Nature Motivated Approaches to Computer Science – II.

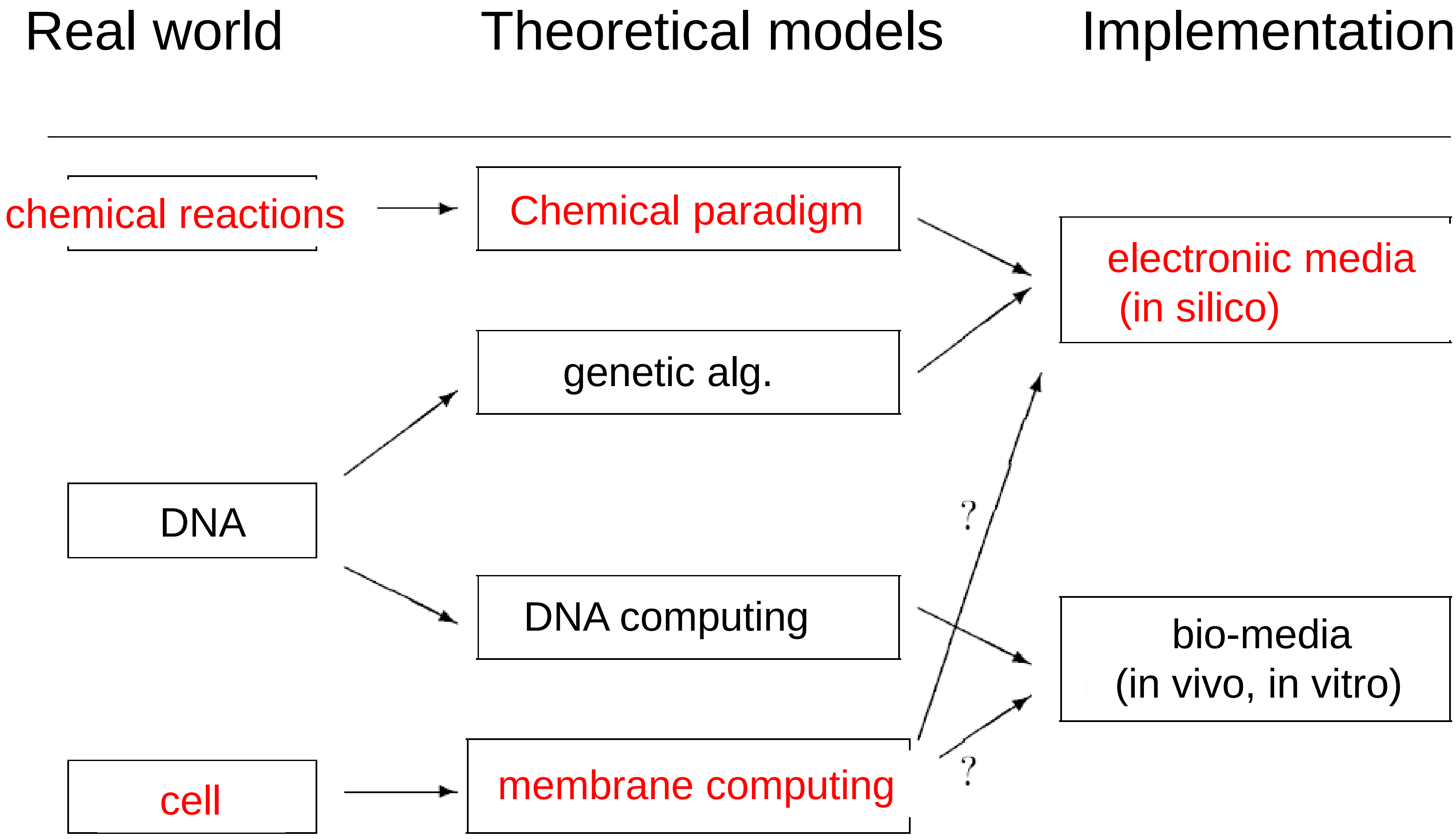
György Vaszil

University of Debrecen, Faculty of Informatics

Department of Computer Science

Potsdam, July 2017

Abstract models



Last week - The chemical model

- Symbolic **chemical solution** with abstract **molecules**, and rules describing **reactions** between them
- Molecules represent **data**, reactions represent **operations**
- **Brownian motion**, as execution model

→**Multisets** of symbols/objects + multiset **rewriting rules**

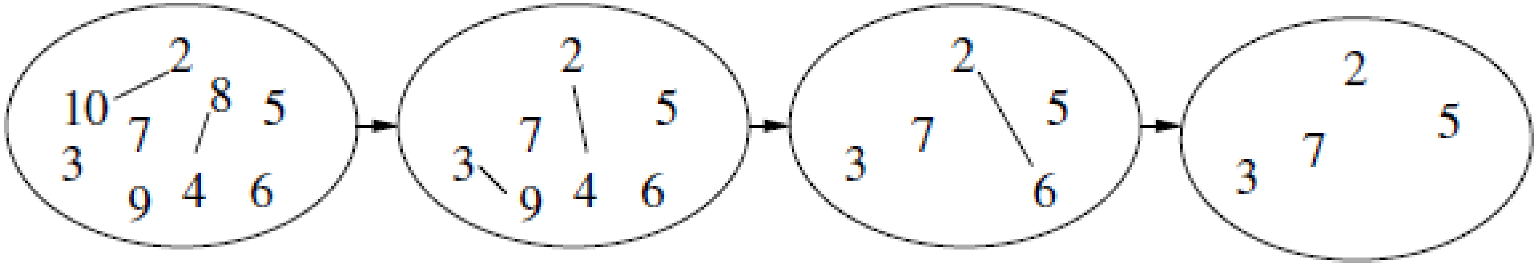
The chemical model/paradigm

- **Multisets** of symbols/objects + multiset **rewriting rules**

Abstract machine	Chemistry
Data	Molecule
Multiset	Solution
Parallelism/nondeterminism	Brownian motion
Computation	Reaction

Gamma example – Primes

$$
\begin{aligned}
primes(N) = \Gamma((R, A))(\{2 \ldots N\}) \ &\textbf{where} \\
R(x, y) &= multiple(x, y) \\
A(x, y) &= \{y\}
\end{aligned}
$$

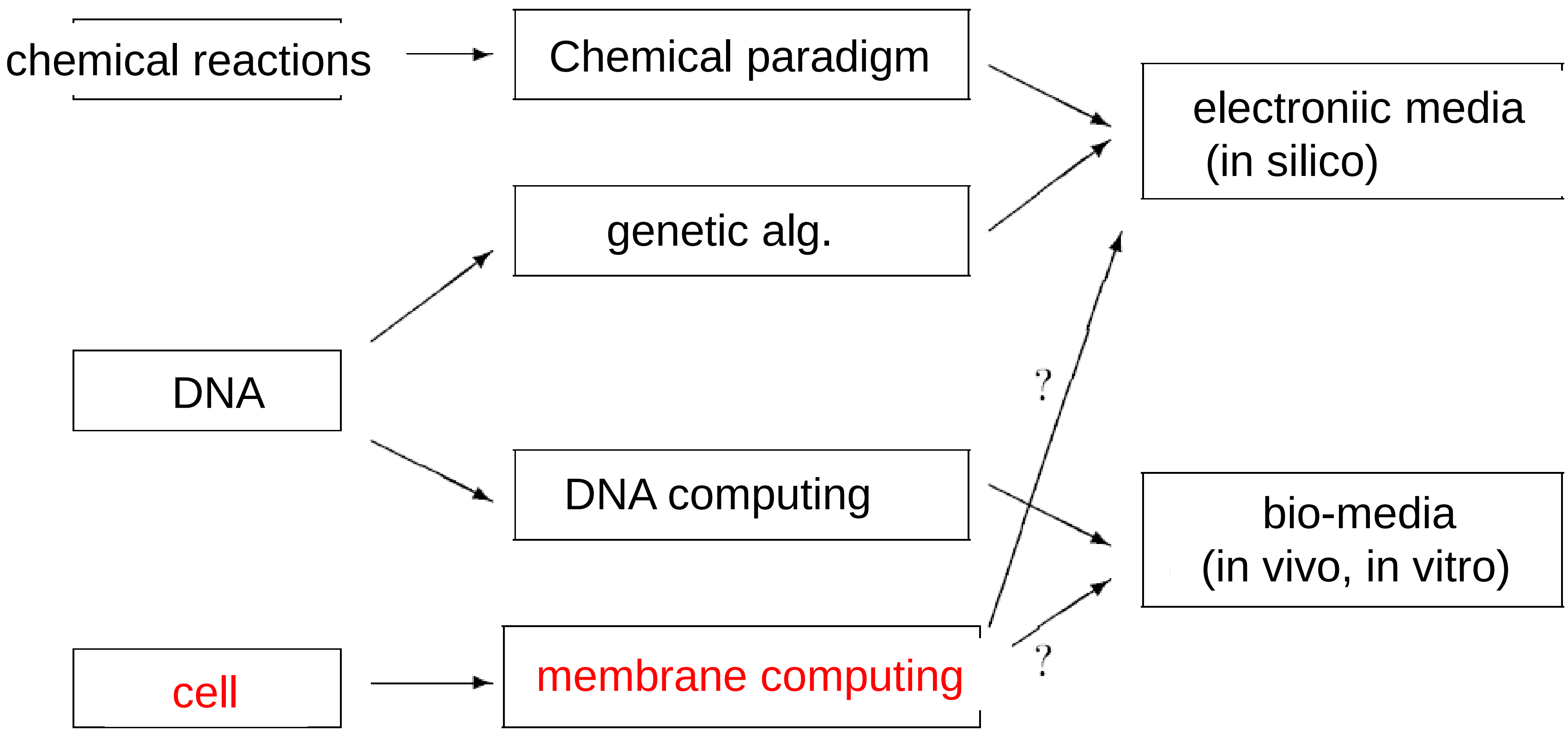


Properties of chemical programs

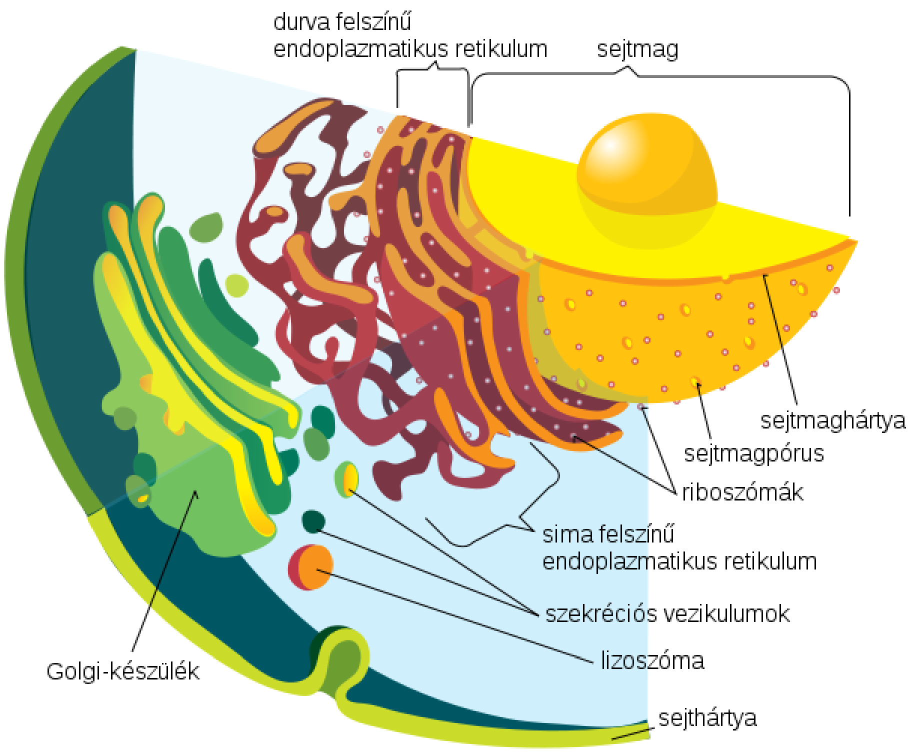
- M**ultiset** of abstract molecules in a
- **solution**, with
- **reactions** (operations):
  reaction **condition** + reaction **result**.
- **Sub-solutions**: „sub-regions” with their own reaction rules (priority, sequentiality).
- Program execution **ends**, if there are no applicable reactions.

→Natural, free from the forced sequentiality of the physical computer architecture

Real world | Theoretical models | Implementation



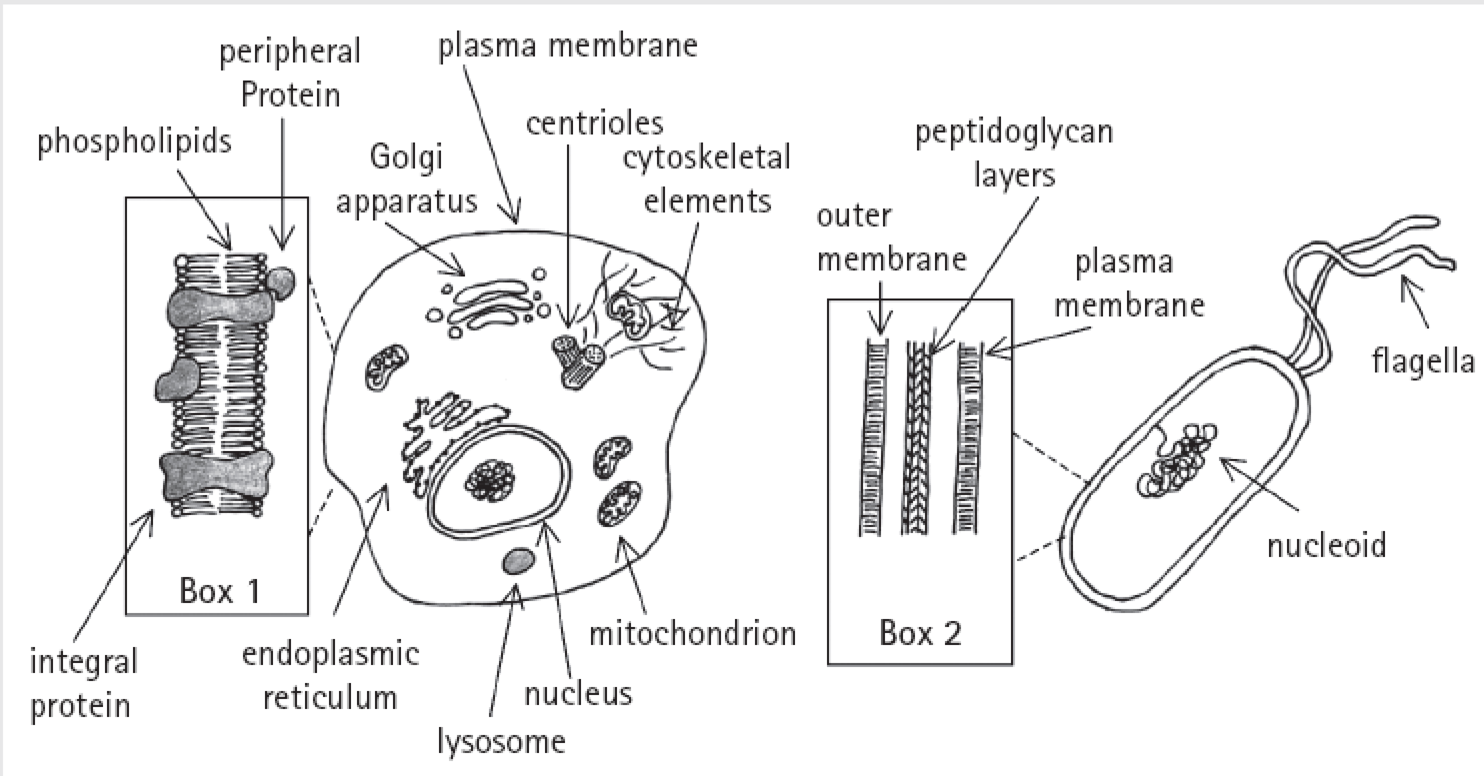
Membrane systems – The biochemical motivation



