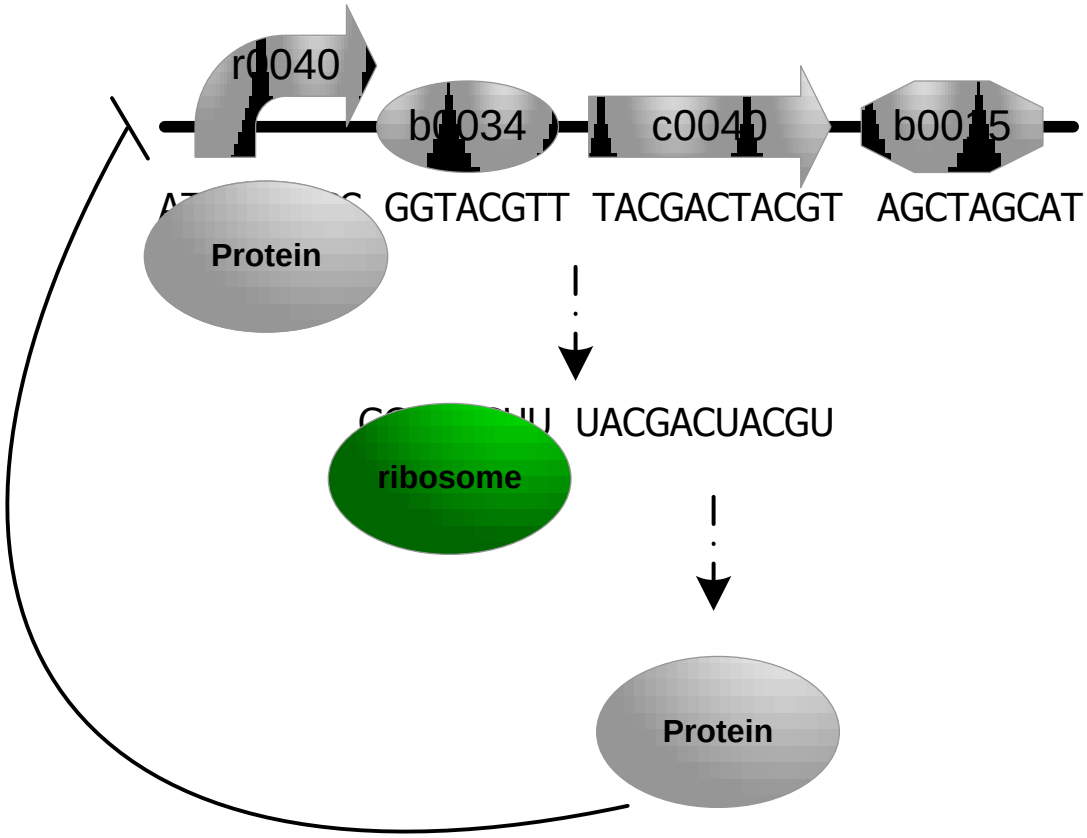
Programmed Self-Assembly

DNA codes for proteins that self-assemble



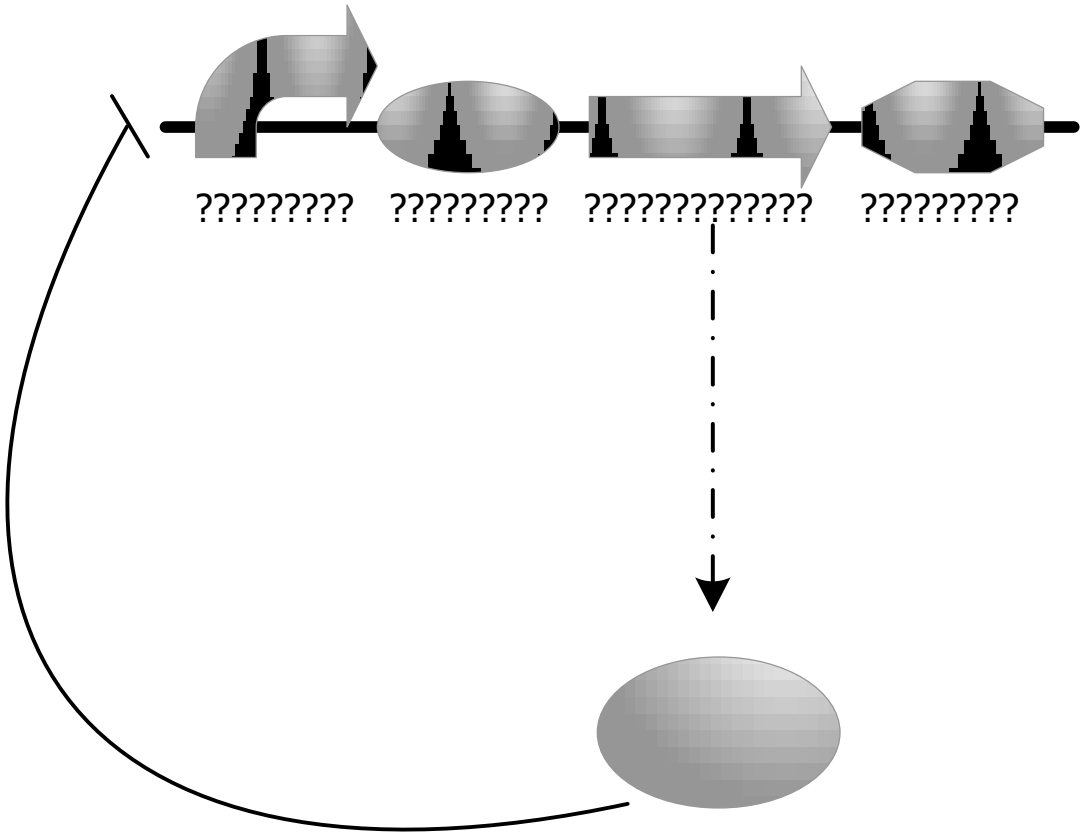
Executing DNA Machine Code

A simplified view of DNA instructions



Compiling Designs to DNA

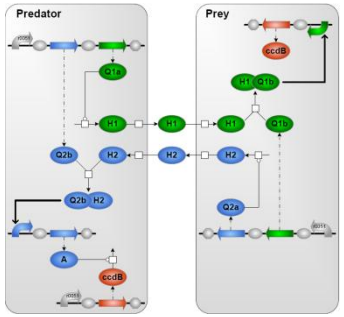
Given a design, automatically determine the DNA



Genetic Engineering of Cells (GEC)

Designing *DNA Software*: new instructions for cells

Step 1: Program device design



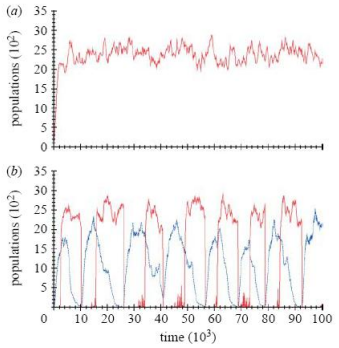

```
c1
[ r0051:prom; rbs; pcr<codes(Q2b)>
; rbs; pcr<codes(Q1a)>; ter
; prom<pos(Q2b-H2)>; rbs; pcr<prot(A)>; ter
; r0051:prom; rbs; pcr<prot(ccdB)>; ter
| Q1a ~ -> H1 | Q2b + H2 <-> Q2b-H2
| A ~ ccdB -> | ccdB ~ Q1a *->{10.0}
| H1 *->{10.0} | H2 *->{10.0}
] | |
c2
[ prom<pos(H1-Q1b)>; rbs; pcr<prot(ccdB)>; ter
; r0051:prom; rbs; pcr<codes(Q1b)>
; rbs; pcr<codes(Q2a)>; ter
| Q2a ~ -> H2 | H1 + Q1b <-> H1-Q1b
| ccdB ~ Q2a *->{10.0}
| H1 *->{10.0} | H2 *->{10.0}
] | |
c1[H1] -> H1 | H1 -> c2[H1]
| c2[H2] -> H2 | H2 -> c1[H2]
```

Step 2: Compile device behaviour

```
rate RMRNADeg = 0.001;
c1 [
init g47 1 |
mrna48 ->{RMRNADeg}  |
g47 ->{0.12} g47 + mrna48  |
mrna48 ->{0.1) mrna48 + luxR  |
mrna48 ->{0.1) mrna48 + lasI  |
lasI ~ ->{1} m3OC12HSL  |
m3OC12HSL ->{10}  |
m3OC6HSL ->{10}  |
luxR + m3OC6HSL ->{0.5) luxR-m3OC6HSL  |
luxR-m3OC6HSL ->{1) luxR + m3OC6HSL  |
init g92 1 |
mrna93 ->{RMRNADeg}  |
g92 ->{1e-6) g92 + mrna93  |
g92 + luxR-m3OC6HSL ->{1) g92-luxR-m3OC6HSL  |
g92-luxR-m3OC6HSL ->{0.8) g92 + luxR-m3OC6HSL  |
g92-luxR-m3OC6HSL ->{0.1) g92-luxR-m3OC6HSL + mrna93  |
mrna93 ->{0.1) mrna93 + ccdA  |
init g115 1 |
mrna116 ->{RMRNADeg}  |
g115 ->{0.12) g115 + mrna116  |
mrna116 ->{0.1) mrna116 + ccdB  |
ccdA ~ ccdB ->{1}  |
ccdB + lasI ->{10) ccdB
] |

c1 [
ccdA ->{0.1)  |
ccdB ->{0.005)  |
lasI ->{0.001)  |
lasR ->{0.001)  |
luxI ->{0.001)  |
luxR ->{0.001}
] |
```

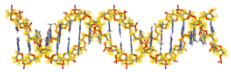
```
c2 [
init g147 1 |
mrna148 ->{RMRNADeg}  |
g147 ->{1e-6) g147 + mrna148  |
g147 + m3OC12HSL-lasR ->{1) g147-m3OC12HSL-lasR  |
g147-m3OC12HSL-lasR ->{0.8) g147 + m3OC12HSL-lasR  |
g147-m3OC12HSL-lasR ->{0.1) g147-m3OC12HSL-lasR + mrna148  |
mrna148 ->{0.1) mrna148 + ccdB  |
m3OC12HSL + lasR ->{0.5) m3OC12HSL-lasR  |
m3OC12HSL-lasR ->{1) m3OC12HSL + lasR  |
luxI ~ ->{1} m3OC6HSL  |
m3OC6HSL ->{10}  |
m3OC12HSL ->{10}  |
ccdB + luxI ->{10} ccdB  |
init g174 1 |
mrna175 ->{RMRNADeg}  |
g174 ->{0.12} g174 + mrna175  |
mrna175 ->{0.1) mrna175 + luxI  |
mrna175 ->{0.1) mrna175 + lasR
] |
c1[m3OC12HSL] ->{0.5} m3OC12HSL |
m3OC12HSL->{0.5} c2[m3OC12HSL] |
c2[m3OC6HSL] ->{0.5} m3OC6HSL |
m3OC6HSL->{0.5} c1[m3OC6HSL] |

c2 [
ccdA ->{0.1)  |
ccdB ->{0.005)  |
lasI ->{0.001)  |
lasR ->{0.001)  |
luxI ->{0.001)  |
luxR ->{0.001)
] |

ccdA ->{0.1)  |
ccdB ->{0.005)  |
lasI ->{0.001)  |
lasR ->{0.001)  |
luxI ->{0.001)  |
luxR ->{0.001)
```

Step 3: Simulate device

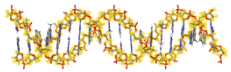


Step 4: Compile device to DNA

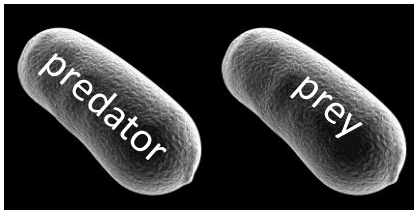
predator



prey

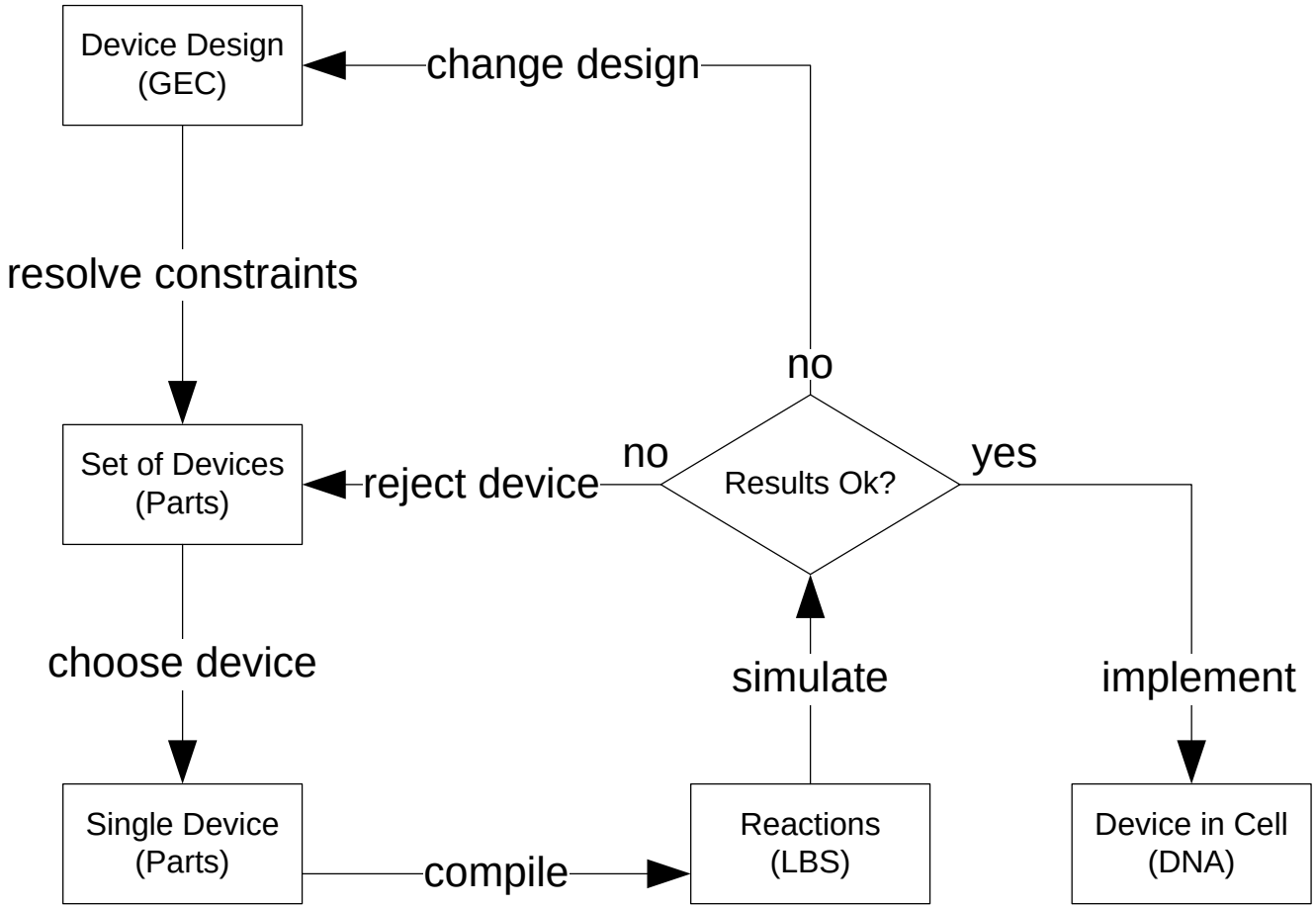


Step 5: Insert DNA into cells

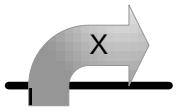


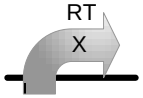
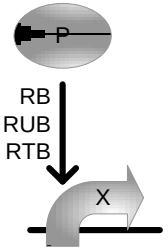
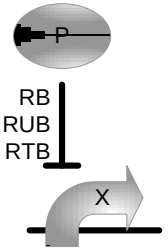
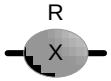
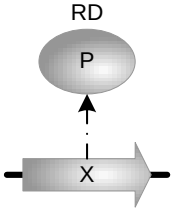
Pedersen & Phillips. Royal Society Interface, 2009

GEC Development Cycle



GEC Language: Parts and Properties

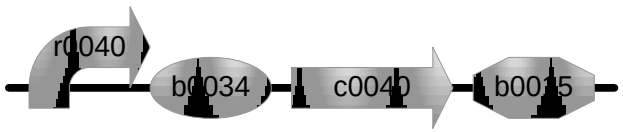
Part	Representation
X:prom	
X:rbs	
X:pcr	
X:ter	

Part Property	Representation
X:prom<con(RT)>	
X:prom<pos(P,RB,RUB,RTB)>	
X:prom<neg(P,RB,RUB,RTB)>	
X:rbs<rate(R)>	
X:pcr<codes(P,RD)>	

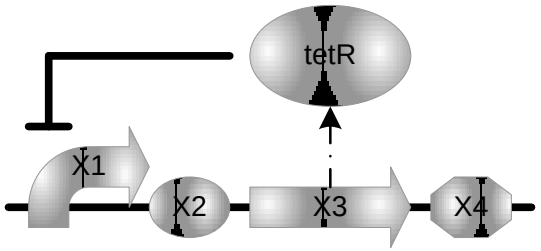
GEC Parts Database

ID	Type	Properties
i723017	pcr	codes(xylR, 0.001)
i723024	pcr	codes(phzM, 0.001)
i723025	pcr	codes(phzS, 0.001)
i723028	pcr	codes(pca, 0.001)
c0051	pcr	codes(cI, 0.001)
c0040	pcr	codes(tetR, 0.001)
c0080	pcr	codes(araC, 0.001)
c0012	pcr	codes(lacI,0.001)
cunknown2	pcr	codes(unknown2, 0.001)
c0061	pcr	codes(luxI,0.001)
c0062	pcr	codes(luxR,0.001)
c0079	pcr	codes(lasR,0.001)
c0078	pcr	codes(lasI,0.001)
cunknown3	pcr	codes(ccdB,0.005)
cunknown4	pcr	codes(ccdA, 0.1)
i723020	prom	pos(toluene-xylR, 0.001, 0.001, 1.0), con(0.0001)
r0051	prom	neg(cI, 1.0, 0.5, 0.00005), con(0.12)
r0040	prom	neg(tetR, 1.0, 0.5, 0.00005), con(0.09)
runknown1	prom	neg(unknown1, 1.0, 0.005, 0.001), con(0.04)
i0500	prom	neg(araC, 1.0, 0.000001, 0.0001), con(0.1)
r0011	prom	neg(lacI, 1.0, 0.5, 0.00005), con(0.1)
runknown2	prom	pos(lasR-m3OC12HSL, 1.0, 0.8, 0.1), pos(luxR-m3OC6HSL, 1.0, 0.8, 0.1), con(0.000001)
b0034	rbs	rate(0.1)
b0015	ter	
cunknown5	pcr	codes(ccdA2, 10.0)
runknown5	prom	con(10.0)

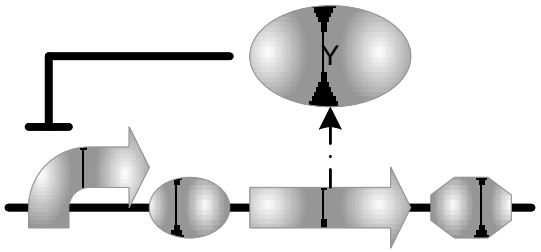
Compiling GEC Design to Parts



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



X1:prom<neg(tetR)>; X2:rbs; X3:pcr<codes(tetR)>; X4:ter



prom<neg(Y)>; rbs; pcr<codes(Y)>; ter

- Specific set of parts:

  [r0040; b0034; c0040; b0015]

- tetR negative feedback

  [r0040; b0034; c0040; b0015]

- Any negative feedback:

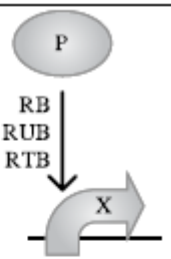
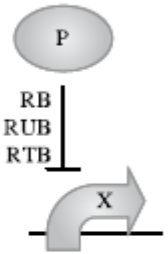
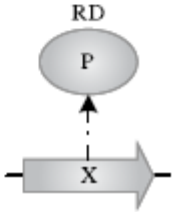
  [r0051; b0034; c0051; b0015]

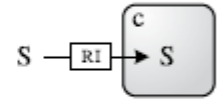
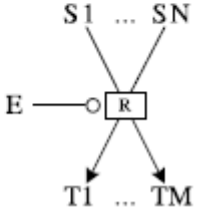
  [r0040; b0034; c0040; b0015]

  [i0500; b0034; c0080; b0015]

  [r0011; b0034; c0012; b0015]

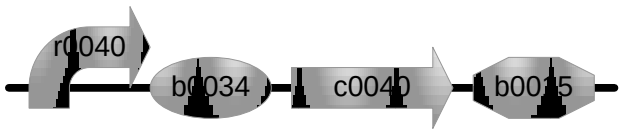
GEC Language: Reactions

part property & reactions	representation
X:prom<pos(P,RB,RUB,RTB)> g + P →{RB} g-P g-P →{RUB} g + P g-P →{RTB} g-P + m	
X:prom<neg(P,RB,RUB,RTB)> g + P →{RB} g-P g-P →{RUB} g + P g-P →{RTB} g-P + m	
X:prom<con(RT)> g →{RT} g + m m →{rdm}	
X:rbs<rate(R)> m →{R} m + p	
X:pcr<codes(P,RD)> p →{RD}	

reaction	representation
c[S] →{RO} S	
S →{RI} c[S]	
E ~ S1 + ... + SN →{R} T1 + ... + TM	

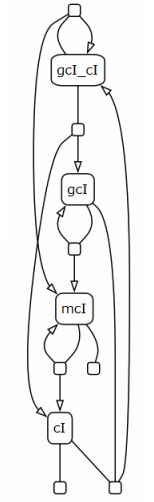
luxR + m3OC6HSL → {1.0} luxR−m3OC6HSL
luxR−m3OC6HSL → {1.0} luxR + m3OC6HSL
lasR + m3OC12HSL → {1.0} lasR−m3OC12HSL
lasR−m3OC12HSL → {1.0} lasR + m3OC12HSL
luxI ~ → {1.0} m3OC6HSL
lasI ~ → {1.0} m3OC12HSL
ccdA ~ ccdB → {1.0}
ccdA2 ~ ccdB → {0.00001}
m3OC6HSL → {1.0} [m3OC6HSL]
m3OC12HSL → {1.0} [m3OC12HSL]
[m3OC6HSL] → {1.0} m3OC6HSL
[m3OC12HSL] → {1.0} m3OC12HSL

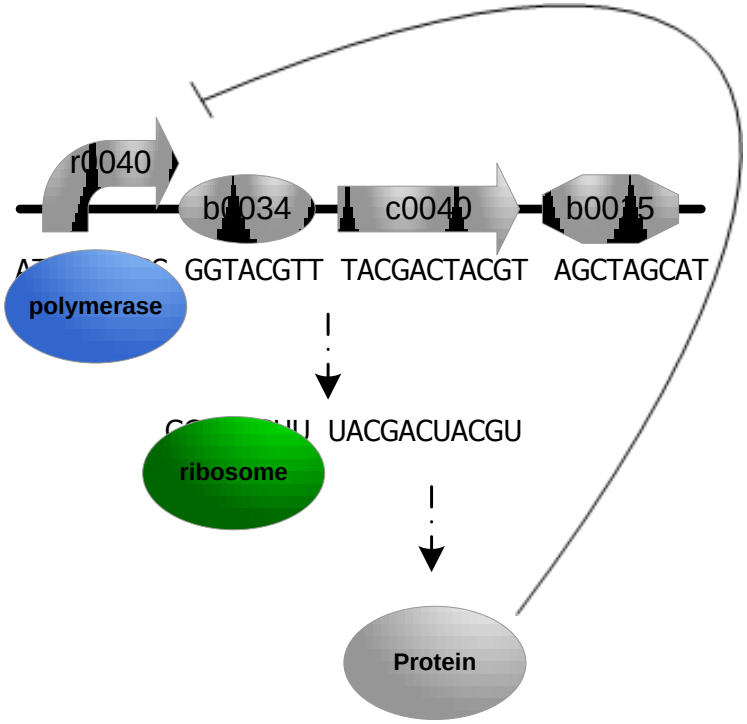
Compiling GEC Parts to Reactions

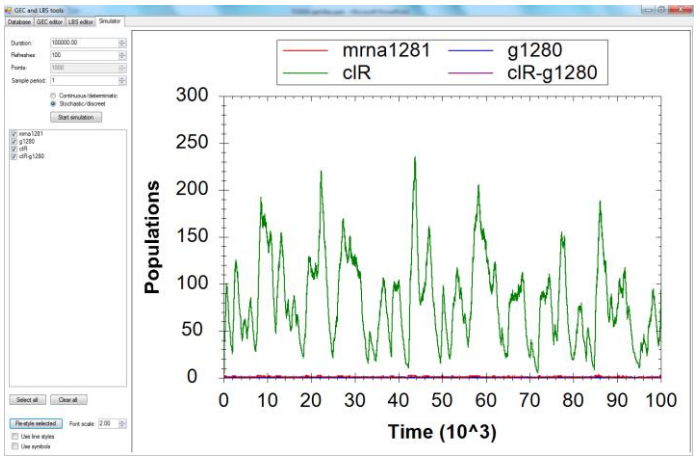


r0040:prom; b0034:rbs; c0040:pcr; b0015:ter

```
rate mrnaDeg = 0.001;
init gcI 1
| gcI ->{0.12} gcI + mcI
| gcI + cI ->{1} gcI_cI
| gcI_cI ->{0.5} gcI + cI
| gcI_cI ->{5e-5} gcI_cI + mcI
| mcI ->{0.1} mcI + cI
| mcI ->{mrnaDeg}
| cI ->{0.001}
```

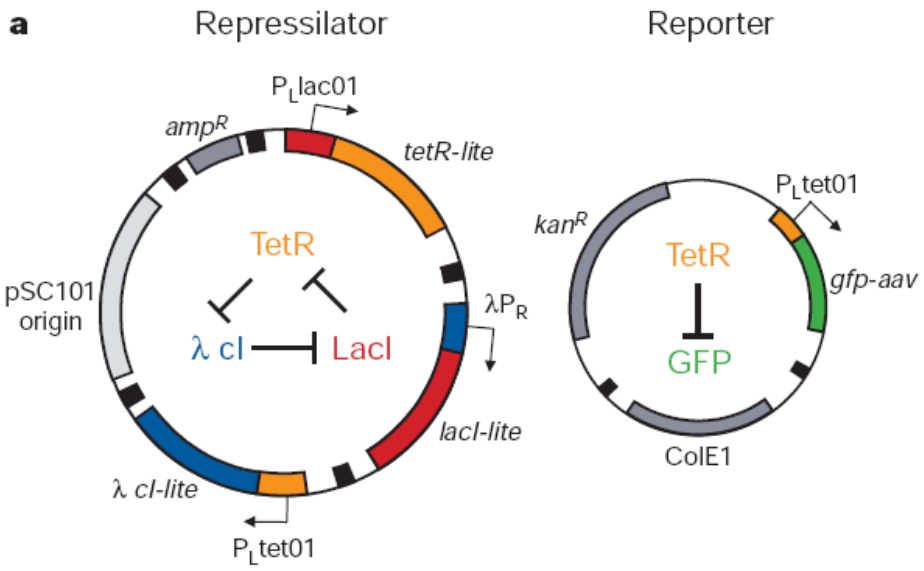


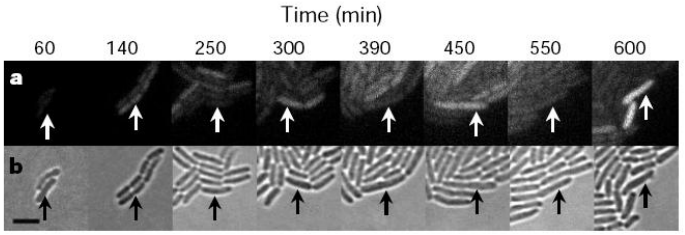




Example: Repressilator

- A gene network engineered in live bacteria.

