On a powerful class of non-universal P systems with active membranes

Antonio E. Porreca Alberto Leporati Claudio Zandron

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca, Italy

14th Developments in Language Theory
London, Ontario, 19 August 2010

Introduction

- Membrane systems (P systems) are devices inspired by structure and functioning of biological cells introduced by G. Păun in 1998

Introduction

- Membrane systems (P systems) are devices inspired by structure and functioning of biological cells introduced by G. Păun in 1998
- They can be used as language recognisers and most variants are computationally universal

Introduction

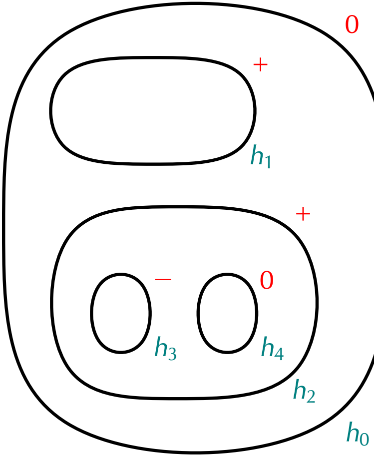
- Membrane systems (P systems) are devices inspired by structure and functioning of biological cells introduced by G. Păun in 1998
- They can be used as language recognisers and most variants are computationally universal
- We show a variant of P system that decides exactly the languages decided by Turing machines working in tetrational (iterated exponential) time and space

Outline

- Description of our variant of P systems
- Languages decidable in tetrational time
- How much space can P systems use? (upper bound)
- Simulating tetrational-space TMs (lower bound)
- Conclusions and open problems

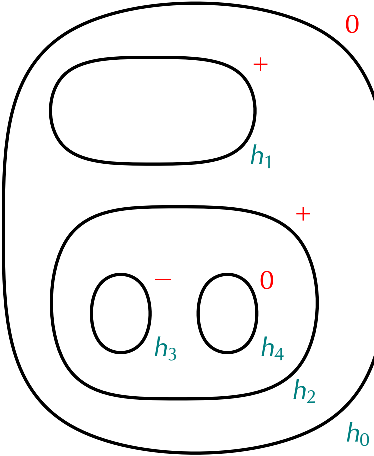
The membrane structure of P systems

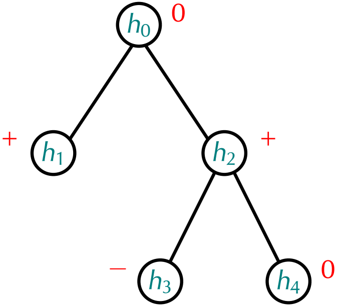
- Membranes divide the cell into regions
- Membranes have fixed label and changeable electrical charge



The membrane structure of P systems

- Membranes divide the cell into regions
- Membranes have fixed label and changeable electrical charge

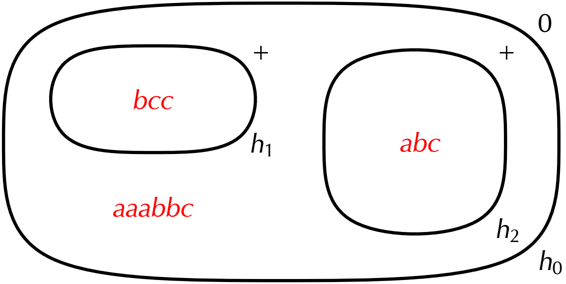




$$[\,[\,]^{+}_{h_1}[\,[\,]^{-}_{h_3}[\,]^{0}_{h_4}]^{+}_{h_2}]^{0}_{h_0}$$

Contents of the membranes

Each membrane contains a multiset of objects
(symbols from an alphabet Γ) representing molecules



$$[aaabbc\ [bcc]^+_{h_1} [abc]^+_{h_2}]^0_{h_0}$$

Evolution rules

- Each pair (*label*, *charge*) has a set of associated rules

Evolution rules

- Each pair (*label*, *charge*) has a set of associated rules
- For instance, the rule $[a]_h^0 \to [\ ]_h^+\ b$ can be applied to a membrane having label h and charge 0

