ACADEMY OF SCIENCES OF MOLDOVA

INSTITUTE OF MATHEMATICS AND COMPUTER SCIENCE

By way of manuscript
U.D.C: 510.58+519.712.5

ALHAZOV Artiom

Small Abstract Computers

SPECIALTY: 01.05.01

COMPUTER SCIENCE FUNDAMENTALS;
COMPUTER PROGRAMMING

Habilitation Thesis in Computer Science

Scientific Consultant: ..... Yurii ROGOZHIN
Ph.D. in Habilitation,
Principal Scientific Researcher

Author: .....

CHIŞINĂU, 2013

ACADEMIA DE ŞTIINŢE A MOLDOVEI

INSTITUTUL DE MATEMATICĂ ŞI INFORMATICĂ

Cu titlu de manuscris
C.Z.U: 510.58+519.712.5

ALHAZOV Artiom

Small Abstract Computers (Maşini Abstracte Mici)

SPECIALITATEA: 01.05.01

BAZELE TEORETICE ALE INFORMATICII; PROGRAMAREA CALCULATOARELOR

Teza de Doctor Habilitat in Informatică

Consultant Ştiinţific: ..... Iurie ROGOJIN
Doctor Habilitat,
Cercetător Ştiinţific Principal

Autorul: .....

CHIŞINĂU, 2013

Acknowledgements

I would like to thank my family for making this work possible and for their precious support, in particular, my parents for sharing their academic experience and for encouraging all my efforts. I would like to express my gratitude to all my co-authors for their ideas, efforts and time spent on producing and describing results we have obtained. I thank Chişinău, Turku, Higashi Hiroshima and Milano for a working environment, and Prof. Ion Petre, Prof. Kenichi Morita, Dr. Katsunobu Imai, Dr. Marco Antoniotti, Prof. Giancarlo Mauri and Dr. Alberto Leporati for making my postdocs possible and so fruitful (and for showing me how to be patient), and computer technology to facilitate my research in Theoretical Computer Science.

The author[1,2] is very much indebted to Prof. Yurii Rogozhin, who brought me to the field of theoretical computer science and continuously provided his expertise, resulting also in (but by far not only) over 60 joint publications, and to Dr. hab. Sergey Verlan for valuable advices in research and life, in particular yielding over 25 joint works. I thank Dr. Rudolf Freund, an incredible colleague who coauthored over 30 of my publications. I thank Dr. Vladimir Rogojin and Sergiu Ivanov for long and exciting hours of fruitful joint research. Many thanks are addressed to Acad. Gheorghe Păun for research experience and writing skills and for having started membrane computing. I thank the professors and colleagues of the State University of Moldova, for having contributed much to my mathematical education.

A special acknowledgment is due to the management of IMI, and in particular to Prof. Svetlana Cojocaru, Dr. Constantin Ciubotaru, Corr. memb. Constantin Gaindric, for their assistance through the years, and to the present and former workers of IMI (and especially to the Laboratory of Programming Systems for being wonderful work colleagues), and the whole P systems community, for their moral support.

[1] Institute of Mathematics and Computer Science (IMI), Academy of Sciences of Moldova. Str. Academiei 5, Chişinău, MD 2028, Moldova. E-mail: artiom@math.md

[2] This work was facilitated by the support of project IST-2001-32008 "MolCoNet" from the European Commission under FP5 programme, project MM2-3034 from the Moldovan Research and Development Association (MRDA) and the U.S. Civilian Research and Development Foundation (CRDF), project 203667 from the Academy of Finland, project and Grant-in-Aid for Scientific Research 20·08364 from the Japan Society for the Promotion of Science, project RetroNet from the Lombardy Region of Italy under the ASTIL Program (regional decree 6119, 20100618), projects STCU-4032 "Power and efficiency of natural computing: neural-like P (membrane) systems" and STCU-5384 "Models of high performance computations based on biological and quantum approaches" from the Science and Technology Center in Ukraine, and projects 06.411.03.04P and 12.819.18.09A "Development of IT support for interoperability of electronic linguistic resources" from the Supreme Council for Science and Technological Development, Academy of Sciences of Moldova.

Contents

Annotations

Adnotare la teza de doctor habilitat "**Small Abstract Computers**" (Maşini Abstracte Mici), prezentată de către Artiom Alhazov pentru obţinerea titlului de *doctor habilitat* în informatică la specialitatea 01.05.01 – Bazele teoretice ale informaticii; programarea calculatoarelor. Teza a fost perfectată în cadrul Institutului de Matematică şi Informatică al Academiei de Ştiinţe a Republicii Moldova, în anul 2013; este scrisă în limba engleză şi constă din Introducere, 6 capitole, concluzii generale şi recomandări şi 8 anexe. Textul de bază constituie 219 de pagini. Bibliografia constă din 291 de titluri. Teza include: 25 de figuri, 9 tabele, 22 de definiţii, 19 leme, 77 de teoreme, 47 de consecinţe, 9 remarci, 24 de exemple şi 54 de formule. Rezultatele obţinute sînt publicate în 120 de lucrări ştiinţifice.

Cuvintele-cheie: informatică teoretică, modele de calcul neconvenţional, computabilitate Turing, complexitate descriptivă, sisteme universale mici, P sisteme, procesare paralelă distribuită de şiruri/multiseturi, promotori/inhibitori/priorităţi, membrane active, polarizări, symport/antiport, determinism, reversibilitate, inserţie-deleţie-substituire, reţele hibride de procesoare evolutive, paralelism maxim şi minim.

Domeniul de studiu îl constituie sistemele membranare şi alte modele formale de calcul, în special *procesarea distribuită şi paralelă de multiseturi/şiruri*, de ex., reţele de procesoare evolutive, sisteme IDP, CPM, etc.

De rînd cu **scopul principal** de determinare a puterii de calcul a modelelor restrînse, lucrarea îşi propune şi următoarele **objective**: depistarea solvabilităţii problemelor în timp polinomial cu modele restrînse, cercetarea complexităţii temporale ale problemelor în dependenţă de caracteristicile modelelor. Restricţii tipice: complexitatea descriptivă mărginită, modalităţile de interacţiune ale obiectelor, o submulţime de obiecte cu caracteristici restrînse, proprietăţi (de ex., determinism, reversibilitate, auto-stabilizare, confluenţă).

Noutatea ştiinţifică. *Noţiuni noi*: derivări temporale în CFG, limitarea multiseturilor, determinism/reversibilitate tare, auto-stabilizare în P sisteme, noi reguli cu membrane active, OHNEP, NCPM5, polimorfism. *Strategii noi*: reprezentarea datelor prin membrane, minimizarea avansată a regulilor, noi combinaţii ale parametrilor complexităţii descriptive, nedeterminismul extrem în membrane active, un nou mod de interacţiune membrană-obiect, mai mult de 3 polarizări, o nouă tehnică de control regulat pentru operaţii pe 1 simbol din şir. *Probleme/modele noi*: probleme nedecizionale computaţional dificile, flexionarea cunivntelor în limba Română, module de dicţionar.

Problemele ştiinţifice soluţionate includ: 1)Stabilirea puterii de calcul a P sistemelor a)tranziţionale cu creare şi dizolvare de membrane, b)cu membrane active fără polarizări, c)deterministe controlate necooperatiste, d)cu energie. 2)Caracterizarea a)clasei problemelor rezolvabile în timp polinomial cu P sisteme cu membrane active fără polarizări, b)puterii exacte a HNEPs cu 1 nod.

Rezultatele principale. *Universalitate*: FsMPMRS cu 23 de reguli, P sisteme tranziţionale cu creare/dizolvare de membrane, AM_0, IDP cu priorităţi. *Tablou de rezultate*: determinism/reversibilitate, NEP, HNEP, OHNEP, rezolvarea problemelor PSPACE cu AM_0. *Caracterizări*: sisteme deterministe necooperatiste controlate, sisteme cu energie.

Semnificaţia teoretică. *Probleme fundamentale* de procesare distribuită paralelă de multiseturi/şiruri. *Cele mai bune rezultate* obţinute la moment pentru $LOP(ncoo, tar)$, reguli în MPMRS, $NOP_1(sym_3)$, noduri în HNEP, polarizări în P sisteme *minpar* eficiente, sincronizare. *Caracterizări importante*: $DMR(ncoo, control)$. *Proprietăţi fundamentale* ale MR: variante ale determinismului, reversibilităţii, auto-stabilizării, pentru procesarea multiseturilor cu diverse caracteristici/moduri. Completitudinea de calcul a sistemelor cu *cooperare slabă* între elemente: $OP(ncoo, tar, \delta, mcre)$ şi $OP(a_0, b_0, c_0, d_0, e_0)$. *Rezultate optime* pentru probleme cunoscute formulate pentru: P sisteme cu symport/antiport, P sisteme cu AM_0, noduri în NEP computaţional complete.

Aplicaţii: Polimorfism. Module de dicţionare. Flexionarea şi marcarea afixelor în cuvintele limbii române. **Impact** OHNEP - o direcţie de cercetare de perspectivă, IDP - dezvoltare importantă în continuare, AM_0 - salt în teoria complexităţii P sistemelor.

Аннотация диссертации доктора хабилитат "**Small Abstract Computers**" (Малые универсальные вычислители), представленной Артёмом Алхазовым на соискание учёной степени доктора наук (*доктора хабилитат*) в информатике по специальности 01.05.01 – Теоретические основы информатики; компьютерное программирование. Диссертация написана в Институте математики и информатики при Академии наук Молдовы (Кишинёв) в 2013 году на английском языке и состоит из: введения, 6 глав, общих заключений и рекомендаций, библиографии из 291 наименований и 8 приложений. Основной текст насчитывает 219 страниц. Работа содержит: 25 рисунков, 9 таблиц, 22 определения, 19 лемм, 77 теорем, 47 следствий, 9 замечаний, 24 примеров, и 54 формул. Полученные результаты опубликованы в 120 научных статьях.

Ключевые слова: теоретическая информатика и неклассические вычисления, вычислимость по Тьюрингу, вычислительная сложность и малые универсальные системы, P системы как параллельные дистрибутивные системы обработки многомножеств и строк, промоторы/ингибиторы и приоритеты, активные мембраны и поляризации, симпорт/антипорт, детерминизм/обратимость, удаление-вставка-подстановка и [гибридные] сети эволюционных процессоров, максимальный и минимальный параллелизм и асинхронный режим. **Область** исследований: мембранные системы и другие формальные модели вычислений, в основном *дистрибутивной параллельной перезаписи многомножеств и строк*: сети эволюционных процессоров, системы IDP, CPM, и пр.

Помимо **основной цели** определения вычислительной силы ограниченных вычислительных моделей, наши **задачи** – нахождение разрешимости задач в полиномиальное время и временно́й сложности задач в зависимости от ингредиентов систем. Наиболее типичны такие ограничения как: ограниченная описательная сложность, способ взаимодействия объектов, подмножество объектов с ограниченными возможностями, свойства (такие как детерминизм, обратимость, само-стабилизация, конфлуэнтность).

Научная новизна. *Ввели*: временно́й результат CFG, сильный детерминизм/обратимость, само-стабилизацию в P системах, новые правила для активных мембран, OHNEP, NCPM5, полиморфизм. *Новые стратегии*: представление данных мембранами, сложные методы минимизации правил, новые комбинации параметров описательной сложности, крайний недетерминизм с активными мембранами, новая тактика взаимодействия мембрана-объект, > 3 поляризаций, новая техника регулярного управления строковыми операциями с 1 символом. *Новые задачи/модели*: сложные небинарные вычислительные задачи, флексии румынского языка, словарные модули.

Решённые задачи в работе включают: 1)Нахождение вычислительной мощности P систем a)транзициональных с созданием и расствореним мембран, b)с активными мембранами без поляризаций, c)детерминированных некооперативных, d)с энергией. 2)Вычисление a)класса задач разрешимых в полиномиальное время P системами с активными мембранами без поляризаций, b)точной мощности HNEPs с 1 узлом. **Главные результаты**. *Универсальность*: FsMPMRS с 23 правилами, P системы с созданием/расствореним мембран, AM_0, IDP с приоритетами. *Картины результатов*: детерминизм/обратимость, NEP, HNEP, OHNEP, решение PSPACE-задач с AM_0. *Характеризации*: детерминированных управляемых некооперативных систем, систем с энергией. **Теоретическая значимость**. *Фундаментальные задачи* дистрибутивной параллельной обработки многомножеств/строк. *Лучшие известные результаты* для LOP $(ncoo, tar)$, количества правил в MPMRS, $NOP_1(sym_3)$, узлов в HNEP, поляризаций эффективных P систем с $minpar$, синхронизация. *Важная характеризация*: DMR $(ncoo, control)$. *фундаментальные свойства* MR: виды детерминизма, обратимости, само-стабилизации, для обработки многомножеств с разными особенностями/режимами. Вычислительная полнота систем со *слабыми взаимодействиями* их элементов: $OP(ncoo, tar, \delta, mcre)$ и $OP(a_0, b_0, c_0, d_0, e_0)$. *Оптимальные результаты* известных задач для: P систем с симпортом/антипортом, с AM_0, узлы в вычислительно полных NEP. **Применения**: полиморфизм, словарные модули, флексии и аннотирование аффиксов в румынском языке. **Влияние** OHNEP: новое перспективное направление исследований, IDP: привело к важному развитию, AM_0: прорывы в теории сложности P систем.

Annotation of the habilitation thesis "**Small Abstract Computers**", submitted by Artiom Alhazov, for fulfillment of the requirements for the *Ph.D. in habilitation* degree, specialty 01.05.01 – Theoretical foundations of computer science; computer programming. The thesis was elaborated at the Institute of Mathematics and Computer Science of the Academy of Sciences of Moldova, Chişinău, in 2013. The thesis is written in English and contains Introduction, 6 chapters, general conclusions and recommendations, bibliography of 291 titles and 8 appendices. The main text amounts to 219 pages. This work includes: 25 figures, 9 tables, 22 definitions, 19 lemmas, 77 theorems, 47 corollaries, 9 remarks, 24 examples, and 54 formulas. The results are published in 120 scientific papers.

Keywords: Theoretical computer science and unconventional computing, Models of computation and Turing computability, Descriptional complexity and small universal systems, P systems as parallel distributed multiset and string processing, Promoters/inhibitors and priorities, Active membranes and polarizations, Symport and antiport, Determinism and reversibility, Insertion-deletion-substitution and [hybrid] networks of evolutionary processors, Maximal and minimal parallelism and asynchronous mode.

The area of the present studies is membrane systems and other formal computational models, mainly *distributed parallel rewriting of multisets and strings*, e.g., networks of evolutionary processors, IDP systems, CPMs, etc. Besides the **main goal** of determining the computational power of restricted computing models, our **objectives** are identifying the polynomial-time problem solvability by a restricted model, and the Time complexity of problems depending on the features. The most typical restrictions are: Bounded descriptional complexity, The way of object interaction, A subset of objects with restricted features, Property (e.g., determinism, reversibility, self-stabilization, confluence).

Scientific novelty. We *introduced*: CFG time-yield, multiset bounding, strong determinism/reversibility, self-stablization in P systems, new active membrane rules, OHNEPs, NCPM5, polymorphism. *New strategies*: encoding data by membranes, advanced rule minimization, new descriptional complexity parameter combinations, extreme non-determinism in active membranes, a new membrane-object interaction, more than 3 polarizations, a new regular control technique for 1-symbol string operations. *New problems/models*: intractable non-decisional computational problems, Romanian language inflections, dictionary modules.

Solved scientific problems include: 1)Finding the computational power of P systems a)transitional with membrane creation and division, b)with active membranes without polarizations, c)deterministic controlled non-cooperative, d)with energy. 2)Characterizing a)the class of problems polynomial-time solvable by P systems with active membranes without polarizations, b)exact power of HNEPs with 1 node.

Most important results. *Universality*: 23-rule FsMPMRS, transitional P systems with membrane creation/dissolution, AM_0, IDPs with priorities. *Studies*: determinism/reversibility, NEPs, HNEPs, OHNEPs, Solving PSPACE problems with AM_0. *Characterizations*: deterministic controlled non-cooperative systems, energy systems.

Theoretical significance. *Fundamental problems* of distributed parallel processing of multisets/strings. *Best known results* for $LOP(ncoo, tar)$, rules in MPMRSs, $NOP_1(sym_3)$, nodes in HNEPs, polarizations of efficient *minpar* P systems, synchronization. *Important characterizations*: $DMR(ncoo, control)$. *Fundamental properties* of MR: variants of determinism, reversibility, self-stabilizations, for multiset processing with different features/modes. Computational completeness of systems with *very weak cooperation* between their elements: $OP(ncoo, tar, \delta, mcre)$ and $OP(a_0, b_0, c_0, d_0, e_0)$. *Optimal results* for known problems for: symport/antiport P systems, P systems with AM_0, nodes in computationally complete NEPs. **Applications**: Polymorphism. Dictionary modules. Inflections and annotating affixes in Romanian. **Impact** OHNEPs: a new perspective research area, IDPs: further important development, AM_0: P systems complexity theory breakthroughs.

Abbreviations and Selected Notations

(**e**)**IDP** – P systems with (exo) insertion and deletion
(**N**)**CPM**(**5**) – (Non-deterministic) circular Post machines (of variant 5)
CF(**G**) – Context-free (grammar)
(**O**)(**H**)**NEP** – (Obligatory) (hybrid) networks of evolutionary processors
(**Fs**)(**MP**)**MPMR** – (Finite-state) (maximally parallel) multiset rewriting
UREM P system – P system with unit rules and energy assigned to membranes
TVDH system – Time-varied distributed H system
TM – Turing machine
RLEM – Reversible logical element with memory
(**co-**)(**N**)**P** – Class of (complements of) problems, decidable by (non-)deterministic Turing machines in polynomial time
SAT – The known NP-complete problem of satisfiability of a boolean formula
#**P** – Class of problems, related to computing the number of solutions for a problem in NP
PP – Probabilistic polynomial time complexity class
PSPACE – Class of problems, decidable by Turing machines in polynomial space
QSAT – The problem of satisfiability of a quantified boolean formula
(**Ps**)**RE** – The family of (Parikh images of) recursively enumerable languages
CS – The family of context-sensitive languages
(**Ps**)**MAT** – The family of (Parikh images of) languages generated by matrix grammars without appearance checking (possibly with erasing productions)
SLIN – Semilinear languages
(**co-**)**NFIN** – (Complements of) finite sets of numbers
$REG \bullet \texttt{Perm}(REG)$ – Regular languages concatenated with permutation of regular languages
$N_kFIN_{(m)}$ – all finite sets of numbers not less than k (and not exceeding $m+k$)
$N_kREG_{(m)}$ – sets of numbers accepted by finite automata (with m states), plus k
$N_{(k)}RE$ – recursively enumerable sets of numbers (not less than k)
$(ncoo, coo, pro, inh, Pri)$ – no cooperation, cooperation, promoters, inhibitors, priorities
$N_aD_sOP_1$ – Sets of numbers accepted by strongly deterministic P systems with symbol objects with 1 membrane (maximal parallelism is assumed by default)
$NRMR$ – Sets of numbers generated by reversible (sequential) multiset rewriting systems
AM_0 – Active membranes without polarizations
PMC – Polynomial-time complexity class, e.g., for membrane systems specified by subscript
$LOP(ncoo, tar)$ – Languages generated by non-cooperative transitional P systems
$NOP_1(sym_3)$ – Number sets generated by P systems with symport rules of weight at most 3
$OP_1^{sequ}(sym_s)$ – Sequential P systems with symport of weight at most s
$minpar$ – Working in the minimally parallel mode
$DMR(ncoo, control)$ – Deterministic multiset rewriting systems with non-cooperative rewriting and control (promoters, inhibitors and/or priorities)
$OP(ncoo, tar, \delta, mcre)$ – Transitional P systems with membrane dissolution and creation
$OP(a_0, b_0, c_0, d_0, e_0)$ – P systems with polarizationless active membranes
$DPs_aOP_{1,1,1}(active_2, a, c)$ – Sets of vectors deterministically accepted by P systems with 1 active membrane and 2 polarizations, using evolution and send-out rules
OtP – Tissue P systems (graph instead of tree, with sequential channels)
$ELSP(ins_1^{0,0}, del_1^{0,0})$ – Languages generated modulo terminal alphabet by P systems with string-objects and insertion and deletion rules of 1 symbol without context
$ELSP(e-ins_1^{0,0}, e-del_1^{0,0})$ – Languages generated by extended P systems with insertion and deletion rules of 1 symbol at the ends of the string without context

INTRODUCTION

Unconventional Computing

The concept of unconventional computing has caught the attention of many minds, and many researchers consider it a breakthrough in theory of processing information. It is a timely and very dynamic domain of research. For instance, the Institute for Scientific Information, ISI, has mentioned in February 2003 the paper introducing membrane computing, [249], first circulated as [250], as *fast breaking in the area of computer science*; in under 15 years there over 1400 papers by over 200 researchers in membrane computing).

The most popular motivations explaining the importance of unconventional computing are the miniaturization (a starting point for massive parallelism or storage), and the Moore's law. It is, however, possible to imagine a number of other reasons to focus on unconventional computing, without directly having applications in mind. For instance, 1) developing new methods of algorithm design for conventional computers (recall, e.g., the success of genetic algorithms and neural networks, both being inspired by nature/life and implemented in silicon), 2) new perspective insights into the fundamental Physics laws, e.g., determinism, reversibility, conservation laws, 3) new measures of information, since we are dealing with non-standard data structures, 4) new methods of data encryption, due to different ways of representing information, 5) the interdisciplinary research bridging Classical computability, Information theory, Number theory, Biology, Physics, etc. Unconventional computing is already successful as fundamental research.

One can consider numerous variants of a number of models, looking for adequacy with respect to the biochemical origins of the idea or for elegance of definitions, strength of results, or similarity with other areas of Theoretical computer science. Most research described in this habilitation thesis belongs to membrane computing, we now briefly introduce this field.

Membrane computing Membrane systems, also called P systems, are a framework of distributed parallel computing models, inspired by some basic features of biological membranes. In membrane systems, objects are placed in regions, defined by a membrane structure, and are evolving by means of "reaction rules", associated with regions or with membranes. The rules are applied non-deterministically, and (in most models) in a maximally parallel manner (no rules are applicable to the remaining objects). Objects can also pass through membranes. Many other features are considered. These notions are used to define the transitions between the configurations of the membrane system, and its evolution is used to define the computation.

Three ways of computing were studied: computing functions (data processing), generat-

ing and accepting sets (of strings, numbers or vectors). Many different classes of P systems have been investigated, and most of them turned out to be computationally complete with respect to the Turing-Church thesis (i.e., equal in power to Turing machines).

It is quite convenient for a researcher in the domain that a comprehensive bibliography of membrane computing (over 1400 titles) can be found on the P systems web page, [284]. There one can also find many articles available for download, including pre-print versions and preliminary proceedings of the meetings; Table 1 contains some of the meetings (downloadable from [284]) where the author has multiple publications. Journal articles are sometimes made available online by the publisher, freely or for the subscribed users. The post-proceedings of the Workshop/Conference on Membrane Computing are published in *Lecture Notes in Computer Science* Series by Springer.

The list of author's articles and links to full articles, abstracts and/or publisher pages can be found at the author's publication webpage, [285].

Table 1: Selected references of Membrane Computing Meetings

Meeting	First time
Conference/Workshop on Membrane Computing	2000
Brainstorming Week on Membrane Computing	2003
ESF Exploratory Workshop on Cellular Computing (Complexity Aspects)	2005

The Topics

The goal of the present thesis is investigating membrane systems as well as a number of other formal computational models, mainly *distributed parallel rewriting of multisets and strings*, such as networks of evolutionary processors, reversible logical elements with memory, number-conservative cellular automata, circular Post systems, insertion-deletion systems, splicing systems, and gene assembly in ciliates. Most of the above mentioned models permit (or inherently have) parallelism and biological inspiration. We underline, however, that we are speaking about formal abstract models of computation. The thesis covers a survey and the authors's research in the fields mentioned above.

