

# Membrane Computing: Power, Efficiency, Applications

(Introduction and Some Recent Results)

Gheorghe Păun  
Romanian Academy, Bucharest,  
RGNC, Sevilla University, Spain  
[george.paun@imar.ro](mailto:george.paun@imar.ro), [gpaun@us.es](mailto:gpaun@us.es)

## Schedule:

- Membrane computing as a branch of natural computing
- A glimpse to the cell biology
- Basic ingredients of a P system
- Theoretical computer science prerequisites (if needed)
- Basic classes of cell-like P systems; Examples
- Computational power (Universality)
- Variants of P systems
- P systems with symport/antiport rules
- Tissue-like and neural-like P systems
- Spiking neural P systems
- Solving hard problems in a feasible time
- Applications in biology/medicine
- Applications in economics (numerical P systems)
- Recent developments, trends, closing discussion

## References:

- Gh. Păun, Computing with Membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143, and *Turku Center for Computer Science-TUCS Report No 208*, 1998 ([www.tucs.fi](http://www.tucs.fi))  
ISI: “fast breaking paper”, “emerging research front in CS” (2003)  
<http://esi-topics.com>
- Gh. Păun, *Membrane Computing. An Introduction*, Springer, 2002
- G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds., *Applications of Membrane Computing*, Springer, 2005
- Gh. Păun, G. Rozenberg, A guide to membrane computing, *Theoretical Computer Sci.*, 287, 1 (2002), 73–100
- Gh. Păun, Introduction to membrane computing, *Proc. Brainstorming Workshop on Uncertainty in Membrane Computing*, Palma de Mallorca, Nov. 2004, 17–65
- Website: <http://psystems.disco.unimib.it>  
(Yearly events: BWMC (February), WMC (summer), TAPS/WAPS (fall))

## SOFTWARE AND APPLICATIONS:

[http://www.dcs.shef.ac.uk/~marian/PSimulatorWeb/P\\_Systems\\_applications.htm](http://www.dcs.shef.ac.uk/~marian/PSimulatorWeb/P_Systems_applications.htm)

Verona (Vincenzo Manca: [vincenzo.manca@univr.it](mailto:vincenzo.manca@univr.it))

Sheffield (Marian Gheorghe: [M.Gheorghe@dcs.shef.ac.uk](mailto:M.Gheorghe@dcs.shef.ac.uk))

Sevilla (Mario Pérez-Jiménez: [marper@us.es](mailto:marper@us.es))

Milano (Giancarlo Mauri: [mauri@disco.unimib.it](mailto:mauri@disco.unimib.it))

Nottingham, Leiden, Vienna, Evry, Iași

Goal: abstracting computing models/ideas from the structure and functioning of living cells (and from their organization in tissues, organs, organisms)

hence not producing models for biologists (although, this is now a tendency)

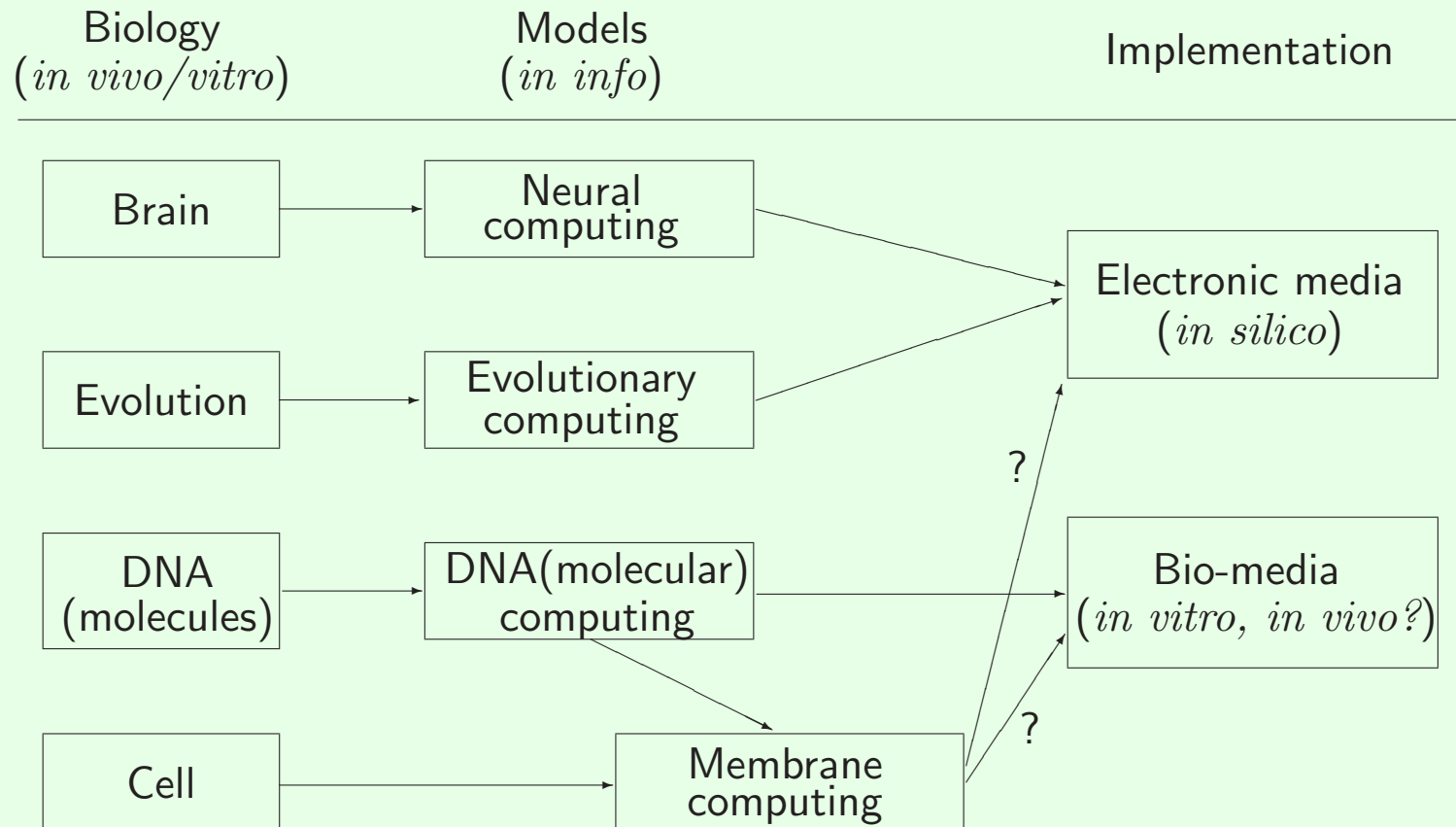
result:

- distributed, parallel computing model
- compartmentalization by means of membranes
- basic data structure: multisets (but also strings; recently, numerical variables)

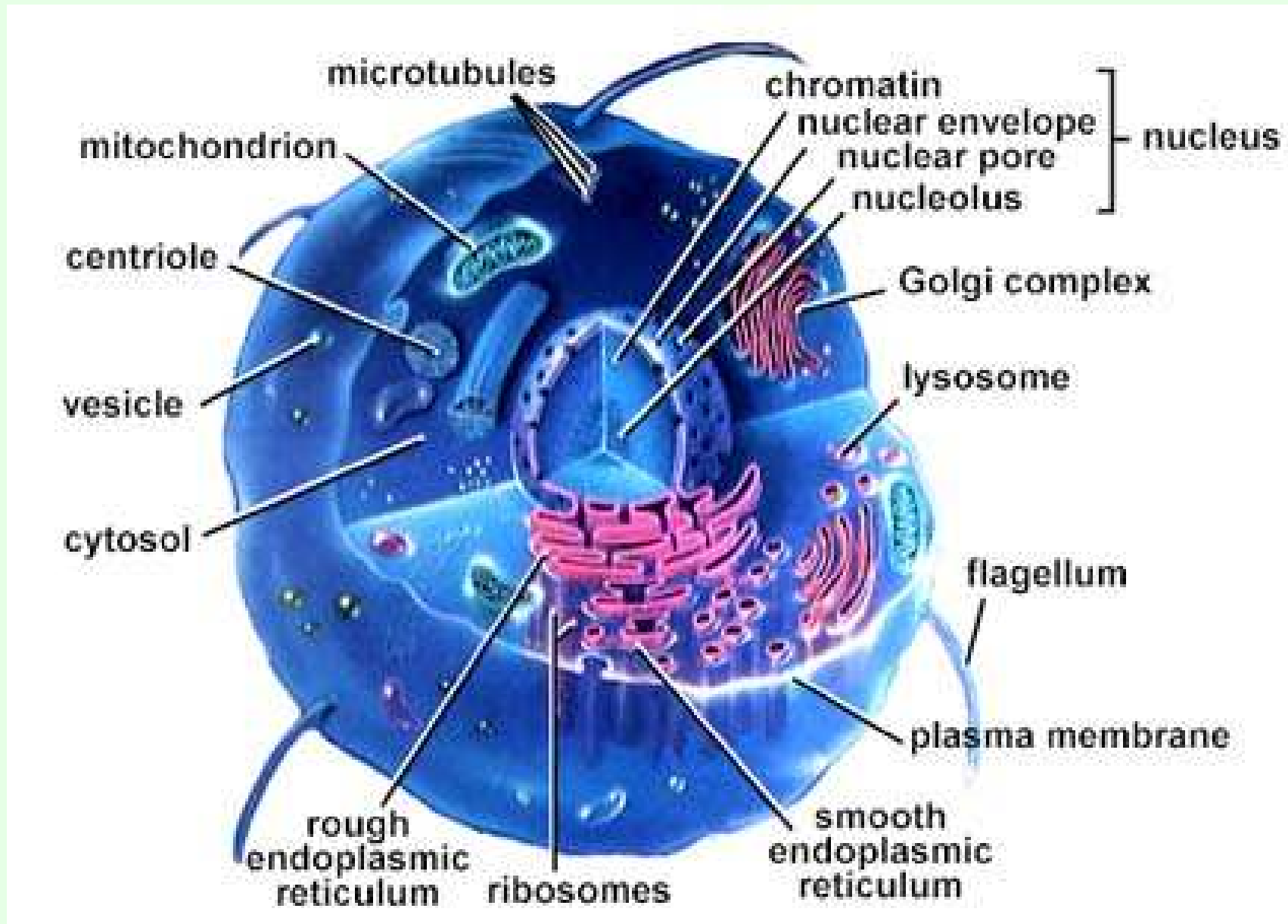
## WHY?

- the cell exists! (challenge for mathematics)
- biology needs new models (discrete, algorithmic; system biology, the whole cell modelling/simulating)
- computer science can learn (e.g., parallelism, coordination, data structure, architecture, operations, strategies)
- computing in vitro/in vivo (“the cell is the smallest computer”)
- distributed extension of molecular computing
- a posteriori: power, efficiency (“solving” NP-complete problems)
- a posteriori: applications in biology, computer graphics, linguistics, economics, etc.
- nice mathematical/computer science problems

## Membrane computing as a branch of natural computing



## A glimpse to the cell biology



## WHAT IS A CELL? (for a mathematician)

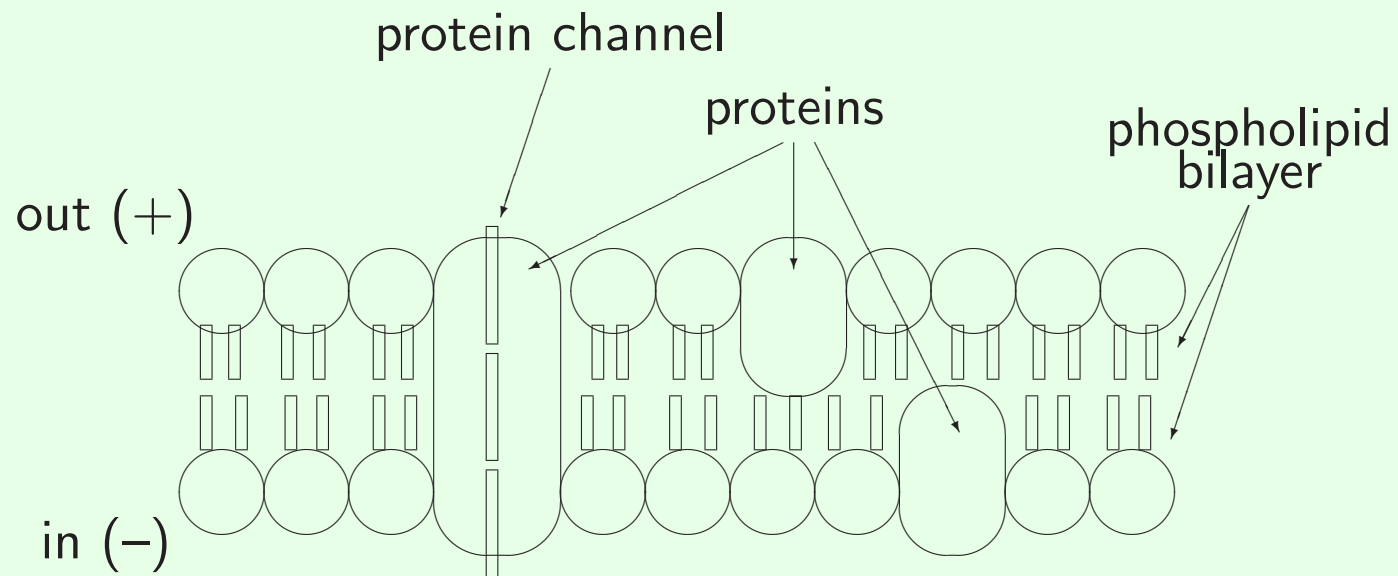
- membranes, separating “inside” from “outside” (hence protected compartments, “reactors”)
- chemicals in solution (hence multisets)
- biochemistry (hence parallelism, nondeterminism, decentralization)
- enzymatic activity/control
- selective passage of chemicals across membranes
- etc.

