A short introduction to P Automata

György Vaszil
University of Debrecen

CMC 16 – Tutorial Day
Outline

- From accepting symport/antiport systems to P automata
  - Analyzing P systems, extended P automata, symport/antiport acceptors
- Variants of P automata and their power
  - The role of different input mappings
  - Complexity classes and equivalent counter automata
  - P automata over infinite alphabets
- Distributed P automata
  - Concatenated languages, efficiency of parallelization
  - Agreement languages, multihead automata, two-way distributed P automata
Membrane systems
Membrane systems, a membrane structure

A hierarchical arrangement of regions where multisets of objects evolve according to given evolutionary rules
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 parallel
- In a synchronized manner

[Paun 2000 (1998)]
Example

\[ a \rightarrow a \begin{pmatrix} b, in_2 \\ c, in_2 \\ c, in_2 \end{pmatrix} \]
\[ aa \rightarrow (a, out) (a, out) \]

We obtain:
\[ \{ b \ldots b, c \ldots c \} \]
\[ \begin{pmatrix} 2n \\ 4n \end{pmatrix} \]
Example

\[
\begin{align*}
[ a, a \ [ \ ]_2 ]_1 & \Rightarrow a \rightarrow a(b,in_2)(c,in_2)(c,in_2) [ a, a \ [ b, c, c, b, c, c \ ]_2 ]_1 \\
& \Rightarrow a \rightarrow a(b,in_2)(c,in_2)(c,in_2) \ldots \Rightarrow a \rightarrow a(b,in_2)(c,in_2)(c,in_2)
\end{align*}
\]

\[
[ a, a \ [ \underbrace{b, \ldots, b}_2, c, \ldots, c \ ]_2 ]_1 \Rightarrow a a \rightarrow (a,\text{out})(a,\text{out}) [ \underbrace{b, \ldots, b}_2, c, \ldots, c \ ]_2 ]_1
\]

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

\[
L = \{ \{ \underbrace{b, \ldots, b}_2, c, \ldots, c \} \mid n \geq 0 \}\]
Automata-like systems

- state transitions
- reading an input
- working with tapes/counters
Automata-like membrane systems

- state transitions
- reading an input
  → operating in an “environment”
- working with tapes/counters
  → using resources represented by multisets
Membrane systems with communication only

Symport/antiport systems

uniporter  antiporter  ATPase  ion channel

ATP  ADP + P_i

phospholipid head
phospholipid tail

Box 1
Example, in communication with the environment

We obtain:

\[ \{ b, \ldots, b, c, \ldots, c \} \]
Example, in communication with the environment

\[
\begin{align*}
a, a, b, c, a, b, b, c, c, & \ldots \quad [a, a [ \ ]_2 ]_1 \Rightarrow \\
a, a, , , a, b, , c, & \ldots \quad [a, b, c, c, a, b, c, c [ \ ]_2 ]_1 \Rightarrow \\
& \ldots \quad [a, b, c, c, a, b, c, c [ b, c, c, b, c, c ]_2 ]_1 \\
& \Rightarrow \ldots \Rightarrow \\
& \ldots \quad [a, b, c, c, a, b, c, c [ b, \ldots, b, c, \ldots, c ]_2 ]_1 \Rightarrow \ldots \quad [ b, \ldots, b, c, \ldots, c ]_2 ]_1 \\
\end{align*}
\]

\[
L = \{ \{b, \ldots, b, c, \ldots, c\} \} \quad | \quad n \geq 0 \}
\]
Symport/antiport systems – The rules

Symport/antiport rules, possibly with promoters:

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

- 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
Towards P automata: Accepting symport/antiport systems
(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
- \( \mu \) – 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

π = (V, μ, E, w₁, ..., wₙ, R₁, ..., Rₙ, F, iᵢᵣₙ)

A configuration:
(u₀, u₁, ..., uₙ), uᵢ ∈ V*, the contents of the environment and the n regions of π

The transition mapping:

δ : V* × (V*)ₙ₊₁ → (V*)ⁿ,
δ(u, (u₀, u₁, ..., uₙ)) ⊆ (u₀', u₁', ..., uₙ')

the multiset entering the skin the configuration the new configuration membrane
Symport/antiport systems and counter automata

the **multiplicity** of the object $a_i$

⇔ the value of counter $i$

the **presence** of the object $q$

⇔ being in the **internal state** $q$
Symport/antiport systems and counter automata

\((q, \text{out}; q' a_i, \text{in}) \iff \) 
- change the \textbf{state} \(q \rightarrow q'\)
- increase the value of counter \(i\)

\((qa_i, \text{out}; q', \text{in}) \iff \) 
- change the \textbf{state} \(q \rightarrow q'\)
- decrease the value of counter \(i\)
Symport/antiport systems and counter automata

\( (q, \text{out}; q_1q_2, \text{in}) \)
\( (q_1a_i, \text{out}) \)
\( (q_2, \text{out}; q_3, \text{in}) \)
\( (q_1q_3, \text{out}; q', \text{in}) \)

\( \leftarrow \rightarrow \)

- change the state \( q \longrightarrow q' \)
- check the emptiness of counter \( i \)

- \( (q_f, \text{out}; f, \text{in}) \)
- no rule for \( f \)

\( \leftarrow \rightarrow \)

\( q_f \) is a final state
Symport/antiport systems and counter automata

\[ N(\Pi) = \{ k \in \mathbb{N} \mid c_0 = (w_0, w_1, \ldots, w_n), \text{ with } |w_{in}|a_i = k, \]

and there is \( c_t \in F \) and a sequence \( c_i \) 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
Symport/antiport systems are computationally complete

The weight of the rules and the number of membranes are bounded "easily''.