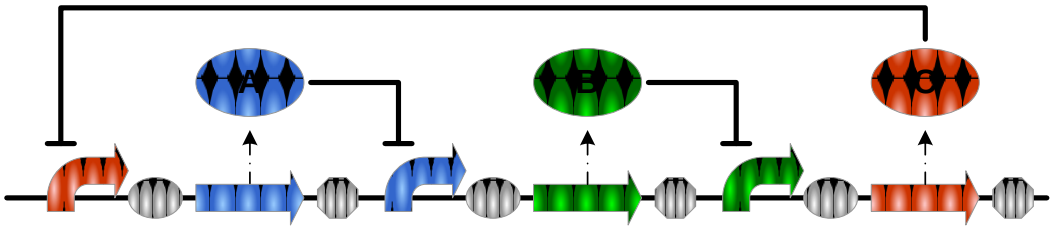




Engineered by Elowitz et al., Nature 2000

Example: Repressilator

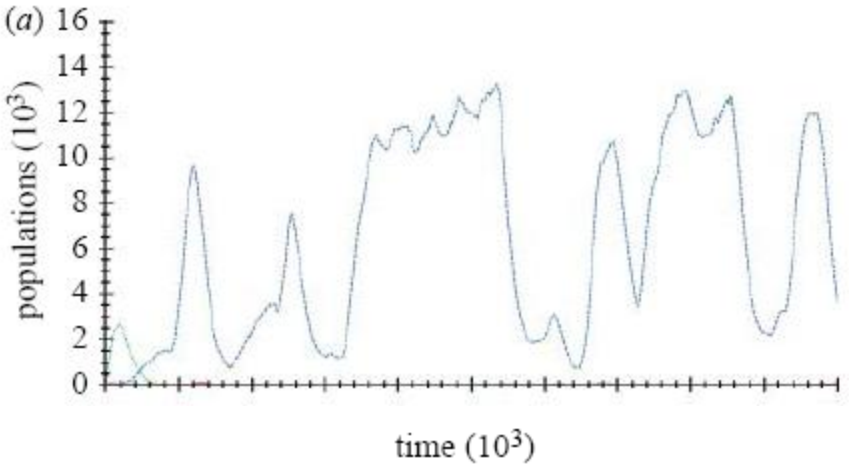
Repressilator Design



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

24 possible solutions, many of them defective, e.g.

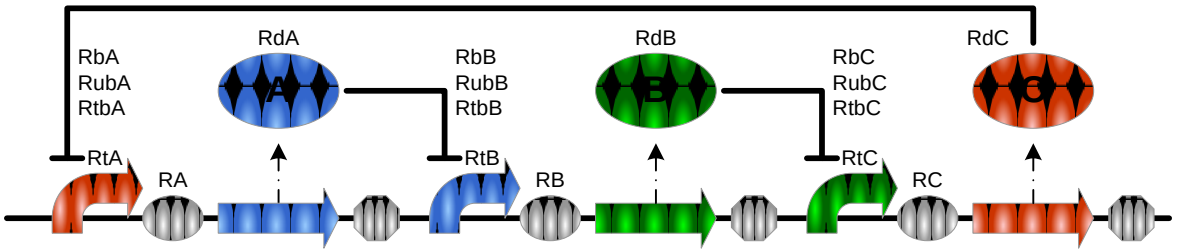
[r0051; b0034; c0040; b0015
; r0040; b0034; c0080; b0015
; i0500; b0034; c0051; b0015]



Engineered by Elowitz et al., Nature 2000

Case Study: Repressilator

Add constraints on rates, e.g. promoter strength

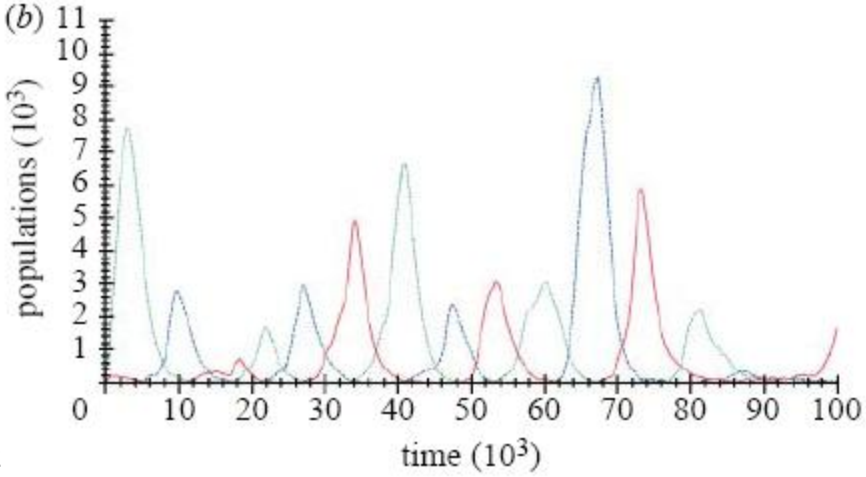


| 0.4 < RubB | RubB < 0.6
| 0.4 < RubC | RubC < 0.6
| 0.4 < RubA | RubA < 0.6

prom<con(RtA),neg(C,RbA,RubA,RtbA)>; rbs<rate(RA)>; pcr<codes(A,RdA)>; ter;
prom<con(RtB),neg(A,RbB,RubB,RtbB)>; rbs<rate(RB)>; pcr<codes(B,RdB)>; ter;
prom<con(RtC),neg(B,RbC,RubC,RtbC)>;rbs<rate(RC)>; pcr<codes(C,RdC)>; ter

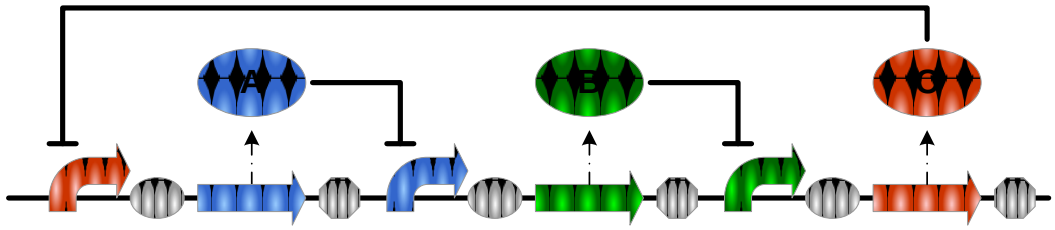
Selects functional Repressilator

[r0040; b0034; c0051; b0015
; r0051; b0034; c0012; b0015
; r0011; b0034; c0040; b0015]

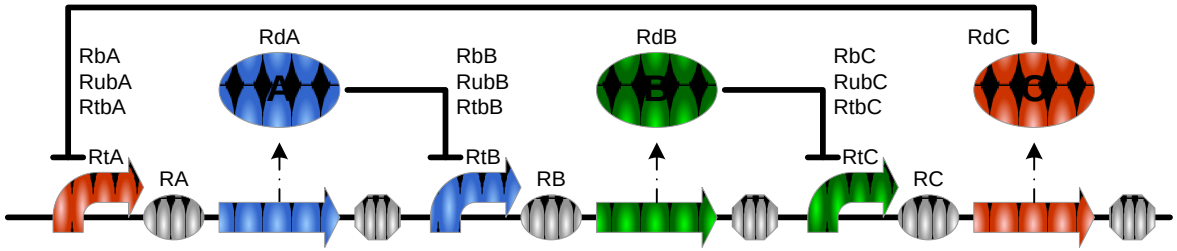


Case Study: Repressilator

Definition of modules



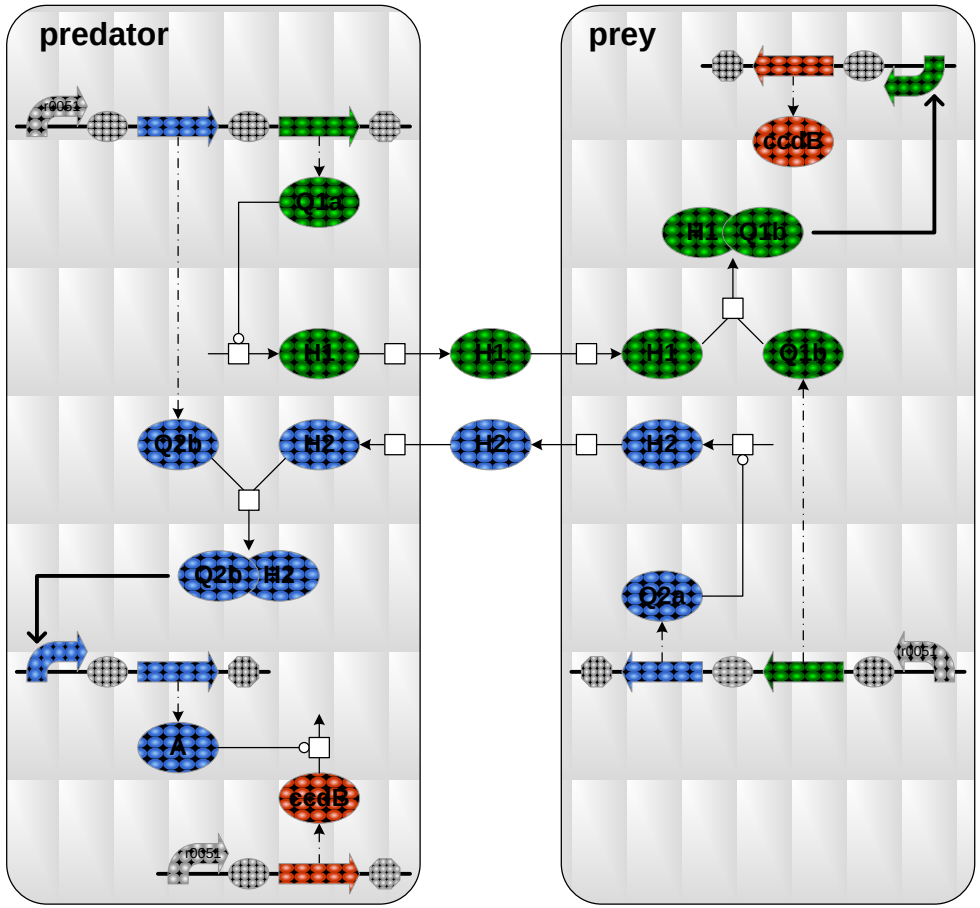

```
module gate(i,o) {
  prom<neg(i)>; rbs; pcr<codes(o)>; ter
};
gate(A,B) | gate(B,C) | gate(C,A)
```




```
module gate(i,o) {
  new RB. new RUB. new RTB. new RT. new R. new RD.
  prom<con(RT), neg(i, RB, RUB, RTB)>;rbs<rate(R)>; pcr<codes(o,RD)>; ter
  | 0.4 < RUB | RUB < 0.6
};
gate(A,B) | gate(B,C) | gate(C,A)
```

Case Study: Predator Prey

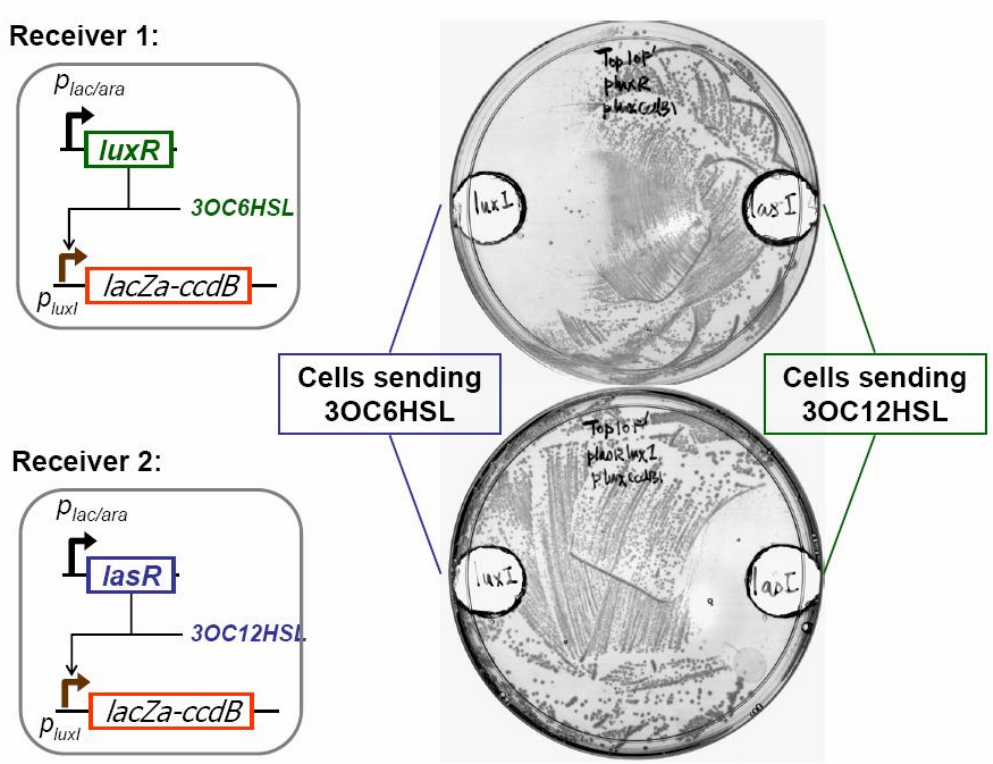
Specified in GEC as follows

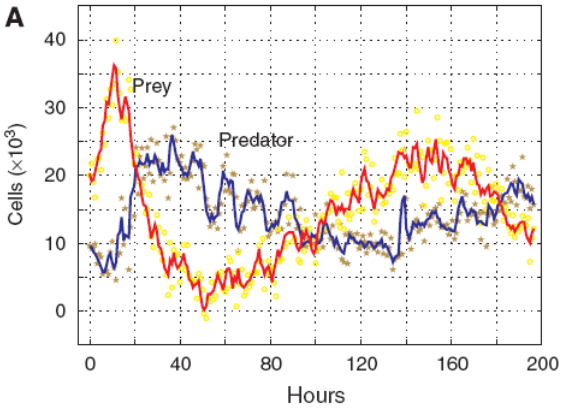


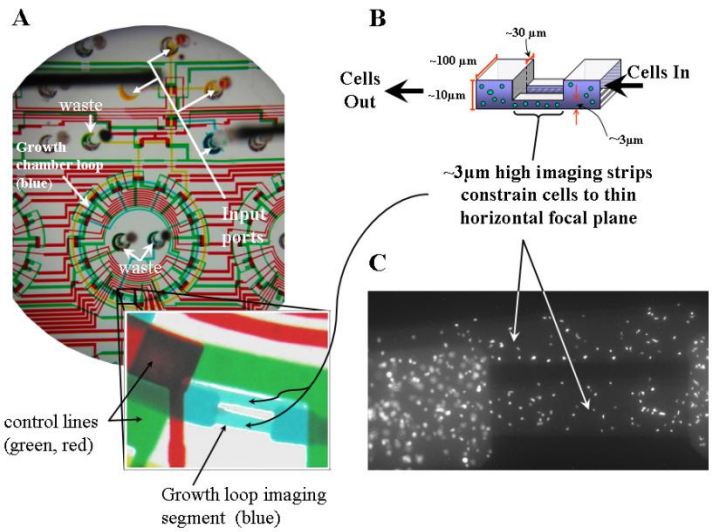
predator
[r0051:prom; rbs; pcr<codes(Q2b)>
; rbs; pcr<codes(Q1a)>; ter
; prom<pos(Q2b-H2)>; rbs; pcr<codes(A)>; ter
; r0051:prom; rbs; pcr<codes(ccdB)>; ter
| Q1a ~-> H1 | Q2b + H2 <-> Q2b-H2
| A ~ccdB ->
]
||
prey
[prom<pos(H1-Q1b)>; rbs; pcr<codes(ccdB)>; ter
; r0051:prom; rbs; pcr<codes(Q1b)>
; rbs; pcr<codes(Q2a)>; ter
| Q2a ~-> H2 | H1 + Q1b <-> H1-Q1b
]
||predator[H1] -> H1 | H1 -> prey[H1]
| prey[H2] -> H2 | H2 -> predator[H2]

Engineered by Balagadde et al., MSB 2008

Case Study: Predator Prey





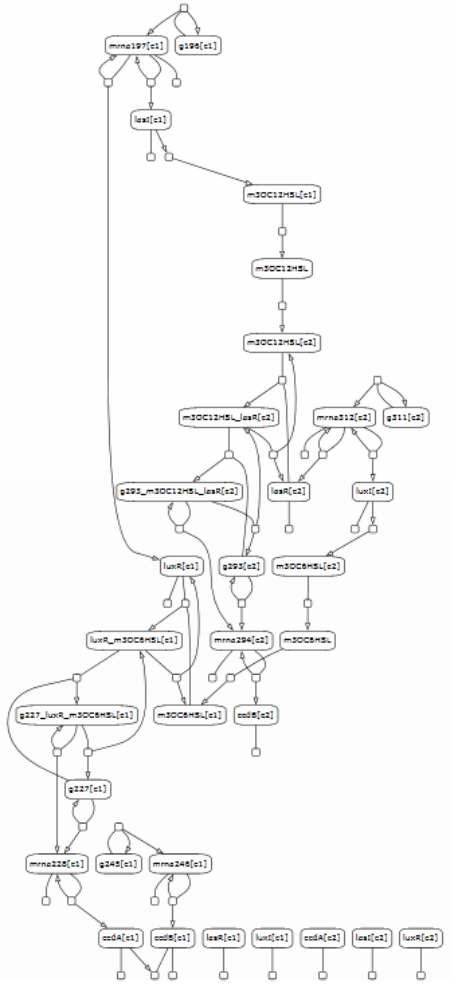


Engineered by Balagadde et al., MSB 2008

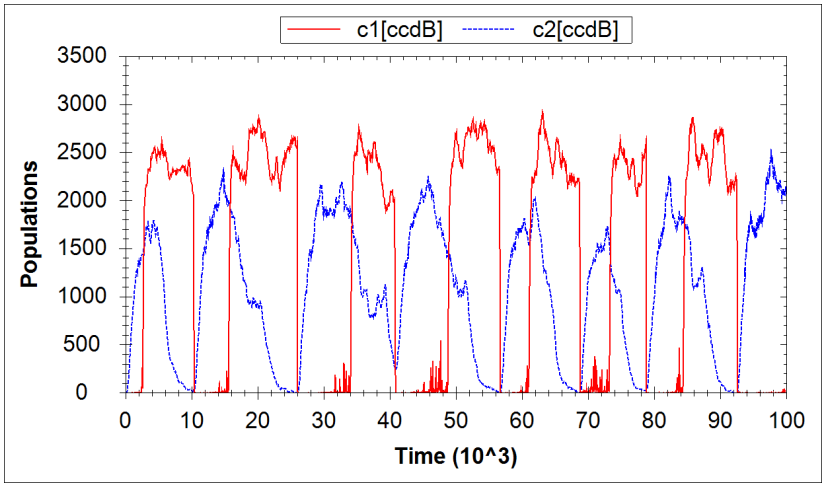
Case Study: Predator Prey

Compile to parts to reactions and simulate

```
rate RMRNADeg = 0.001;
c1 [
init g47 1 |
mrna48 ->{RMRNADeg}  |
g47 ->{0.12} g47 + mrna48  |
mrna48 ->{0.1} mrna48 + luxR  |
mrna48 ->{0.1} mrna48 + lasI  |
lasI ~  ->{1} m3OC12HSL  |
m3OC12HSL ->{10}  |
m3OC6HSL ->{10}  |
luxR + m3OC6HSL ->{0.5} luxR-m3OC6HSL  |
luxR-m3OC6HSL ->{1} luxR + m3OC6HSL  |
init g92 1 |
mrna93 ->{RMRNADeg}  |
g92 ->{1e-6} g92 + mrna93  |
g92 + luxR-m3OC6HSL ->{1} g92-luxR-m3OC6HSL  |
g92-luxR-m3OC6HSL ->{0.8} g92 + luxR-m3OC6HSL  |
g92-luxR-m3OC6HSL ->{0.1} g92-luxR-m3OC6HSL +
mrna93  |
mrna93 ->{0.1} mrna93 + ccdA  |
init g115 1 |
mrna116 ->{RMRNADeg}  |
g115 ->{0.12} g115 + mrna116  |
mrna116 ->{0.1} mrna116 + ccdB  |
ccdA ~ ccdB ->{1}  |
ccdB + lasI ->{10} ccdB
] |

c1 [
ccdA ->{0.1}  |
ccdB ->{0.005}  |
lasI ->{0.001}  |
lasR ->{0.001}  |
luxI ->{0.001}  |
luxR ->{0.001}
] |

c2 [
init g147 1 |
mrna148 ->{RMRNADeg}  |
g147 ->{1e-6} g147 + mrna148  |
g147 + m3OC12HSL-lasR ->{1} g147-m3OC12HSL-lasR  |
g147-m3OC12HSL-lasR ->{0.8} g147 + m3OC12HSL-lasR  |
g147-m3OC12HSL-lasR ->{0.1} g147-m3OC12HSL-lasR +
mrna148  |
mrna148 ->{0.1} mrna148 + ccdB  |
m3OC12HSL + lasR ->{0.5} m3OC12HSL-lasR  |
m3OC12HSL-lasR ->{1} m3OC12HSL + lasR  |
luxI ~  ->{1} m3OC6HSL  |
m3OC6HSL ->{10}  |
m3OC12HSL ->{10}  |
ccdB + luxI ->{10} ccdB  |
init g174 1 |
mrna175 ->{RMRNADeg}  |
g174 ->{0.12} g174 + mrna175  |
mrna175 ->{0.1} mrna175 + luxI  |
mrna175 ->{0.1} mrna175 + lasR
] |
c1[m3OC12HSL] ->{0.5} m3OC12HSL |
m3OC12HSL->{0.5} c2[m3OC12HSL] |
c2[m3OC6HSL] ->{0.5} m3OC6HSL |
m3OC6HSL->{0.5} c1[m3OC6HSL] |

c2 [
ccdA ->{0.1}  |
ccdB ->{0.005}  |
lasI ->{0.001}  |
lasR ->{0.001}  |
luxI ->{0.001}  |
luxR ->{0.001}
]|

ccdA ->{0.1}  |
ccdB ->{0.005}  |
lasI ->{0.001}  |
lasR ->{0.001}  |
luxI ->{0.001}  |
luxR ->{0.001}
```



r0051; b0034; c0062; b0034; c0078; b0015;
runknown2; b0034; cunknown4; b0015;
r0051; b0034; cunknown3; b0015
||
runknown2; b0034; cunknown3; b0015;
r0051;b0034;c0061;b0034;c0079;b0015



GEC Calculus

$P ::=$	$u{:}t(Q^t)$	part u of type t with properties Q^t
	$\mid \mathbf{0}$	empty program
	$\mid p(\tilde{u})\{P_1\}; P_2$	definition of module p with formals $\tilde{u}$
	$\mid p(\tilde{A})$	invocation of module p with actuals $\tilde{A}$
	$\mid P \mid C$	constraint C associated to program P
	$\mid P_1 \| P_2$	parallel composition of P_1 and P_2
	$\mid P_1 ; P_2$	sequential composition of P_1 and P_2
	$\mid c[P]$	compartment c containing program P
	$\mid$ new $x.P$	local variable x inside program P
$C ::=$	R	reaction
	$\mid T$	transport reaction
	$\mid K$	numerical constraint
	$\mid C_1 \mid C_2$	conjunction of C_1 and C_2
$R ::=$	$S \overset{r}{\sim} \sum m_i \cdot S_j \to^r \sum m_i \cdot S_j$	reactants S_i, products S_j
$T ::=$	$S \to^r c[S]$	transport of S into compartment c
	$\mid c[S] \to^r S$	transport of S out of compartment c
$K ::=$	$E_1 > E_2$	expression E_1 greater than E_2
$E ::=$	r	real number or variable
	$\mid E_1 \otimes E_2$	arithmetic operation $\otimes$ on E_1 and E_2
$A ::=$	r	real number or variable
	$\mid S$	species

(i) $[\![u : t(Q^t)]\!] \triangleq (\{(u)\}, \Theta)$, where

$$\Theta = \{(\theta_i, \rho_i, \sigma_i, \mathrm{FS}(Q_i) \setminus \sigma_i) \mid u\theta_i : t(Q_i) \in \mathcal{K}_b,\ Q^t\theta_i \subseteq Q_i,\ \mathrm{Dom}(\theta_i) = \mathrm{FV}(u : t(Q^t)),\ \rho_i = \mathrm{Dom_s}(\theta_i), \sigma_i = \mathrm{FS}(Q^t\theta_i)\}.$$

(ii) $[\![P|C]\!] \triangleq (\Delta, \Theta_1 \bigtriangledown \Theta_2)$, where

$$(\Delta, \Theta_1) = [\![P]\!] \quad \text{and} \quad \Theta_2 = [\![C]\!].$$

(iii) $[\![P_1 \| P_2]\!] \triangleq (\Delta_1 \cup \Delta_2, \Theta_1 \bigtriangledown \Theta_2)$, where

$$(\Delta_1, \Theta_1) = [\![P_1]\!] \quad \text{and} \quad (\Delta_2, \Theta_2) = [\![P_2]\!].$$

(iv) $[\![P_1; P_2]\!] \triangleq (\{\delta_{1_i}\delta_{2_j}\}_{I\times J}, \Theta_1 \bigtriangledown \Theta_2)$, where

$$(\{\delta_{1_i}\}_I, \Theta_1) = [\![P_1]\!] \quad \text{and} \quad (\{\delta_{2_j}\}_J, \Theta_2) = [\![P_2]\!].$$

(v) $[\![c[P]]\!] \triangleq (\Delta, \{(\theta, \emptyset, \emptyset, \emptyset) \mid (\theta, \rho, \sigma, \tau) \in \Theta\})$, where $(\Delta, \Theta) = [\![P]\!]$.

(vi) $[\![R]\!] \triangleq \{(\theta_i, \mathrm{Dom_s}(\theta_i), \mathrm{FS}(R\theta_i), \emptyset) \mid R\theta_i \in \mathcal{K}_r, \mathrm{Dom}(\theta_i) = \mathrm{FV}(R)\}$.

Proposition 3.1 (piecewise injectivity). *For any context $\mathcal{C}(\bullet)$ and any compartment-free program P with $\mathrm{FP}(P)=\emptyset$, $[\![\mathcal{C}(P)]\!] = \Delta\{(\theta_i, \rho_i, \sigma_i, \tau_i)\}$, it holds that θ_i is injective on the domain $\mathrm{FV}(P) \cap \mathrm{Dom_s}(\theta_i)$.*

Proposition 3.2 (non-interference). *For any basic program $P = u : t(Q^t)$ and any compartment-free context $\mathcal{C}(\bullet)$ with $[\![\mathcal{C}(P)]\!] = \Delta\{(\theta_i, \rho_i, \sigma_i, \tau_i)\}$, it holds that $u\theta_i : t(Q) \in \mathcal{K}_b$ for some Q and $\sigma_i \cap (\mathrm{FS}(Q) \setminus \mathrm{FS}(Q^t\theta_i)) = \emptyset$.*

(i) $\theta : X \hookrightarrow N_\mathrm{s} \cup N_\mathrm{p} \cup \mathbb{R}$ is a finite, partial function (the substitution).