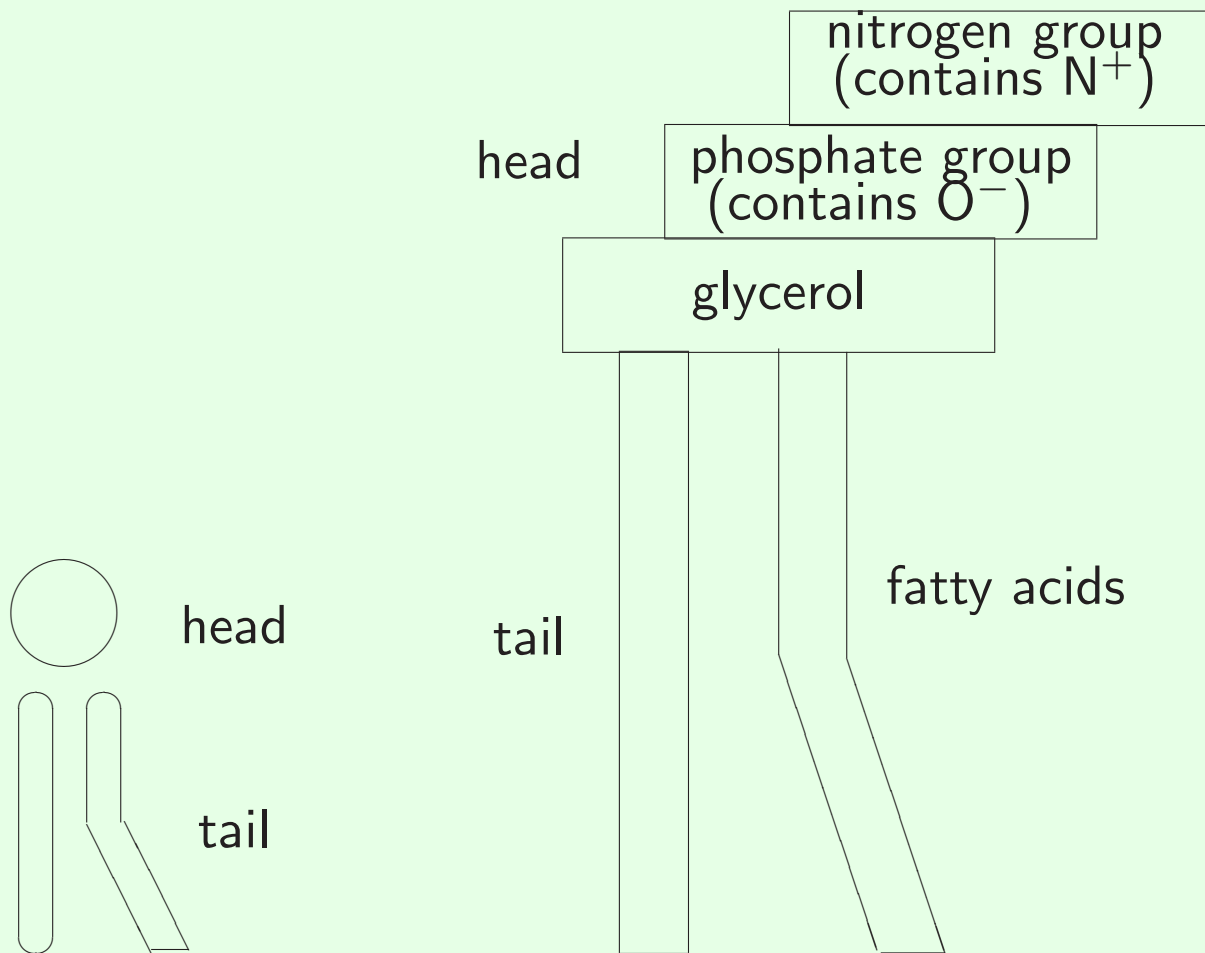
Importance of membranes for biology: . . .

MARCUS: *Life = DNA software + membrane hardware*

## THE STRUCTURE OF PLASMA MEMBRANE

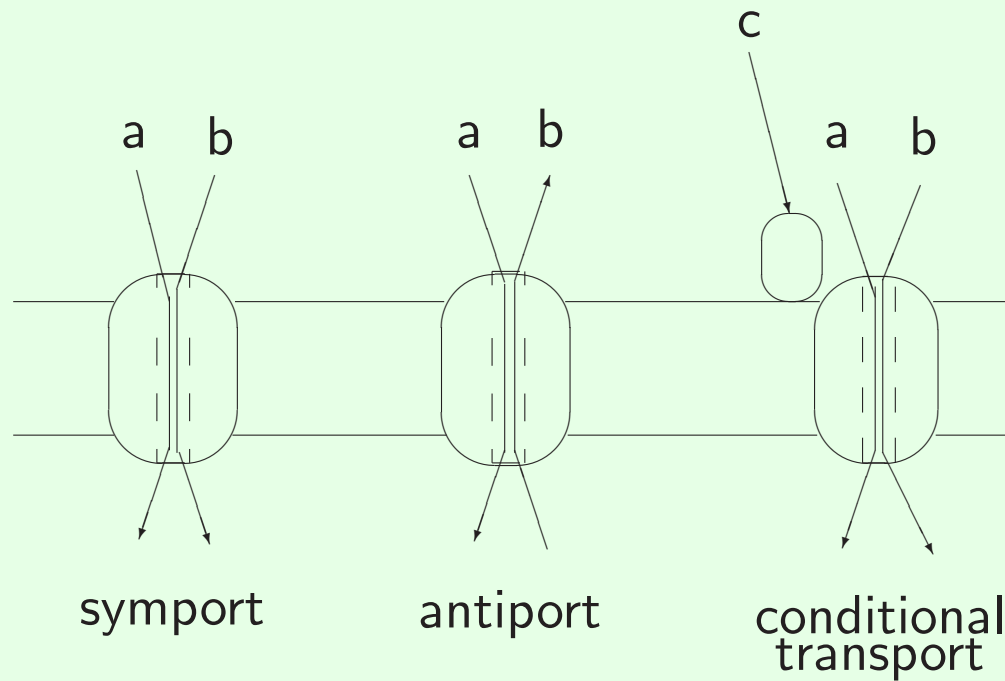
The *fluid-mosaic model*, S. Singer and G. Nicolson, 1972



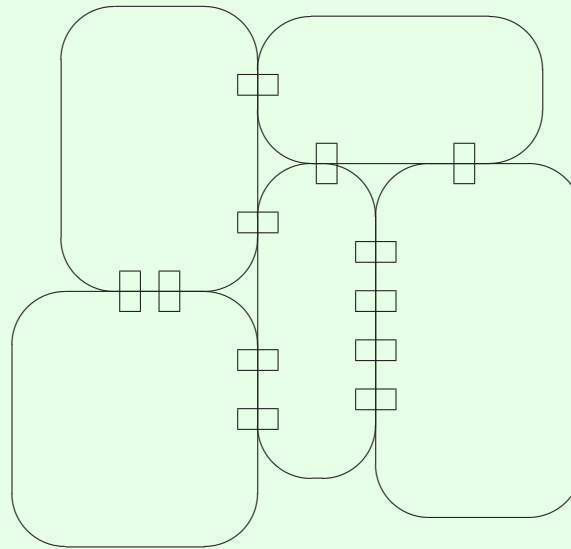


Trans-Membrane Transport  
passive (diffusion – concentration based)  
active (protein channels)  
vesicle-mediated

Important case of active transport: coupled transport  
symport  
antiport

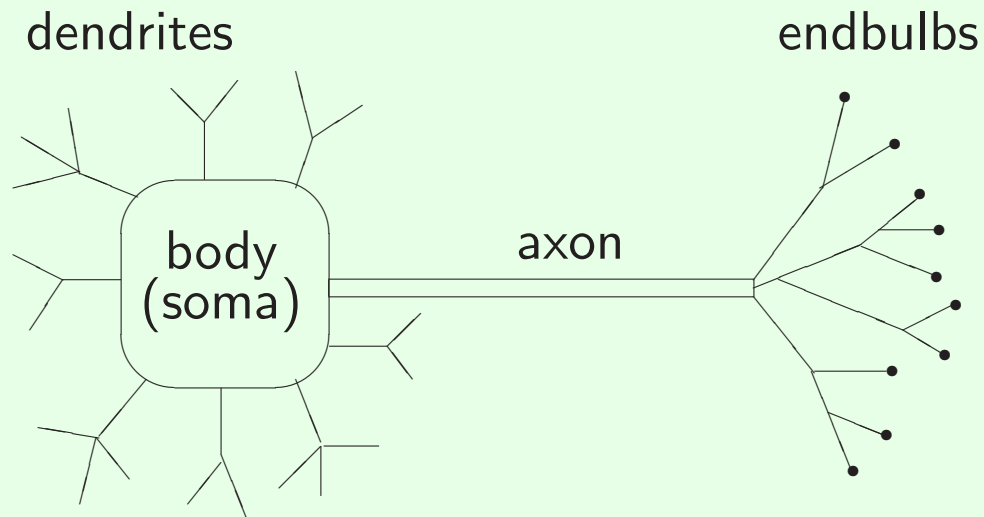


## Inter-cellular communication



( $\implies$  tissue-like membrane systems)