- **bounded number of membranes and rules**
  [Păun, Păun 2002], [Freund, Oswald 2003], [Freund, Păun 2003], [Frisco, Hoogeboom 2003]

- **minimal cooperation**
  [Kari, Martín-Vide, Păun 2004], [Bernardini, Păun 2003],
  [Frisco 2004], [Margenstern, Rogozhin, Rogozhin, Verlan 2004], [Vaszil 2004], [Alhazov, Freund, Rogozhin 2005]

- **symport only**
  [Frisco, Hoogeboom 2004], [Alhazov, Freund, Rogozhin 2005]
Symport/antiport systems are computationally complete

The number of rules and symbols can also be bounded by simulating universal counter machines.
From multiplicities (numbers) to sequences (strings)...

Consider the **multiset sequences** accepted by antiport P systems
Accepted multiset sequences - Example

initial configuration:

\[
\begin{array}{c}
A \\
B
\end{array}
\]

rules:

\[
\begin{align*}
(A, & \text{out}; cD, \text{in}) \\
(B, & \text{out}; eD, \text{in}) \\
(D, & \text{out}; F, \text{in}) \\
(F, & \text{in})
\end{align*}
\]
Accepted multiset sequences - Example

configuration: \[ cD eD \]

rules:

- \( (A, \text{out}; cD, \text{in}) \)
- \( (B, \text{out}; eD, \text{in}) \)
- \( (D, \text{out}; F, \text{in}) \)
- \( (F, \text{in}) \)
Accepted multiset sequences - Example

configuration:    rules:

\[ c F e F \]  \[ (A, \text{out}; cD, \text{in}) \]
\[ (B, \text{out}; eD, \text{in}) \]
\[ (D, \text{out}; F, \text{in}) \]
\[ (F, \text{in}) \]
Accepted multiset sequences - Example

configuration:  
\[
\begin{array}{ccc}
F & F \\
\end{array}
\]

rules:
\[
(A, \text{out}; cD, \text{in}) \\
(B, \text{out}; eD, \text{in}) \\
(D, \text{out}; F, \text{in}) \\
(F, \text{in})
\]
final configuration:

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

\[
\{ \{c,e,D,D\}, \{F,F\} \}
\]
Accepting P systems – What we have so far...

- An antiport 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
From accepting symport/antiport systems to P automata:

Analyzing P systems, extended P automata, symport/antiport acceptors
How to map the accepted multiset sequences to accepted strings?

1. Analyzing P systems, extended P automata and P system acceptors:

   - **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**
From accepting symport/antiport systems to P automata:

Analyzing P systems,
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

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:

\[
\{ ce, ec \}
\]
The power of analyzing P systems

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

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

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

<table>
<thead>
<tr>
<th>symbols</th>
<th>(z) - ary digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_1)</td>
<td>(\leftrightarrow)</td>
</tr>
<tr>
<td>(a_2)</td>
<td>(\leftrightarrow)</td>
</tr>
<tr>
<td></td>
<td>(\vdots)</td>
</tr>
<tr>
<td>(a_{z-1})</td>
<td>(\leftrightarrow)</td>
</tr>
</tbody>
</table>
The numerical encoding

\[ w = a_{i_1} \ldots a_{i_k} \in \Sigma^* \quad \iff \quad \text{code}(w) = (i_1) \ldots (i_k) \in \mathbb{N} \]

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

\[ \text{code}(wa_i) = \text{code}(w) \cdot z + i \]

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-nontamental 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?
From accepting symport/antiport systems to P automata:

Extended P automata
Finite (extended) P automata with antiport rules

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

- a set of terminal objects \( T \subseteq V \)

- rules of two types:
  1. \((q, \text{out}; pa, \text{in})\) with \( q, p \in V - T, \ a \in T \)
  2. \((pa, \text{out}; r, \text{in})\) 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

Finite (extended) P automata with antiport rules

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

\[ L(\Pi) = \bigcup \text{str}_T(u_1) \cdot \text{str}_T(u_2) \cdot \ldots \cdot \text{str}_T(u_t) \]

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

\( L \) is regular if and only if it is accepted by a finite P automaton with antiport rules.
From accepting symport/antiport systems to P automata:

Symport/antiport acceptors
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

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

\[ L(\Pi) = \bigcup \text{str}_T(u_1) \cdot \text{str}_T(u_2) \cdot \ldots \cdot \text{str}_T(\bar{u}_t) \in T^* \]

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
- \( \text{str}_T(u) \in T^* \) 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 $u, v \in (V - T)^*, |u| \geq |v|, a \in T$

P automata:

Variants of P automata and their power, the role of different input mappings
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

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 configuration \( E_1 \times \ldots \times E_n, E_i \subseteq V^* \) with \( E_i \) being finite, or \( E_i = V^* \)
Given a **regular grammar** with:

\[ S \rightarrow aA, A \rightarrow bS, S \rightarrow \varepsilon \]