(ii) $\rho \subseteq X$ is a set of variables over which θ is injective, i.e. $\forall\, x_1, x_2 \in \rho.\ (x_1 \neq x_2) \Rightarrow (\theta(x_1) \neq \theta(x_2))$.

(iii) $\sigma, \tau \subseteq \mathcal{S}$ are, respectively, the species names that have been used in the current context and the species names that are excluded for use, and $\sigma \cap \tau = \emptyset$.

GEC Compilation Algorithm

$$L ::= R \;:\; T \;:\; \mathbf{0} \;:\; L_1|L_2 \;:\; c[L]$$

Given a set $\mathcal{L}$ of LBS models, we let (par $\mathcal{L}$) denote their parallel composition; this operator is commutative, so the order is insignificant. With the above motivation in mind, the translation function takes the following form:

$$[\![P]\!]_\Gamma = (L, D, M, Pr, F, G, H)$$

where

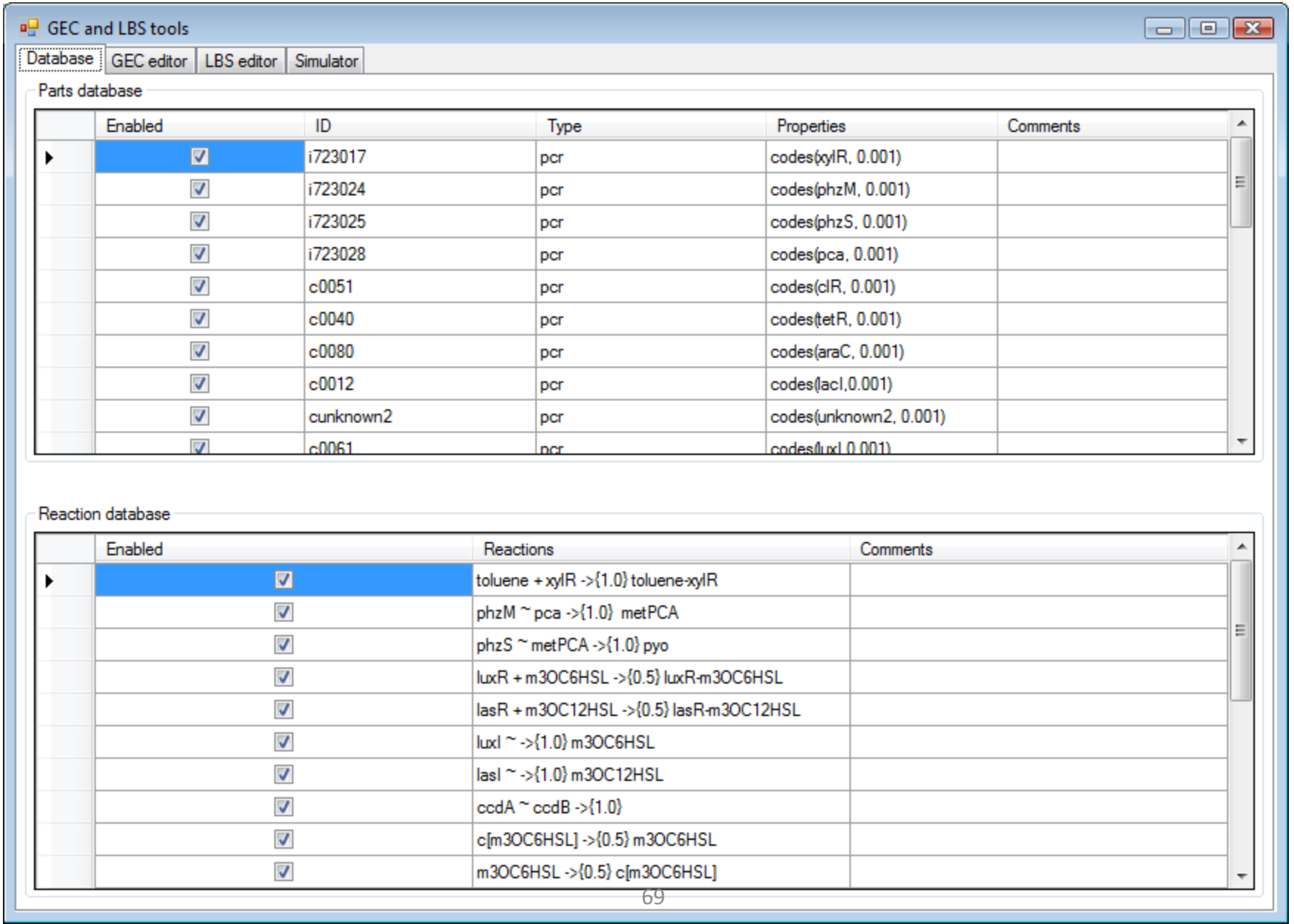
- L is an LBS program.
- D is a set of degradation reactions.
- $M \subset U$ is a set of mRNA names.
- $Pr \subset U$ is a set of protein names.
- F is a function of the form $f(m,p) = R$ mapping pairs $(m,p) \in U \times U$ of mRNA and protein names to a reaction.
- G is a function of the form $g(m) = R$ mapping an mRNA species name $m \in U$ to a reaction.
- H is a function of the form $h(p) = R$ mapping a protein name $p \in U$ to a reaction.

The translation is defined inductively on GEC programs as follows, where we again assume a global mRNA degradation rate rdm.

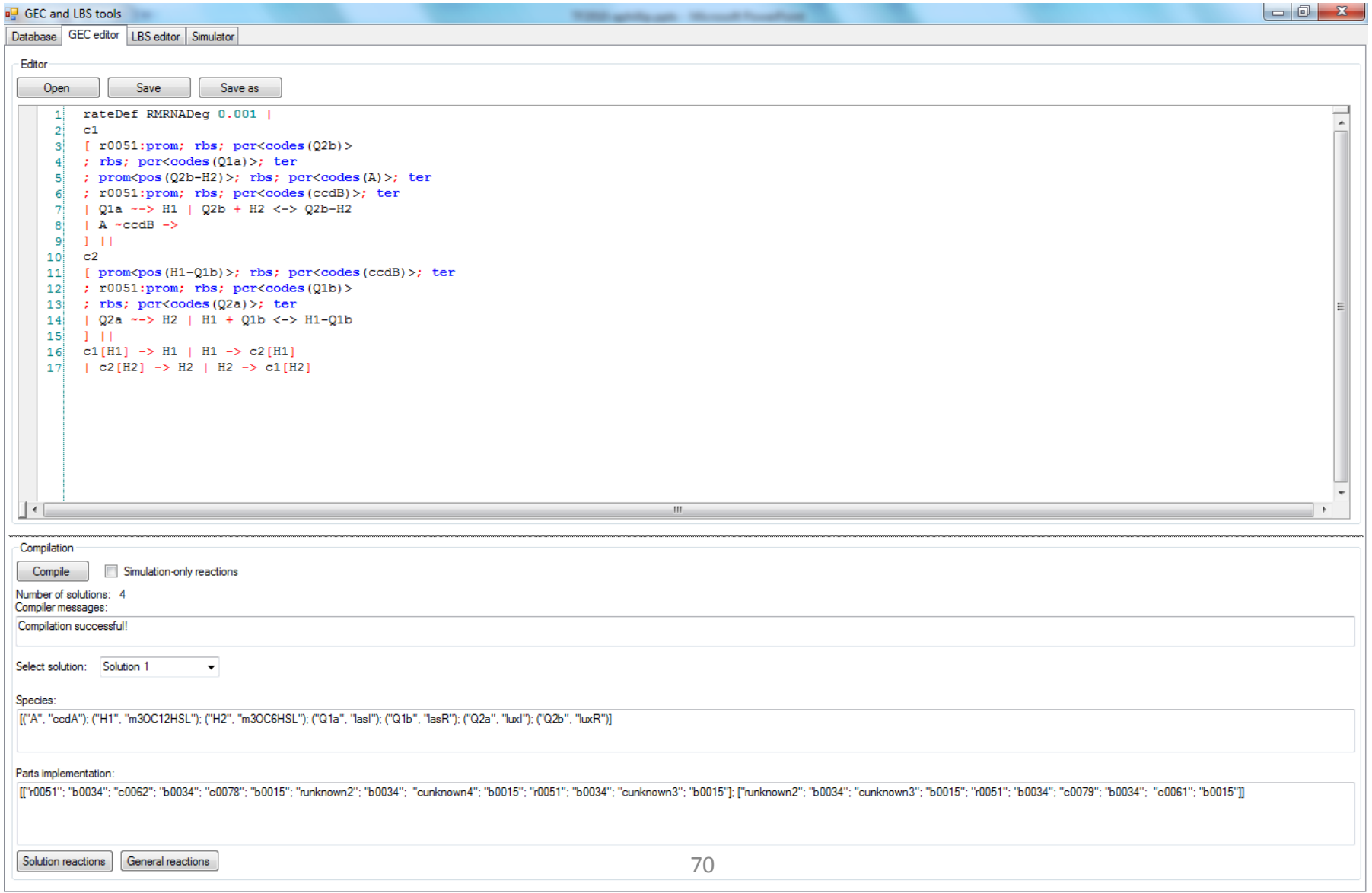
1. $[\![u : \text{prom}(Q)]\!]_\Gamma \triangleq (\text{par}\{\text{reacs}(q) \mid q \in Q\} \mid m \to^{rdm}, \{m\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$ where
   - $\text{reacs}(\text{con}(rt)) \triangleq g \to^{rt} g + m.$
   - $\text{reacs}(\text{pos}(S, rb, rub, rtb)) \triangleq g + S \to^{rb} g\text{-}S \mid g\text{-}S \to^{rub} g + S \mid g\text{-}S \to^{rtb} g\text{-}S + m.$
   - $\text{reacs}(\text{neg}(S, rb, rub, rtb)) \triangleq g + S \to^{rb} g\text{-}S \mid g\text{-}S \to^{rub} g + S \mid g\text{-}S \to^{rtb} g\text{-}S + m.$

   with g and m fresh.
2. $[\![u : \text{rbs}(\{\text{rate}(r)\})]\!]_\Gamma \triangleq (\mathbf{0}, \emptyset, \emptyset, \emptyset, \{f\}, \emptyset, \emptyset)$ where $f(m,p) \triangleq m \to^r p.$
3. $[\![u : \text{pcr}(\{\text{codes}(p,r)\})]\!]_\Gamma \triangleq (\mathbf{0}, \{p \to^r\}, \emptyset, \{p\}, \emptyset, \emptyset, \emptyset).$
4. $[\![u : \text{ter}]\!]_\Gamma \triangleq (\mathbf{0}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset).$
5. $[\![\mathbf{0}]\!]_\Gamma \triangleq (\mathbf{0}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset).$
6. $[\![p(\widetilde{u})\ \{P_1\}\ ;\ P_2]\!]_\Gamma \triangleq [\![P_2]\!]_{\Gamma\{p \mapsto f\}}$ where $f(\widetilde{A}) \triangleq [\![P_1\{\widetilde{A}/\widetilde{u}\}]\!].$
7. $[\![p(\widetilde{A})]\!]_\Gamma \triangleq f(\widetilde{A})$ where $f \triangleq \Gamma(p).$
8. $[\![P \mid C]\!]_\Gamma \triangleq (L_1 \mid L_2, D, M, Pr, F, G, H)$ where $(L_1, D, M, Pr, F, G, H) \triangleq [\![P]\!]_\Gamma$ and $L_2 \triangleq [\![C]\!].$
9. $[\![P_1 \parallel P_2]\!]_\Gamma \triangleq (L_1 \mid L_2, D_1 \cup D_2, M_1 \cup M_2, Pr_1 \cup Pr_2, F_1 \cup F_2, G_1 \cup G_2, H_1 \cup H_2)$ where $(L_1, D_1, M_1, Pr_1, F_1, G_1, H_1) \triangleq [\![P_1]\!]_\Gamma$ and $(L_2, D_2, M_2, Pr_2, F_2, G_2, H_2) \triangleq [\![P_2]\!]_\Gamma.$
10. $[\![P_1\ ;\ P_2]\!]_\Gamma \triangleq (L_1 \mid L_2 \mid L, D_1 \cup D_2, M, Pr, F_1' \cup F_2', G, H)$ where $(L_1, D_1, M_1, Pr_1, F_1, G_1, H_1) \triangleq [\![P_1]\!]_\Gamma,$ $(L_2, D_2, M_2, Pr_2, F_2, G_2, H_2) \triangleq [\![P_2]\!]_\Gamma,$ $L \triangleq \text{par}\{g(m) \mid g \in G_2, m \in M_1\} \cup \text{par}\{h(p) \mid h \in H_1, p \in Pr_2\},$ $M \triangleq \begin{cases} M_1 & \text{if } M_2 = \emptyset \\ M_2 & \text{otherwise} \end{cases}$ $Pr \triangleq \begin{cases} Pr_2 & \text{if } Pr_1 = \emptyset \\ Pr_1 & \text{otherwise} \end{cases}$ $(F_1', H_1') \triangleq \begin{cases} (F_1, H_1) & \text{if } Pr_2 = \emptyset \\ (\emptyset, \emptyset) & \text{otherwise} \end{cases}$ $(F_2', G_2') \triangleq \begin{cases} (F_2, G_2) & \text{if } M_1 = \emptyset \\ (\emptyset, \emptyset) & \text{otherwise} \end{cases}$ $G \triangleq \{g \mid g(m) \triangleq f(m,p), f \in F_1, p \in Pr_2\} \cup G_1 \cup G_2',$ $H \triangleq \{h \mid h(p) \triangleq f(m,p), f \in F_2, m \in M_1\} \cup H_2 \cup H_1'.$
11. $[\![c[P]]\!]_\Gamma \triangleq (c[L], D, M, Pr, F, G, H)$ where $(L, D, M, Pr, F, G, H) \triangleq [\![P_1]\!]_\Gamma.$
12. $[\![\text{new } x.P]\!]_\Gamma \triangleq [\![P[x'/x]]\!]_\Gamma$ for some fresh x'.
13. $[\![R]\!] \triangleq R.$
14. $[\![T]\!] \triangleq T.$
15. $[\![K]\!] \triangleq \mathbf{0}.$
16. $[\![C_1 \mid C_2]\!] \triangleq [\![C_1]\!] \mid [\![C_2]\!].$

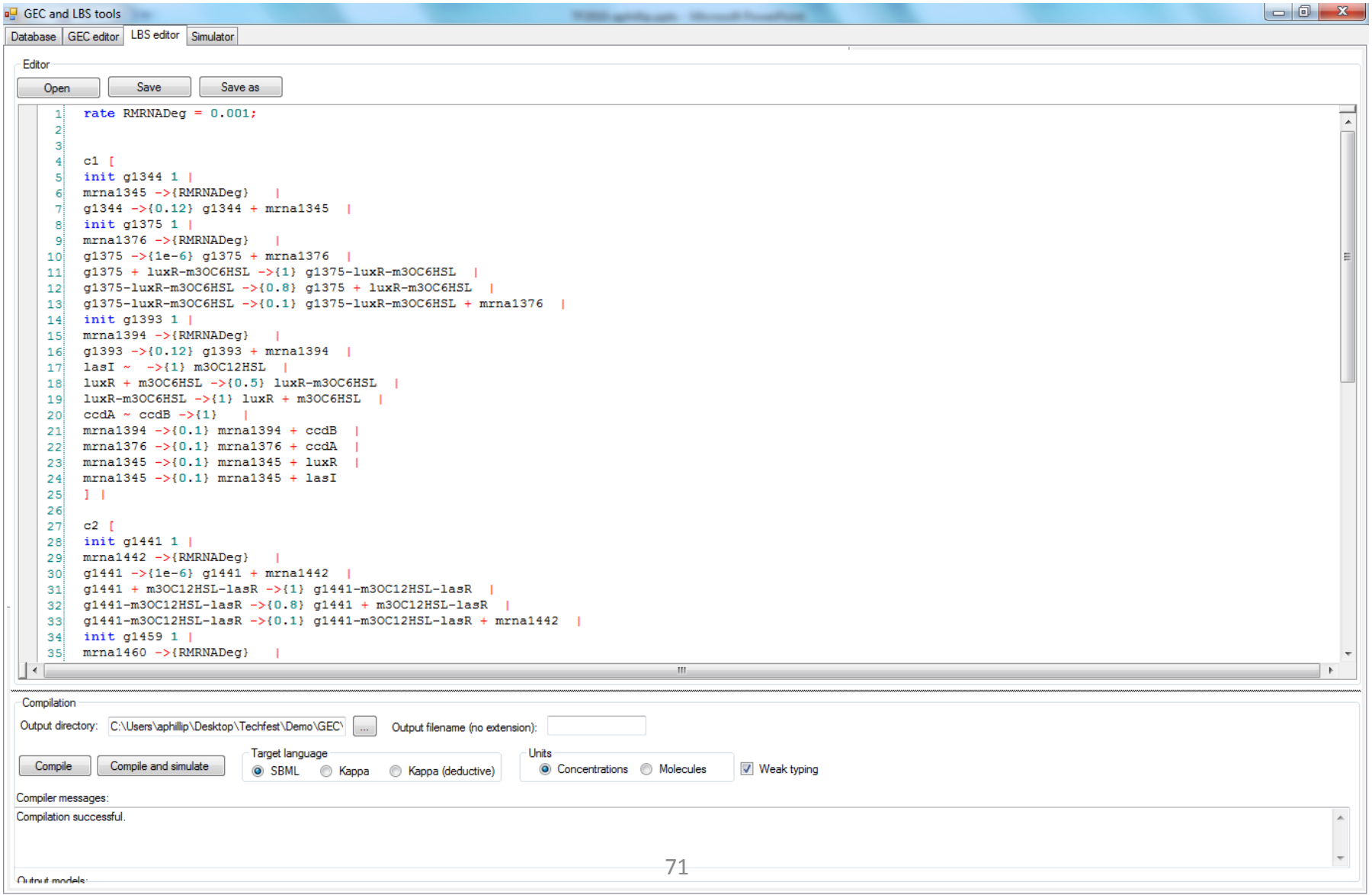
GEC Demo: Databases



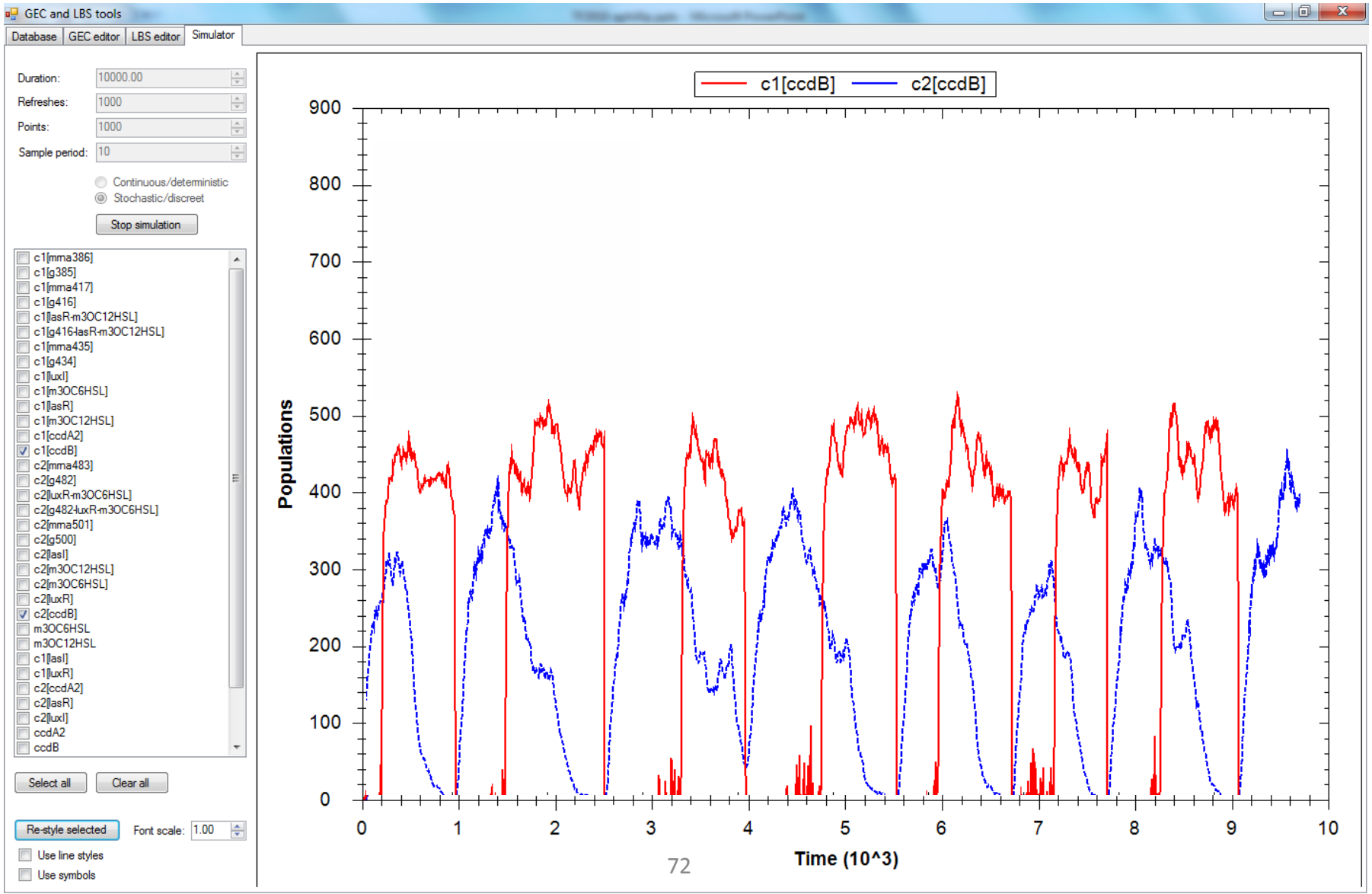
GEC Demo: Models



GEC Demo: Reactions

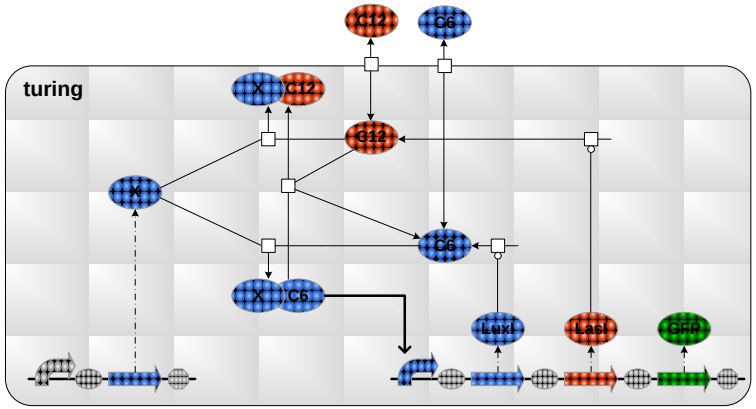


GEC Demo: Simulations

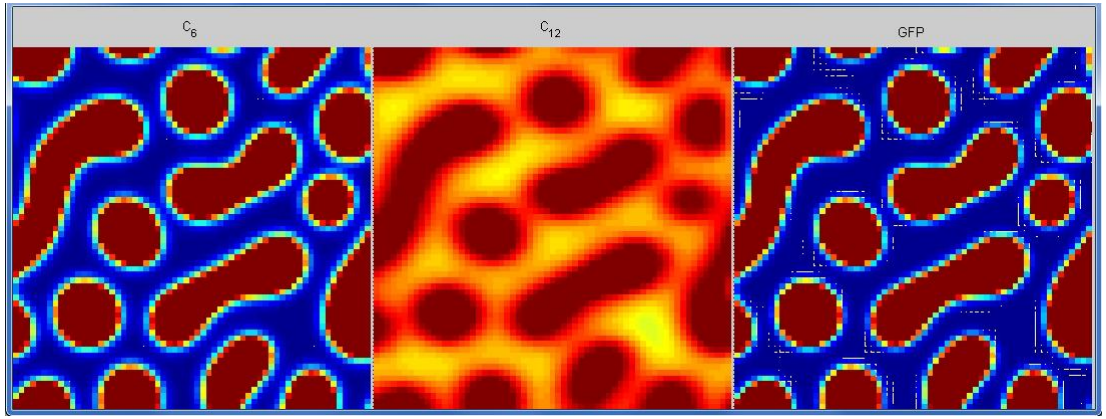


Scientific Challenges

- Engineer Turing Patterns in living cells




```
turing
[ prom; rbs; pcr<codes(X)>; ter
| prom<pos(X-C6)>;
   rbs; pcr<codes(luxI)>;
   rbs; pcr<codes(lasI)>;
   rbs; pcr<codes(gfp)>; ter
| luxI ~ -> C6 | lasI ~ -> C12
| X + C12 -> X-C12 | X-C12 -> X + C12
| X + C6 -> X-C6 | X-C6 -> X + C6
| X-C6 + C12 *->{1.0} X-C12 + C6
]
| C6 ->c[C6] | c[C6] -> C6
| C12 -> c[C12] | c[C12] -> C12
```



- Engineer bacteria to fix nitrogen for plants

Future Work

Parts

- Better part characterisation
- More realistic part properties

User Interface

- Visual editor
- Web-based tool

Analysis

- Integrated ODE analysis

Implementation

- Optimise translation to DNA sequences

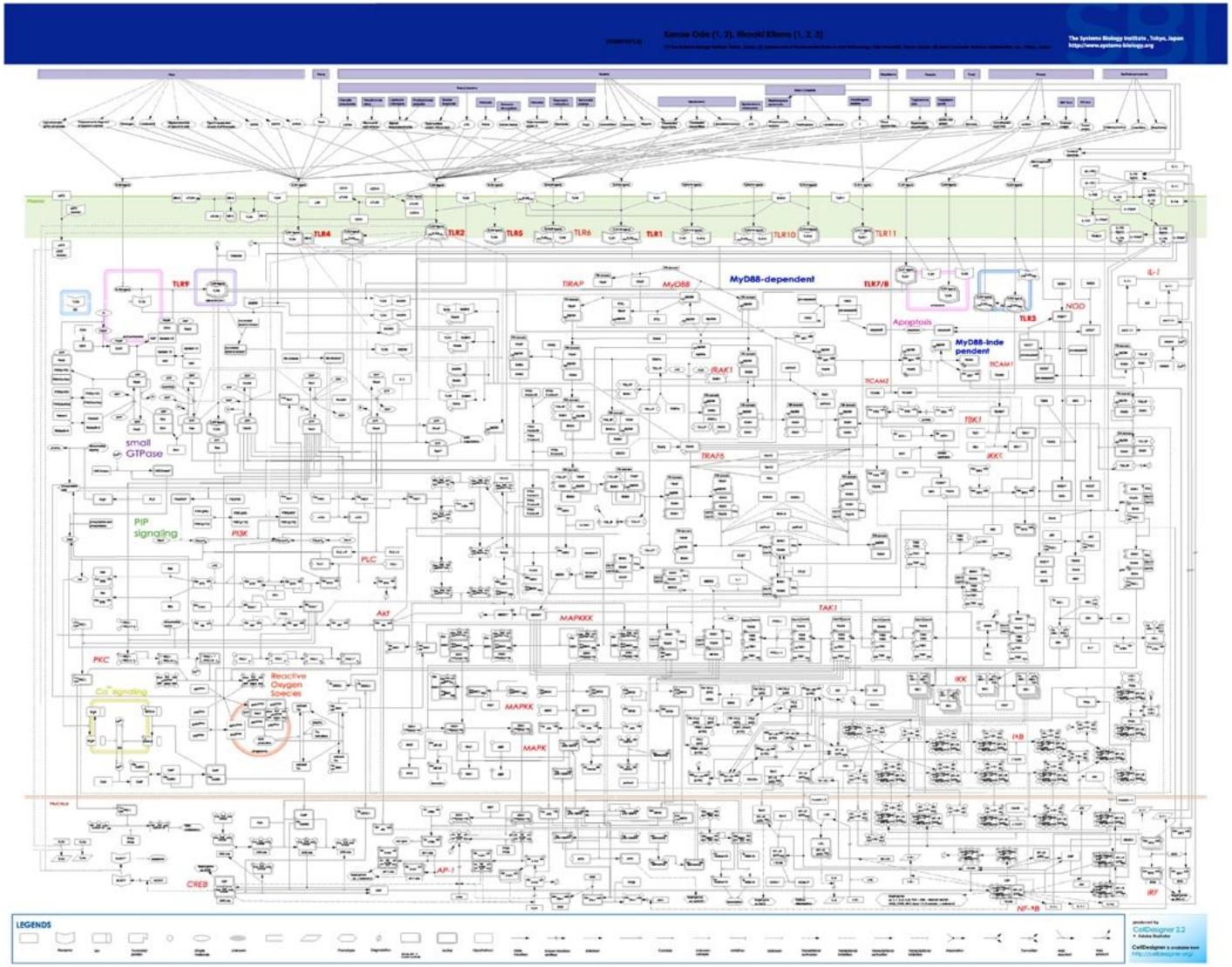
A Programming Language for Biological Processes