The models of membrane systems we consider here are: maximally parallel multiset rewriting, without and with cooperation, without and with promoters/inhibitors/priorities, deterministic or not, reversible or not; P systems with symport/antiport; P systems with active membranes; P systems with insertion-deletion; P systems with ciliate operations; P systems with energy, etc.

The main objectives are related to the following questions:

- What restrictions can be placed on a model/variant such that it is still computationally complete?

- What is the computational power of a given computing model with certain restrictions (e.g., when maximally parallel object cooperation is not enough for the computational completeness)?

- Can particular problems be solved in a polynomial number of steps by some model with certain restrictions (or what is the time complexity of some problem depending on the features and restrictions of the model)?

The most typical restrictions are bounding the descriptional complexity (e.g., the number of membranes or the number of objects), restricting the way of object interaction (e.g., the number of objects involved in a rule, or the way in which they are involved), distinguishing a subset of objects with restricted features (e.g., catalysts, bi-stable catalysts, protons) and considering P systems with some property (e.g., determinism, reversibility, self-stabilization, confluence, ultimate confluence, always halting).

The methods typically used are comparison with (e.g., simulation of) known computational devices (finite automata, context-free grammars, regulated grammars, register machines, 0L systems, grammar systems, other classes of P systems).

What is New?

Below we highlight a few manifestation of scientific novelty in the author's research reflected in this habilitation thesis.

By introducing time-yield of context-free grammars, we gave a new perspective to the P systems language family; we also improved the estimates of their power. Defining the bounding operation for multisets, we showed that determinism is critical for the power of non-cooperative controlled multiset rewriting, precisely characterizing the restricted case. Introducing stronger variants of determinism and reversibility, we brought the research of these properties for membrane systems closer to the more classical computing models, also giving some syntactic characterization of the properties; we characterized the power of most cases of sequential and maximally parallel multiset rewriting systems. We were the first to define (variants of) the self-stabilization property for membrane systems, also characterizing the power of most cases.

By encoding data in the multiplicity of membranes rather than the multiplicity of the objects, we solved the open problem of the power of non-cooperative P systems with membrane creation and division, showing their computational completeness. With additional ideas, we also adapted this technique for P systems with active membranes without polarizations. By inventing advanced minimization techniques, we showed that 23 rules suffice for a universal maximally parallel multiset rewriting system. We improved a number of results for symport/antiport P systems, reaching new "corners" of the space of combination of descriptional complexity parameters.

Using a new technique with extreme non-determinism, we simulated Turing machines working in exponential space, by P systems with polynomial alphabet. A new look at register machines let us construct computational complete P systems with active membranes with 2 polarizations with a single membrane – this is, clearly, one of the optimal results.

Not only have we decreased the known number of polarizations of efficient P systems with active membranes from three to two, but also we were the first ones to consider exact solutions to intractable computational problems that go beyond deciding. As for showing the characterization of PSPACE by P systems with non-elementary division (the upper bound of their power has been known) without polarizations, we have invented a new membrane-to-object interaction technique, checking the time of membrane dissolution. For P systems work in the minimally parallel way, we established their efficiency; for this we were the first to consider more than three polarizations, and we invented new variants of active membrane rules, to accomplish the intermediate result; we invented a way of simulating the new rules with the standard ones to answer the original question.

We proceed by describing what is new here for the string-object computational models. We invented a regular control technique making it possible to force the computation by one-symbol operations in the needed direction, yielding the computational completeness of the networks of evolutionary processors (NEPs) already with two nodes. Not only have we improved the state of the art for the hybrid NEPs, but we also introduced the obligatory hybrid ones (OHNEPs), opening quite a perspective research direction. For both OHNEPs and exo insertion-deletion P systems, we have introduced and used a tool, variant 5 of circular Post machines, leading to surprising results.

We proposed a new implementation of the inflection model for the Romanian language by P systems. We constructed dictionary modules from P systems, proposing a new meaning of computations. New ideas and rule types were needed to improve the synchronization time on P systems. Finally, we should mention that we have introduced a new feature of membrane system, that is motivated by the cell nucleus – the polymorphism.

This is only a partial description of the novelties reflected in this work.

Theory and Applications

Scientific problems solved in this thesis include: 1)Finding the computational power of a)transitional P systems with membrane creation and division, b)P systems with active membranes without polarizations, c)deterministic controlled non-cooperative P systems, d)P systems with energy. 2)Characterizing a)the class of problems polynomial-time solvable by P systems with active membranes without polarizations, b)exact power of hybrid networks of evolutionary processors with 1 node.

In the following the reader can see the manifestations of theoretical significance and applied value of the research reflected in the present habilitation thesis. We have addressed a number of fundamental problems of distributed parallel processing of multisets and strings, and we proved the best known bounds, e.g., for the membrane systems language family and for the number of rules in maximally parallel multiset rewriting systems.

We have proved the best known results for the optimization problems considered by different authors, e.g., the power of symport-3 in one membrane, the number of nodes in the hybrid networks of evolutionary processors, the number of polarizations of efficient P systems with minimal parallelism, and synchronization time of P systems.

We obtained important characterizations of rewriting systems, e.g., deterministic con-

trolled non-cooperative multiset rewriting systems.

We produced a landscape of results for the fundamental properties of multiset rewriting, such as variants of determinism, reversibility, and self-stabilizations, for multiset processing with different features (e.g., kinds of cooperation and control) working in different modes.

We showed the computational completeness of systems with very weak forms of cooperation between the elements of these systems, e.g., non-cooperative transitional P systems with membrane creation and dissolution, and P systems with polarizationless active membranes.

We obtained the optimal results for some problems studied by multiple groups of authors, e.g., different problems for P systems with symport/antiport, different problems for P systems with polarizationless active membranes, the number of nodes in the computationally complete networks of evolutionary processors.

Impact We only mention a few cases where further investigation by other authors emerges from the publications reflected here. We defined a perspective research direction, that of obligatory hybrid networks of evolutionary processors. Out of the results reflected here, those on insertion-deletion systems, have been further developed answering the original open problem. Our research for P systems with active membranes computing the permanent of a binary matrix has lead to a few subsequent breakthroughs in complexity theory of P systems.

Applications One of the applications is polymorphic P systems. Its use is providing a framework where rules can dynamically change during the computation, which is important for problems of symbolic computation and computer algebra.

Other applications deal with linguistics. We proposed an efficient implementation of dictionaries by membrane systems, using membrane (tree) structure to represent the prefix tree of the dictionary. We discovered suitability of P systems for performing inflections of words in the Romanian language. We also proposed P systems annotating affixes of the Romanian language, also elaborating a model that accounts for complex derivation steps that may consist of multiple affixes, changing terminations and/or alternations in the root.

Main **conferences** of presentation of the results reflected in the present habilitation thesis: 13th Conference on Membrane Computing, Budapest, Hungary; 10th, 8th, 7th and 4th Brainstorming Week on Membrane Computing, Sevilla, Spain; 12th Conference on Membrane Computing, Fontainebleau, France; 11th Conference on Membrane Computing, Jena, Germany; LA Symposium 2010, Kyoto, Japan; Unconventional Computing 2010, Tokyo, Japan; 10th Workshop on Membrane Computing, Curtea de Argeş, Romania; Languages and Automata Theory and Applications 2007, Tarragona, Spain; Machines, Computations and Universality 2007, Orléans, France; 7th Workshop on Membrane Computing, Leiden, Netherlands. Other conferences with results of thesis: ICTCS2012/Varese, IWANN2011/Málaga, RIMS2011/Kyoto, BWMC9/Sevilla, Theorietag2010/Kassel, RC2010/Bremen, IWNC2009/Himeji, LATA2008/Tarragona, DCFS2008/Ontario, WMC9/Edinburgh, DNA13/Memphis TN, WMC8/Thessaloniki, BWMC5/Sevilla.

The author has published over 160 papers. Besides having written about 20 single-author publications, he has also collaborated with over 50 coauthors from 14 countries. The author

has publications in the most prestigious journals in Theoretical computer science, e.g., *Theoretical Computer Science* and *Information Processing Letters* by Elsevier, *Natural Computing* and *Acta Informatica* by Springer, *Fundamenta Informaticae* by the Polish Mathematical Society, *International Journal of Foundations of Computer Science* by World Scientific, and *New Generation Computing* by Ohmsha Ltd. and Springer Japan. He has over 30 articles in *Lecture Notes of Computer Science* published by Springer, and chapters in monographs by the *Oxford University Press*, by the *Cambridge Scholars Publishing*, and in the Natural Computing Series by *Springer*.

By the completion of this thesis, DBLP has showed 37 journal papers and 37 conference ones, and Google Scholar has reported the author's **h-index** of 16 and **i10-index** of 30, having registered over 880 **citations**. Out of over 200 researchers working in the areas mentioned below, the author is one of the main contributors to, e.g., the following research: maximally parallel multiset rewriting, non-cooperative P systems with/without control, transitional P systems, symport/antiport, evolution-communication P systems, active membranes, reversibility in membrane computing, and the networks of evolutionary processors. The author has produced a number of **applied** results, e.g., on the multicriterial bottleneck transportation problem, sorting, synchronization, chaining in logic, polymorphism, inflections in the Romanian language, and annotating affixes in the Romanian language.

The Structure

Following the present introductory chapter, Chapter 1 introduces the prerequisites needed to understand the topics, and tools used for obtaining the results, as well as the basic definitions for the corresponding models of computation. We also analyze the situation in the associated fields of study, as the basis for presenting the results in the subsequent chapters.

We define preliminaries of the Formal Language Theory, Grammars, and Automata. We describe the work of networks of evolutionary processors, and then we continue with membrane systems. In case of symbol-objects, we start with multiset rewriting, maximal parallelism, and define a few properties of interest. Then we introduce transitional P systems, symport/antiport rules, P systems with active membranes, and two energy models. In case of string-objects, we introduce rules replicating strings and the framework of active membrane for this case; we define insertion-deletion P systems. Finally, we define the framework of solving decisional problems with P systems, and introduce the minimally parallel mode.

Chapter 2 mainly discusses multiset rewriting. First, we consider sequentializing the output of maximally parallel non-cooperative multiset rewriting, and the problem of characterizing the associated family of languages, fundamental for the domain. Second, we add to multiset processing such features of control (of rule applicability) as promoters, inhibitors and priorities, and discuss the critical role of determinism for non-cooperative systems, looking also at different computational modes. Third, we consider cooperative systems, in framework of determinism, reversibility and the strong version of these properties, establishing how such properties influence the computational power of multiset rewriting systems, both for parallel and sequential mode. Fourth, we consider variants of self-stabilization property, and proceed with characterization of the corresponding systems. Finally, we look at the role

of distributivity, by switching directly to the transitional membrane systems, enriched with possibility of membrane creation and membrane dissolution (yielding dynamically changing underlying tree structures for the distributed computations).

In Chapter 3 we consider static membrane structure and rules moving objects between the neighboring regions, without even changing them. It is well known that this setting already yields computational completeness, provided that we have a region with unbounded supply of some objects.

First, we present a concrete small universal P system with 23 antiport rules, explaining the minimization strategies used to construct it. Second, we present the state-of-the-art of symport/antiport P systems containing our recent results, paying special attention to the descriptional complexity parameters, Finally, we present the latest study of P systems with one membrane and symport only, aiming to improve the characterization of their power.

In Chapter 4 we describe the results for the models where membranes play an active role in the rules: they inhibit local parallelism of all rules except rewriting, and they may control the applicability of rules via their polarization (the dynamic information stored on the membrane). It is particularly interesting to see complex interactions, even leading to unpredictable behavior, also in case of one polarization (i.e., when the membrane itself cannot change, except being created or destroyed).

We show the universality without polarizations and unboundedly many membranes, and the universality with two polarizations (i.e., storing as little as possible information) already with two membranes. We show that with two polarizations P systems with active membranes can efficiently solve NP-complete problems, then we show how to compute even more. Next, we consider non-elementary membrane division, and show solutions of a PSPACE-complete problem, even without polarizations. After that, we look at a less synchronized computational mode, the minimal parallelism, where maximality of parallelism by definition is not enforced locally, but only globally. Surprisingly, in this case P systems are still efficient, but they seem to require more polarizations than traditionally considered.

Then we proceed to the energy models. The first one can be viewed as having infinitely many numeric membrane polarizations, but only modifying their value by adding or subtracting, only being able to check the value by applicability of subtraction with non-negative result. The second model associates energy to objects and regions instead of membranes. Both models are computationally complete if and only if parallelism is forced by the mode.

In Chapter 5 we switch to string-objects, i.e., instead of only having a finite set of different symbols as objects and tracking just their multiplicity in the nodes of the underlying distributed structure, we now have strings of symbols as objects. Not incidentally, in some cases for computational completeness it suffices to only have one object during the computation, or have no interaction (even indirectly) between different strings.

First, we consider the framework of the networks of evolutionary processors (NEPs), where the nodes of the computational structure only have such elementary operations as inserting one symbol, deleting one symbol, or substituting one symbol by one symbol. With the help of regular expressions serving as filters for entering and leaving the nodes, computational completeness is obtained already with two nodes. In the hybrid models, the place of applying the elementary operations mentioned above may be restricted to the left or right

end of the string, but now the filters may only verify presence/absence of some symbols in the string. We present subregular characterization of NEPs with one nodes and computational completeness with seven nodes. Then we study the recently defined obligatory operations.

Second, we look at the first two of the elementary operations mentioned above. In general, there are defined insertion and deletion of substrings not restricted to one symbol, and their applicability may be controlled by left and right context. However, the most interesting questions concern insertion and deletion of single symbols, without context. In order for such systems to yield non-trivial computation, we let the process be controlled by the nodes of the distributed structure, thus returning to the framework of P systems. We note that if the strings evolve independently, their behavior is a just union of behaviors of systems with one string, which are non-deterministic string-processing systems with states (compare being in node j with being in state j). Particularly, we focus on possibilities to restrict the associated control structure to be a tree.

After considering P systems with insertion and deletion anywhere in the strings, we take the recently introduced case where the insertion and deletion are only applicable in the left or right ends of the string. We prove similar results, using a tool that we have recently introduced, specifically for obtaining these kinds of results for string processing systems.

Finally, we recall that another interesting operation on string is splicing. We mention a number of our results, without describing them in detail.

Chapter 6 is devoted to the applications of areas of our research. First, we look at the task of generating inflections of words in the Romanian language. We also mention the problem of annotating affixes in Romanian words. One of the reasons why it was interesting to consider Romanian language is that it is considerably more flective than, e.g., English. Second, we present P systems describing a few operations with dictionaries. We found the membrane structure very suitable for representing and working with the prefix tree of a dictionary. One interesting point is that the constructions we present are reusable models, in contrast to the typical approach that the role of a computation of a membrane system is limited to giving a result. Third, we present a solution to the problem of synchronizing activity of nodes of a tree. Our deterministic solution works in time $3h + 3$, where h is the depth of the tree. Fourth, we present a new feature of P system: the polymorphism. Polymorphic systems can change the applicable rules during the computation. A special aspect of polymorphism is that this is the first model of P systems, where the set of possible rules is not restricted to be finite by the initial description of the system. Polymorphic P systems exhibit cases of different behavior from the standard ones; they should be useful in a number of problems requiring this. Finally, we list a few other applicative directions of our recent research that we decided not to describe in detail in this habilitation thesis.

Following the applications, we give general conclusions, particularly discussing the most important results. Following the bibliography, we present appendices containing some information of local scope, namely, relationships of membrane systems with the time-yield of context-free grammars, definitions of generalized promoter-inhibitor-priority conditions, the definition of variant 5 of the circular Post machines used for the proofs, and two technical proofs: one for the minimally parallel P systems with standard rule types and six polarization, and the other one for the sequential case of P systems with unit rules and energy assigned to membranes.

1. PREREQUISITES AND OVERVIEW

1.1 Formal Language Prerequisites

An *alphabet* is a finite non-empty set V of abstract *symbols*. The set of all words over V (the free monoid generated by V under the operation of concatenation) is denoted by V^*, we denote the concatenation operation by $\bullet$ (which is written only when necessary), the empty word is denoted by λ, and $V^* \setminus \{\lambda\}$ is denoted by V^+. Any set $L \subseteq V^*$ is called a (formal) *language* (over V), and a set $\mathbf{F}$ of formal languages may be called a *family* of languages. We write $\mathbf{F}(k)$ to mean all languages of $\mathbf{F}$ over (at most) k-symbol alphabet.

For a word $w \in V^*$ and a symbol $a \in V$, the number of occurrences of a in w is written as $|w|_a$. Similarly, the number of occurrences of symbols from a set S in w is written as $|w|_S$. The set of all symbols appearing in w is denoted by $alph(w)$. The permutations of a word $w \in V^*$ are $\texttt{Perm}(w) = \{x \in V^* : |x|_a = |w|_a \ \forall a \in V\}$. We denote the set of all permutations of the words in L by $\texttt{Perm}(L)$, and we extend this notation to families of languages. We use FIN, REG, LIN, CF, MAT, CS, RE to denote finite, regular, linear, context-free, matrix without appearance checking and with erasing rules, context-sensitive and recursively enumerable families of languages, respectively. The family of languages generated by extended (tabled) interactionless L systems is denoted by $E(T)0L$. We also refer the reader to [269] and [193].

For a finite set V, a (finite) multiset over V is a mapping from V into $\mathbb{N}$; we say that M' is a submultiset of M if $M'(a) \leq M(a)$ for all $a \in V$. In the following, we will use $\subseteq$ both for the subset as well as the submultiset relation. We also write $|x|$ and $|M|$ to denote the length of a word $x \in V^*$ and the weight (i.e., sum of multiplicities $M(a)$ of all symbols $a \in V$) of a multiset M, respectively. We use notations of set difference for multisets, meaning that the multiplicity of each symbol is the difference of corresponding multiplicities, assuming it is non-negative. It is common to write $\langle m_1, a_1\rangle \cdots \langle m_n, a_n\rangle$ to denote a multiset M such that $M(a_i) = m_i$, $1 \leq i \leq n$. Whenever it is clear from the context, M can be represented as any string x the Parikh vector of which with respect to $a_1, \cdots, a_n$ is $(m_1, \cdots, m_n)$.

We often use string notation to denote the multisets. When speaking about membrane systems, keep in mind that the order in which symbols are written is irrelevant, unless we speak about the symbols sent to the environment. In particular, speaking about the contents of some membrane, when we write $a_1^{n_1} \cdots a_m^{n_m}$ (or any permutation of it), we mean a multiset consisting of n_i instances of symbol a_i, $1 \leq i \leq m$.

The set of non-negative integers is denoted by $\mathbb{N}$; a set S of non-negative integers is called *co-finite* if $\mathbb{N} \setminus S$ is finite. The family of all finite (co-finite) sets of non-negative integers is denoted by $NFIN$ ($coNFIN$, respectively). The family of all recursively enumerable sets

of non-negative integers is denoted by NRE. Throughout the thesis, by "number" we will mean a non-negative integer. We write SEG_1 to denote finite consecutive numeric segments and SEG_2 to denote all elements of SEG_1 with the same parity:

$$\begin{aligned} SEG_1 &= \{\{j \mid m \leq j \leq n\} \mid m, n \in \mathbb{N}\}, \\ SEG_2 &= \{\{i + 2j \mid 0 \leq j \leq m\} \mid i, m \in \mathbb{N}\}. \end{aligned}$$

We write N_jFIN_k to denote the family of all sets of numbers each not smaller than j, of cardinality k.

A linear number set of non-negative is a set S that can be defined by numbers p_0, p_1, $\cdots$, p_k as $S = \{p_0 + \sum_{i=1}^{k} n_i p_i \mid n_i \geq 0,\ 1 \leq i \leq k\}$. Linear sets are a subclass of $NREG$. We call a class of sets sublinear if it is a proper subclass of linear sets.

Let $\{a_1, \cdots, a_n\}$ be an arbitrary alphabet; the *Parikh vector* associated with x with respect to $a_1, \cdots, a_n$ is $(|x|_{a_1}, \cdots, |x|_{a_n})$. The *Parikh image* of a language L over $a_1, \cdots, a_n$ is the set of all Parikh vectors of strings in L. For a family of languages $\mathbf{F}$, the family of Parikh images of languages in $\mathbf{F}$ is denoted by $Ps\mathbf{F}$. A *length set* of a language L is a set $\{|x| \mid x \in L\}$ denoted by $N(L)$, and notation $N\mathbf{F}$ means $\{N(L) \mid L \in \mathbf{F}\}$.

For a word $u \in V^*$, we define the sets of proper prefixes, proper suffixes and non-empty suffixes of u by

$$\begin{aligned} PPref(u) &= \{x \mid u = xy, |y| \geq 1\}, \\ PSuf(u) &= \{y \mid u = xy, |x| \geq 1\}, \\ NSuf(u) &= \{y \mid u = xy, |y| \geq 1\}, \text{ respectively.} \end{aligned}$$

We use symbol $\sqcup\!\sqcup$ to denote the shuffle operation on words, $u \sqcup\!\sqcup \lambda = \lambda \sqcup\!\sqcup u = \{u\}$, $u \in V^*$; $au \sqcup\!\sqcup bv = a(u \sqcup\!\sqcup bv) \cup b(au \sqcup\!\sqcup v)$.

The shuffle operation is defined on two words $x, y \in V^*$ by

$$\begin{aligned} \sqcup\!\sqcup(x, y) = \{x_1 y_1 x_2 y_2 \ldots x_n y_n \mid n \geq 1,\ x_i, y_j \in V^*, \\ x = x_1 x_2 \ldots x_n,\ y = y_1 y_2 \ldots y_n\}. \end{aligned}$$

Let $L_1, L_2 \in V^*$ be two languages. Then

$$\sqcup\!\sqcup\,(L_1, L_2) = \bigcup\nolimits_{x \in L_1, y \in L_2} \sqcup\!\sqcup\,(x, y).$$

The shuffle of regular languages is known to be regular.

1.1.1 Grammars

A *type-0 generative grammar* is a quadruple $G = (N, T, S, P)$, where N and T are disjoint alphabets, called the nonterminal and terminal alphabet, respectively, $S \in N$ is the start symbol or the axiom, and P is a finite set of productions or rewriting rules of the form $u \to v$, where $u \in (N \cup T)^* N (N \cup T)^*$ and $v \in (N \cup T)^*$. For two strings x and y in $(N \cup T)^*$, we say that x directly derives y in G, denoted by $x \Longrightarrow_G v$, if there is a production $u \to v$ in P such that $x = x_1 u x_2$ and $y = x_1 v x_2$, $x_1, x_2 \in (N \cup T)^*$ holds. The transitive and reflexive closure of $\Longrightarrow_G$ is denoted by $\Longrightarrow_G^*$. The language $L(G)$ generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow_G^* w\}$.

A grammar is called *context-sensitive* or *type-1* if $|u| \leq |v|$ for all $u \to v$ in P, except possibly $S \to \lambda$, provided that S never appears at the right side of any production. A grammar is called *context-free* or *type-2* if $u \in N$ for all $u \to v$ in P. A context-free grammar is called *linear* if $|v|_N = 1$. We say that a context-free grammar is *regular* or *type-3* if $v \in NT^* \cup T^*$ for all $u \to v$ in P. Without restricting generality, we can additionally require that $v \in NT \cup T$, except possibly $S \to \lambda$, provided that S never appears at the right side of any production.

