Unconventional Computing and Systems Biology, Finite Automata

Andrei Paun

Dept of Artificial Intelligence, UPM

John Jack

North Carolina State University

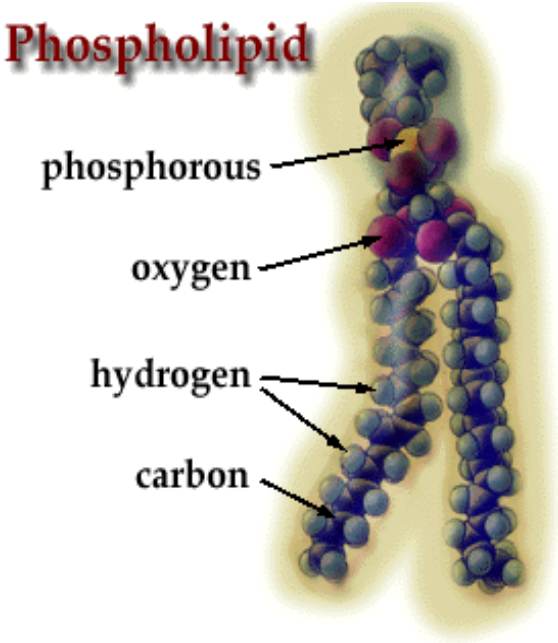
Summary

- 1. Unconventional Computing (UC)
  - Spiking P Systems
  - Proteins on membranes
- 2. Systems Biology (SB)
  - Discrete simulations
- Simulating latency in HIV
- 3. Finite Automata (FA)
  - Cover automata
  - Results

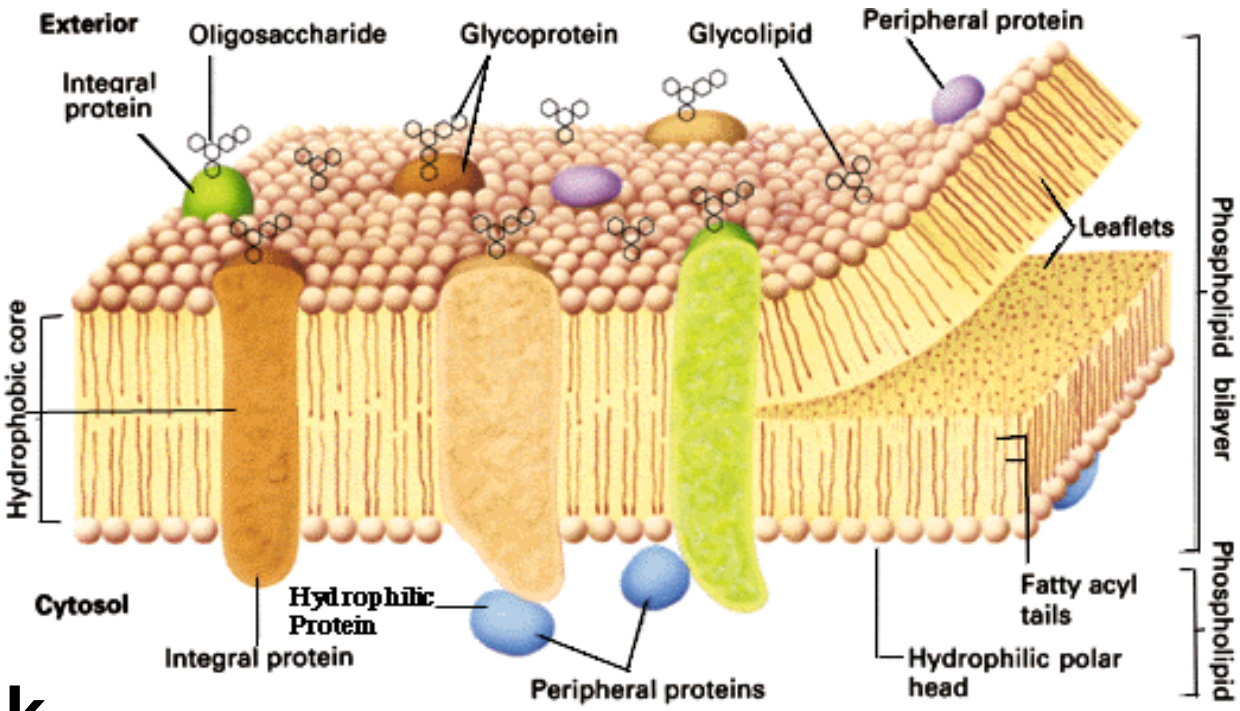
1 UC: motivation

- Compute with cells
- Now we can manipulate cells better
- Parallelism
- Limits to the current silicone computer

Membrane's Structure



Building block



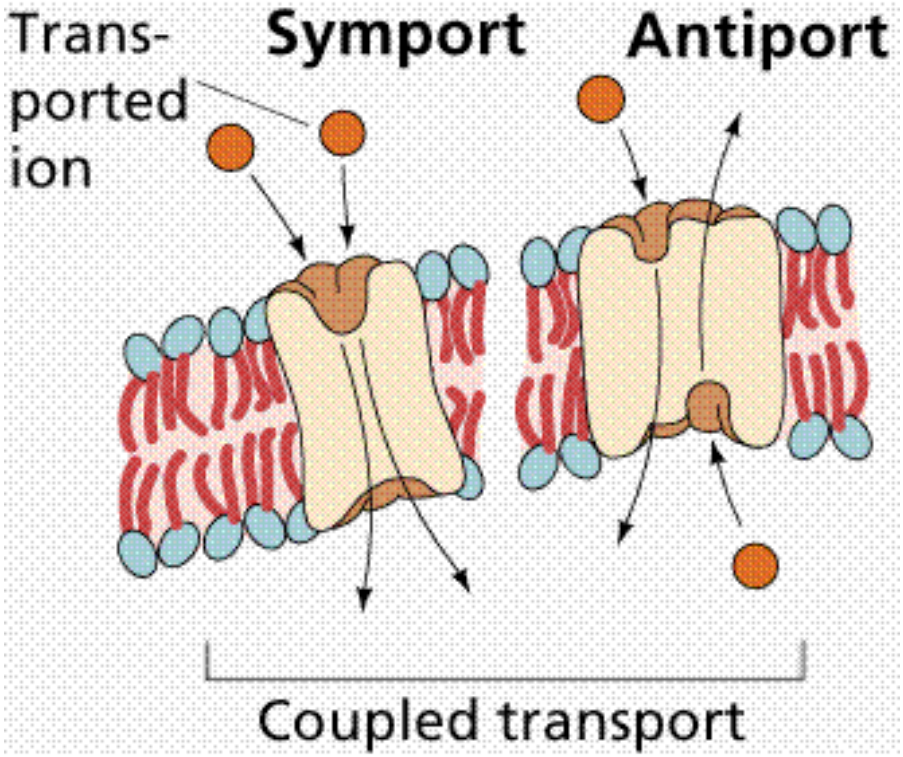
Transversal view

Picture from http://ntri.tamuk.edu/cell/membranes.html

Trans-membrane transport

- trans-membrane transfer of molecules can take place in three main ways:
  - **active transport**
  - passive transport
  - vesicle-mediated transport

Active transport



Picture from
www.emc.maricopa.edu/…/BIOBK

Done through protein channels

Membrane Computing

- biochemical inspiration (compute with cells)
- parallel computing devices
- distributed computing devices
- many variants
  - Symport/antiport; Traces
  - Timed systems; with proteins

P System components

- membrane structure: several cell-membranes hierarchically embedded in a main membrane: the skin membrane
- the output membrane: one elementary membrane specified as the output
- regions (delimited by membranes)-contain objects and evolution rules

Symport/Antiport P Systems

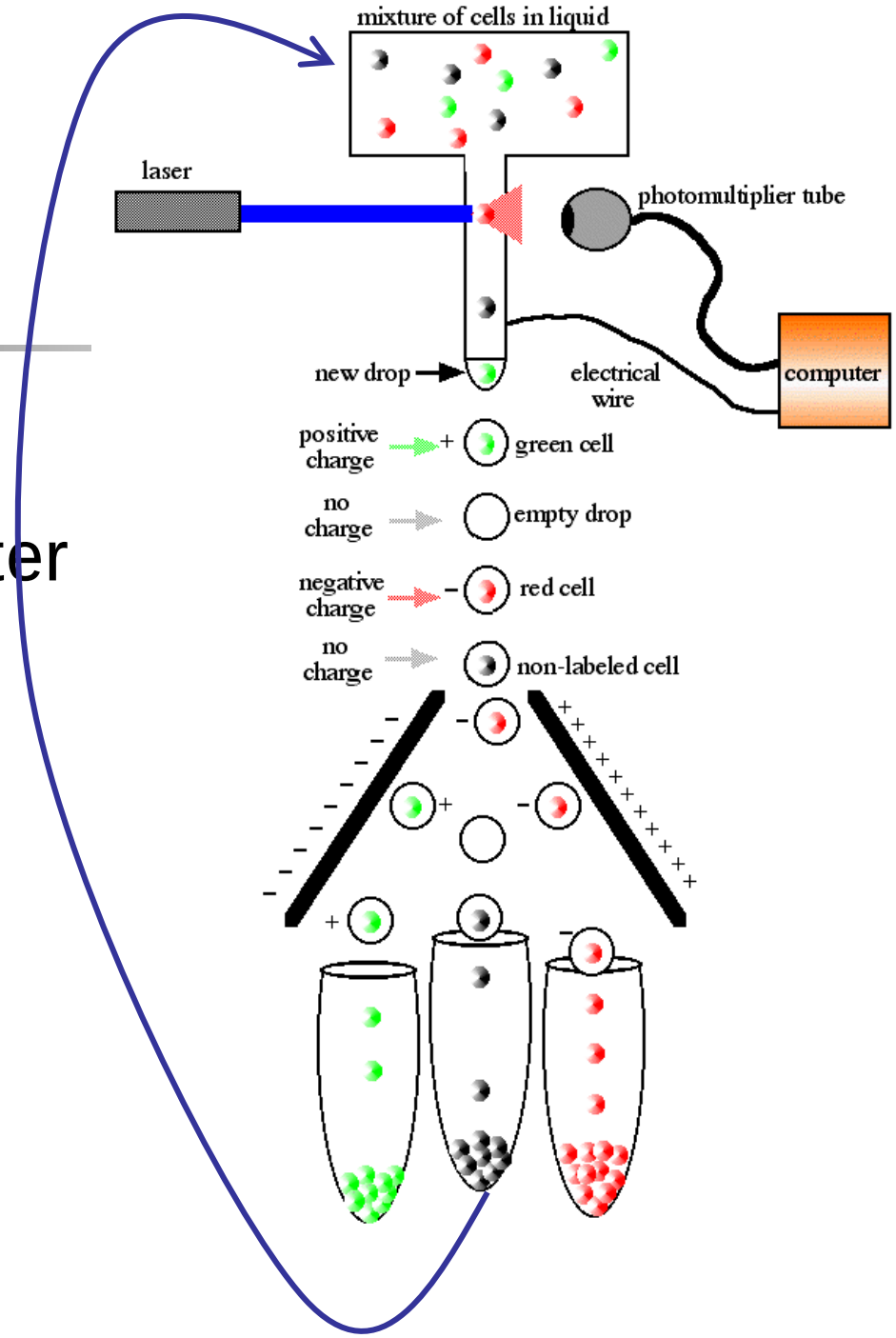
- computation by communication only (no creation/destruction of objects)
- computational universality
- the rules correspond to well-known biochemical processes
- conservation law is observed
- the environment is an active participant to the computation

Timed P systems: motivation

- Closer to “nature” and biomolecular tools and techniques
- Time as support for computation
- Why time?
  - Cell compute=cell accumulate the result
  - Cell unhappy
  - Cell adapt and behave unpredictably

FACS

- **F**luorescence **A**ctivated **C**ell **S**orter
- cells “undisturbed”
- a “feedback” mechanism is possible



Timed symport/antiport systems

- Normal symport/antiport systems
- The output could be considered also for non-halting computations
- The result is the time it takes the system to go from one pre-set configuration to the second pre-set configuration