Evolution rules

- Each pair (*label*, *charge*) has a set of associated rules
- For instance, the rule $[a]_h^0 \to [\ ]_h^+ \ b$ can be applied to a membrane having label h and charge 0
- The charge represents a kind of state of the membrane, controlling which set of rules can be applied

Evolution rules

- Each pair (*label*, *charge*) has a set of associated rules
- For instance, the rule $[a]_h^0 \to [\ ]_h^+ \ b$ can be applied to a membrane having label h and charge 0
- The charge represents a kind of state of the membrane, controlling which set of rules can be applied
- We only use three kinds of rule in this paper:

Evolution rules

- Each pair (*label*, *charge*) has a set of associated rules
- For instance, the rule $[a]_h^0 \to [\,]_h^+ \; b$ can be applied to a membrane having label h and charge 0
- The charge represents a kind of state of the membrane, controlling which set of rules can be applied
- We only use three kinds of rule in this paper:
  - Send-out communication rules: $[a]_h^\alpha \to [\,]_h^\beta \; b$

Evolution rules

- Each pair (*label*, *charge*) has a set of associated rules
- For instance, the rule $[a]_h^0 \to [\,]_h^+ \; b$ can be applied to a membrane having label h and charge 0
- The charge represents a kind of state of the membrane, controlling which set of rules can be applied
- We only use three kinds of rule in this paper:
  - Send-out communication rules: $[a]_h^\alpha \to [\,]_h^\beta \; b$
  - Send-in communication rules: $a \; [\,]_h^\alpha \to [b]_h^\beta$

Evolution rules

- Each pair (*label*, *charge*) has a set of associated rules
- For instance, the rule $[a]_h^0 \to [\,]_h^+ \, b$ can be applied to a membrane having label h and charge 0
- The charge represents a kind of state of the membrane, controlling which set of rules can be applied
- We only use three kinds of rule in this paper:
  - Send-out communication rules: $[a]_h^\alpha \to [\,]_h^\beta \, b$
  - Send-in communication rules: $a \, [\,]_h^\alpha \to [b]_h^\beta$
  - Nonelementary division rules

Nonelementary division

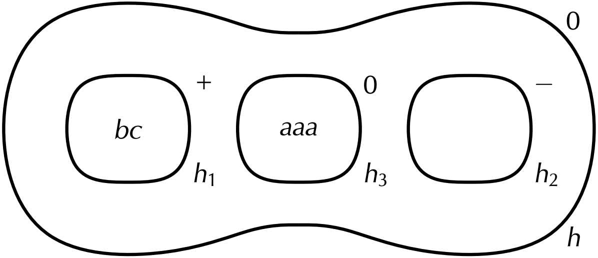
- A nonelementary division rule separates positive and negative children membranes

Nonelementary division

- A nonelementary division rule separates positive and negative children membranes
- The neutral children membranes are duplicated instead

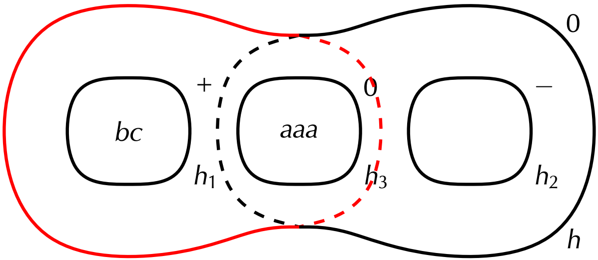
Nonelementary division

- A nonelementary division rule separates positive and negative children membranes
- The neutral children membranes are duplicated instead
- For instance: consider $[\,[\,]_{h_1}^{+}\,[\,]_{h_2}^{-}]_{h}^{0} \rightarrow [\,[\,]_{h_1}^{0}]_{h}^{+}\ [\,[\,]_{h_2}^{0}]_{h}^{-}$