We recall now a concept dual to a type-0 generative grammar, called a *type-0 analytic grammar* [270]. A type-0 analytic grammar $G = (N, T, S, P)$ is a quadruple, where N, T, S are defined in the same way as for a generative grammar, and P is a finite set of productions of the form $u \to v$, where $u \in (N \cup T)^*$ and $v \in (N \cup T)^* N (N \cup T)^*$. The derivation relation is defined for a type-0 analytic grammar analogously to the derivation relation for a type-0 generative grammar. The language $L(G)$ recognized or accepted by a type-0 analytic grammar $G = (N, T, S, P)$ is defined as $L(G) = \{w \in T^* \mid w \Longrightarrow_G^* S\}$.

It is well-known that for the type-0 analytic grammar G' obtained from a type-0 generative grammar G with interchanging the left and the right hand sides of the productions in G, it holds that $L(G') = L(G)$.

A type-0 generative grammar $G = (N, T, S, P)$ is in Kuroda normal form if every rule in P is one of the following forms: $A \longrightarrow a$, $A \longrightarrow \lambda$, $A \longrightarrow BC$, $AB \longrightarrow CD$, where $A, B, C, D \in N$ and $a \in T$.

Analogously, we can say that a type-0 analytic grammar $G = (N, T, S, P)$ is in Kuroda-like normal form if every production in P is one of the following forms: $a \longrightarrow A$, $\lambda \longrightarrow A$, $AB \longrightarrow C$, $AB \longrightarrow CD$, where $A, B, C, D \in N$ and $a \in T$.

It is well-known that the type-0 generative grammars in Kuroda normal form determine the class of recursively enumerable languages and it can immediately be seen that the same statement holds for the type-0 analytic grammars in Kuroda-like normal form.

1.1.2 Matrix grammars

A context-free *matrix grammar* (without appearance checking) is a construct $G = (N, T, S, M)$ where N and T are sets of *non-terminal* and *terminal symbols*, respectively, with $N \cap T = \emptyset$, $S \in N$ is the *start symbol*, M is a finite set of *matrices*, $M = \{m_i \mid 1 \leq i \leq n\}$, where the matrices m_i are sequences of the form $m_i = (m_{i,1}, \cdots, m_{i,n_i})$, $n_i \geq 1$, $1 \leq i \leq n$, and $m_{i,j}$, $1 \leq j \leq n_i$, $1 \leq i \leq n$, are context-free productions over (N, T).

For $m_i = (m_{i,1}, \cdots, m_{i,n_i})$ and $v, w \in (N \cup T)^*$ we define $v \Longrightarrow_{m_i} w$ if and only if there are $w_0, w_1, \cdots, w_{n_i} \in (N \cup T)^*$ such that $w_0 = v$, $w_{n_i} = w$, and for each j, $1 \leq j \leq n_i$, w_j is the result of the application of $m_{i,j}$ to w_{j-1}. The language generated by G is $L(G) =$

$\{w \in T^* \mid S \Longrightarrow_{m_{i_1}} w_1 \cdots \Longrightarrow_{m_{i_k}} w_k = w,$

$w_j \in (N \cup T)^*,\ m_{i_j} \in M \text{ for } 1 \leq j \leq k,\ k \geq 1\}.$

According to the definitions given in [162], the last matrix can already finish with a terminal word without having applied the whole sequence of productions.

The family of languages generated by matrix grammars without appearance checking is denoted by MAT. It is known that for the family of Parikh sets of languages generated by matrix grammars $PsMAT$ we have $PsCF \subset PsMAT \subset PsRE$. Further details about

matrix grammars can be found in [162] and in [269]. We only mention that the power of matrix grammars is not decreased if we only work with matrix grammars in the *f-binary normal form* where N is the disjoint union of N_1, N_2, and $\{S, f\}$, and M contains rules of the following forms:

1. $(S \to XA)$, with $X \in N_1$, $A \in N_2$;
2. $(X \to Y, A \to x)$, with $X \in N_1$, $Y \in N_1 \cup \{f\}$, $A \in N_2$, and $x \in (N_2 \cup T)^*, |x| \leq 2$;
3. $(f \to \lambda)$.

Moreover, there is only one matrix of type 1 and only one matrix of type 3, which is only used in the last step of a derivation yielding a terminal result.

1.1.3 Finite automata

Definition 1.1 *A finite automaton is a tuple $A = (\Sigma, Q, q_0, \delta, F)$, where Σ is an input alphabet, Q is the set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta : Q \times \Sigma \longrightarrow 2^Q$ is the transition mapping.*

The function δ is naturally extended from symbols to strings. The language accepted by A is the set $L(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset\}$.

We also say that a set M of numbers is accepted by a finite automaton A if $M = \{|w| \mid w \in L(A)$, i.e., M is the length set of the language accepted by A. By $N_j REG_k$ we denote the family of all sets M of numbers each not smaller than j, such that $\{x - j \mid x \in M\}$ is accepted by some finite automaton with k states, with at least one transition from every non-final state.

1.1.4 Counter automata

In the universality proofs we will use counter automata with conflicting counters. The meaning of the conflicting counters varies slightly for each result, so it is explained locally.

Definition 1.2 *A non-deterministic counter automaton is a tuple $M = (Q, q_0, q_f, P, C)$, where*

- *Q is the set of states,*
- *$q_0 \in Q$ is the initial state,*
- *$q_f \in Q$ is the final state,*
- *P is the set of instructions of types $(q \to q', i+)$, $(q \to q', i-)$ and $(q \to q', i = 0)$, modifying the state and incrementing or decrementing counter i by one, or verifying whether the value of the counter is zero,*
- *C is the set of counters.*

A computation of M consists of transitions between the states from Q with updating/checking the counters. Attempting to decrement a counter with value zero, or to zero-test a counter with a non-zero value leads to aborting a computation without producing a result. Without restricting generality, we assume that for every state, there is at least one instruction from it. Counter automata are known to be computationally complete: for every recursively enumerable set U of non-negative integers there exists a counter automaton generating precisely elements of U in the first counter, all others having zero value.

The computation starts in state q_0 with all counters being zero, and the result (here, a set of numbers) is the set of all values of a dedicated counter reached by all computations of M in state q_f.

1.1.5 Register machines

The most common way of establishing computational completeness of some variant of a computational model is simulating *register machines*. Here we briefly recall their definition and some of their computational properties. A *register machine* is a tuple $M = (m, Q, I, q_0, q_f)$ where m is the number of registers, I is the set of instructions bijectively labeled by elements of Q, $q_0 \in Q$ is the initial label, and $q_f \in Q$ is the final label. The instructions of M can be of the following forms:

- $(q_1, [RjP], q_2, q_3)$, with $q_1 \in Q \setminus \{q_f\}$, $q_2, q_3 \in Q$, $1 \leq j \leq m$.
  Increase the value of register j by one, and non-deterministically jump to instruction q_2 or q_3. This instruction is usually called *increment*.

- $(q_1, \langle RjZM \rangle, q_2, q_3)$, with $q_1 \in Q \setminus \{q_f\}$, $q_2, q_3 \in Q$, $1 \leq j \leq m$.
  If the value of register j is zero then jump to instruction q_3, otherwise decrease the value of register j by one and jump to instruction q_2. The two cases of this instruction are usually called *zero-test* and *decrement*, respectively.

- $(q_f : HALT)$. Stop the execution of the register machine.

A register machine is *deterministic* if $q_2 = q_3$ in all its $[RjP]$ instructions. A *configuration* of a register machine is described by the contents of each register and by the value of the program counter, which indicates the next instruction to be executed. Computations start by executing the first instruction of I (labeled with q_0), and terminate with reaching a $HALT$-instruction.

Register machines provide a simple universal computational model [233]. We here consider register machines used as *accepting* or as *generating* devices. In accepting register machines, a vector of non-negative integers is accepted if and only if the register machine halts having it as input. Usually, without loss of generality, we may assume that the instruction $q_f : HALT$ always appears exactly once in I, with label q_f. In the generative case, we start with empty registers and take the results of all possible halting computations.

Reversible register machines

We now consider an equivalent definition, suited for reversibility. Register machines now have increment, unconditional decrement and test instructions, [236], see also [233]. An

m-register machine is defined by a tuple $M = (m, Q, I, q_0, q_f)$, where m is the number of registers, I is a set of instructions bijectively labeled by elements of Q, $q_0 \in Q$ is the initial label, and $q_f \in Q$ is the final label. The allowed instructions are:

- $(q_1, \langle Rj \rangle, q_2, q_3)$ - jump to instruction q_2 if the contents of register j is zero, otherwise proceed to instruction q_3;
- $(q_1, [RjP], q_2, q_3)$ - add one to the contents of register j and proceed to either instruction q_2 or q_3, non-deterministically;
- $(q_1, [RjM], q_2, q_3)$ - subtract one from the contents of register i and proceed to either instruction q_2 or q_3, non-deterministically;
- $(q_f : HALT)$ - terminate; it is a unique instruction with label q_f.

As for subtract instructions, the computation is blocked if the contents of the corresponding register is zero. Without restricting generality, we assume that a test of a register always precedes its subtraction. (In the previous model with addition and conditional subtraction instructions, reversibility would be more difficult to describe.) A configuration of a register machine is defined by the current instruction and the contents of all registers, which are non-negative integers.

If $q_2 = q_3$ for any instruction $(q_1, [RjP], q_2, q_3)$ and for any instruction $(q_1, [RjM], q_2, q_3)$, then the machine is called deterministic. Clearly, this is necessary and sufficient for the global transition (partial) mapping not to be multi-valued.

A register machine is called reversible if in the case that there is more than one instruction leading to some instruction q, then exactly two exist, they test the same register, one leads to q if the register is zero and the other one leads to q if the register is positive. It is not difficult to check that this requirement is a necessary and sufficient condition for the global transition mapping to be injective. Let us formally state the reversibility of a register machine: for any two different instructions (q_1, OP_1, q_3, q_4) and (q_2, OP_2, q_5, q_6), it holds that $q_3 \neq q_5$ and $q_4 \neq q_6$. Moreover,

$$\text{if } q_3 = q_6 \text{ or } q_4 = q_5, \text{ then } OP_1 = OP_2 = \langle RkZ \rangle \text{ for some } 1 \leq k \leq m.$$

It has been shown ([236]) that reversible register machines are universal (a straightforward simulation of, e.g., reversible Turing Machines [143], would not be reversible). It follows that non-deterministic reversible register machines can generate any recursively enumerable set of non-negative integers as a value of the first register by all its possible computations starting from all registers having zero value.

Being used as decision devices, register machines may halt in an accepting state with label q_{yes} or in a rejecting state q_{no}, respectively. In the following, we shall call a specific model of P systems *computationally complete* if and only if for any register machine M we can effectively construct an equivalent P system Π of that type simulating each step of M in a bounded number of steps and yielding the same results.

Korec register machines

A deterministic *register machine* is the following construction:

$$M = (m, Q, I, q_0, q_f),$$

where Q is a set of states, $R = \{R_1, \ldots, R_m\}$ is the set of registers, $q_0 \in Q$ is the initial state, $q_f \in Q$ is the final state and I is a set of instructions (called also rules) of the following form:

1. $(q_1, [RjP], q_2) \in I$, $q_1, q_2 \in Q$, $q_1 \neq q_2$, $R_j \in R$ (being in state q_1, increase register R_j and go to state q_2).

2. $(q_1, [RjM], q_2) \in I$, $q_1, q_2 \in Q$, $q_1 \neq q_2$, $R_j \in R$ (being in state q_1, decrease register R_j and go to state q_2).

3. $(q_1, \langle Rj \rangle, q_2, q_3) \in I$, $q_1, q_2, q_3 \in Q, R_j \in R$ (being in state q_1, go to q_2 if register R_j is not zero or to q_3 otherwise).

4. $(q_1, \langle RjZM \rangle, q_2, q_3) \in I$, $q_1, q_2, q_3 \in Q$, $R_j \in R$ (being in state q_1, decrease register R_j and go to q_2 if successful or to q_3 otherwise).

5. $(q_f, HALT)$ (may be associated only to the final state q_f).

We note that for each state q_1 there is only one instruction of the types above.

A configuration of a register machine is given by the $(k+1)$-tuple $(q, n_1, \cdots, n_m)$, where $q \in Q$ and $n_i \in \mathbb{N}, 1 \leq i \leq k$, describing the current state of the machine as well as the contents of all registers. A transition of the register machine consists in updating/checking the value of a register according to an instruction of one of types above and by changing the current state to another one. We say that the machine stops if it reaches the state q_f. We say that M computes a value $y \in \mathbb{N}$ on the *input* $x \in \mathbb{N}$ if, starting from the initial configuration $(q_0, x, 0, \cdots, 0)$, it reaches the final configuration $(q_f, y, 0, \cdots, 0)$.

It is well-known that register machines compute all partial recursive functions and only them, [233]. For every $m \in \mathbb{N}$, with every register machine M having m registers, an m-ary partial recursive function Φ_M^m is associated. Let $\Phi_0, \Phi_1, \Phi_2, \cdots$, be a fixed admissible enumeration of the set of unary partial recursive functions. Then, a register machine M is said to be *strongly universal* if there exists a recursive function g such that $\Phi_x(y) = \Phi_M^2(g(x), y)$ holds for all $x, y \in \mathbb{N}$.

We also note that the power and the efficiency of a register machine M depends on the set of instructions that are used. In [207] several sets of instructions are investigated. In particular, it is shown that there are strongly universal register machines with 22 instructions of form $[RjP]$ and $\langle RjZM \rangle$. Moreover, these machines can be effectively constructed.

Figure 1.1 shows this special universal register machine (more precisely in [207] only a machine with 32 instructions of type $[RjP]$, $[RjM]$ and $\langle RjZ \rangle$ is constructed, and the machine below may be simply obtained from that one).

Here is the list of rules of this machine.

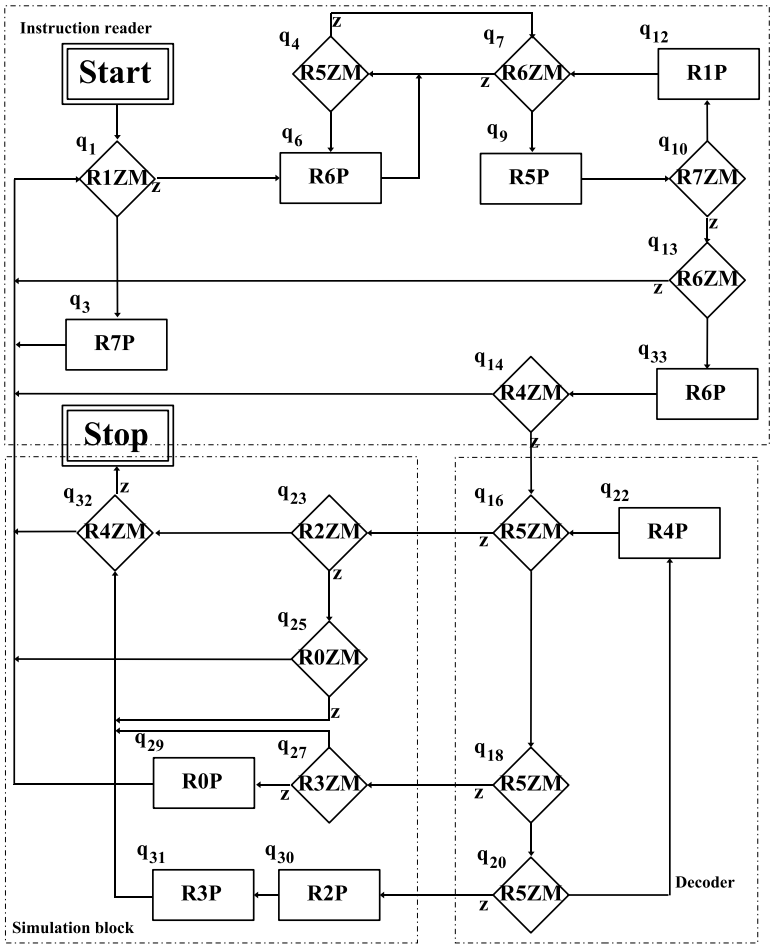


Figure 1.1: Flowchart of the strongly universal machine

$(q_1, \langle R1ZM\rangle, q_3, q_6)$ $(q_3, [R7P], q_1)$ $(q_4, \langle R5ZM\rangle, q_6, q_7)$
$(q_6, [R6P], q_4)$ $(q_7, \langle R6ZM\rangle, q_9, q_4)$ $(q_9, [R5P], q_{10})$
$(q_{10}, \langle R7ZM\rangle, q_{12}, q_{13})$ $(q_{12}, [R1P], q_7)$ $(q_{13}, \langle R6ZM\rangle, q_{33}, q_1)$
$(q_{33}, [R6P], q_{14})$ $(q_{14}, \langle R4ZM\rangle, q_1, q_{16})$ $(q_{16}, \langle R5ZM\rangle, q_{18}, q_{23})$
$(q_{18}, \langle R5ZM\rangle, q_{20}, q_{27})$ $(q_{20}, \langle R5ZM\rangle, q_{22}, q_{30})$ $(q_{22}, [R4P], q_{16})$
$(q_{23}, \langle R2ZM\rangle, q_{32}, q_{25})$ $(q_{25}, \langle R0ZM\rangle, q_1, q_{32})$ $(q_{27}, \langle R3ZM\rangle, q_{32}, q_1)$
$(q_{29}, [R0P], q_1)$ $(q_{30}, [R2P], q_{31})$ $(q_{31}, [R3P], q_{32})$
$(q_{32}, \langle R4ZM\rangle, q_1, q_f)$

Computational completeness

Register machines provide a simple universal computational model [233]. The results proved in [174] (based on the results established in [233]) as well as in [181] and [179] immediately lead to the following results:

Proposition 1.1 *For any partial recursive function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$ ($\alpha > 0,\ \beta > 0$) there exists a deterministic register machine M with $(\max\{\alpha, \beta\}+2)$ registers computing f in such a way that, when starting with n_1 to n_α in registers 1 to α, M has computed $f(n_1, \cdots, n_\alpha) = (r_1, \cdots, r_\beta)$ if it halts in the final label q_f with registers 1 to β containing r_1 to r_β, and*

all other registers being empty; if the final label cannot be reached, $f(n_1, \cdots, n_\alpha)$ remains undefined.

Register machines can also be used as *accepting* or as *generating* devices. In accepting register machines, a vector of non-negative integers is accepted if and only if the register machine halts. The following Proposition is a direct consequence of Proposition 1.1.

Proposition 1.2 *For any recursively enumerable set $L \subseteq Ps(\alpha)RE$ of vectors of non-negative integers there exists a deterministic register machine M with $(\alpha + 2)$ registers accepting L in such a way that, when starting with n_1 to n_α in registers 1 to α, M has accepted $(n_1, \ldots, n_\alpha) \in L$ if and only if it halts in the final label q_f with all registers being empty.*

To generate vectors of non-negative integers we have to use non-deterministic register machines. The following Proposition is also a direct consequence of Proposition 1.1.

Proposition 1.3 *For any recursively enumerable set $L \subseteq Ps(\beta)RE$ of vectors of non-negative integers there exists a non-deterministic register machine M with $(\beta + 2)$ registers generating L in such a way that, when starting with all registers being empty, M has generated $(r_1, \ldots, r_\beta) \in L$ if it halts in the final label q_f with registers 1 to β containing r_1 to r_β, and all other registers being empty.*

A direct consequence of the results exposed in [233] is that in Propositions 1.1 and 1.3 we may assume without loss of generality that only $[RjP]$ instructions are applied to the output registers. This fact will be used to decrease the number of membranes of energy-based P systems simulating register machines; in particular, for each output register one membrane will suffice, whereas to simulate the behavior of the other registers we will need two membranes.

A register machine is called *partially blind* if performing a zero-test blocks the computation, thus leading to no result. We can reflect this situation by omitting l_3 from all $\langle RjZM\rangle$ instructions, turning them into $[RjM]$ instructions. However, unless all non-output registers have value zero at halting, the result of a computation is discarded; note that this is an implicit final zero-test, imposed by the definition and not affecting the power of register machines in the general case. It is known [170] that partially blind register machines characterize $PsMAT$.

Tape output

When considering the generation of languages, we use the model of a *register machine with output tape*, which also uses a tape operation:

– $(q_1 : [SaW], q_a)$
  Write symbol a on the output tape and go to q_2.

We then also specify the output alphabet T in the description of the register machine with output tape, i.e., we write $M = (m, T, Q, I, q_0, q_f)$.

The following result is also folklore (e.g., see [233]):

Proposition 1.4 *Let $L \subseteq V^*$ be a recursively enumerable language. Then L can be generated by a register machine with output tape with 2 registers.*

Table 1.1: Variants of circular Post machines

CPM0	CPM1	CPM2	CPM3	CPM4	CPM5
$\mathbf{p}x \to \mathbf{q}$	$\mathbf{p}x \to \mathbf{q}$	$\mathbf{p}x \to \mathbf{q}$	$\mathbf{p}x \to \mathbf{q}$	$\mathbf{p}x \to \mathbf{q}$	$\mathbf{p}x \to \mathbf{q}$
$\mathbf{p}x \to y\mathbf{q}$	$\mathbf{p}x \to y\mathbf{q}$	$\mathbf{p}x \to y\mathbf{q}$	$\mathbf{p}x \to y\mathbf{q}$	$\mathbf{p}x \to y\mathbf{q}$	
$\mathbf{p}0 \to y\mathbf{q}0$	$\mathbf{p}x \to x\mathbf{q}0$	$\mathbf{p}x \to y\mathbf{q}0$	$\mathbf{p}x \to yz\mathbf{q}$	$\mathbf{p}x \to yx\mathbf{q}$	$\mathbf{p} \to y\mathbf{q}$

1.1.6 Circular Post machines

The following model (CPMs) has been introduced in [214], where it was shown that all introduced variants of CPMs (CPM0-CPM4, see Table 1.1) are computationally complete, and moreover, the same statement holds for CPMs with two symbols.

In [213], [91] several universal CPMs of variant 0 (CPM0) having small size were constructed, among them in [91] a universal CPM0 with 6 states and 6 symbols. We only consider the deterministic variant of CPM0s.

Definition 1.3 *A* circular Post machine *(of type 0) is a tuple* $(\Sigma, Q, \mathbf{q}_1, \mathbf{q}_f, R)$ *with a finite alphabet* Σ *where* $0 \in \Sigma$ *is the blank, a finite set of states* Q*, the initial state* $\mathbf{q}_1 \in Q$*, the final state* $\mathbf{q}_f \in Q$*, and a finite set of instructions* R *with all instructions having one of the forms* $\mathbf{p}x \to \mathbf{q}$ *(erasing the symbol read by deleting a symbol),* $\mathbf{p}x \to y\mathbf{q}$ *(overwriting and moving to the right),* $\mathbf{p}0 \to y\mathbf{q}0$ *(overwriting and inserting a blank symbol), where* $x, y \in \Sigma$ *and* $\mathbf{p}, \mathbf{q} \in Q$*,* $\mathbf{p} \neq \mathbf{q}_f$*. We also refer to all instructions with* $\mathbf{q}_f$ *in the left hand side as halt instructions.*

We also refer to all instructions with $\mathbf{q}_f$ in the right hand side as halt instructions. The storage of this machine is a circular tape, the read and write head moves only in one direction (to the right), and with the possibility to delete a cell or to create and insert a new cell with a blank.

Notice that a circular tape can be thought of as a finite string of symbols (from the one following the state to the one preceding the state in the circular representation). In this way, CPM0 is a finite-state machine, which reads the leftmost symbol of the string, possibly consuming it, and uses the symbol+state information to change the state, possibly writing a symbol on the right.

Table 1.1 summarizes the difference between the variants of CPMs, [214] (we also include CPM5 in Appendix A3). The difference between CPMk, $0 \leq k \leq 4$, is only in the way the lengthening instruction works: whether it applies to any symbol or only to the blank, whether it introduces one of the two new symbols after the state, and whether one of the new symbols equals the one read. Although all variants are computationally equivalent, these distinctions can affect the size of the smallest universal machines.