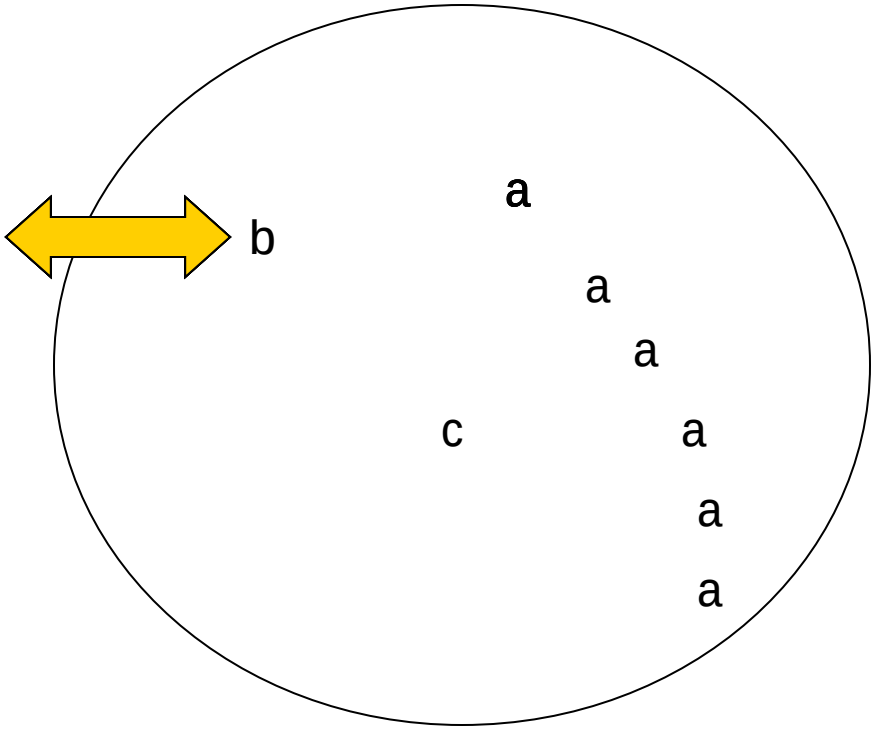
[Ibarra 2005] O.H. Ibarra, A. Paun, Counting Time in Computing with Cells, *DNA11 conference*, 2005

Example

$$C_{start} = abc$$

$$C_{stop} = a^5$$

a b



- Result: 4

(b,out; ab,in)

Timed systems results

- "Normal" symport/antiport systems results:

$$\mathrm{NOP}_3(sym_1, anti_1) = NRE - \{0,1,2,3,4\}$$ [Vaszil 2004]

$$\mathrm{NOP}_4(sym_1, anti_1) = NRE$$ [Frisco 2004]

$$\mathrm{NOP}_1(sym_0, anti_2) = NRE$$ [Freund; Frisco 2002]

- Timed systems results:

$$\mathrm{NTP}_3(sym_1, anti_1) = NRE$$

$$\mathrm{NTP}_1(sym_0, anti_2) = NRE$$

1 UC: proteins on membranes

Membranes with proteins

- Cell communication is done mostly using proteins
- Symport/antiport are performed through protein channels (limited parallelism)

- Defined in: A. Paun, B. Popa, P Systems with Proteins on Membranes, DLT 2006.

Motivation of research

- Extension to Symport/Antiport systems
- SA systems are widely studied but contain some non-natural features

- Max. parallelism forces us to forbid rules (a,in) for skin membrane and a in E

Motivation (contd.)

- We want to capture also the catalytic/enzymatic properties of trans-membrane proteins or the ones localized at the membranes

- Current estimates put the number of these proteins at about 50% of the total proteins of a cell

Motivation (contd.)

- The reactions helped by the membrane proteins cannot happen in a massively parallel manner

- The number of the proteins impose the upper bound for the number of reactions applied simultaneously

Types of rules

- Res

Type	Rule	Effect
1res	$[_i p \mid a \rightarrow [_i p \mid b$	
	$a[_i p \mid \rightarrow b[_i p \mid$	modify an object, but not move
2res	$[_i p \mid a \rightarrow a[_i p \mid$	
	$a[_i p \mid \rightarrow [_i p \mid a$	move an object, but not modify
3res	$[_i p \mid a \rightarrow b[_i p \mid$	
	$a[_i p \mid \rightarrow [_i p \mid b$	modify and move one object
4res	$a[_i p \mid b \rightarrow b[_i p \mid a$	interchange two objects
5res	$a[_i p \mid b \rightarrow c[_i p \mid d$	interchange and modify two objects

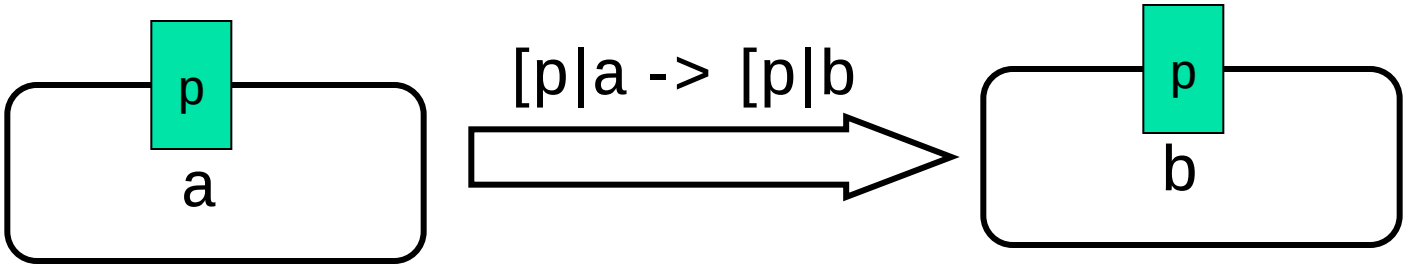
Types of rules

- Cp

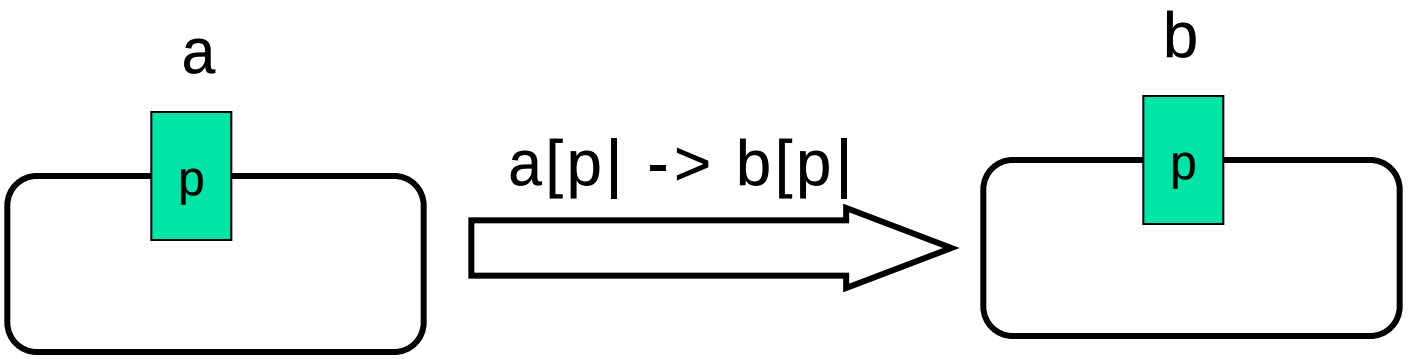
Type	Rule	Effect (besides changing also the protein)
1cp	$[_i p \mid a \rightarrow [_i p' \mid b$	
	$a[_i p \mid \rightarrow b[_i p' \mid$	modify an object, but not move
2cp	$[_i p \mid a \rightarrow a[_i p' \mid$	
	$a[_i p \mid \rightarrow [_i p' \mid a$	move an object, but not modify
3cp	$[_i p \mid a \rightarrow b[_i p' \mid$	
	$a[_i p \mid \rightarrow [_i p' \mid b$	modify and move one object
4cp	$a[_i p \mid b \rightarrow b[_i p' \mid a$	interchange two objects
5cp	$a[_i p \mid b \rightarrow c[_i p \mid d$	interchange and modify two objects

Examples

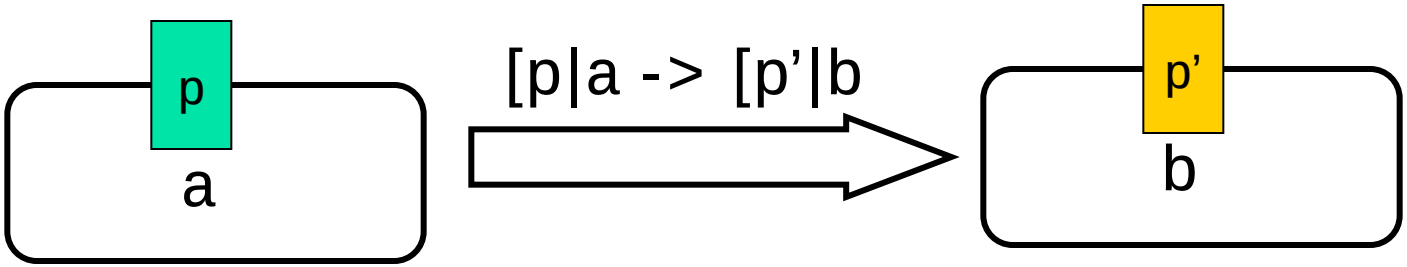
1res:



1res:

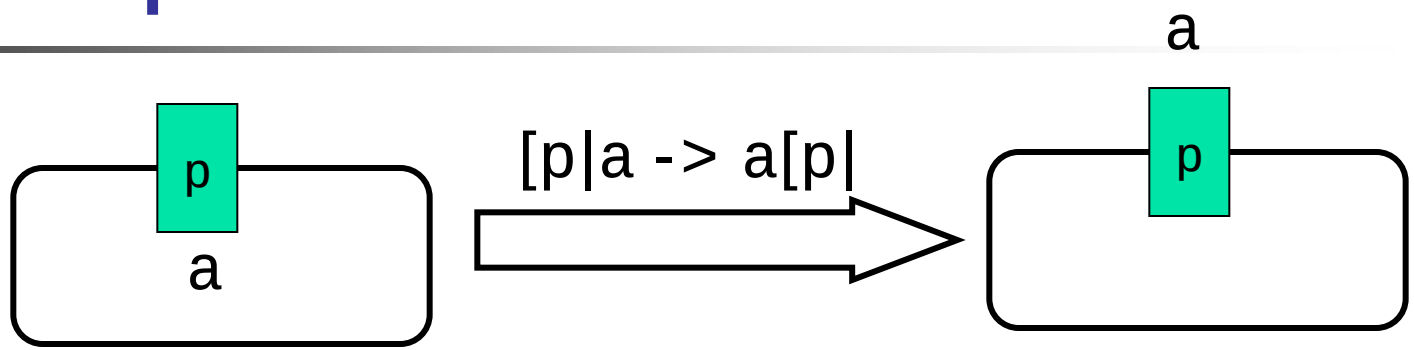


1cp:

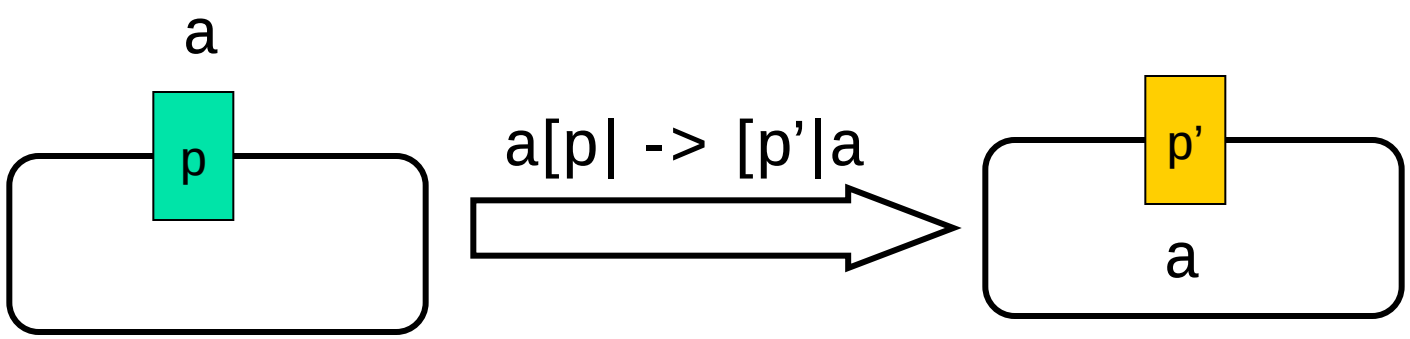


Examples

2res:

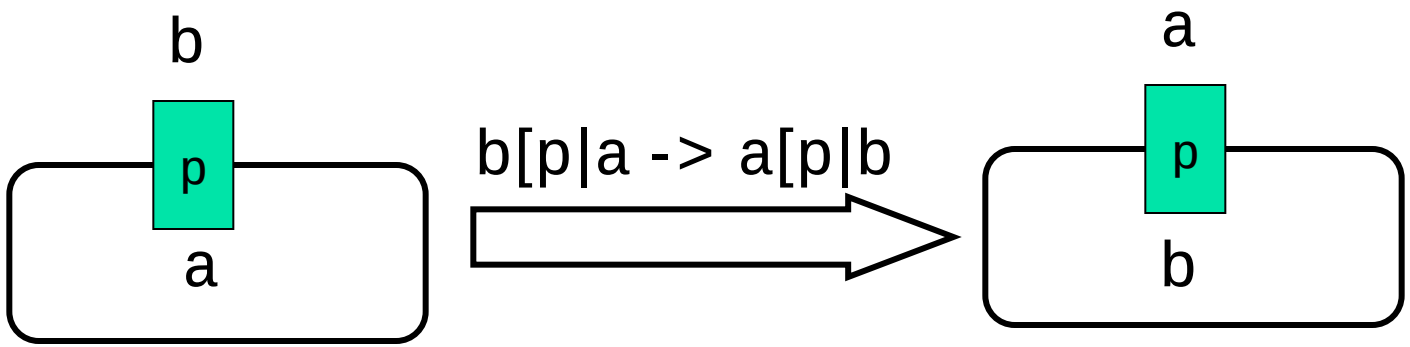


2cp:

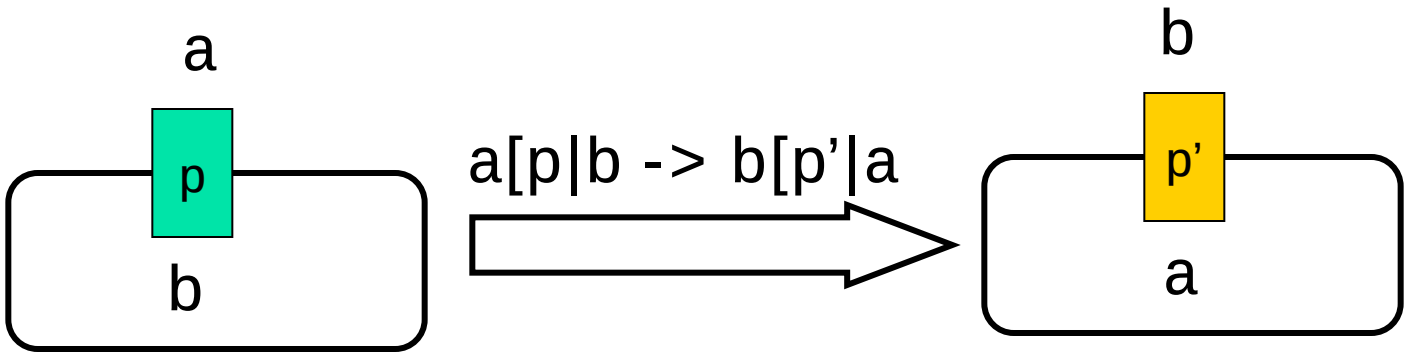


Examples

4res:

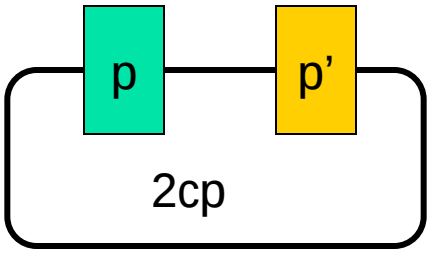


4cp:



Results

$$\mathrm{NOP}_1(pro_2, 2cp) = \mathrm{NRE}$$



2cp: uniport with change of protein

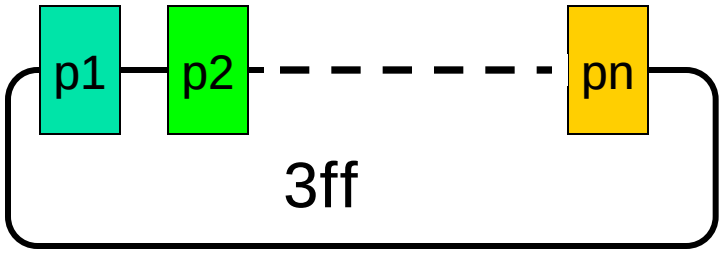
a[p|->[p’|a

[p|a->a[p’|

Results

$$\mathrm{NOP}_1(pro_*, 3\,ff) = \mathrm{NRE}$$

Improved later



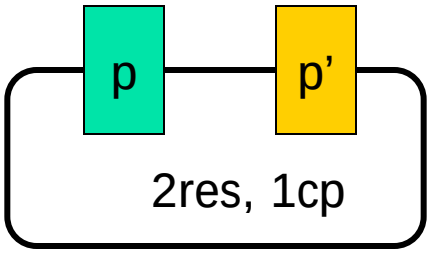
2cp: modify and move one object

a[p|->[p'|b

[p|a->b[p'|

Results

$$\mathrm{NOP}_1(pro_2, 2res, 1cp) = \mathrm{NRE}$$



2res: uniport
a[p|->[p|a
[p|a->a[p|

1cp:
[p|a->[p’|b
a[p|->b[p’|

Previous results in this area

- $NOP_1(pro_2, 2cpp) = NRE$
- $NOP_1(pro_*, 3ffp) = NRE$
- $NOP_1(pro_2, 2res, 4cpp) = NRE$
- $NOP_1(pro_2, 2res, 1cpp) = NRE$
- $NOP_1(pro_*, 1res, 2ffp) = NRE$

- In [Paun Popa 2006]

More previous results (ffp)

- $NOP_1(pro_7, 3ffp) = NRE$
- $NOP_1(pro_7, 2ffp, 4ffp) = NRE$
- $NOP_1(pro_{10}, 1res, 2ffp) = NRE$ **[Krishna 2006]**
- $NOP_1(pro_7, 1ffp, 2ffp) = NRE$
- $NOP_1(pro_9, 1ffp, 2res) = NRE$
- $NOP_1(pro_9, 2ffp, 3res) = NRE$
- $NOP_1(pro_8, 1ffp, 3res) = NRE$
- $NOP_1(pro_9, 4ffp, 3res) = NRE$
- $NOP_1(pro_8, 2ffp, 5res) = NRE$

Description of proof technique

- In [Paun Popa 2006] we used the proteins to control the simulation of each type of rule and usually as a Program Counter in the register machine
- In [Krishna 2006] the novel idea was to simulate with each protein a specific rule type associated with a specific register: all Sub(r1,XXX,YYY) use same protein

New results

- OLD: $NOP_1(pro_9, 4ffp,3res)=NRE$
- OLDISH: $NOP_1(pro_8, 4ffp,3res)=NRE$

- NEW, time: $NTOP_1(pro_7, 4ffp,3res)=NRE$
- NEW: $NOP_1(pro_7, 4ffp,3res)=NRE$

New results (2)

- Old: $NOP_1(pro_8, 2ffp,5res)=NRE$
- Oldish: $NOP_1(pro_7, 2ffp,5res)=NRE$

- New, time: $NTOP_1(pro_3, 2ffp,5res)=NRE$
- New: $NOP_1(pro_4, 2ffp,5res)=NRE$

- limited parallelism
- P systems with proteins on membranes enforce ≤ n-Parallelism where n is the number of proteins
- Some of the results require unbounded number of proteins, thus normal parallelism

1 UC: Spiking Systems

SNP

Represented as directed graph.

Neurons: nodes. Synapses: directed edges.

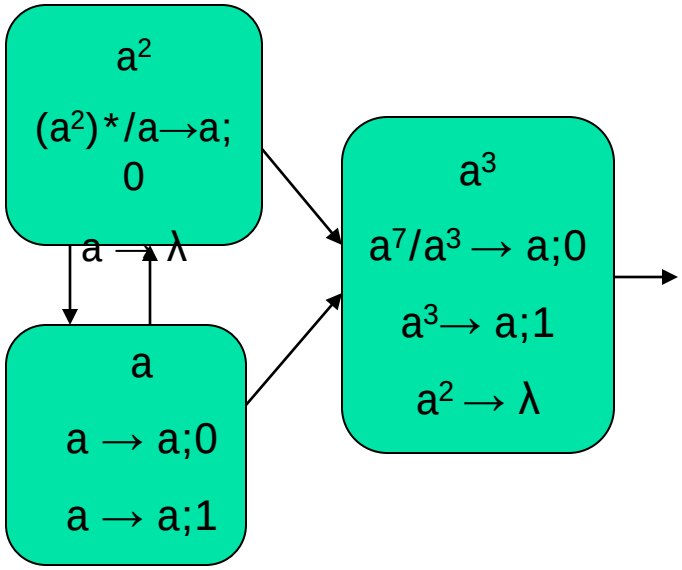
Only one symbol a used (to represent spike).

Initial configuration: spikes distributed in the neurons.

Number of spikes n in neuron is represented by string a^n.

Spikes are created and sent along all outgoing synapses from a neuron when that neuron ‘fires’.

- Maximal parallelism:
  - At each step, all fireable neurons must fire.
  - Each fireable neuron fires using one of the rules in the neuron, chosen nondeterministically.



- Firing with a delay: $E/a^j \rightarrow a;t$
  - Neuron is ‘closed’ during the time delay, t
  - A closed neuron is inactive (does not fire & loses any spikes sent to it) during the delay.

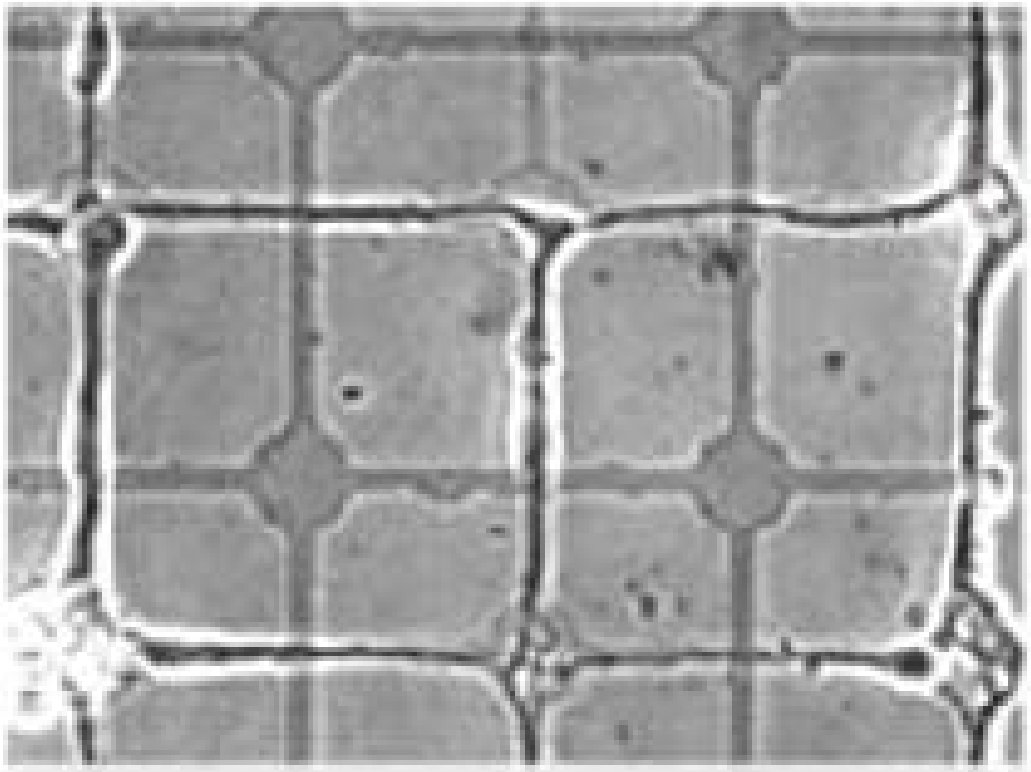
Motivation of study

- Previous maximal parallelism drawbacks
- The spikes are transmitted much faster than any signalling pathway
- More “spikes” stored=>more probable to spike
- It was observed that the neurons which receive many spikes, tend to fire more often