Luca Cardelli, Andrew Phillips

Systems Biology

- The Human Genome project:
  - Map out the complete genetic code in humans
  - To unravel the mysteries of how the human body functions
  - The code raised many more questions than answers
- Systems Biology:
  - Understand and predict the behaviour of biological systems
- Two complementary approaches:
  - Look at experimental results and infer system properties
  - Build detailed models of systems and test these in the lab
- Biological Modelling:
  - Conduct virtual experiments, saving time and resources
  - Clarify key mechanisms of how a biological system functions
  - Beginning to play a role in understanding disease

Large, Complex, Biological Models



Biological Programming

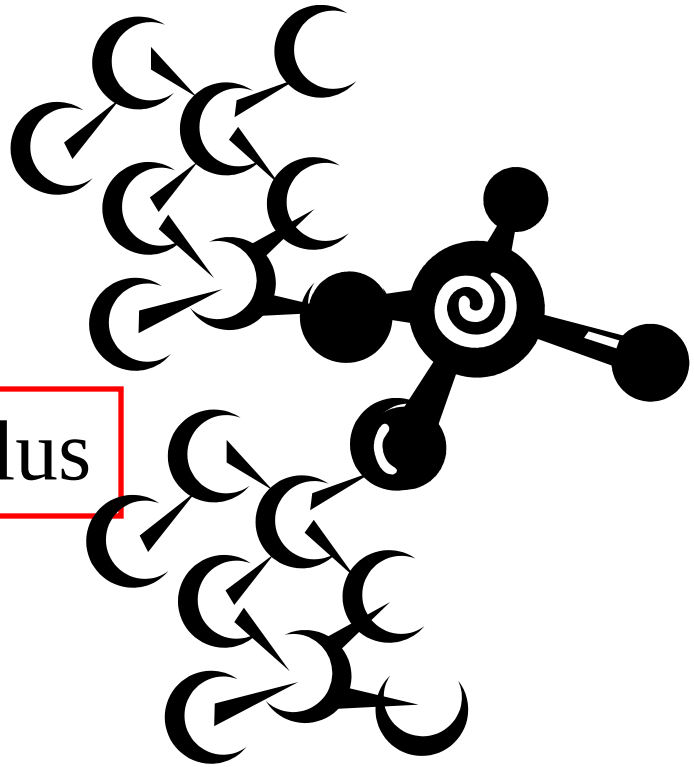
- Complex Models:
  - Difficult to understand, maintain and extend
  - Hundreds of reactions, soon to be tens of thousands
  - Would not write a program as a single list of thousands of instructions
- Modularity:
  - Need a way of decomposing a model into building blocks
  - Not your average computer programs
  - Massive parallelism, each instruction has a certain probability
  - Suggests a need for a biological programming language

Programming Languages for Biology

Languages for complex, parallel computer systems:

stochastic π-calculus

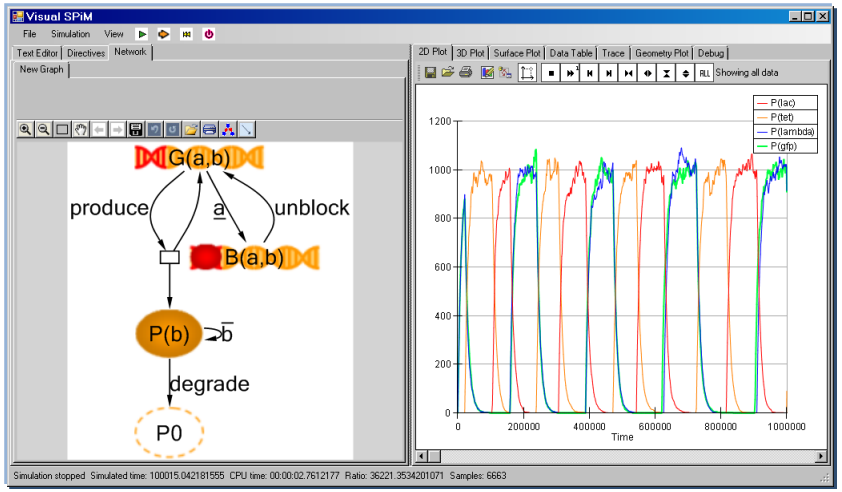
Languages for complex, parallel biological systems:



π-calculus by [Milner et al. 1989]. Stochastic version by [Priami et al. 1995]
First used in a biological context by [Regev et al. 2001]

SPiM: Stochastic π for Biology

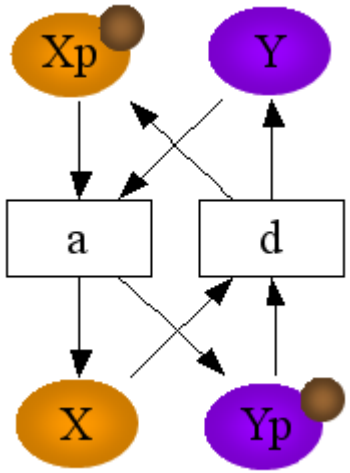
- A variant of stochastic π calculus
  - Supports expressive power of π
  - Graphical syntax and semantics
  - Biological constructs, e.g. complexation
  - Efficient implementation



Phillips and Cardelli, 2007

Message-Passing Approach

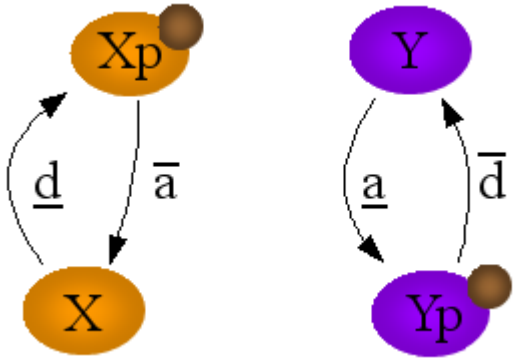
Chemical Reactions



$Xp + Y \rightarrow_a X + Yp$

$X + Yp \rightarrow_d Xp + Y$

SPiM Processes



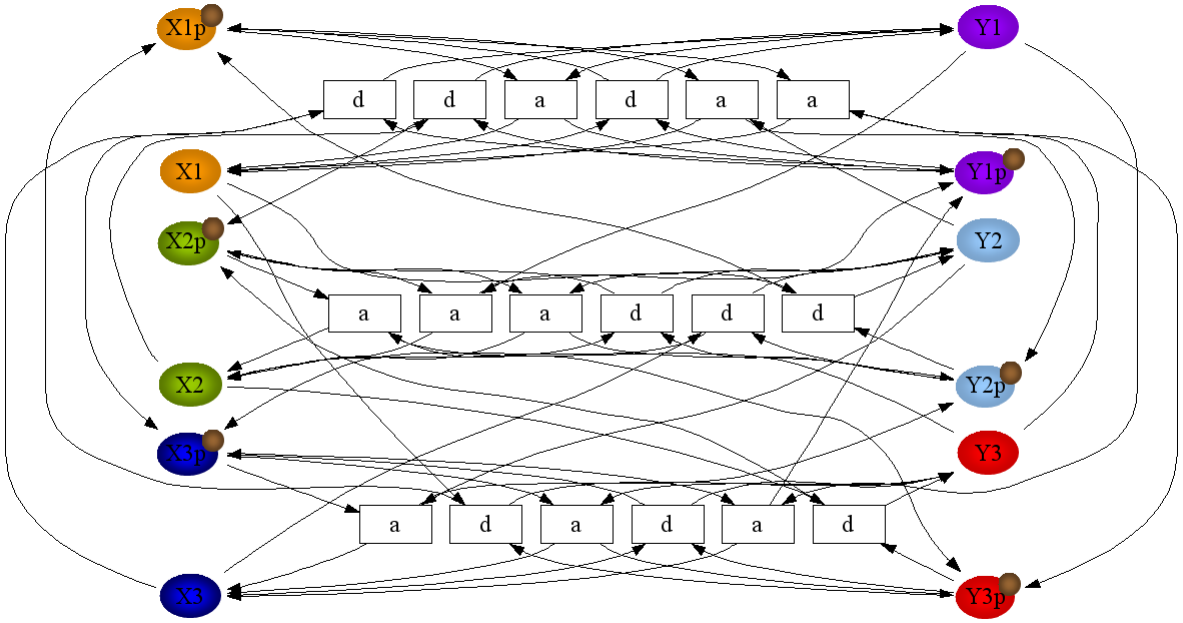
$Xp = \bar{a}.\ X$ $\quad$ $Y = \underline{a}.\ Yp$

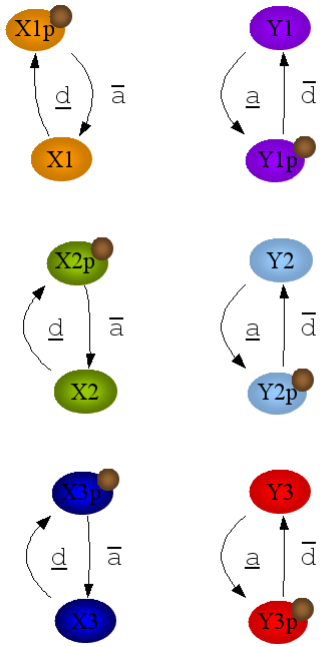
$X = \underline{d}.\ Xp$ $\quad$ $Yp = \bar{d}.\ Y$

Compact, Modular Models

Chemical Reactions

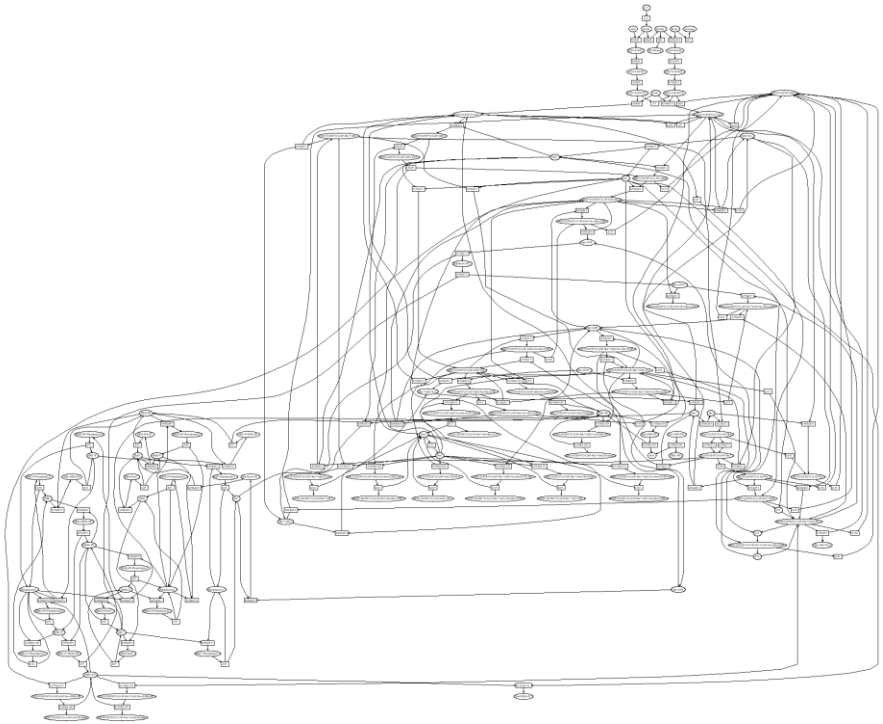
SPiM Processes



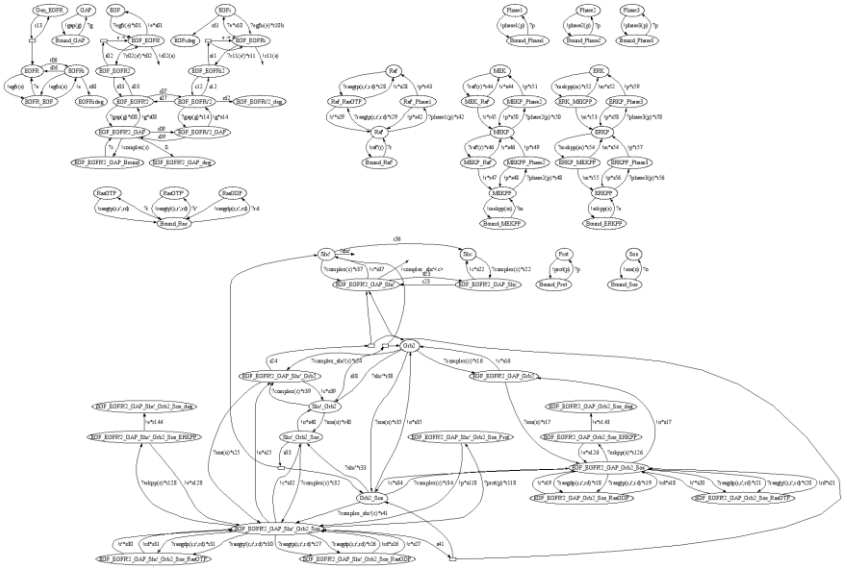


Improving Modularity with SPiM

EGFR chemical reactions



EGFR SPiM processes



Original model by Hornberg et al. 2005

Wang et al., BMC Systems Biology 2009

SPiM Syntax

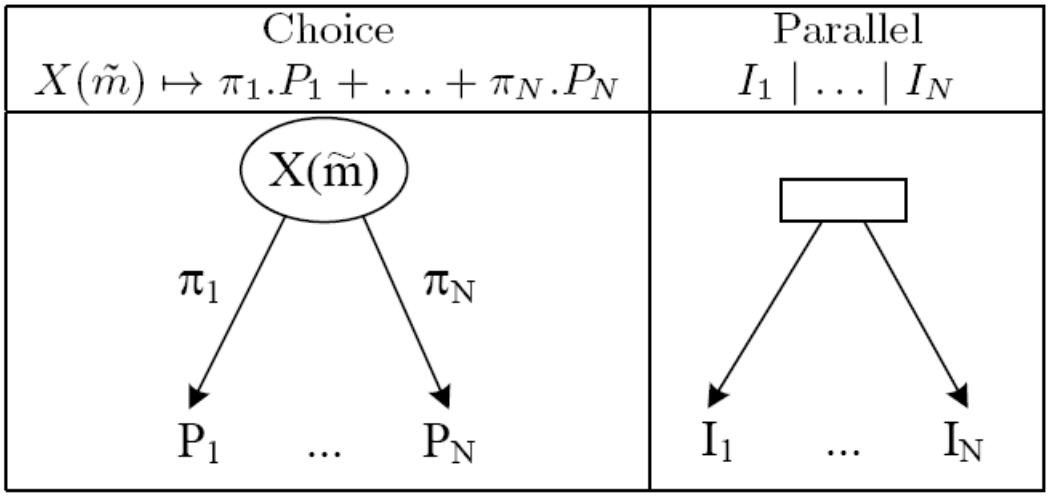
π::=	$\underline{x}(m)$	**Receive** value m on channel x
	$\bar{x}\langle n\rangle$	**Send** value n on channel x
	$\bar{x}(m)$	**Send** restricted value m on channel x
	r	**Delay** at rate r
M::=	$\pi_1.P_1 + ... + \pi_N.P_N$	**Choice** between actions
P::=	$P_1 \mid ... \mid P_M$	**Parallel** composition of processes
	$X(n)$	**Species** X with parameters n
	$(x_1,...,x_N)\ P$	**Restriction** of channels $x_1,...,x_N$ to P
D::=	P	**Definition** of a process
	M	**Definition** of a choice
E::=	$X_1(m_1) = D_1$,...,	**Definitions** for X_i with parameters m_i
	$X_N(m_N) = D_N$	
S::=	E,P	**System** of E and P

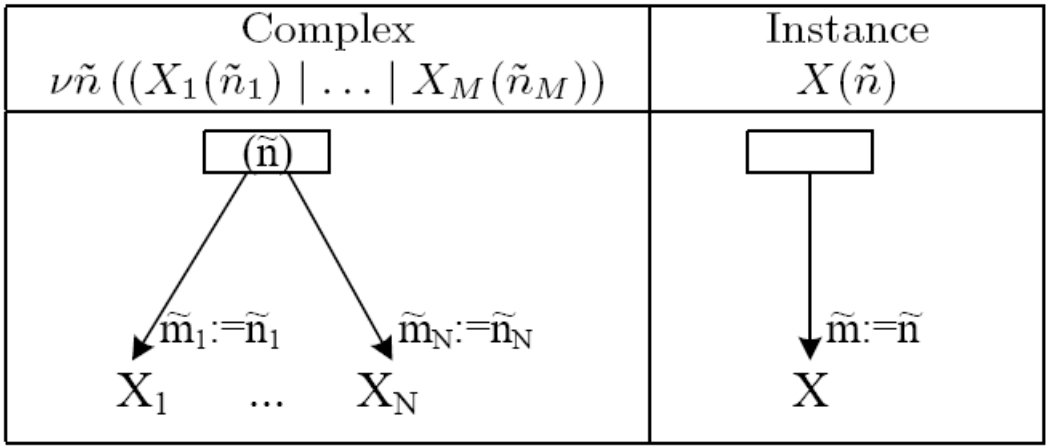
Normal Form Syntax

- Every process is equivalent to a normal form

$$
\begin{array}{rcl}
P ::= & I_1 \mid \ldots \mid I_N & \text{Species} \\
I ::= & X(\tilde{n}) & \text{Instance} \\
\mid & \nu\tilde{z}\,((X_1(\tilde{n}_1) \mid \ldots \mid X_M(\tilde{n}_M)) & \text{Complex} \\
C ::= & \pi_1.P_1 + \ldots + \pi_N.P_N & \text{Choice} \\
E ::= & X_1(\tilde{m}_1) \mapsto C_1, \ldots, X_N(\tilde{m}_N) \mapsto C_N & \text{Environment}
\end{array}
$$

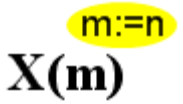
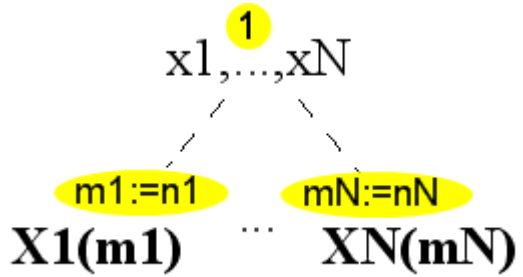
Graphical Syntax: Environment



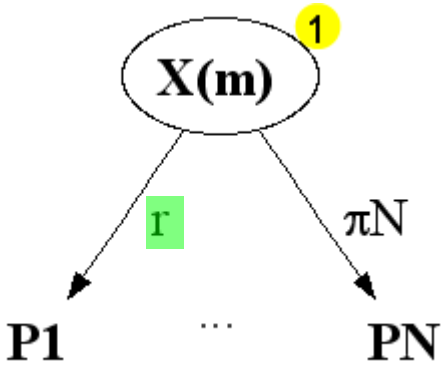


Phillips, Cardelli and Castagna, 2006

Graphical Syntax: Processes

Parallel	Species	Restriction
$P_1 \mid \ldots \mid P_M$	$X(n)$, if $X(m) = C$	$\nu(x_1,\ldots x_N)\,(X_1\,(n_1) \mid \ldots \mid X_N\,(n_N))$
		

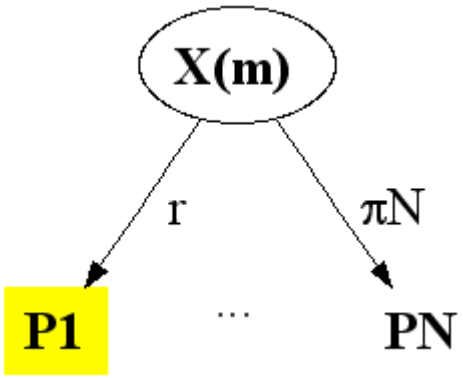
Graphical Semantics: Delay



$$X(m) = r.P_1 + \ldots + \pi_N.P_N$$

$$X(m)$$

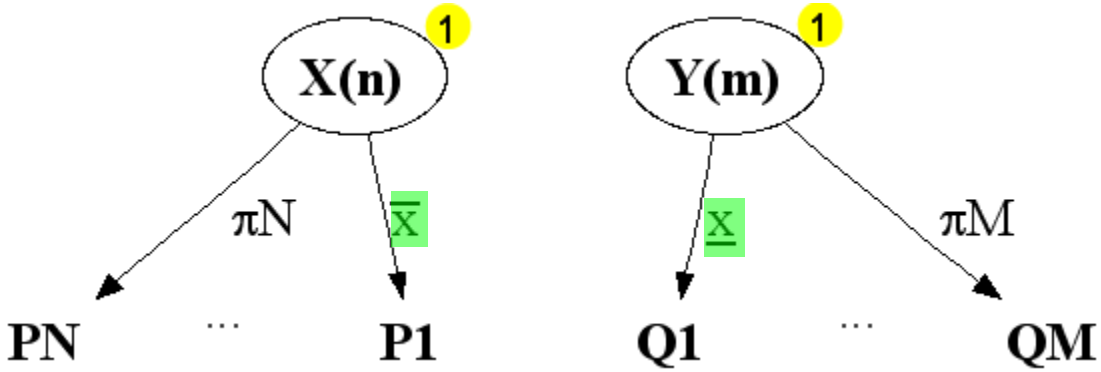
Graphical Semantics: Delay



$$X(m) = r.P_1 + \ldots + \pi_N.P_N$$

$$X(m) \longrightarrow P_1$$

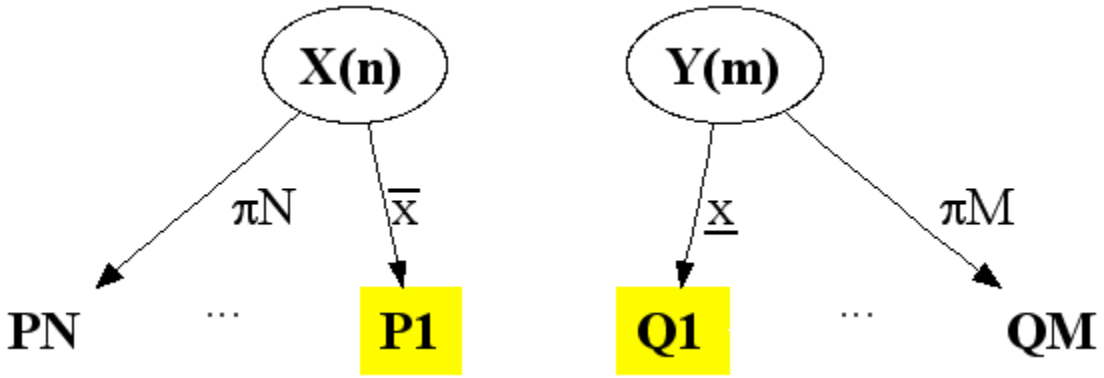
Graphical Semantics: Interaction



$$X(n) = \bar{x}.P_1 + \ldots + \pi_N.P_N \quad , \quad Y(m) = \underline{x}.Q_1 + \ldots + \pi_M.Q_M$$

$$X(n) \mid Y(m)$$

Graphical Semantics: Interaction



$$X(n) = \bar{x}.P_1 + \ldots + \pi_N.P_N \quad , \quad Y(m) = \underline{x}.Q_1 + \ldots + \pi_M.Q_M$$

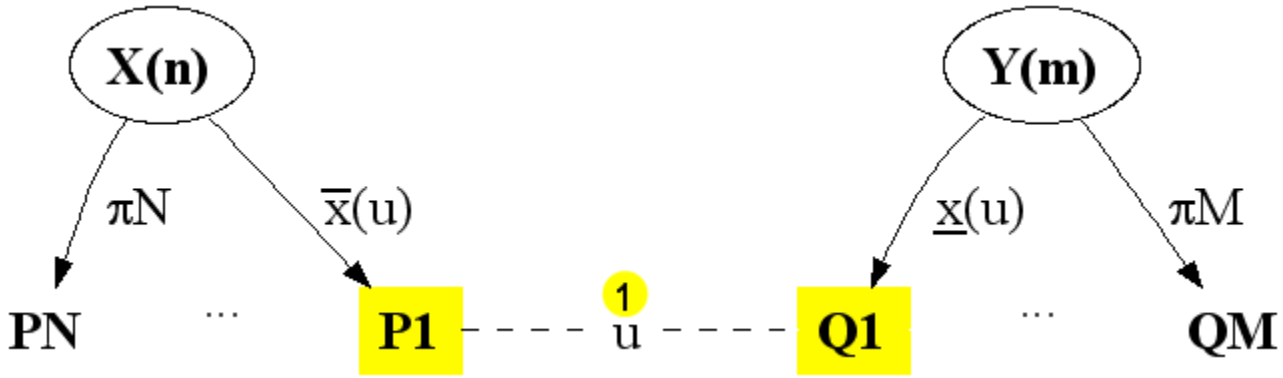
$$X(n) \mid Y(m) \longrightarrow P_1 \mid Q_1$$

Graphical Semantics: Binding

$$X(n) = \bar{x}(u).P_1 + \ldots + \pi_N.P_N \ , \ \ Y(m) = \underline{x}(u).Q_1 + \ldots + \pi_M.Q_M$$

$$X(n) \mid Y(m)$$

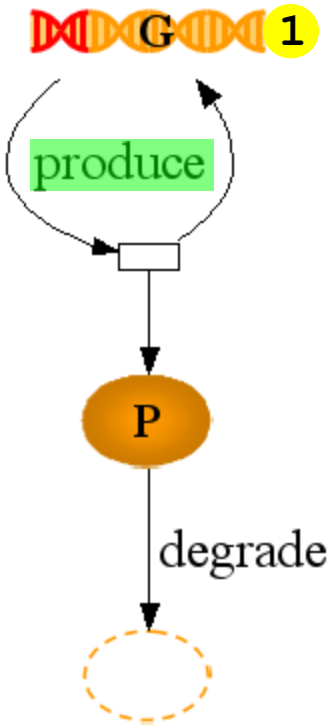
Graphical Semantics: Binding



$$X(n) = \bar{x}(u).P_1 + \ldots + \pi_N.P_N \ , \ \ Y(m) = \underline{x}(u).Q_1 + \ldots + \pi_M.Q_M$$

$$X(n) \mid Y(m) \longrightarrow (\nu u)\,(P_1 \mid Q_1)$$

Production: $G \xrightarrow{\text{produce}} G + P \qquad P \xrightarrow{\text{degrade}} \varnothing$

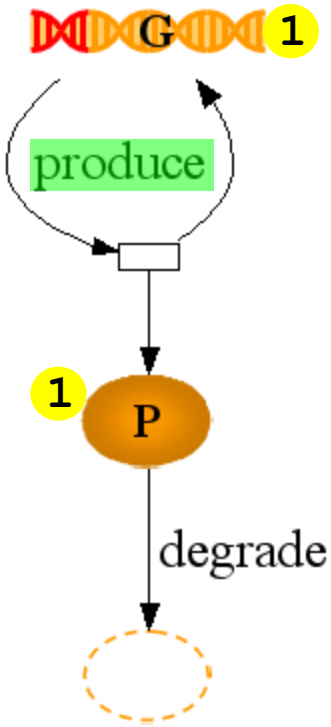


reaction	rate	propensity (s^{-1})
produce	0.1	0.1·1
degrade	0.001	0.001·0

G = produce.(P | G)
P = degrade.0

- A protein *P* can be produced with propensity 0.1
- Probability of a reaction depends on propensity
- Exact simulation: what happens next?