1.2 Networks of Evolutionary Processors

Evolutionary processors For an alphabet V, we say that a rule $a \to b$, with $a, b \in V \cup \{\lambda\}$ is a *substitution rule* if both a and b are different from λ; it is a *deletion rule* if $a \neq \lambda$ and $b = \lambda$; and, it is an *insertion rule* if $a = \lambda$ and $b \neq \lambda$. The set of all substitution rules,

deletion rules, and insertion rules over an alphabet V is denoted by Sub_V, Del_V, and Ins_V, respectively. Given such rules π, ρ, σ, and a word $w \in V^*$, we define the following *actions* of σ on w: If $\pi \equiv a \to b \in Sub_V$, $\rho \equiv a \to \lambda \in Del_V$, and $\sigma \equiv \lambda \to a \in Ins_V$, then

$$\pi^*(w) = \begin{cases} \{ubv : \exists u, v \in V^*(w = uav)\}, \\ \{w\}, \quad \text{otherwise} \end{cases} \tag{1.1}$$

$$\rho^*(w) = \begin{cases} \{uv : \exists u, v \in V^*(w = uav)\}, \\ \{w\}, \quad \text{otherwise} \end{cases} \tag{1.2}$$

$$\rho^r(w) = \begin{cases} \{u : w = ua\}, \\ \{w\}, \quad \text{otherwise} \end{cases} \tag{1.3}$$

$$\rho^l(w) = \begin{cases} \{v : w = av\}, \\ \{w\}, \quad \text{otherwise} \end{cases} \tag{1.4}$$

$$\sigma^*(w) = \{uav : \exists u, v, \in V^*(w = uv)\}, \tag{1.5}$$

$$\sigma^r(w) = \{wa\},\ \sigma^l(w) = \{aw\}. \tag{1.6}$$

Symbol $\alpha \in \{*, l, r\}$ denotes the way of applying an insertion or a deletion rule to a word, namely, at any position ($a = *$), in the left-hand end ($a = l$), or in the right-hand end ($a = r$) of the word, respectively. Note that a substitution rule can be applied at any position. For every rule σ, action $\alpha \in \{*, l, r\}$, and $L \subseteq V^*$, we define the $\alpha - action$ of σ on L by $\sigma^\alpha(L) = \bigcup_{w \in L} \sigma^\alpha(w)$. For a given finite set of rules M, we define the $\alpha - action$ of M on a word w and on a language L by $M^\alpha(w) = \bigcup_{\sigma \in M} \sigma^\alpha(w)$ and $M^\alpha(L) = \bigcup_{w \in L} M^\alpha(w)$, respectively.

An evolutionary processor consists of a set of evolutionary operations and a filtering mechanism.

For two disjoint subsets P and F of an alphabet V and a word over V, predicates $\varphi^{(1)}$ and $\varphi^{(2)}$ are defined as follows:

$$\begin{aligned} \varphi^{(1)}(w; P, F) &\equiv P \subseteq alph(w) \wedge F \cap alph(w) = \emptyset \text{ and} \\ \varphi^{(2)}(w; P, F) &\equiv alph(w) \cap P \neq \emptyset \wedge F \cap alph(w) = \emptyset. \end{aligned}$$

The construction of these predicates is based on *random-context conditions* defined by the two sets P (*permitting contexts*) and F (*forbidding contexts*).

For every language $L \subseteq V^*$ we define $\varphi^i(L, P, F) = \{w \in L \mid \varphi^i(w; P, F)\}$, $i = 1, 2$.

An *evolutionary processor over* V is a 5-tuple (M, PI, FI, PO, FO), where:

- Either $M \subseteq Sub_V$ or $M \subseteq Del_V$ or $M \subseteq Ins_V$. The set M represents the set of evolutionary rules of the processor. Notice that every processor is dedicated to only one type of the evolutionary operations.

- $PI, FI \subseteq V$ are the *input* permitting/forbidding contexts of the processor, while $PO, FO \subseteq V$ are the *output* permitting/forbidding contexts of the processor.

The set of evolutionary processors over V is denoted by EP_V.

Hybrid networks

Definition 1.4 *A* hybrid network of evolutionary processors *(an HNEP, shortly) is a 7-tuple* $\Gamma = (V, H, \mathcal{N}, C_0, \alpha, \beta, i_0)$*, where the following conditions hold:*

- V *is the alphabet of the network.*

- $H = (X_H, E_H)$ *is an undirected graph with set of vertices or nodes* X_H *and set of edges* E_H. H *is called the underlying graph of the network.*

- $\mathcal{N} : X_H \longrightarrow EP_V$ *is a mapping which associates the evolutionary processor* $\mathcal{N}(x) = (M_x, PI_x, FI_x, PO_x, FO_x)$ *with each node* $x \in X_H$.

- $C_0 : X_H \longrightarrow 2^{V^*}$ *is a mapping which identifies the initial configuration of the network. It associates a finite set of words with each node of the graph* H.

- $\alpha : X_H \longrightarrow \{*, l, r\}$; $\alpha(x)$ *defines the action mode of the rules performed in node* x *on the words occurring in that node.*

- $\beta : X_H \longrightarrow \{(1), (2)\}$ *defines the type of the input/output filters of a node. More precisely, for every node,* $x \in X_H$, *we define the following filters: the input filter is given as* $\mu_x(\cdot) = \varphi^{\beta(x)}(\cdot; PI_x, FI_x)$, *and the output filter is defined as* $\tau_x(\cdot) = \varphi^{\beta(x)}(\cdot, PO_x, FO_x)$. *That is,* $\mu_x(w)$ *(resp.*τ_x*) indicates whether or not the word* w *can pass the input (resp. output) filter of* x. *More generally,* $\mu_x(L)$ *(resp.* $\tau_x(L)$*) is the set of words of* L *that can pass the input (resp. output) filter of* x.

- $i_0 \in X_H$ *is the output node of* Γ.

We say that $card(X_H)$ is the size of Γ. An HNEP is said to be a complete HNEP, if its underlying graph is a complete graph.

A configuration of an HNEP Γ, as above, is a mapping $C : X_H \longrightarrow 2^{V^*}$ which associates a set of words with each node of the graph. A component $C(x)$ of a configuration C is the set of words that can be found in the node x in this configuration, hence a configuration can be considered as the sets of words which are present in the nodes of the network at a given moment.

A configuration can change either by an evolutionary step or by a communication step. When it changes by an evolutionary step, then each component $C(x)$ of the configuration C is altered in accordance with the set of evolutionary rules M_x associated with the node x and the way of applying these rules, $\alpha(x)$. Formally, the configuration C' is obtained in one evolutionary step from the configuration C, written as $C \Longrightarrow C'$, iff $C'(x) = M_x^{\alpha(x)}(C(x))$ for all $x \in X_H$.

When the configuration changes by a communication step, then each language processor $\mathcal{N}(x)$, where $x \in X_H$, sends a copy of its each word to every node processor, where the node is connected with x, provided that this word is able to pass the output filter of x, and receives all the words which are sent by processors of nodes connected with x, provided that these words are able to pass the input filter of x. Those words which are not able to pass the respective output filter, remain at the node. Formally, we say that configuration C' is obtained in one communication step from configuration C, written as $C \vdash C'$, iff $C'(x) = (C(x) - \tau_x(C(x))) \bigcup_{\{x,y\} \in E_H} (\tau_y(C(y)) \cap \mu_x(C(y)))$ holds for all $x \in X_H$.

Computation and result For an HNEP Γ, the computation in Γ is a sequence of configurations C_0, C_1,C_2, ..., where C_0 is the initial configuration of Γ, $C_{2i} \Longrightarrow C_{2i+1}$ and $C_{2i+1} \vdash C_{2i+2}$, for all $i \geq 0$.

HNEPs can be considered both language generating devices (generating hybrid networks of evolutionary processors or GHNEPs) and language accepting devices (accepting hybrid networks of evolutionary processors or AHNEPs).

In the case of GHNEPs we define the generated language as the set of all words which appear in the output node at some step of the computation. Formally, the language generated by a generating hybrid network of evolutionary processors Γ is $L(\Gamma) = \bigcup_{s\geq 0} C_s(i_0)$.

In the case of AHNEPs, in addition to the components above, we distinguish an input alphabet and a network alphabet, V and U, where $V \subseteq U$, and instead of an initial configuration, we indicate an input node i_I. Thus, for an AHNEP, we use the notation $\Gamma = (V, U, H, \mathcal{N}, i_I, \alpha, \beta, i_0)$.

The computation by an AHNEP Γ for an input word $w \in V^*$ is a sequence of configurations $C_0^{(w)}$, $C_1^{(w)}$,$C_2^{(w)}$, ..., where $C_0^{(w)}$ is the initial configuration of Γ, with $C_0^{(w)}(i_I) = \{w\}$ and $C_0^{(w)}(x) = \emptyset$, for $x \in G$, $x \neq i_I$, and $C_{2i}^{(w)} \Longrightarrow C_{2i+1}^{(w)}$, $C_{2i+1}^{(w)} \vdash C_{2i+2}^{(w)}$, for all $i > 0$.

A computation as above is said to be accepting if there exists a configuration in which the set of words that can be found in the output node i_o is non-empty. The language accepted by Γ is defined by

$$L(\Gamma) = \{w \in V^* \mid \text{ the computation by } \Gamma \text{ on } w \text{ is an accepting one}\}.$$

Obligatory networks The model of OHNEPs is obtained from the model of HNEPs by excluding the second case, i.e., "$\{w\}$,otherwise", from (1)-(4). Hence, for a string to remain in a node, it is *obligatory* for it to evolve via some rule from Sub_V, Del_V or Ins_V.

Notice that the definition of OHNEPs is thus simpler and more uniform than that of HNEPs. In the same time, using the power of the underlying graph, it makes it possible to even reach the computational completeness with nodes only having one operation, and without filters, [29].

Basic model of NEPs The concept of NEPs is simpler than that of HNEPs. We find it suitable here to define the former in terms of the latter, by modifying the following:

- Only the $*$ mode exists for evolutionary processors.
- A permitting filter and a forbidden filter are combined into one filter, which may be any regular language. A string passes the filter iff it belongs to the corresponding regular language. The filtering mode β loses its meaning.

The power of NEPs Insertion, deletion, and substitution are fundamental operations in formal language theory, their power and limits have obtained much attention during the years. Due to their simplicity, language generating mechanisms based on these operations are of particular interest. *Networks of evolutionary processors* (NEPs, for short), introduced in [148], are proper examples for distributed variants of these constructs. Motivated by some models of massively parallel computer architectures, networks of language processors have been introduced in [160]. Inspired by biological processes, a special type of networks of language processors was introduced in [148], called networks with evolutionary processors, because the allowed productions model the point mutation known from biology. Results on networks of evolutionary processors can be found e. g. in [148], [147], [146]. In [147] it was shown that networks of evolutionary processors are universal in that sense that they can generate any recursively enumerable language, and that networks with six nodes are

sufficient to get all recursively enumerable languages. The notion of an HNEP, as a language generating device, was introduced in [229] and the concept of an AHNEP was defined in [224]. In [159] it was shown that, for an alphabet V, GHNEPs with $27 + 3 \cdot card(V)$ nodes are computationally complete. For accepting HNEPs, in [222] it was shown that for any recursively enumerable language there exists a recognizing AHNEP with 31 nodes; the result was improved in [221] where the number of necessary nodes was reduced to 24. Furthermore, in [221] the authors demonstrated a method to construct for any NP-language L an AHNEP with 24 nodes which decides L in polynomial time. A significant improvement is presented in Section 5.1.

1.3 P Systems with Symbol-Objects

Membrane computing is a theoretical framework of parallel distributed multiset processing. It has been introduced by Gheorghe Păun in 1998, and has been an active research area; see [284] for the comprehensive bibliography and [252],[257] for a systematic survey. Membrane systems are also called P systems.

1.3.1 Multiset rewriting

Consider a finite alphabet O.

A multiset rewriting rule is given by $r : u \to v$, where $u \in O^+$, $v \in O^*$. This rule can be applied to a multiset w if $|w|_a \geq |u|_a$ for all $a \in O$, and the result is w', denoted $w \Rightarrow w'$, if $|wv|_a = |w'u|_a$ for $a \in O$. If a rule has a promoter a, we write it as $u \to v|_a$. If a rule has an inhibitor a, we write it as $u \to v|_{\neg a}$. The priority relationship is denoted by $>$. We also refer to u and v as $lhs(r)$ and $rhs(r)$ if $r : u \to v$ (even if it has a promoter or an inhibitor). Applicability of rules with additional control depends on the corresponding condition (presence of symbol a, absence of symbol a, or inapplicability of rules with higher priority, respectively). In the following we will not speak about promoters in the sequential case, as rules $u \to v|_a$ and $ua \to va$ then are equivalent. The relation $\Rightarrow$ is defined as the union of relations corresponding to the application of all rules.

A multiset rewriting system is a tuple (O, T, w_0, R), where O is the alphabet, T is the terminal alphabet (it may be omitted if $T = O$), w_0 is the initial multiset, and R is a (non-empty) finite set of rules. In the accepting case, T is the input alphabet, and the computation starts when an arbitrary multiset over T is added to w_0. Consider a multiset rewriting system Π with alphabet O. A configuration is a multiset u. The space $\mathcal{C}$ of configurations (i.e., of multisets over O) is essentially an $|O|$-dimensional space with non-negative integer coordinates. The relation $\Rightarrow$ induces an infinite labeled graph on $\mathcal{C}$ (with each configuration labeling a node and each set of rules whose application leads from a configuration C to a configuration C' labeling an edge leading from C to C'). The halting configurations (and only these) have out-degree zero.

Maximal Parallelism

We now consider the case that instead of applying one rule at each step, a multiset of rules may be applied. Moreover, it is required that the chosen multiset of rules is not extendable; this is the usual definition of maximally parallel transitions ([252]). We refer to this variant as P systems for short, see Note 1.1.

Features Maximal parallelism is the most common computation mode in membrane computing, see also Definition 4.8 in [185].

Priorities are subsumed by conditional contexts. This statement is valid even when the rules are not restricted to non-cooperative ones, and when determinism is not required, in either derivation mode (also see [172]). From [172] we already know that in the case of rules without context conditions, the context conditions in the new rules are only sets of atomic inhibitors, which also follows from the construction given above. If a fixed enumeration of the elements of the alphabet is assumed, then multisets are isomorphic to vectors. In that sense, sequential multiset processing corresponds to vector addition systems (see, e.g., [170]). The most famous membrane computing model where determinism is a criterion of universality versus decidability is the model of catalytic P systems, see [171] and [197]. It is also known that non-cooperative rewriting P systems with either promoters or inhibitors are computationally complete, [133], [134], [135]. It is natural to ask a question about the deterministic case, addressed in Section 2.2.

Note 1.1 *In fact, there exists a large variety of models and variants of P systems, including the ones with sequential mutliset rewriting. Yet, maximal parallelism remains by far the most studied transition mode, so the maximal parallelism is the default mode of P systems, and it is assumed unless stated otherwise.*

Reversibility and Determinism

Reversibility is an important property of computational systems. It has been well studied for circuits of logical elements ([165]), circuits of memory elements ([234]), cellular automata ([235]), Turing machines ([143], [237]), register machines ([236]). Sequential reversible P systems have been considered in [216], in the energy-based model, simulating Fredkin gates and thus reversible circuits. It is known that (see, e.g, [170]) the generative power of sequential multiset rewriting systems equals $PsMAT$, even without requiring additional properties like reversibility. In connection with reachability issues, we would also like to mention the connection with temporal logic [164]. Self-stabilization is a known concept in conventional distributed computing, introduced by E. Dijkstra in [163], as well as in systems biology. Studying all these properties is quite important for multiset processing; this direction is presented in Sections 2.3 and 2.4.

Throughout this thesis, and in particular in this section, by *reachable* we mean reachable from the initial configuration(s). We now define two properties; extending the requirement from reachable configurations to all configurations, we obtain their strong variants (in case of accepting systems, the initial configurations are obtained by adding arbitrary multisets over a fixed subalphabet to a fixed initial multiset).

Definition 1.5 *We call a multiset rewriting system* Π **strongly reversible** *if every configuration has in-degree at most one;* Π *is called* **reversible** *if every reachable configuration has in-degree at most one. We call* Π **strongly deterministic** *if every configuration has out-degree at most one;* Π *is called* **deterministic** *if every reachable configuration has out-degree at most one.*

Note that for strong reversibility it is crucial that in-degree is the number of transitions from all preimages, not only from the reachable ones, and that for strong determinism the out-degree of all configurations, not only of the reachable ones, is at most one. On the other hand, the properties reversibility and determinism only refer to the configurations in the actual computation of the system.

Result of Computations

We now proceed by defining a computation as a sequence of transitions, starting in the initial configuration, and ending in some halting configuration if it is finite. The result of a halting computation is the number of terminal objects inside the system when it halts (or the number of input objects when the system starts, in the accepting case). The set $N(\Pi)$ of numbers generated by a multiset processing system Π is the set of results of all its computations. The family of number sets generated by reversible multiset rewriting systems with features α is denoted by $NRMR(\alpha)_T$, where $\alpha \subseteq \{ncoo, coo, pro, inh, Pri\}$ and the braces of the set notation are omitted, with *ncoo*, *coo*, *pro*, *inh*, and *Pri* meaning that we use non-cooperative rules, cooperative rules, promoters, inhibitors, and priorities on the rules, respectively. Subscript T means that only terminal objects contribute to the result of computations; if $T = O$, we omit specifying it in the description and we then also omit the subscript T in the notation. In the case of accepting systems, we write N_a instead of N, and subscript T has no meaning. For strongly reversible systems, we replace R by R_s. For deterministic (strongly deterministic) systems, we replace R by D (D_s, respectively). In the maximally parallel case, we replace MR by OP in the notation.

We say that a class of systems generating or accepting NRE is universal.

Representing words Words may be represented by string-objects, for both theoretical research and applications, see Chapter 7 of [257]. Represent a word may be represented by a single symbol object, or by a few objects of the form (letter,position) as in, e.g., [131], [132]. Positions of the letters in a word may be represented by nested membranes. The corresponding letters can be then encoded by objects in the associated regions, membrane types or membrane labels. Working with such a representation, even implementing a rule $a \to bc$ requires sophisticated types of rules, like creating a membrane around existing membrane, as defined in [144]. In accepting mode, linear order is induced by the order in which the system brings the input symbols inside, see, e.g., [155]. The model of languages generated by systems with parallel applications of non-cooperative rules that rewrite symbol objects and/or send them between the regions has been introduced already in 2000, [258].

1.3.2 Transitional P systems

In the following we consider multiset processing distributed over a tree structure. A membrane system is defined by a construct

$\Pi = (O, \mu, w_1, \cdots, w_m, R_1, \cdots, R_m, i_0)$, where

O is a finite set of objects,

μ is a hierarchical structure of membranes, bijectively labeled by $1, \cdots, m$, (see Note 1.2), the interior of each membrane defines a region; the environment is referred to as region 0,

w_i is the initial multiset in region $i, 1 \leq i \leq m$,

R_i is the set of rules of region $i, 1 \leq i \leq m$,

i_0 is the output region; when languages are considered, $i_0 = 0$ is assumed.

Note 1.2 *For specific purposes, it may be convenient to use different symbols as membrane labels. In that case, the role of* $1, \cdots, m$ *is played by the order in which multisets* w_i *and sets* R_i *are listed.*

The rules of a membrane systems have the form $u \to v$, where $u \in O^+$, $v \in (O \times Tar)^*$. The target indications from $Tar = \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$ are written as a subscript, and target *here* is typically omitted. In case of non-cooperative rules, $u \in O$. We also use the target $in!$, meaning that the corresponding object is sent to all inner membranes; a necessary number of copies is made. (This last case is used for synchronization in P systems, when the rules are specified without even knowing the number of inner membranes).

The rules are applied in maximally parallel way: no further rule should be applicable to the idle objects. In case of non-cooperative systems, the concept of maximal parallelism is the same as in L systems: all objects evolve by the associated rules in the corresponding regions (except objects a in regions i such that R_i does not contain any rule $a \to u$, but these objects do not contribute to the result). The choice of rules is non-deterministic.

A configuration of a P system is a construct which contains the information about the hierarchical structure of membranes as well as the contents of every membrane at a definite moment of time. The process of applying all rules which are applicable in the current configuration and thus obtaining a new configuration is called a transition. A sequence of transitions is called a computation. The computation halts when such a configuration is reached that no rules are applicable. The result of a (halting) computation is the *sequence* of objects sent to the environment (all the permutations of the symbols sent out in the same time are considered). The language $L(\Pi)$ generated by a P system Π is the union of the results of all computations. The family of languages generated by non-cooperative transitional P systems with at most m membranes is denoted by $LOP_m(ncoo, tar)$. If the number of membranes is not bounded, m is replaced by $*$ or omitted. If the target indications of the form in_j are not used, tar is replaced by out.

Membrane creation

The transitional model can be enhanced with *membrane creation* (*mcre*) and *membrane dissolution* (δ) possibilities. In the notation we then add H (the set of labels for membranes, say, of cardinality n) after O, and we specify n sets of rules instead of m ones.

These rules can be of the forms (1) $a \to v$ and (2) $a \to [\ b\ ]_i$, where $a, b \in O$, $i \in H$, and either $v \in (O \times tar)^*$ or $v \in (O \times tar)^*\{\delta\}$, with $tar = \{here, out\} \cup \{in_j \mid j \in H\}$. The presence of δ on the right-hand side of a rule means that the application of the rule leads to the dissolution of the membrane (and all its contents, objects and membranes alike, become elements of the surrounding region). The skin membrane is never dissolved. A rule $a \to [\ b\ ]_i$ of type (2) means that object a produces a new membrane, with label i, containing the object b.

We also mention restricted membrane creation (denoted $mcre_r$): in region i it is only possible to create membranes with label i.

Knowing the label of the membrane, we also know the rules associated with it. We recall that the number of membrane labels (i.e., kinds of membranes) is n and the rules are associated to the membrane labels, while the number of membranes initially is m and can change during the computation. The rules are used in the non-deterministic (the objects and the rules are chosen non-deterministically) maximally parallel way (no further object can evolve after having chosen the objects for the rules), thus obtaining transitions between a configuration of the system to another one.

Computing power of the transitional model Almost all known characterizations and even bounds for generative power of different variants of membrane systems with various ingredients and different descriptional complexity bounds are expressed in terms of REG, MAT, $ET0L$ and RE, their length sets and Parikh sets (and much less often in terms of FIN or other subregular families, or CF or CS, or those accepted by log-tape bounded Turing machines, [158], [196]). The known technique of flattening the structure (this is "folklore" in membrane computing; see, e.g., [266], [185]). It is known (see, e.g., [7]) that computing all transitions from a configuration with k objects takes polynomial time with respect to k. The regularity of It has been shown already in [258] that non-cooperative transitional P systems generate at least all regular languages. It is a natural to ask the question about their exact power, addressed in Section 2.1.

Another important question was to determine the computational power of non-cooperative P systems with dynamic membrane structure (e.g., with membrane dissolution and creation); this research direction is presented in Section 2.5. An even more difficult question, about the power of P systems with active membranes without polarization, has been formulated and solved, see Section 4.2.