Cells contain regions

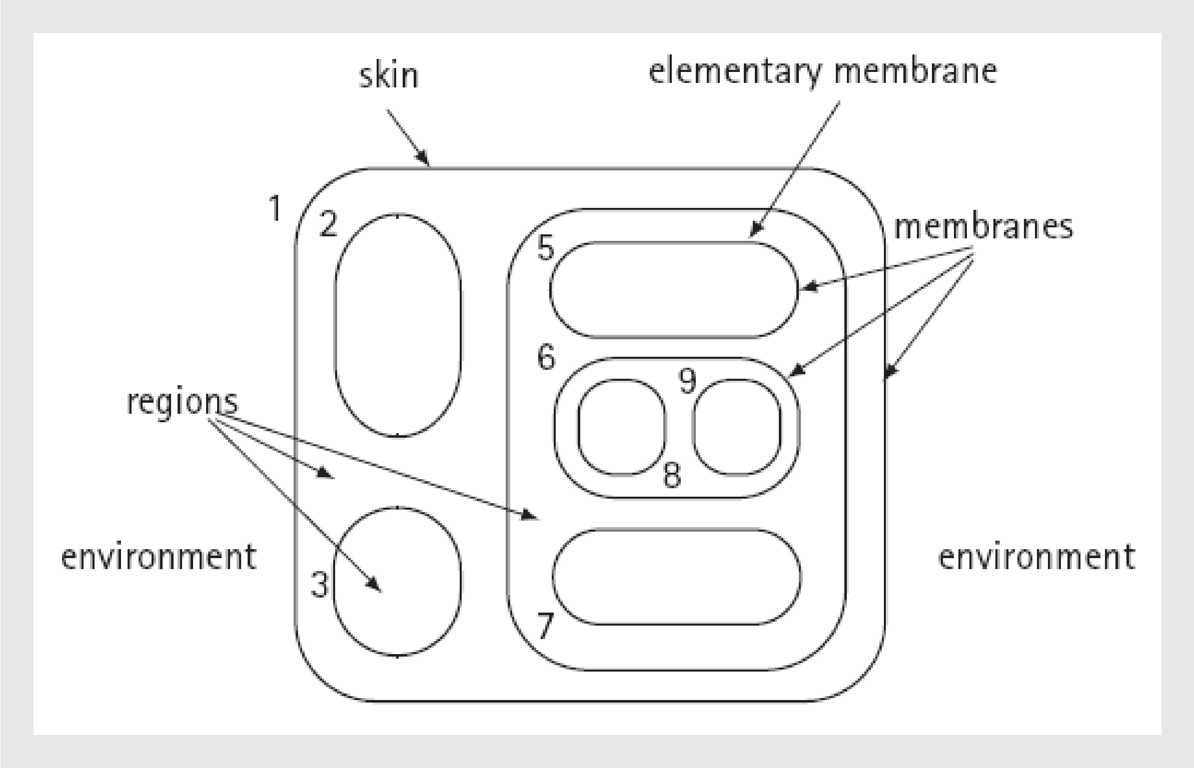
- Regions are enclosed by membranes
- Different regions have different biochemical processes inside
- Membranes also regulate traffic between the regions

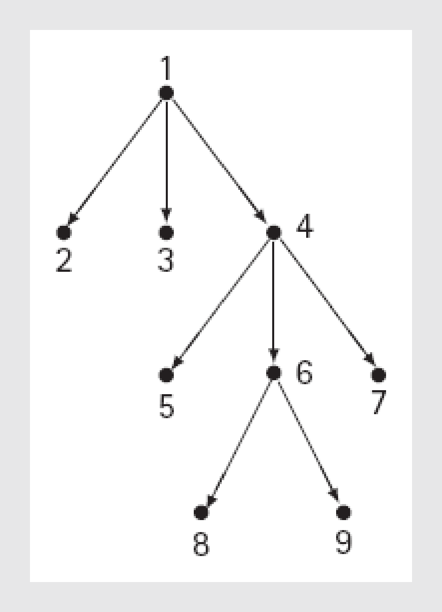
Membrane systems



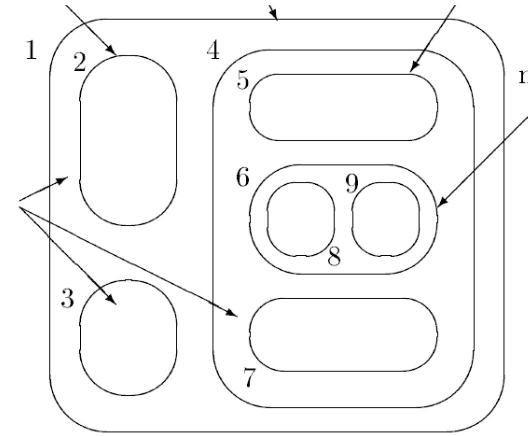
Membrane systems, a membrane structure

A hierarchical arrangement of regions where multisets of objects evolve according to given evolutionary rules





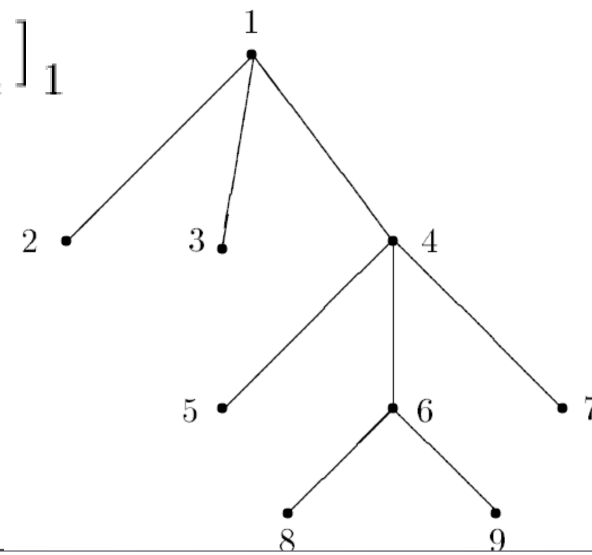
The membrane structure



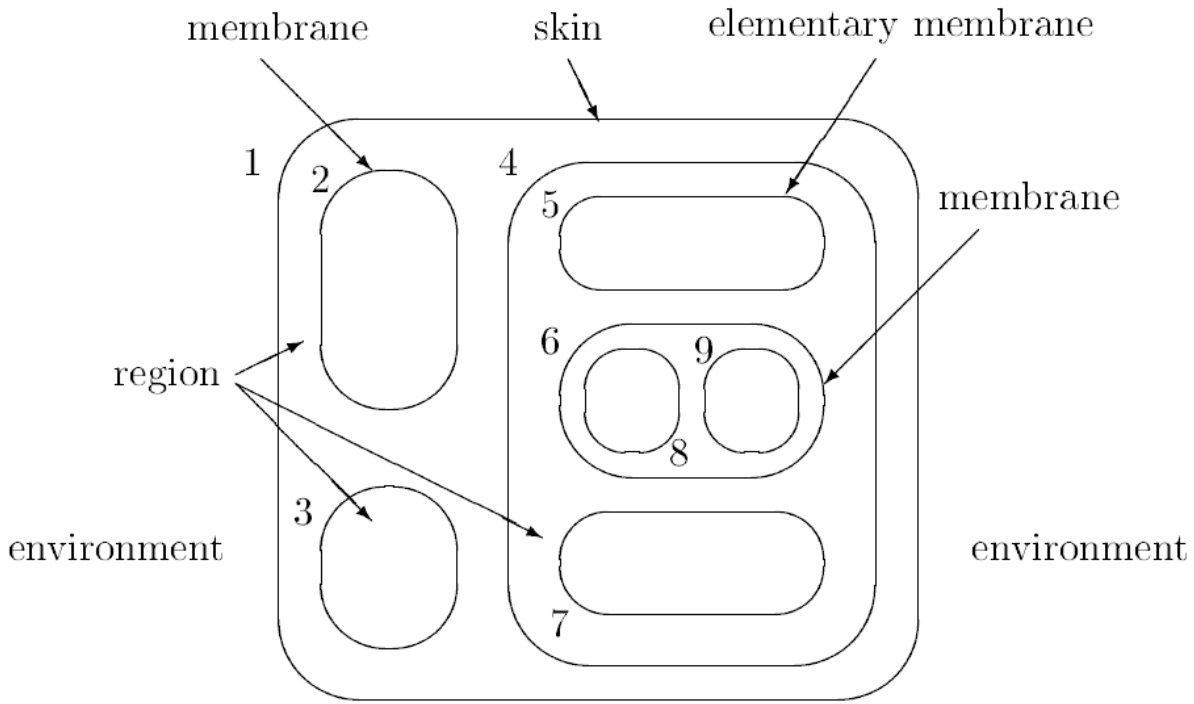
Can be described by

- a tree, or a
- string of parentheses

$$[_1 \; [_2 \;]_2 \; [_3 \;]_3 \; [_4 \; [_5 \;]_5 \; [_6 \; [_8 \;]_8 \; [_9 \;]_9 \;]_6 \; [_7 \;]_7 \;]_4 \;]_1$$



The membrane structure

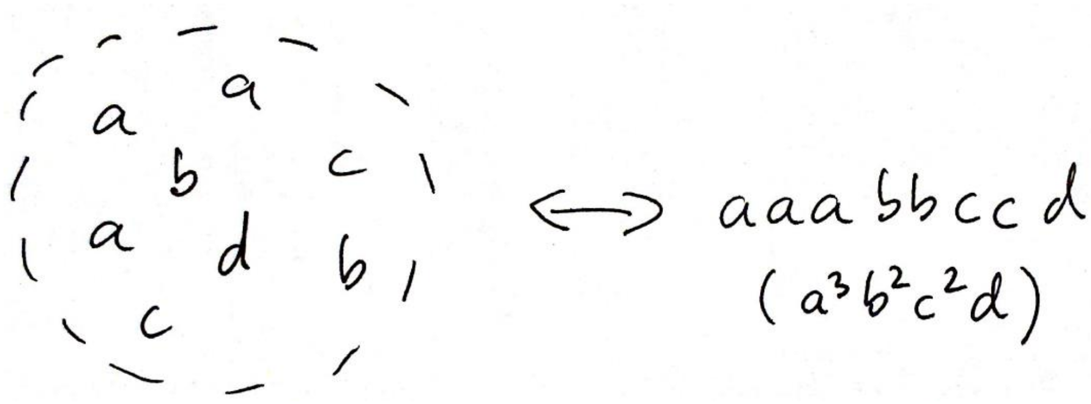


- membrane <--> enclosed region
- outer (skin) membrane, environment
- “inside”, “outside”

The objects

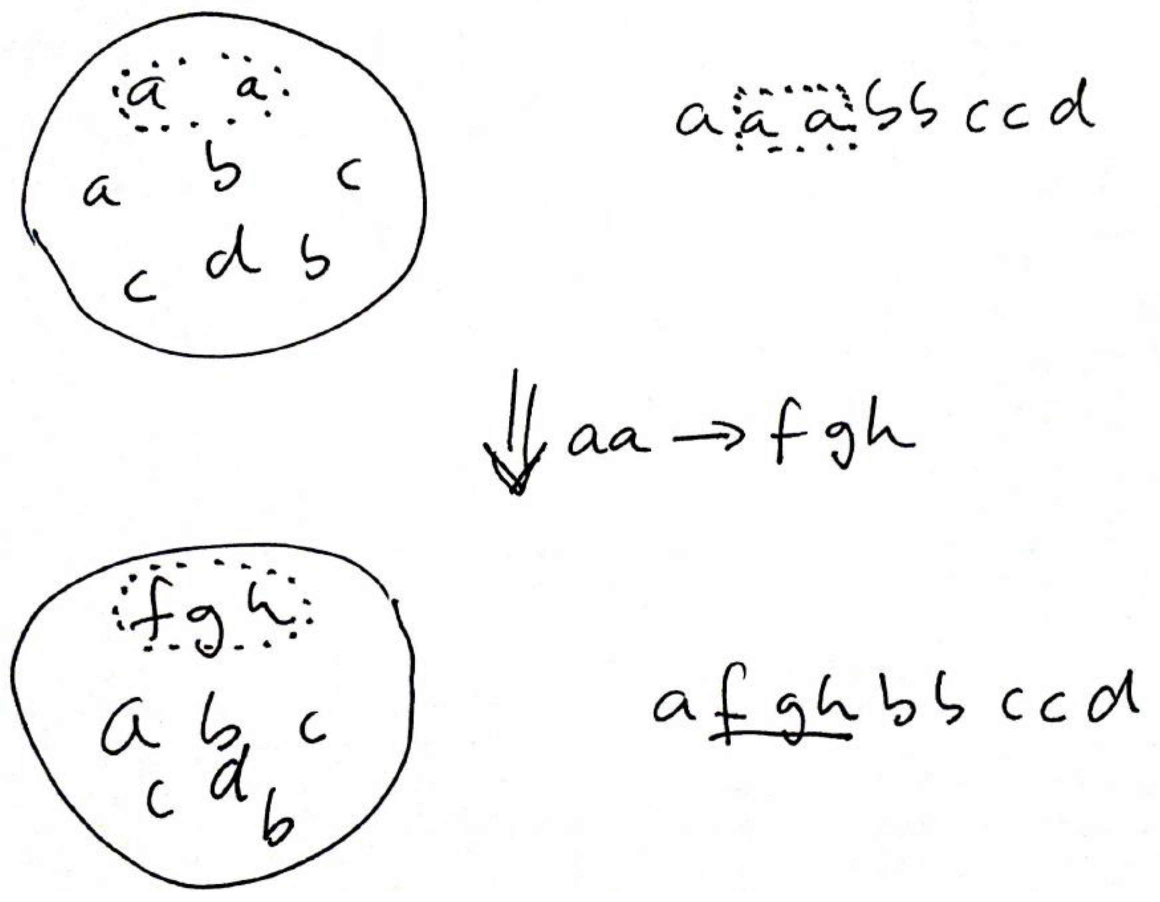
The regions (membranes) contain **multisets of objects**

- object: **symbol** from a finite alphabet
- mutisets are represented be **strings** over the object alphabet



The rules

Applying the multiset rewriting rule $aa \rightarrow fgh$



Maximal parallel rule application

An example:

w_i : $aaabb$

R_i : $\underbrace{aa \rightarrow cc}_{\hat{=}\, r_1}$, $\underbrace{ab \rightarrow dd}_{\hat{=}\, r_2}$

There are two possibilities:

1. $aaabb \Rightarrow ccddb$

2. $aaabb \Rightarrow adddd$

Membrane systems, multiset rewriting rules

$$a^2bc^3 \rightarrow ba^2c(da, out)(ca, in)$$

The rules

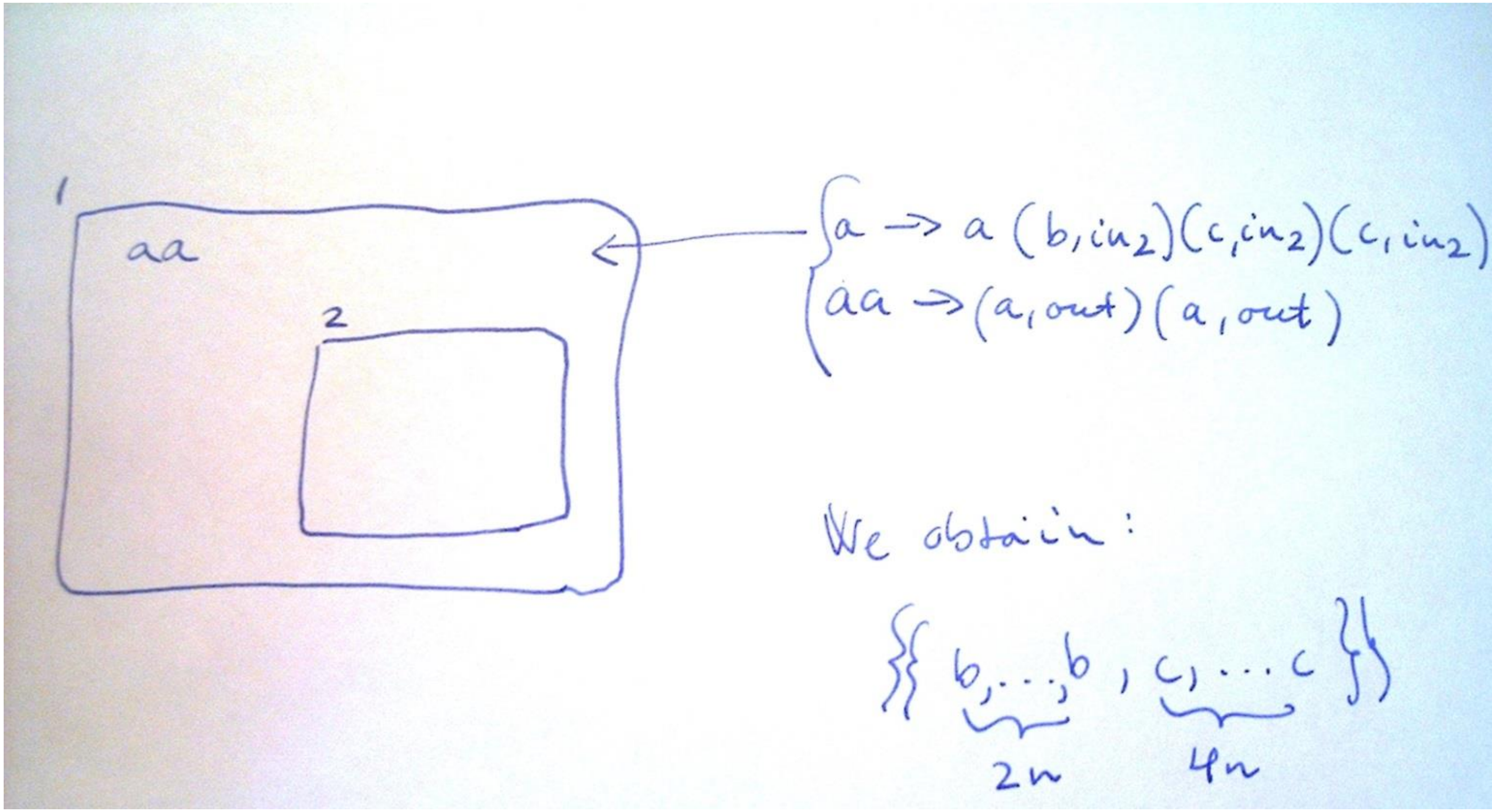
- change the objects
- move the objects between neighboring regions

The rules are applied

- in maximal parallel way
- In a synchronized manner

P systems – [Paun 2000 (1998)]

Example



The computation

- Start in an **initial configuration**
- A computational **step**: Apply the rules in a **maximal parallel** way in all regions
- **Repeat** the rule application step as long as possible (until a **final configuration** is reached)
- The **result** of the computation is given by the **multiplicities** of certain objects in certain regions.