**initial configuration:**

- \( S \)

**rules:**

- \((S, out; aA, in)\)
- \((A, out; bS, in)\)
- \((A, out; bF, in)\)

**final configuration:** \( F \) is in the region
Given a **regular grammar** with:

\[
S \rightarrow aA, \ A \rightarrow bS, \ S \rightarrow \varepsilon
\]

configuration: \(a\)

\(A\)

rules:

- \((S, \text{out}; aA, \text{in})\)
- \((A, \text{out}; bS, \text{in})\)
- \((A, \text{out}; bF, \text{in})\)

\((F, \text{in})\)

final configuration: \(F\) is in the region
Given a regular grammar with:

\[ S \rightarrow aA, \ A \rightarrow bS, \ S \rightarrow \varepsilon \]

configuration: \[ a \ b \]

rules:

\[
\begin{align*}
(S, \text{out}; aA, \text{in}) \\
(A, \text{out}; bS, \text{in}) \\
(A, \text{out}; bF, \text{in}) \\
(F, \text{in})
\end{align*}
\]

final configuration: \( F \) is in the region
P automaton – An example

Given a regular grammar with:

configuration: rules:

\[ S \rightarrow aA, A \rightarrow bS, S \rightarrow \varepsilon \]

configuration: rules:

\[ a \quad b \quad \ldots \quad a \quad A \]

\[ (S, out; aA, in) \]

\[ (A, out; bS, in) \]

\[ (A, out; bF, in) \]

\[ (F, in) \]

final configuration: \( F \) is in the region
P automaton – An example

Given a regular grammar with:

configuration: rules:

$$S \rightarrow aA, A \rightarrow bS, S \rightarrow \varepsilon$$

final configuration: $F$ is in the region
Given a **regular grammar** with:

\[ S \rightarrow aA, A \rightarrow bS, S \rightarrow \varepsilon \]

**final configuration:**

- \( F \)

**rules:**

- \((S, \text{out}; aA, \text{in})\)
- \((A, \text{out}; bS, \text{in})\)
- \((A, \text{out}; bF, \text{in})\)
- \((F, \text{in})\)

**final configuration:** \( F \) is in the region
Given a **regular grammar** with rules: \( S \rightarrow aA, A \rightarrow bS, S \rightarrow \varepsilon \)

final configuration:

The set of accepted **multiset sequences**:

\[
\{ \{a, A\}, b, S\}, \ldots, \{a, A\}, \{b, F\} \}
\]

or using the string notation for multisets:

\[
\{ aA, bS, \ldots, aA, bF \}
\]
Given a regular grammar with:

\[ A \rightarrow aB, \ A \rightarrow a \in P \]

initial configuration:

rules:

- \((A, \text{out}; aB, \text{in})\)
- \((A, \text{out}; aF, \text{in})\)

for all rules of the grammar

final configuration: \(F\) is in the region
P automaton – An example

Given a regular grammar with:

\[ A \rightarrow aB, \ A \rightarrow a \in P \]

final configuration:

The set of accepted multiset sequences:

\[ \{a_1 B_1, a_2 B_2, \ldots, a_s F \mid a_1 a_2 \ldots a_s \in L \} \]
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:

\[
\begin{align*}
  w_1 &= a\#, \\
  P_1 &= \{(a^i, \text{in}; a, \text{out})|_t, (a^{i-1}, \text{out})|_{t'} \mid t = [q_j, a_i, q_k], \ i > 1\} \cup \\
          &\quad \{(a, \text{in}; a, \text{out})|_t \mid t = [q_j, a_1, q_k]\}, \\
  w_2 &= \{\{t, t' \mid t \in TR\}\}, \\
  P_2 &= \{(#, \text{in}; t_0, \text{out}) \mid t_0 = [q_0, a_i, q]\} \cup \\
          &\quad \{(t, \text{in}; t', \text{out}), (t', \text{in}; s, \text{out}) \mid t \in TR, \ s \in next(t')\}, \\
  F_2 &= \{ \{\{t, t' \mid t \in TR\}\} - \{\{s'\}\} \mid \text{for} \\
          &\quad \text{all } s' \in TR' \text{ such that } s' = [q, a_i, q_f]', \ q_f \in F\}. 
\end{align*}
\]
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:

$$\{a \ldots a, a \ldots a, \ldots, a \ldots a \mid a_{i_1} a_{i_2} \ldots a_{i_s} \in L\}$$
P automaton – A third example

The set of accepted multiset sequences:

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

$$f : V^* \rightarrow 2^{T^*}$$

The **language accepted** by a P automaton $\Pi$:

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

The first example: \( \{ a_1 B_1, a_2 B_2, \ldots, a_s F \mid a_1 a_2 \ldots a_s \in L \} \)
- 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: \( V = \{ a \}, \ f(a^i) = \{ a_i \}, \ a_i \in T = \{ a_1, \ldots, a_t \} \)
An other example – Input mapping with permutation

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

\[(Aa, AB) \Rightarrow (Aaa, AB) \Rightarrow \ldots \Rightarrow (Aa\ldots a, AB) \Rightarrow (B^{2k}, AA) \Rightarrow (b^{2k+1}, AAB)\]