1.3.3 Symport/antiport

Definition 1.6 *A* P system (of degree $d \geq 1$) with antiport and/or symport rules *(in this section called P system for short) is a construct*

$$
\begin{array}{rcl}
\Pi & = & (O, T, E, \mu, w_1, \cdots, w_d, R_1, \cdots, R_d, i_0), \text{ where} \\
O & & \text{is the alphabet of objects;} \\
T \subseteq O & & \text{is the alphabet of terminal objects;} \\
E \subseteq O & & \text{is the set of objects occurring unboundedly in the environment;} \\
\mu & & \text{is a membrane structure consisting of } d \text{ membranes} \\
 & & \text{(usually labeled } i \text{ and represented by corresponding} \\
 & & \text{brackets } [\ \text{ and } \ ]_i,\ 1 \leq i \leq d); \\
w_i, & & 1 \leq i \leq d, \text{ are strings over } O \text{ associated with the regions} \\
 & & 1, 2, \cdots, d \text{ of } \mu; \text{ they represent multisets of objects initially} \\
 & & \text{present in the regions of } \mu; \\
R_i, & & 1 \leq i \leq d, \text{ are finite sets of rules of the form } (u, out; v, in), \text{ with} \\
 & & u \neq \lambda \text{ and } v \neq \lambda \text{(antiport rule), and } (x, out) \text{ or } (x, in), \text{ with} \\
 & & x \neq \lambda \text{(symport rule)}; \\
i_0, & & 1 \leq i_0 \leq d, \text{ specifies the output membrane of } \Pi.
\end{array}
$$

The antiport rule $(u, out; v, in)$ in R_i exchanges the multiset u inside membrane i with the multiset v outside membrane i; the symport rule (x, out) sends the multiset x out of membrane i and (x, in) takes x in from the region surrounding membrane i (if i is the skin membrane, then x has to contain at least one symbol not in E). The membrane structure μ and the multisets represented by w_i, $1 \leq i \leq d$, in Π constitute the *initial configuration* of the system.

In the *maximally parallel mode*, a transition from one configuration to another one is obtained by the application of a maximal multiset of rules. The system continues maximally parallel transition steps until there remain no applicable rules in any region of Π; then the system halts. We consider the vector of multiplicities of objects from T contained in the output membrane i_0 at the moment when the system halts as the *result* of the underlying computation of Π; observe that here we do not count the non-terminal objects present in the output membrane. The set of results of all halting computations possible in Π is denoted by $Ps(\Pi)$, respectively. The family of all sets of vectors of natural numbers computable by P systems with d membranes and using rules of type α is denoted by $PsO_EP_d(\alpha)$. When the parameter d is not bounded, it is replaced by $*$.

We consider variants of P systems using only rules of very restricted types α: $anti_k$ indicates that only antiport rules of weight at most k are used, where the *weight* of an antiport rule $(u, out; v, in)$ is defined as $\max\{|u|, |v|\}$; sym_k indicates that only symport rules of weight at most k are used, where the *weight* of a symport rule (x, out) or (x, in) is defined as $|x|$. For an antiport rule $(u, out; v, in)$, we may refine the weight by also considering the total number of symbols specified in the rule – the *size* of an antiport rule is defined as $|u| + |v|$; then $anti_k^s$ indicates that only antiport rules of weight at most k and size

at most s are used. For example, $anti_2^3$ allows antiport rules of the forms $(a, out; b, in)$, $(ab, out; c, in)$, and $(c, out; ab, in)$, whereas $anti_2$ (coinciding with $anti_2^4$) also allows rules of the form $(ab, out; cd, in)$, with a, b, c, d being single objects. The *size* of a symport rule equals its weight, hence, we do not need additional notions.

When using the *minimally parallel mode* $(amin)$, in each transition step we choose a multiset of rules from R in such a way that this chosen multiset includes at least one rule from every set of rules (in this chapter we only consider the partitioning of the rules according to the given membrane structure) containing applicable rules. In the *asynchronous* $(asyn)$ and the *sequential mode* $(sequ)$, in each transition step we apply an arbitrary number of rules/exactly one rule, respectively. The corresponding families of sets of vectors of natural numbers generated by P systems with d membranes and using rules of type α in the transition mode X are denoted by $PsO_EP_d^{(X)}(\alpha)$, $X \in \{amin, asyn, sequ\}$.

If at the end of a computation only a bounded number of at most k nonterminal objects remains in the output membrane, we replace the subscript E by $-k$; if we do not distinguish between terminal and nonterminal symbols, then we simply omit the subscript E.

All these variants of P systems with antiport and/or symport rules can also be considered as accepting devices, the input being given as the numbers of objects in the distinguished membrane i_0. In that case we write Ps_a instead of Ps; in the accepting case, the subscripts E and $-k$ to O are of no meaning and therefore omitted. The families specified by deterministic systems are denoted by adding D in front of O.

When we are only interested in the number of symbols in the output/input membrane, we replace Ps by N.

In the case that only m objects are needed to generate/accept a set of natural numbers, for all $X \in \{max, amin, asyn, sequ\}$ we obtain the families $NO_mP_d^{(X)}(\alpha)$ and $N_aO_mP_d^{(X)}(\alpha)$, respectively. We usually omit superscript (max).

The power of communication P systems with antiport and symport rules were introduced in [247] where the first results concerning computational completeness were established: $NRE = NOP_2(anti_2, sym_2) = NOP_5(anti_1, sym_2)$. The computational completeness with one membrane independently was shown in [189] and [178] as well as in [175], where this result was obtained based on a more general model with channels through membranes. A deterministic simulation of register machines by P systems with antiport rules of weight two first was established in [180]. Tissue P systems were introduced in [230], and tissue-like P systems with channel states were investigated in [182]. P systems with minimal symport and antiport rules first were investigated in [142], where nine membranes were used to achieve computational completeness. A deterministic proof using three cells for the tissue case first was presented in [280]. The descriptional complexity of P systems with respect to the number of objects first was considered in [254], where three objects were shown to be sufficient for obtaining computational completeness in four membranes. and for tissue P systems even with only one object computational completeness was established in [177]. universal P systems with a small number of antiport rules were described in [157], [176]. published in [198]. $NtOP_1$ $(sym_2) \subseteq NFIN$ from was shown in [188], respectively. Evolution-communication P systems were introduced in [149]. The minimally parallel mode was introduced in [152], yet the concept was already considered in [252], p. 84, and called

minimal synchronization there. A formal framework for P systems was developed in [185].

The first universal P system with symbol-objects has been reported in [157]: it had 73 symport/antiport rules. Together with some optimizations the authors decreased this number to 30, at a price of increasing the size of rules. A very interesting direction is decreasing this number; a significant improvement is described in Section 3.1.

Membrane division rules were added to (tissue) P systems with symport and antiport, to solve SAT in a uniform way, [255]. Decisional framework for active membranes has been formally described in [253]. The original definition of separation can be found in [80].

Symport rules move predefined groups objects to a neighboring region [247]. With symport-3 (i.e., symport rules only, of weight up to 3), one proved in [188] that 13 extra objects suffice for computational completeness. One naturally asks about characterization of the computational power of symport-3 with fewer superfluous objects. This direction is reported in Section 3.3.

Tissue

In tissue P systems, the role of membranes is played by membrane channels between cells as well as between the cells and the environment. In contrast to tree-like P systems, in tissue P systems every cell may have connection with the environment and every cell may have connection with any other cell, so the underlying topology may be any graph. We assume that we speak about the model where rules are assigned to membranes or channels.

In the tuple description of a P system, d (the number of cells) is added before O, μ is removed, ch (the set of edges over $\{0, 1, \cdots, d\}$) is added after w_d, and the sets of rules are defined for each channel in ch.

To denote (the results of computation of) all tissue P systems, we take the notation for tree-like P systems and replace P by tP.

For tissue P systems with antiport rules (here symport can be considered a particular case of antiport), a *rule* in $R_{(i,j)}$ is written as x/y ($xy \neq \lambda$) and its application means moving the objects specified by x from cell i (from the environment, if $i = 0$) to cell j, at the same time moving the objects specified by y in the opposite direction.

1.3.4 Active membranes

A P system *with active membranes* (of degree $m \geq 1$) is a construct of the form

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

where O is the alphabet of objects, $E = \{0, \cdots, n-1\}$ with $n \geq 1$ is the set of electrical charges (polarizations), μ is the membrane structure (with m membranes, bijectively labeled with $1, 2, \cdots, m$ and with initial polarization specified; by H we denote the set of labels $\{1, 2, \cdots, m\}$), $w_1, \cdots, w_m$ are strings over O indicating the multisets of objects at the beginning present in the m regions of μ, and R is a finite set of rules of the following forms:

(a) $[\ a \to v\ ]_h^i$, $a \in O$, $v \in O^*$, $h \in H$, $i \in E$
(evolution rules, used in parallel in the region of membrane h, provided that the polarization of the membrane is i);

(*b*) $a[\ ]_h^i \to [\ b\ ]_h^j$, $a, b \in O$, $h \in H$, $i, j \in E$
(communication rules, sending an object into a membrane, possibly changing the polarization of the membrane);

(*c*) $[\ a\ ]_h^i \to [\ \ ]_h^j b$, $a, b \in O$, $h \in H$, $i, j \in E$
(communication rules, sending an object out of a membrane, possibly changing the polarization of the membrane);

(*d*) $[\ a\ ]_h^i \to b$, $a, b \in O$, $h \in H$, $i \in E$
(membrane dissolution rules; in reaction with an object, the membrane is dissolved);

(*e*) $[\ a\ ]_h^i \to [\ b\ ]_h^j [\ c\ ]_h^k$, $a, b, c \in O$, $h \in H$, $i, j, k \in E$
(division rules for elementary membranes; in reaction with an object, the membrane is divided into two membranes with the same label, possibly of different polarizations, and the object specified in the rule is replaced in the two new membranes by possibly new objects).

(f_0) $[\ [\ \ ]_{h_1} [\ \ ]_{h_2}\ ]_{h_0} \to [\ [\ \ ]_{h_1}\ ]_{h_0} [\ [\ \ ]_{h_2}\ ]_{h_0}$, $h_0, h_1, h_2 \in H$
(polarizationless division rules for non–elementary membranes. If the membrane with label h_0 contains other membranes than those with labels h_1, h_2, these membranes and their contents are duplicated and placed in both new copies of the membrane h_0; all membranes and objects placed inside membranes h_1, h_2, as well as the objects from membrane h_0 placed outside membranes h_1 and h_2, are reproduced in the new copies of membrane h_0).

An output is associated with a halting computation – and only with halting computations – in the form of the objects sent into the environment during the computation. When using a P system Π for decision problems, we also specify an input membrane i_0, where we put the input to be analyzed is put in addition to the axiom w_{i_0}.

The rules of type (a) are considered to only involve objects, while all other rules are assumed to involve objects and membranes mentioned in their left-hand side. An application of a rule consists in subtracting a multiset described in the left-hand side from a corresponding region (i.e., associated to a membrane with label h and polarization i for rules of types (a), (c), (d) and (e), or immediately outer of such a membrane for rules of type (b)), adding a multiset described in the right-hand side of the rule to the corresponding region (that can be the same as the region from where the left-hand side multiset was subtracted, immediately inner or immediately outer, depending on the rule type), and updating the membrane structure accordingly if needed (changing membrane polarization, dividing or dissolving a membrane).

The rules can only be applied simultaneously if they involve different objects and membranes (we repeat that rules of type (a) are not considered to involve a membrane), and such parallelism is maximal if no further rules are applicable to objects and membranes that were not involved.

The power of active membranes P systems with active membranes [253] are parallel computation devices inspired by the internal working of biological cells. It is already known

that polynomial-space P systems and polynomial-space Turing machines are equal in computing power [263], but the proof of this result does not generalize to larger space bounds. The exponential-space case is considered in Section 4.1.

The first efficient *semi–uniform solution* to SAT was given by Gh. Păun in [253], using division for non–elementary membranes and three electrical charges. This result was improved by Gh. Păun, Y. Suzuki, H. Tanaka, and T. Yokomori in [260] using only division for elementary membranes. The probabilistic polynomial complexity class **PP**, also called Majority-P, has been introduced in [191]. Naturally, it is interesting to consider only two charges (polarizations, or states of membranes), as reported in Section 4.3 and in Section 4.2. The question about the power of P systems with active membranes without non-elementary membrane division beyond $NP \cup co - NP$ is considered in Section 4.4.

P. Sosík in [272] provides an efficient *semi–uniform solution* to QSAT (quantified satisfiability problem), a well known **PSPACE**–complete problem, in the framework of P systems with active membranes but using cell division rules for non–elementary membranes. It is well known that QSAT is a **PSPACE**–complete problem [246]. An interesting question is the role of polarizations. In Section 4.5 it is shown that QSAT can be solved even without polarizations.

Solving SAT in a **minimally parallel** way, using non-elementary membrane division (replicating both objects and inner membranes), [152]. However, non-elementary membrane division allows to rely on the synchronization via the global parallelism. A natural question is whether it could be avoided, shedding more light on the nature of minimal parallelism itself. This question is addressed in Section 4.6, using more polarizations and/or non-standard rules.

Active membranes - rule variants

If all rules do not depend on the membrane polarizations and do not modify them, we denote them as (a_0)–(f_0), and we omit the polarizations in the specification of rules.

Generally, rules of type (a) are executed in parallel, while at most one rule out of all rules of types $(b), (c), (d), (e)$ can be applied to the same membrane in the same step. We also speak about the sequential version

$$(a''_s)\ [\ a\ ]^{e}_{h} \rightarrow [\ u\ ]^{e'}_{h} \text{ for } a \in O,\ u \in O^*,\ h \in H \text{ and } e, e' \in E.$$

of rules (a) (let us use $''$ to indicate that the rule is allowed to change the polarization of the membrane) and their modifications $(b_0), (c_0), (d_0), (e_0)$, (a'_{0s}), $(b'_0), (c'_0), (e'_0)$ (here, 0 represents that the rules neither distinguish polarization nor change it, while $'$ means that the rule is allowed to change membrane label).

1.3.5 Energy assigned to membranes

Considering the energy balancing of processes in a cell first was investigated in [259] and then in [166]. There the energies of all rules to be used in a given step in a membrane are summed up; if the total amount of energies is positive ([259]) or within a given range ([166]), then this multiset of rules can be applied if it is maximal with this property. membrane systems

where the rules are directly assigned to the membranes (as, for example, in [175]). For sequential variants of P systems see, for example, [167] and [168]. In the sequential mode as this was originally defined for P systems with unit rules and energy assigned to membranes in [173]. *Energy-based P systems*, [259], [217, 216, 215], are a model of computation in the framework of Membrane Computing in which a given amount of energy is associated to each object, and the energy manipulated during computations is taken into account by means of conservative rules. In [217, 216] energy-based P systems are used to simulate Fredkin gates and Fredkin circuits, respectively. In [215] it is proved that energy-based P systems working in the sequential way and using a form of local priorities associated to the rules are computationally complete. However, in [215] the precise characterization of the computational power of energy-based P systems without priorities is left open. This question is considered in Section 4.8.

A *P system of degree* $d+1$ *with unit rules and energy assigned to membranes* is a construct Π of the form

$$\Pi = (O, \mu, e_1, \cdots, e_d, w_0, w_1, \cdots, w_d, R_1, \cdots, R_d), \text{ where}$$

- O is an alphabet of *objects*;
- μ is a *membrane structure* (with the membranes labeled by numbers $0, \cdots, d$ in a one-to-one manner);
- $e_1, \cdots, e_d \in \mathbb{N}$ are the initial energy values assigned to the membranes $1, \cdots, d$;
- $w_0, \cdots, w_d$ are multisets over V associated with the membrane regions $0, \cdots, d$ of μ;
- $R_1, \cdots, R_d$ are finite sets of *unit rules* associated with the membranes $1, \cdots, d$, which are of the form $(\alpha : a, \Delta e, b)$, where $\alpha \in \{in, out\}$, $a, b \in O$, and $|\Delta e|$ is the amount of energy that, for $\Delta e \geq 0$, is added to or, for $\Delta e < 0$, is subtracted from e_i (the energy assigned to membrane i) by the application of the rule.

Instead of $R_1, \cdots, R_d$, we can specify only one set of rules R with

$$R := \{(\alpha_i : a, \Delta e, b) \mid (\alpha : a, \Delta e, b) \in R_i,\ 1 \leq i \leq d\}.$$

In each step, only one rule may be applied to each membrane. This renames the object, moves it across membrane i in the direction specified by α, and changes its energy from e_i to $e_i + \Delta e$ (the new value must be non-negative in order for the rule to be applicable). Moreover, we consider some sort of priorities: out of the applicable rules to a membrane, one of the rules with $\max |\Delta e|$ has to be used.

A sequence of transitions is called a *computation*; it is *successful* if and only if it halts. The *result* of a successful computation is considered to be the distribution of energy values assigned to the membranes (a non-halting computation does not produce a result). The set of Parikh vectors generated by Π (in the representation as multisets over $\{e_i \mid 1 \leq i \leq d\}$) is denoted by $L(\Pi)$. Observe that in this model we do not take into account the environment and the energy assigned to the skin membrane.

1.3.6 Energy-based P systems

In the following model the energy may be present in regions rather than being associated to membranes. We now recall the definition of energy-based P systems as given in [216].

An *energy-based P system* of degree $m \geq 1$ is a tuple

$$\Pi = (A, \varepsilon, \mu, e, w_1, \cdots, w_m, R_1, \cdots, R_m, i_{in}, i_{out}), \text{ where:}$$

- A is a finite set of objects called the alphabet;
- $\varepsilon : A \to \mathbb{N}$ is a mapping that associates to each object $a \in A$ the value $\varepsilon(a)$ (also denoted by ε_a), which can be viewed as the "energy value of a". If $\varepsilon(a) = \ell$, we also say that object a *embeds* ℓ units of energy;
- μ is a description of a tree structure consisting of m membranes, injectively labeled with elements from the set $\{1, \cdots, m\}$;
- $e \notin A$ is a special symbol denoting one free energy unit;
- w_i, for $1 \leq i \leq m$, specifies multisets (over $A \cup \{e\}$) of objects initially present in region i. We will sometimes assume that the number of e's (but not of objects from A) in some regions of the system is unbounded;
- R_i, for $1 \leq i \leq m$, is a finite set of multiset rewriting rules over $A \cup \{e\}$ associated with region i. Rules can be of the following types:

$$ae^k \to (b, p) \qquad \text{and} \qquad b \to (a, p)e^k \tag{1.7}$$

  where $a, b \in A$, $p \in \{here, in(j), out \mid 1 \leq j \leq m\}$, and k is a non-negative integer. We also omit target *here*, together with a comma and parentheses. Rules satisfy the conservativeness condition $\varepsilon(a) + k = \varepsilon(b)$;
- $i_{in} \in \{1, 2, \cdots, m\}$ specifies the input region of Π;
- $i_{out} \in \{0, 1, \cdots, m\}$ specifies the output region of Π ($i_{out} = 0$ corresponds to the environment).

Remark 1.1 *In the above definition we excluded rules of types $e \to (e, p)$, originally included in [216]. It is easy to see that this does not influence the computational power of energy-based P systems.*

When a rule of the type $ae^k \to (b, p)$ is applied, the object a, in presence of k free energy units, is allowed to be transformed into object b (note that $\varepsilon_a + k = \varepsilon_b$, for the conservativeness condition). If $p = \text{here}$, then the new object b remains in the same region; if $p = \text{out}$, then b exits from the current membrane. Finally, if $p = \text{in}(j)$, then b enters into the membrane labeled with j, which must be directly contained inside the membrane associated to the region where the rule is applied. The meaning of rule $b \to (a, p)e^k$, where k is a positive integer number, is similar: the object b is allowed to be transformed into object a by releasing k units of free energy (also here, $\varepsilon_b = \varepsilon_a + k$). As above, the new object a may

optionally move one level up or down into the membrane structure. The k free energy units might then be used by another rule to produce "more energetic" objects from "less energetic" ones. When $k = 0$ the rule $ae^k \to (b, p)$, also written as $a \to (b, p)$, transforms the object a into the object b (note that in this case $\varepsilon_b = \varepsilon_a$) and moves it (if $p \neq$ here) upward or downward into the membrane hierarchy, without acquiring or releasing any free energy unit. A similar observation applies to rules $b \to (a, p)e^k$ when $k = 0$.

Rules can be applied either in the sequential or in the maximally parallel mode. When working in the *sequential mode*, at each computation step (a global clock is assumed) exactly one enabled rule is non-deterministically chosen and applied in the whole system. When working in the *maximally parallel mode*, instead, at each computation step in each region of the system a maximal multiset of applicable rules is selected, and then all those rules are applied in parallel. Here *maximal* means that no further rule is applicable to objects that are "idle", that is, not already used by some other rule. If two or more maximal sets of applicable rules exist, then one of them is non-deterministically chosen.

A configuration of Π is the tuple $(M_1, \cdots, M_m)$ of multisets (over $A \cup \{e\}$) of objects contained in each region of the system; $(w_1, \cdots, w_m)$ is the *initial* configuration. A configuration where no rule can be further applied on is said to be *final*. A computation is a sequence of transitions between configurations of Π, starting from the initial one. A computation is *successful* if and only if it reaches a final configuration or, in other words, it *halts*. The multiset $w_{i_{\text{in}}}$ of objects occurring inside the input membrane is the *input* for the computation, whereas the multiset of objects occurring inside the output membrane (or ejected from the skin, if $i_{\text{out}} = 0$) in the final configuration is the *output* of the computation. A non-halting computation produces no output. As an alternative, we can consider the Parikh vectors associated with the multisets, and see energy-based P systems as computing devices that transform (input) Parikh vectors to (output) Parikh vectors. We may also assume that energy-based P systems have $\alpha \geq 1$ input membranes and $\beta \geq 1$ output membranes, instead of one. This modification does not increase the computational power of energy-based P systems, since for any fixed value of $\alpha \geq 1$ (resp., $\beta \geq 1$), the set $\mathbb{N}^\alpha$ (resp., $\mathbb{N}^\beta$) is isomorphic to $\mathbb{N}$, as it is easily shown by using the well known Cantor mapping.

In what follows sometimes we will use energy-based P systems as generating devices: we will disregard the input membrane, and will consider the multisets (or Parikh vectors) produced in the output membrane at the end of the (halting) non-deterministic computations of the system. In particular, in the output multisets we will only count the number of free energy units contained in the β output regions in the final configuration. We will denote the family of β-dimensional vectors generated in this way by energy-based P systems with at most m membranes and unbounded energy by $Ps(\beta)OP_m(energy_*)$. The union of all these classes for β ranging through the set of all non-negative integers is denoted by $PsOP_m(energy_*)$. When $\beta = 1$, the class $Ps(\beta)OP_m(energy_*)$ will be written as $\mathbb{N}OP_m(energy_*)$. In all cases we will replace the subscript m by $*$ if no bound is placed on the number of membranes. If instead of maximal parallelism we assume that the P system evolves sequentially, we will add the superscript *sequ* to P in the notation.

1.3.7 Polymorphism

The biological idea that different actions are carried out by different objects, which too can be acted upon. (This last idea was also considered in, e.g., [167] and [2]. Suppose we want to be able to manipulate the rules of the system during its computation. A number of papers has been written in this direction (see, e.g., GP systems [167], rule creation [140], activators [2], inhibiting/deinhibiting rules [151] and symport/antiport of rules [150]), but in most of them the rules are predefined in the description of the system. a problem informally stated by Gh. Păun in Section "Where Is the Nucleus?" of [248] by proposing a computational variant based on one simple difference: the rules are taken from the current configuration rather than from the description of the P system itself. The idea of a nucleus was also considered in [274], but such a presentation had the following drawbacks. First, one described the dynamics of the rules in a high-level programming language (good for simulators, but otherwise too powerful extension having the power of conventional computers directly built into the definition). Second, this dynamics of the rules did not depend on the actual configuration of the membrane system (no direct feedback from objects to rules). In the polymorphic model, the dynamics of rules is defined by exactly the same mechanism as the standard dynamics of objects.