Example

$$[\ a, a\ [\ \ ]_2\ ]_1 \Rightarrow^{a \to a(b,in_2)(c,in_2)(c,in_2)} [\ a, a\ [\ b, c, c,\ \ b, c, c\ ]_2\ ]_1$$

$$\Rightarrow^{a \to a(b,in_2)(c,in_2)(c,in_2)} \ldots \Rightarrow^{a \to a(b,in_2)(c,in_2)(c,in_2)}$$

$$[\ a, a\ [\ \underbrace{b, \ldots, b}_{2n}, \underbrace{c, \ldots, c}_{4n}\ ]_2\ ]_1 \Rightarrow^{aa \to (a,out)(a,out)} [\ \ [\ \underbrace{b, \ldots, b}_{2n}, \underbrace{c, \ldots, c}_{4n}\ ]_2\ ]_1$$

A language of **multisets** - the contents of a specified region:

$$L = \{\ \ \{\{\underbrace{b, \ldots, b}_{2n}, \underbrace{c, \ldots, c}_{4n}\}\}\ \mid n \geq 0\}$$

Some very basic results about this very basic setup...

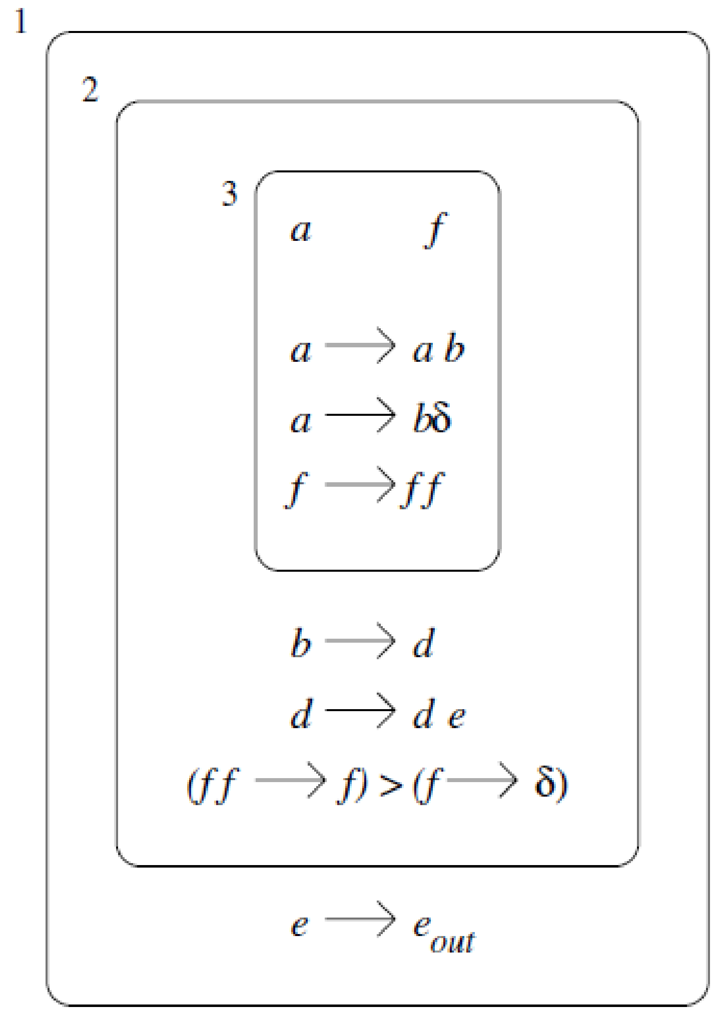
- Having **two membranes** is sufficient
- Systems with rules having **one object** on the left-hand side are **weak**: they compute the length sets of **context-free languages**
- Systems with rules having at least **two objects** on the left-hand side can compute any **recursively enumerable set** of numbers (compute “anything”)

[Paun 2002]

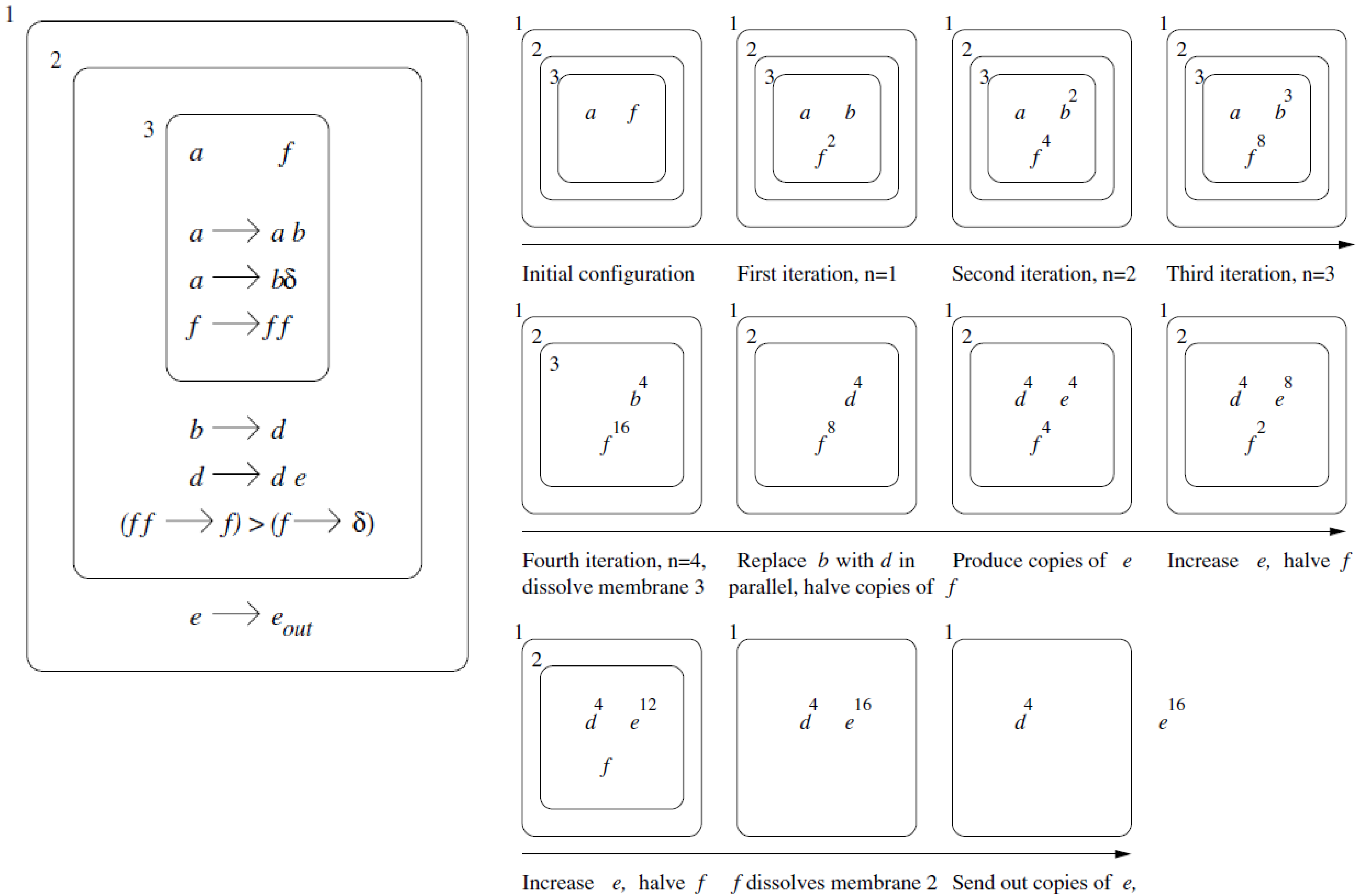
There are many more features that can be added

- For example:

Computing/generating square numbers using **membrane division**



The computation



There are many interesting results, for example:

- Polynomial solutions to several NP complete problems
  - formula satisfiabilty
  - Hamiltonian path
  - discrete logarithm
  - …

Outline

Our goal: Using P systems to describe string languages – construct automata-like membrane systems

- Membrane systems (P systems) with communication rules only, accepting P systems
- P automata
  - The computational power of P automata
  - P automata over infinite alphabets

Automata-like systems

- state **transitions**
- reading an **input**
- working with **tapes/counters**

(What are counter automata and why are they interesting?)

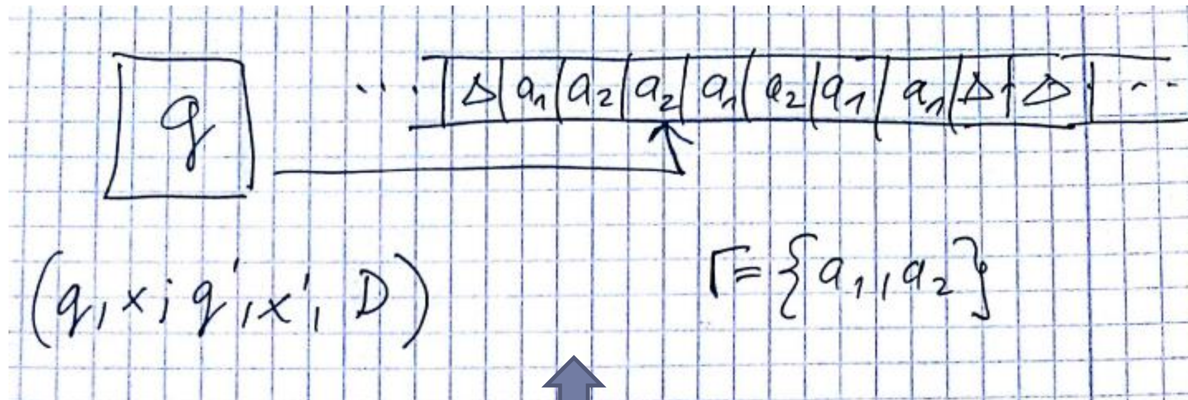
(What are counter automata and why are they interesting?)

1. Turing machines can compute “anything” with finite control, and a tape containing a string of symbols
2. The tape can be simulated by two stacks
3. A stack of symbols can be simulated by two counters storing numbers
4. Four counters can be simulated by two counters

→ Anything can be computed by counter machines (register machines)

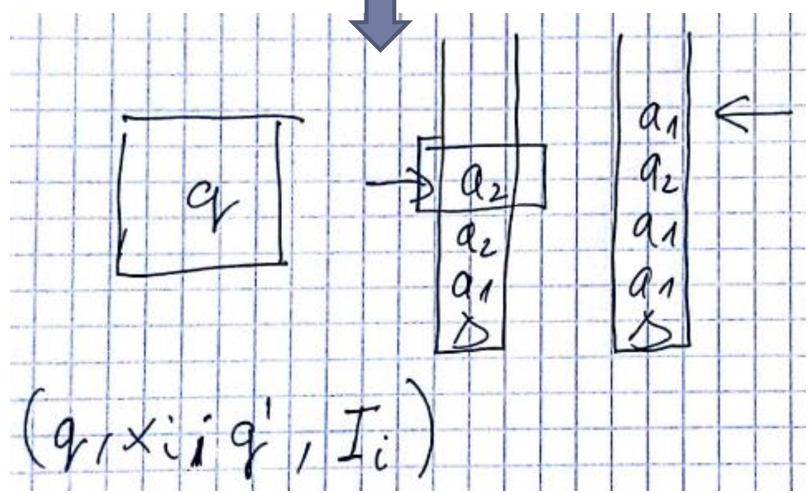
(What are counter automata and why are they interesting?)

A Turing machine **tape** can be simulated by **two stacks:**



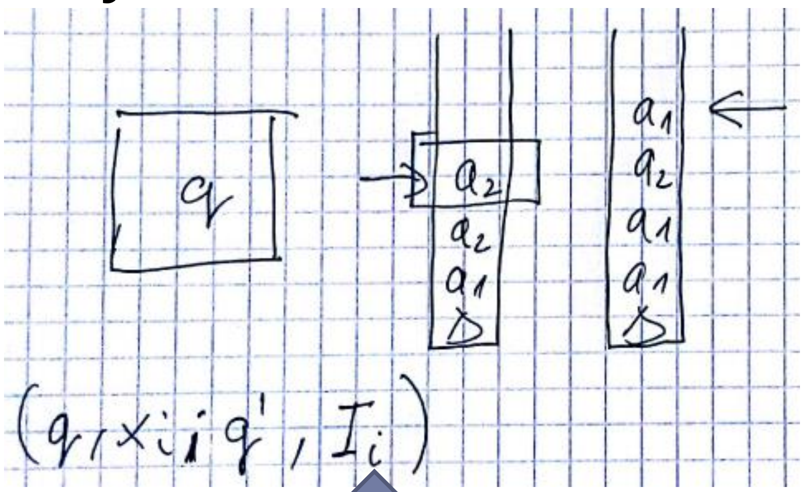
- rewrite
- move the head

- pop and push the new symbol
- pop from one stack, push on the other



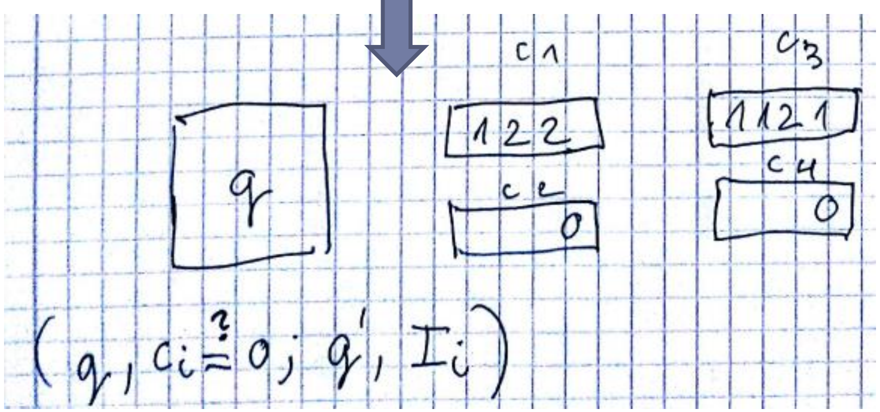
(What are counter automata and why are they interesting?)

A **stack** can be simulated by **two counters:**



- *a1a2a2* in the stack
- pop , push

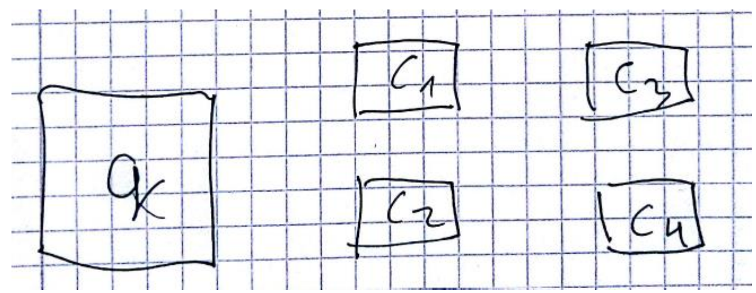
- 122 in the counter
- pop: divide by 10
- push *ax*: multiply by 10 and add x



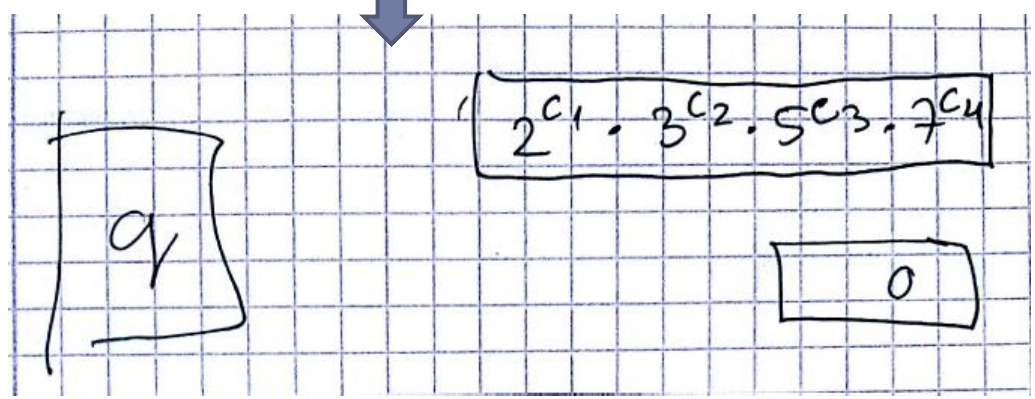
(What are counter automata and why are they interesting?)

Any number of **counters** can be simulated by **two counters:**

- increment *ci*
- decrement *ci*
- does *ci* contain 0?