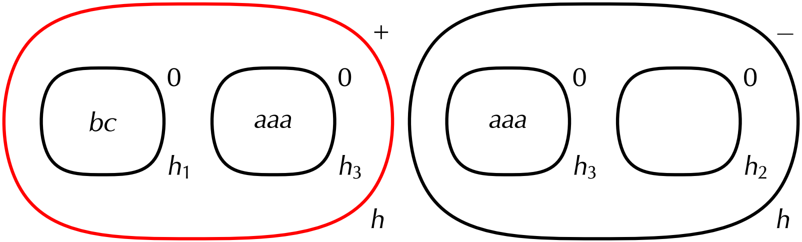
Nonelementary division

- A nonelementary division rule separates positive and negative children membranes
- The neutral children membranes are duplicated instead
- For instance: consider $[\,[\,]^{+}_{h_1}[\,]^{-}_{h_2}]^{0}_{h} \to [\,[\,]^{0}_{h_1}]^{+}_{h}\ [\,[\,]^{0}_{h_2}]^{-}_{h}$



Nonelementary division

- A nonelementary division rule separates positive and negative children membranes
- The neutral children membranes are duplicated instead
- For instance: consider $[\,[\,]^{+}_{h_1}\,[\,]^{-}_{h_2}]^{0}_{h} \rightarrow [\,[\,]^{0}_{h_1}]^{+}_{h}\;[\,[\,]^{0}_{h_2}]^{-}_{h}$



Recogniser P systems

- To decide a language L, we map each string x to a P system Π_x

Recogniser P systems

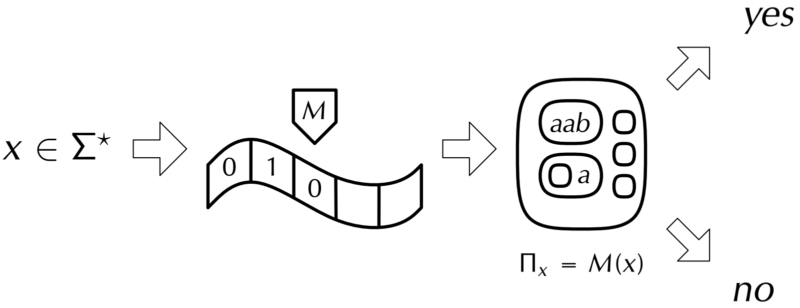
- To decide a language L, we map each string x to a P system Π_x
- Π_x decides $x \in L$ by emitting an object *yes* or *no* from the outermost membrane and halting

Recogniser P systems

- To decide a language L, we map each string x to a P system Π_x
- Π_x decides $x \in L$ by emitting an object *yes* or *no* from the outermost membrane and halting
- The map $x \mapsto \Pi_x$ is computed by a polytime Turing machine M

Recogniser P systems

- To decide a language L, we map each string x to a P system Π_x
- Π_x decides $x \in L$ by emitting an object *yes* or *no* from the outermost membrane and halting
- The map $x \mapsto \Pi_x$ is computed by a polytime Turing machine M



The complexity class **PTETRA**

Denote by ${}^n2$ the tetration (iterated exponentiation) operation

$$ {}^n2 = \underbrace{2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}}_{n \text{ times}} $$

The complexity class **PTETRA**

Denote by ${}^{n}2$ the tetration (iterated exponentiation) operation

$$ {}^{n}2 = \underbrace{2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}}_{n \text{ times}} $$

Definition

PTETRA is the class of languages decided by Turing machines operating in "tetrational time" ${}^{p(n)}2$ for some polynomial p

The complexity class **PTETRA**

Denote by ${}^{n}2$ the tetration (iterated exponentiation) operation

$$ {}^{n}2 = \underbrace{2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}}_{n \text{ times}} $$