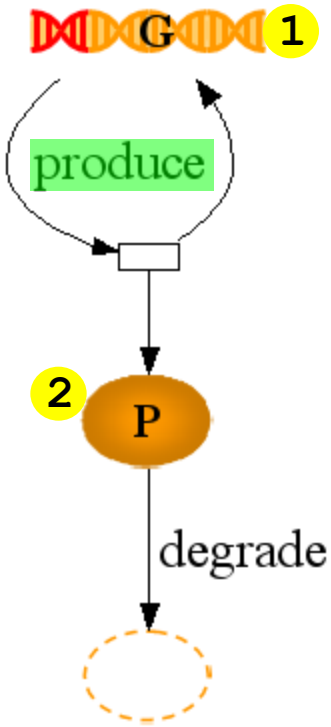
Production: $G \xrightarrow{\text{produce}} G + P \qquad P \xrightarrow{\text{degrade}} \varnothing$



reaction	propensity (s^{-1})
produce	0.1
degrade	0.001

- Another protein *P* can be produced
- 100 times more likely to produce than degrade

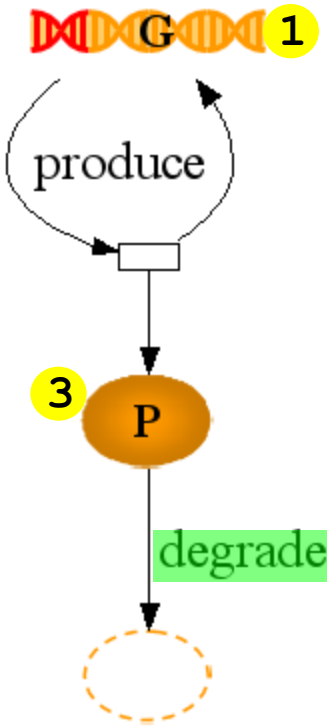
Production: $G \xrightarrow{\text{produce}} G + P \qquad P \xrightarrow{\text{degrade}} \varnothing$



reaction	propensity (s^{-1})
produce	0.1
degrade	0.001·2

- And another...

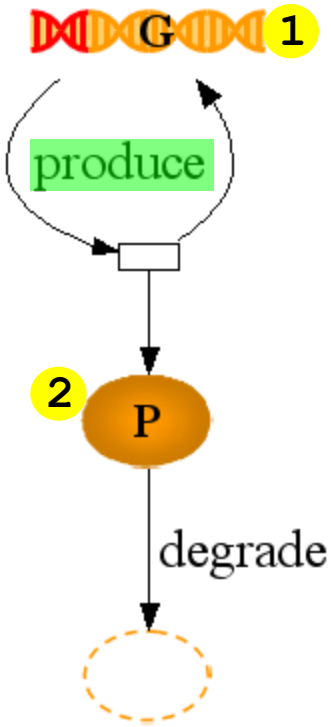
Production: $G \xrightarrow{\text{produce}} G + P \qquad P \xrightarrow{\text{degrade}} \varnothing$



reaction	propensity (s^{-1})
produce	0.1
degrade	0.001·3

- A protein *b* can be degraded at rate 0.001
- Low probability, but still possible

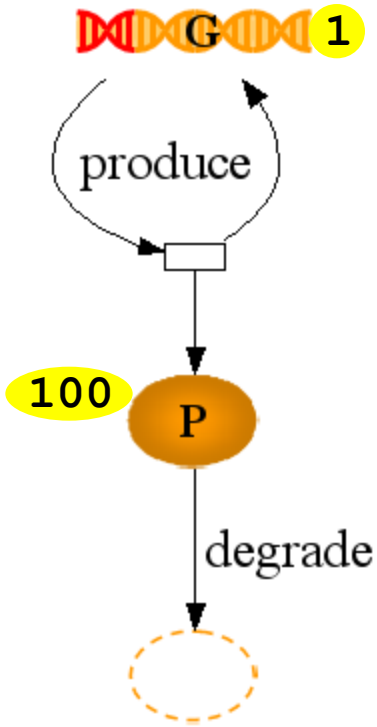
Production: $G \xrightarrow{\text{produce}} G + P \qquad P \xrightarrow{\text{degrade}} \varnothing$



reaction	propensity (s^{-1})
produce	0.1
degrade	0.001·2

- Eventually...

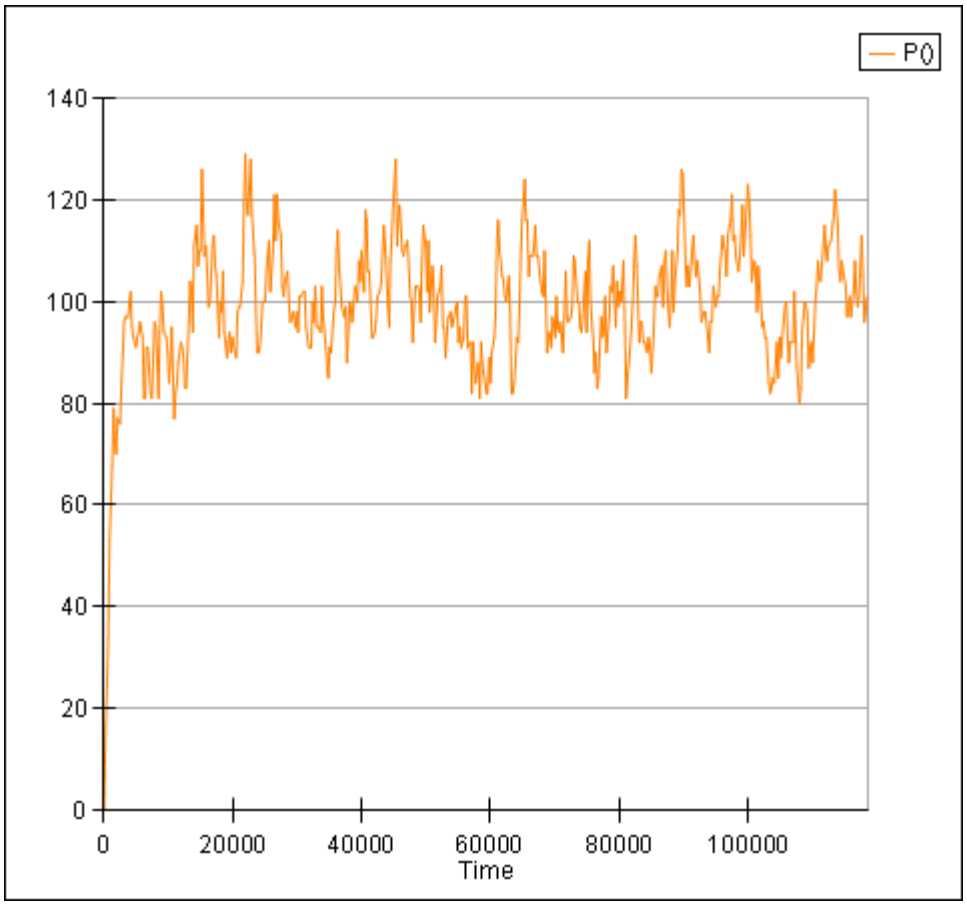
Production: $G \xrightarrow{\text{produce}} G + P \qquad P \xrightarrow{\text{degrade}} \varnothing$



reaction	propensity (s^{-1})
produce	0.1
degrade	0.001·100

- Equilibrium at about 100 proteins.
- Propensities of both reactions are equal.

Gene Simulation

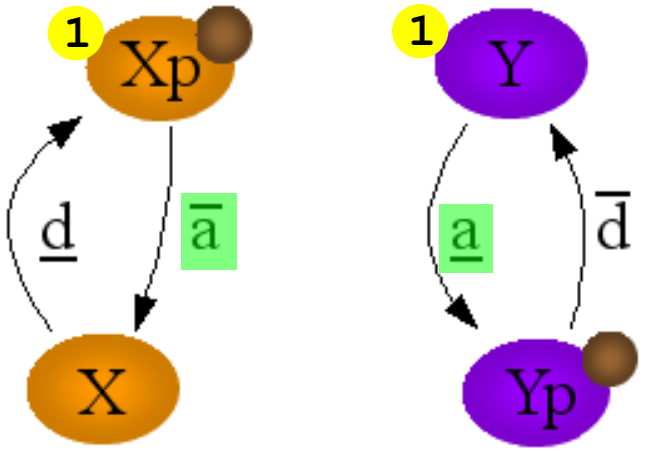


- Simulation results show evolution over time
- Level of protein *P* fluctuates around 100

Interaction: $Xp + Y \;{}_{d}\!\leftrightarrow^{a}\; X + Yp$

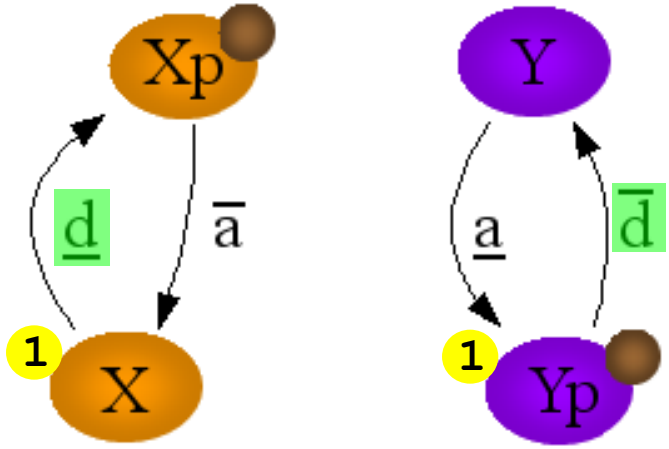
Xp = $\bar{a}$. X Y = $\underline{a}$. Yp

X = $\underline{d}$. Xp Yp = $\bar{d}$. Y



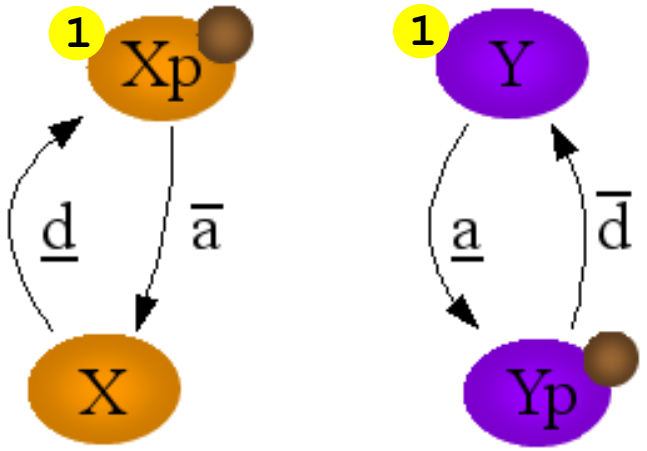
- *Xp* and *Y* can interact on channel *a*
- *Xp* activates *Y* by sending its phosphate group

Interaction: $Xp + Y \;{}_{d}\!\leftrightarrow^{a}\; X + Yp$



- *X* and *Yp* can interact on channel *d*

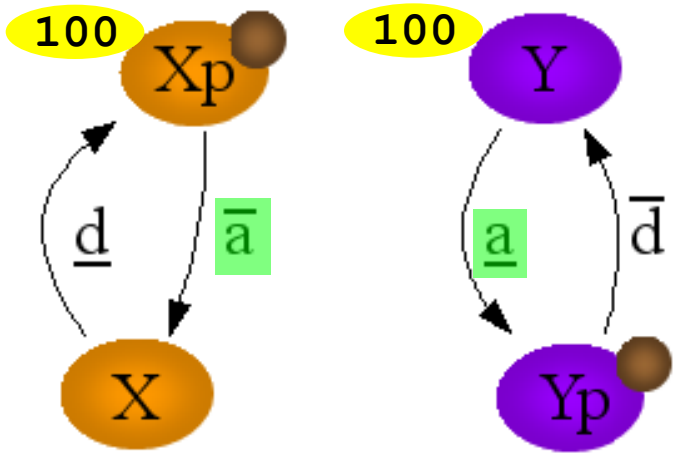
Interaction: $Xp + Y \;{}_{d}\!\leftrightarrow^{a}\; X + Yp$



- Interactions can continue indefinitely...

Interaction: $Xp + Y \;{}_{d}\!\leftrightarrow^{a}\; X + Yp$

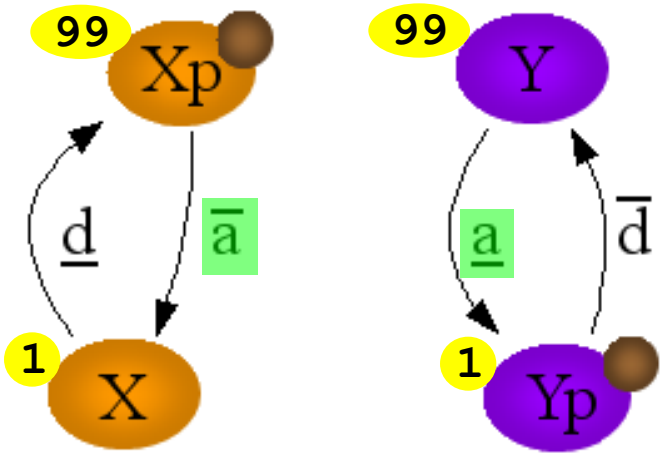
reaction	propensity (s^{-1})
a	100·100·100
d	0



- What happens if we mix 100·*Xp* and 100·*Y* ?
- Assume *rate*(*a*) = $100s^{-1}$ and *rate*(*d*) = $10s^{-1}$
- An *Xp* and *Y* protein can interact on channel *a*.

Interaction: $Xp + Y \;{}_{d}\!\leftrightarrow^{a}\; X + Yp$

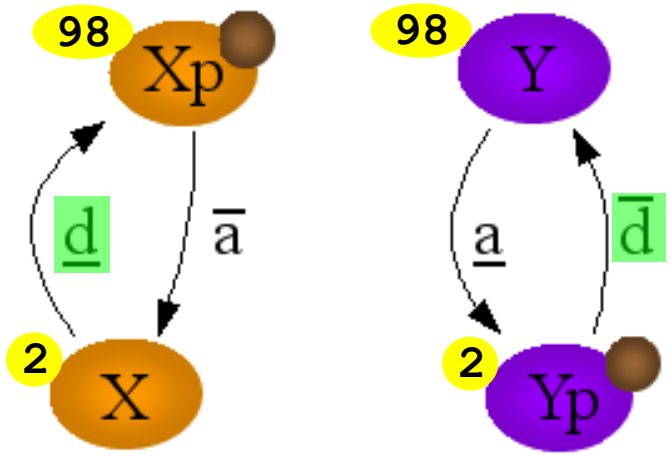
reaction	propensity (s^{-1})
a	100·99·99
d	10·1·1



- An additional *Xp* and *Y* protein can interact.

Interaction: $Xp + Y \underset{d}{\leftrightarrow}^{a} X + Yp$

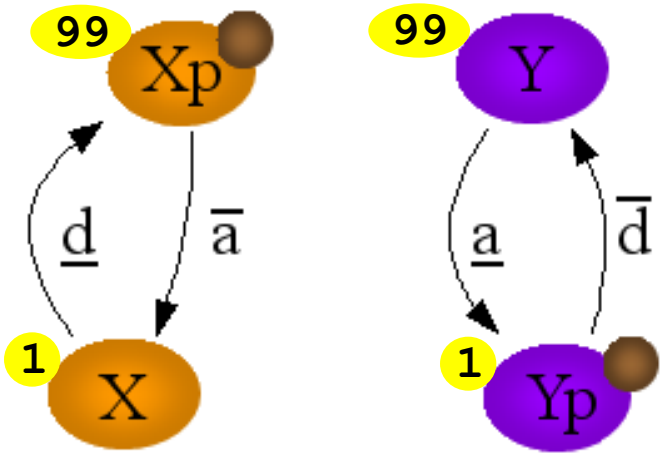
reaction	propensity (s^{-1})
a	100·98·98
d	10·2·2



- An *X* and *Yp* protein can interact

Interaction: $Xp + Y \;{}_{d}\!\leftrightarrow^{a}\; X + Yp$

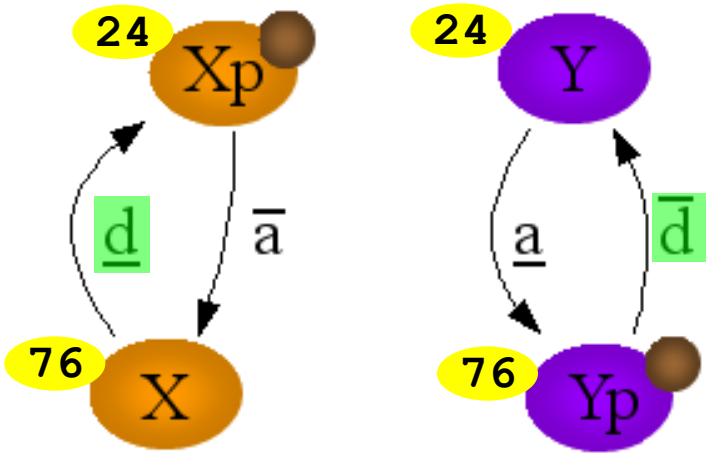
reaction	propensity (s^{-1})
a	100·99·99
d	10·1·1



- Eventually an equilibrium is reached...

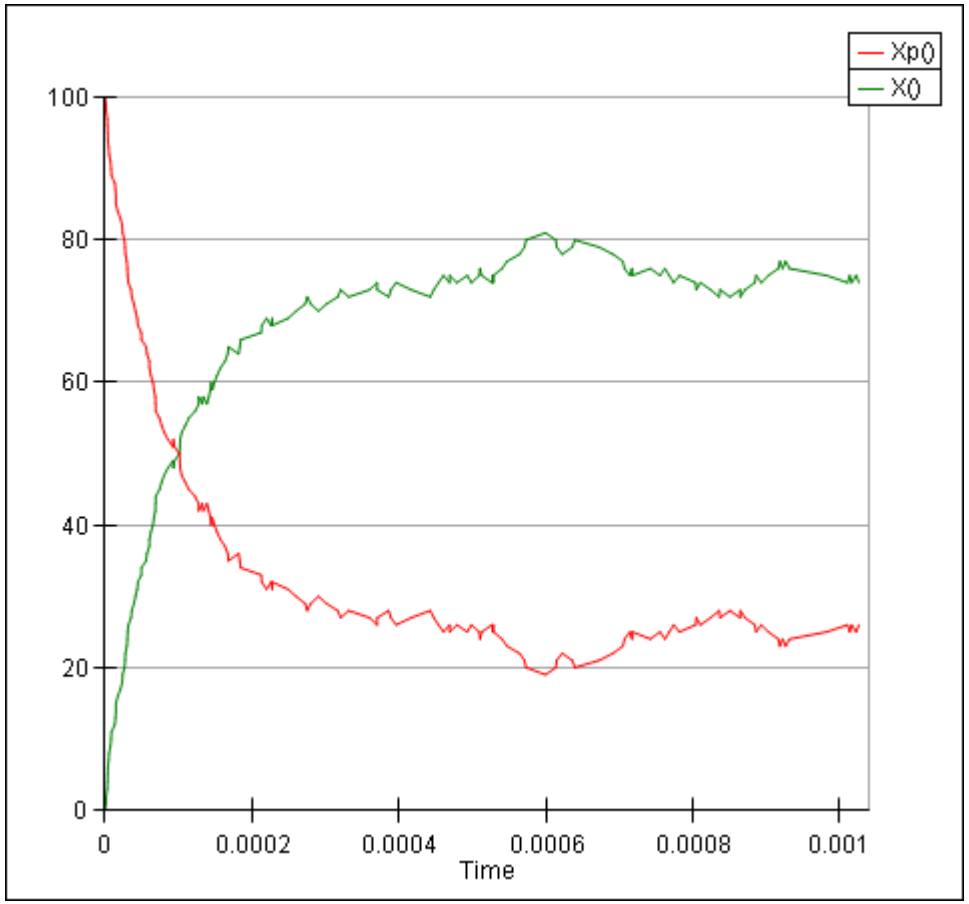
Interaction: $Xp + Y \;{}_{d}\!\leftrightarrow^{a}\; X + Yp$

reaction	propensity (s^{-1})
a	100·24·24
d	10·76·76



- At equilibrium when rate(a)·[Xp][Y] ≈ rate(d)·[X][Yp]

Interaction: $Xp + Y \;{}_{d}\!\leftrightarrow^{a}\; X + Yp$

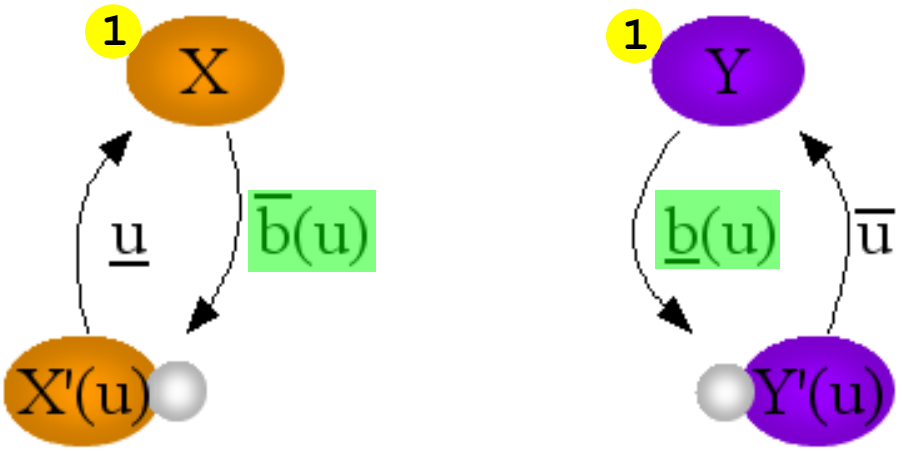


- At equilibrium: $100s^{-1}\cdot[Xp][Y] \approx 10s^{-1}\cdot[X][Yp]$
- Approximately $24\cdot Xp$ and $76\cdot X$

Binding: $X + Y \;{}_{u}\!\leftrightarrow^{b}\; X'Y'$

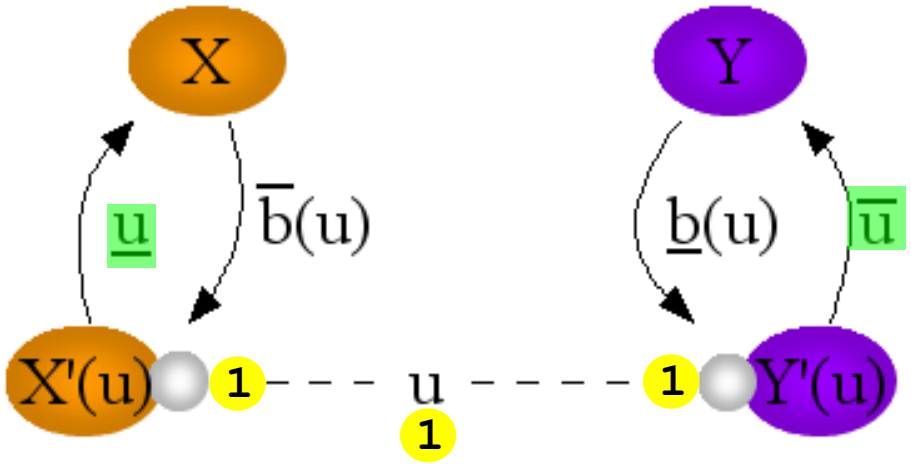
X = $\bar{b}(u).X'$ Y = $\underline{b}(u).Y'$

X' = $\underline{u}.X$ Y' = $\bar{u}.Y$



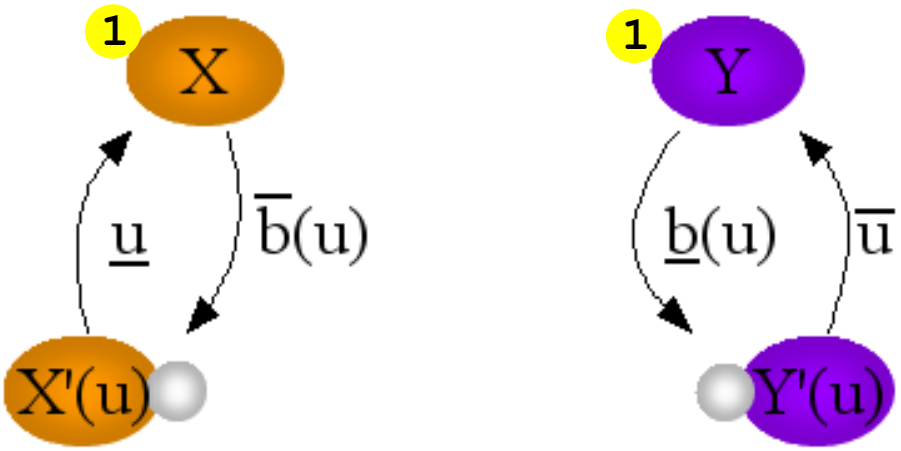
- *X* and *Y* can bind on channel *b*

Binding: $X + Y \;{}_{u}\!\leftrightarrow^{\mathrm{b}}\; X'Y'$



- *X′* and *Y′* can unbind on channel *u*

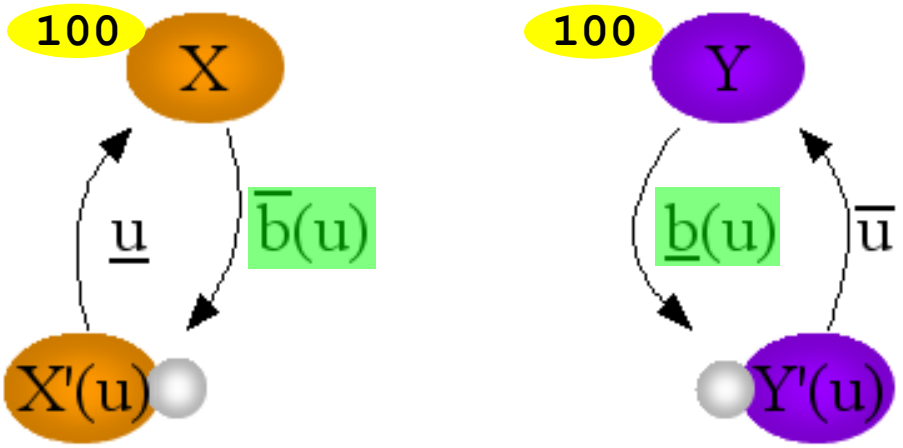
Binding: $X + Y \;{}_{u}\!\leftrightarrow^{\mathrm{b}}\; X'Y'$



- Binding and unbinding can continue indefinitely...