- multiply *c1* with the ith prime
- divide *c1* by the ith prime
- is *c1* divisible by the ith prime?

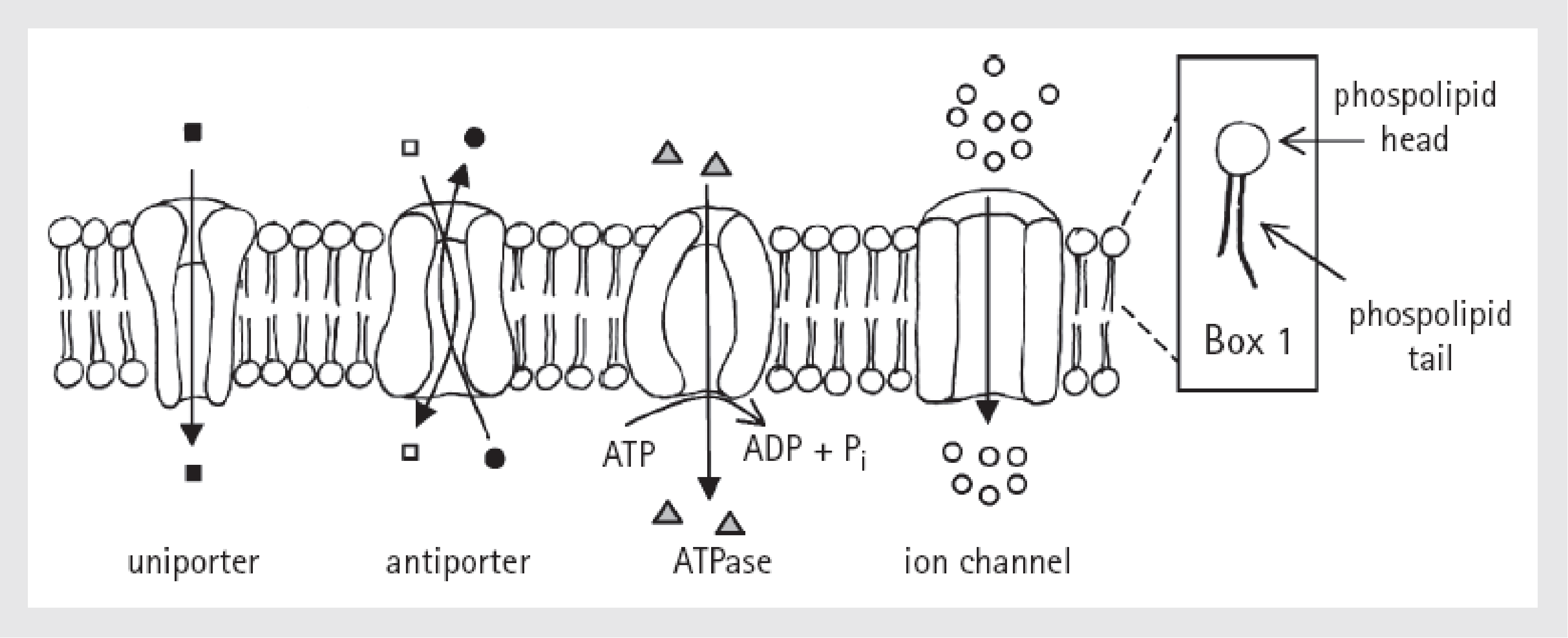


Automata-like membrane systems

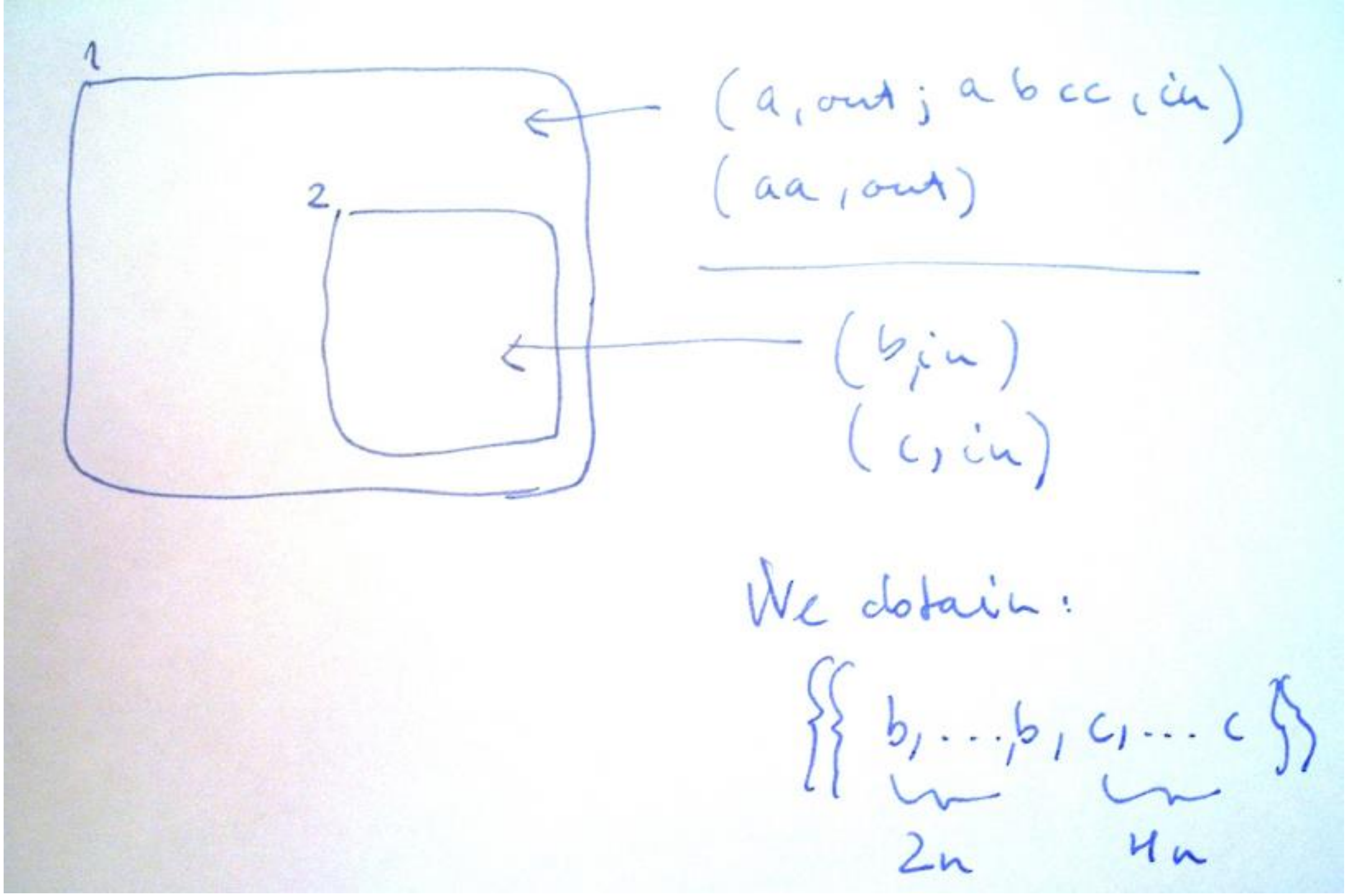
- state **transitions**
- reading an **input**
  $\longrightarrow$ operating in an "**environment**"
- working with **tapes/counters**
  $\longrightarrow$ using resources represented by **multisets**

Membrane systems with communication only

Symport/antiport systems



Example, in communication with the environment



Example, in communication with the environment

(a, out ; a b c c , in)

$$a, a, b, c, a, b, b, c, c, c, \ldots \quad [\; a, a \;[\;\;]_2\;]_1 \Rightarrow$$

$$a, a, \;, \;, a, b, \;, \;, \;, c, \ldots \quad [\; a, b, c, c, \;\; a, b, c, c \;[\;\;]_2\;]_1 \Rightarrow$$

$$\ldots \quad [\; a, b, c, c, \;\; a, b, c, c \;[\; b, c, c, \;\; b, c, c \;]_2\;]_1$$

$$\Rightarrow \ldots \Rightarrow$$

(aa , out)

$$\ldots \quad [\; a, b, c, c, a, b, c, c \;[\; \underbrace{b, \ldots, b}_{2n}, \underbrace{c, \ldots, c}_{4n} \;]_2\;]_1 \Rightarrow \qquad [\;\;[\; \underbrace{b, \ldots, b}_{2n+1}, \underbrace{c, \ldots, c}_{4n+1} \;]_2\;]_1$$

$$L = \{ \;\; \{\{\underbrace{b, \ldots, b}_{2n}, \underbrace{c, \ldots, c}_{4n}\}\} \;\mid n \geq 0\}$$

Symport/antiport systems – The rules

Symport/antiport rules, possibly with promoters:

$$(x, out; y, in)|_Z, \text{ with } Z \in \{z, \neg z\}, \;\; x, y, z \in V^*$$

- Maximal **parallel** rule application
- **out** - leave **to the** upper, **parent** region, **in** - enter **from the parent** region
- The parent of the **skin** region is the **environment**

(Accepting) symport/antiport systems

$$\Pi = (V, \mu, E, w_1, \ldots, w_n, R_1, \ldots, R_n, F, i_{in})$$

V	–	a finite **alphabet of objects**
μ	–	a **membrane** structure
$E \subseteq V$	–	a set of objects, the ones which can be found in the **environment**
$w_i \in V^*$	–	the **initial contents** of region i
R_i	–	sets of **symport/antiport rules** associated to region i
F	–	a set of **final configurations**
i_{in}	–	the label of the **input membrane**, if $i = 0$ the input is read from the **environment**

The transitions

$$\Pi = (V, \mu, E, w_1, \ldots, w_n, R_1, \ldots, R_n, F, i_{in})$$

A configuration:

$(u_0, u_1, \ldots, u_n)$, $u_i \in V^*$, the **contents** of the environment and the n **regions** of Π

The transition mapping:

$$\delta : V^* \times (V^*)^{n+1} \to (V^*)^n,$$
$$\delta(u, (u_0, u_1, \ldots, u_n)) \ni (u'_0, u'_1, \ldots, u'_n)$$

the **multiset** entering the **skin membrane** — the **configuration** — the **new configuration**

Symport/antiport systems and counter automata

the **multiplicity** of the object a_i	$\longleftrightarrow$	the value of **counter** i
the **presence** of the object q	$\longleftrightarrow$	being in the **internal state** q

Symport/antiport systems and counter automata

$(q, out; q'a_i, in) \quad \longleftrightarrow$

- change the **state** $q \longrightarrow q'$
- **increase** the value of **counter** i

$(qa_i, out; q', in) \quad \longleftrightarrow$

- change the **state** $q \longrightarrow q'$
- **decrease** the value of **counter** i

Symport/antiport systems and counter automata

$(q, out; q_1 q_2, in)$
$(q_1 a_i, out$ *; q1', in)*
$(q_2, out; q_3, in)$
$(q_1 q_3, out; q', in)$
(q1'q3, out; q", in)

$\longleftrightarrow$

- change the **state** $q \longrightarrow q'$
- check the **emptiness** of **counter** i
- if *ci* is not empty: *q*→*q"*

- $(q_f, out; f, in)$
- **no rule** for f

$\longleftrightarrow$ q_f is a **final** state

Symport/antiport systems and counter automata

$$N(\Pi) = \{k \in \mathbb{N} \mid c_0 = (w_0, w_1, \ldots, w_n), \text{ with } |w_{in}|_{a_i} = k, \text{ and there is } c_t \in F \text{ and a sequence } c_i \text{ with } \delta(u_{i+1}, c_i) = c_{i+1} \text{ for all } 0 \leq i \leq t-1\}$$

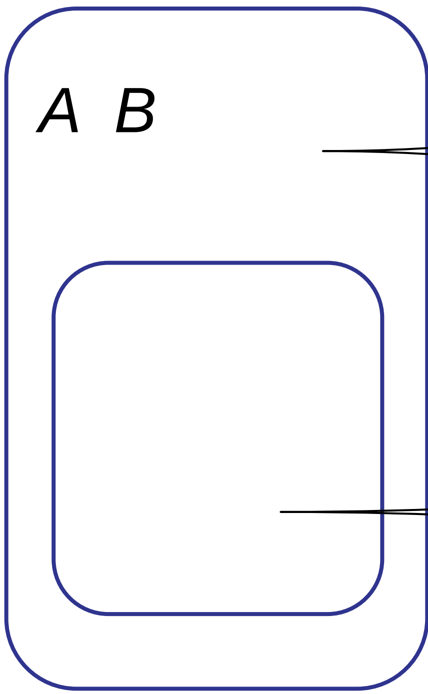
- c_0 is the **initial configuration**
- $a_i \in V$ is the object corresponding to the **input counter**
- F is the set of **halting configurations**

From multiplicities (numbers) to sequences (strings)...

Consider the **multiset sequences** accepted by antiport P systems

Accepted multiset sequences - Example

initial configuration:



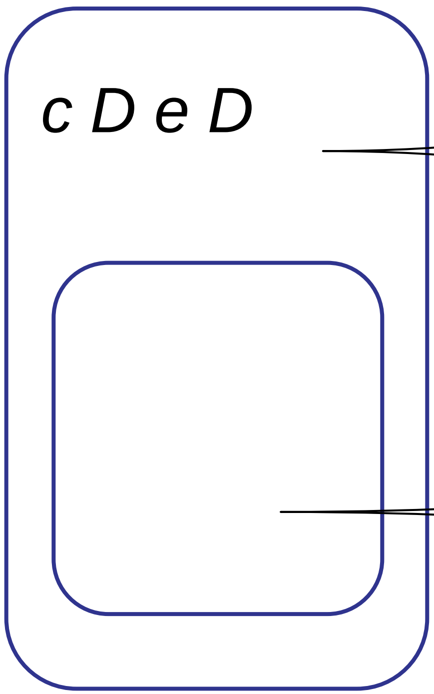
rules:

(A, out; cD, in)
(B, out; eD, in)
(D, out; F, in)

(F, in)

Accepted multiset sequences - Example

configuration: rules:

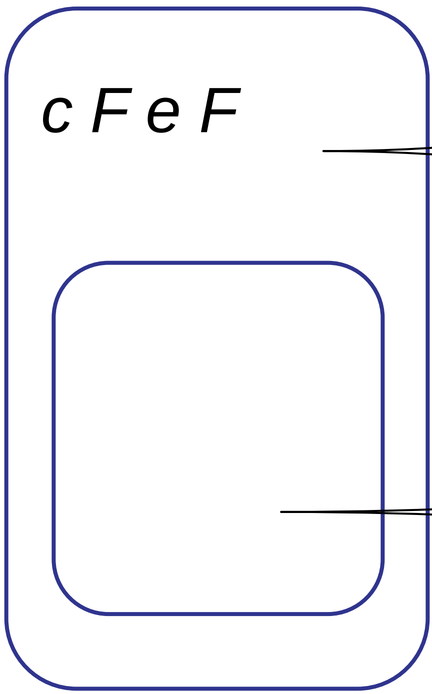


(A, out; cD, in)
(B, out; eD, in)
(D, out; F, in)

(F, in)

Accepted multiset sequences - Example

configuration: rules:

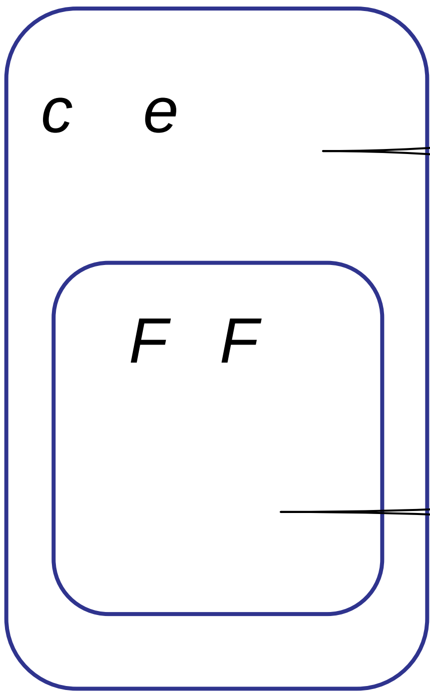


(A, out; cD, in)
(B, out; eD, in)
(D, out; F, in)

(F, in)

Accepted multiset sequences - Example

configuration: rules:

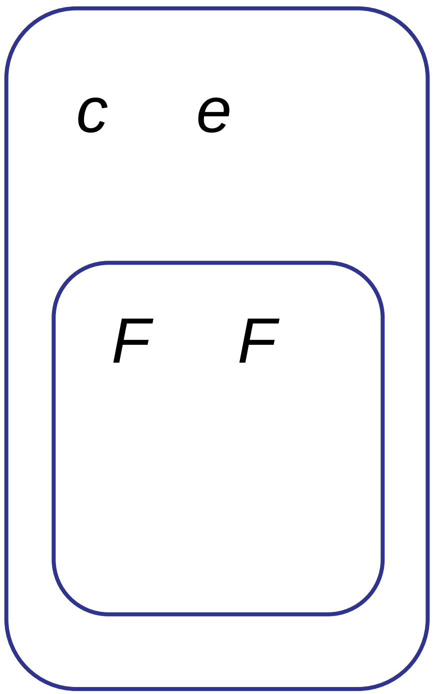


(A, out; cD, in)
(B, out; eD, in)
(D, out; F, in)

(F, in)

Accepted multiset sequences - Example

final
configuration:



The (set of) accepted **multiset sequence(s)**:

$\{ \ \{c,e,D,D\}\{F,F\} \ \}$

Accepting P systems – What we have so far…

- A **P system** in an **environment**
- Given an **initial configuration**
- A **sequence of multisets** is read from the environment during the computation
- The multiset sequence is **accepted** if the computation ends in a **halting configuration**

Characterizing string languages/1

How to **map** the accepted **multiset sequences** to accepted **strings**?