## THE NEURON

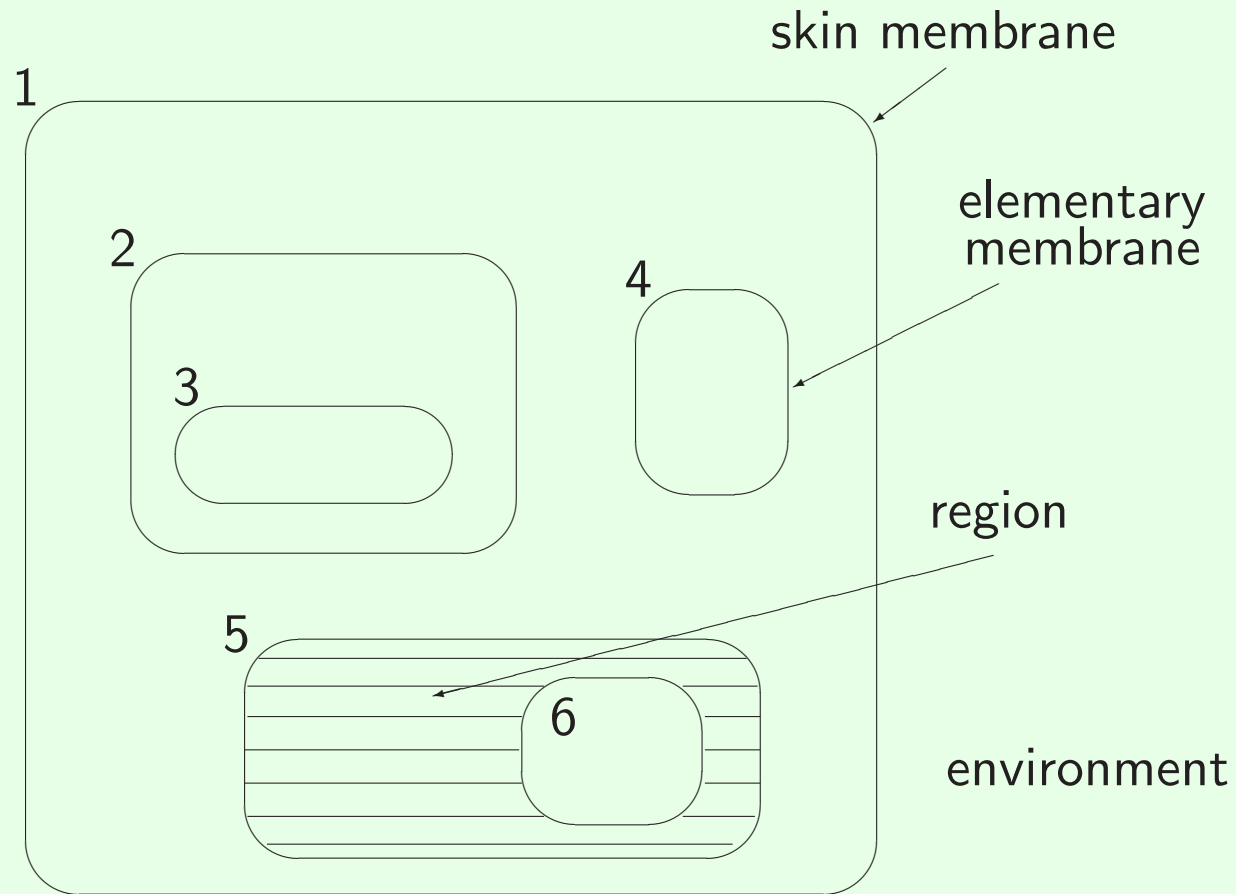


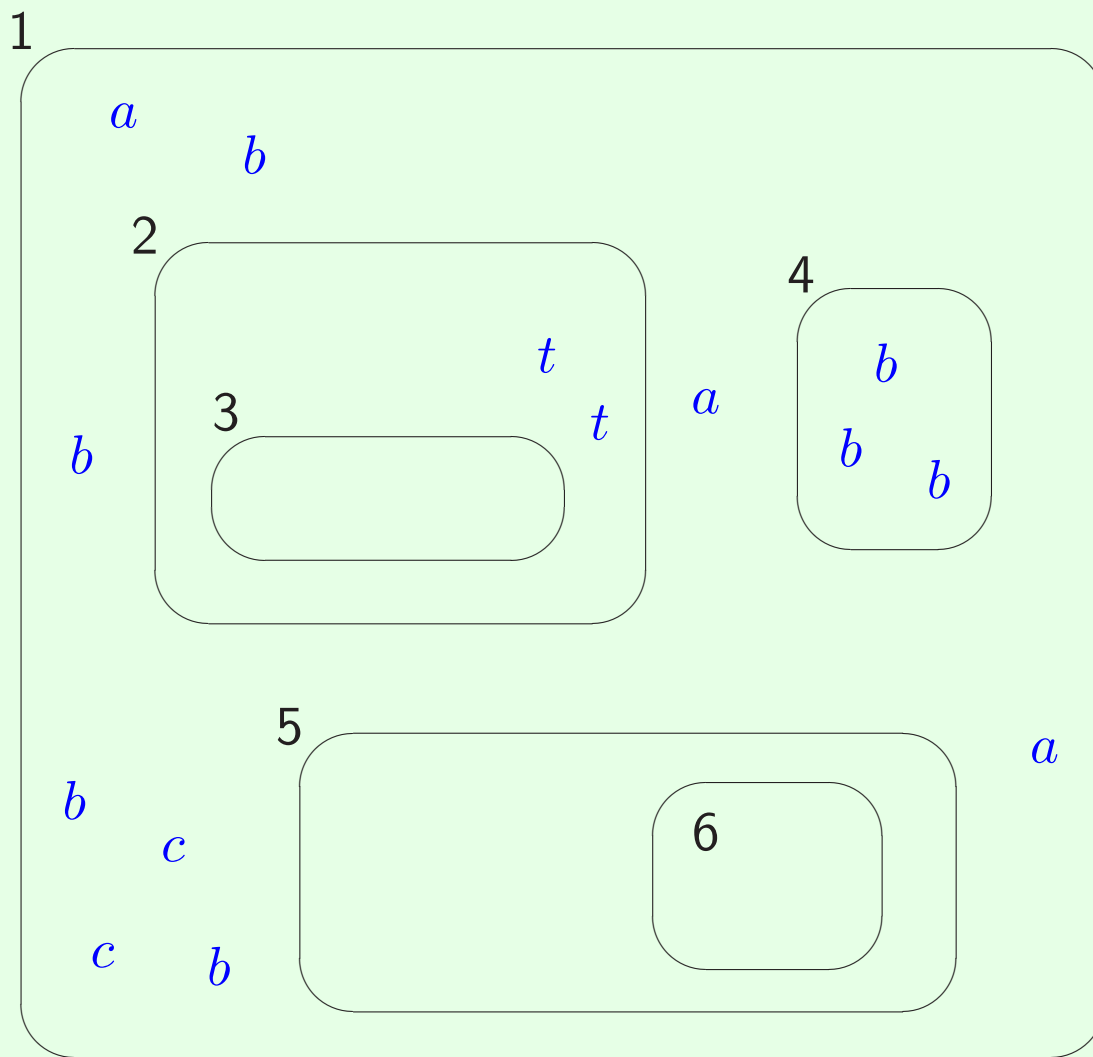
( $\implies$  neural-like membrane systems)

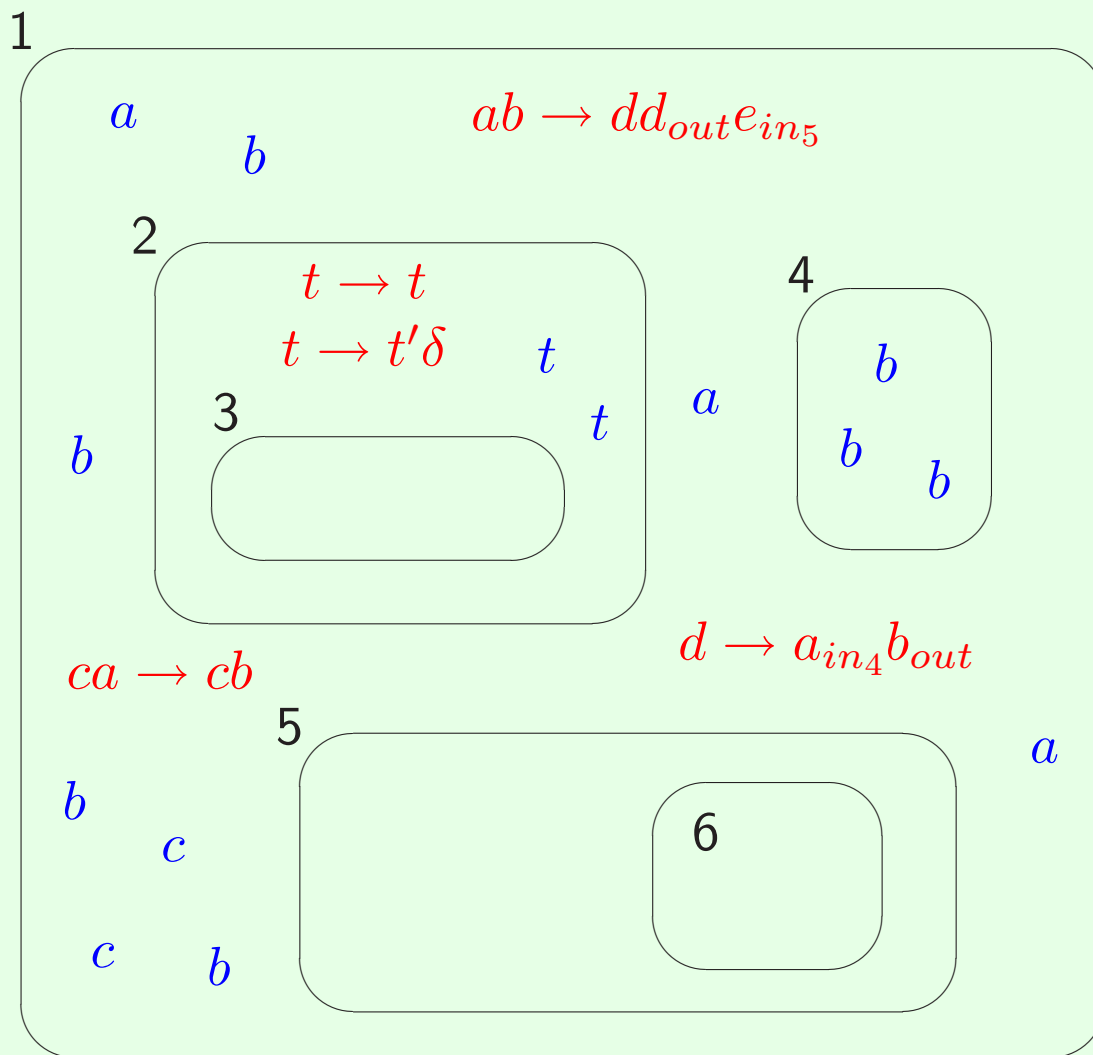
W. Maass movie about spiking neurons:

[http://www.igi.tugraz.at/tnatschl/spike\\_trains\\_eng.html](http://www.igi.tugraz.at/tnatschl/spike_trains_eng.html)

## Basic ingredients of a P system







## Functioning (basic ingredients):

- nondeterministic choice of rules and objects
- maximal parallelism
- transition, computation, halting
- internal output, external output, traces

## Prerequisites:

- strings, languages, multisets
- grammars, automata
- register machines
- computational complexity

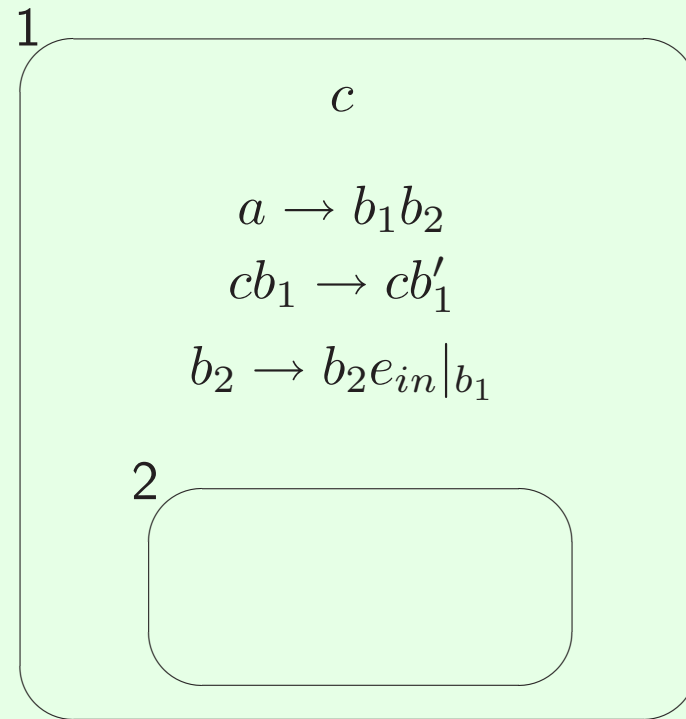
## Basic classes of cell-like P systems

$$\Pi = (O, \mu, w_1, \dots, w_m, R_1, \dots, R_m, i_o),$$

where:

- $O$  = alphabet of objects
- $\mu$  = (labeled) membrane structure of degree  $m$
- $w_i$  = strings/multisets over  $O$
- $R_i$  = sets of evolution rules  
typical form  $ab \rightarrow (a, here)(c, in_2)(c, out)$
- $i_o$  = the output membrane

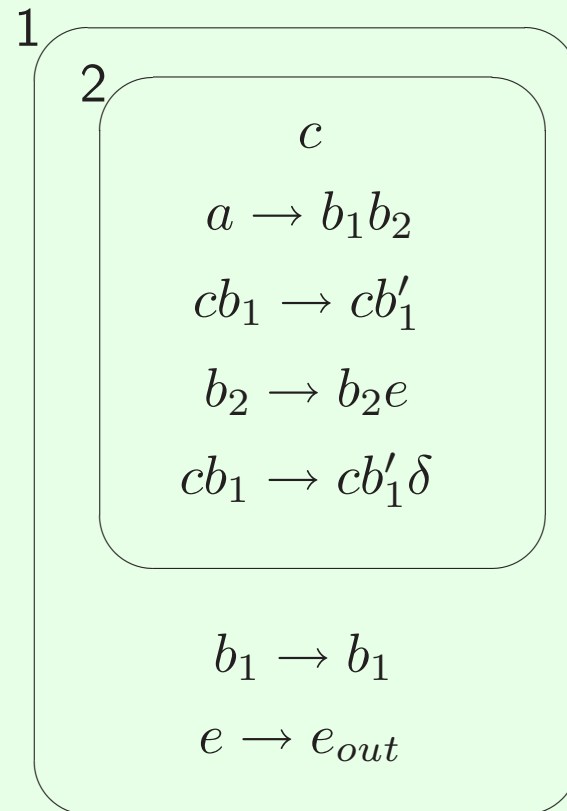
## EXAMPLES



Computing system:  $n \longrightarrow n^2$  (catalyst, promoter, determinism, internal output)