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

\[L = \{a^{n-2}b^n \mid n=2^k, k>1\}\]

could be the accepted language.
Input mapping with permutation

\[ f : V^* \rightarrow 2T^* \]

- \( f = f_{\text{perm}} \) if \( V = T \) and

\[ f(v) = \{a_1a_2\ldots a_s \mid |v| = s, \text{ and } a_1a_2\ldots a_s \text{ is a permutation of the elements of } v\} \]

The previous example:

\[ \{a^2, a^4, a^8, a^{24}, \ldots, a^{2^k}, b^{2^k+1} \} \]

\[ a^{2^k+1-2} \]

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:

\[
\{ ce, ec \}
\]
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

Permutation:

- \( f = f_{\text{perm}} \) if \( V = T \) and
  \[ f(v) = \{a_1a_2\ldots a_s \mid |v| = s, \text{ and } a_1a_2\ldots a_s \text{ is a permutation of the elements of } v \} \]

Remainder of division by \( k \):

- \( f = f_{k,\text{rem}} \) 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

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

\[
(1 \text{ or } 2), \ (10 \text{ or } 20) + (1 \text{ or } 2), \ (110 \text{ or } 120 \text{ or } 210 \text{ or } 220) + (1 \text{ or } 2), \ldots
\]

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

The accepted language: 
\[L_{\text{rev}}= \{ww^{-1} \mid w \text{ is a string over } \{a_1, a_2\}\}\]
A classification of (interesting) input mappings:

- \( f = f_{\text{perm}} \) if and only if \( V = T \), and
  \[ f(v) = \{a_1a_2\ldots a_s \mid |v| = s, \text{ and } a_1a_2\ldots a_s \text{ is a permutation of the elements of } v \} \]
  (Examples 3, 4)

- \( f \in \text{TRANS} \) 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 \( v \).
  (Examples 1, 2, 5)
Variants of P automata and their power:

Complexity classes and equivalent counter automata
To determine the computation power of P automata...

...consider the workspace they have available for their computation:
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:

2. In case of \( f \in \text{TRANS} \):
   - **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.
A Turing machine with $\text{SPACEBOUND}(n)$

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

\[ \text{input } w \]

\[ \text{worktape} \]

\[ \text{SPACEBOUND}(n) \]
Turing machines with restricted space bound

1. After reading $d_1$ input cells:
Turing machines with \textit{restricted} space bound

2. After reading $d_2$ input tape cells:
A nondeterministic 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.

**Notation:**

\[
1\text{LOGSPACE}, \ r1\text{LOGSPACE}, \\
1\text{LINSPACE}, \ r1\text{LINSPACEAE}
\]
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** \( 1\text{LOGSPACE} \), but it is **not in** \( r1\text{LOGSPACE} \).
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:

- \( r1LINSPACE = LINSpace \)


The power of systems with mappings in TRANS

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

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

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

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

$$f_1(x) = a \text{ for } x = a^k,$$ and $$f_1(x) = \{\varepsilon\} \text{ if } x \text{ is the empty multiset.}$$
The characterization of CS in more detail

For any P automaton $\Pi$ with object alphabet $V$ and mapping $f : V^* \rightarrow 2^{T^*}$ for some alphabet $T$, such that $f$ is linear-space computable, the language $L(\Pi, f) \subseteq T^*$ is context-sensitive.
The **language** by Example 5 (with $f$ from $\textit{TRANS}$):

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

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

Systems with mappings from \textit{TRANS}

initial configuration: 

rules:

\begin{align*}
(C, \text{out}; AC, \text{in}) \\
(AC, \text{out}; BD, \text{in}) \\
(AD, \text{out}; BD, \text{in}) \\
(B, \text{out})
\end{align*}

final configuration: A single $D$ is in the region

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

Consider:

\[
\begin{align*}
 f_1(AC') &= \{ab\}, \\
 f_1(BD) &= \{ac\} \\
 f_2(AC') &= \{aac\}, \\
 f_2(BD) &= \{bbd\}
\end{align*}
\]
There are simple linear languages which cannot be characterized with systems using \( f_{\text{perm}} \).

\[
L = \{(ab)^n(ac)^n \mid n \geq 1\} \notin \mathcal{L}_{\text{PERM}}(PA)
\]

On the other hand:

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

Systems with permutation mappings

Let us investigate the **power** systems with **permutation** mappings.
The power of P automata with permutation mapping

\[ \mathcal{L}_X(\text{PA}, f_{\text{perm}}) \subseteq r1\text{LOGSPACE} = \mathcal{L}_{\text{seq}}(\text{PA}, \text{TRANS}) \]

where \( X \in \{\text{seq, par}\} \).

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

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

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

The proof idea of...

... the inclusion:

\[ \mathcal{L}_X(\text{PA}, f_{\text{perm}}) \subset \mathcal{L}(\text{RCMA}) = \text{r1LOGSPACE} \]

where \( X \in \{ \text{seq, par} \} \).
Restricted $k$-counter machine acceptor (RCMA):

- A nondeterministic machine with a one-way read-only input tape and $k$ counters, which can be incremented, decremented and checked for zero.

- The reading head can read more than one input symbols in one computational step.