We define a polymorphic P system as a tuple

$$\Pi = (O, T, \mu, w_s, w_{1L}, w_{1R}, \cdots, w_{mL}, w_{mR}, \varphi, i_{out}),$$

where O is a finite alphabet, μ is a tree structure consisting of $2m+1$ membranes, bijectively labeled by elements of $H = \{s\} \cup \{iL, iR \mid 1 \leq i \leq m\}$ (the skin membrane is labeled by s; we also require for $1 \leq i \leq m$ that the parent membrane of iL is the same as the parent membrane of iR), w_i is a string describing the contents of region i, $1 \leq i \leq m$, and φ is a mapping from $\{1, \cdots, m\}$ to the features of the rules described below. The set $T \subseteq O$ describes the output objects, while $i_{out} \in H \cup \{0\}$ is the output region (0 corresponds to the environment).

Notice that the rules of a P system are not explicitly given in its description. Essentially, such a system has m rules, and these rules change as the contents of regions other than skin changes. Initially, for $1 \leq i \leq m$ rule $i : w_{iL} \to (w_{iR}, \varphi(i))$ belongs to the region defined by the parent membrane of iL and iR. If w_{iL} is empty, then the rule is considered disabled. For every step of the computation each rule is defined in the same way, taking the current contents of iL and iR instead of initial ones.

In what follows we mainly consider a single feature, i.e., target indications. In this case, the range of φ is $Tar = \{in_i \mid i \in H\} \cup \{here, out\}$. We denote the class of all polymorphic P systems with cooperative rules and target indications and at most k membranes by

$$OP_k(polym_{+d}(coo), tar).$$

In the notation above, the number k is replaced by $*$ or omitted if no bound is specified. The subscript $+d$ means that the rules can be disabled; we write $-d$ instead, if w_{iL} is never empty for $1 \leq i \leq m$ during any computation. We prefix this notation with D if we restrict the class to the deterministic systems (for every input if it is specified, see below).

A computation is a sequence of configurations starting in the initial configuration, corresponding to the transitions induced by non-deterministic maximally parallel application

of rules; it is called halting if no rules are applicable to the last configuration. In the latter case the multiset of objects from T in region i_{out} is called the result.

If we want to compute instead of generating, we extend the tuple Π by the description of the input as follows. In the definition of the P system, we insert the input alphabet $\Sigma \subset O$ after O and we insert the input region i_{in} after φ. In this case, the input multiset over Σ is added to $w_{i_{in}}$ before the computation starts. If we want to accept instead of computing, we remove T and i_{out} from the description of the P system; the input is considered accepted if and only if the system may halt. If we want to decide instead of computing, we construct a system that always halts with either `yes` or `no` in the output region, such that this answer uniquely depends on the input; the input is accepted if and only if the answer is `yes`.

The class of polymorphic P systems with cooperative rules and target indications, allowing disabled rules, with at most k membranes, defines a family of sets of numbers, of sets of vectors or of functions, respectively denoted by

$$NOP_k(polym_{+d}(coo), tar),\ PsOP_k(polym_{+d}(coo), tar),$$
$$fDOP_k(polym_{+d}(coo), tar).$$

In a similar way it is possible to replace cooperative rules with a more restricted set, remove target indications or add more features to the polymorphic P systems, modifying the notation accordingly.

1.4 String-objects. String Replication

Let us recall the basics of P systems with string objects and input. The membrane structure μ is defined as a rooted tree with nodes labeled $1, \cdots, p$. The objects of the system are strings (or words) over a finite alphabet O. A sub-alphabet $\Sigma \subseteq O$ is specified, as well as the input region i_0, $1 \leq i_0 \leq p$. We now define the model with cooperative rewriting rules (i.e. string rewriting rules, not limited by context-free ones) with string replication and target indications.

A rule $a \to u_1$, where $a \in O^+$ and $u_1 \in O^*$, can transform any string of the form $w_1 a w_2$ into $w_1 u_1 w_2$. Application of a rule $a \to u_1||u_2|| \cdots ||u_k$ transforms any string of the form $w_1 a w_2$ into the multiset of strings $w_1 u_1 w_2$, $w_1 u_2 w_2$, $\cdots$, $w_1 u_k w_2$. If in the right side of the rule (u_i, t) is written instead of some u_i, $1 \leq i \leq k$, $t \in \{out\} \cup \{in_j \mid 1 \leq j \leq p\}$, then the corresponding string would be sent to the region specified by t.

Hence, such a P system is formally defined as follows:

$$\begin{aligned}
\Pi &= (O, \Sigma, \mu, M_1, \cdots, M_p, R_1, \cdots, R_p, i_0), \text{ where} \\
M_i &\quad \text{is the multiset of strings initially present in region } i, 1 \leq i \leq p, \\
R_i &\quad \text{is the set of rules of region } i,\ 1 \leq i \leq p, \\
&\quad \text{and } O, \Sigma, \mu, i_0 \text{ are described above.}
\end{aligned}$$

The initial configuration contains the input string(s) over Σ in region i_0 and strings M_i in regions i. Rules of the system are applied in parallel to all strings in the system. The computation consists in non-deterministic application of the rules in a region to a string

in that region. The computation halts when no rules are applicable. The result of the computation is the set of all words sent out of the outermost region (called skin).

1.4.1 Active membranes

Definition 1.7 *A P system with string-objects and input is a tuple*

$\Pi = \big(O, \Sigma, H, E, \mu, M_1, \cdots, M_p, R, i_0\big)$, *where:*

- O *is the working alphabet of the system (the objects are strings over* O*),*
- Σ *is an input alphabet,*
- H *is an alphabet whose elements are called labels,* i_0 *identifies the input region,*
- E *is the set of polarizations,*
- μ *is a membrane structure (a rooted tree) consisting of* p *membranes injectively labeled by elements of* H*,*
- M_i *is an* initial *multiset of strings over* O *associated with membrane* i*,* $1 \leq i \leq p$*,*
- R *is a finite set of rules defining the behavior of objects from* O^* *and of membranes labeled by elements of* H*.*

A configuration of a P system is its "snapshot", i.e., the current membrane structure and the multisets of string-objects present in regions of the system. The initial configuration is $C_0 = (\mu, M_1, \cdots, M_p)$. Each subsequent configuration C' is obtained from the previous configuration C by maximally parallel application of rules to objects and membranes. This is denoted by $C \Rightarrow C'$ (no further rules are applicable together with the rules that transform C into C'). A computation is thus a sequence of configurations starting from C_0, respecting relation $\Rightarrow$ and ending in a halting configuration (i.e., such one that no rules are applicable). If M is a multiset of strings over the input alphabet $\Sigma \subseteq O$, then the *initial configuration* of a P system Π with an input M over alphabet Σ and input region i_0 is $(\mu, M_1, \ldots, M_{i_0-1}, M_{i_0} \cup M, M_{i_0+1}, \ldots, M_p)$.

To speak about P systems with active membranes, we need to specify the rules, i.e., the elements of the set R in the description of a P system. Due to the nature of the problem of the applications that we consider, the standard model was generalized in the following:

- Cooperative rules: a rule operates on a substring of an object (otherwise, the system cannot even distinguish different permutations of a string); this feature is represented by a superscript * in the rule types;
- String replication (to return the result without removing it from the dictionary);
- Membrane creation (to add words to the dictionary).

Hence, the rules can be of the following forms:

(a^*) $[\ a \to b\ ]_h^e$ for $h \in H, e \in E, a, b \in O^*$- evolution rules
(associated with membranes and depending on the label and the polarization of the membranes, but not directly involving the membranes: the membranes are neither taking part in the application of these rules nor are they modified by them);

(a_r^*) $[\ a \to b||c\ ]_h^e$ for $h \in H, e \in E, a, b, c \in O^*$ (as above, but with string replication);

(b^*) $a[\quad]_h^{e_1} \to [\ b\ ]_h^{e_2}$ for $h \in H, e_1, e_2 \in E, a, b \in O^*$ - communication rules
(an object is introduced into the membrane, possibly modified; the polarization of the membrane can be modified, but not its label);

(c^*) $[\ a\ ]_h^{e_1} \to [\quad]_h^{e_2} b$ for $h \in H, e_1, e_2 \in E, a, b \in O^*$ - communication rules
(an object is sent out of the membrane, possibly modified; also the polarization of the membrane can be modified, but not its label);

(d^*) $[\ a\ ]_h^e \to b$ for $h \in H, e \in E, a, b \in O^*$ - dissolving rules
(in reaction with an object, a membrane can be dissolved, while the object specified in the rule can be modified);

(g^*) $[\ a\ \to [\ b\ ]_g^{e_2}\ ]_h^{e_1}$ for $g, h \in H,\ e_1, e_2 \in E,\ a, b \in O^*$ - membrane creation rules
(an object is moved into a newly created membrane, possibly modified).

Additionally, we will write $\emptyset$ in place of some strings on the right-hand side of the rules, meaning that the entire string is deleted.

The rules of types $(a^*), (a_r^*)$ and (g^*) are considered to only involve objects, while all other rules are assumed to involve objects and membranes mentioned in their left-hand side. An application of a rule consists in replacing a substring described in the left-hand side of a string in the corresponding region (i.e., associated to a membrane with label h and polarization e for rules of types $(a^*), (a_r^*)$ and (d^*), or associated to a membrane with label h and polarization e_1 for rules of type (c^*), or immediately outer of such a membrane for rules of type (b^*)), by a string described in the right-hand side of the rule, moving the string to the corresponding region (that can be the same as the source region immediately inner or immediately outer, depending on the rule type), and updating the membrane structure accordingly if needed (changing membrane polarization, creating or dissolving a membrane). Only the rules involving different objects and membranes can only be applied in parallel; such parallelism is maximal if no further rules are applicable in parallel.

1.4.2 Insertion/deletion

An insertion-deletion system is a grammar-like string-processing formal computational model, making sense even without the distributed structure. We will, however, skip the definitions and notations that are different from insertion-deletion P systems, and most of the presentation for the non-distributed model, referring the interested reader to, e.g., [88], proceeding with the P systems framework.

Let V be and alphabet and let I, D be finite sets of triples of strings (over V) of the form (u, α, v), $\alpha \neq \lambda$. An insertion rule $(u, \alpha, v) \in I$ indicates that the string α can be inserted in between u and v, while a deletion rule $(u, \alpha, v) \in D$ indicates that α can be removed

from the context (u, v). As stated otherwise, $(u, \alpha, v) \in I$ corresponds to the rewriting rule $uv \to u\alpha v$, and $(u, \alpha, v) \in D$ corresponds to the rewriting rule $u\alpha v \to uv$. We refer by $\Longrightarrow$ to the relation defined by an insertion or deletion rule. The complexity of sets of insertion and deletion rules I, D is described by the vector $(n, m, m'; p, q, q')$ called *size*, where

$$\begin{aligned} n &= \max\{|\alpha| \mid (u, \alpha, v) \in I\}, \qquad p = \max\{|\alpha| \mid (u, \alpha, v) \in D\}, \\ m &= \max\{|u| \mid (u, \alpha, v) \in I\}, \qquad q = \max\{|u| \mid (u, \alpha, v) \in D\}, \\ m' &= \max\{|v| \mid (u, \alpha, v) \in I\}, \qquad q' = \max\{|v| \mid (u, \alpha, v) \in D\}. \end{aligned}$$

An *insertion-deletion P system* is the construct

$\Pi = (O, T, \mu, M_1, \cdots, M_n, R_1, \cdots, R_n)$, where

- O is a finite alphabet,
- $T \subseteq O$ is the terminal alphabet,
- μ is the membrane (tree) structure of the system which has n membranes (nodes) and it can be represented by a word over the alphabet of correctly nested marked parentheses,
- M_i, for each $1 \leq i \leq n$ is a finite language associated to the membrane i,
- R_i, for each $1 \leq i \leq n$ is a set of rules associated to membrane i, of the following forms: $(u, x, v; tar)_a$, where (u, x, v) is an insertion rule, and $(u, x, v; tar)_e$, where (u, x, v) is a deletion rule, and the *target indicator tar* is from the set $\{here, in_j, out \mid 1 \leq j \leq n\}$, where j is a label of immediately inner membrane of membrane i.

An n-tuple $(N_1, \cdots, N_n)$ of finite languages over O is called a configuration of Π. The transition between the configurations consists of applying the insertion and deletion rules in parallel to all possible strings, non-deterministically, and following the target indications associated with the rules.

A sequence of transitions between configurations of a given insertion-deletion P system Π starting from the initial configuration is called a computation with respect to Π. We say that Π generates $L(\Pi)$, the result of its computations. It consists of all strings over T ever sent out of the system during its computations.

Insertion-deletion *tissue* P systems are defined in an analogous manner, however the membranes are not necessarily arranged in a tree structure; insertion and deletion rules sending strings between any regions i and j are allowed. This means that the rules have the same form except the set used for tar, which becomes $\{j \mid 1 \leq j \leq n\}$.

We denote by $ELSP_k(ins_p^{m,m'}, del_p^{q,q'})$ the family of languages $L(\Pi)$ generated by insertion-deletion P systems with $k \geq 1$ membranes and insertion and deletion rules of size at most $(n, m, m'; p, q, q')$. We omit the letter E if $T = O$ and replace k by $*$ if k is not fixed. In this thesis we also consider insertion-deletion P systems where deletion rules have a priority over insertion rules; the corresponding class is denoted as $(E)LSP_k(ins_p^{m,m'} < del_p^{q,q'})$. Letter "t" is inserted before P to denote classes for the tissue P systems case, e.g., $ELStP_k(ins_p^{m,m'}, del_p^{q,q'})$.

Sometimes we are only interested in the multiplicities of each symbol in the output words, i.e., in the Parikh image of the languages described above. In this case we say that a family of sets of vectors is generated and we replace L by Ps in the notation above, e.g., $PsStP_k(ins_p^{m,m'}, del_p^{q,q'})$. In case of exo-operations introduced below, we replace ins by $e-ins$ and del by $e-del$.

Exo Insertion and Deletion

We say that insertion or deletion is applied at the left (right) end of the string if the place of insertion or deletion is restricted accordingly. In particular, for the rules without context, the operations can be written as

$$x \Longrightarrow_{ins_l(\alpha)} \alpha x,\ x \Longrightarrow_{ins_r(\alpha)} x\alpha,\ \alpha x \Longrightarrow_{del_l(\alpha)} x,\ x\alpha \Longrightarrow_{del_r(\alpha)} x.$$

In transitional P systems, we write the target indication in parentheses after α.

For example, if rule $ins_l(x, out)$ is applied to string w then the string xw will be sent to the outer region.

Let us illustrate the difference between IDPs and eIDPs by a simple example. A system $\Pi = (\{a,b\}, \{a,b\}, [\ ]_1, \{babbab\}, \{del(b), ins(a)\})$ generates language $L(\Pi) = \{w \in \{a,b\}^* \mid |w|_b \leq 4\}$. However, a system $\Pi' = (\{a,b\}, \{a,b\}, [\ ]_1, \{babbab\}, \{del_l(b), ins_r(a)\})$ generates language $L(\Pi') = (\{\lambda\} \cup \{b\})\{abbab\}\{a\}^*$.

We assume that every string represented in the region has arbitrary many copies. Hence, all rules applicable to a string are applied in the same step, the original string remains, and the multiplicities are not tracked.

The power of insertion and deletion Insertion systems, without using the deletion operation, were first considered in [190], however the idea of the context adjoining was exploited long before by [223]. Both insertion and deletion operations were first considered together in [206] and related formal language investigations can be found in several places; we mention only [204], [231] and [251]. In the last few years, the study of these operations has received a new motivation from molecular computing, see, for example, [161], [205], [256], [275], because, from the biological point of view, insertion-deletion operations correspond to mismatched annealing of DNA sequences. As it was shown in [225], the context dependency may be replaced by insertion and deletion of strings of sufficient length, in a context-free manner. If the length is not sufficient (less than two) then such systems are decidable and a characterization of them was shown in [278]. Similar investigations were continued in [232] and [210] on insertion-deletion systems with one-sided contexts, i.e. where the context dependency is present only from the left (right) side of all insertion and deletion rules. These articles also give some combinations of rule parameters that lead to systems which are not computationally complete. However, if these systems are combined with the distributed computing framework of P systems [252], then their computational power may strictly increase, see [211], [209]. It is not difficult to see that dropping the requirement of the uniqueness of the instructions with the same label, the power of partially blind register machines does not change, see, e.g., [170].

Insertion and deletion are fundamental string operations, introduced in the formal language theory mainly with linguistic motivation. The biological motivation is that these operations correspond to mismatched annealing of DNA sequences. They are also present in the evolution processes as point mutations as well as in RNA editing, see [141], [271] and [256]. It was shown that such additional control permits to increase the computational power up to computationally completeness results for all four cases, improving the results from [211] and [209]. However, the framework of P systems cannot increase the computational power to such extent in all cases, namely it was shown that if context-free insertion and deletion rules using at most two symbols are considered, i.e. systems of size $(2, 0, 0; 2, 0, 0)$, then the corresponding P systems are still not computationally complete [208], [278]. It is thus interesting to consider conditions that would allow such systems to reach computational completeness. We should also mention the research in [203]: one considers insertion at one end of a string coupled with deletion at the other end. Even when the pairing is not prescribed, the universality is still achieved. However, the size of the inserted and deleted substrings is not bounded. It has been shown in [214] that any Turing machine can be simulated by a CPM0.

Splicing Head splicing systems (H systems) [194] were one of the first theoretical models of biomolecular computing (DNA-computing). From the formal language theory point of view, the computational power of the obtained model is rather limited, only regular languages can be generated. Various additional control mechanisms were proposed in order to "overcome" this obstacle and to generate all recursively enumerable languages. An overview of such mechanisms can be found in [256]. The number of rules is a measure of the size of the system. This approach is coherent with investigations related to small universal Turing machines, e.g. [267]. Test tube systems based on splicing, introduced in [156], communicate through redistribution of the contents of the test tubes via filters that are simply sets of letters (in a similar way to the *separate* operation of Lipton-Adleman [218], [1]). After a series of results, the number of tubes sufficient to achieve this result was established to be 3 [264]. The computational power of splicing test tube systems with two tubes is still an open question. A simple possibility to turn splicing-based systems into computationally complete devices are time-varying distributed H systems (TVDH systems). Such systems work like H systems, but on each computational step the set of active rules is changed in a cycle. These sets are called *components*. It was shown [256] that 7 components are enough for the computational completeness; further this number was reduced to 1 [226], [227]. This last result shows a fine difference between the definitions of a computational step in H systems. If one iterates the splicing operation while keeping all generated strings, then such systems are regular. If only the result of each splicing step is kept, then the resulting systems are computationally complete. An overview of results on TVDH systems may be found in [228]. Like for small universal Turing machines, we are interested in such universal systems that have a small number of rules. A first result was obtained in [265] where a universal splicing P system with 8 rules was shown. We also consider a class of H systems which can be viewed as a counterpart of the matrix grammars in the regulated rewriting area. These systems are called double splicing extended H systems [256].

1.5 Computing with P Systems

When a configuration is reached where no rule can be applied, the computation stops, and the multiplicity of objects sent into environment during the computation is said to be computed by the system along that computation. We denote by $Ps(\Pi)$ the set of vectors generated in this way (by means of all computations) by a system Π. If we take into account the sequence of symbols as they are sent out into the environment (when two or more objects leave the system at the same moment, then all permutations of these objects are considered), then we obtain the string language generated by Π which is denoted by $L(\Pi)$. When considering Π as an accepting system for set of vectors, we put the input multiset into the skin membrane and accept by halting computations.

We denote the resulting families generated by such P systems by $XO_{n_1,n_2,n_3}P_{n_4,n_5,n_6}F$ where (1) X is either L for languages or Ps for sets of vectors of non-negative integers; we add the subscript a when considering accepting systems; (2) F is the list of features used in the model (e.g., we consider $(ncoo, tar, mcre, \delta)$, $(ncoo, tar, mcre_r, \delta)$, $(active_1, a, b, c, d, e)$, and $(active_2, a, c)$); (3) the numbers n_4, n_5, n_6 represent the bounds on the starting number of membranes, the maximal number of membranes in any computation, and the number of membrane labels, $*$ representing the absence of a bound (if all three numbers are $*$, then we simply omit them); (4) the numbers n_1, n_2, n_3 have the same meaning, but for the objects inside the system; the middle parameter, n_2 or n_5, can be replaced by n'_2/n_2 or n'_5/n_5 where the primed numbers indicate the bounds on the number of objects or membranes ever present in the system during halting computations only, thus refining this complexity measure.

1.5.1 Decisional framework

The P systems of interest here are those for which all computations give the same result. This is because it is enough to consider one computation to obtain all information about the result.

Definition 1.8 *A P system with output is* confluent *if (a) all computations halt; and (b) at the end of all computations of the system, region i_0 contains the same multiset of objects from T.*

In this case one can say that the multiset mentioned in (b) is the result given by a P system, so this property is already sufficient for convenient usage of P systems for computation.

However, one can still speak about a stronger property: a P system is *strongly confluent* if not only the result of all computation is the same, but the entire halting configuration is the same. A yet stronger property is determinism: a P system is called *deterministic* if it only has one computation.

Let us recall that a decision problem X is a pair (I_X, θ_X) where I_X is a language over a finite alphabet (whose elements are called *instances*) and θ_X is a total Boolean function over I_X. To solve this kind of problems, we consider P systems as *recognizer languages* devices.

Definition 1.9 *A* recognizer P system *is a P system with external output such that: (a) the working alphabet contains two distinguished elements* yes *and* no*; (b) all computations halt;*

and (c) if $\mathcal{C}$ is a computation of the system, then either object yes *or object* no *(but not both) must have been released into the environment, and only in the last step of the computation.*

In recognizer P systems, we say that a computation $\mathcal{C}$ is an *accepting computation* (resp. *rejecting computation*) if the object *yes* (resp. *no*) appears in the environment associated with the corresponding halting configuration of $\mathcal{C}$.

Now, we deal with recognizer membrane systems *with an input membrane* solving decision problems in a *uniform* way in the following sense: all instances of a decision problem with the same *size* (according to a previously fixed polynomial time computable criterion) are processed by the same system, on which an appropriate input, representing the specific instance, is supplied.

Definition 1.10 *A* P system with an input membrane *is a tuple (Π, Σ, i_Π), where: (a) Π is a P system with external output; with the working alphabet O; (b) $\Sigma \subset O$ is an (input) subalphabet, and the initial multisets are over $O \setminus \Sigma$; (c) i_Π is the label of a distinguished (input) membrane.*

Definition 1.11 *Let $X = (I_X, \theta_X)$ be a decision problem. We say that X is solvable in polynomial time by a family $\mathbf{\Pi} = (\Pi(n))_{n\in\mathbb{N}}$ of recognizer membrane systems* with an input membrane*, and we denote it by $X \in \mathbf{PMC}_{\mathcal{R}}$, if*

- *The family $\mathbf{\Pi}$ is polynomially uniform by TM: some deterministic TM constructs in polynomial time the system $\Pi(n)$ from $n \in \mathbb{N}$.*
- *There exists a pair (cod, s) of polynomial-time computable functions whose domain is I_X, such that for each $u \in I_X$, $s(u)$ is a natural number and $cod(u)$ is an input multiset of the system $\Pi(s(u))$, verifying the following:*
- *The family $\mathbf{\Pi}$ is polynomially bounded with regard (X, cod, s); that is, there exists a polynomial function $p(n)$ such that for each $u \in I_X$ every computation of the system $\Pi(s(u))$ with input $cod(u)$ halts in at most $p(|u|)$ steps.*
- *The family $\mathbf{\Pi}$ is sound with regard to (X, cod, s); that is, for each instance of the problem $u \in I_X$ such that there exists an accepting computation of $\Pi(s(u))$ with input $cod(u)$, we have $\theta_X(u) = 1$.*
- *The family $\mathbf{\Pi}$ is complete with regard to (X, cod, s); that is, for each instance of the problem $u \in I_X$ such that $\theta_X(u) = 1$, every computation of $\Pi(s(u))$ with input $cod(u)$ is an accepting one.*

We say that the family $\mathbf{\Pi}$ is a *uniform solution* to the problem X. The complexity classes $\mathbf{PMC}_{\mathcal{R}}$ are closed under complement and closed under polynomial time reduction, in the classical sense.