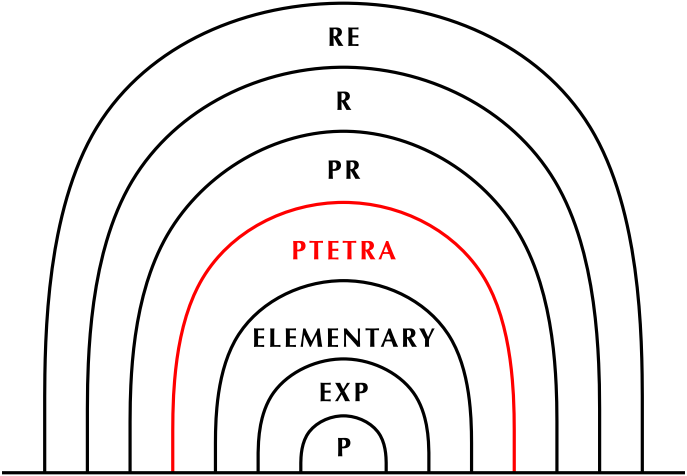
Definition

PTETRA is the class of languages decided by Turing machines operating in "tetrational time" ${}^{p(n)}2$ for some polynomial p

Proposition

PTETRA *coincides with tetrational space, and is closed under complement, union, intersection, and polytime reductions*

How large is **PTETRA**?



All inclusions in this Venn diagram are proper

Space required to simulate P systems

Fact

A P system can be simulated by a TM using the same amount of space (number of membranes and objects) modulo a polynomial overhead

Space required to simulate P systems

Fact

A P system can be simulated by a TM using the same amount of space (number of membranes and objects) modulo a polynomial overhead

- The number of objects in our P systems only increases via nonelementary membrane division

Space required to simulate P systems

Fact

A P system can be simulated by a TM using the same amount of space (number of membranes and objects) modulo a polynomial overhead

- The number of objects in our P systems only increases via nonelementary membrane division
- We need to count the maximum number of membranes during the computation…

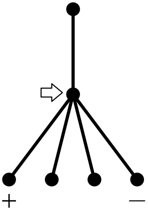
Space required to simulate P systems

Fact

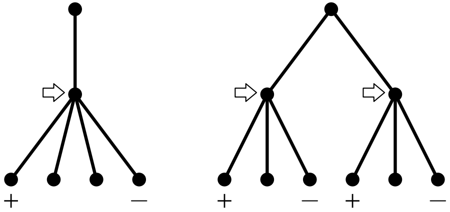
A P system can be simulated by a TM using the same amount of space (number of membranes and objects) modulo a polynomial overhead

- The number of objects in our P systems only increases via nonelementary membrane division
- We need to count the maximum number of membranes during the computation...
- ... keeping in mind that the initial number is $m = \text{poly}(n)$

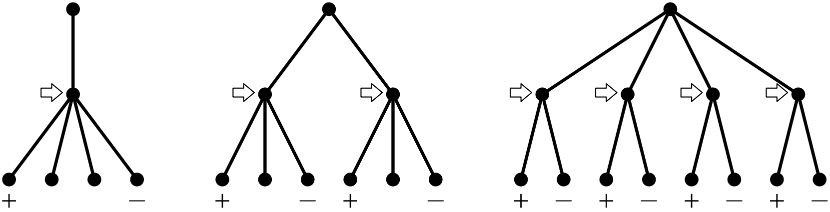
Nonelementary division stops in a finite number of steps



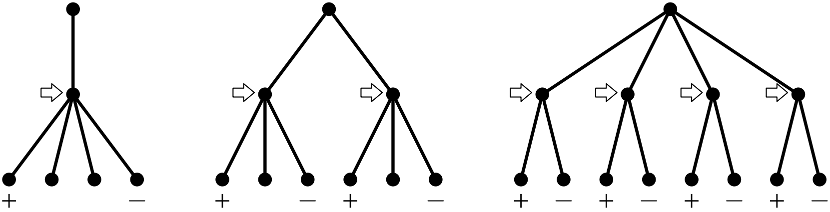
Nonelementary division stops in a finite number of steps

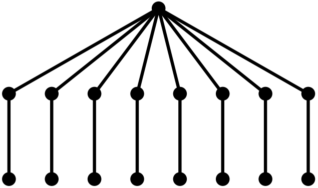


Nonelementary division stops in a finite number of steps



Nonelementary division stops in a finite number of steps





no further division
is possible here

Applying division to all levels

Proposition

Suppose we apply all possible division rules to the complete m-ary tree of m levels, and that the rules are applied level-by-level, bottom-up (this is the worst case). Then the final number of nodes is bounded by $2^{O(m^2)}$