1. **Analyzing** P systems, **extended** P automata

   - **Terminals** and **nonterminals** – only terminal symbols are taken into account
   - The **input multisets** are **mapped** to **sets of strings** which can be **constructed** from the **terminals**

Characterizing string languages/1

Analyzing P systems:

- a set of **terminal objects** $T \subseteq V$
- $i_{in} = 0$, the input is read from the **environment**
- F is the set of **halting configurations**

[R. Freund, M. Oswald: A short note on analysing P systems. *Bulletin of the EATCS*, 78 (October 2002), 231–236.]

Analyzing P systems

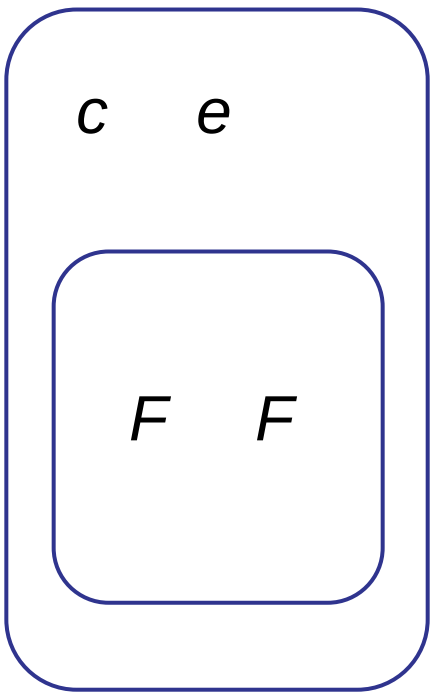
$$\Pi = (V, \mu, E, w_1, \ldots, w_n, R_1, \ldots, R_n, F, i_{in})$$

$$L(\Pi) \;=\; \bigcup str_T(u_1) \cdot str_T(u_2) \cdot \ldots \cdot str_T(u_t)$$

for all $c_t \in F$ and sequence c_i with $\delta(u_{i+1}, c_i) = c_{i+1}$ $0 \leq i \leq t-1$,

- c_0 is the **initial** configuration
- $str_T(u) \subseteq T^*$ is the set of **terminal strings** corresponding to the multiset $u \in V^*$
- F is the set of **halting** configurations

The previous example:



The (set of) accepted **multiset sequence(s)**:

$$\{\ \{c,e,D,D\}\{F,F\}\ \}$$

If the set of **terminal** symbols is $T=\{e,c\}$, then the **accepted strings** are:

$$\boxed{\{\ ce,\ ec\ \}}$$

The power of analyzing P systems

Any **recursively enumerable** language can be **accepted** by an analyzing P system having **one membrane**.

[R. Freund, M. Oswald: A short note on analysing P systems. *Bulletin of the EATCS*, 78 (October 2002), 231–236.]

The proof idea

1. **Read the input** object sequence
2. **Create a numerical encoding** of the object sequence in the "input counter"
3. Simulate the computation of a **counter machine**

The **terminal-nonterminal** distinction is **essential**: nonterminals provide the "**workspace**" for the computation.

The numerical encoding

$$\Sigma = \{a_1, \ldots, a_{z-1}\}$$

$$\begin{array}{ccc}
\underline{\text{symbols}} & & \underline{z - \text{ary digits}} \\
 & & \\
a_1 & \longleftrightarrow & (1) \\
a_2 & \longleftrightarrow & (2) \\
 & \vdots & \\
a_{z-1} & \longleftrightarrow & (z-1)
\end{array}$$

The numerical encoding

$$w = a_{i_1} \dots a_{i_k} \in \Sigma^* \quad \longleftrightarrow \quad code(w) = (i_1) \dots (i_k) \in \mathbb{N}$$

The encoding of an input word is created step by step, with each new symbol a_i:

$$code(wa_i) = code(w) \cdot z + i$$

Simple arithmetic operations, they can be done by the counter machine.

The proof idea again

1. **Read the input** object sequence
2. **Create a numerical encoding** of the object sequence in the "input counter"
3. Simulate the computation of a **counter machine**

The **terminal-nonterminal** distinction is **essential**: nonterminals provide the "**workspace**" for the computation.

What is the **role of the different features**? What are those which are **necessary for reaching universal power**? Is it possible to **restrict the power** of the system in any “interesting” way?

Finite (extended) P automata

$$\Pi = (V, [\ ], w_1, R_1, F)$$

- a set of **terminal objects** $T \subseteq V$
- **rules of two types**:
  1. $(q, out; pa, in)$ with $q, p \in V - T,\ a \in T$
  2. $(pa, out; r, in)$ with $p, r \in V - T,\ a \in T$

     For any rule of type 1. with $p \in V - T$, there is only one rule of type 2. with the same $p \in V - T$
- F contains **halting** configurations with a “final” nonterminal inside the system

[R. Freund, M. Oswald, L. Staiger: Omega-P automata with communication rules. *Lecture Notes in Computer Sci.*, 2933 (2004), 203–217]

Finite (extended) P automata

$$\Pi = (V, [\ ], w_1, R_1, F)$$

$$L(\Pi) \;=\; \bigcup str_T(u_1) \cdot str_T(u_2) \cdot \ldots \cdot str_T(u_t)$$

for all $c_t \in F$ and sequence c_i with $\delta(u_{i+1}, c_i) = c_{i+1}$ $0 \leq i \leq t-1$,

L is regular if and only if it is accepted by a finite P automaton with antiport rules.

Exponential space symport/antiport acceptors

$$\Pi = (V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, F)$$

- a set of **terminal objects** $T \subseteq V$ containing a distinguished symbol \$
- the input is read from the **environment**
- **rules of four types** in the **skin membrane**:
  1. $(u, out; v, in)$ with $u, v \in (V - T)^*$, $|u| \geq |v|$
  2. $(u, out; va, in)$ with $u, v \in (V - T)^*$, $|u| \geq |v|$, $a \in T$
  3. $(u, out; v, in)|_a$ with $u, v \in (V - T)^*$
  4. for every $a \in T$, $(a, out; v, in)$
- **rules** of the form $(u, out; v, in)$ with $u, v \in (V - T)^*$ in the **other regions**

Exponential space symport/antiport acceptors

$$\Pi = (V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, F)$$

$$L(\Pi) \;=\; \bigcup str_T(u_1) \cdot str_T(u_2) \cdot \ldots \cdot str_T(\bar{u}_t) \in T^*$$

for all $c_t \in F$ and sequence c_i with $\delta(u_{i+1}, c_i) = c_{i+1},\; 0 \leq i \leq t-1,\;\; \$ \in u_t,\; \bar{u}_t = u_t - \$$

- c_0 is the **initial** configuration
- $str_T(u) \in T^*$ is the set of **terminal strings** corresponding to the multiset $u \in V^*$
- F is the set of **halting configurations**

The power of exponential space symport/antiport acceptors

A **language** L **is accepted** by an exponential-space symport/antiport acceptor **if and only if** L **is context-sensitive**.

A **language** L **is regular** if and only if it can be accepted by an exponential-space symport/antiport acceptor **using only rules of type 1. and 2**.

1. $(u, out; v, in)$ with $u, v \in (V - T)^*$, $|u| \geq |v|$
2. $(u, out; va, in)$ with $u, v \in (V - T)^*$, $|u| \geq |v|$, $a \in T$

[O.H. Ibarra, Gh. Paun: Characterization of context-sensitive languages and other language classes in terms of symport/antiport P systems. *Theoretical Computer Sci.*, 358 (2006), 88–103]

Characterizing string languages/2

How to **map** the accepted **multiset sequences** to accepted **strings**?

2. P automata:

   - **No distinction** between **terminals** and **nonterminals**
   - The **input multisets** can be **mapped** to (sets of) **strings** using **any** (nonerasing) **mapping**.

   - (**Sequential rule application** is also considered.)

P automata

- An **antiport P system** in an **environment** from where the **input** is read
- Given an **initial configuration** and a set of final **(accepting) configurations**
- A **sequence of multisets** is read from the environment during the computation
- The multiset sequence is **accepted** if the computation ends in an **accepting configuration**

[E. Csuhaj-Varju, Gy. Vaszil: P automata or purely communicating accepting P systems. *Lecture Notes in Computer Sci.*, 2597 (2003), 219–233]

P automaton

A P automaton is

$$\Pi = (V, \mu, P_1, \ldots, P_n, c_0, \mathcal{F})$$

with

- object alphabet
- membrane structure
- rules corresponding to the regions
- initial configuration $c_0 = (w_1, \ldots, w_n)$, $w_i \in V^*$
- set of accepting configuratior $E_1 \times \ldots \times E_n,\ E_i \subseteq V^*$ with E_i being finite, or $E_i = V^*$

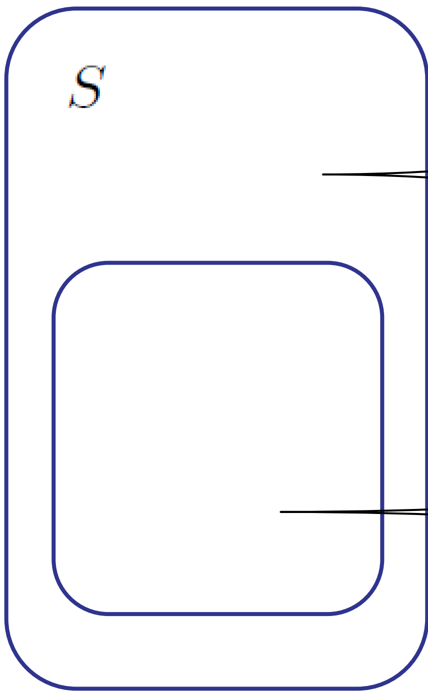
Examples...

- Variants of P automata for regular languages – different ways of mapping multiset sequences to strngs

P automaton – An example

Given a **regular grammar** with: $S \to aA, A \to bS, S \to \varepsilon$

initial configuration:



rules:

$(S, out; aA, in)$

$(A, out; bS, in)$

$(A, out; bF, in)$

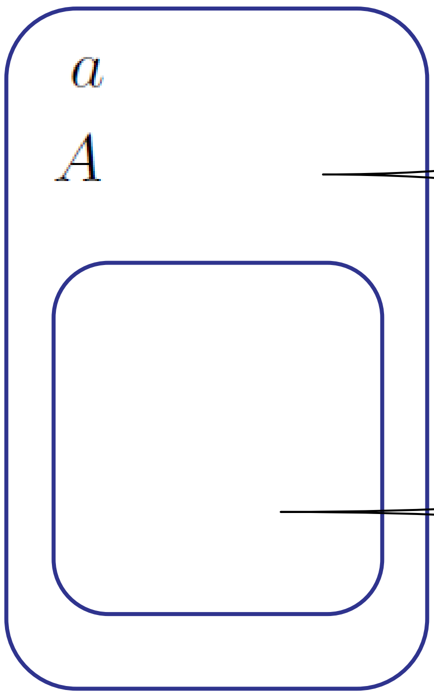
(F, in)

final configuration: F is in the region

P automaton – An example

Given a **regular grammar** with: $S \to aA, A \to bS, S \to \varepsilon$

configuration: rules:



$(S, out; aA, in)$

$(A, out; bS, in)$

$(A, out; bF, in)$

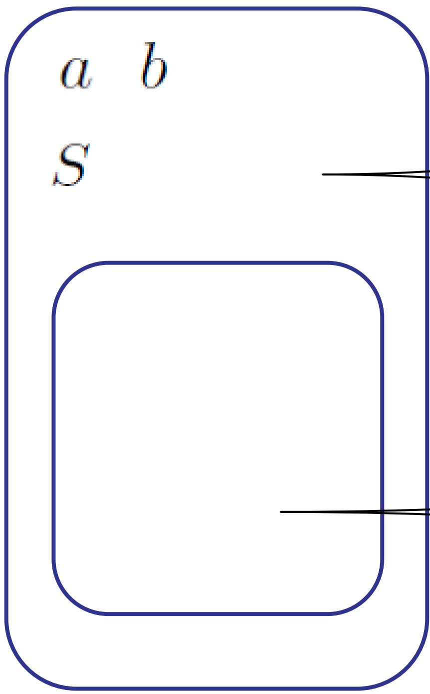
(F, in)

final configuration: F is in the region

P automaton – An example

Given a **regular grammar** with: $S \to aA, A \to bS, S \to \varepsilon$

configuration: rules:



$(S, out; aA, in)$

$(A, out; bS, in)$

$(A, out; bF, in)$

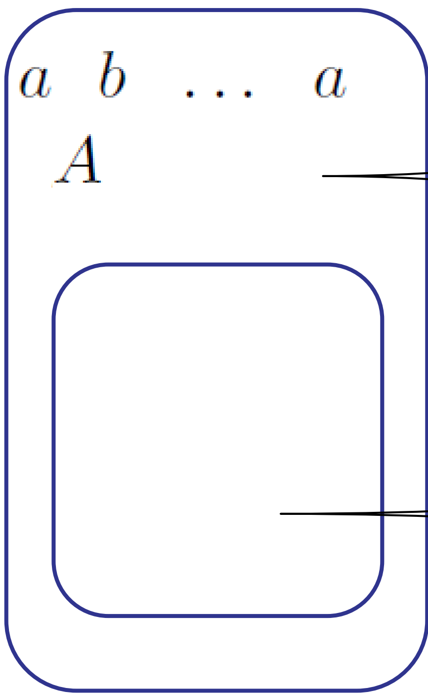
(F, in)

final configuration: F is in the region

P automaton – An example

Given a **regular grammar** with: $S \to aA, A \to bS, S \to \varepsilon$

configuration: rules:



$(S, out; aA, in)$

$(A, out; bS, in)$

$(A, out; bF, in)$

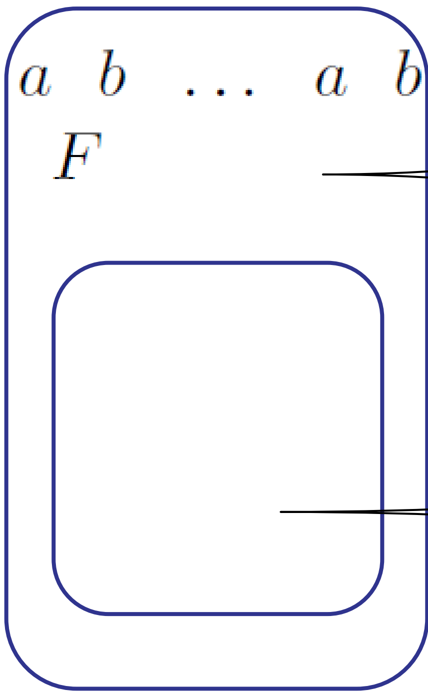
(F, in)

final configuration: F is in the region

P automaton – An example

Given a **regular grammar** with: $S \rightarrow aA, A \rightarrow bS, S \rightarrow \varepsilon$

configuration: rules:



$(S, out; aA, in)$

$(A, out; bS, in)$

$(A, out; bF, in)$

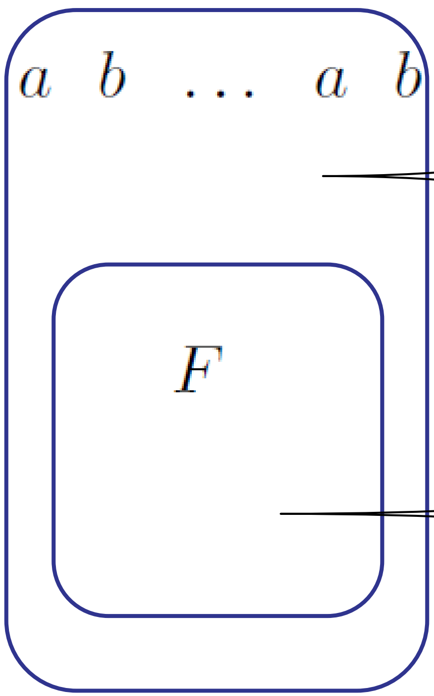
(F, in)

final configuration: F is in the region

P automaton – An example

Given a **regular grammar** with: $S \to aA, A \to bS, S \to \varepsilon$

final configuration:



rules:

$(S, out; aA, in)$

$(A, out; bS, in)$

$(A, out; bF, in)$

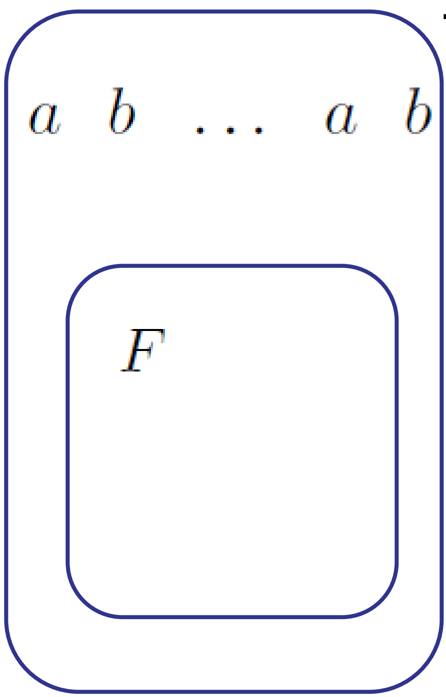
(F, in)

final configuration: F is in the region

P automata – An example

Given a **regular grammar** with rules: $S \to aA, A \to bS, S \to \varepsilon$

final configuration:



The set of accepted **multiset sequences**:

{ {a, A}{b, S}…{a, A} {b, F} }

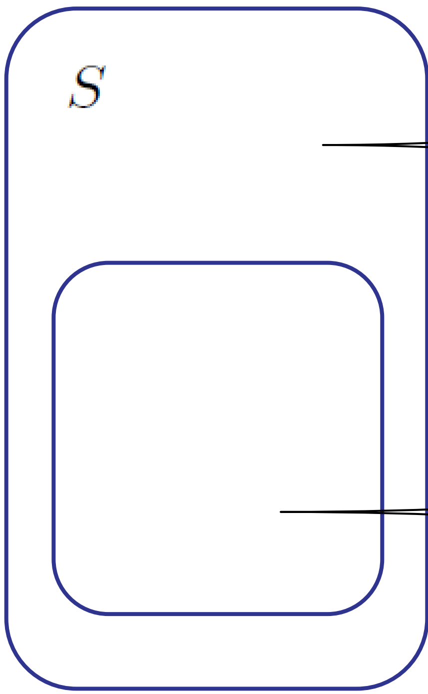
or using the string notation for multisets:

{ aA, bS,… ,aA, bF }

P automaton – An example

Given a **regular grammar** with: $A \rightarrow aB, A \rightarrow a \in P$

initial configuration:



rules:

$$(A, out; aB, in)$$
$$(A, out; aF, in)$$

for all rules of the grammar

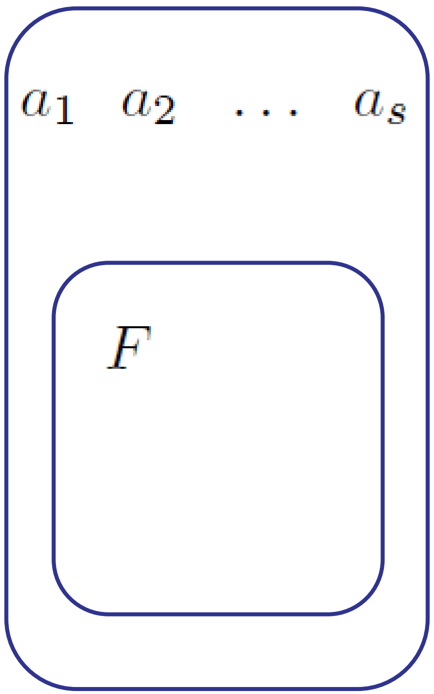
$$(F, in)$$

final configuration: F is in the region

P automaton – An example

Given a **regular grammar** with: $A \to aB, A \to a \in P$

final configuration:



The set of accepted **multiset sequences**:

$$\boxed{\{a_1B_1, a_2B_2, \ldots, a_sF \mid a_1a_2\ldots a_s \in L\}}$$

P automaton – An other example

A **finite automaton** $M = (\Sigma_1, Q, \delta, q_0, F)$ $\Sigma_1 = \{a_1, \ldots, a_k\}$

A simulating P automaton with 2 membranes:

$$w_1 = a\#,$$

$$P_1 = \{(a^i, in; a, out)|_t, (a^{i-1}, out)|_{t'} \mid t = [q_j, a_i, q_k],\ i > 1\} \cup$$
$$\{(a, in; a, out)|_t \mid t = [q_j, a_1, q_k]\},$$

$$w_2 = \{\{t, t' \mid t \in TR\}\},$$

$$P_2 = \{(\#, in; t_0, out) \mid t_0 = [q_0, a_i, q]\} \cup$$
$$\{(t, in; t', out), (t', in; s, out) \mid t \in TR,\ s \in next(t')\},$$

$$F_2 = \{\ \{\{t, t' \mid t \in TR\}\} - \{\{s'\}\} \mid \text{for}$$
$$\text{all } s' \in TR' \text{ such that } s' = [q, a_i, q_f]',\ q_f \in F\}.$$

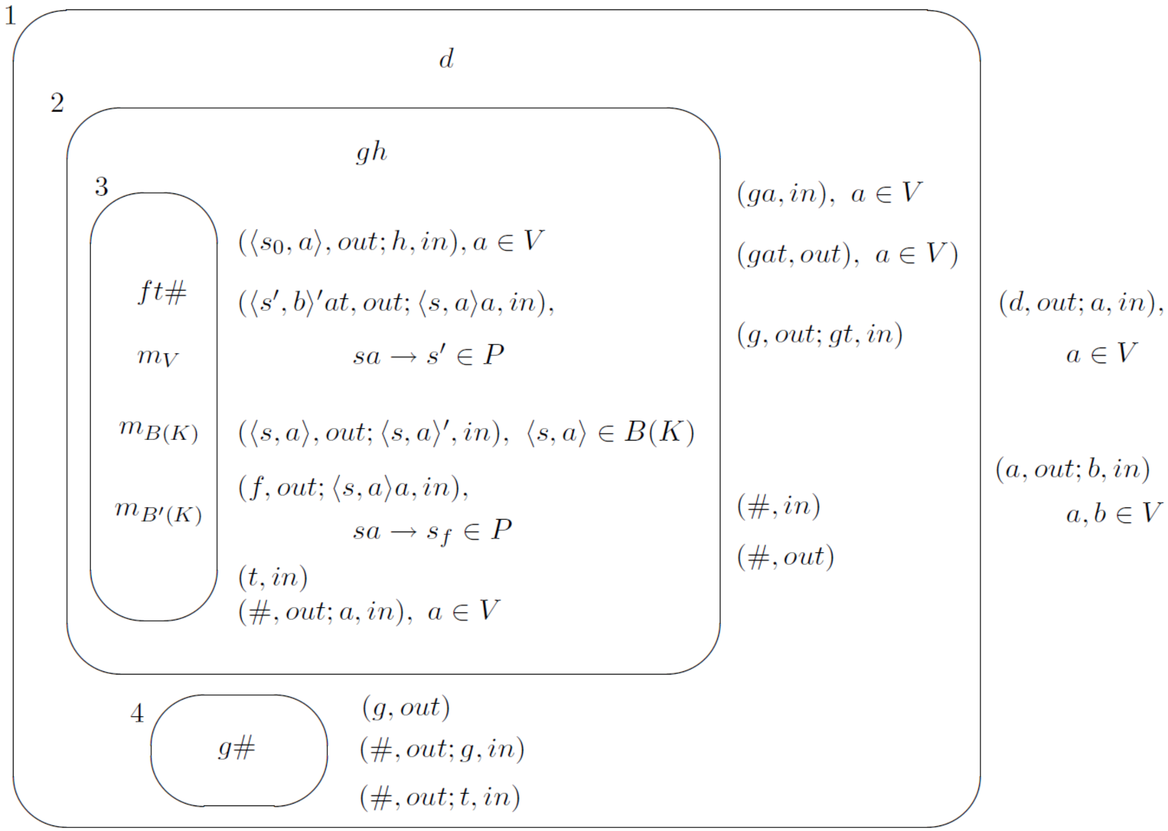
P automaton – An other example

The system simulates a **finite automaton** over $\Sigma_1 = \{a_1, \ldots, a_k\}$ with 2 membranes, and **sequential** rule application.

In this case, it is done in such a way that the accepted **multiset sequences** are:

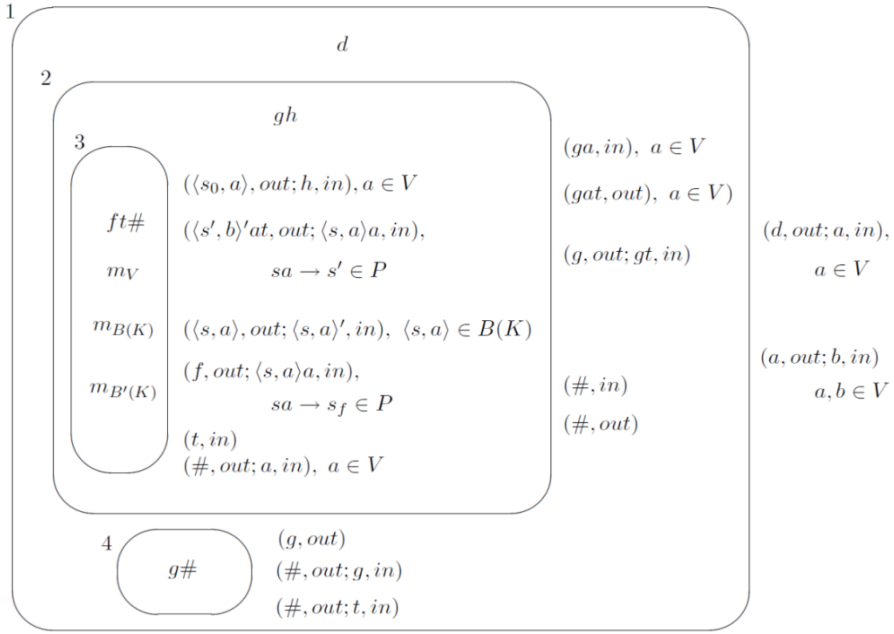
$$\boxed{\{\underbrace{a \ldots a}_{i_1}, \underbrace{a \ldots a}_{i_2}, \ldots, \underbrace{a \ldots a}_{i_s} \mid a_{i_1} a_{i_2} \ldots a_{i_s} \in L\}}$$

P automaton – A third example



[R. Freund, M. Kogler, Gh. Paun, and M. J. Perez-Jimenez. On the power of P and dP automata. *Annals of Bucharest University Mathematics-Informatics Series*, LVIII:5-22, 2009.]

P automaton – A third example



The set of accepted **multiset sequences**:

$$\boxed{\{a_1, a_2, \ldots, a_s \mid a_1 a_2 \ldots a_s \in L\}}$$

P automata

- An **antiport P system** in an **environment** from where the **input** is read
- Given an **initial configuration** and a set of final **(accepting) configurations**
- A **sequence of multisets** is read from the environment during the computation
- The multiset sequence is **accepted** if the computation ends in an **accepting configuration**

- The string interpretation of the accepted **multiset sequence** is **provided by an input mapping**

The input mapping

An **input mapping** maps the **sequences of multisets** over the object alphabet V **to strings** over an alphabet T:

$$\boxed{f : V^* \to 2^{T^*}}$$

The **language accepted** by a P automaton Π :

$$L(\Pi, f) = \{f(v_1) \dots f(v_s) \mid v_1, \dots, v_s \text{ is an accepted multiset sequence of } \Pi\}$$

The input mapping

The **first** example: $\{a_1B_1, a_2B_2, \ldots, a_sF \mid a_1a_2\ldots a_s \in L\}$

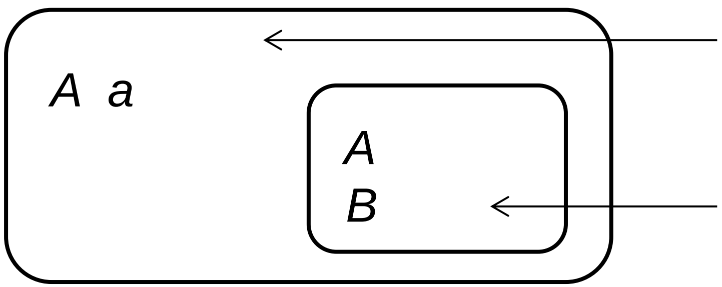
- the mapping: $V = N \cup T,\ f(aA) = \{a\}$ where $A \in N,\ a \in T$

The **second** example: $\{a^{i_1}, a^{i_2}, \ldots, a^{i_s} \mid a_{i_1}a_{i_2}\ldots a_{i_s} \in L\}$

- the mapping: $V = \{a\},\ f(a^i) = \{a_i\},\ a_i \in T = \{a_1, \ldots, a_t\}$

(The third example: $\{a_1, a_2, \ldots, a_s \mid a_1a_2\ldots a_s \in L\}$)

An other example – Input mapping with permutation



(a out; aa in)|A , (a out; bb in)|B

(A out; A in), (B out; A in), (B in)

A configuration sequence, **maximal parallel** rule application:

(Aa, AB) == > (Aaa, AB)== > … == >(Aa…a,AB)== > (Ba^{2k},AA)== > (b^{2k+1},AAB)

If (V*, AAB) is an accepting state, then

$\underbrace{a^2,a^4,a^8,a^{24}\ \ldots,a^{2k}}_{a^{2k+1-2}},b^{2k+1}$ is the accepted multiset sequence

$L=\{a^{n-2}\ b^n \mid n=2^k,\ k>1\}$ could be the accepted language

Input mapping with permutation

$$\boxed{f : V^* \to 2^{T^*}}$$

- $f = f_{perm}$ if $V = T$ and

$$f(v) = \{a_1 a_2 \dots a_s \mid |v| = s, \text{ and } a_1 a_2 \dots a_s \text{ is a permutation of the elements of } v\}$$

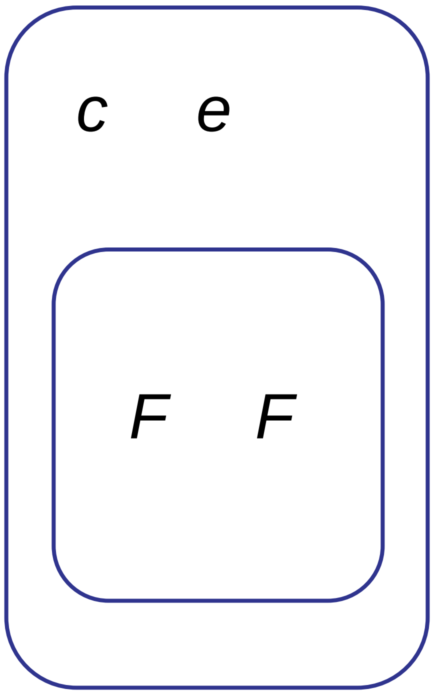
The previous example:

$\underbrace{a^2,a^4,a^8,a^{24} \dots,a^{2k}}_{a^{2k+1-2}},b^{2k+1}$ is the accepted multiset sequence

$L=\{a^{n-2}\ b^n \mid n=2^k, k>1\}$ is the accepted language

What can a “reasonable” input mapping be?

A previous example – input mapping with erasing



The (set of) accepted **multiset sequence(s)**:

$$\{ \ \{c,e,D,D\}\{F,F\} \ \}$$

If the set of terminal symbols is $T=\{e,c\}$, then the accepted strings are:

$$\boxed{\{ \ ce, \ ec \ \}}$$

The desired properties of the input mapping: nonerasing

If **erasing** is allowed, **any language** is easily **obtained** with simple systems having just **one membrane** (extended P automata, analyzing P systems).

Recall the results of [Freund, Oswald 2002]

Therefore, we study **input mappings** that are **nonerasing**.

The desired properties of the input mapping: simplicity

- The **power** of the **system** should **not come** from the **power** of a **complex** input **mapping**

The input mapping should be **simple** from the point of view of **computational complexity**:

Different kinds of input mappings

$$\boxed{f : V^* \to 2^{T^*}}$$

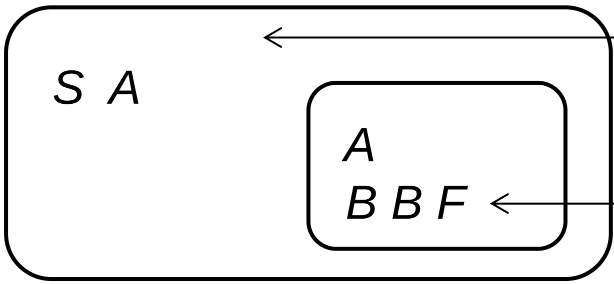
Permutation:

- $f = f_{perm}$ if $V = T$ and
  $f(v) = \{a_1 a_2 \ldots a_s \mid |v| = s, \text{ and } a_1 a_2 \ldots a_s \text{ is a permutation of the elements of } v\}$

Remainder of division by *k*:

- $f = f_{k,rem}$ if $T = \{a_1, a_2, \ldots\}$ and
  $f(v) = \{a_i \mid |v| \text{ divided by } k \text{ gives } i \text{ as remainder}\}$

Example, remainder



(S out; a in)|A , (S out; aa in)|A, (a out; a^{10} in)|A
(Sa out, Sb in)|B, (Saa out; Sbb in)|B, (a^{10} out; a in)|B

(A out; A in), (B out; A in), (B out; B in), (B in), (a in; F out), (b in)

- The number of *a*-s entering the system while *A* is present in the outer region:

$$\underbrace{(1 \text{ or } 2),}_{v_1}\ \underbrace{(10 \text{ or } 20) + (1 \text{ or } 2),}_{v_2}\ \underbrace{(110 \text{ or } 120 \text{ or } 210 \text{ or } 220) + (1 \text{ or } 2),}_{v_3}\ \ldots\ \ldots\ \underbrace{\underline{\qquad\quad}}_{v_m}$$

- If the number of a-s in v_5 is 11212, then $f_{10,rem}(v_1)f_{10,rem}(v_2)\ldots f_{10,rem}(v_5)=a_1a_1a_2a_1a_2$

- The accepted language: $L_{rev}= \{ww^{-1} \mid w \text{ is a string over } \{a_1, a_2\}\}$

A classification of (interesting) input mappings:

- $f = f_{perm}$ if and only if $V = T$, and $f(v) = \{a_1 a_2 \ldots a_s \mid |v| = s, \text{ and } a_1 a_2 \ldots a_s \text{ is a permutation of the elements of}$

(Examples 3, 4)

- $f \in \mathrm{TRANS}$ if and only if, we have $f(v) = \{w\}$ for some $w \in T^*$ which is obtained by applying a **finite transducer** to the string representation of the multiset .