Binding: $X + Y \;{}_{u}\!\leftrightarrow^{b}\; X'Y'$

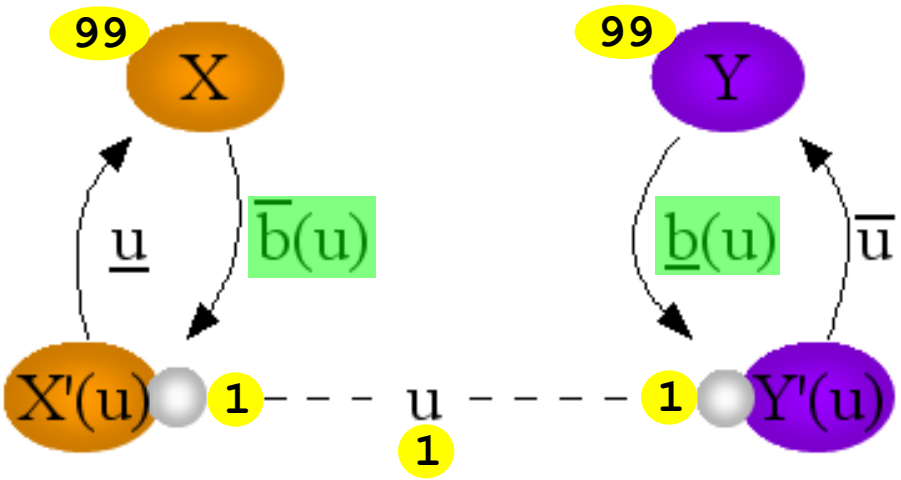
reaction	propensity (s^{-1})
b	100·100·100
u	0



- What happens if we mix 100×*Xp* and 100×*Y* ?
- Assume *rate*(*b*) = 100s^{-1} and *rate*(*u*) = 10s^{-1}
- An *X* and *Y* protein can bind on channel *b*.

Binding: $X + Y \;{}_{u}\!\leftrightarrow^{b}\; X'Y'$

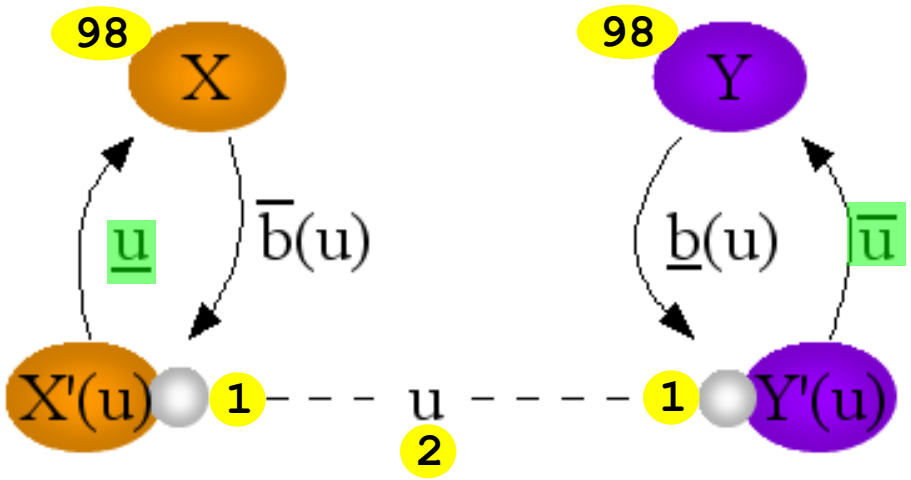
reaction	propensity (s^{-1})
b	100·99·99
u	10·1



- An additional *X* and *Y* protein can bind.

Binding: $X + Y \ {}_{u}\leftrightarrow^{b} \ X'Y'$

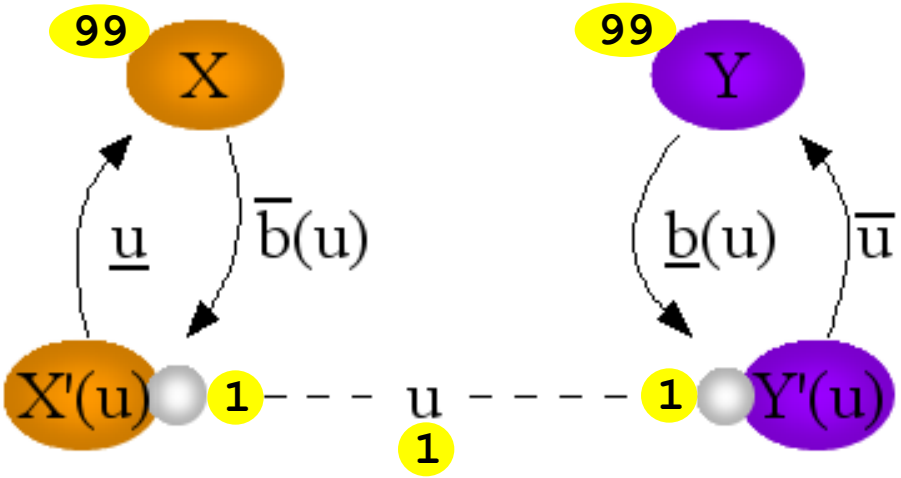
reaction	propensity (s^{-1})
b	100·98·98
u	10·2



- An *X′* and *Y′* protein can unbind on channel *u*

Binding: $X + Y \;{}_{u}\!\leftrightarrow^{b}\; X'Y'$

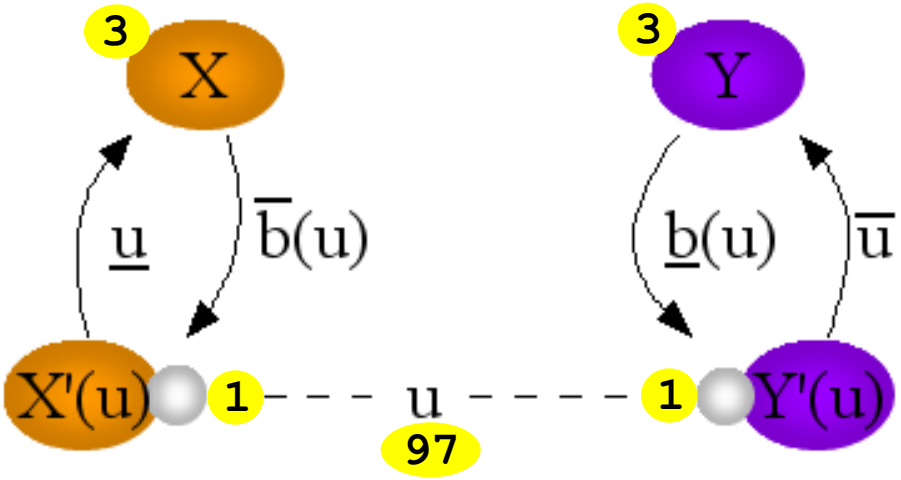
reaction	propensity (s^{-1})
b	100·99·99
u	10·1



- Eventually...

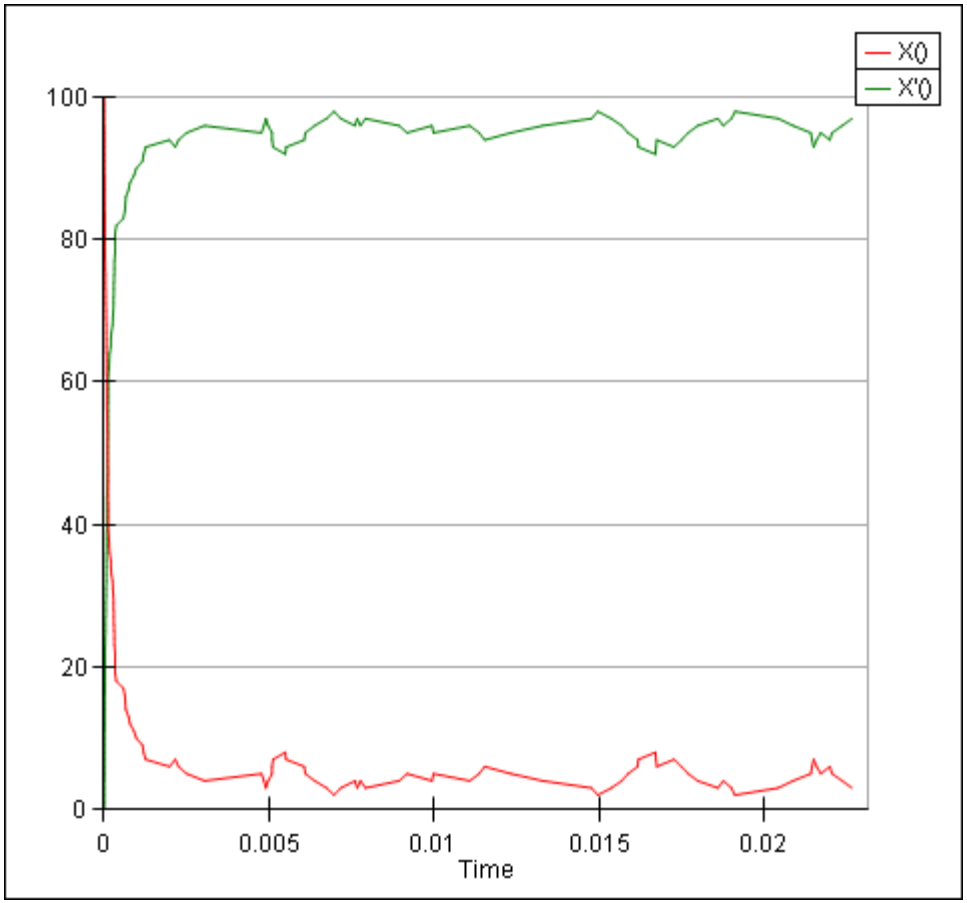
Binding: $X + Y \;{}_{u}\!\leftrightarrow^{b}\; X'Y'$

reaction	propensity (s^{-1})
b	100·3·3
u	10·97



- At equilibrium when rate(b)×[X][Y] ≈ rate(u)×(u) ([X'] [Y'])

Binding: $X + Y \;{}_{-b}\!\leftrightarrow^{+b}\; X'Y'$



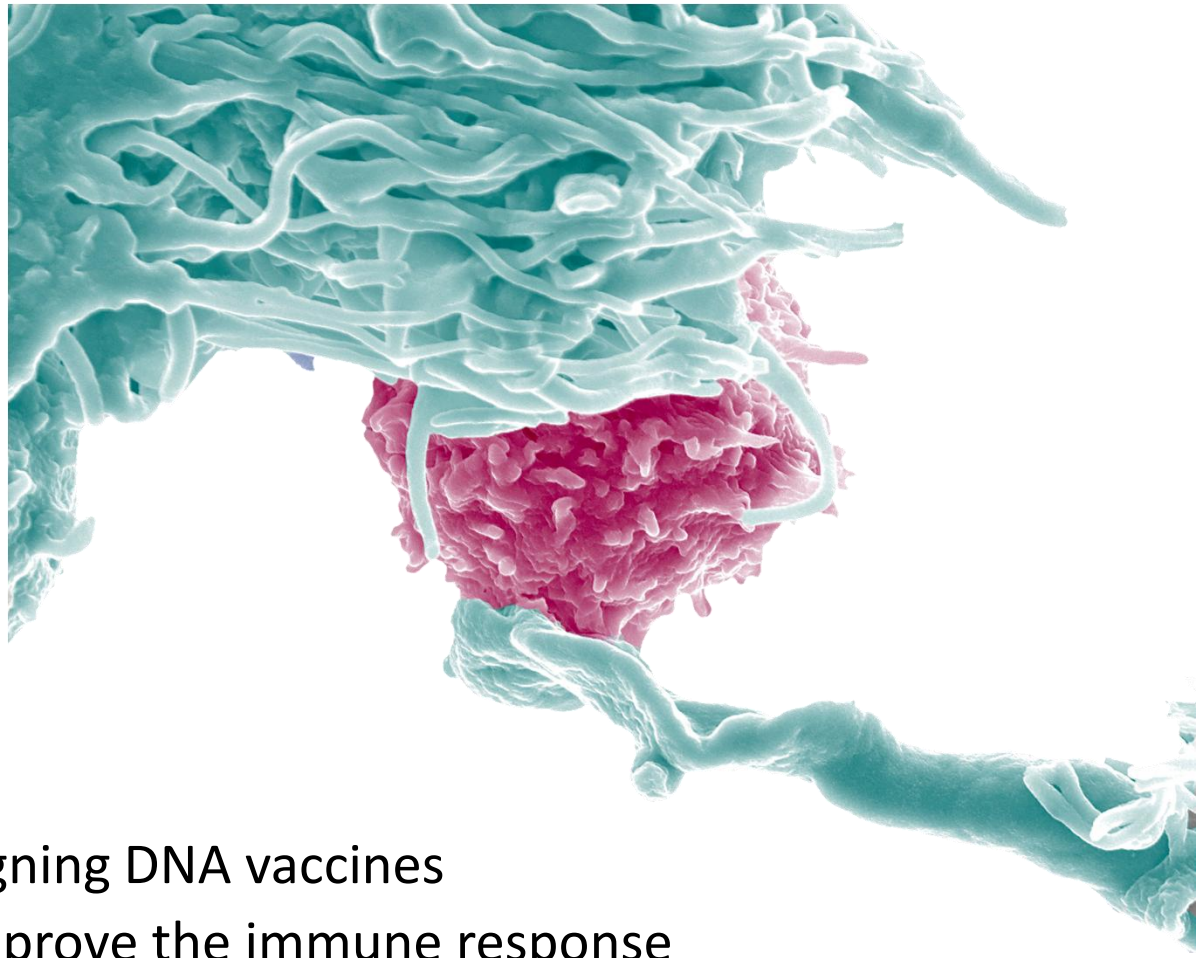
- At equilibrium: $100s^{-1}\cdot[X][Y] = 10s^{-1}\cdot[X'Y']$
- Approximately $3\cdot X$ and $97\cdot X'Y'$

Programming the Immune System

Neil Dalchau, Luca Cardelli
Leonard Goldstein, Tim Elliott,
Joern Werner & Andrew Phillips