Applying division to all levels

Proposition

Suppose we apply all possible division rules to the complete m-ary tree of m levels, and that the rules are applied level-by-level, bottom-up (this is the worst case). Then the final number of nodes is bounded by $2^{O(m^2)}$

Proposition

A family of P system $\{\Pi_x : x \in \Sigma^\star\}$ can be simulated by a Turing machine in space $2^{p(n)} \cdot p(n)$ for some polynomial p

Applying division to all levels

Proposition

Suppose we apply all possible division rules to the complete m-ary tree of m levels, and that the rules are applied level-by-level, bottom-up (this is the worst case). Then the final number of nodes is bounded by ${}^{O(m^2)}2$

Proposition

A family of P system $\{\Pi_x : x \in \Sigma^\star\}$ *can be simulated by a Turing machine in space* ${}^{p(n)}2 \cdot p(n)$ *for some polynomial p*

Theorem

The class of languages decidable by this variant of P systems is a subset of **PTETRA**

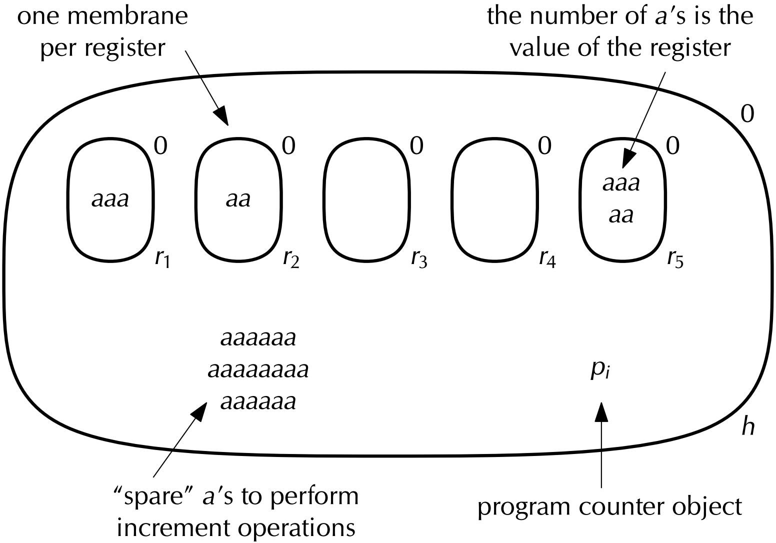
Can the upper bound be actually achieved?

- By using a few auxiliary membranes and objects...
- ... and forcing the divisions to occurr in a bottom-up order by using the electrical charges...
- ... we can produce tetrationally many membranes and, as a consequence, tetrationally many objects

Can the upper bound be actually achieved?

- By using a few auxiliary membranes and objects…
- … and forcing the divisions to occurr in a bottom-up order by using the electrical charges…
- … we can produce tetrationally many membranes and, as a consequence, tetrationally many objects
- How can we exploit this feature?

Simulating register (counter) machines



Solving **PTETRA** problems via P systems

By exploiting the nonelementary division process
we can produce tetrationally many spare a's

Solving **PTETRA** problems via P systems

By exploiting the nonelementary division process
we can produce tetrationally many spare a's

Proposition

Tetrational-space register machines (hence TMs) can be simulated by our variant of P system

Solving **PTETRA** problems via P systems

By exploiting the nonelementary division process
we can produce tetrationally many spare a's

Proposition

Tetrational-space register machines (hence TMs) can be simulated by our variant of P system

Theorem

The class of languages decided by our variant of P system is exactly **PTETRA**

Conclusions and open problems

- We described a variant of P systems that, although non computationally universal, decides a very large class of languages

Conclusions and open problems

- We described a variant of P systems that, although non computationally universal, decides a very large class of languages
- Do other nature-inspired computing devices characterise **PTETRA** or other similarly large classes?

Conclusions and open problems

- We described a variant of P systems that, although non computationally universal, decides a very large class of languages
- Do other nature-inspired computing devices characterise **PTETRA** or other similarly large classes?
- Can we characterise the primitive recursive languages?

Thanks for your attention!