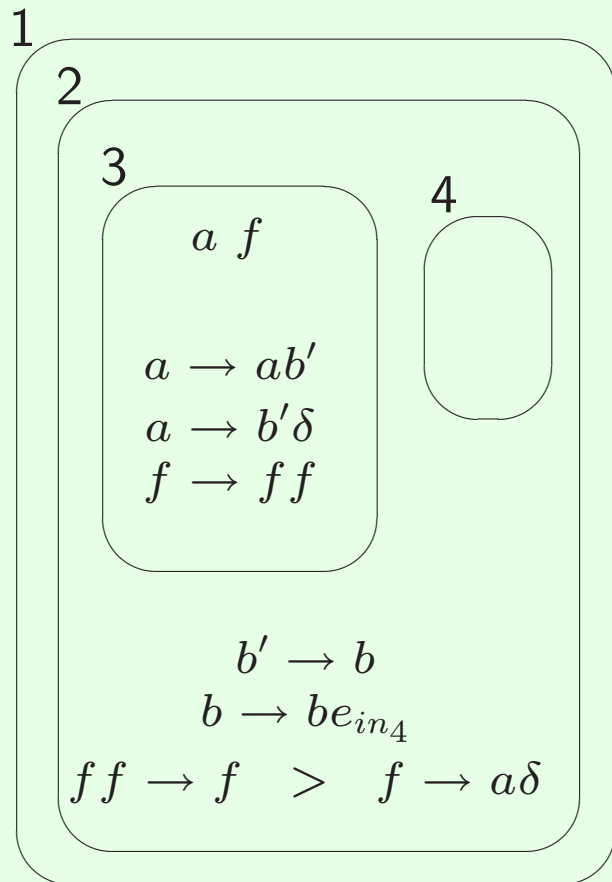
Input (in membrane 1):  $a^n$

Output (in membrane 2):  $e^{n^2}$



The same function ( $n \longrightarrow n^2$ ), with catalyst, dissolution, nondeterminism, external output

Generating :  $\{n^2 \mid n \geq 1\}$



|               |                      |                  |
|---------------|----------------------|------------------|
| 0             | $a f$                |                  |
| 1             | $ab' f f$            |                  |
| ...           | ...                  |                  |
| $m \geq 0$    | $ab'^m f 2^m$        |                  |
| $m + 1$       | $b'^{m+1} f 2^{m+1}$ | $\delta$         |
| $m + 2$       | $b^{m+1} f 2^m$      |                  |
| $m + 3$       | $b^{m+1} f 2^{m-1}$  | $e_{in_4}^{m+1}$ |
| ...           | ...                  | ...              |
| $2m + 1$      | $b^{m+1} f 2$        | $e_{in_4}^{m+1}$ |
| $2m + 2$      | $b^{m+1} f$          | $e_{in_4}^{m+1}$ |
| $2m + 3$      | $b^{m+1} a \delta$   | $e_{in_4}^{m+1}$ |
| $m + 1$ times | <b>HALT!</b>         |                  |

$(m + 1) \times (m + 1)$

$N(\Pi) = \{n^2 \mid n \geq 1\}$

## Computational power (Universality)

Families  $NOP_m(\alpha, tar)$ ,  $\alpha \in \{coo, ncoo, cat\} \cup \{cat_i \mid i \geq 1\}$ ,  $m \geq 1$  or  $m = *$ .

**Lemma 1.** (collapsing hierarchy)  $NOP_*(\alpha, tar) = NOP_m(\alpha, tar)$ ,

for all  $\alpha \in \{ncoo, cat, coo\}$  and  $m \geq 2$ .

**Theorem 1.**  $NOP_*(ncoo, tar) = NOP_1(ncoo) = NCF$ .

Proof: use Lemma 1 and CD grammar systems

**Theorem 2.**  $NOP_*(coo, tar) = NOP_m(coo, tar) = NRE$ , for all  $m \geq 1$ .

**Theorem 3.** [Sosik: 8], [Sosik, Freund: 6], [Freund, Kari, Sosik, Oswald: 2]

$$NOP_2(cat_2, tar) = NRE$$

**Theorem 4.** [Ibarra]  $NRE - NOP_*(cat_1) \neq \emptyset$

( $\implies$  extensions, for more power, with one catalyst – and for adequacy to biology)

## Basic Extensions

Membrane dissolution – action  $\delta$ : rules  $u \rightarrow v\delta$

It helps (but not too much):

**Theorem 5.**  $NE0L \subseteq NOP_2(ncoo, tar, \delta)$ ,

$$NOP_*(ncoo, tar, \delta) \subseteq NET0L.$$

Priority among rules

$$\Pi = (O, \mu, w_1, \dots, w_m, (R_1, \rho_1), \dots, (R_m, \rho_m), i_o)$$

The power of priority:

**Theorem 6 (universality with one catalyst).**

$$NOP_2(cat_1, tar, pri) = NRE.$$

Proof: use matrix grammars with appearance checking (in the Z-binary normal form)

**Theorem 7.** [Sburlan]  $NET0L = NOP_1(ncoo, tar, pri)$ .

**Proof of Theorem 6:**  $NOP_2(cat_1, in, pri) = NRE$

Enough to prove  $NRE \subseteq NOP_2(cat, tar, pri)$ . Consider a matrix grammar with appearance checking  $G = (N, \{a\}, S, M, F)$  in the Z-binary normal form. Assume the grammar given in the standard notation. In particular, this means that for each  $X \in N_1$  there is a matrix  $(X \rightarrow Y, A \rightarrow \alpha)$  in  $M$ .

We construct the following system with catalysts and priorities:

$$\Pi = (O, \{c\}, [[ ]_2]_1, w_1, \lambda, (R_1, \rho_1), (\emptyset, \emptyset), 2),$$

where

$$O = N_1 \cup N_2 \cup \{a, c, H, Z, \#\},$$

$$w_1 = cX_{init}A_{init}H,$$

and the set  $R_1$  contains the following rules ( $h$  is the morphism defined by  $h(\alpha) = \alpha$ ,  $\alpha \in N_2$ , and  $h(a) = (a, in_2)$ ):

$$r_i : X \rightarrow Y,$$

$$r'_i : cA \rightarrow c h(x), \text{ for } m_i : (X \rightarrow Y, A \rightarrow x), 1 \leq i \leq k,$$

$$r_t : cH \rightarrow c\#,$$

$$r_\infty : \# \rightarrow \#,$$

$$r_i : cX \rightarrow cY, \text{ for } m_i : (X \rightarrow Y, A \rightarrow \#), k + 1 \leq i \leq n,$$

$$r_A : A \rightarrow \#, \text{ for all } A \in N_2,$$

$$r_f : H \rightarrow \lambda.$$

The priorities are as follows:

$$r_i > r'_j, \text{ for } 1 \leq i, j \leq k \text{ and } i \neq j,$$

$$r_i > r_A, \text{ for } 1 \leq i \leq k \text{ and } A \in N_2,$$

$$r_i > r_f, \text{ for } 1 \leq i \leq k,$$

$$r_i > r_B, \text{ for } k + 1 \leq i \leq n \text{ and } B \neq A, \text{ where } m_i : (X \rightarrow Y, A \rightarrow \#),$$

$$r_i > r_f, \text{ for } k + 1 \leq i \leq n.$$

All the work is done in the skin membrane; membrane 2 is only used for collecting the result of computations.

Let us assume that in the skin membrane we have a multiset  $cXwH$ ; initially,  $X = X_{init}$  and  $w = A_{init}$ . Because for each  $X \in N_1$  there is a rule  $X \rightarrow Y$  in a matrix from  $M$ , there is a rule  $r_i, 1 \leq i \leq n$ , which can be applied, hence the rule  $r_f : H \rightarrow \lambda$  cannot be used.

We have two possibilities: to use a rule  $r_i : X \rightarrow Y$  with  $1 \leq i \leq k$ , or a rule  $r_i : cX \rightarrow cY$  with  $k + 1 \leq i \leq n$ .

A rule of the first type forbids the use of any rule  $r_B : B \rightarrow \#$ ,  $B \in N_2$ , as well as of all rules  $r'_j$  with  $j \neq i$ . Thus, if  $m_i : (X \rightarrow Y, A \rightarrow x)$  and  $A$  is present in  $w$ , then the rule  $r'_i : cA \rightarrow c h(x)$  is used, and in this way we correctly simulate the matrix  $m_i$ . If  $A$  is not present, then the rule  $r_t : cH \rightarrow c\#$  must be used (because of the maximal parallelism). The trap object  $\#$  will evolve forever by means of the rule  $r_\infty : \# \rightarrow \#$ , hence the computation never stops.

If we start with the rule  $r_i : cX \rightarrow cY$ , for some  $m_i : (X \rightarrow Y, A \rightarrow \#)$ ,  $k + 1 \leq i \leq n$ , then the only applicable rule is  $r_A : A \rightarrow \#$ , because  $r_i$  has priority over all rules  $r_B$  with  $B \neq A$ . Thus, if  $A$  is present, then the trap object is introduced, otherwise not.

Therefore, in both cases we correctly simulate a matrix from  $M$ , and the process can be iterated. When the symbol  $Z$  is introduced (this means that the derivation in  $G$  is terminal), because no rule  $r_i, 1 \leq i \leq n$ , can be applied, we can use the rule  $r_f : H \rightarrow \lambda$ , and the computation stops. The objects  $c$  and  $Z$  remain in the skin membrane; all copies of  $a$  were sent to the output membrane. (If we “wrongly” use the rule  $r_t : cH \rightarrow c\#$ , instead of  $r_f : H \rightarrow \lambda$ , then the computation never stops, but we can avoid this by choosing the rule  $r_f$ ).

Consequently,  $length(L(G)) = N(\Pi)$ , and the proof is complete.

## Variants of P systems

### Types of rules:

$u \rightarrow v$  with targets in  $v$   
(possibly conditional: promoters or inhibitors)  
particular cases:  $ca \rightarrow cu$  (catalytic)  
 $a \rightarrow u$  (non-cooperative)

$(ab, in), (ab, out)$  – symport (in general,  $(x, in), (x, out)$ )  
 $(a, in; b, out)$  – antiport (in general,  $(u, in; v, out)$ )

$xu]_i.vy \rightarrow xu']_i.u'y$  – boundary (Manca, Bernardini)

$ab \rightarrow a_{tar_1}b_{tar_2}$  – communication (Sosik)

$ab \rightarrow a_{tar_1}b_{tar_2}c_{come}$

$a \rightarrow a_{tar}$

$$\begin{aligned}
a[ ]_i &\rightarrow [b]_i \\
[a]_i &\rightarrow b[ ]_i \\
[a]_i &\rightarrow b \\
a &\rightarrow [b]_i \\
[a]_i &\rightarrow [b]_j[c]_k \\
[a]_i[b]_j &\rightarrow [c]_k \\
[a]_i[ ]_j &\rightarrow [[b]_i]_j \\
[[a]_i]_j &\rightarrow [b]_i[ ]_j \\
[u]_i &\rightarrow [ ]_i[u]_{@j} \\
[Q]_i &\rightarrow [O - Q]_j[Q]_k
\end{aligned}$$

go in  
 go out  
 membrane dissolution  
 membrane creation  
 membrane division  
 membrane merging  
 endocytosis  
 exocytosis  
 gemmation  
 separation

and some (a few) others

## **P systems with symport/antiport rules**

Symport rules  $(x, in)$  or  $(x, out)$

$(|x| = \text{weight})$

Antiport rules  $(x, out; y, in)$ ,  $x, y$  strings

$(\max(|x|, |y|) = \text{weight})$

System

$$\Pi = (O, \mu, w_1, \dots, w_m, E, R_1, \dots, R_m, i_o),$$

where  $E \subseteq O$  is the set of objects which appear in the environment in arbitrarily many copies

Families  $NOP_m(sym_p, anti_q)$

Power:

**Theorem 8 (universality).**

$$\begin{aligned} NRE &= NOP_1(sym_0, anti_2) \\ &= NOP_3(sym_2, anti_0) \\ &= NOP_2(sym_3, anti_0) \\ &= NOP_3(sym_1, anti_1) \end{aligned}$$