Motivation 2

- Same assumptions in “integrate-and-fire” modeling work

- Assume instantly



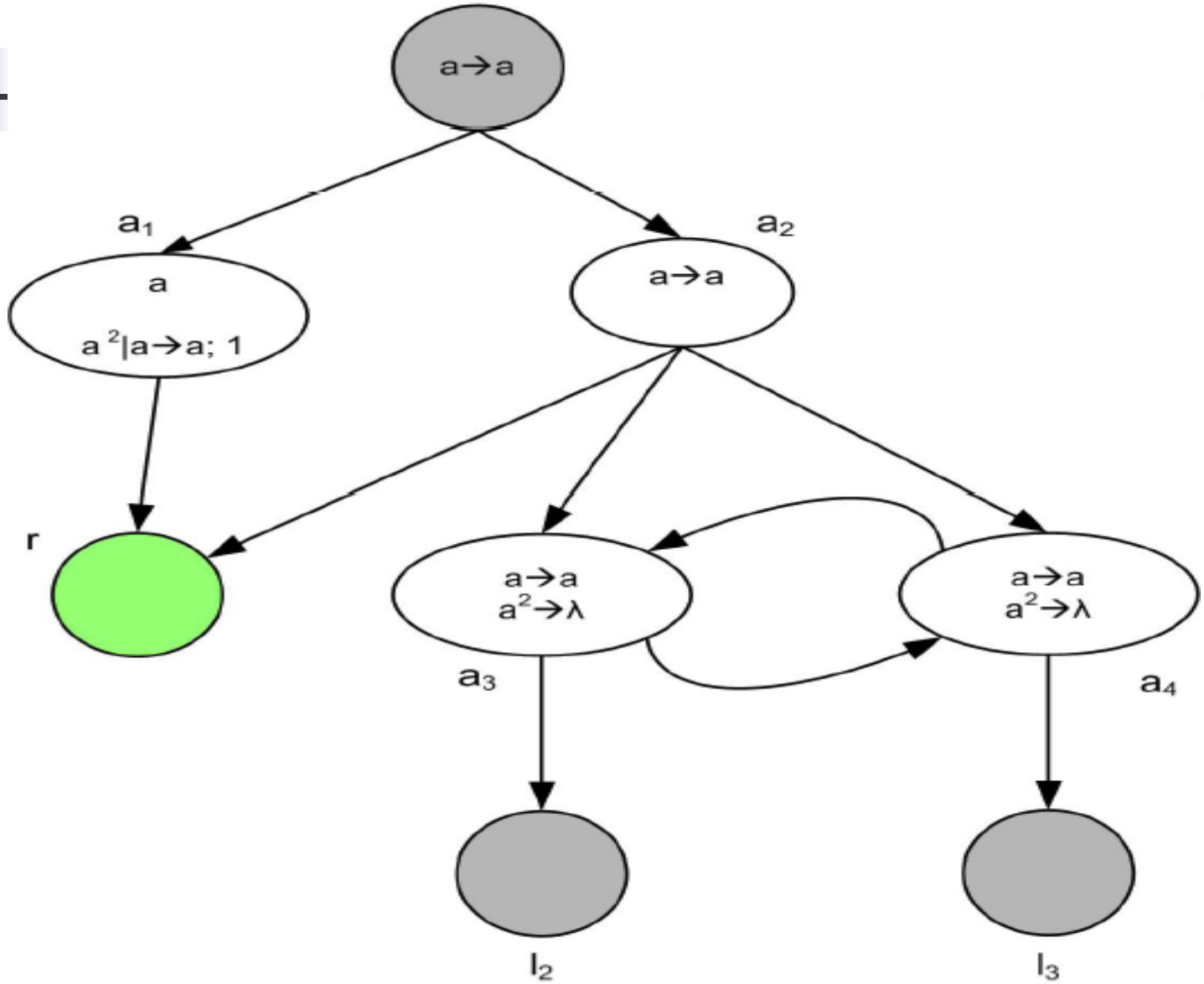
Extension

Max sequentiality:

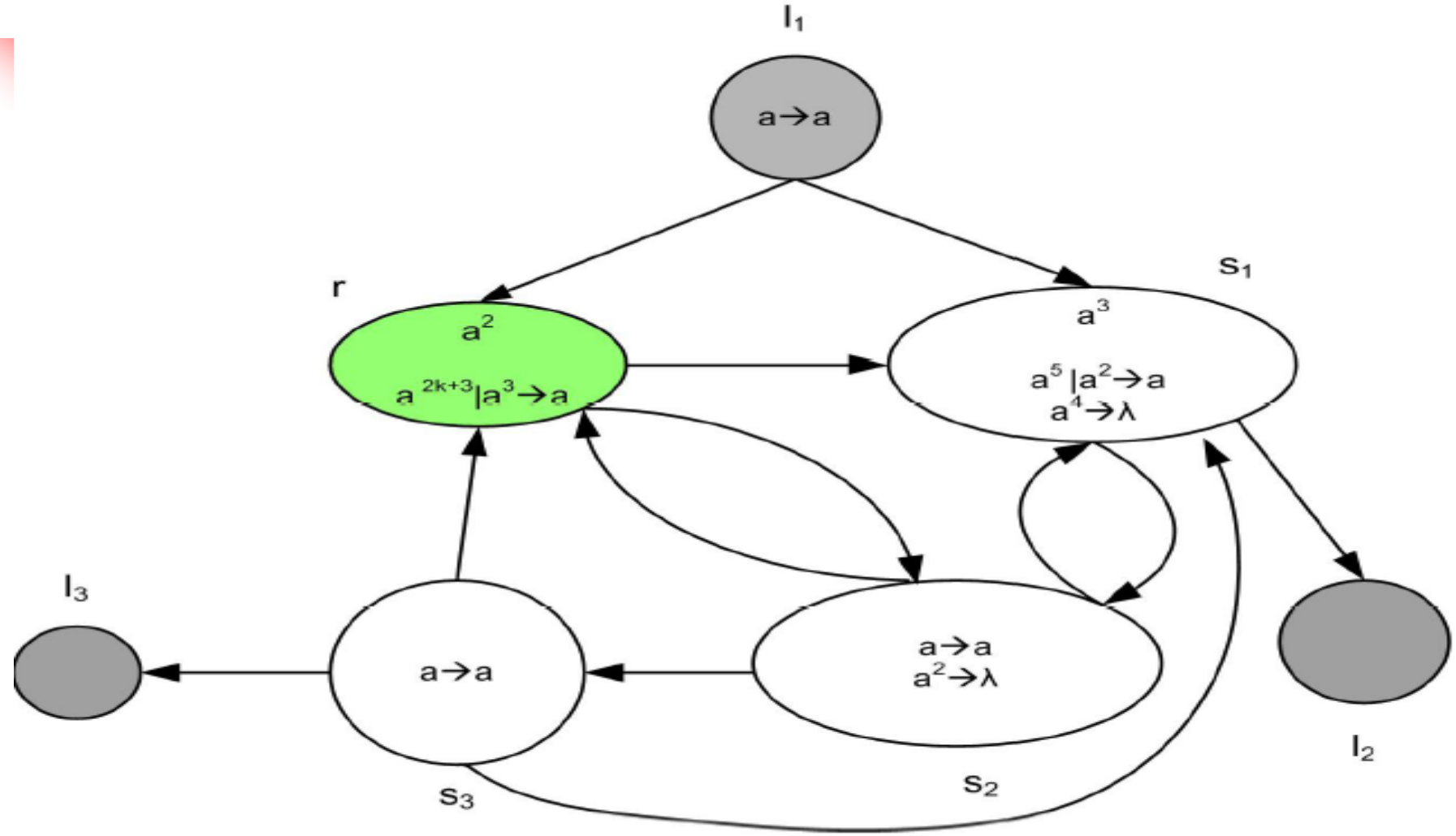
- From the system only the neuron with the maximum number of spikes (and active) will fire next.
- If more than one neuron are active at the same time and hold the maximum number of spikes, then we choose nondeterministically between them.

Max pseudo-sequentiality: allow to fire all the neurons that are active and hold the max number of spikes

Simulating the ADD module



Simulating the SUB module

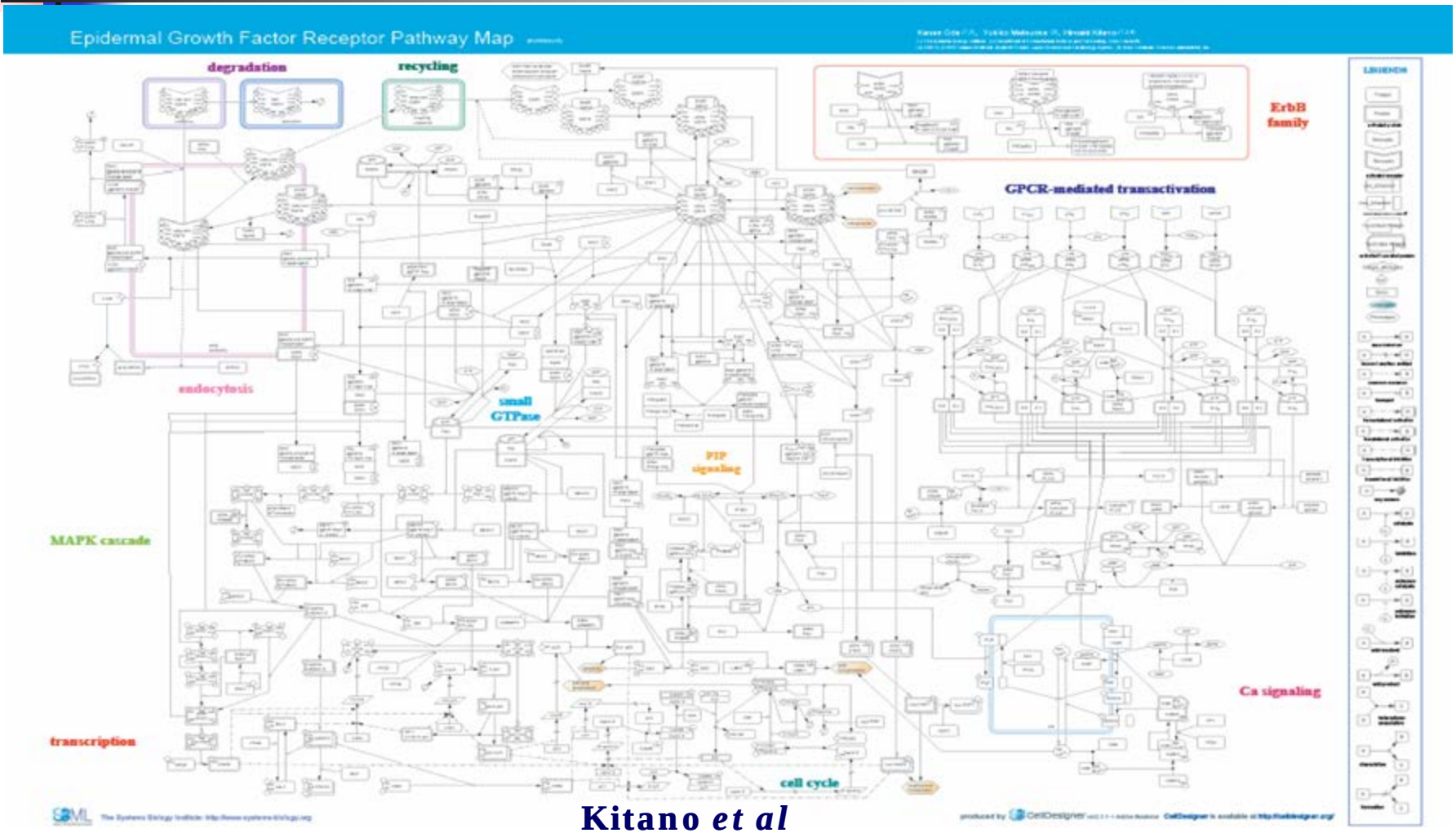


Results overview

- Max pseudosequentiality requires 104 neurons for universality
- Extended systems in max sequentiality: 90 (now 70) neurons needed
- Max strong sequentiality: not yet bounded the number of neurons for non-extended neurons
- Min sequentiality: a bit more complicated in the SUB module