(Examples 1, 2, 5)

To determine the computation power of P automata…

…consider the **workspace** they have available for their computation.

- How does the power depend on the input mapping?
- How does the power depend on sequential or maximal parallel rule application?

To determine the computation power of P automata…

…consider the **workspace** they have available for their computation:

1. In case of "**erasing**" input mappings, the **number** of **objects** inside the system **does not depend** on the **length** of the input.

To determine the computation power of P automata…

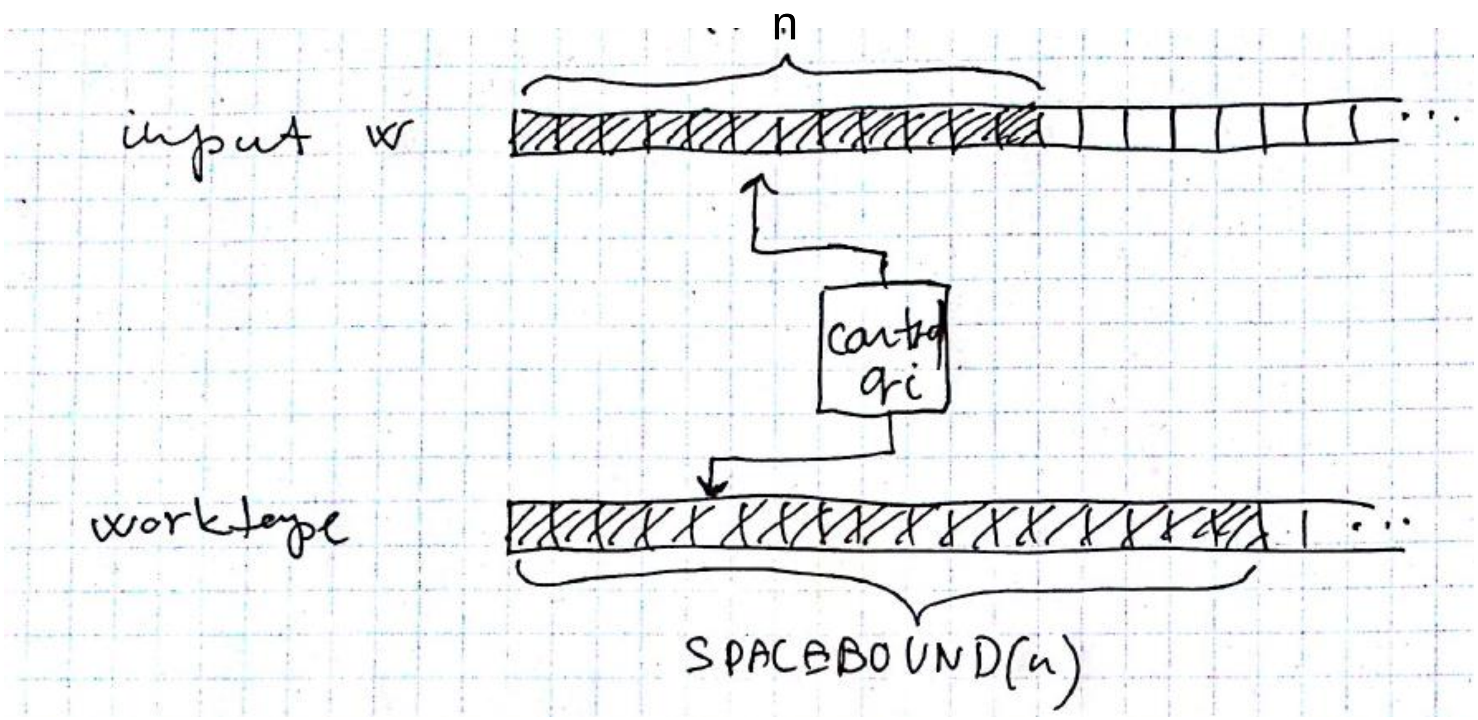
…consider the **workspace** they have available for their computation: (*d* is the number of computational steps so far)

2. In case of $f \in \mathrm{TRANS}$:
   - **sequential** rule application**: configurations** can be recorded by a Turing machine on $\log c \cdot d \sim \log d$ **tape cells**
   - **parallel** rule application: **configurations** can be recorded by a Turing machine on $\log c^d \sim d$ **tape cells**

This **limited workspace** becomes available **step-by-step**, it is bounded by d , the length of the **already processed part** of the input → **restricted space bounded** Turing machines.

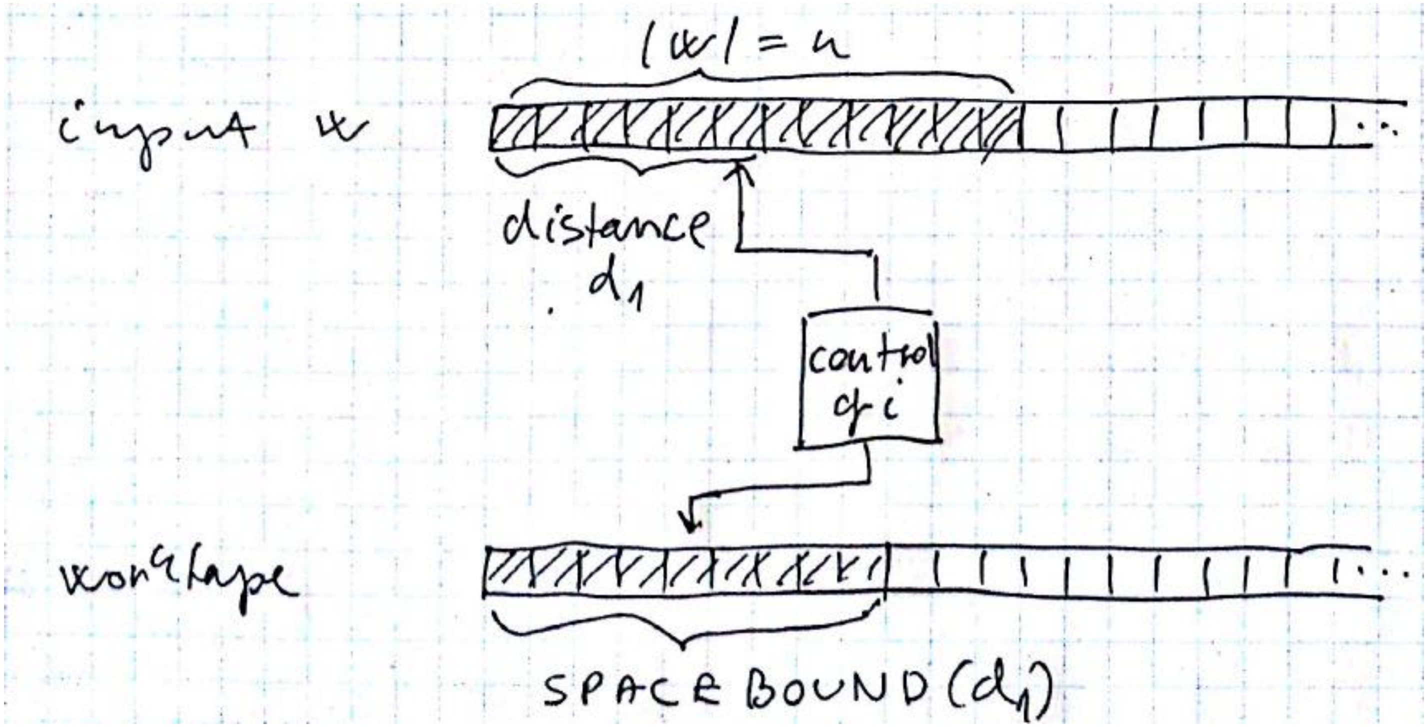
A Turing machine with SPACEBOUND(n)

The length of the available worktape is bounded by the length of the input:



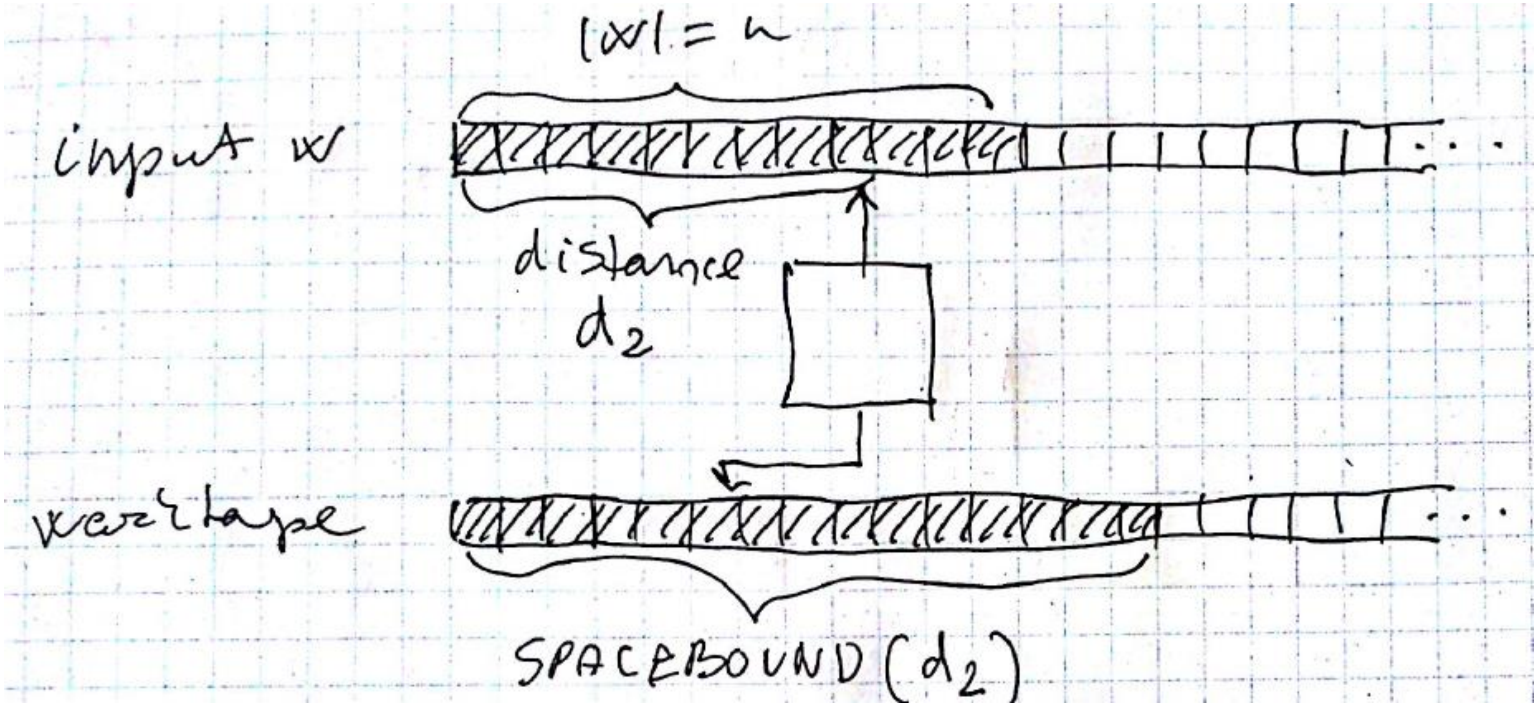
Turing machines with *restricted* space bound

1. After reading d_1 input cells:



Turing machines with *restricted* space bound

2. After reading d_2 input tape cells:



Turing machines with restricted space bound

A nondetermininstic Turing machine with a **one-way** input tape is **restricted** $S(n)$ **space bounded** if the number of **nonempty cells** on the worktape(s) is **bounded by** $S(d)$, where d is the **distance of the reading head** from the left-end of the one-way input tape.

Notations for **logarithmic** space bound:

1LOGSPACE, r1LOGSPACE,

1LINSPACE, r1LINSPCAE

Restricted space complexity

The **restricted** space complexity **classes** are **not** necessarily the **same** as the „usual” ones.

Consider for example:

$$L = \{xy \mid x \in \{1,2,\ldots,9\}\{0,1,\ldots,9\}^*, y \in \{\#\}^+,\ val(x) = |y|\}.$$

(11############ is in *L, 3#####* is not in *L)*

L is **in** *1LOGSPACE*, but it is **not in** *r1LOGSPACE*.

Restricted space complexity

The **restricted logarithmic space** bound:

- $r1LOGSPACE \subset 1LOGSPACE$
- In the **deterministic** case, it is equal to the **strong logarithmic space** bound.

The **restricted linear space** bound:

- $r1LINSPACE = LINSPACE$

[E. Csuhaj-Varju, O.H. Ibarra, Gy. Vaszil: On the computational complexity of P automata. *Lecture Notes in Computer Sci.*, 3384 (2005), 77–90.]

[M. Kutrib, J. Provillard, Gy. Vaszil, M. Wendtland: Deterministic One-Way Turing Machines with Sublinear Space. *Fundam. Inform.* 136(1-2): 139-155 (2015)]

The power of systems with mappings by finite transducers

1. $\mathcal{L}_{par}(PA, TRANS) = r1LINSPACE = CS$

   For **any** kind of $f : V^* \to 2^{T^*}$ as long as it is **not more** complex than **linear space computable** (by Turing machines), $L(\Pi, f) \in \mathrm{CS}$.

2. $\mathcal{L}_{seq}(PA, TRANS) = r1LOGSPACE \subset 1LOGSPACE$

[E. Csuhaj-Varju, O.H. Ibarra, Gy. Vaszil: On the computational complexity of P automata. *Lecture Notes in Computer Sci.*, 3384 (2005), 77–90.]

The characterization of CS in more detail

For **any context-sensitive** language L, a **P automaton** Π can be constructed, such that $L = L(\Pi, f_1)$ for a mapping f_1 where

$f_1(x) = a$ for $x = a^k$, and $f_1(x) = \{\varepsilon\}$ if x is the empty multiset.

The characterization of CS in more detail

For **any P automaton** Π with object alphabet V and mapping $f : V^* \to 2^{T^*}$ for some alphabet T, such that f is **linear-space computable**, the language $L(\Pi, f) \subseteq T^*$ is **context-sensitive**.

Mappings in *TRANS* and the mapping f_{perm}

The **language** by Example 5 (with *f* from *TRANS*):

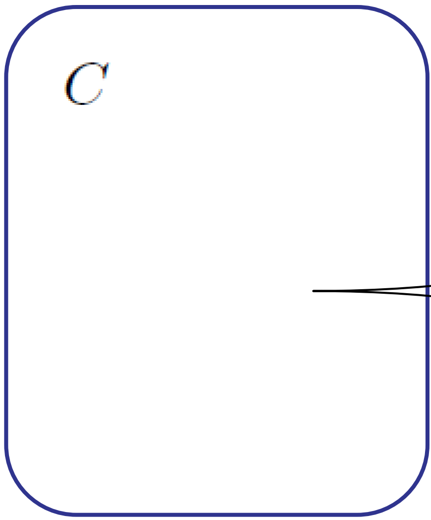
$$L_{rev} = \{ww^{-1} \mid w \text{ is a string over } \{a, b\}\}$$

This is **interesting** because L_{rev} **cannot** be **characterized** using **permutations** as shown in:

[R. Freund, M. Kogler, Gh. Paun, and M. J. Perez-Jimenez. On the power of P and dP automata. *Annals of Bucharest University Mathematics-Informatics Series*, LVIII:5-22, 2009.]

Systems with mappings from *TRANS*

initial configuration:



rules:

$$\left.\begin{array}{l}(C, out; AC, in)\\(AC, out; BD, in)\\(AD, out; BD, in)\\(B, out)\\\text{final configuration: A single } D \text{ is in the region}\end{array}\right.$$

The accepted multiset sequences: $\{(AC)^n(BD)^n \mid n \geq 1\}$

Consider: $f_1(AC) = \{ab\},\ f_1(BD) = \{ac\}$

$f_2(AC) = \{aac\},\ f_2(BD) = \{bbd\}$

There are simple **linear languages** which cannot be characterized with systems using f_{perm} .