- The sum of the values stored in the counters can increase in one step at most as much as the number of symbols read.
A counter machine model

\[ \mathcal{L}_X (PA, f_{perm}) = \mathcal{L}(SRCMA), \ X \in \{seq, par\} \]

Special restricted $k$-counter machine acceptor (SRCMA):

- A nondeterministic machine with a one-way read-only input tape and $k$ counters, which can be incremented, decremented and checked for zero.
- The reading head can read more than one input symbols in one computational step.
- The sum of the values stored in the counters can increase in one step at most as much as the number of symbols read minus one.
An infinite hierarchy of P automata languages

\[ \mathcal{L}(\text{SRCMA}, k) \subset \mathcal{L}(\text{SRCMA}, 2k + 4), \quad k \geq 1 \]

[Csuhaj-Varjú, Vaszil 2013]

- As a consequence of the SRCMA hierarchy:

For every \( r \), there is an \( s > r \), and a language \( L \), such that \( L \) is accepted by a P automaton with permutation mapping and \( s \) membranes, but not with \( r \) membranes (seq, par).
The power of P automata, general formulation

**Notation:** for $S : \mathbb{N} \rightarrow \mathbb{N},$

$L \in \text{NSPACE}(S)$— as usual

$L \in r1\text{NSPACE}(S)$— 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
Let $\Pi$ be a $P$ automaton, and let $S : \mathbb{N} \rightarrow \mathbb{N}$, 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 $L(\Pi, f) \in r1NSPACE(\log(S) + S_f)$. 
Variants of P automata and their power:

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 $Z \in \{z, \neg z\}$, $x, y, z \in V^*$

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

$$f_2 : \{a\}^* \rightarrow 2^{T^*} \text{ with } f_2(a, \ldots, a) = \{a_i\}$$
A language $L$ is regular if and only if there is a $P$ finite automaton $\Pi$ 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.

→ an automata-like device over infinite alphabets

→ P finite automaton - regular languages over infinite alphabets
P finite automata over infinite alphabets

\[ f_2 : \{a\}^* \rightarrow 2^{T^*} \text{ with } f_2(a, \ldots, a) = \{a_i\} \]

\[ T = \{a_1, a_2, \ldots, a_i, \ldots\} \leftrightarrow \{\{a\}\}, \{\{a, a\}\}, \ldots, \{\{a, \ldots, a\}\}, \ldots \]
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 $\Delta$-regular expressions
Finite memory automata

An accepted string: $a_1a_2a_1a_3a_2a_4a_3a_4$

initialization
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 – \( \Delta \) regular expressions

Let \( \Delta \) be an infinite alphabet.

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

"\( L \) is regular if described by a \( \Delta \)-regular expression"

Checking relationships between symbols is possible, but:

\[ \{a_ia_i \mid i \geq 1\} \]

cannot be characterized this way
Finite memory automata and $\Delta$ regular expressions

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

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

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

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

$L_2$ is also easily described by FMA
P finite automata for $L_1$

\[ L = \{ f(a^{2i}) \mid i \geq 1 \} = \{ a^{2i} \mid i \geq 1 \} \]
P finite automata for $L_2$

$$L(T) = \{ (a \cdots a)(a \cdots a) \mid i \geq 1 \} = \{ a^i a^i \mid i \geq 1 \}$$
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.
Distributed P automata
Examples of parallelism in P systems

The use **maximal parallel** way of rule application for “zero-check”. A register machine-like instruction: 

\[(l_1, R_i-, l_2, l_3)\]

or

\[(l_1, l_1', l_1''', in)\]

\[(l_1', A, out; l_1'', in) (l_1'', out; l_1^{iv}, in)\]

\[(l_1^{iv}, l_1'''', out; l_2, in)\]

\[(l_1'l_1^{iv}, out; l_3, in)\]
Examples of parallelism in P systems

Creating **exponential** workspace in **linear** time: \([a_i] \rightarrow [t_i][f_i]\)

\[
\begin{align*}
& a_1a_2 \ldots a_n \\
& t_1a_2 \ldots a_n \\
& f_1a_2 \ldots a_n \\
& t_1t_2 \ldots a_n \\
& f_1t_2 \ldots a_n \\
& f_1f_2 \ldots a_n \\
& t_1t_2 \ldots t_n \\
\end{align*}
\]

\[
\begin{align*}
& t_1f_2 \ldots a_n \\
& f_1f_2 \ldots a_n \\
& \vdots \\
& f_1f_2 \ldots f_n \\
\end{align*}
\]
Examples of parallelism in P systems

We have seen:

- **maximal parallel** rule application
- the creation of **exponential** workspace in **linear** time

The **system** is still viewed as **one processing unit** and the whole input is given to it.
Distributed P systems

The idea behind distributed P systems is different:

- The system is composed of several processing units or components
- The components process (different parts of) the input in parallel
- The components communicate with each other
Distributed P automata languages

\[ \Delta = (V, \Pi_1, \ldots, \Pi_n, R), \]

where \( \Pi_i \) are P automata, and \( R \) is an infinite set of inter-component communication rules:

\[ z_i \mid (s_i, u/v, s_j) \mid z_j, \quad 1 \leq i, j \leq n, \quad i \neq j, \quad uv \in V^+ \]

1. The accepted concatenated language:

\[
L_{concat}(\Delta, f, \Sigma) = \{w_1 \ldots w_n \in \Sigma^* \mid w_i = f_i(v_{i,1}) \ldots f_i(v_{i,s_i}) \text{ and } \\
\alpha_i = v_{i,1} \ldots v_{i,s_i}, \quad 1 \leq i \leq n, \text{ for an n-tuple of accepted multiset sequences } (\alpha_1, \ldots, \alpha_n)\}.
\]

Possible questions concerning concatenation languages

- Is it possible to **split the input** into pieces?