2 Systems Biology (2 SB)

Complexity of Signal Transduction Pathways



Kitano *et al*

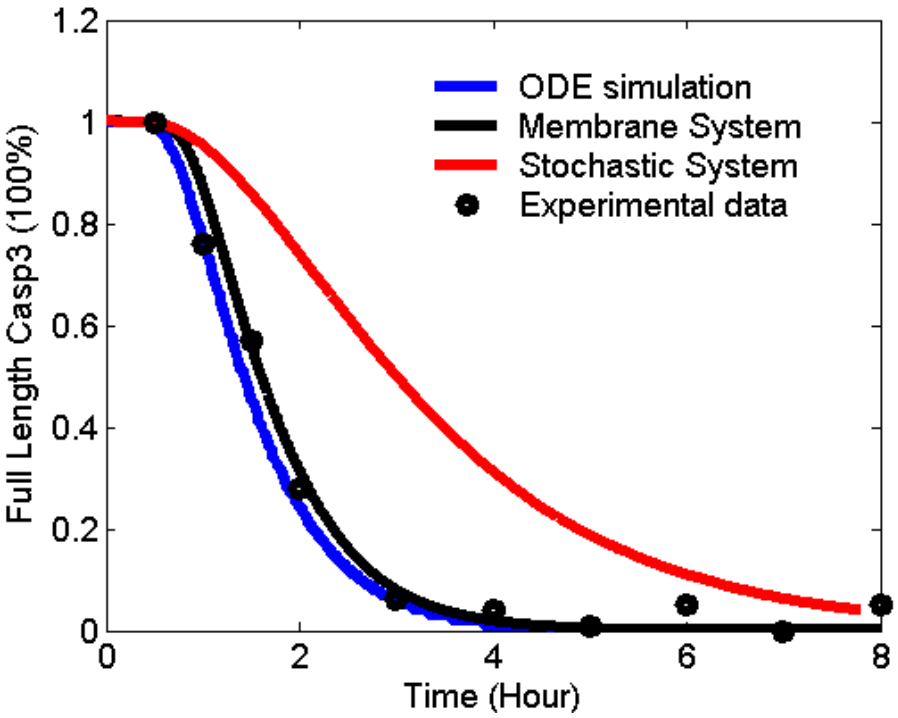
Motivation 2

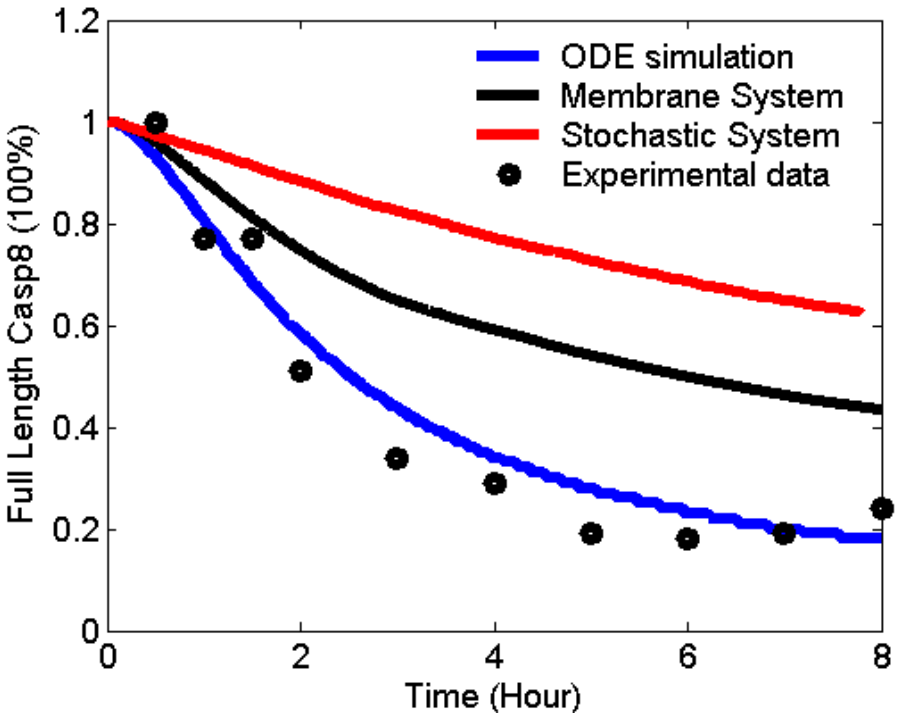
- ODE: hard to update, gives average behavior, assumes large numbers of molecules
  - Not always good simulation results
- Gillespie: slow
- membrane systems: easy to update, fast, discrete (different than ODE)

Alternative simulation method

- ODE assumes large numbers of copies of each type simulated
- membrane systems “process” each reaction/molecule individually (discrete system)
- Signaling pathways: small numbers of molecules, thus we believe it is better to use membrane systems

Comparison of the simulation methods





Preliminary results

- Simulation very close to biological observations
- Several orders of magnitude faster than Gillespie (3.5min vs. 6 hours)
- Extensible (easy to add new reactions)
- JAVA implementation accepts SBML input

Improvements

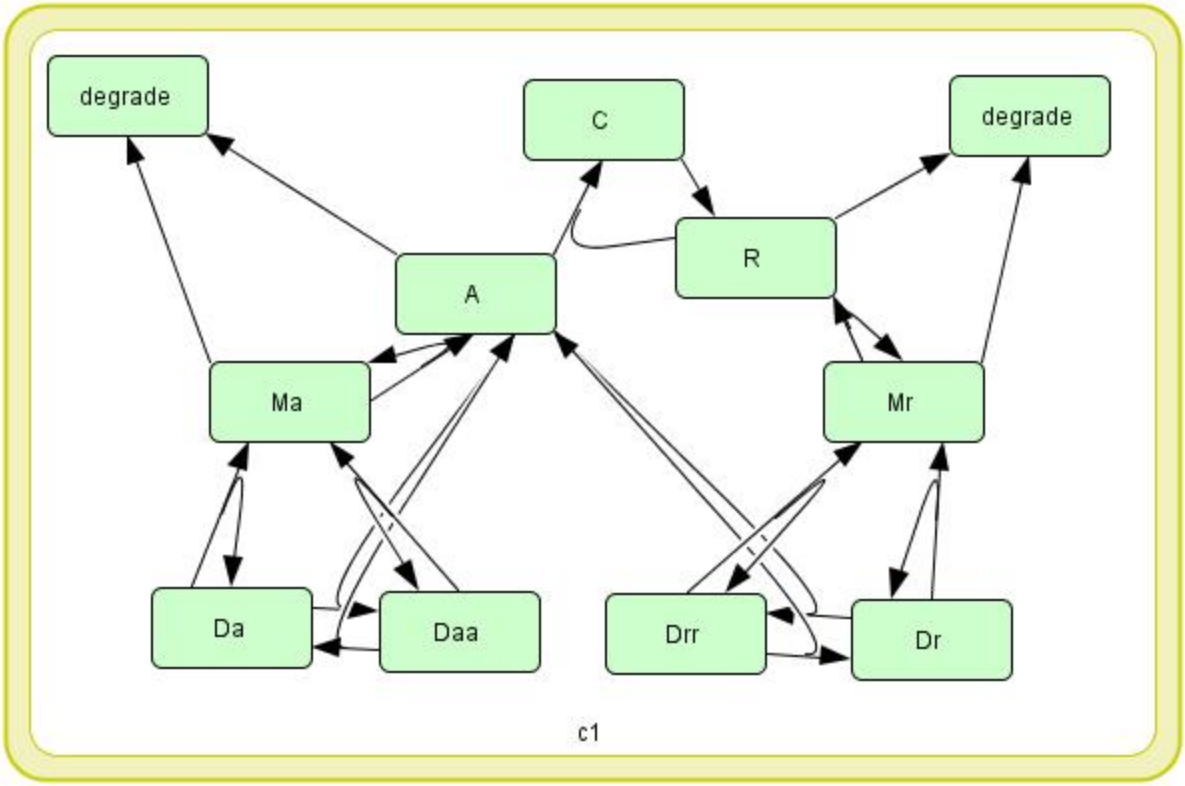
- Nondeterministic behavior of the system
- Implementation of a heap rather than sorting of reaction times at each step (reduces the time complexity of one step in the simulation from O(n log n) to O(log n) where n is the number of rules simulated)

Circadian Rhythm Model

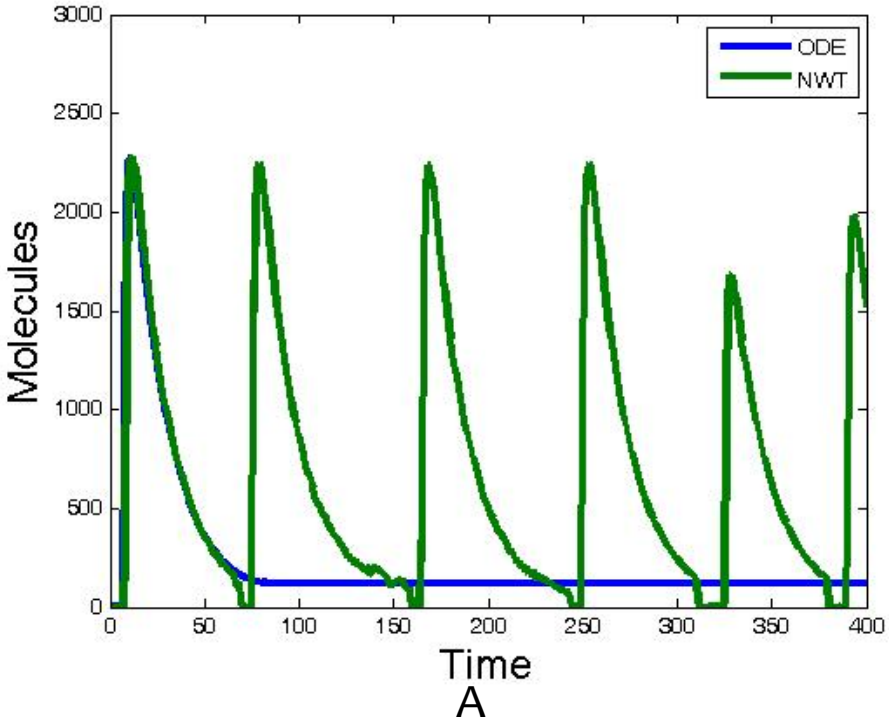
- To illustrate the effectiveness of our technique on an existing model, we consider the Circadian Rhythm model described in [Vilar 02]

- This model was designed to show intrinsic biochemical noise can induce oscillations