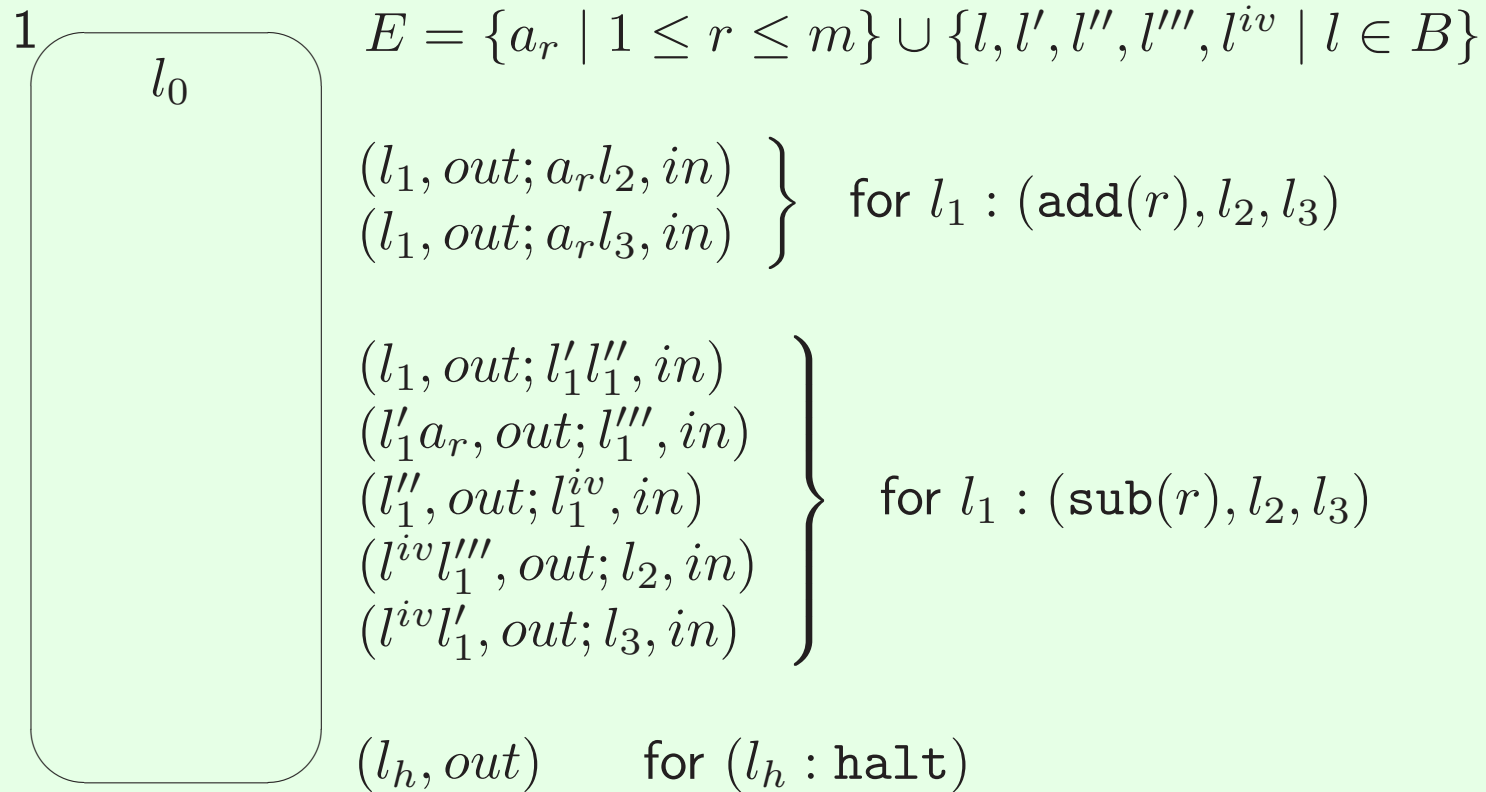
[Freund, Păun]

[Frisco, Hoogeboom]

[FH], [A. Păun]

[Vaszil]

An example of a symport/antiport P system (simulating a register machine)



## Tissue-like and neural-like P systems

Idea: membrane placed in the nodes of a graph, “go” for communication

The case of symport/antiport rules:

- symport rules  $(i, x, j)$
- antiport rules  $(i, x/y, j)$  (environment labeled with 0)

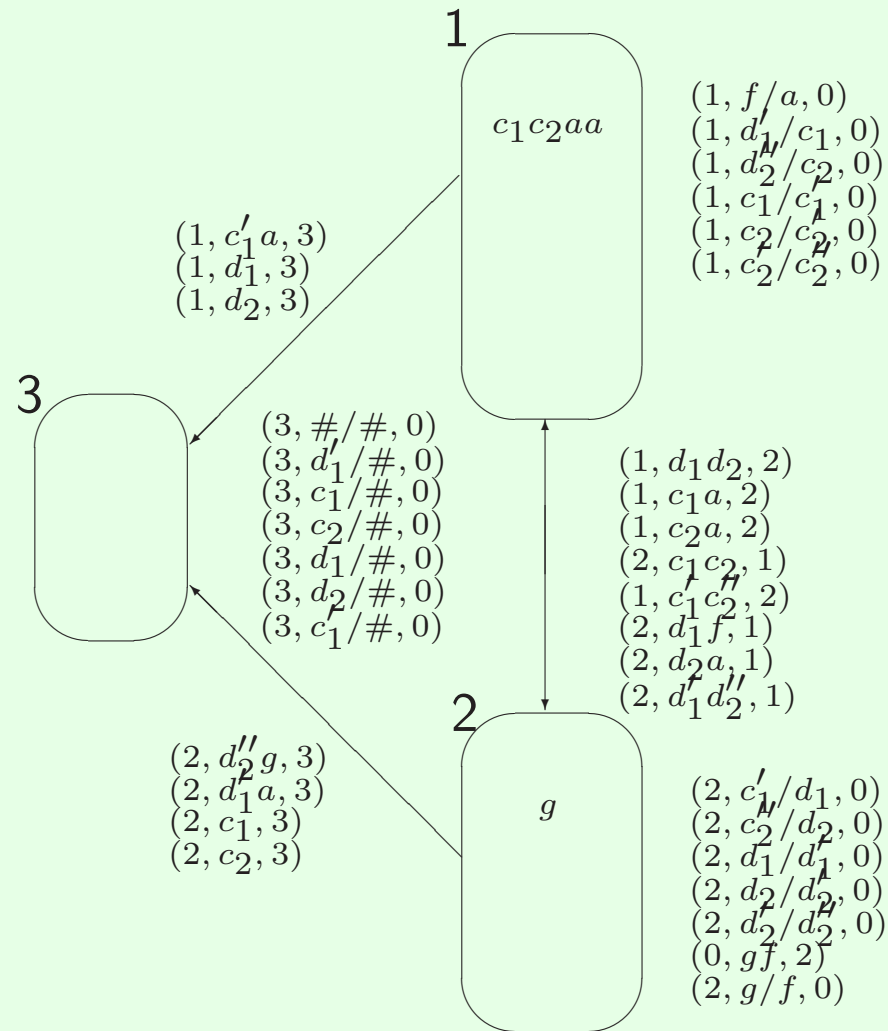
System

$$\Pi = (O, w_1, \dots, w_m, E, R, i_o),$$

where  $E$  the set of objects present in arbitrarily many copies in the environment

Families  $NOtP_{m,n}(sym_p, anti_q)$  (at most  $m$  membranes, in a graph with at most  $n$  nodes in a connected component)

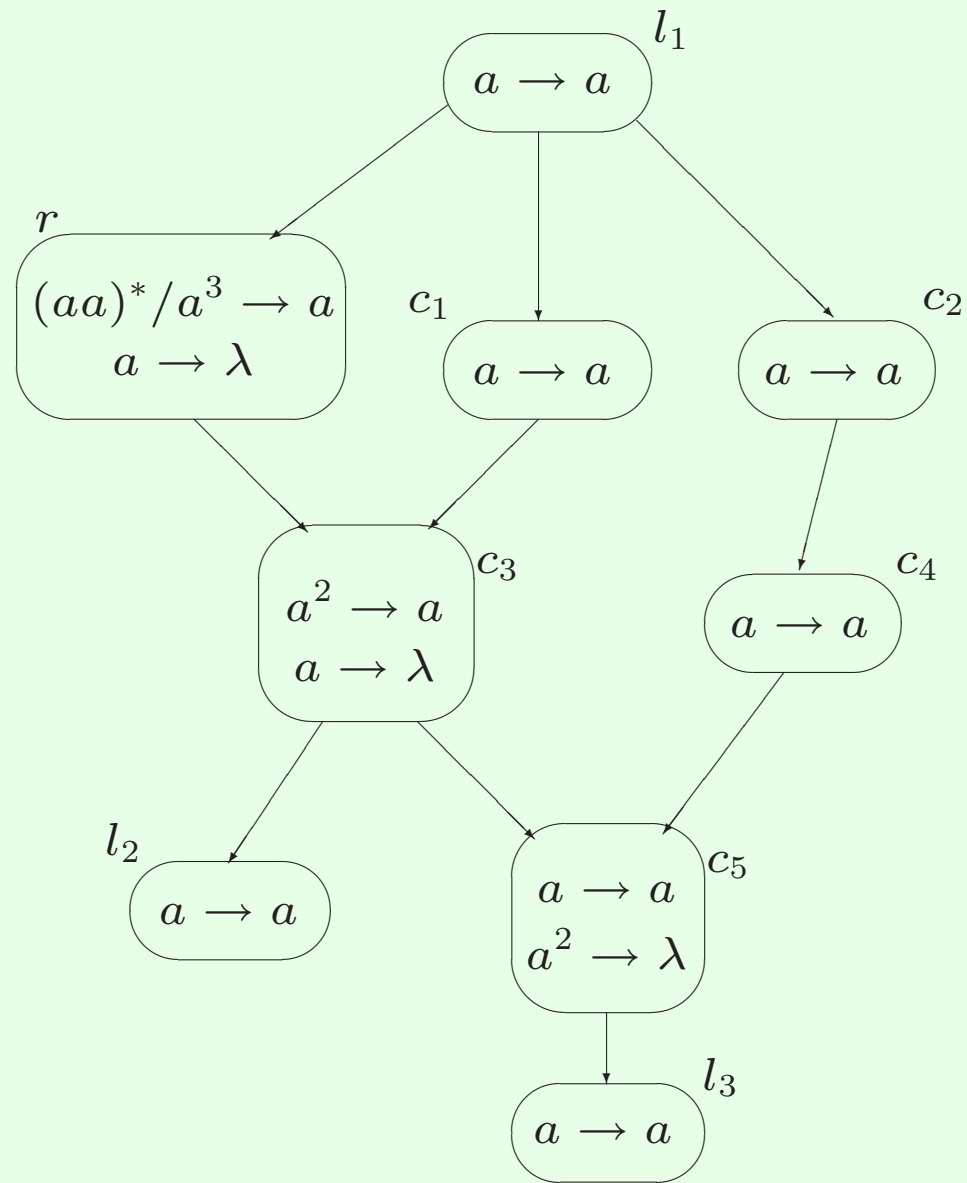
Example:  $N(\Pi) = \{2^n \mid n \geq 2\}$



**Theorem 9.**  $NRE = NOtP_{2,2}(sym_1, anti_1)$   
 $= NOtP_{2,2}(sym_2, anti_0)$

[Freund et al.]

Neural-like P systems



$l_1 : (\text{SUB}(r), l_2, l_3)$

**FORMAL DEFINITION:** a *spiking neural P system* (in short, an SN P system), of degree  $m \geq 1$ , is a construct of the form

$$\Pi = (O, \sigma_1, \dots, \sigma_m, \text{syn}, \text{in}, \text{out}),$$

where:

1.  $O = \{a\}$  is the singleton alphabet ( $a$  is called *spike*);
2.  $\sigma_1, \dots, \sigma_m$  are *neurons*, of the form

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m,$$

where:

- a)  $n_i \geq 0$  is the *initial number of spikes* contained by the neuron;
- b)  $R_i$  is a finite set of *rules* of the following two forms:

- (1)  $E/a^c \rightarrow a; d$ , where  $E$  is a regular expression with  $a$  the only symbol used,  $c \geq 1$ , and  $d \geq 0$ ;
  - (2)  $a^s \rightarrow \lambda$ , for some  $s \geq 1$ , with the restriction that  $a^s \in L(E)$  for no rule  $E/a^c \rightarrow a; d$  of type (1) from  $R_i$ ;
3.  $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$  with  $(i, i) \notin syn$  for  $1 \leq i \leq m$  (*synapses among neurons*);
  4.  $in, out \in \{1, 2, \dots, m\}$  indicate the *input* and the *output neuron*.

only **out** = generative system

only **in** = accepting system

both **in, out** = computing system

Spike trains, types of output

FAMILIES:  $Spik_{gen}P_m(rule_k, cons_p, forg_q)$  – generative

$Spik_{acc}P_m(rule_k, cons_p, forg_q)$  – accepting ( $DSpik$ , if deterministic)

**Theorem 1.**  $NFIN = Spik_{gen}P_1(rule_*, cons_1, forg_0) = Spik_{gen}P_2(rule_*, cons_*, forg_*)$

**Theorem 2.**  $Spik_{gen}P_*(rule_2, cons_3, forg_3) = Spik_{acc}P_*(rule_2, cons_3, forg_2) = NRE$ .

**Theorem 3.**  $SLIN_1 = Spik_{gen}P_*(rule_k, cons_p, forg_q, bound_s)$ , for all  $k \geq 3$ ,  $q \geq 3$ ,  $p \geq 3$ , and  $s \geq 3$ .