We denote by $\mathcal{AM}^0(\alpha, \beta)$, where $\alpha \in \{-d, +d\}$ and $\beta \in \{-ne, +ne\}$, the class of all recognizer P systems with polarizationless active membranes such that: $-d$ forbids rules (d_0), and $-ne$ forbids rules (f_0).

A conjecture known in the membrane computing area under the name of the P–conjecture (proposed by Gh. Păun in 2005) is that $\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}^0(+d,-ne)}$.

Theorem 1.1 *The following statements hold:*

(1) $\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}^0(-d,\beta)} = \mathbf{PMC}^*_{\mathcal{AM}^0(-d,\beta)}$, *for each* $\beta \in \{-ne, +ne\}$.

(2) $\mathbf{NP} \subseteq \mathbf{PMC}^*_{\mathcal{AM}^0(+d,+ne)}$.

1.5.2 Minimal parallelism

In [185], the *minimal parallelism* has been formalized as follows (we assume each set R_j is associated to a membrane):

$$\begin{aligned} App(\Pi, C, min) = \ \{ \ & R' \in App(\Pi, C, asyn) \mid \text{ there is no} \\ & R'' \in App(\Pi, C, asyn) \text{ such that} \\ & (R'' - R') \cap R_j \neq \emptyset \text{ for some } j \text{ with } R' \cap R_j = \emptyset, \ 1 \leq j \leq h \ \}. \end{aligned}$$

We are not going to define all notations used here. In our context, this definition means that minimally parallel application of rules to a configuration consists of all applicable multisets R' that cannot be extended by a rule corresponding to a membrane for which no rule appears in R'.

There exist different interpretations of minimal parallelism. For instance, the original definition of maximal parallelism introduced in [152] is formalized in [279] and called there *base vector minimal parallelism*:

$$\begin{aligned} App(\Pi, C, min_G) = \ \{ \ & R''' \in App(\Pi, C, asyn) \mid \text{ there is} \\ & R' \in App(\Pi, C, asyn), \text{ such that } R' \subseteq R''', \\ & |R' \cap R_j| \leq 1 \text{ for all } j, \ 1 \leq j \leq h, \text{ and} \\ & \text{there is no } R'' \in App(\Pi, C, asyn) \text{ such that} \\ & (R'' - R') \cap R_j \neq \emptyset \text{ for some } j \text{ with } R' \cap R_j = \emptyset, \ 1 \leq j \leq h \ \}. \end{aligned}$$

Without discussing all technicalities, we point out that base vector minimally parallel application of rules consists of all extensions of multisets R', which represent maximally parallel choice of sets R_j used sequentially. Hence, the latter mode is identical to the following:

$$\begin{aligned} \{ \ & R''' \in App(\Pi, C, asyn) \mid \exists R' \in App(\Pi, C, seq_{set}), R' \subseteq R''' \\ & \text{and } \nexists R'' \in App(\Pi, C, seq_{set}) : R' \subseteq R'' \ \}, \text{ where } |R' \cap R_j| \leq 1 \text{ for all } j, \ 1 \leq j \leq h. \} \end{aligned}$$

In this way, one can first restrict applicable multisets to those having at most one rule corresponding to a membrane, then take maximally parallel ones from them (i.e., those whose extensions do not belong to the same restriction), and finally take their unrestricted extensions.

Luckily, if a P system with active membranes does not use rules of type (b) or its modifications, then it works equally well for both definitions of minimal parallelism. Indeed, in one step a membrane reacts only with objects in the associated region. This means that selection of rules for each membrane is done independently, so different membranes do not compete for objects and the system behaves identically in both modes. Hence, we can simply follow

the basic idea introduced already in [152]: for every membrane, at least one rule - if possible - has to be used.

The following remarks describe applicability, maximal applicability and applying rules, respectively.

- The rules of type (a) may be applied in parallel. At one step, a membrane can be the subject of *only one* rule of types $(a'_{0s}), (a''_s)$ and $(b), (c), (d), (e)$ with their modifications.
- In one step, one object of a membrane can be used by only one rule (non-deterministically chosen), but **for every membrane at least one object** that can evolve by one rule of any form, must evolve (no rules associated to a membrane are applied only if none are applicable for the objects that do not evolve).
- If at the same time a membrane is divided by a rule of type (e) and there are objects in this membrane which evolve by means of rules of type (a), then we suppose that first the evolution rules of type (a) are used, and then the division is produced. Of course, this process takes only one step.

Particular problems

The detailed overview of membrane P systems with replication can be found in [252]. In Section 6.1, cooperative rules are used in this framework, addressing the problem of inflections in words of Romanian language, assuming that the inflection group is known, [219]. The question of determining the inflection group is addressed in [154], [153].

Synchronization The synchronization problem is well known for cellular automata, where it was intensively studied under the name of the *firing squad synchronization problem* (FSSP): a line of soldiers have to fire at the same time after the appropriate order of a general which stands at one end of the line, see, e.g., references in [97], [98]. The first solution of the problem was found by Goto, see [192]. It works on any cellular automaton on the line with n cells in the minimal time, $2n-2$ steps, and requiring several thousands of states. A bit later, Minsky found his famous solution which works in $3n$, see [233] with a much smaller number of states, 13 states. Then, a race to find a cellular automaton with the smallest number of states which synchronizes in $3n$ started. See the above papers for references and for the best results and for generalizations to the planar case, see [276] for results and references. Many studies have been dedicated to general synchronization principles occurring during the cell cycle; although some results are still controversial, it is widely recognized that these aspects might lead to an understanding of general biological principles used to study the normal cell cycle, see [273]. The synchronization problem as defined above was studied in [145] for two classes of P systems: transitional P systems and P systems with priorities and polarizations. In the first case, a non-deterministic solution to FSSP was presented and for the second case a deterministic solution was found. These solutions need time $3h$ and $4n + 2h$ respectively, where n is the number of membranes and h is the depth of the membrane tree. Improving these solutions is an interesting task, addressed in Section 6.3.

1.6 Conclusions to Chapter 1

This chapter introduces basic prerequisites of formal language theory, grammars, automata and machines. It presents the variants of matrix grammars, finite automata, counter automata, register machines and circular Post machines needed for the proofs given in this thesis. The models of networks of evolutionary processors (NEPs), hybrid NEPs and obligatory hybrid NEPs are introduced.

A number of models of membrane systems with symbol-objects is defined, such as multiset rewriting, transitional P systems (together with membrane creation and dissolution), P systems with symport and antiport (and tissue P systems), P systems with active membranes (and their rule variants), energy models of P systems, and polymorphic P systems. The concepts of maximal parallelism, result of computation, reversibility and determinism are introduced.

In the case of P systems with string-objects, the rewriting model with string replication is defined, as well as P systems with active membranes. Yet another model presented is that of P systems with insertion and deletion (and its variant with exo-operations). Splicing operation is mentioned.

Next, general definitions for computing with P systems are given, including the decisional framework and minimal parallelism.

The existing results in the area are recalled, formulating and motivating a number of problems that have been unsolved, giving the basis for the underlying research of this thesis. Some of these topics are decreasing the known number of nodes in the universal networks of hybrid networks of evolutionary processors, investigating the power of deterministic controlled multiset rewriting, effect of reversibility, determinism and self-stabilization, exact power of non-cooperative transitional P systems, with or without membrane creation and dissolution, decreasing the total number of *rules* in P systems known to be universal, the exact power of one-membrane P systems with symport only, the power of active membranes without polarizations, the power of P systems working in exponential space, a number of computational complexity questions for P systems with active membranes, including the case of minimal parallelism, characterization of energy models, etc.

2. MULTISET REWRITING. PROPERTIES

This chapter is devoted to the heart of membrane computing, i.e., multiset rewriting. It can be cooperative or non-cooperative, it happens sequentially, asynchronously, or maximally parallel, it can be controlled by promoters, inhibitors or priorities. A number of general properties is studied. Languages are obtained as sequences of symbols sent out. Section 2.5 considers multiset processing enhanced by the distributivity of the system, and even by the dynamic modification of the underlying tree structure.

In Section 2.1 we present a study of the family of languages generated by the transitional membrane systems without cooperation and without additional ingredients. The fundamental nature of these basic systems makes it possible to also define the corresponding family of languages in terms of derivation trees of context-free grammars. We also compare this family to the well-known language families and discuss its properties. We give an example of a language considerably more "difficult" than the currently established lower bounds.

All the processing is done by multisets, and one considers the order of sending the objects in the environment as their order in the output word. Informally, the family of languages we are interested in is the family generated by systems with parallel applications of non-cooperative rules that rewrite symbol objects and/or send them between the regions. This model has been introduced already in 2000, [258]. The nature of $LOP_*(ncoo, tar)$ is quite fundamental, and in the same time it is not characterized in terms of well-studied families. This is why we refer to it here as *the membrane systems language family.*

Non-cooperative P systems with either promoters or inhibitors (of weight not restricted to one) are known to be computationally complete. In Section 2.2 we show that the power of the deterministic subclass of such systems is subregular in the asynchronous mode and in the maximally parallel mode. This provides the second case known in membrane computing where determinism is a criterion of universality versus decidability, after the model of catalytic P systems.

In Section 2.3 we study reversibility and determinism aspects and the strong versions of these properties of sequential multiset processing systems and of maximally parallel systems, from the computability point of view. In the sequential case, syntactic criteria are established for both strong determinism and strong reversibility. In the parallel case, a criterion is established for strong determinism, whereas strong reversibility is shown to be decidable.

In the sequential case, without control all four classes – deterministic, strongly deterministic, reversible, strongly reversible – are not universal, whereas in the parallel case deterministic systems are universal. When allowing inhibitors, the first and the third class become universal in both models, whereas with priorities all of them are universal. In the maximally parallel case, strongly deterministic systems with both promoters and inhibitors

are universal. We also present a few more specific results and conjectures.

In Section 2.4 we discuss a notion of self-stabilization, inspired from biology and engineering. Multiple variants of formalization of this notion are considered, and we discuss how such properties affect the computational power of multiset rewriting systems.

In Section 2.5, it is essential that not only multiset rewriting is distributed over a tree structure, but also that such a structure is dynamic. We present quite surprising results that non-cooperative rewriting reaches computational completeness when equipped with membrane creation and membrane dissolution. Intuitively, this is possible due to the cooperation between an object and a fact of existence of a membrane.

2.1 The P Systems Language Family

We start by presenting a study of the family of languages generated by the transitional membrane systems without cooperation and without additional ingredients. The fundamental nature of these basic systems makes it possible to also define the corresponding family of languages in terms of derivation trees of context-free grammars. We also compare this family to the well-known language families and discuss its properties. We also give an example of a language which is considerably more "difficult" than the currently established lower bounds. The word "difficult" informally refers to two kinds of non-context-freeness needed to describe the language. The internal one can be captured by permutations, while the external one can be captured by an intersection of linear languages.

We recall that the configurations of membrane systems (with symbol objects) consist of multisets over a finite alphabet, distributed across a tree structure. Therefore, even such a relatively simple structure as a word (i.e., a sequence of symbols) is not explicitly present in the system. To speak of languages as sets of words, one first needs to represent them in membrane systems, and there are a few ways to do it.

Represent words by string objects. Rather many papers take this approach, see Chapter 7 of [257], but only few consider parallel operations on words. Moreover, a tuple of sets or multisets of words is already a quite complicated structure. The third drawback is that it is very difficult to define an elegant way of interactions between strings. Polarizations and splicing are examples of that; however, these are difficult to use in applications. In this subsection we focus on symbol objects.

Represent a word by a single symbol object, or by a few objects of the form (letter,position) as in, e.g., [131], [132]. This only works for words of bounded length, i.e., one can speak about at most finite languages.

Represent positions of the letters in a word by nested membranes. The corresponding letters can be then encoded by objects in the associated regions, membrane types or membrane labels. Working with such a representation, even implementing a rule $a \to bc$ requires sophisticated types of rules, like creating a membrane around existing membrane, as defined in [144].

Consider letters as digits and then view words as numbers, or use some other encoding of words into numbers or multisets. Clearly, the concept of words ceases to be direct with

such encoding. Moreover, implementing basic word operations in this way requires a lot of number processing, not to speak of parallel word operations.

Work in accepting mode, see, e.g., [155]. It is necessary to remark that in this section we are mainly speaking of non-cooperative P systems, and working in accepting mode without cooperation yields rather trivial subregular results. More exactly, such systems would accept either the empty language or only the empty word, or all the words over some subalphabet.

Consider traces of objects across membranes. This is an unusual approach in the sense that the result is not obtained from the final configuration, but rather from the behavior of a specific object during the computation. This makes it necessary to introduce an observer, complicating the model.

Do all the processing by multisets, and regard the order of sending the objects in the environment as their order in the output word. In case of ejecting multiple symbols in the same step, the output word is formed from any of their permutations. One can say that this approach also needs an implicit observer, but at least this observer only inspects the environment and it is "the simplest possible". This section is devoted to this concept.

Informally, the family of languages we are interested in is the family generated by systems with parallel applications of non-cooperative rules that rewrite symbol objects and/or send them between the regions. This model has been introduced already in 2000, [258]. Surprisingly, this language family did not yet receive enough attention of researchers. Almost all known characterizations and even bounds for generative power of different variants of membrane systems with various ingredients and different descriptional complexity bounds are expressed in terms of REG, MAT, $ET0L$ and RE, their length sets and Parikh sets (and much less often in terms of FIN or other subregular families, or CF or CS, or those accepted by log-tape bounded Turing machines, [158], [196]). The membrane systems language family presents interest since we show it lies between regular and context-sensitive families, being incomparable with well-studied intermediate ones. As we show in this section, the nature of $LOP_*(ncoo, tar)$ is quite fundamental, and in the same time it is not characterized in terms of well-studied families. This is why we refer to it here as *the membrane systems language family.*

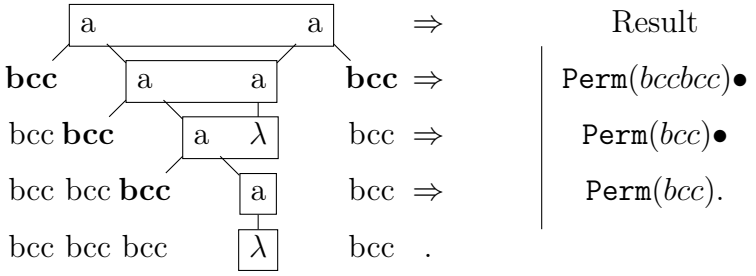


Figure 2.1: A computation and a word generated by a P system from Example 2.1

Example 2.1 *To illustrate generating languages, consider the following P system:*

$$\Pi = (\{a, b, c\}, [\ \ ]_1, a^2, \{a \to \lambda, a \to a\ b_{out} c^2_{out}\}, 0).$$

Each of the two symbols a has a non-deterministic choice whether to be erased or to reproduce itself while sending a copy of b and two copies of c into the environment. Therefore, the contents of region 1 can remain a^2 for an arbitrary number $m \geq 0$ of steps, and after that at least one copy of a is erased. The other copy of a can reproduce itself for another $n \geq 0$ steps before being erased. Each of the first m steps, two copies of b and four copies of c are sent out, while in each of the next n steps, only one copy of b and two copies of c are ejected. Therefore, $L(\Pi) = (\texttt{Perm}(bccbcc))^*(\texttt{Perm}(bcc))^*$. A computation is illustrated in Figure 2.1; the lines are only used to hint how the rules are applied.

We first show that for every membrane system without cooperation, there is a system from the same class with one membrane, generating the same language.

Lemma 2.1 *$LOP_*(ncoo, tar) = LOP_1(ncoo, out)$.*

Proof. Consider an arbitrary transitional membrane system Π (without cooperation or additional ingredients). The known technique of flattening the structure (this is "folklore" in membrane computing; see, e.g., [266], [185]) consists of transforming Π in the following way. Object a in region associated to membrane i is transformed into object (a, i) in the region associated to the single membrane. The alphabet, initial configuration and rules are transformed accordingly. Clearly, the configurations of both systems are bisimilar, and the environment output is the same. □

The following exposition uses the definitions of *time-yield* of context-free grammars and its relation to P systems. These definitions and results can be found in the Appendix A1.

2.1.1 Comparison with known families

Theorem 2.1 *[258] $LOP(ncoo, tar) \supseteq REG$.*

Proof. Consider an arbitrary regular language. Then there exists a complete finite automaton $M = (Q, \Sigma, q_0, F, \delta)$ accepting it. We construct a context-free grammar $G = (Q, \Sigma, q_0, P)$, where $P = \delta \cup \{q \to \lambda \mid q \in F\}$. The order of symbols accepted by M corresponds to the order of symbols generated by G, and the derivation can only finish when the final state is reached. Hence, $L_t(G) = L(M)$, and the theorem statement follows. □

Theorem 2.2 *$LOP(ncoo, tar) \subseteq CS$.*

Proof. Consider a context-free grammar $G = (N, T, S, P)$ in the First normal form. We construct a grammar $G' = (N \cup \{\#_1, L, R, F, \#_2\}, T, S', P')$:

$$\begin{aligned} P' &= \{S' \to \#_1 LS\#_2, L\#_2 \to R\#_2, \#_1 R \to \#_1 L, \#_1 R \to F, F\#_2 \to \lambda\} \\ &\cup \{LA \to uL \mid (A \to u) \in P\} \cup \{La \to aL, Fa \to aF \mid a \in T\} \\ &\cup \{aR \to Ra \mid a \in N \cup T\}. \end{aligned}$$

The symbols $\#_1, \#_2$ mark the edges, the role of symbol L is to apply productions P to all non-terminals, left-to-right, while skipping the terminals. While reaching the end marker, symbol L changes into R and returns to the beginning marker, where it either changes back

to L to iterate the process, or to F to check whether the derivation is finished. Hence, $L(G') = L_t(G)$. Note that the length of sentential forms in any derivation (of some word with n symbols in G') is at most $n+3$, because the only shortening productions are the ones removing $\#_1, \#_2$ and F, and each should be applied just once. Therefore, $L_t(G) \in CS$. □

We now proceed to showing that the membrane systems language family does not contain the family of linear languages. To show this, we first define the notions of unbounded yield and unbounded time of a non-terminal.

Definition 2.1 *Consider a grammar $G = (N, T, S, P)$. We say that $A \in N$ has an unbounded yield if $L_t(G_A)$ is an infinite language, i.e., there is no upper bound on the length of words generated from A.*

It is easy to see that $L_t(G_A)$ is infinite if and only if $L(G_A)$ is infinite; decidability of this property is well-known from the theory of CF grammars.

Definition 2.2 *Consider a grammar $G = (N, T, S, P)$. We say that $A \in N$ has unbounded time if the set of all derivation trees (for terminated derivations) in G_A is infinite, i.e., there is no upper bound on the number of parallel steps of terminated derivations in G_A.*

It is easy to see that A has unbounded time if $L(G_A) \neq \emptyset$ and $A \Rightarrow^+ A$, so decidability of this property is well-known from the CF grammar theory.

Lemma 2.2 *Let $G = (N, T, P, S)$ be a context-free grammar in the Third normal form. If for every rule $(A \to BC) \in P$, symbol B does not have unbounded time, than $L_t(G) \in REG$.*

The proof can be found in [43].

Lemma 2.3 $L = \{a^n b^n \mid n \geq 1\} \notin LOP(ncoo, tar)$.

Once again, the proof can be found in [43].

Corollary 2.1 $LIN \not\subseteq LOP(ncoo, tar)$.

Lemma 2.4 *The family $LOP(ncoo, tar)$ is closed under permutations.*

Proof. Transform a given grammar $G = (N, T, S, P)$. Replace the terminal symbols a are by non-terminals a_N throughout the description of G, and then add rules $a_N \to a_N$, $a_N \to a$, $a \in T$ to P. Like in the first example, the order of generating terminals is arbitrary. □

Corollary 2.2 $\texttt{Perm}(REG) \subseteq LOP(ncoo, tar)$.

Proof. Follows from regularity theorem 2.1 and permutation closure lemma 2.4. □

The results of comparison of the membrane system family with the well-known language families can be summarized as follows:

Theorem 2.3 *$LOP(ncoo, tar)$ strictly contains REG and $\texttt{Perm}(REG)$, is strictly contained in CS, and is incomparable with LIN and CF.*

Proof. All inclusions and incomparabilities have been shown in or directly follow from Theorem 2.1, Corollary 2.2, Theorem 2.2, Corollary 2.1 and Corollary A1.1 with Theorem A1.1. The strictness of the first inclusions follows from the fact that REG and $\texttt{Perm}(REG)$ are incomparable, while the strictness of the latter inclusion holds since $LOP(ncoo, tar)$ only contains semilinear languages. □

The lower bound can be strengthened as follows:

Theorem 2.4 $LOP(ncoo, tar) \supseteq REG \bullet \texttt{Perm}(REG)$.

Proof. Indeed, consider the construction from the regularity theorem. Instead of erasing the symbol corresponding to the final state, rewrite it into the axiom of the grammar generating the second regular language, to which the permutation technique is applied. □

Example 2.2 $LOP(ncoo, tar) \ni L_2 = \bigcup_{m,n\geq 1}(abc)^m\texttt{Perm}((def)^n)$.

2.1.2 Closure properties

It has been shown above that the family of languages generated by basic membrane systems is closed under permutations. We now recall a few other closure properties (see [43] for the proofs).

Lemma 2.5 *The family $LOP(ncoo, tar)$ is closed under erasing/renaming morphisms.*

Corollary 2.3 $\{a^nb^nc^n \mid n \geq 1\} \notin LOP(ncoo, tar)$.

Corollary 2.4 *$LOP(ncoo, tar)$ is not closed under intersection with regular languages.*

Theorem 2.5 *$LOP(ncoo, tar)$ is closed under union and not closed under intersection or complement.*

Lemma 2.6 $L = \bigcup_{m,n\geq 1} \texttt{Perm}((ab)^m)c^n \notin LOP(ncoo, tar)$.

Corollary 2.5 *$LOP(ncoo, tar)$ is not closed under concatenation or taking the mirror image.*

2.1.3 A difficult language

In this section we give an example of a language in $LOP(ncoo, tar)$ which is considerably more "difficult" than languages in $REG \bullet \texttt{Perm}(REG)$, in the sense informally explained below. Besides permutations of symbols sent out in the same time, it exhibits another kind of non-context-freeness. This second source of "difficulty" alone can, however, be captured as an intersection of two linear languages.

$$\begin{aligned} \Pi_D &= (\{D, D', a, b, c, a', b', c'\}, [\ ]_1, D^2, R), \\ R &= \{D \to (abc)_{out}D'D',\ D \to (abc)_{out},\ D' \to (a'b'c')_{out}DD,\ D' \to (a'b'c')_{out}\}. \end{aligned}$$

The contents of region 1 is a population of objects D, initially 2, which are primed if the step is odd. Assume that there are k objects inside the system. At every step, every symbol

D is either erased or doubled (and primed or de-primed), so the next step the number of objects inside the system will be any even number between 0 and $2k$. In addition to that, the output during that step is $\texttt{Perm}((abc)^k)$, primed if the step is odd. Hence, the generated language can be described as

$$L(\Pi_D) = \{ \texttt{Perm}((abc)^{2k_0})\texttt{Perm}((a'b'c')^{2k_1}) \cdots \texttt{Perm}((abc)^{2k_{2t}})\texttt{Perm}((a'b'c')^{2k_{2t+1}}) \mid k_0 = 1,\ 0 \leq k_i \leq 2k_{i-1},\ 1 \leq i \leq 2t+1,\ t \geq 0\}.$$

For an idea of how complex a language generated by some non-cooperative membrane system be, imagine that the skin may contain populations of multiple symbols that can (like D in the example above) be erased or multiplied (with different periods), and also rewritten into each other. The same, of course, happens in usual context-free grammars, but since the terminal symbols are collected from the derivation tree level by level instead of left to right, the effect is quite different.