- Is the distributed computation **more efficient** than the non-distributed one?
Distributed P automata:

Concatenated languages, efficiency of parallelization
Concatenated languages

- The input is a **string**:

  splitting the input $\leftrightarrow$ cutting the string into pieces

- The **efficiency** of the distributed computation:

  the number of **computational** steps / time

  +

  the amount of **communication**
Distributed P automata with $f_{\text{perm}}$

- There are simple context-sensitive languages which cannot be characterized with $f_{\text{perm}}$.

\[ \mathcal{L}_{\text{concat}}(dPA, f_{\text{perm}}) \subset CS \]

For example:

\[ L = \{ (wf(w))^n \mid w \in \{a, b\}^*, n \geq 2 \} \notin \mathcal{L}_{\text{concat}}(dPA, f_{\text{perm}}) \]

where \[ f(a) = a', f(b) = b' \]
Distributed P automata with $f_{perm}$

On the other hand: $\mathcal{L}_{concat}(dPA, f_{perm}) = CS$
The parallelizability of languages

A language is \((k, l, m)\)-efficiently parallelizable with respect to a class of input mappings \(F\) for some \(k, m > 1, l \geq 1\),

- \(L\) can be accepted with balanced computations of a \(dP\) automaton \(d\Pi\) with \(k\) components such that it is \(L(d\Pi, f)\) where \(f \in F\) and \(\text{Com}(d\Pi) \leq l\),

- for all (non-distributed) \(P\) automata \(\Pi\) and input mapping \(f' \in F\) such that \(L = L(\Pi, f')\), we have

\[
\lim_{x \in L, |x| \to \infty} \frac{\text{time}_{\Pi}(x)}{\text{time}_{d\Pi}(x)} \geq m
\]
The parallelizability of languages

A language is \((k, l, m)\) efficiently parallelizable with respect to a class of input mappings \(F\) for some \(k, m > 1, l \geq 1\), if it can be accepted by a dP automaton with \(k\) components, such that the dP automaton uses a finite amount of communication while being \(m\) times faster than any non-distributed P automaton which accepts \(L\) with any input mapping from the class \(F\).
The parallelizability of REGular languages

1. All regular languages can be accepted by balanced computations of some dP automaton.

For transduction mappings:

\[ A \rightarrow aB \]

\[ \begin{align*}
S &\rightarrow (A, \text{out}; aB, \text{in}) \\
&\rightarrow ((X, B), \text{out}; c(X, D), \text{in})
\end{align*} \]

final conf.: \( (X, \varepsilon) \)

\[ \begin{align*}
X &\rightarrow aB \\
&\rightarrow cD
\end{align*} \]

\[ \begin{align*}
&\rightarrow (\$, \text{out}; a(X, B), \text{in}) \\
&\rightarrow ((X, B), \text{out}; c(X, D), \text{in})
\end{align*} \]

final conf.: \( X \)

\[ f_1(aA) = \{a\} \]

\[ f_2(a(B, C)) = \{a\} \]
1. All **regular languages** can be accepted by balanced computations of some dP automaton.

For permutation mappings:

Example 3 can also be modified based on the same idea for a dP system with 2 components and $f_1 = f_2 = f_{\text{PERM}}$
The parallelizability of REG

1. All regular languages can be accepted by balanced computations of some dP automaton.

   We can use input mappings of any type.

2. What about the efficiency of parallelization?
The parallelizability of regular languages

**Efficiency** with respect to $f_{\text{PERM}}$

There are $(k, l, m)$-efficiently parallelizable regular languages with respect to $f_{\text{PERM}}$

This holds because there are regular languages where the order of no letters can be exchanged.

→ **No two letters** can be read in the same step.
   → There is **no P automaton** which needs **less steps** than the **number of letters**.

The parallelizability of regular languages

Efficiency with respect to input mappings

For any regular $L$ and $c > 0$ there exists a P automaton $\Pi$ such that $L = L(\Pi, f)$, $f \in TRANS$ and for any $w \in L$ with $|w| = n$, it holds that $time_\Pi(w) \leq c \cdot n$.

Take the finite automaton $M = (T, Q, q_0, \delta, F)$ with $L = L(M)$ it needs $n$ steps.

Let $\Pi' = (Q \cup T \cup T', [\quad]_2, 1, P'_1, P'_2, c_0, F)$, $T' = \{ab \mid a, b \in T\}$ and $P'_1 = \{(q''ab, in; q, out) \mid q'' \in \delta(q', b) \text{ for some } q' \in \delta(q, a)\} \cup \{(q'a, in; q, out) \mid q' \in \delta(q, a), \ q' \in F\}$,

$P'_2 = \{(a, in) \mid a \in T \cup T'\}$.

with $f'(qa) = a$ and $f'(qab) = ab$
The parallelizability of regular languages

Efficiency with respect to $TRANS$

There are no $(k, l, m)$-efficiently parallelizable regular languages with respect to $TRANS$.