Normal forms

## PROBLEMS:

- Theorems 1, 3 for the accepting case
- Find classes of systems for which  $D < ND$

Actually, **strong determinism**:  $L(E) \cap L(E') = \emptyset$  in each neuron

- Find classes of systems for which  $SD < D$

## SMALL UNIVERSAL SN P SYSTEMS

**Theorem 4.** *There is a universal computing SN P system with standard rules having 84 neurons, and one with extended rules which has 49 neurons.*

**Theorem 5.** *There is a universal generating SN P system with standard rules having 76 neurons, and one with extended rules which has 50 neurons.*

Korec, TCS, 1996 (plus “code optimization”)

Generating/translating languages, processing infinite sequences, etc.

## Solving hard problems in a feasible time

### Complexity Classes for Membrane Computing

Let  $f : \mathbf{N} \longrightarrow \mathbf{N}$ , “acceptable”,  $X$  a given class of membrane systems, and  $A$  a decidability problem; denote by  $A(n)$  the instance of size  $n$  of  $A$ .

Problem  $A$  belongs to  $\mathbf{MC}_X(f)$  if a family of membrane systems  $\Pi_A = (\Pi_A(1), \Pi_A(2), \dots)$  of type  $X$  exists such that:

1.  $\Pi_A$  is an *semi-uniform* family: there is a Turing machine which constructs  $\Pi_A(n)$  in polynomial time starting from  $A(n)$  (then  $\Pi_A(n)$  has a polynomial size)
2. Each  $\Pi_A(n)$  is *confluent*: there is a constant  $t_{A(n)}$  and a distinguished object *yes* such that  $\Pi_A(n)$  always halts in less than  $t_{A(n)}$  steps, and either it sends out the object *yes* exactly at time  $t_{A(n)}$ , or this object remains inside the system;

3.  $\Pi_A(n)$  is *sound*, that is, it answers the instance of size  $n$  of problem  $A$  in the following way: if  $\Pi_A(n)$  sends out the object *yes*, then the answer to  $A(n)$  is “yes”; if  $\Pi_A(n)$  halts without sending out the object *yes*, then the answer to  $A(n)$  is “no”.
4.  $\Pi_A$  is *f-efficient*, that is,  $t_{A(n)} \leq f(n)$  for all  $n \geq 1$ .

Important: the computations can be nondeterministic, the system can grow exponentially during the computation

The union of all classes  $\mathbf{MC}_X(f)$  for  $f$  a linear function is denoted by  $\mathbf{LMC}_X$ , and the union of all classes  $\mathbf{MC}_X(f)$  for  $f$  a polynomial is denoted by  $\mathbf{PMC}_X$ .

Problems with  $\mathbf{LMC}_X$ : systems constructed in polynomial time, reduction of problems in polynomial time

Suggestion: construction of systems by P systems, reductions by P systems

Ways to get exponential workspace:

1. membrane division
2. membrane creation
3. string replication

Membrane division (systems with active membranes):

$$\Pi = (O, H, \mu, w_1, \dots, w_m, R),$$

where (polarized membranes) the rules of  $R$  are of the forms

(a)  $[a \rightarrow v]_h^\alpha$ ,  
for  $h \in H, \alpha \in \{+, -, 0\}, a \in O, v \in O^*$

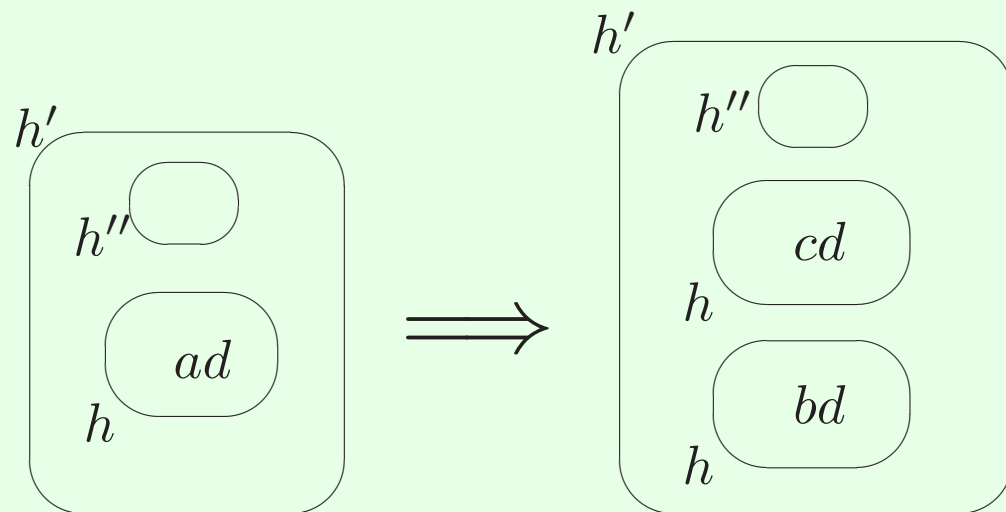
(b)  $a[ ]_h^{\alpha_1} \rightarrow [b]_h^{\alpha_2}$ ,  
for  $h \in H, \alpha_1, \alpha_2 \in \{+, -, 0\}, a, b \in O$

(c)  $[a]_h^{\alpha_1} \rightarrow [ ]_h^{\alpha_2} b$ ,  
for  $h \in H, \alpha_1, \alpha_2 \in \{+, -, 0\}, a, b \in O$

(d)  $[a]_h^\alpha \rightarrow b$ ,  
for  $h \in H, \alpha \in \{+, -, 0\}, a, b \in O$

$$(e) [a]_h^{\alpha_1} \rightarrow [b]_h^{\alpha_2} [c]_h^{\alpha_3},$$

for  $h \in H, \alpha_1, \alpha_2, \alpha_3 \in \{+, -, 0\}, a, b, c \in O$



Nondeterministic, maximally parallel use of rules, in the bottom-up mode, transition, halting computations, result  $N(\Pi)$ , family  $NOP_m(activ, (a, b, c))$ , etc

## Results

**Theorem 10 (universality).**  $NO P_3(activ, (a, b, c)) = NRE$ .

**Theorem 11 (efficiency).** The SAT problem can be solved in linear time with respect to the number of variables and the number of clauses by a P system with active membranes using rules of the forms  $(a)$ ,  $(b)$ ,  $(c)$ ,  $(e)$  (with 2-division only).

Proof. Let us consider a propositional formula  $C = C_1 \wedge C_2 \wedge \dots \wedge C_m$ , with  $C_i = y_{i,1} \vee \dots \vee y_{i,p_i}$ , for some  $m \geq 1, p_i \geq 1$ , and  $y_{i,j} \in \{x_k, \neg x_k \mid 1 \leq k \leq n\}$ , for each  $1 \leq i \leq m, 1 \leq j \leq p_i$ .

We construct the system

$$\Pi = (O, H, \mu, w_1, w_2, R),$$

where

$$O = \{a_i, t_i, f_i \mid 1 \leq i \leq n\} \cup \{r_i \mid 0 \leq i \leq m\} \\ \cup \{c_i \mid 1 \leq i \leq m+1\} \cup \{d_i \mid 0 \leq i \leq n\} \cup \{yes\},$$

$$H = \{1, 2\},$$

$$\mu = \left[ \begin{array}{c} 0 \\ 2 \end{array} \right]_1^0,$$

$$w_1 = \lambda,$$

$$w_2 = a_1 a_2 \dots a_n d_0,$$

and the set  $R$  contains the following rules (we also give explanations about the role of these rules in the computation meant to solve SAT):

1.  $[a_i]_2^0 \rightarrow [t_i]_2^0 [f_i]_2^0, 1 \leq i \leq n.$

The objects  $a_i$  correspond to variables  $x_i, 1 \leq i \leq n.$  By using a rule as above, for  $i$  non-deterministically chosen, we produce the truth values *true* and *false* assigned to variable  $x_i,$  placed in two separate copies of membrane 2. Note that the charge remains the same for both membranes, namely neutral, hence the process can continue. In this way, in  $n$  steps we assign truth values to all variables, hence we get all  $2^n$  truth-assignments, placed in  $2^n$  separate copies of membrane 2. In turn, these  $2^n$  copies of membrane 2 are placed in membrane 1 – the system always has only two levels of membranes. Note that in spite of the fact that in each step the object  $a_i$  is non-deterministically chosen, after  $n$  steps we get the same result, irrespective of which objects were used in each step.

$$2. [d_k \rightarrow d_{k+1}]_2^0, \quad 0 \leq k \leq n - 2.$$

$$3. [d_{n-1} \rightarrow d_n c_1]_2^0.$$

$$4. [d_n]_2^0 \rightarrow [ ]_2^+ d_n.$$

The objects  $d_i$  are counters. Initially,  $d_0$  is placed in membrane 2. By division (when using rules of type 1), we introduce copies of the counter in each new membrane. In each step, in each membrane with label 2, we pass from  $d_k$  to  $d_{k+1}$ , thus “counting to  $n$ ”. In step  $n$  we introduce both  $d_n$  and  $c_1$ ; the former objects will exit the membrane (at step  $n + 1$ ), changing its polarization to positive, the latter one will be used at the subsequent steps as shown below. Note that at step  $n$  we have also completed the generation of all truth-assignments.

$$5. [t_i \rightarrow r_{h_{i,1}} \cdots r_{h_{i,j_i}}]_2^+, \quad 1 \leq i \leq n, \text{ and the clauses } C_{h_{i,1}}, \dots, C_{h_{i,j_i}} \text{ contain the literal } x_i.$$

6.  $[f_i \rightarrow r_{h_{i,1}} \dots r_{h_{i,j_i}}]_2^+$ ,  $1 \leq i \leq n$ , and the clauses  $C_{h_{i,1}}, \dots, C_{h_{i,j_i}}$  contain the literal  $\neg x_i$ .

In one step, in parallel in all  $2^n$  membranes with label 2 (and polarization +), we look for the clauses satisfied by the truth-assignments from each membrane. This is also done in parallel for all truth values present in each membrane (the rules of type 5, 6 are used in parallel, they do not actively involve the membranes). If, after completing this step, there is at least one membrane with label 2 which contains all symbols  $r_1, r_2, \dots, r_m$ , this means that the truth-assignment from that membrane satisfies all clauses, hence it satisfies formula  $C$ . Otherwise (if in no membrane we get all objects  $r_1, r_2, \dots, r_m$ ), the formula  $C$  is not satisfiable. Note that the satisfiability was already solved, in  $n + 2$  steps, making essential use of the parallelism, but we still have to “read” the answer, and send a suitable signal out of the system. This will be done by the following rules.

7.  $[r_1]_2^+ \rightarrow [ ]_2^- r_1$ .

At the same time for all  $2^n$  membranes with label 2, we check whether or not  $r_1$  is present in each membrane. If this is the case, then  $r_1$  is sent out of the membrane where it is present, changing in this way the polarization of that membrane, to negative. The membranes which do not contain the object  $r_1$  remain positively charged and they will no longer evolve, as no further rule can be applied to them.

8.  $[c_i \rightarrow c_{i+1}]_2^-, 1 \leq i \leq m$ .

9.  $[r_k \rightarrow r_{k-1}]_2^-, 2 \leq k \leq m$ .

10.  $r_1[ ]_2^- \rightarrow [r_0]_2^+$ .