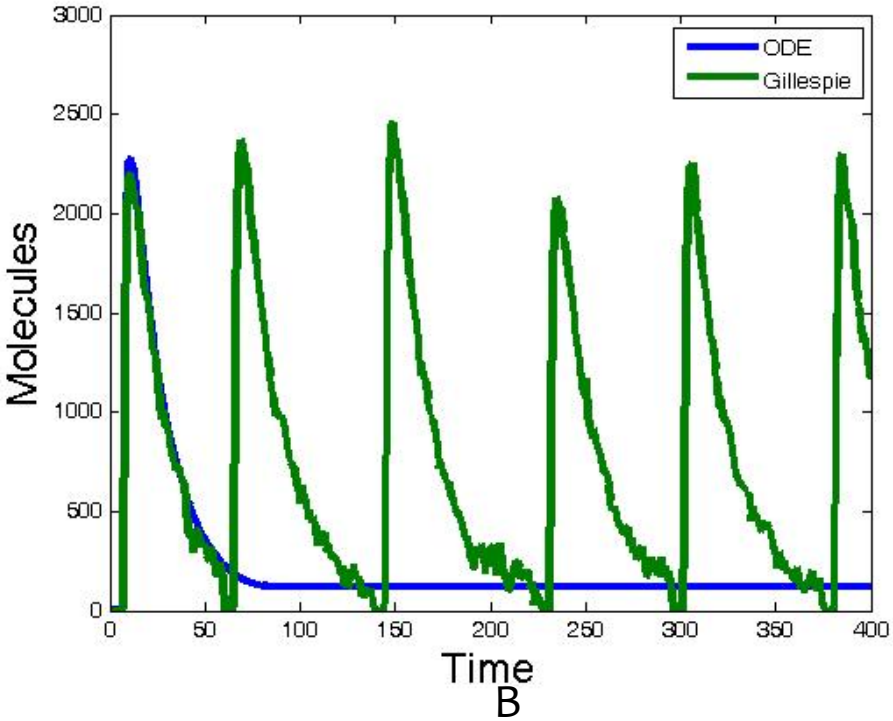
Circadian Rhythm Model



Circadian Rhythm Results

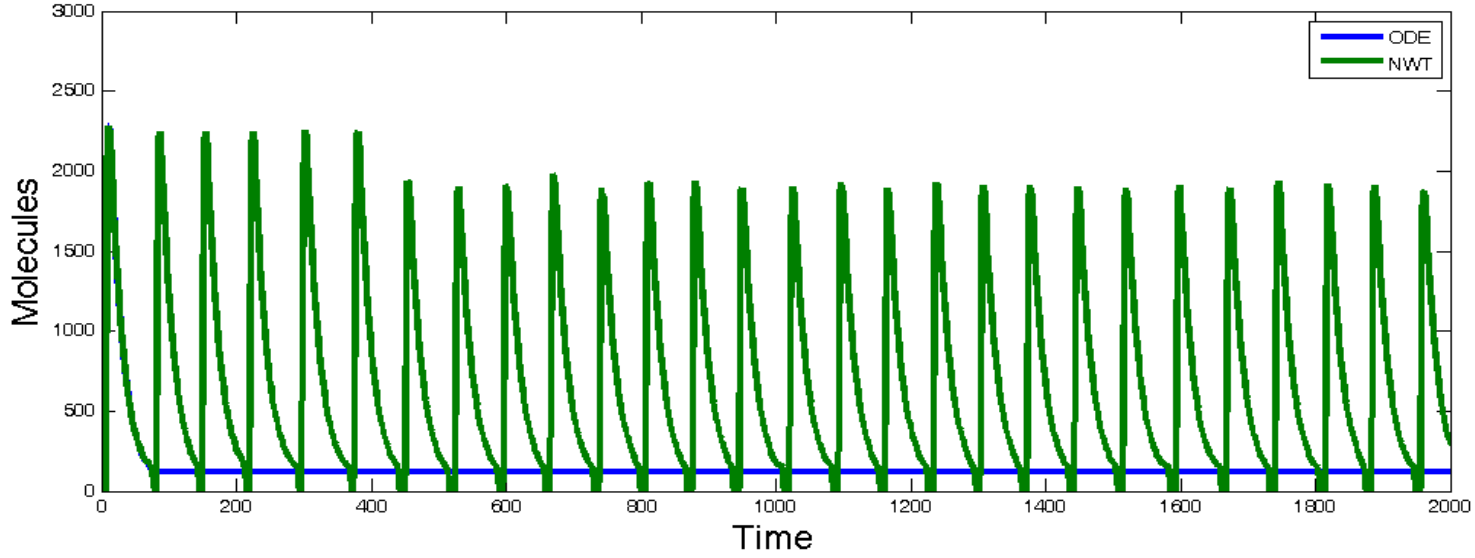


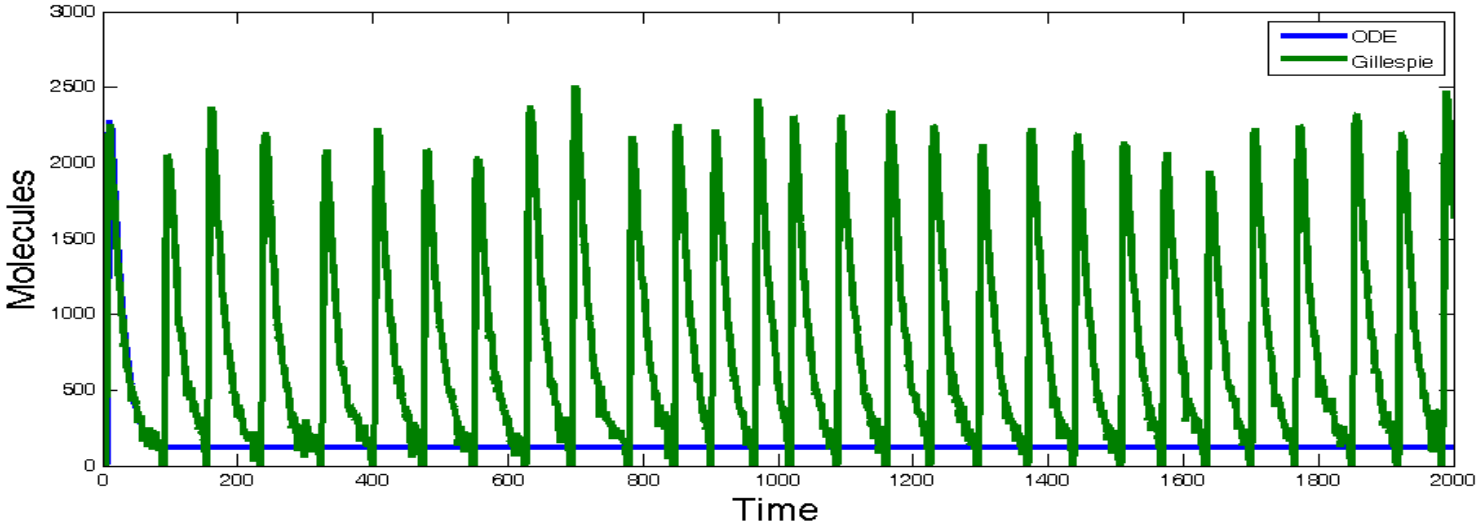
A



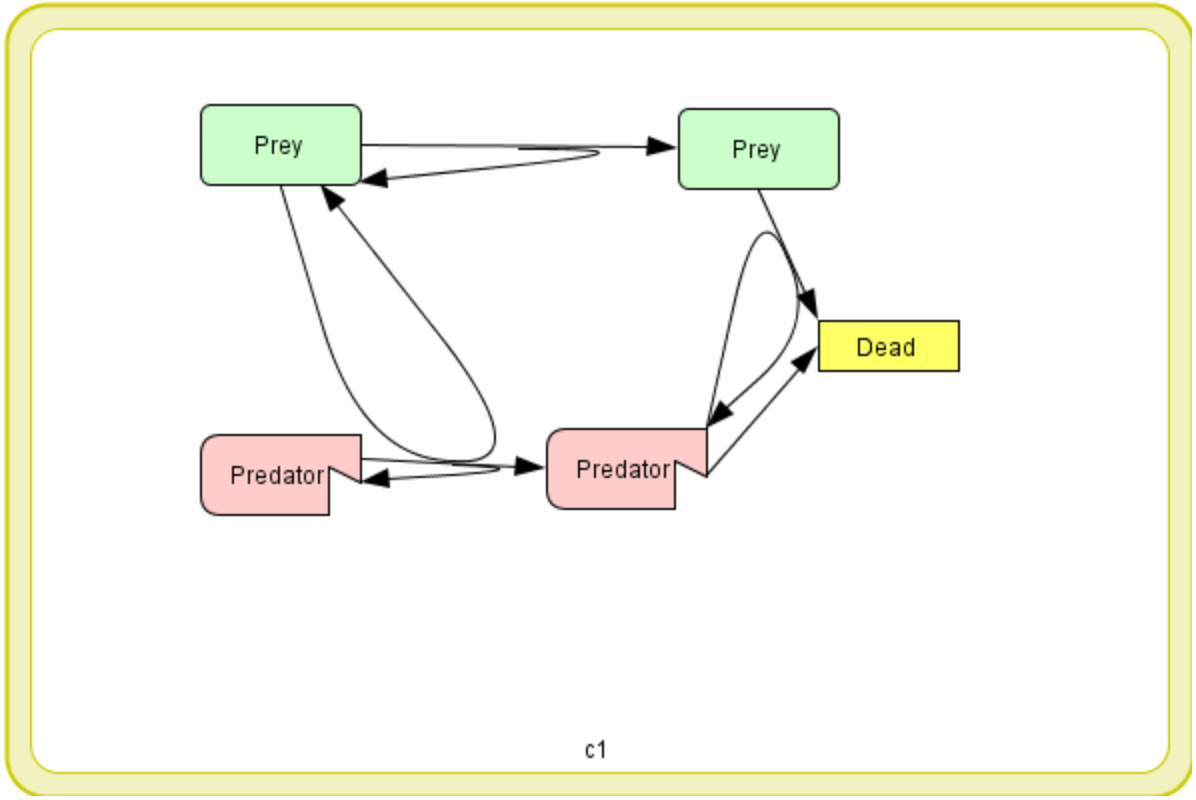
B

Circadian Rhythm Results (NWT)