Programming the Immune System

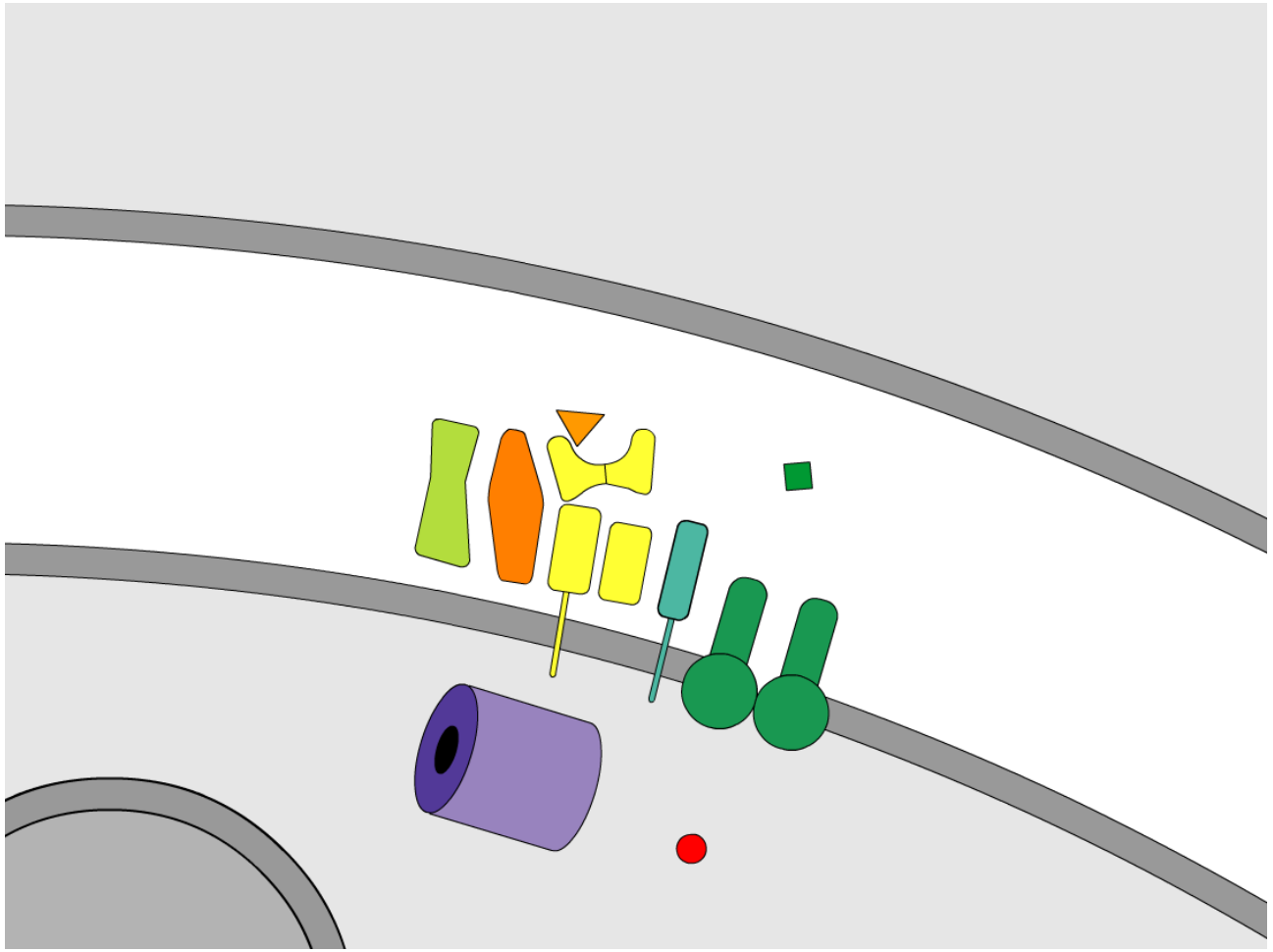
Understanding What to Program



Designing DNA vaccines
to improve the immune response

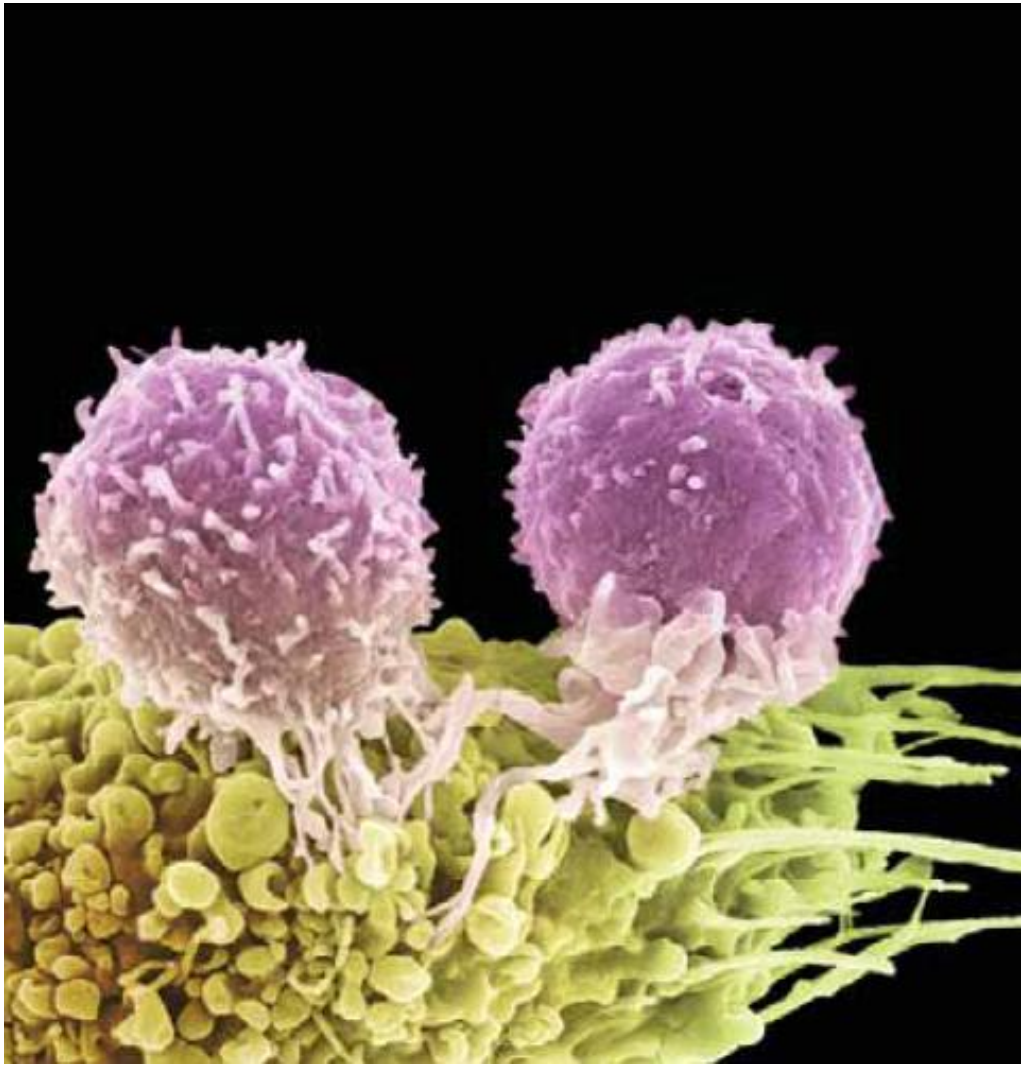
Science Photo Library

MHC: A Biological Virus Scanner



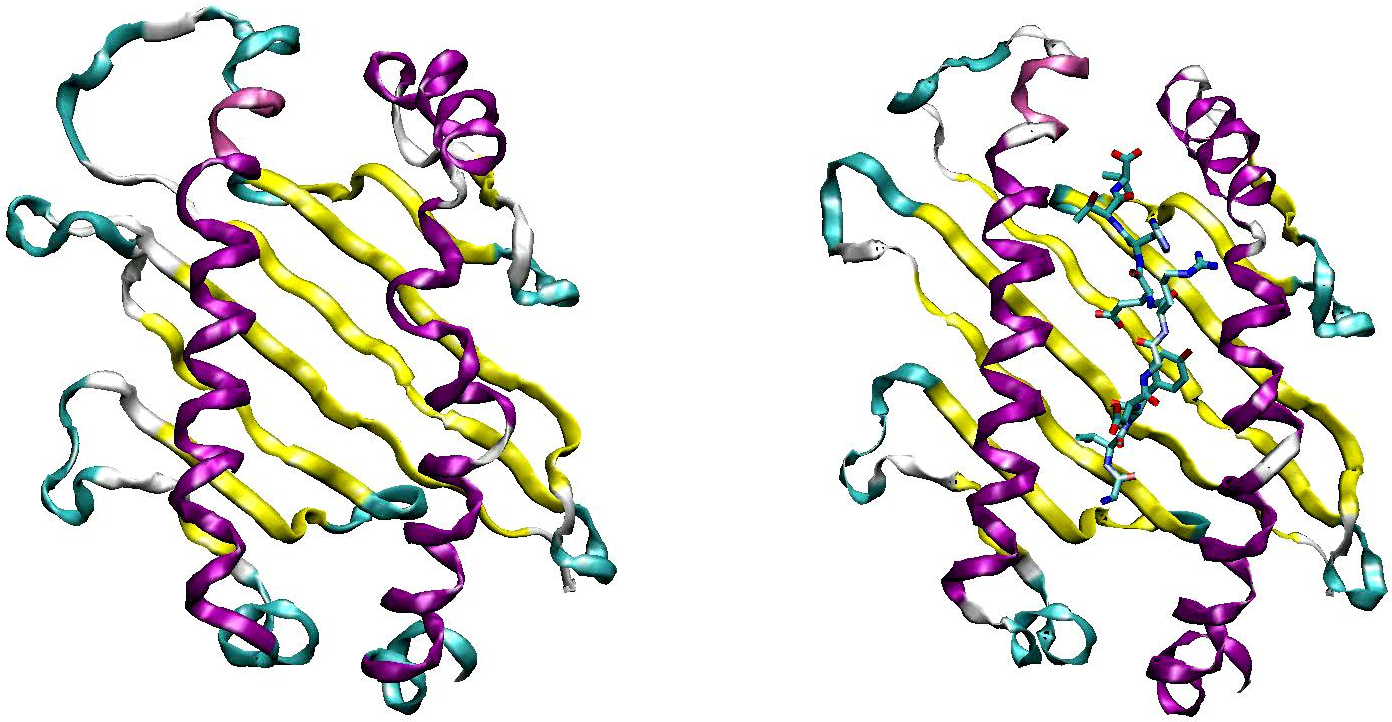
MHC: A Biological Virus Scanner

T lymphocytes targeting a cancer cell

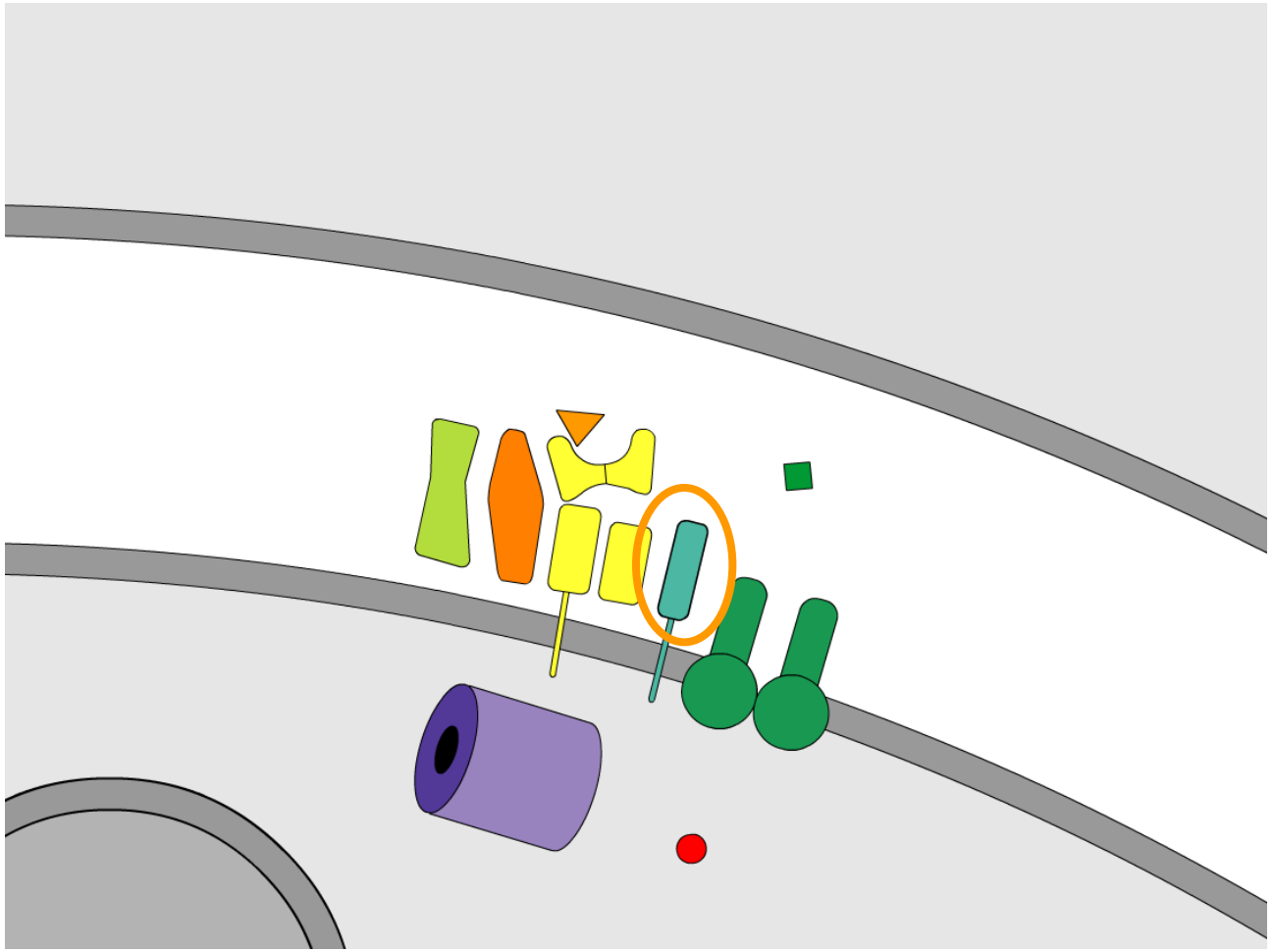


MHC I Structure

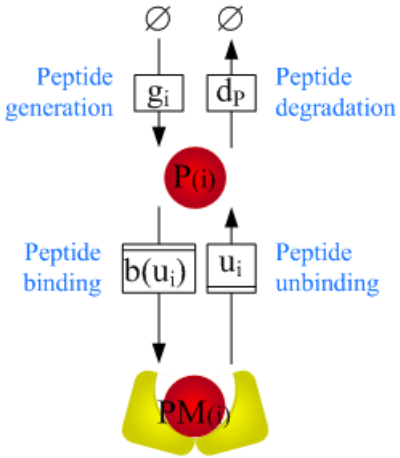
- Interaction of MHC I with peptide

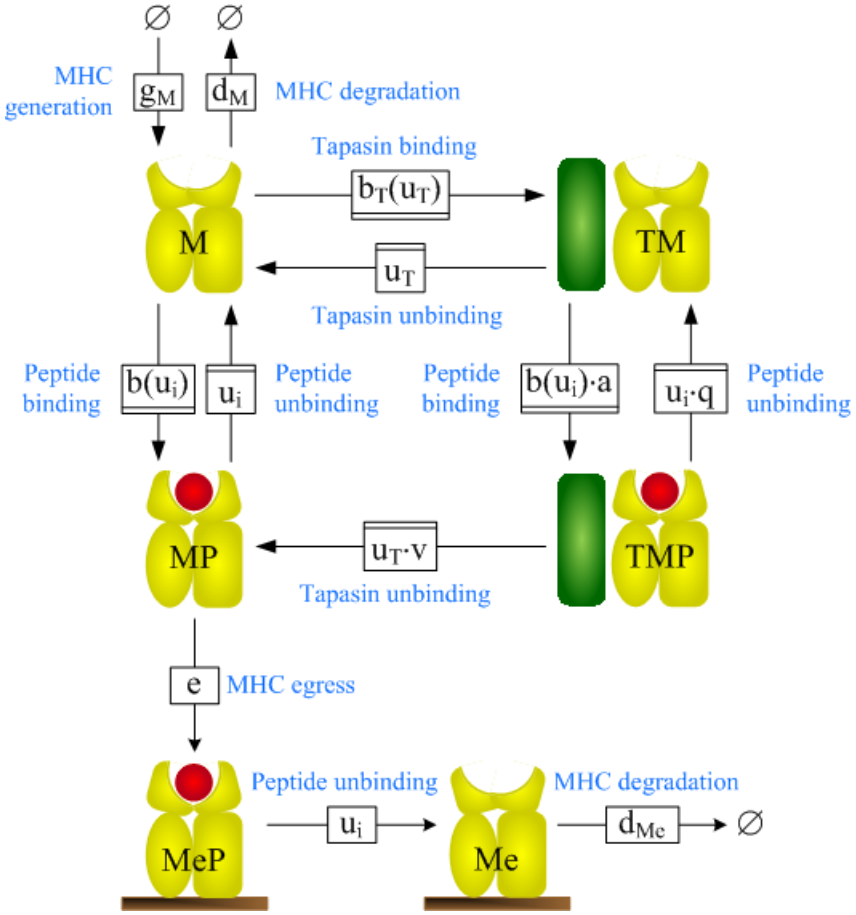


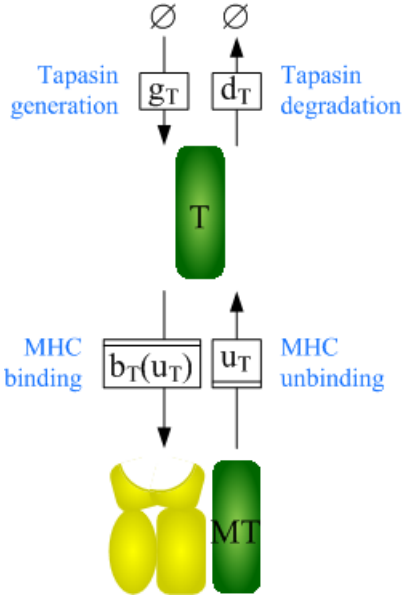
Tapasin affects relative presentation



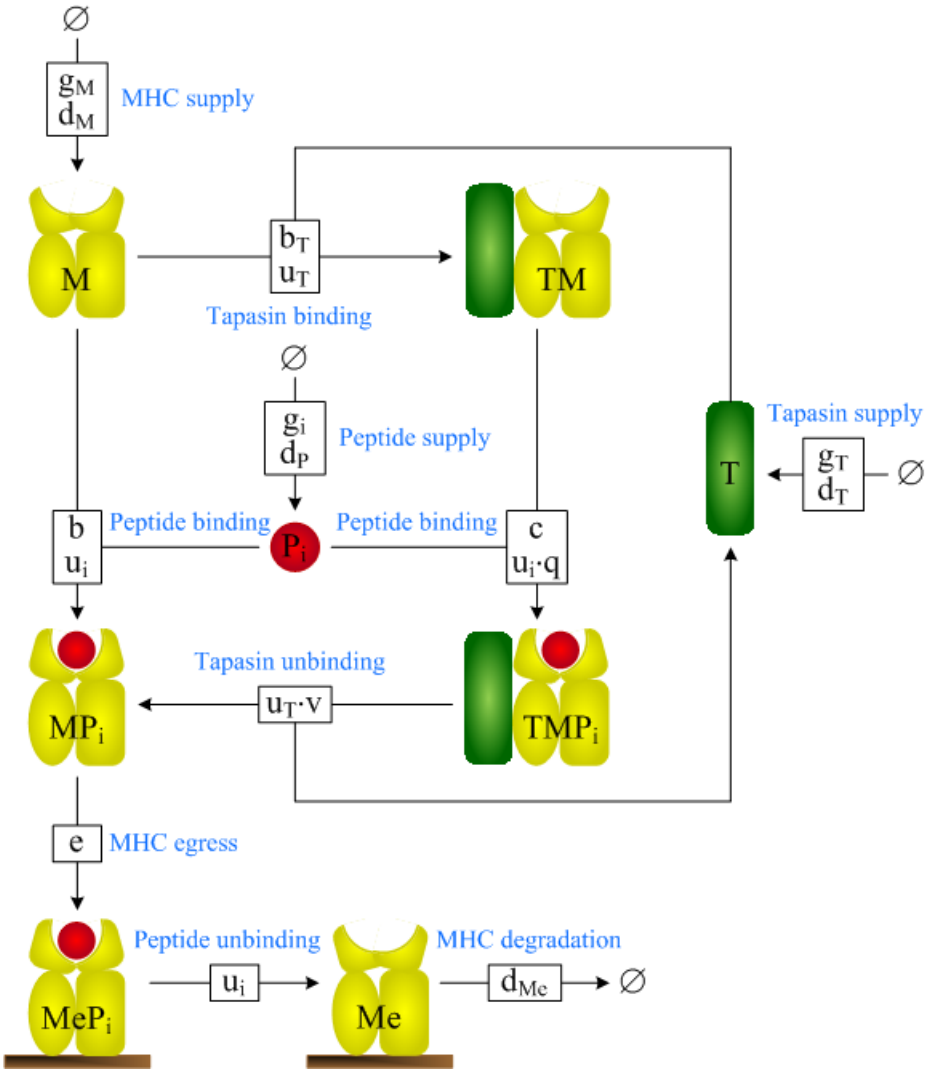
Individual-based model







Reaction-based model



$$\varnothing \underset{d_M}{\overset{g_M}{\rightleftarrows}} M$$

$$\varnothing \underset{d_T}{\overset{g_T}{\rightleftarrows}} T$$

$$M + T \underset{u_T}{\overset{b_T}{\rightleftarrows}} TM$$

$$Me \overset{d_{Me}}{\rightarrow} \varnothing$$

$$\varnothing \underset{d_P}{\overset{g_i}{\rightleftarrows}} P_i$$

$$M + P_i \underset{u_i}{\overset{b}{\rightleftarrows}} MP_i$$

$$TM + P_i \underset{u_i \cdot q}{\overset{c}{\rightleftarrows}} TMP_i$$

$$TMP_i \overset{u_T \cdot v}{\rightarrow} T + MP_i$$

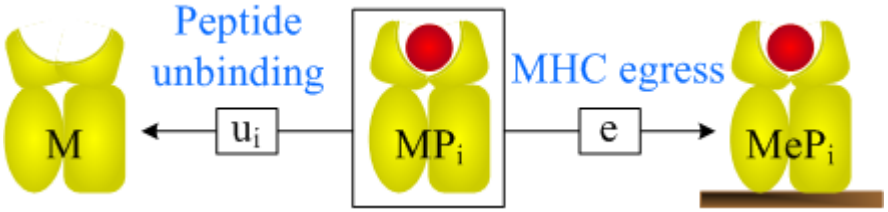
$$MP_i \overset{e}{\rightarrow} MeP_i$$

$$MeP_i \overset{u_i}{\rightarrow} Me$$

Principle of peptide filtering

A single peptide-MHC complex

Competition between unbinding and egress



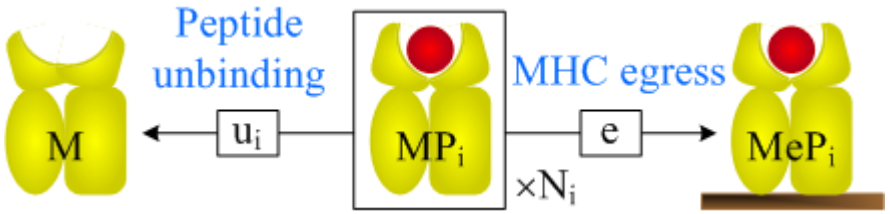
$$P(unbind) = \frac{u_i}{u_i + e}$$

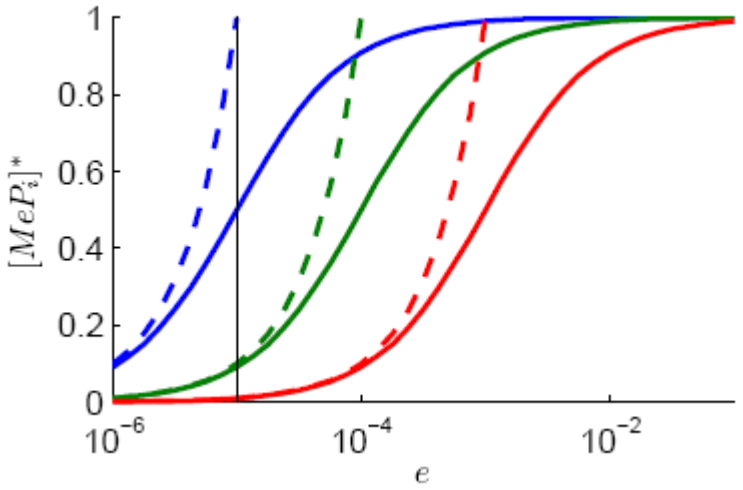
$$P(egress) = \frac{e}{u_i + e}$$

Principle of peptide filtering

Multiple peptide-MHC complexes

Determine expected number of egressed complexes

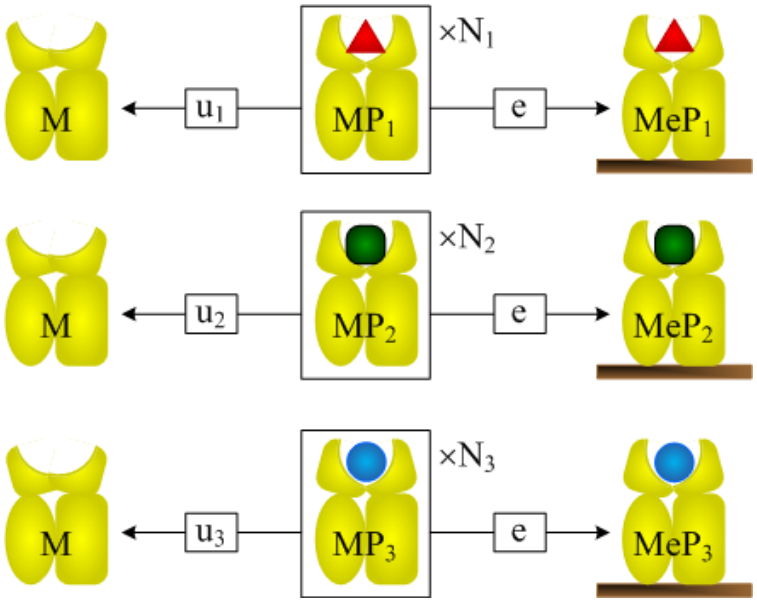


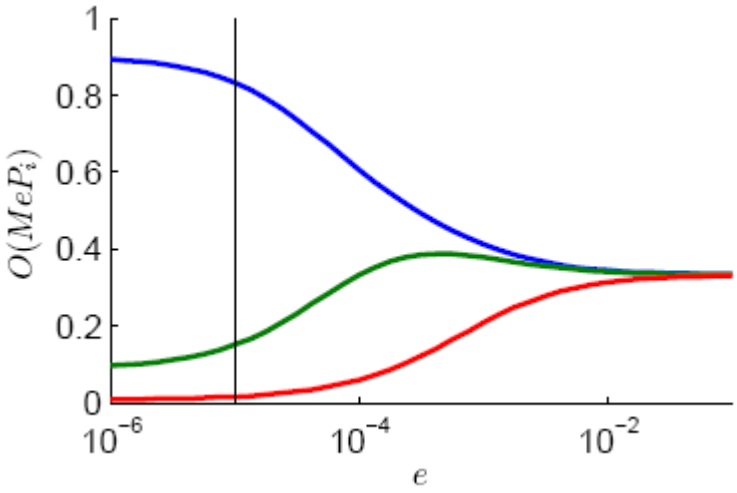


$$[MeP_i]^* = N_i \frac{e}{u_i + e}$$

Principle of peptide optimisation

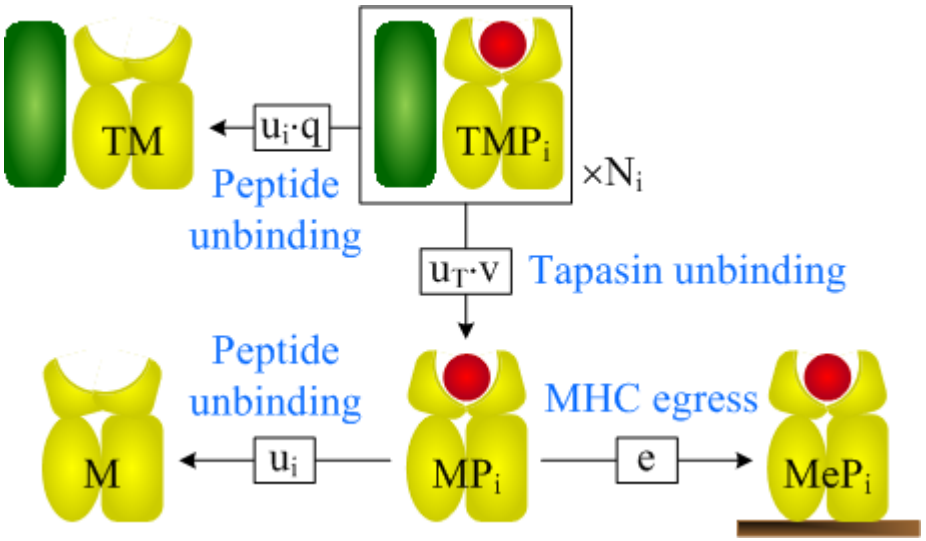
Populations of multiple peptide-MHC complexes





$$O(MeP_i) = \frac{[MeP_i]^*}{\sum_k [MeP_k]^*}$$

Peptide optimisation with tapasin



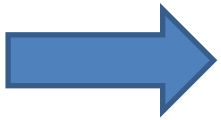
$$[MeP_i]^* = N_i \frac{(u_T v)}{u_i q + (u_T v)} \frac{e}{u_i + e}$$

$$= N_i \frac{x}{u_i + x} \frac{e}{u_i + e} \quad \overset{e,x \to 0}{\longrightarrow} \quad N_i \frac{ex}{u_i^2} \qquad x = \frac{u_T v}{q}$$

$$O(MeP_i) = \frac{[MeP_i]^*}{\sum_k [MeP_k]^*} \quad \overset{e,x \to 0}{\longrightarrow} \quad \frac{N_i / u_i^2}{\sum_k N_k / u_k^2}$$

ODE model

$$
\begin{aligned}
\varnothing &\underset{d_M}{\overset{g_M}{\rightleftarrows}} M \\
\varnothing &\underset{d_T}{\overset{g_T}{\rightleftarrows}} T \\
M + T &\underset{u_T}{\overset{b_T}{\rightleftarrows}} TM \\
Me &\xrightarrow{d_{Me}} \varnothing \\
\varnothing &\underset{d_P}{\overset{g_i}{\rightleftarrows}} P_i \\
M + P_i &\underset{u_i}{\overset{b}{\rightleftarrows}} MP_i \\
TM + P_i &\underset{u_i \cdot q}{\overset{c}{\rightleftarrows}} TMP_i \\
TMP_i &\xrightarrow{u_T \cdot v} T + MP_i \\
MP_i &\xrightarrow{e} MeP_i \\
MeP_i &\xrightarrow{u_i} Me
\end{aligned}
$$



$$
\begin{aligned}
\frac{d[M]}{dt} &= \sum_i u_i[MP_i] + u_T[TM] + g_M \\
&\quad - (b\sum_i [P_i] + b_T[T] + d_M)[M] && \textbf{[1]} \\
\frac{d[T]}{dt} &= u_T[TM] + g_T + u_T v \sum_i [TMP_i] - (b_T[M] + d_T)[T] && \textbf{[2]} \\
\frac{d[MP_i]}{dt} &= b[M][P_i] + u_T v[TMP_i] - (u_i + e)[MP_i] && \textbf{[3]} \\
\frac{d[TM]}{dt} &= b_T[M][T] + q\sum_i u_i[TMP_i] - (u_T + c\sum_i [P_i])[TM] && \textbf{[4]} \\
\frac{d[TMP_i]}{dt} &= ba[TM][P_i] - (u_i q + u_T v)[TMP_i] && \textbf{[5]} \\
\frac{d[P_i]}{dt} &= u_i[MP_i] + u_i q[TMP_i] + g_i \\
&\quad - (b[M] + c[TM] + d_P)[P_i] && \textbf{[6]} \\
\frac{d[MeP_i]}{dt} &= e[MP_i] - u_i[MeP_i] && \textbf{[7]} \\
\frac{d[Me]}{dt} &= \sum_i u_i[MeP_i] - d_{Me}[Me] && \textbf{[8]}
\end{aligned}
$$

ODE analysis of peptide filtering

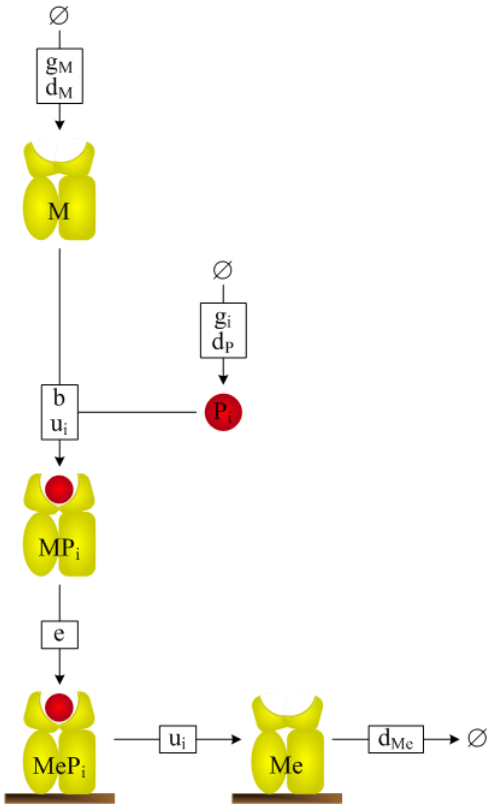
$$[MeP_i]^* = \frac{1}{u_i}\frac{e}{u_i + e}(b[M]^* + \frac{x}{u_i + x}c[TM]^*)[P_i]^*$$

$$[TM]^* \gg [M]^* \qquad [P_i]^* \approx g_i/d_P$$

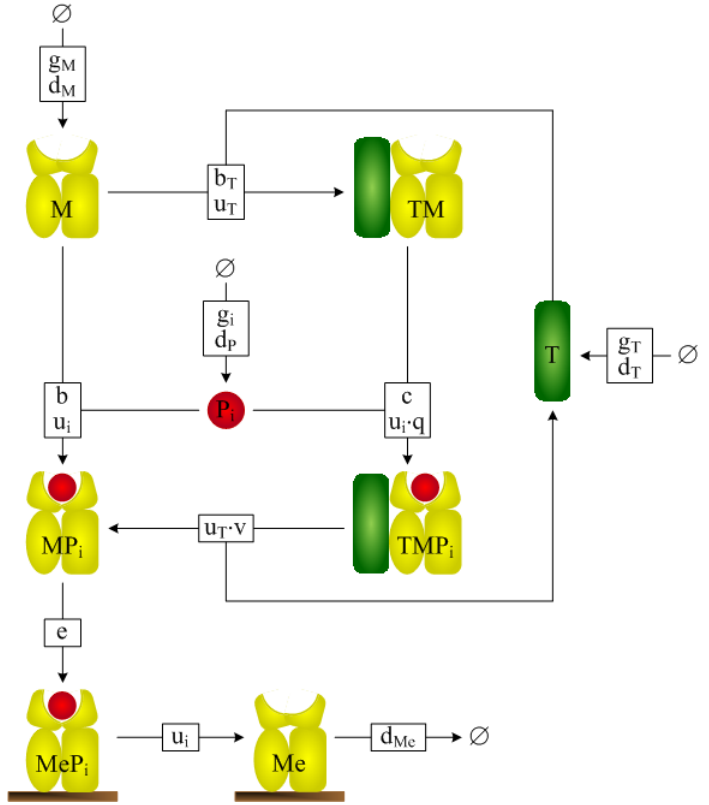
$$[MeP_i]^* \approx \underbrace{C}_{\text{Supply}} \; \underbrace{g_i}_{\text{Tapasin}} \; \underbrace{\frac{x}{u_i + x}}_{\text{ER}} \; \underbrace{\frac{e}{u_i + e}}_{\text{Surface}} \; \frac{1}{u_i}$$

$$x = u_T v/q \qquad C = c[TM]^*/d_P$$

ODE analysis of peptide optimisation



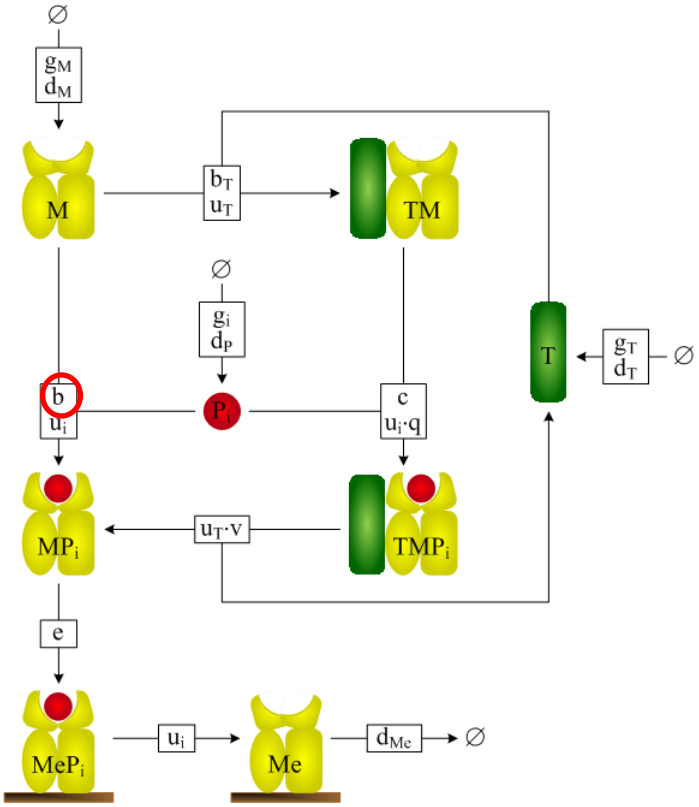
$$\frac{[MeP_i]^*}{\sum_k [MeP_k]^*} = \frac{g_i/(u_i(u_i+e))}{\sum_k g_k/(u_k(u_k+e))} \xrightarrow{e,x\to 0} \frac{g_i/u_i^2}{\sum_k g_k/u_k^2}$$

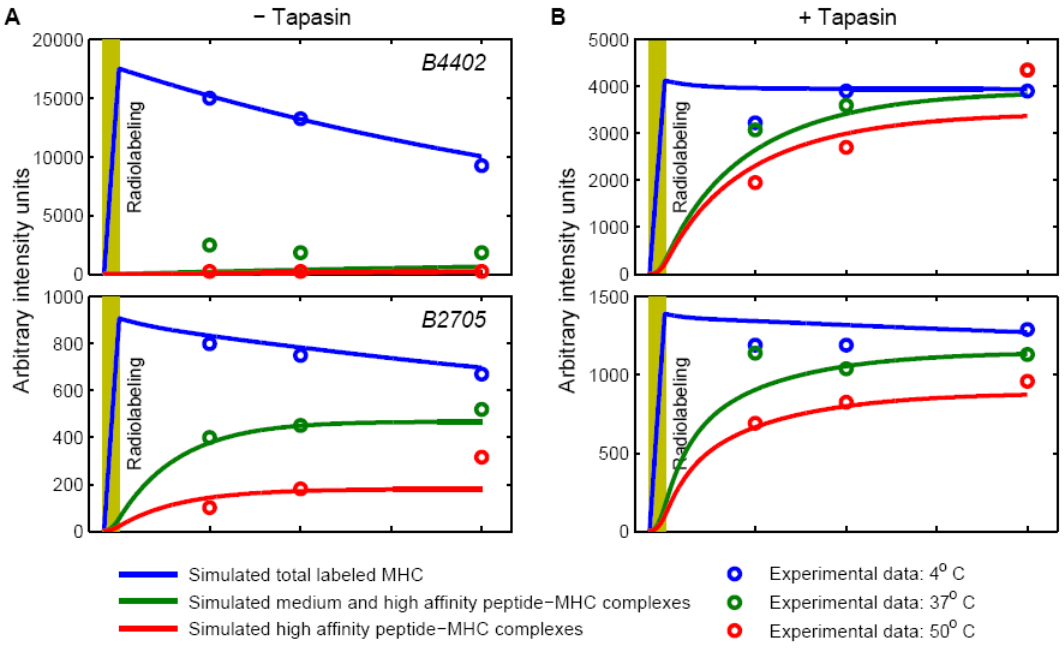


$$\frac{[MeP_i]^*}{\sum_k [MeP_k]^*} = \frac{g_i/(u_i(u_i+e)(u_i+x))}{\sum_k g_k/(u_k(u_k+e)(u_k+x))} \xrightarrow{e,x\to 0} \frac{g_i/u_i^3}{\sum_k g_k/u_k^3}$$