In the copies of membrane 2 with a negative polarization, hence only those where we have found  $r_1$  in the previous step, we perform two operations in parallel: we count the number of satisfied clauses, by means of increasing the subscript of  $c_k$ , and we decrease the subscripts of all objects  $r_j$  from that membrane. Thus, if the second clause was satisfied by the truth-assignment from a membrane, that is  $r_2$  was present in a membrane, then  $r_2$  becomes  $r_1$  – hence the rule of type 7 can be applied again. Note the important fact that passing from  $r_2$  to  $r_1$  is possible only in membranes where we already had  $r_1$ , hence we check whether the second clause is satisfied only after knowing that the first clause was satisfied. At the same time, the object  $r_1$  from the skin membrane returns to membranes with label 2, changed to  $r_0$  (which will never evolve again), and returning the polarization of the membrane to positive (this makes possible the use of rules of type 7).

Note also another important fact: we have in the skin membrane a number of copies of  $r_1$  equal to the number of membranes with negative charge; a rule of type 7 is of general type (b), which involves the membrane; thus, because of this fact and because of the parallelism, each membrane which previously contained an object  $r_1$  will now contain an object  $r_0$ .

The rules of types 7, 8, 9, 10 are applied as many times as possible (in one step rules of type 7, in the next one rules of types 8, 9, and 10, and then we repeat the cycle). Clearly, if a membrane does not contain an object  $r_i$ , then that membrane will stop evolving at the time when  $r_i$  is supposed to reach the subscript 0. In this way, after  $2m$  steps we can find whether there is a membrane which contains all objects  $r_1, r_2, \dots, r_m$ . The membranes with this property, and only they, will get the object  $c_{m+1}$ . Note that at that moment the last object  $r_1$ , originating in  $r_m$ , enters the corresponding membrane, and changes its polarization to positive.

11.  $[c_{m+1}]_2^+ \rightarrow [ ]_2^+ yes.$

12.  $[yes]_1^0 \rightarrow [ ]_1^0 yes.$

The object  $c_{m+1}$  exits the membrane that has contained a truth-assignment that has satisfied formula  $C$ , producing the object  $yes$ . This object is then sent to the environment, telling us that the formula is satisfiable, and the computation stops. This is the  $(n + 2m + 4)$ th step of the computation. If formula  $C$  is not satisfiable, then the computation stops earlier and we get no output, because no membrane was able to produce the object  $c_{m+1}$ , hence the object  $yes$ .

The satisfiability of formula  $C$  was decided in linear time, and this concludes the proof.

Let us denote by  $D$  the class of P systems with active membranes, using rules of types (a), (b), (c), (e), with 2-division.

**Theorem 12.**  $\text{SAT} \in \text{LMC}_D$

Proof. We have to check the four conditions from the definition of the class  $\text{LMC}_D$  for the previous family of membrane systems. Consider  $m$  fixed, and denote by  $\Pi_{\text{SAT}}(n)$  the system constructed for an instance  $C = C_1 \wedge C_2 \wedge \dots \wedge C_m$  of SAT which involves  $n$  variables. From the proof of the previous theorem it is clear that the system  $\Pi_{\text{SAT}}(n)$  always halts; specifically, it either halts after  $t_{\text{SAT}}(n) = n + 2m + 4$  steps and sends out the object *yes*, or halts earlier but in that case no object is sent out. Thus, the system is confluent, and linearly efficient. From the explanations given together with the rules, we also know that the system is sound: the instance  $\text{SAT}(n)$  is satisfiable if and only if the system  $\Pi_{\text{SAT}}(n)$  sends the object *yes* to the environment.

It remains to prove that the family  $\Pi_{\text{SAT}}(n)$  is uniform, that is it can be constructed in polynomial time starting from  $\text{SAT}(n)$ . A detailed, formal construction of a Turing machine which reads an instance of SAT and produces a P system as in the previous proof is totally cumbersome, although straightforward in principle. Here are some hints for such a construction.

We codify all subscripts in a given base.

Suppose that the SAT instance is given in a standard normal form, that is without any clause containing a literal twice, with the clauses numbered from 1 to  $m$ , and with the variables numbered from 1 to  $n$  (hence, without gaps in numberings). Thus, the longest subscript will have  $\max(n, m)$  digits even when the codification is made in base 1. This observation gives precise upper bounds for the length of rules of our systems (the rules involve objects associated with variables and with clauses, membranes of two types only, polarizations of three types, as well as additional symbols, such as  $\rightarrow$  and the parentheses used for indicating the membranes). Note that we have in total  $4n + 2m + 4$  rules, but the rules are clustered in types.

Suppose also that the instance we consider is non-trivial, in the sense that there is no clause  $C_i$  which contains both  $x_j$  and  $\neg x_j$ ; a clause which contains both a variable and its negation is always true, hence it can be removed without changing the satisfiability of  $C$ . This implies the fact that each clause contains at most  $n$  literals (one further literal would repeat a variable).

Consider, for example, the sequence of rules of type 1; the same considerations hold for all types. We have the sequence

$$[a_1]_2^0 \rightarrow [t_1]_2^0 [f_1]_2^0, [a_2]_2^0 \rightarrow [t_2]_2^0 [f_2]_2^0, \dots, [a_n]_2^0 \rightarrow [t_n]_2^0 [f_n]_2^0.$$

(We have used a comma to separate the rules.) Such a sequence can be constructed as follows: write  $[a_1]_2^0 \rightarrow [t_1]_2^0 [f_1]_2^0$ , on the tape of the Turing machine. This takes a constant time, proportional to the number of symbols we have to write. Then, copy this sequence of symbols to the right of the comma, at the same time increasing the subscripts of  $a, t, f$  by one. Continue in this way, copying the string of symbols between the last two commas to the right of the last comma, at the same time increasing the subscripts of  $a, t, f$ . When the subscripts become equal to  $n$ , stop.

In order to copy a string of length  $k$  to the right of it, with a marker in between the two copies, we have to move the head of the Turing machine back and forth for each symbol of the string, carrying it (mark it, go to the right, write the symbol, return to the marked symbol, unmark it and continue with the symbol placed to the left of it, and so on and so forth) at most a distance  $k$ . This means a number of steps of the order of  $k^2$ . (The copying procedure can be more efficient if at the same time we carry  $r$  symbols, for a given  $r$  greater than 1, but still the time is quadratic.) The longest rule in our sequence has the length of the order of  $n$  (actually, something like  $3n + q$ , depending on the way of codifying the auxiliary symbols  $a, t, f, [, ], \rightarrow, +, -, 1$ ). Therefore, copying one rule takes a time of the order of  $n^2$ . We have  $n$  rules, hence the total time is of the order of  $n^3$ .

Similar reasoning holds for all rules of types 1, 2, 8, 9 (where we have a sequence of rules), while for rules of types 3, 4, 7, 10, 11, 12 the polynomial bound of steps is ensured by the fact that we have a single rule of each type.

The case of rules of types 5, 6 is slightly different, because also the clauses are involved. For each rule of the form  $[t_i \rightarrow r_{h_{i,1}} \cdots r_{h_{i,j_i}}]_2^+$ ,  $1 \leq i \leq n$ , we also have to parse each clause of our formula from left to right. Because we deal with formulae which are non-trivial, the length of  $C$  is bounded, and we have at most  $n$  literals in each clause. Therefore, the string we have to parse when writing a rule as mentioned above is polynomially bounded with respect to  $m$  and  $n$ . Then, writing the whole sequence of rules is polynomially bounded with respect to the length of one rule, the longest one. In total, a polynomial time is needed for constructing all rules of type 5. The same assertion holds for rules of type 6.

Consequently, when  $m$  is fixed, we can construct all rules of  $\Pi$  in a time which is bounded by a polynomial with respect to  $n$ . If we also consider  $m$  as a variable, then the polynomial will depend on both  $n$  and  $m$ .

Most of the previous operations – especially, copying strings – can be performed much faster if we use multi-tape Turing machines, but we are only interested in the fact that the total time is polynomial, without taking care of the degree or the coefficients of the specific polynomial which bounds the total time.

Clearly, also the other elements of  $\Pi_{\text{SAT}}(n)$  – the alphabet, the set of labels, the initial membrane structure, and the contents of the two initial regions – can be constructed in a polynomial time with respect to  $n$  (and  $m$ ). We conclude that the family of membrane systems we use is uniform, and this ends the proof of the fact that  $\text{SAT} \in \mathbf{LMC}_D$ .

**Corollary.**  $SAT \in LMC_D$ .

**Theorem 13.** The directed HPP can be solved in linear time with respect to the number of nodes of the graph by P systems using rules of the forms (a), (b), (c), and (e), with d-division, and in quadratic time when using 2-division.

Improved results (uniform instead of semi-uniform constructions, deterministic instead of confluent computations, etc):

[Sevilla team: Perez-Jimenez, Sancho-Caparrini,  
Riscos-Nunez, etc]  
[Alhazov, Pan, etc]

Using division for non-elementary membranes:

**Theorem 14.**  $QBS \in \mathbf{PSPACE} \cap \mathbf{PMC}_{D'}$

[Sosik], [Alhazov, Pan]

Conjecture [Sosik]: division of non-elementary membranes is necessary

## Applications:

- biology, medicine (continuous versus discrete mathematics)
- computer science (computer graphics, sorting/ranking, 2D languages, cryptography, general model of distributed-parallel computing)
- linguistics (modeling framework, parsing)
- optimization (membrane algorithms [Nishida, 2004])
- economics ([Warsaw group], numerical P systems)

## A recent application in biology/medicine:

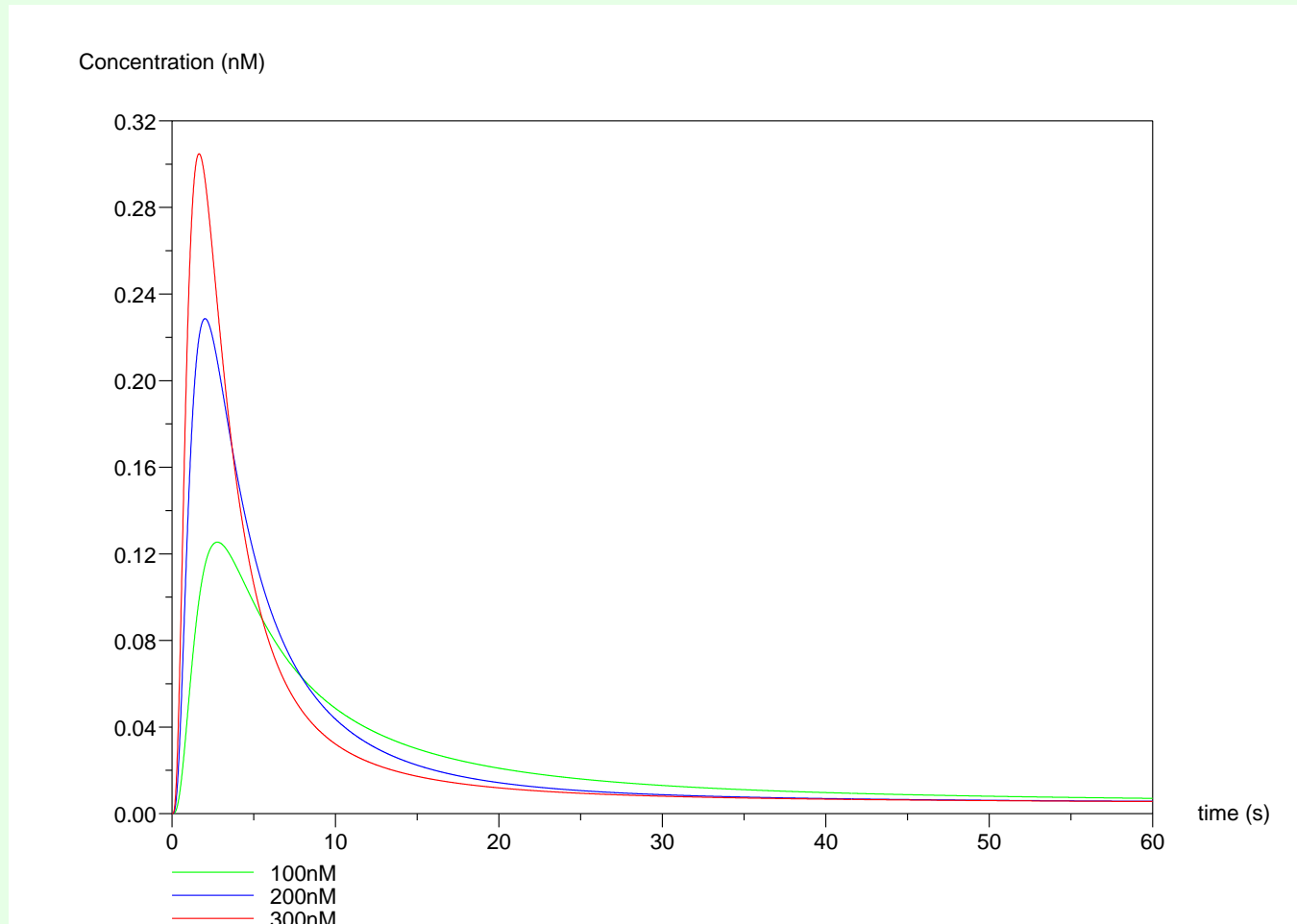
M.J. Pérez–Jiménez, F.J. Romero–Campero:

A Study of the Robustness of the EGFR Signalling Cascade Using Continuous Membrane Systems.