This holds because with input mappings from $TRANS$ there is no “fastest” (non-distributed) P automata for a regular language.

Speedup with a linear factor is always possible which is a “problem” if we need to satisfy

$$\lim_{x \in L, |x| \to \infty} \frac{\text{time}_\Pi(x)}{\text{time}_d\Pi(x)} \geq m$$
Distributed P automata:

Agreement languages, multihead automata, two-way P automata
Distributed P automata languages

2. The accepted **weak agreement** language:

\[ L_{w, \text{agree}}(\Delta, f, \Sigma) = \{ w \in \Sigma^* \mid w = f_i(v_{i,1}) \ldots f_i(v_{i,s_i}) = f_j(v_{j,1}) \ldots f_j(v_{j,s_j}) \]

for all 1 \leq i, j \leq n, where \( \alpha_i = v_{i,1} \ldots v_{i,s_i}, 1 \leq i \leq n \) and \((\alpha_1, \ldots, \alpha_n)\) is an n-tuple of accepted multiset sequences of \(\Delta\).

3. The accepted **strong agreement** language:

\[ L_{s, \text{agree}}(\Delta, g, \Sigma) = \{ w \in \Sigma^* \mid w = g(v_1) \ldots g(v_s) \text{ and } \alpha = v_1 \ldots v_s, \text{ for an} \]

n-tuple of accepted multiset sequences \((\alpha, \ldots, \alpha)\) of \(\Delta\).
A variant of dP automata

A dP automaton $\Delta$ is called finite, if the number of configurations reachable from its initial configuration is finite.

The power of distributed (finite) P automata

- Languages accepted by one- and two-way k-head finite automata can be characterized by the agreement languages of finite dP automata.

Multi-head finite automaton

3-head finite automaton
Non-deterministic one-way k-head finite automaton (a 1NFA(k))

\[ M = (Q, \Sigma, k, \delta, \triangleright, \triangleleft, q_0, F) \]

movement in one direction
Non-deterministic two-way k-head finite automaton (a 2NFA(k))

\[ M = (Q, \Sigma, k, \delta, \triangleright, \triangleleft, q_0, F) \]

Movement in two directions
One-way multi-head finite automata vs. finite dP automata

The weak agreement language of a finite dP automaton (with respect to the mapping $f_{perm}$) is equal to the language of a one-way multi-head finite automaton.

Theorem

For any finite dP automaton $\Delta = (O, \Pi_1, \ldots, \Pi_k, R, \mathcal{F})$, $k \geq 2$, a non-deterministic one-way $k$-head finite automaton $M = (Q, O, k, \delta, \triangleright, \triangleleft, q_0, F)$ can be constructed such that $L_{w,agree}(\Delta, f_{perm}, O) = L(M)$. 
One-way multi-head finite automata vs. finite dP automata

The language of any one-way finite multi-head automaton can be obtained as the strong or weak agreement language of a finite dP automaton with respect to the mapping $f_{\text{perm}}$.

**Theorem**

For any non-deterministic one-way $k$-head finite automaton $M$, $k \geq 2$, with input alphabet $\Sigma$ we can construct a dP automaton $\Delta$ of degree $k$, such that

$$L(M) = L_{\text{s,agree}}(\Delta, f_{\text{perm}}, \Sigma) = L_{\text{w,agree}}(\Delta, f_{\text{perm}}, \Sigma).$$
Two-way multi-head finite automata vs. finite dP automata

We need the notion of a two-way dP automaton and we show how two-way finite dP automata characterize the language family accepted by non-deterministic two-way multi-head finite automata.

We first recall that alphabets of the form $\Sigma \cup \bar{\Sigma}$, where $\Sigma$ is an alphabet itself and $\bar{\Sigma} = \{\bar{a} \mid a \in \Sigma\}$ are called double alphabets.
Two-way trail

\[ w = w_1 \bar{x}_1 w_2 x_2 w_3 \bar{x}_3 w_4 x_4 \ldots \bar{x}_{n-2} w_{n-1} x_{n-1} w_n \]

1. For all \( w' \) with \( w = w'w'' \), it holds that \( |w'|_\Sigma \geq |w'|_\Sigma \);
2. for all \( w'' \) with \( w = w'w'' \), it holds that \( |w'|_\Sigma \geq 2 \cdot |w'|_\Sigma \);
3. for all \( 1 \leq i \leq n - 2 \), the subwords \( w_i \bar{x}_i \bar{w}_{i+1} \) are of the form \( w'_i x \bar{x}_i \bar{x}w'_{i+1} \)
   where \( w_i = w'_ix \), \( \bar{w}_{i+1} = \bar{x}w'_{i+1} \), and the subwords \( \bar{w}_{i+1} x_{i+1} w_{i+2} \) are of the
   form \( \bar{w}'_{i+1} x x_{i+1} x w'_{i+2} \) where \( \bar{w}_{i+1} = \bar{w}'_{i+1} \bar{x} \), \( w_{i+2} = x w'_{i+2} \);
4. for all \( 1 \leq i \leq n - 2 \), the subwords \( w_i \bar{x}_i \bar{w}_{i+1} \) satisfy one of the properties
   (a) \( \bar{w}_{i+1} = \bar{w}'_i \bar{w}'_{i+1} \), or (b) \( w_i = w^R_{i+1} w'_i \), while the subwords \( \bar{w}_{i+1} x_{i+1} w_{i+2} \)
   satisfy one of the properties (c) \( w_{i+2} = w^R_{i+1} w'_{i+2} \), or (d) \( \bar{w}_{i+1} = \bar{w}'_{i+2} \bar{w}'_{i+1} \),
   depending on the length of \( w_i \), \( w_{i+1} \), and \( w_{i+2} \).
Two-way trails and 2NFA(k)