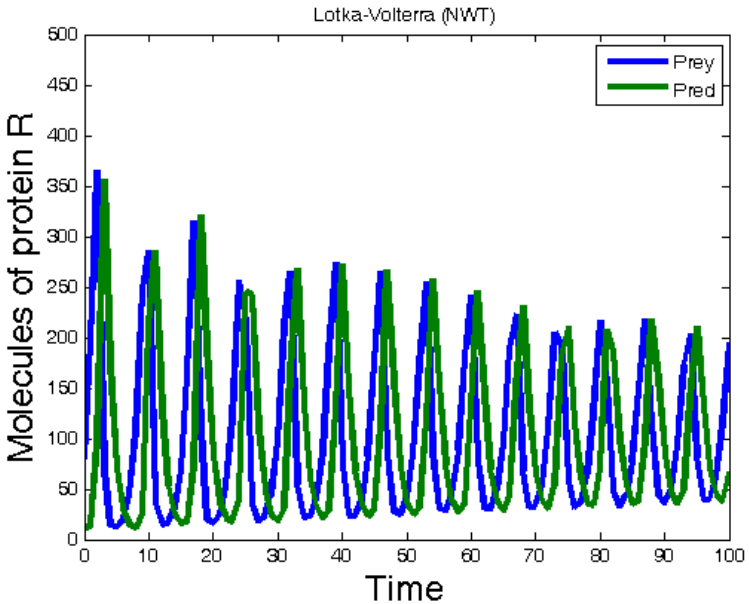


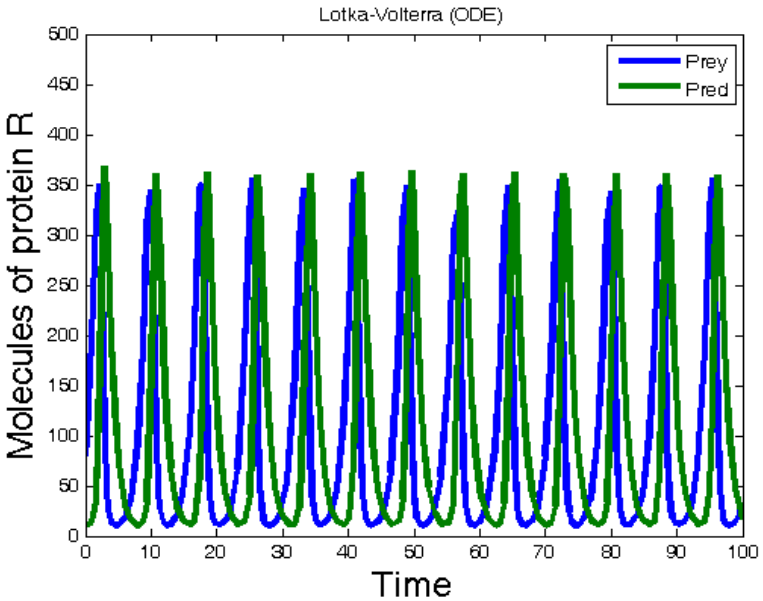


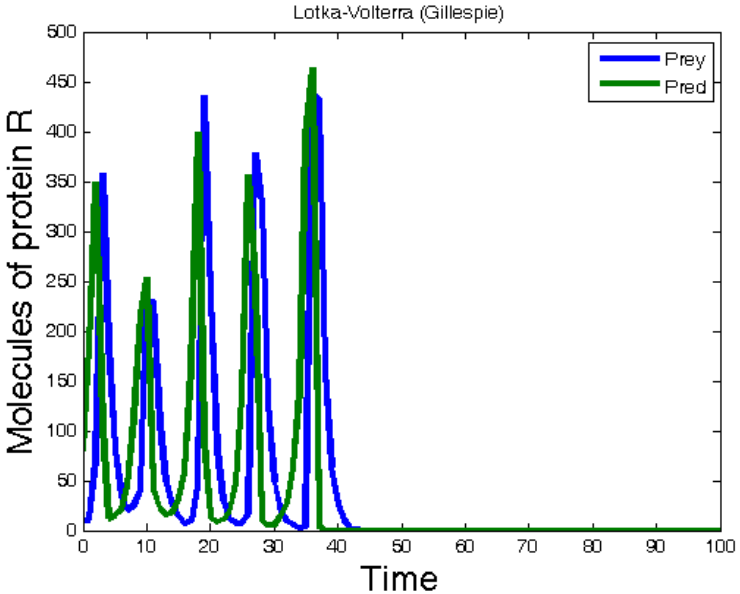
Lotka-Voltera Model



Lotka-Volterra Model







Simulation of HIV influence on FAS apoptosis

- The previous model for apoptosis extended to include proteins from the HIV
- Most studied HIV-1 (99% of infections)
- Nov 21, 2006 WHO: HIV is pandemic
- 1% of the world population infected
- 7.3% of infected people died in 2006

Apoptosis important in HIV

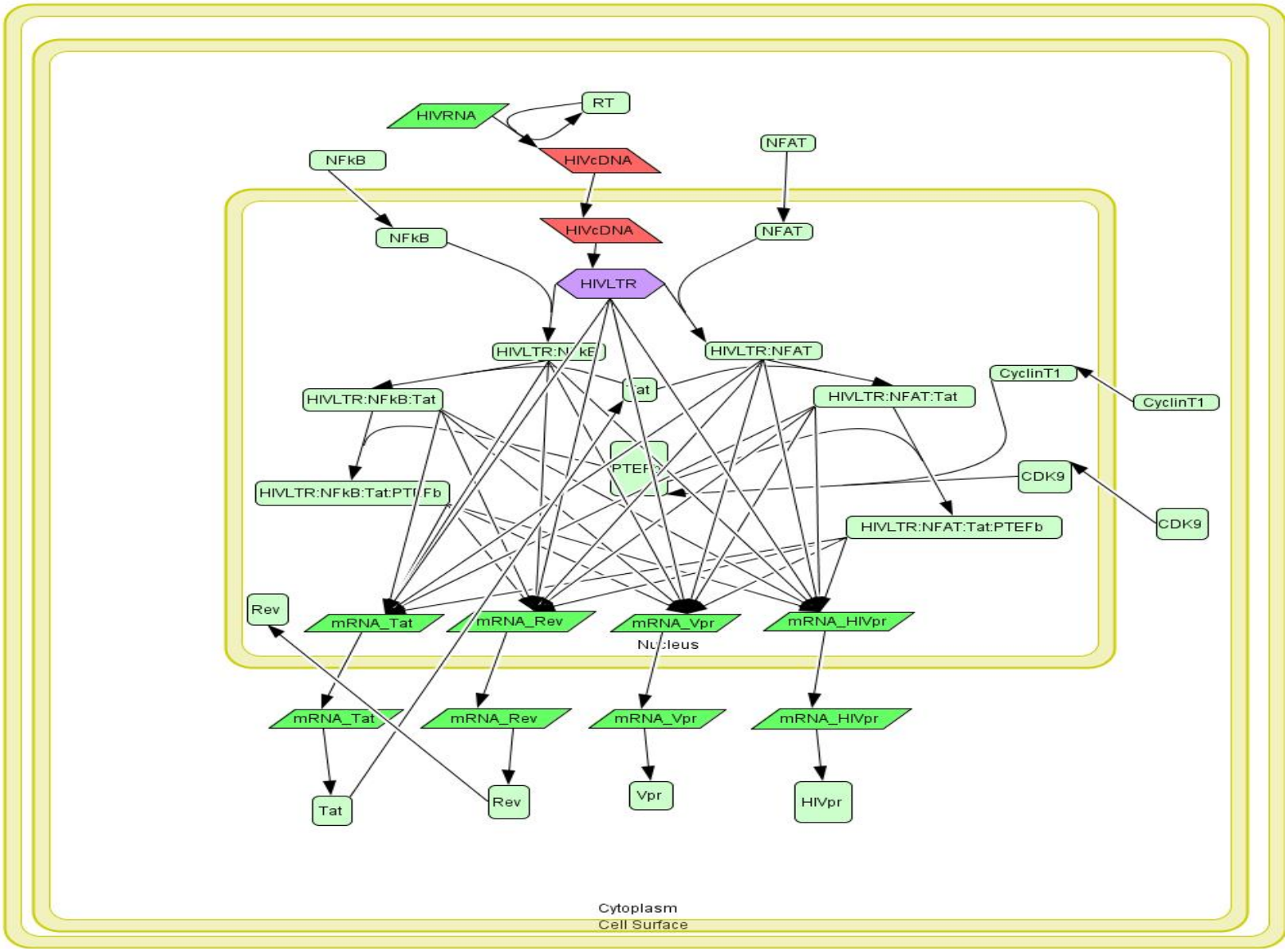
- Infects immune cells
- Initial infection through the R5 strand (co-receptor CCR5)
- After immune system is weakened (by apoptosis) X4 variant of the virus is more predominant (CXCR4 co-receptor)
- X4 emerges through mutations from R5

HIV latency

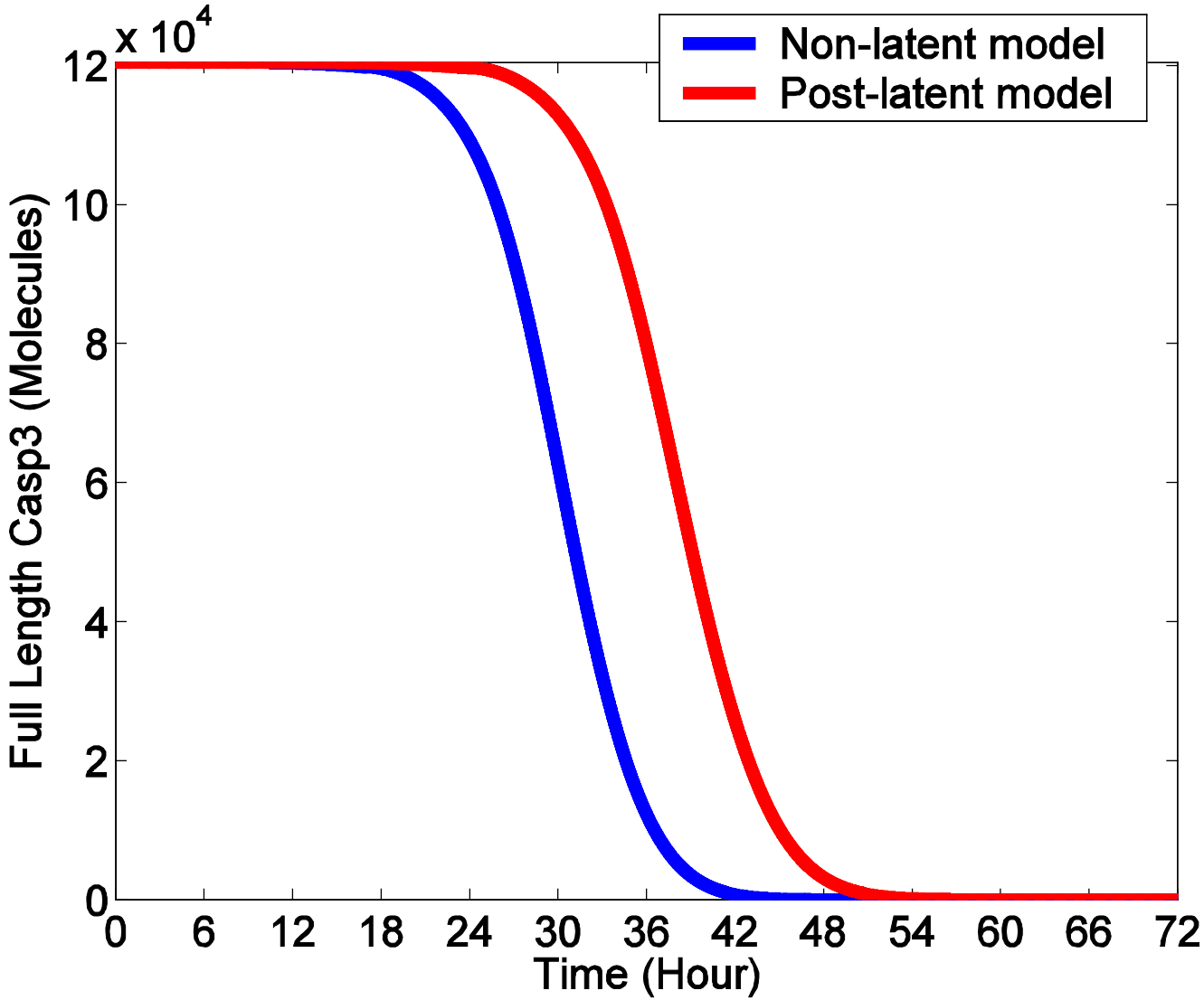
- Infected CD4+ T cells can become memory cells
- the main obstacle against HIV cure