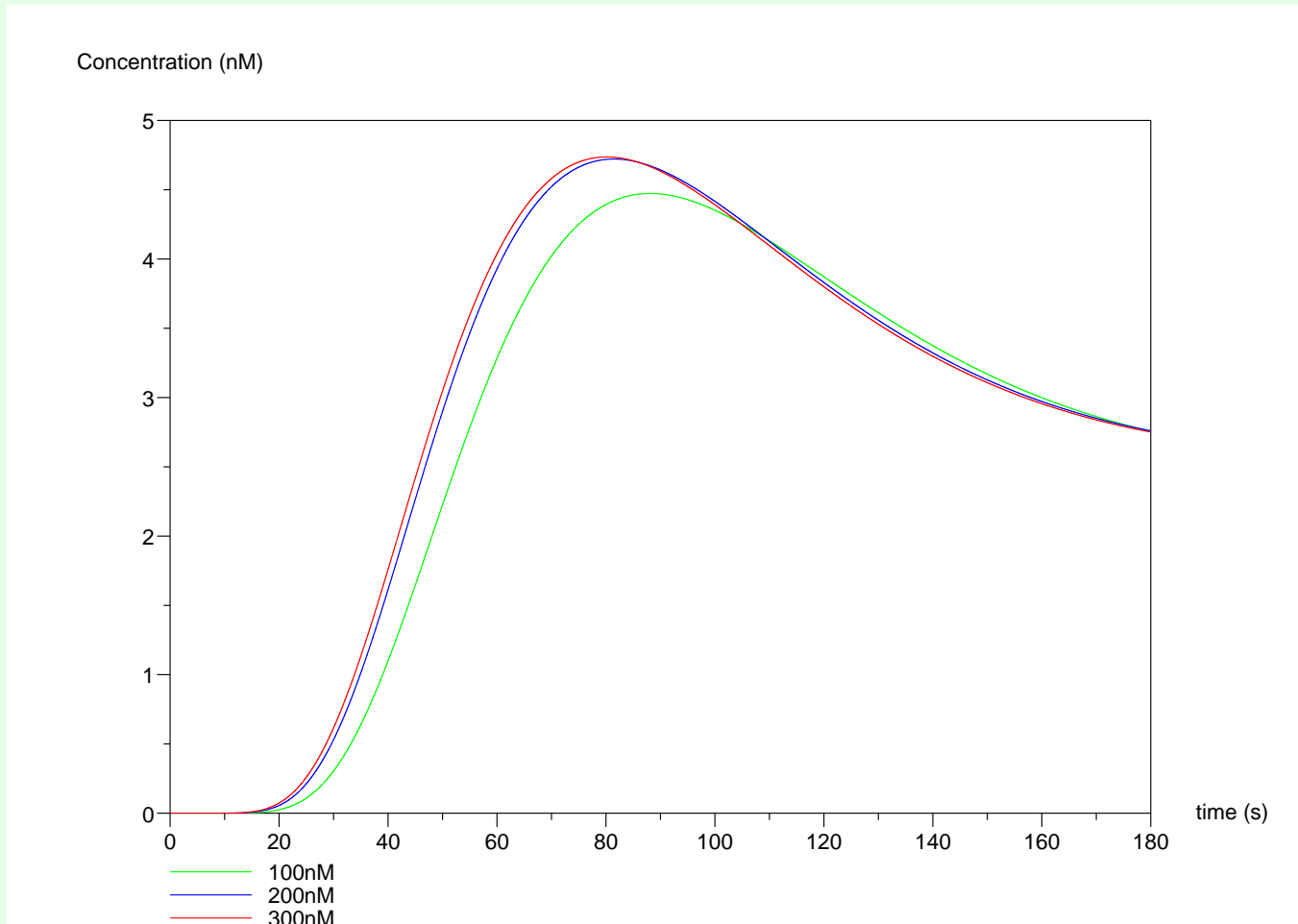
In *Mechanisms, Symbols, and Models Underlying Cognition. First International Work-Conference on the Interplay between Natural and Artificial Computation, IWINAC 2005* (J. Mira, J.R. Alvarez, eds.), LNCS 3561, Springer, Berlin, 2005, 268–278.

- 60 proteins, 160 reactions/rules
- reaction rates from literature
- results as in experiments

## Typical outputs:



The EGF receptor activation by auto-phosphorylation  
(with a rapid decay after a high peak in the first 5 seconds)



The evolution of the kinase MEK  
(proving a surprising robustness of the signalling cascade)

## Applications in economics:

- J. Bartosik, W. Korczynski, etc (accounting, human resource management, etc)
- Gh. Păun, R. Păun (general interpretation, paired rules:  $([u \rightarrow v]_i; [u' \rightarrow v']_j)$ )
- Gh. Păun, R. Păun: [Numerical P systems](#)

**Basic idea:** numerical variables in regions, evolving by “production functions”, whose value is distributed according to “repartition protocols”; dynamical systems approach (sequences of configurations), but also computing device (the set of values of a specified variable).

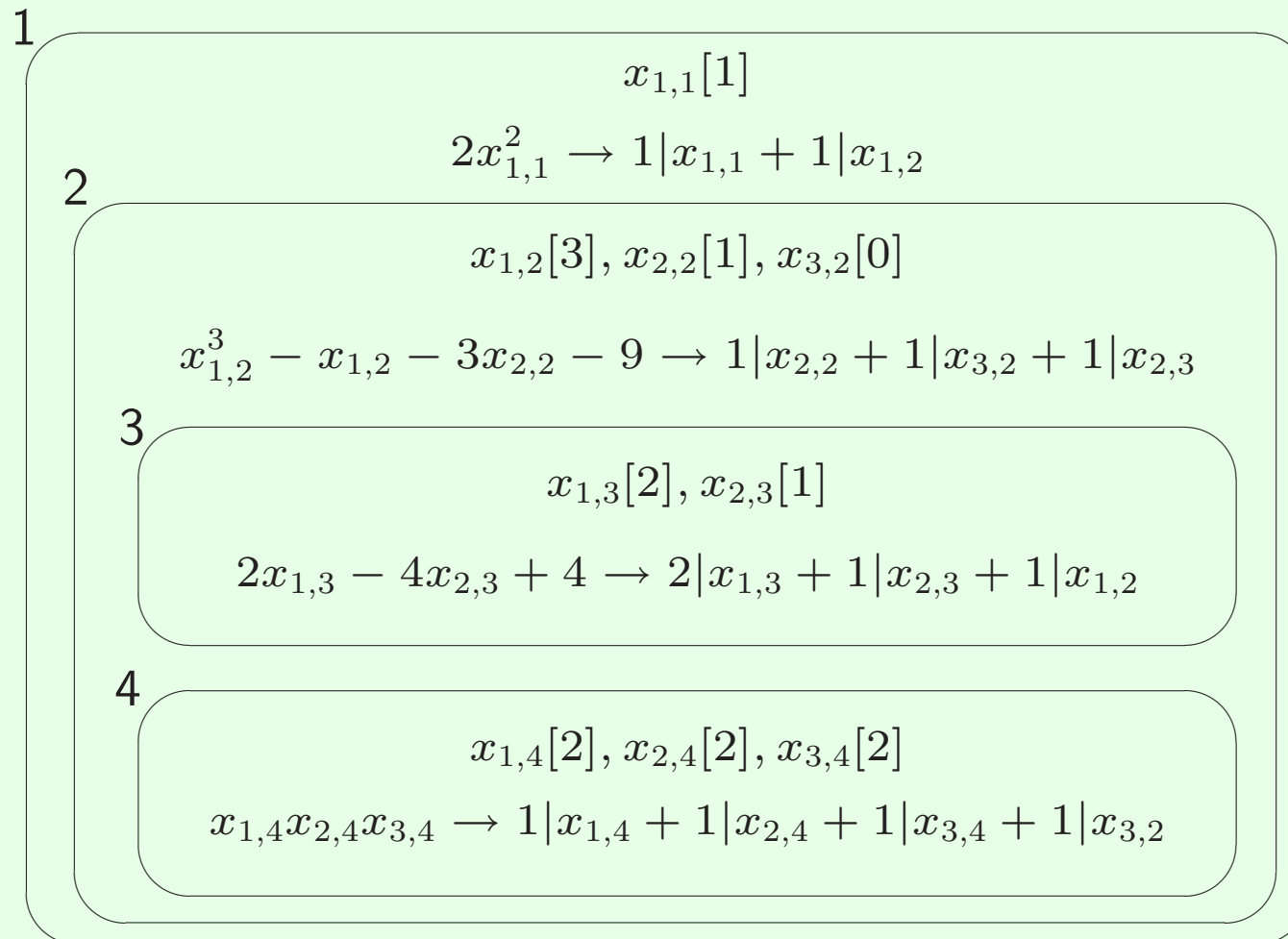
### Theorem 15.

$$SLIN_1^+ \subset DSET^+ P_*(poly^1(1), nneg, div)$$

$$N^+RE = SET^+ P_8(pol^5(5), div) = SET^+ P_7(poly^5(6), div)$$

+ many research topics and open problems

Example:



## Some recent results/ideas/applications:

- Bounding the number of objects used (descriptive complexity)  
Gh. Păun, J. Pazos, M.J. Pérez-Jiménez, A. Rodríguez-Patón:  
Symport/antiport P systems with three objects are universal,  
*Fundamenta Informaticae*, 64, 1–4 (2005), 353–367

[Freund et al., 2005]

- $n \geq 2$  objects and  $m \geq 1$  membranes with  $n + m = 6$  sufficient!
- 1 object is sufficient for tissue P systems

- Working with objects **on** membranes (like in brane calculi [Cardelli et al.])  
Operations: phago, pino, exo, mate, drip, bud

$$\begin{aligned}
 \text{mate} & : [ ]_{ua} [ ]_v \rightarrow [ ]_{uxv}, \\
 \text{drip} & : [ ]_{uav} \rightarrow [ ]_{ux} [ ]_v,
 \end{aligned}$$

Unexpected universality:

L. Cardelli, Gh. Păun, An universality result for a (mem)brane calculus based on mate/drip operations, In *Cellular Computing. Complexity Aspects* (M.A. Gutiérrez-Naranjo, Gh. Păun, M.J. Pérez-Jiménez, eds.), Fenix Editora, Sevilla, 2005, 75–94.

- Minimal parallelism: if at least a rule from a set/region **can** be used, then at least one **must** be used

Universality again: G. Ciobanu, Gh. Păun, The minimal parallelism is still universal (for P systems with symport/antiport rules), manuscript, 2005.

- Time-free/independent P systems [Cavaliere, Deufemia, Sburlan, 2005]
- Population P systems [Gheorghe and his group]
- Quantum, fuzzy, rough
- Sevilla carpet [Sevilla team]

## Nishida's membrane algorithms:

- candidate solutions in regions, processed locally (local sub-algorithms)
- better solutions go down
- static membrane structure – dynamical membrane structure
- two-phases algorithms

Excellent solutions for Travelling Salesman Problem (benchmark instances)

- rapid convergence
- good average and worst solutions (hence reliable method)
- in most cases, better solutions than simulated annealing

Still, many problems remains: check for other problems, compare with sub-algorithms, more membrane computing features, parallel implementations (no free lunch theorem)

**Recent:** L. Huang, N. Wang, J. Tao; G. Ciobanu, D. Zaharie; A. Leporati, D. Pagani

## Research topics, trends:

- more realistic classes (biology: asynchronous)
- more interesting classes (mathematics: small families)
- improving existing results (nr. of membranes, borderlines, etc)
- deterministic systems (D-catalytic = universal?)
- clock-free systems (no clock, sequential, etc.)
- non-crisp approaches (probabilistic, fuzzy, rough)
- complexity classes, separations, characterizations
- tissue P systems, neural P systems, population P systems
- quantum P systems
- dynamic systems approach (life as the goal)
- implementations/simulations
- connections with other domains (Petri nets, brane calculi, etc)
- applications (biology, linguistics, computer graphics, economics, etc)
- implications in computer science
- and many others

# Thank you!

...and please do not forget: <http://psystems.disco.unimib.it>