Peptide optimisation over time

Three representative peptides P_{low}, P_{med}, P_{high}



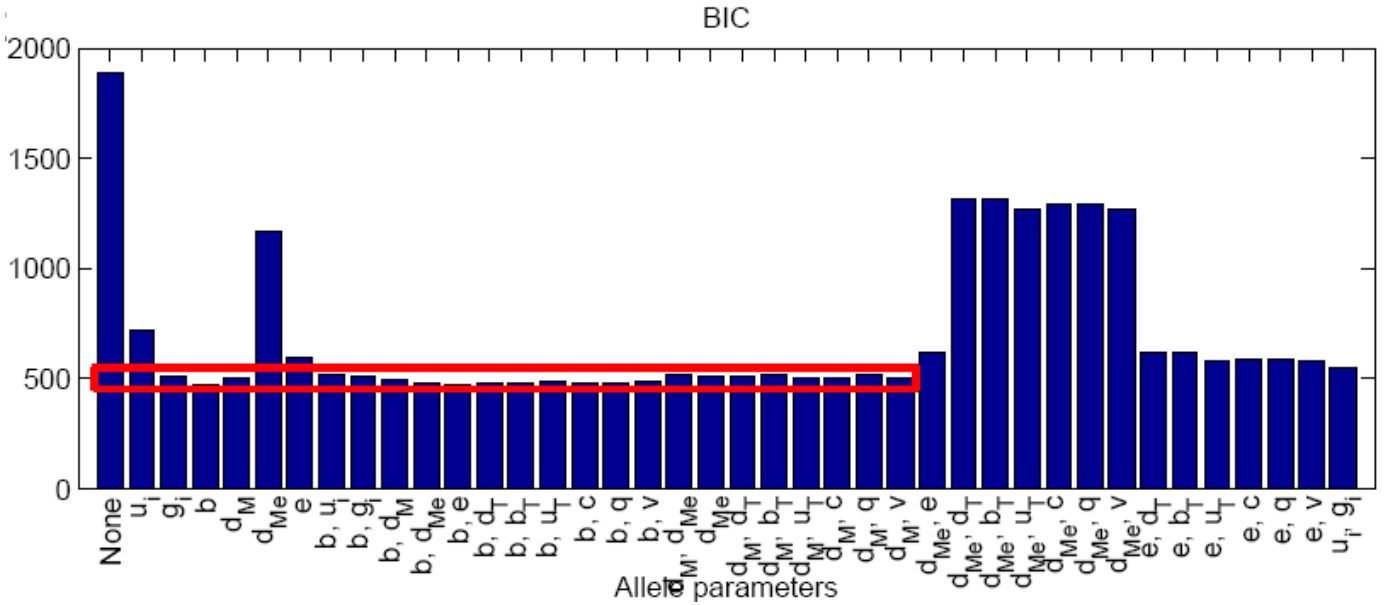


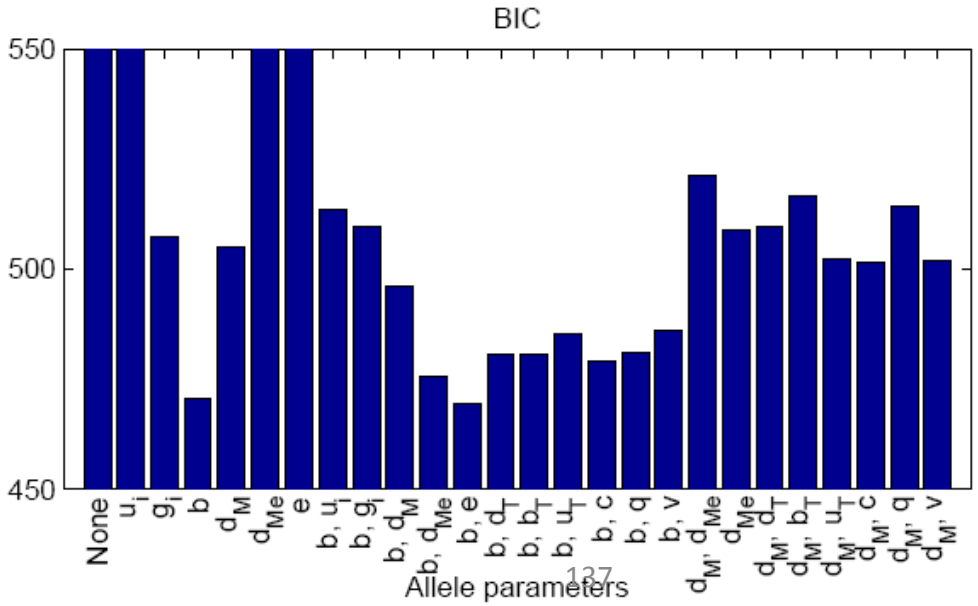
Experiments by Williams...Elliott, Immunity 2002

Model Parameters

Description	Parameter	Measured	Range	M_b
Production of tapasin in the ER	g_T		Fixed	
Degradation of tapasin in the ER	d_T		$10^{-6} - 10^{-2}$	1.726×10^{-3}
Production of MHC in the ER	g_M		Fixed	
Degradation of MHC in the ER	d_M	$2 - 3 \times 10^{-4}\ s^{-1}$ [11, 12]	$10^{-6} - 10^{-1}$	7.989×10^{-5}
Degradation of MHC at the cell surface	d_{Me}	$2.4 \times 10^{-4}\ s^{-1}$ [6]	$10^{-6} - 10^{-1}$	9.329×10^{-5}
Degradation of peptides in the ER	d_P	$0.13\ s^{-1}$ [5]	Fixed	1.3×10^{-1}
Binding of tapasin to MHC	b_T		$10^{-11} - 10^{-5}$	1.663×10^{-9}
Unbinding of tapasin from empty MHC	u_T		$10^{-6} - 10^{-1}$	1.185×10^{-6}
Binding of peptide to tapasin-bound MHC	c		$10^{-8} - 10^{-2}$	8.303×10^{-8}
Effect of tapasin on peptide-MHC unbinding	q		$1 - 10^{5}$	2.104×10^{4}
Effect of peptide on tapasin-MHC unbinding	v		$1 - 10^{3}$	9.363×10^{2}
Binding of peptide to empty MHC	b_{B4402}	$0.2 - 2 \times 10^{6}\ M^{-1}s^{-1}$ [11, 13]	$10^{-11} - 10^{-5}$	3.177×10^{-11}
	b_{B2705}		$10^{-11} - 10^{-5}$	1.945×10^{-9}
	b_{B4405}		$10^{-11} - 10^{-5}$	4.367×10^{-9}
Egression of loaded MHC from the ER	e		$10^{-4} - 1$	1.142×10^{-1}
Unbinding of peptides from MHC	u_{low}	$7.8 \times 10^{-6} - 4 \times 10^{-3}\ s^{-1}$ [13]	$10^{-8} - 10^{-2}$	8.764×10^{-4}
	u_{medium}		$10^{-8} - 10^{-2}$	5.658×10^{-6}
	u_{high}		$10^{-8} - 10^{-2}$	4.177×10^{-7}
Active transport of peptides into ER	g_{low}	$1.3 - 3.3 \times 10^{-6}\ M\ s^{-1}$ [5]	$1 - 10^{5}$	2.093×10^{4}
	g_{medium}		$1 - 10^{5}$	1.759×10^{4}
	g_{high}		$1 - 10^{5}$	1.064×10^{4}

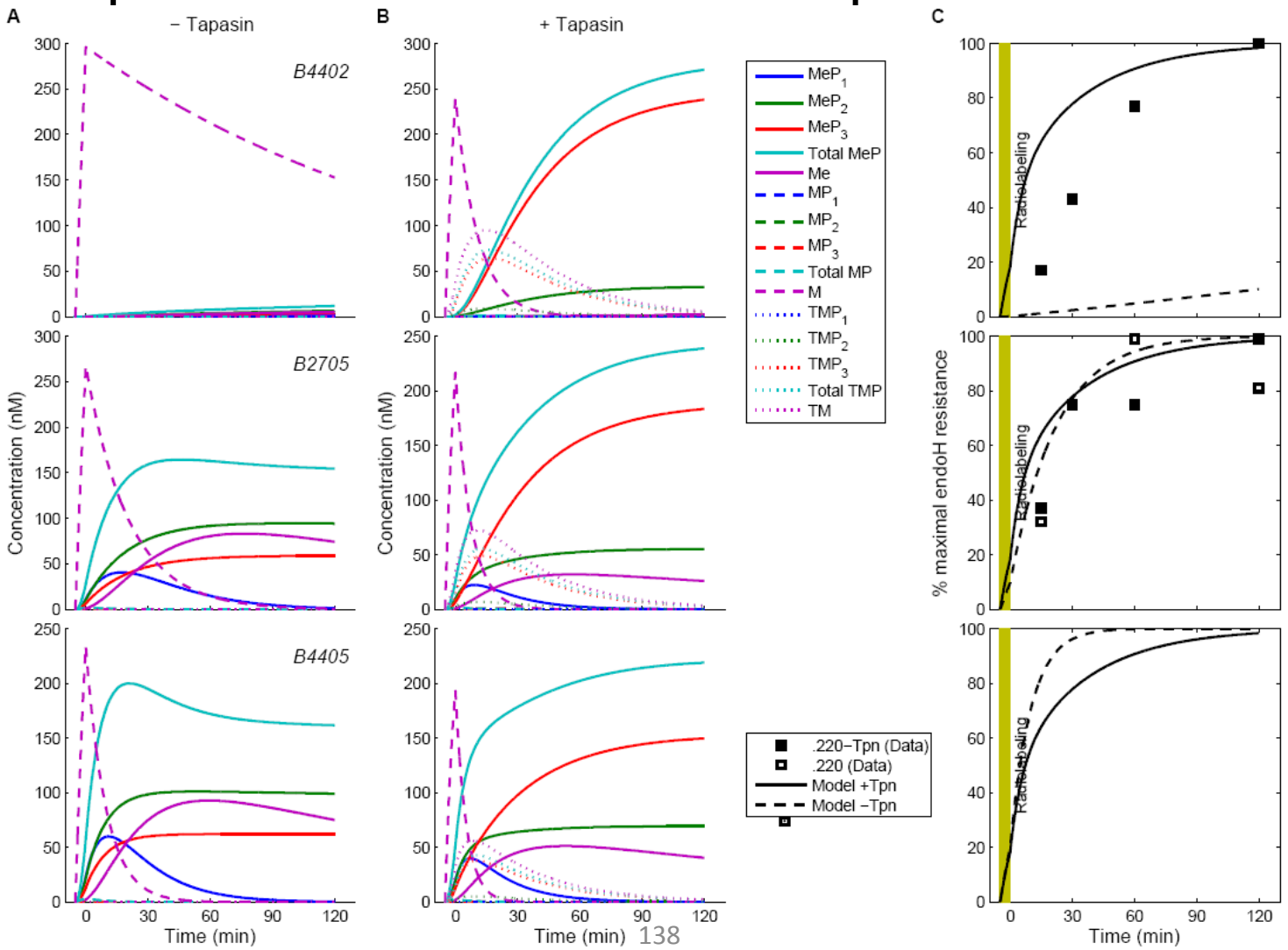
Fitting Parameters to Data





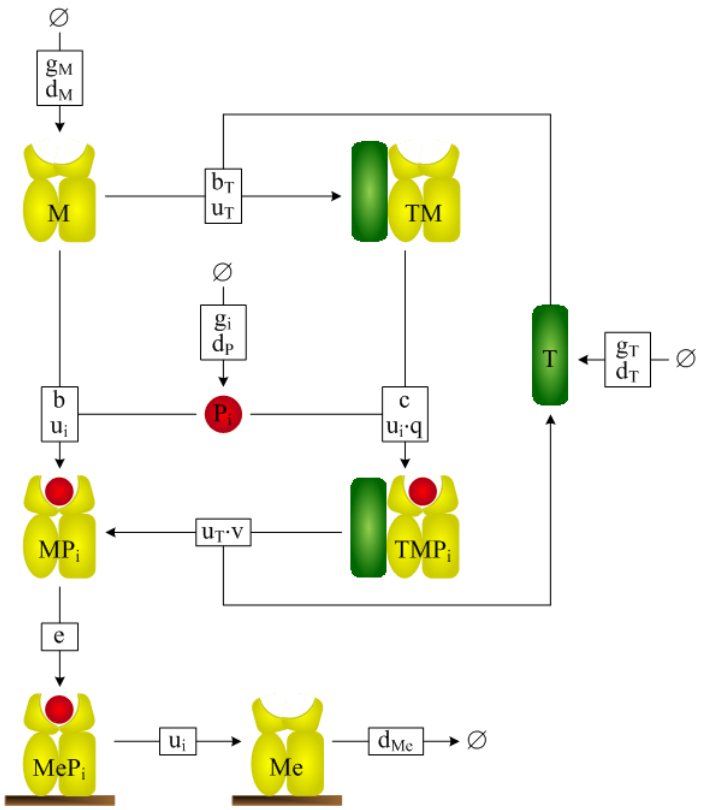
Peptide optimisation over time

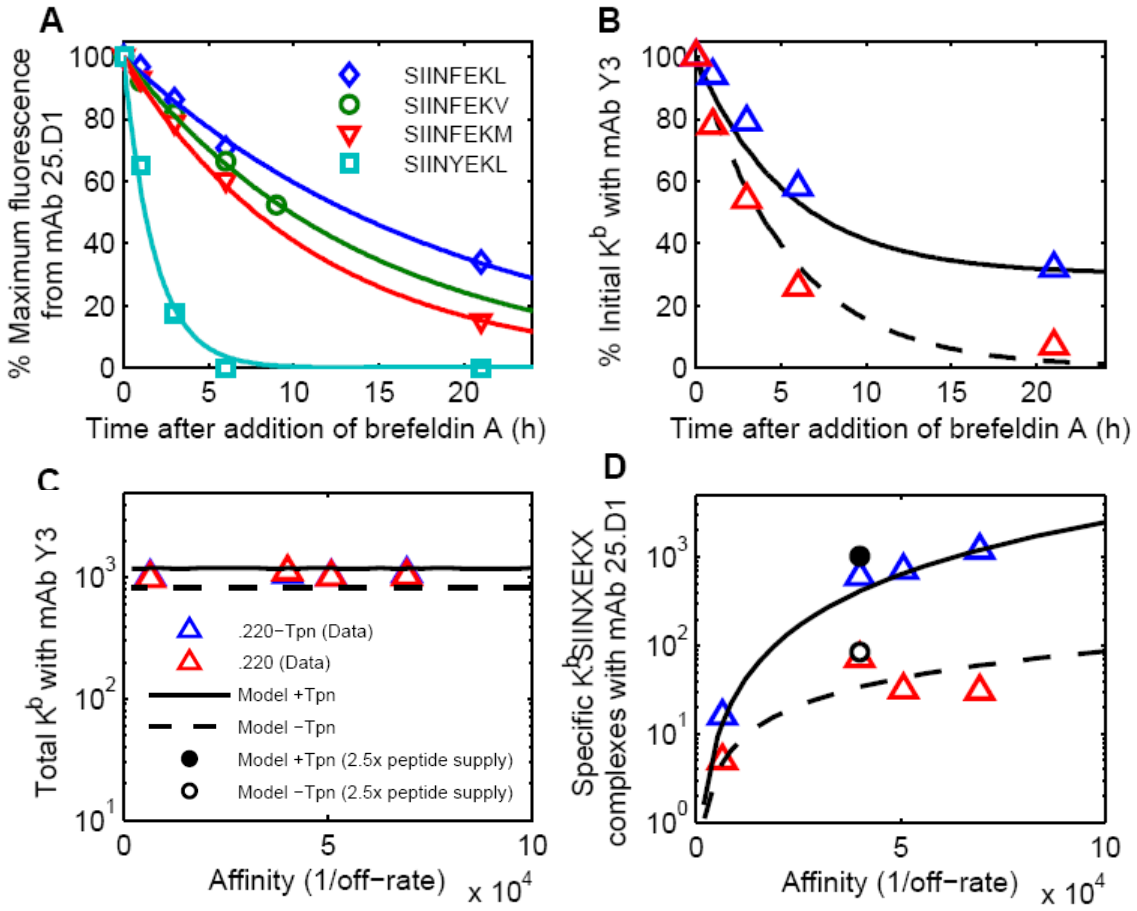
Separate plots for different MHC complexes



Peptide optimisation at steady-state

A SIINFEKL peptide and 2 background peptides

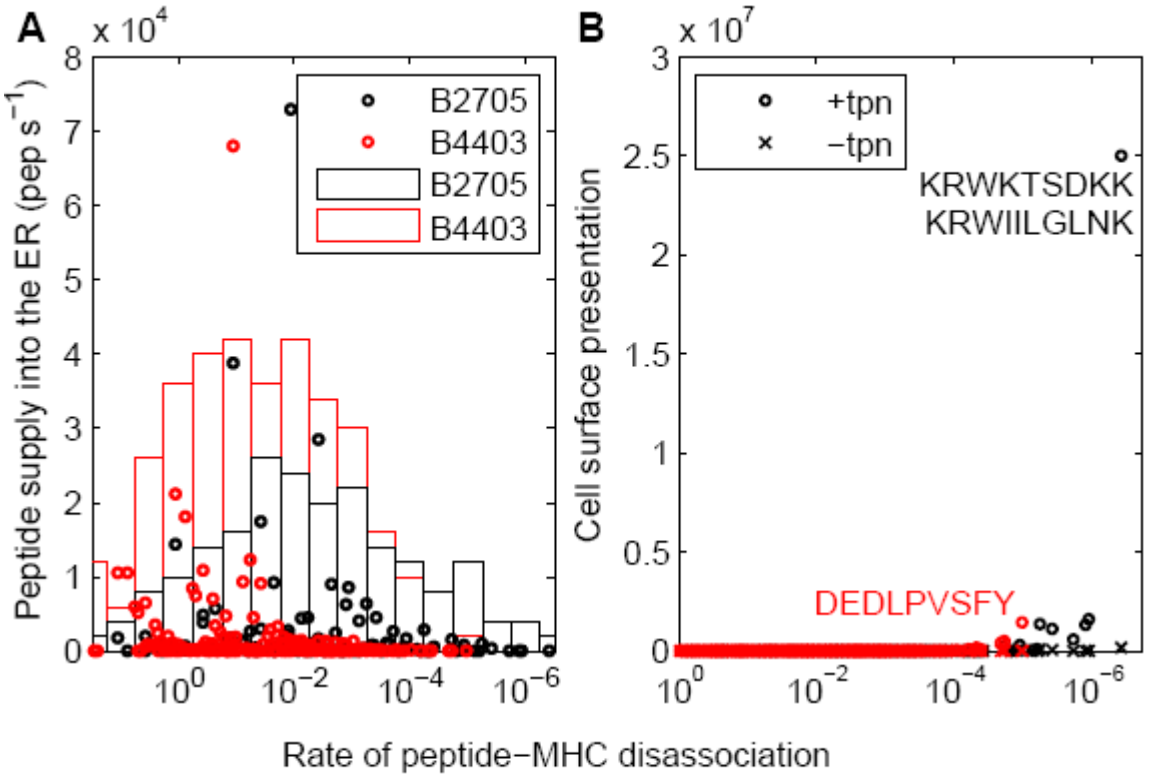




Experiments by Howarth...Elliott, PNAS 2004

Optimisation of HIV peptides

- Chop up protein sequence of HIV into peptides
- Predict presentation from peptide off-rates and abundance.



Summary

- A kinetic model of MHC class I antigen presentation

  interactions with the chaperone molecule tapasin.

- Principle of peptide filtering

  quantify peptide optimisation as a function of peptide supply and peptide unbinding rates.

- Tapasin improves peptide optimisation

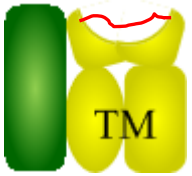
  by accelerating peptide unbinding.

- Peptide optimisation across MHC class I alleles

  can be explained by differences in peptide binding.

Scientific Challenges

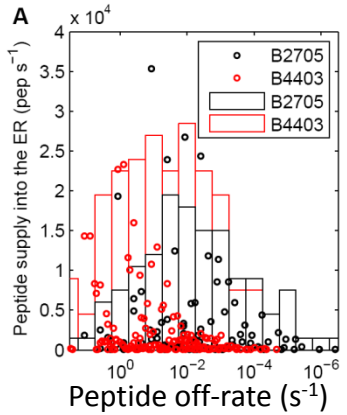
- Predict the immune response to a given virus



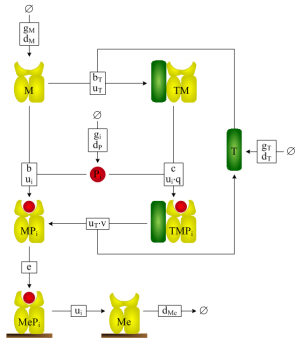
BIMAS Off-rate predictor

MPLSYQHFRKLLLLDDEAGPLEEELPRLA....

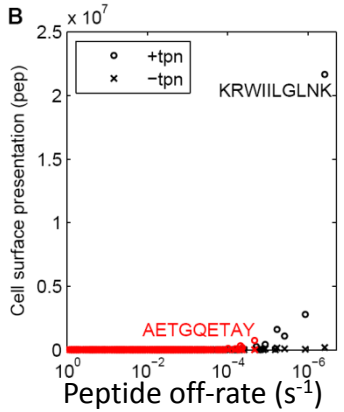
Viral protein sequence



Off-rates and abundance



Kinetic model



Cell-surface presentation

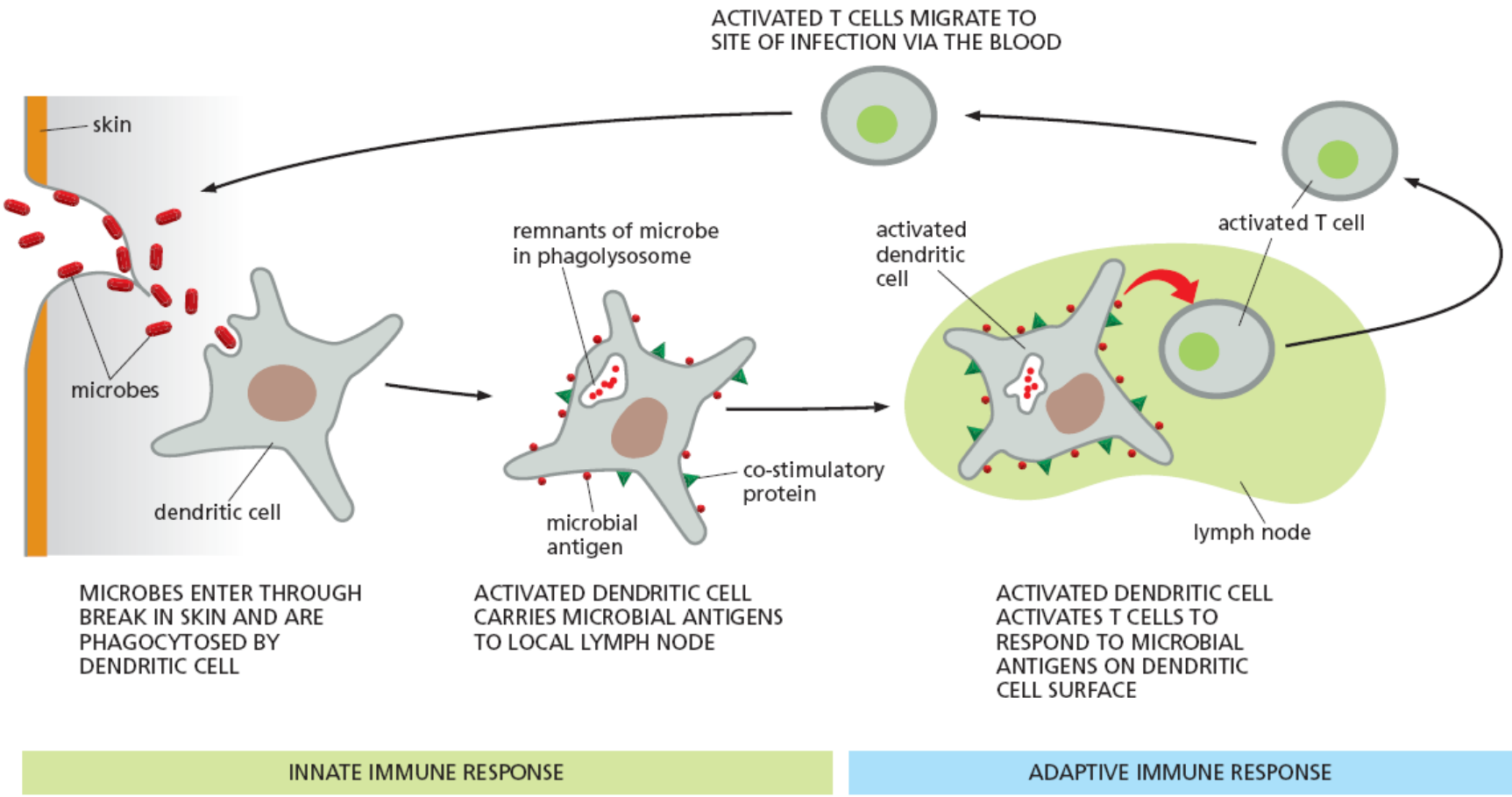
- Towards a virtual immune system

Future Work

- Constrain peptide editing model with additional experimental results
- Predict effects of tapasin for different alleles (HLA-B8, H2-K^b)
- Include additional chaperones and MHC conformational changes.
- Unify peptide competition in the ER, presentation at the cell surface and T-cell activation for H2-K^b

Future Work

Predicting the adaptive immune response



Tutorial Summary

- Programming DNA Computers
  - **Microsoft Research:** Matthew Lakin, Luca Cardelli
  - **University of Munich:** Simon Youssef
- Programming Genetic Devices
  - **Microsoft Research:** Neil Dalchau
  - **University of Edinburgh:** Michael Pedersen
  - **University of Cambridge:** James Brown
- Programming the Immune System
  - **Microsoft Research:** Neil Dalchau, Luca Cardelli
  - **University of Cambridge:** Leonard Goldstein
  - **University of Southampton:** Tim Elliott, Joern Werner

Thanks

MSR Computational Science Lab

Focus

Research and development of novel computational approaches to tackle fundamental problems in science in areas of societal importance

Groups

- Computational Biology

  methods to understand how living things work

- Computational Ecology & Environmental Science

  methods to understand the structure, function and future of Life on Earth

- Technology & Tools Group

  Implement next generation of software tools for science