- We model the apoptosis in the infected T cells: both active and dormant

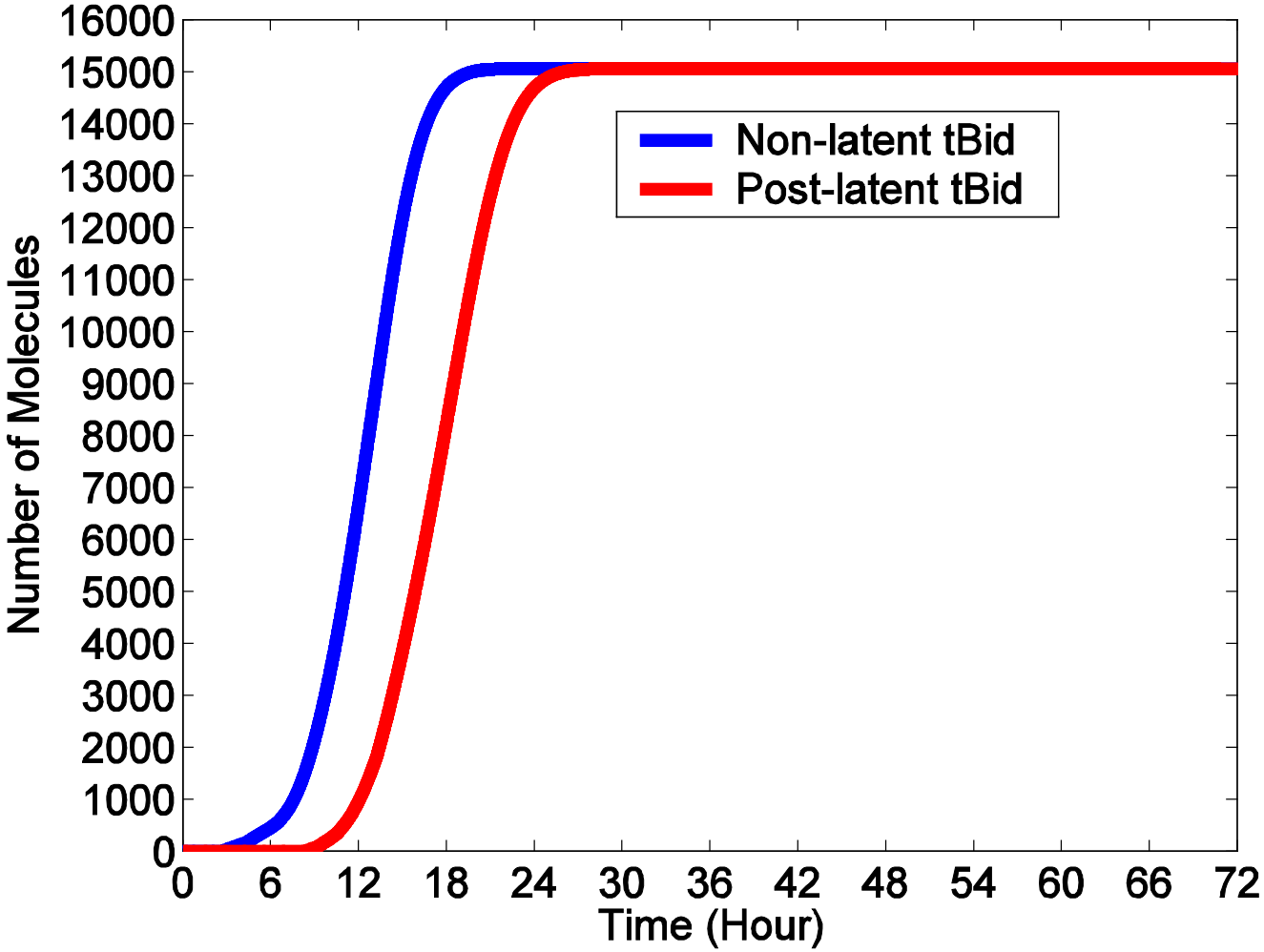
HIV Proteins



Simulation of apoptosis in infected cells, both latent and non-latent

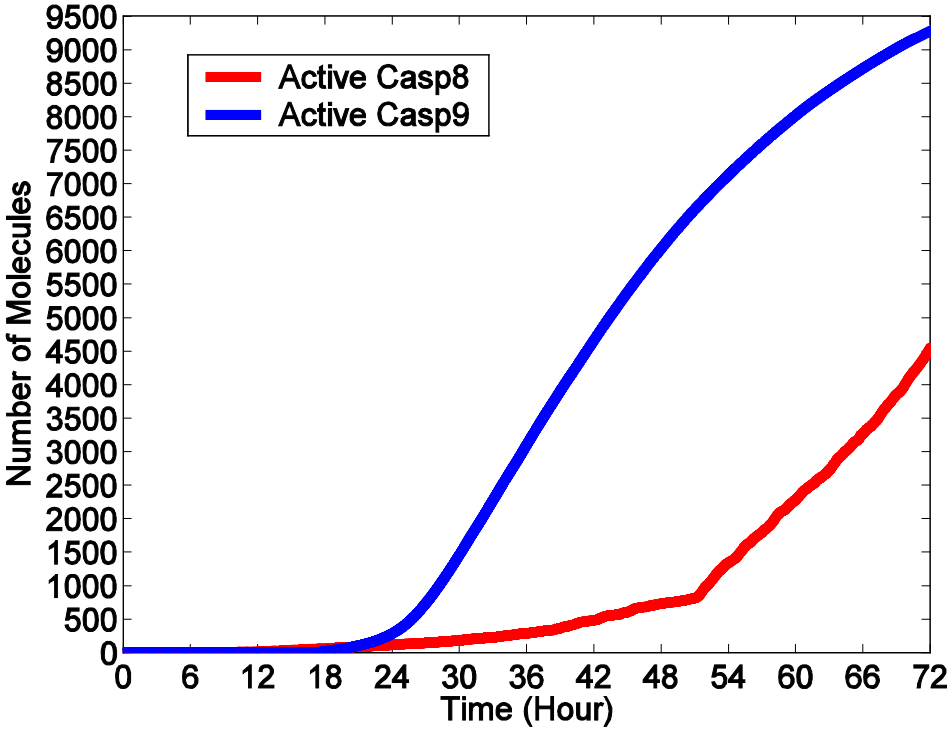


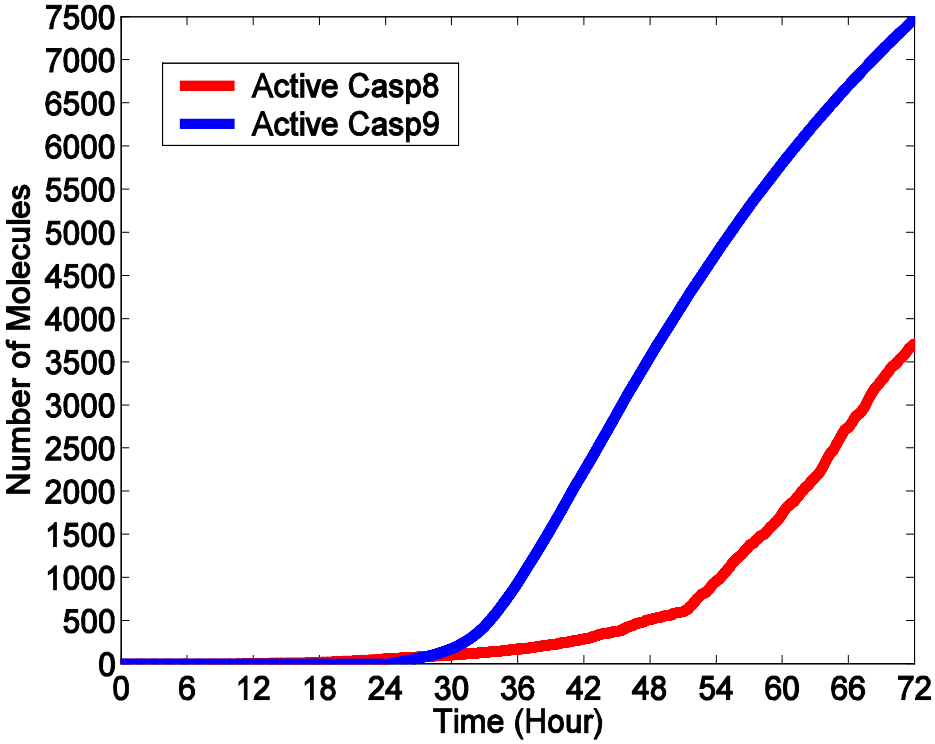
tBid signals the induction of the type II pathway



- non-latent

post-latent





Meaning

- Administer the cocktail of drugs
- Re-activate the memory T cells
- Keep patient alive for 42 hours
  - Cytotoxicity
- HIV virus would not be in the T cells

Simulation of latent cells

- These are the first results reported about the length of life of a latently infected cell that is re-activated
- Due to the scarcity of the experiments on latent cells
- Re-activated cells live about 6 hours less than the normally infected cells (42 vs 48 hours)

Future work for SB area

- Implementation / simulation of cells

- Stochastic approach for the STP simulations
- improvement of current results
- other (better) models

3 FA: Cover Automata

- DFA with a counter, for finite languages
- Variation of Hopcroft's algorithm exists
- Still O(n log n)
- We have still determinsm
- We lose the uniquess of the minimal machine
- In real life around 7% improvement

3 FA: Hopcroft's algorithm

- Hopcroft's algorithm is the best known algorithm for minimization of (general) DFA
- An upper bound on the runtime of the algorithm was proven in the original paper by Hopcroft, but no lower bound was given since

Hopcroft's algorithm

- Described in 1971 in 7 pages
- Has time complexity of O (n log n) and space complexity of O(n)
- Finds a partition according to the equivalance relation on states

Hopcroft's algorithm

- Starts with the coarsest partiton on states (F, Q-F) and refines this partition according to the "splitting" of states in the partition
- This splitting of states proceeds somehow in a backward manner: two states from the same partition that go with the same letter in different partitions are split
- Continue this until no more splitting is possible

Hopcroft's algorithm

```
P={F, Q-F}
For all a in ∑ add (min(F, Q-F),a) to S
While S≠ø do
  get (C,a) from S
  for each B in P that is split by (C,a) do
    replace B in P by both B' and B''
    for all b in∑ do
    if (B,b) in S then replace it by (B',b) and (B'', b)
                  else add (min(B',B''),b) to S
```

Implementation choices

- There are three points of flexibility for implementation of the algorithm
- In line 2: the strategy for S
- In line 9: if |B'|=|B''| which one is added to S
- And in line 8: according to the implementation of S how is B' and B'' replacing B

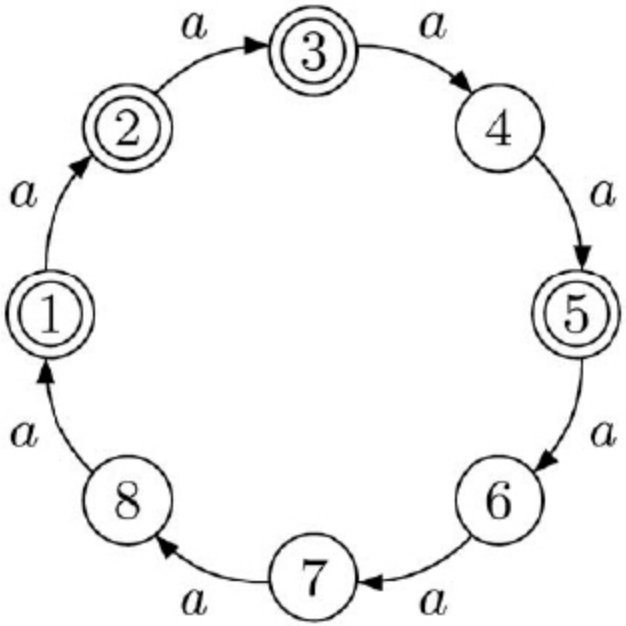
Results for DFA

- In the worst case scenario (queue implementation) for unary languages with 2^n states we have:
- Final states=non final states= 2^{n-1}
- Final states preceeding final states: 2^{n-2}
- etc.

- The worst possible case is reached by deBruijn words:
- Every possible word

of length 3 appears

exactly once



- Automaton for n=3

- Solutie similara pentru DFCA

Stack is better than queue

- The absolute worst case run-time complexity for the Hopcroft's minimization algorithm for DFCA for unary languages is reached when the splitter list S in the algorithm is following a FIFO strategy and only for automata having a structure induced by de Bruijn words of size n.
- In that setting the algorithm will pass through the queue S exactly $n2^{n-1}$ states for the input automaton of size 2^n. Thus for m states of the input automaton we have exactly $m/2 \log_2 m$ states passing through S.
- (linear for stack, O(n log n) for queue)

Other DFCA result (2005)

- Incremental construction of DFCA (save space and time)

Algorithm	States	Memory req.	Time/time with trie	l	#Σ
Körner	3905	70k	1.512s/1.961s	5	5
Incremnt.	18	1.8k	0.461s	5	5
Körner	19530	1.4M	40.52s/52.706s	6	5
Incremnt.	21	2.2k	3.196s	6	5
Körner	97655	7.0M	24min 49.26s/34min 6.944s	7	5
Incremnt.	24	2.7k	22.420s	7	5

Future work: cover automata

- Experiments on different languages and different implementation strategies
- Experiments on random languages
- Study the case of “random” changes of the strategy between LIFO and FIFO
- Union of two cover automata with different l-s

4 Plan for future (ideas)

- DFCA used for counting the number of nullomers in GenBank
- DFCA for compression of genomes
- Log-gain procedure developed by V. Manca verify the stability to noise
- RECRUIT good PhD students

Thank you !!!