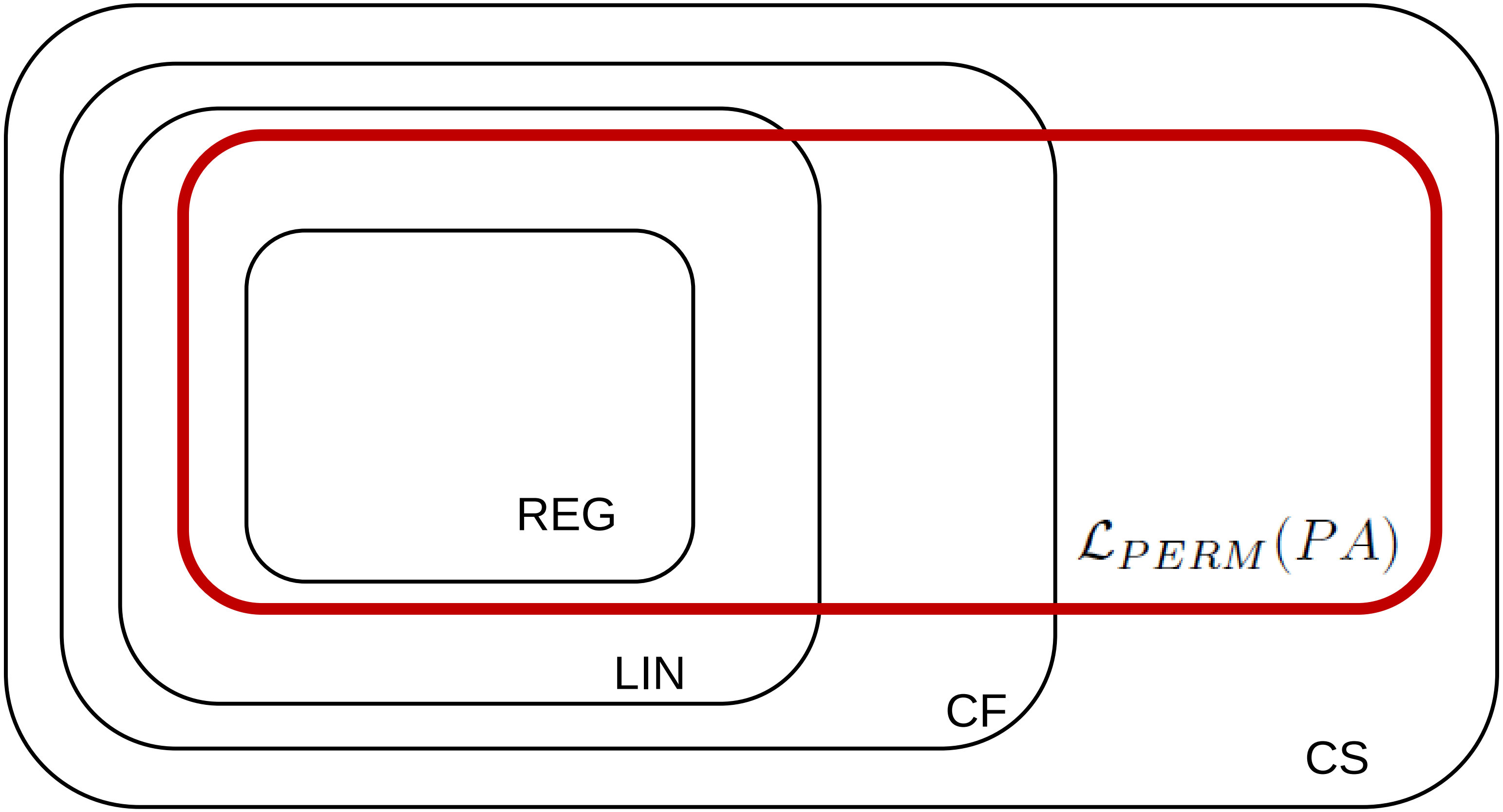
$$\boxed{L = \{(ab)^n(ac)^n \mid n \geq 1\} \notin \mathcal{L}_{PERM}(PA)}$$

On the other hand:

$$\{(aac)^n(bbd)^n \mid n \geq 1\} \in \mathcal{L}_{PERM}(PA)$$

[Paun, G., Perez-Jimenez, M.J.: Solving problems in a distributed way in membrane computing: dP systems. *International Journal of Computing, Communication and Control* V(2), 238–250 (2010)]

Systems with permutation mappings



[R. Freund, M. Kogler, Gh. Paun, and M. J. Pérez-Jiménez. On the power of P and dP automata. *Annals of Bucharest University Mathematics-Informatics Series*, LVIII:5-22, 2009.]

Let us investigate the **power** systems with **permutation** mappings.

The power of P automata with permutation mapping

$$\boxed{\begin{array}{r}\mathcal{L}_X(\mathrm{PA}, f_{perm}) \subset \mathrm{r1LOGSPACE} = \mathcal{L}_{seq}(PA, TRANS) \\ where\ X \in \{seq, par\}.\end{array}}$$

- The **inclusion** is shown by a **counter machine model – RCMA**
- The **strictness** is shown using:

$$L_1 = \{(ab)^n \# w \mid w \in \{1\}\{0,1\}^* \;\; val(w) = n > 1\}$$

and a lemma from [Freund, Kogler, Paun, Pérez-Jiménez 2010]

[E. Csuhaj-Varjú, Gy. Vaszil: On counter machines vs. dP automata. *LNCS 8340*, 138-150, 2014]

The power of P automata, general formulation

Notation: for $S : \mathbb{N} \to \mathbb{N}$,

$L \in NSPACE(S)$– as usual

$L \in r1NSPACE(S)$– there is a Turing machine with a one-way read-only input tape accepting L using a workspace of at most $S(d)$ in each step of an accepting computation where d is the number of cells read on the input tape

The power of P automata, general formulation

Let Π be a P automaton, and let $S : \mathbb{N} \to \mathbb{N}$, such that $S(d)$ bounds the number of objects inside the system in the i-th step of functioning, $d \leq i$ being the number of transitions in which a nonempty multiset was imported into the system from the environment.

If f is non-erasing and $f \in NSPACE(S_f)$, then $L(\Pi, f) \in r1NSPACE(\log(S) + S_f)$.

P automata over infinite alphabets

An interesting restriction of P automata

P finite automata:

- the object **alphabet** $V \cup \{a\}$ contains a **distinguished symbol** a
- the **skin** region contains **rules of the form** $(x, in; y, out)|_Z$ with $x \in \{a\}^*$, $y \in (V \cup \{a\})^*$, $Z \in \{z, \neg z\}$, $z \in V^*$
- the **other membranes** contain **rules of the form** $(x, in; y, out)|_Z$ with $Z \in \{z, \neg z\}$, $x, y, z \in V^*$

[J.Dassow, Gy. Vaszil: P finite automata and regular languages over countably infinite alphabets. *Lecture Notes in Computer Sci.*, 4361 (2006), 367–381.]

P finite automata

As the **input** multisets **can only contain the symbol** a, it is appropriate to have

$$f_2 : \{a\}^* \to 2^{T^*} \text{ with } f_2(\underbrace{a, \ldots, a}_{i}) = \{a_i\}$$

P finite automata

A **language** L **is regular** if and only if there is a **P finite automaton** Π with object alphabet $V \cup \{a\}$, such that $L = L(\Pi, f_2)$.

P automata over infinite alphabets

Because of the **maximal parallel rule application**, the number of possible inputs is **infinite**, thus, we might map the input multisets to an **infinite alphabet**.

$\longrightarrow$ an automata-like **device over infinite alphabets**

$\longrightarrow$ **P finite automaton - regular** languages over infinite alphabets

P finite automata over infinite alphabets

$$f_2 : \{a\}^* \to 2^{T^*} \text{ with } f_2(\underbrace{a, \ldots, a}_{i}) = \{a_i\}$$

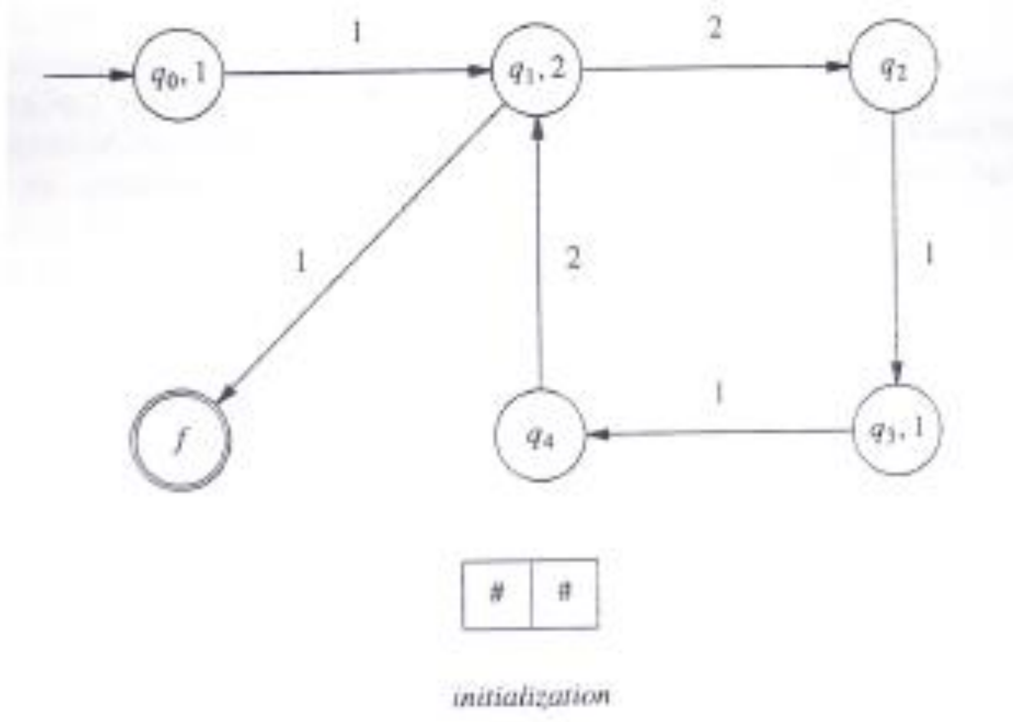
$$T = \{a_1, a_2, \ldots, a_i, \ldots\} \longleftrightarrow \{\{a\}\}, \{\{a, a\}\}, \ldots, \{\{\underbrace{a, \ldots, a}_{i}\}\}, \ldots$$

How to classify languages over infinite alphabets?

Two "natural" analogues of **regular languages**:

- [M. Kaminski, N. Francez 1994] - languages accepted by **finite-memory automata**
- [F. Otto 1985] - languages characterized by **Δ-regular expressions**

Finite memory automata



An accepted string: $a_1a_2a_1a_3a_2a_4a_3a_4$

Regularity - finite memory automata

"L is regular if accepted by a **finite memory automaton**"

Checking equality of symbols is "easy", but:

$$\{a_{2i} \mid i \geq 1\}$$

cannot be characterized this way

Regularity – Δ regular expressions

Let Δ be an infinite alphabet.

- $\emptyset$ and ε denote the empty set and $\{\varepsilon\}$, respectively,
- $a_i \in \Delta$ denotes $\{a_i\}$,
- <u>for $a_i \in \Delta$, $j \geq 1$, expression $a_{i,j}$ denotes $\{a_{i+kj} \mid k \geq 0\}$,</u>
- if r, s are Δ-regular expressions denoting R, S, then $r + s$, rs, and r^* denote $R \cup S$, RS, and R^*, respectively.

Regularity – Δ regular expressions

"L is regular if **described by a Δ-regular expression**"

Checking relationships between symbols is possible, but:

$$\{a_i a_i \mid i \geq 1\}$$

cannot be characterized this way

Finite memory automata and Δ regular expressions

$$L_1 = \{a_{2i} \mid i \geq 1\} \notin \mathcal{L}(FMA) \text{ but } L_1 \in \mathcal{L}(\Delta - RegExp),$$

and

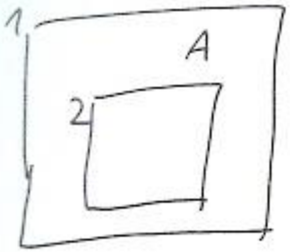
$$L_2 = \{a_i a_i \mid i \geq 1\} \in \mathcal{L}(FMA) \text{ but } L_2 \notin \mathcal{L}(\Delta - RegExp),$$

thus $\mathcal{L}(FMA)$ and $L \in \mathcal{L}(\Delta - RegExp)$ are **incomparable**.

L_1 is described by the expression $a_{2,2}$

L_2 is also easily described by FMA

P finite automata for L_1

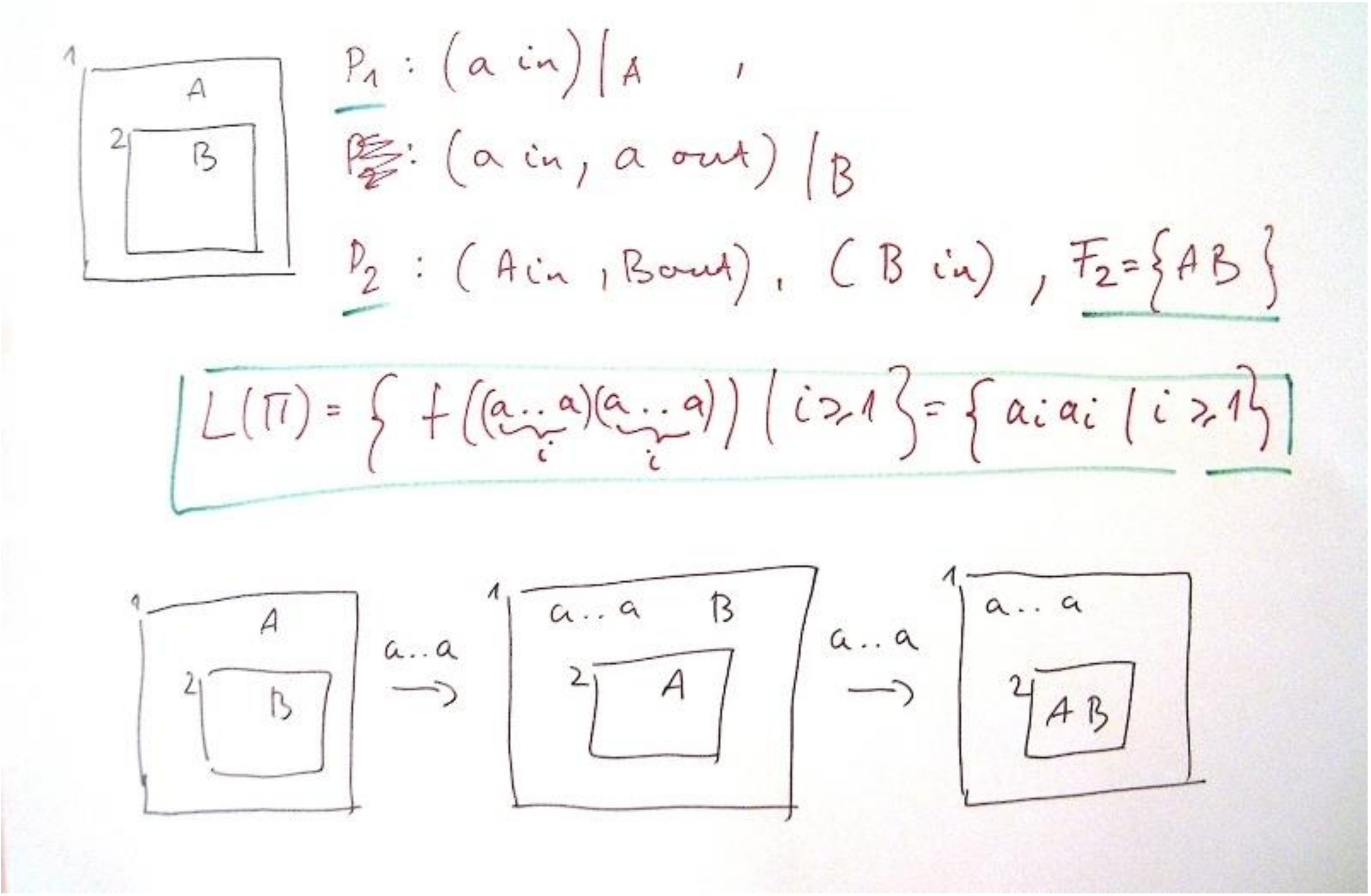


$P_1 : (aa\ in)|_A$

$P_2 : (A\ in)$

$$\boxed{L = \{ f(\underbrace{a \ldots a}_{2i}) \mid i \geq 1 \} = \{ a_{2i} \mid i \geq 1 \}}$$

P finite automata for L_2



Using **P finite automata**, we might obtain a more **appropriate definition** of **regular languages** over **infinite** alphabets.

This is an interesting research direction which is **still open.**

Thank you for your attention!

- Membrane systems (P systems) with communication rules only, accepting P systems
- P automata
  - The computational power of P automata
  - P automata over infinite alphabets