Using the notion of the two-way trail, it is easy to see that a computation in a 2NFA$(k)$ $M = (Q, \Sigma, k, \delta, \triangleright, \triangleleft, q_0, F)$ is accepting if and only if it is given by a sequence states $q_0, q_1, \ldots, q_s$, $q_s = q_f \in F$, where $q_j \in \delta(q_{j-1}, x_{j-1,1}, \ldots, x_{j-1,k})$, $1 \leq j \leq s - 1$ such that $w = x_{1,i} \ldots x_{s,i} \in (\Sigma \cup \tilde{\Sigma})^*$ is a two-way trail and $x_{j,i} \in (\Sigma \cup \tilde{\Sigma} \cup \{\lambda\})$ for $1 \leq j \leq s$, $1 \leq i \leq k$.

The concept of a two-way trail can be extended to the concept of a two-way multiset trail in an obvious manner.
Two-way dP automaton

A dP automaton $\Delta = (O', \Pi_1, \ldots, \Pi_k, R, \mathcal{F})$ where $O' = O \cup \tilde{O}$ is a double alphabet is called a two-way dP automaton if any multiset $u_i$ which enters component $\Pi_i$, $1 \leq i \leq k$, in the course of a computation consists of either objects of $O$, or objects of $\tilde{O}$, or it is the empty multiset.

Obviously, if a two-way dP automaton is a finite dP automaton, then we speak of a two-way finite dP automaton.
Reduction mapping for two-way trails

\[ h : (\Sigma \cup \bar{\Sigma})^* \rightarrow 2^{(\Sigma \cup \bar{\Sigma})^*} \]

- \( w_1 \bar{x}_1 w_2 \in h(w) \) if and only if \( w = w_1 x_1 \bar{x}_2 \bar{x}_1 w_2 \), for some \( x_1, x_2 \in \Sigma \), or
- \( w_1 x_1 w_2 \in h(w) \) if and only if \( w = w_1 \bar{x}_1 x_2 x_1 w_2 \), for some \( x_1, x_2 \in \Sigma \), or
- \( h(w) = w \) if and only if \( w \in \Sigma^* \), or
- let \( h(w) \) be undefined otherwise.

Let us define \( h^0(w) = h(w) \), \( h^i(w) = h(h^{i-1}(w)) \) for \( i \geq 0 \), and let \( h^*(w) = h^i(w) \) for some \( i \), such that \( h^i(w) = h^{i+1}(w) \).
Languages of two-way dP automata

**Strong agreement language**

\[ L_{s,agreel}(\Delta, f, \Sigma) = \{ w \in \Sigma^* \mid w = g(v_1) \ldots g(v_s), \alpha = v_1 \ldots v_s, \] 
where \( v_j \in O^*(\Sigma), 1 \leq j \leq s, \) and \( \alpha = h_m^*(\beta_i), \) 
for a \( k \)-tuple of accepted multiset sequences \( (\beta_1, \ldots, \beta_k), \)
where \( \beta_i \) is a two-way multiset trail, \( 1 \leq i \leq k \} \).

**Weak agreement language**

\[ L_{w,agreel}(\Delta, f, \Sigma) = \{ w \in \Sigma^* \mid w = h^*(u_i), 1 \leq i \leq k, u_i = g(v_{1,i}) \ldots g(v_{s_i,i}), \] 
is a two-way trail, \( \alpha_i = v_{1,i} \ldots v_{s_i,i}, s_i \geq 1, (\alpha_1, \ldots, \alpha_k) \) 
is a \( k \)-tuple of multiset sequences accepted by \( \Delta \}. \]
Results

Theorem

Any language that can be accepted by a non-deterministic two-way $k$-head finite automaton for $k \geq 2$ is equal to the strong or weak agreement language of a two-way finite dP automaton of degree $k$ with respect to the mapping $f_{perm}$.

Theorem

Any language which is the weak agreement language with respect to the mapping $f_{perm}$ of a two-way finite dP automaton of degree $k$, $k \geq 2$, can be accepted by a non-deterministic two-way $k$-head finite automaton.
Summary

• From accepting symport/antiport systems to P automata
  • Analyzing P systems, extended P automata, symport/antiport acceptors

• Variants of P automata and their power
  • The role of different input mappings
  • Complexity classes and equivalent counter automata
  • P automata over infinite alphabets

• Distributed P automata
  • Concatenated languages, efficiency of parallelization
  • Agreement languages, multihead automata, two-way distributed P automata