Another upper bound has been recently proved.

2.1.4 Parsability

We first recall a few existing results.

Lemma 2.7 *Random access machines can be simulated by Turing machines with polynomial slowdown.*

This result will lead to a much simpler proof of the parsability result.

From Section A1 and the relationship between the result of non-cooperative P systems and the time-yield of context-free grammars, we have the following: one membrane is enough (Lemma 2.1), and the following two results hold:

Lemma 2.8 *[43] Any non-cooperative P system can be transformed into an equivalent one such that objects having no rules to evolve them are never produced.*

This condition means that the evolution of any object inside the system eventually leads to some number (possibly zero) of objects in the environment.

Lemma 2.9 *([43]) Any non-cooperative P system can be transformed into an equivalent one such that the initial contents is $w_1 = \{S\}$, and*

- *S does not appear in the right-hand side of any rule, and*
- *R_1 has no erasing rules, except possibly $\{S\} \to \emptyset$.*

This result means that no object can be erased, except the axiom which may only be erased immediately.

We now proceed to the parsability result.

Theorem 2.6 $LOP_*(ncoo, tar) \subseteq \mathbf{P}$.

Proof. The proof consists of three parts. First, a few known results are used to simplify the statement of the theorem. Second, a finite-state automaton (with transitions labeled by multisets of terminals) of polynomial size is constructed. Third, acceptance problem is reduced to a search problem in a graph of a polynomial size.

Thanks to Lemma 2.7, the rest of the proof can be explained at the level of random access machines. Due to Lemma 2.1, we assume that an arbitrary membrane system language L is given by a one-membrane system $\Pi = ([\ \ ]_1, O, w_1, R_1)$. It is known from Lemma 2.8 that the condition specified in it does not restrict the generality. Hence, from now we assume that every object A inside the system corresponds to at least one rule that rewrites A. Without restricting generality, we also assume the normal form specified in Lemma 2.9. In this case, it is clear that if $w \in T^n$, then during any computation of Π generating w, the number of objects inside the system can never exceed $\max(n, 1)$.

We now build a finite automaton $A = (Q, \Sigma, q_0, \delta, F)$ such that any word $w' \in T^{\leq n}$ is accepted by A if and only if $w' \in L(\Pi)$. Accepting by an automaton with transitions labeled by multisets is understood as follows: a transition labeled by a multiset of weight k can be followed if the multiset composed of the next k input symbols equals the transition label; in this case these input symbols are read.

We define Q as the set of multisets of at most $\max(n, 1)$ objects, Σ as the set of multisets of at most n objects, q_0 is the singleton multiset $\{S\}$, and $F = \{\emptyset\}$. It only remains to define the transition mapping δ of A. We say that $q' \in \delta(q, s)$ if $[\ q\ ]_1 \Rightarrow [\ q'\ ]_1 s$. It is known (see, e.g., [7]) that computing all transitions from a configuration with k objects takes polynomial time with respect to k; here, $k \leq \max(n, 1)$ (the degree of such a polynomial does not exceed $|R_1|$), and, moreover, the number of configurations reachable in one step is also polynomial.

Note that $|Q|$ is polynomial with respect to n (and the degree of such a polynomial does not exceed $|O|+1$).[1] Hence, A can be built from n and Π in polynomial time, and, moreover, the size of the description of A is also polynomial. Of course, it suffices to only examine the reachable states of A.

Running each transition $q' \in \delta(q, s)$ of A on w can be actually done in time $O(|s|)$; however, there are two problems. Firstly, A is non-deterministic, and secondly, A may have transitions labeled by an empty multiset, and removing empty multiset transitions or non-determinism might need too much time or space, or even increase its size too much. Instead, we reduce parsing by A to a graph reachability problem.

Consider a graph $\Gamma = (V, U)$, where $V = \{0, \cdots, n\} \times Q$ and U consists of such transitions $((i, q), (j, q'))$ that $i \leq j$ and $q' \in \delta(q, s)$, where s equals the multiset consisting of $w[i+1], \cdots, w[j]$. Finally, $w \in L = L_t(G)$ if and only if $w \in L(A)$, and $w \in L(A)$ if and only if there is a path from $(0, q_0)$ to (n, e) in Γ. Note: alternatively, search in A incrementally by prefixes of w. □

It is known from [43] that membrane systems language family is included in the family of context-sensitive languages, see also Lemma 2.9. In [43] one also claims that membrane systems language family is **semilinear**. No formal proof is given, but there is an almost immediate observation that such language is letter-equivalent to that generated by the context-

[1] Indeed, multisets of size $\leq n$ over O bijectively correspond to multisets of size exactly n over $O \cup \{\lambda\}$. Let $|O| = m$. Moreover, multisets of size n over $O \cup \{\lambda\}$ correspond to n-combinations of $m+1$ possible elements with repetition. For $n > 0$, their number is $|Q| = \binom{n+m}{n} \leq n^{m+1}$.

free language with the same rules (languages are letter-equivalent if for every word in one of them there is a word in the other one with the same multiplicities of all symbols; indeed, the difference is only in the order of output). Semilinearity thus follows from Parikh theorem. By Theorem 2.6, we improve the upper bound:

Corollary 2.6 $LOP_*(ncoo, tar) \subseteq CS \cap SLIN \cap \mathbf{P}$.

An Example Consider a word $w = babbaa$ and a P system

$$\begin{aligned} \Pi &= ([\ ]_1, \{S', S, a, b\}, \{S'\}, R), \text{ where} \\ R &= \{p : \{S'\} \to \{S\},\ q : \{S\} \to \{S^2\},\ r : \{S\} \to \{a, b\}_{out}\}. \end{aligned}$$

Only objects S, S' are productive inside the system, and only objects a, b may be sent outside. Since $|w| = 6$, we only need to examine (28) multisets over S, S' of size up to 6 elements. However, out of them only $\{S'\}$, $\{S\}$, $\emptyset$, $\{S^2\}$, $\{S^4\}$, $\{S^6\}$ are reachable. The automaton would look as in Figure 2.2 (to simplify the picture, we wrote i instead of $\{a^i, b^i\}$ as labels):

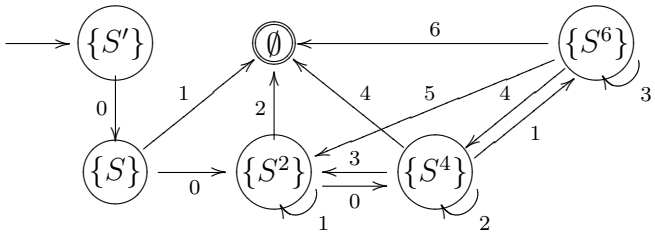


Figure 2.2: Bounded transition graph as a finite automaton

We now check the word w:

- states after reading λ: $\{S'\}$, $\{S\}$, $\{S^2\}$, $\{S^4\}$;
- states after reading ba: $\emptyset$, $\{S^2\}$, $\{S^4\}$, $\{S^6\}$;
- states after reading $babbaa$: $\emptyset$, $\{S^4\}$. The input is accepted.

This result means that membrane systems languages can be parsed in polynomial time. However, the degree of such polynomials in the algorithm deciding membership problem presented in [41] depends on the number of rules and the size of the alphabet in the underlying P system.

Conclusions In this section we have reconsidered the family of languages generated by transitional P systems without cooperation and without additional control. It was shown that one membrane is enough, and a characterization of this family was given via derivation trees of context-free grammars. Next, three normal forms were given for the corresponding grammars. It was than shown that the membrane systems language family lies between regular and context-sensitive families of languages, and it is incomparable with linear and with context-free languages. Then, the lower bound was strengthened to $REG \bullet \mathtt{Perm}(REG)$.

An example of a considerably more "difficult" language was given than the lower bound mentioned above. We also mention another upper bound result, i.e., membrane systems languages can be parsed in polynomial time.

The membrane systems language family was shown to be closed under union, permutations, erasing/renaming morphisms. It is not closed under intersection, intersection with regular languages, complement, concatenation or taking the mirror image.

The following are examples of questions that are still not answered.

- Clearly, $LOP(ncoo, tar) \not\supseteq MAT$. What about $LOP(ncoo, tar) \subseteq MAT$?
- Is $LOP(ncoo, tar)$ closed under arbitrary morphisms? Conjecture: no. The difficulty is to handle $h(a) = bc$ if many symbols a can be produced in the same step.
- Look for sharper lower and upper bounds.

2.2 Deterministic Non-Cooperative Systems

Non-cooperative P systems with either promoters or inhibitors (of weight not restricted to one) are known to be computationally complete. In this section we show that the power of the deterministic subclass of such systems is computationally complete in the sequential mode, but only subregular in the asynchronous mode and in the maximally parallel mode.

The most famous membrane computing model where determinism is a criterion of universality versus decidability is the model of catalytic P systems, see [171] and [197].

It is also known that non-cooperative rewriting P systems with either promoters or inhibitors are computationally complete, [133], [134], [135]. Moreover, the proof satisfies some additional properties:

- Either promoters of weight 2 or inhibitors of weight 2 are enough.
- The system is non-deterministic, but it restores the previous configuration if the guess is wrong, which leads to correct simulations with probability 1.

The purpose of this section is to show that computational completeness cannot be achieved by deterministic systems when working in the asynchronous or in the maximally parallel mode. The definitions of one-region P systems with advanced control are mainly needed for this section only; they are given in the Appendix A2.

2.2.1 Lower bounds

Example 2.3 *Let H be an arbitrary finite set of numbers and $K = \max(H) + 1$; then we construct the following deterministic accepting P system with promoters and inhibitors:*

$$\begin{aligned}
\Pi &= (O, \{a\}, s_0 f_0 \cdots f_K, R', R), \\
O &= \{a\} \cup \{s_i, f_i \mid 0 \leq i \leq K\}, \\
R' &= \{s_i \to s_{i+1} \mid 0 \leq i \leq K-1\} \cup \{f_i \to f_i \mid 0 \leq i \leq K\}, \\
R &= \{s_i \to s_{i+1}|_{a^{i+1}}, \mid 0 \leq i \leq K-1\} \\
&\cup \left\{f_i \to f_i|_{s_i, \neg a^{i+1}}, \mid 0 \leq i < K,\ i \notin H\right\} \cup \{f_K \to f_K|_{s_K}\}.
\end{aligned}$$

The system step by step, by the application of the rule $s_i \to s_{i+1}|_{a^{i+1}}$, $0 \leq i < K$, checks if (at least) $i+1$ copies of the symbol a are present. If the computation stops after i steps, i.e., if the input has consisted of exactly i copies of a, then this input is accepted if and only if $i \in H$, as exactly in this case the system does not start an infinite loop with using $f_i \to f_i|_{s_i,\neg a^{i+1}}$. If the input has contained more than $\max(H)$ copies of a, then the system arrives in the state s_K and will loop forever with $f_K \to f_K|_{s_K}$. Therefore, exactly H is accepted. To accept the complement of H instead, we simply change $i \notin H$ to $i \in H$ and as well omit the rule $f_K \to f_K|_{s_K}$. It is easy to see that for the maximally parallel mode, we can replace each rule $f_i \to f_i|_{s_i,\neg a^{i+1}}$ by the corresponding rule $f_i \to f_i|_{s_i}$; in this case, this rule may be applied with still some a being present while the system passes through the state s_i, but it will not get into an infinite loop in that case.

In sum, we have shown that

$$\begin{aligned} N_aDOP_1^{asyn}\left(ncoo,(pro_{1,*},inh_{1,*})_1\right) &\supseteq FIN \cup coNFIN \text{ and} \\ N_aDOP_1^{maxpar}\left(ncoo,pro_{1,*}\right) &\supseteq FIN \cup coNFIN. \end{aligned}$$

Example 2.4 *For P systems working in the maximally parallel way we can even construct a system with inhibitors only:*

$$\begin{aligned} \Pi &= (O,\{a\},ts_K,R',R), \\ O &= \{a,t\} \cup \{s_i \mid 0 \leq i \leq K\}, \\ R' &= \{s_i \to ts_{i-1}, s_i \to s_i \mid 1 \leq i \leq K\} \cup \{t \to \lambda, s_0 \to s_0\}, \\ R &= \{s_i \to ts_{i-1}|_{\neg a^i} \mid 1 \leq i \leq K\} \\ &\cup \{t \to \lambda\} \cup \{s_i \to s_i|_{\neg t} \mid 0 \leq i \leq K,\ i \notin H\}. \end{aligned}$$

This construction does not carry over to the case of the asynchronous mode, as the rule $t \to \lambda$ is applied in parallel to the rules $s_i \to ts_{i-1}|_{\neg a^i}$ until the input a^i is reached. In this case, the system cannot change the state s_i anymore, and then it starts to loop if and only if $i \notin H$. To accept the complement of H instead, change $i \in H$ to $i \notin H$, i.e., in sum, we have proved that

$$N_aDOP_1^{maxpar}\left(ncoo,inh_{1,*}\right) \supseteq FIN \cup coNFIN.$$

As we shall show later, all the inclusions stated in Example 2.3 and Example 2.4 are equalities.

2.2.2 Upper bounds and characterizations

In this section we mainly investigate deterministic accepting P systems with context conditions and priorities on the rules (*deterministic P systems* for short) using only non-cooperative rules and working in the sequential, the asynchronous, and the maximally parallel mode.

Remark 2.1 *We first notice that maximal parallelism in systems with non-cooperative rules means the total parallelism for all symbols to which at least one rule is applicable, and*

determinism guarantees that "at least one" is "exactly one" for all reachable configurations and objects. Determinism in the sequential mode requires that at most one symbol has an associated applicable rule for all reachable configurations. Surprisingly enough, in the case of the asynchronous mode we face an even worse situation than in the case of maximal parallelism – if more than one copy of a specific symbol is present in the configuration, then no rule can be applicable to such a symbol in order not to violate the condition of determinism.

We now define the *bounding* operation over multisets, with a parameter $k \in \mathbb{N}$ as follows:

$$\text{for } u \in O^*,\ b_k(u) = v \text{ with } |v|_a = \min(|u|_a, k) \text{ for all } a \in O.$$

The mapping b_k "crops" the multisets by removing copies of every object a present in more than k copies until exactly k remain. For two multisets u, u', $b_k(u) = b_k(u')$ if for every $a \in O$, either $|u|_a = |u'|_a < k$, or $|u|_a \geq k$ and $|u'|_a \geq k$. Mapping b_k induces an equivalence relation, mapping O^* into $(k+1)^{|O|}$ equivalence classes. Each equivalence class corresponds to specifying, for each $a \in O^*$, whether no copy, one copy, or $\cdots$ $k-1$ copies, or "k copies or more" are present. We denote the range of b_k by $\{0, \cdots, k\}^O$.

Lemma 2.10 *Context conditions are equivalent to predicates defined on boundings.*

Proof. We start by representing context conditions by predicates on boundings. Consider a rule with a simple context condition (r, p, Q), and let the current configuration be C. Then, it suffices to take $k \geq \max(|p|, \max\{|q| \mid q \in Q\})$ and $C' = b_k(C)$. The applicability condition for (r, p, Q) may be expressed as $p \subseteq C' \wedge \left(\bigwedge_{q \in Q} q \not\subseteq C'\right)$. Indeed, $x \subseteq C \longleftrightarrow x \subseteq C'$ for every multiset x with $|x| \leq k$, because for every $a \in O$, $|x|_a \leq |C|_a \longleftrightarrow |x|_a \leq \min(|C|_a, k)$ holds if $|x|_a \leq k$. Finally, we notice that context conditions which are not simple can be represented by a disjunction of the corresponding predicates.

Conversely, we show that any predicate $E \subseteq \{0, \cdots, k\}^O$ for the bounding mapping b_k for rule r can be represented by some context conditions. For each multiset $c \in E$, we construct a simple context condition to the effect of "contains c, but, for each a contained in c for less than k times, not more than $|c|_a$ symbols a":

$$\left\{ \left(r, c, \left\{ a^{|c|_a+1} \middle| \, |c|_a < k \right\}\right) \mid c \in E \right\}.$$

Joining multiple simple context conditions over the same rule into one rule with context conditions concludes the proof. □

The following theorem is valid even when the rules are not restricted to non-cooperative ones, and when determinism is not required, in either derivation mode (also see [172]).

Theorem 2.7 *Priorities are subsumed by conditional contexts.*

Proof. A rule is prohibited from being applicable due to a priority relation if and only if at least one of the rules with higher priority might be applied. Let r be a rule of a P system $(O, \Sigma, w, R', R, >)$, and let $r_1 > r, \cdots, r_n > r$. Hence, the rule r is not blocked by the rules $r_1, \cdots, r_n$ if and only if the left-hand sides of the rules $r_1, \cdots, r_n$, i.e., $lhs(r_1), \cdots, lhs(r_n)$, are not present in the current configuration or the context conditions given in these rules are not fulfilled. According to Lemma 2.10, these context conditions can be formulated as

predicates on the bounding b_k where k is the maximum of weights of all left-hand sides, promoters, and inhibitors in the rules with higher priority $r_1, \cdots, r_n$. Together with the context conditions from r itself, we finally get context conditions for a new rule r' simulating r, but also incorporating the conditions of the priority relation. Performing this transformation for all rules r concludes the proof. □

Remark 2.2 *From [172] we already know that in the case of rules without context conditions, the context conditions in the new rules are only sets of atomic inhibitors, which also follows from the construction given above. A careful investigation of the construction given in the proof of Theorem 2.7 reveals the fact that the maximal weights for the promoters and inhibitors to be used in the new system are bounded by the number k in the bounding b_k.*

Remark 2.3 *As in a P system $(O, \Sigma, w, R', R, >)$ the set of rules R' can easily be deduced from the set of rules with context conditions R, in the following we omit R' in the description of the P system. Moreover, for systems having only rules with a simple context condition, we omit d in the description of the families of sets of numbers and simply write $N_\delta OP_1^\alpha(\beta, pro_{k,l}, inh_{k',l'}, pri)$. Moreover, each control mechanism not used can be omitted, e.g., if no priorities and only promoters are used, we only write $N_\delta OP_1^\alpha(\beta, pro_{k,l})$.*

2.2.3 Sequential systems

The proof of the following theorem gives some intuition why, for deterministic non-cooperative systems, there are severe differences between the sequential mode and the asynchronous or the maximally parallel mode (throughout the rest of the section we do not deal with sequential systems anymore).

Theorem 2.8 $N_aDOP_1^{sequ}(ncoo, pro_{1,1}, inh_{1,1}) = NRE$.

Proof. Let $M = (m, Q, I, q_0, q_f)$ be an arbitrary deterministic register machine. We simulate M by a deterministic P system $\Pi = (O, \{a_1\}, l_0, R)$:

$$\begin{aligned} O &= \{a_j \mid 1 \le j \le m\} \cup \{q, q_1, q_2 \mid q \in Q\}, \\ R &= \{q \to a_j q' \mid (q : [RjP], q') \in I\} \\ &\cup \{q \to q_1|_{a_j},\ a_j \to a_j'|_{q_1, \neg a_j'},\ q_1 \to q_2|_{a_j'},\ a_j' \to \lambda|_{q_2},\ q_2 \to q'|_{\neg a_j'}, \\ &\quad q \to q''|_{\neg a_j} \mid (q : \langle RjZM\rangle, q', q'') \in I\}. \end{aligned}$$

Π is deterministic, non-cooperative, and it accepts the same set as M. □

In the construction of the deterministic P system in the proof above, the rule $a_j \to a_j'|_{q_1, \neg a_j'}$ used in the sequential mode can be applied exactly once, priming exactly one symbol a_j to be deleted afterwards. Intuitively, in the asynchronous or the maximally parallel mode, it is impossible to choose only one symbol out of an unbounded number of copies to be deleted. The bounding operation defined above will help to formalize this intuition.

2.2.4 Asynchronous and maximally parallel systems

Fix an arbitrary deterministic controlled non-cooperative P system. Take k as the maximum of size of all multisets in all context conditions. Then, the bounding does not influence applicability of rules, and $b_k(u)$ is halting if and only if u is halting. We proceed by showing that bounding induces equivalence classes preserved by any computation.

Lemma 2.11 *Assume $u \Rightarrow x$ and $v \Rightarrow y$. Then $b_k(u) = b_k(v)$ implies $b_k(x) = b_k(y)$.*

Proof. Equality $b_k(u) = b_k(v)$ means that for every symbol $a \in O$, if $|u|_a \neq |v_a|$ then $|u|_a \geq k$ and $|v|_a \geq k$, and we have a few cases to be considered. If no rule is applicable to a, then the inequality of symbols a will be indistinguishable after bounding also in the next step (both with at least k copies of a). Otherwise, exactly one rule r is applicable to a (by determinism, and bounding does not affect applicability), then the difference of the multiplicities of the symbol a may only lead to differences of the multiplicities of symbols b for all $b \in rhs(r)$. However, either all copies of a are erased by the rule $a \to \lambda$ or else at least one copy of a symbol b will be generated from each copy of a by this rule alone, so $|x|_b \geq |u|_a \geq k$ and $|y|_b \geq |v|_a \geq k$; hence, all differences of multiplicities of an object b in u and v will be indistinguishable after bounding in this case, too. □

Corollary 2.7 *If $b_k(u) = b_k(v)$, then u is accepted if and only if v is accepted.*

Proof. Let w be the fixed part of the initial configuration. Then we consider computations from uw and from vw. Clearly, $b_k(uw) = b_k(vw)$. Equality of boundings is preserved by one computation step, and hence, by any number of computation steps.

Assume the contrary of the claim: one of the computations halts after s steps, while the other one does not, i.e., let $uw \Rightarrow^s u'$ and $vw \Rightarrow^s v'$. By the previous paragraph, $b_k(u') = b_k(v')$. Since bounding does not affect applicability of rules, either both u' and v' are halting, or none of them. The contradiction proves the claim. □

We should like to notice that the arguments in the proofs of Lemma 2.11 and Corollary 2.7 are given for the maximal parallel mode; following the observation stated at the end of Remark 2.1, these two results can also be argued for the asynchronous mode.

Theorem 2.9 *For deterministic P systems working in the asynchronous or in the maximally parallel mode, we have the following characterization:*

$$\begin{aligned} NFIN \cup coNFIN &= N_aDOP_1^{asyn}(ncoo, pro_{1,*}, inh_{1,*}) \\ = N_aDOP_1^{maxpar}(ncoo, pro_{1,*}) &= N_aDOP_1^{maxpar}(ncoo, inh_{1,*}) \\ = N_aDOP_1^{asyn}\left(ncoo, \left(pro_{*,*}, inh_{*,*}\right)_*, pri\right) &= N_aDOP_1^{maxpar}\left(ncoo, \left(pro_{*,*}, inh_{*,*}\right)_*, pri\right). \end{aligned}$$

Proof. Each equivalence class induced by bounding is completely accepted or completely rejected. If no infinite equivalence class is accepted, then the accepted set is finite (containing numbers not exceeding $(k-1)\cdot|O|$). If at least one infinite equivalence class is accepted, then the rejected set is finite (containing numbers not exceeding $(k-1)\cdot|O|$). This proves the "at most $NFIN \cup coNFIN$" part.

In Examples 2.3 and 2.4 we have already shown that

$$\begin{aligned} N_aDOP_1^{\alpha}(ncoo, pro_{1,*}, inh_{1,*}) &\supseteq FIN \cup coNFIN,\ \alpha \in \{asyn, maxpar\}, \\ N_aDOP_1^{maxpar}(ncoo, \gamma_{1,*}) &\supseteq FIN \cup coNFIN,\ \gamma \in \{pro, inh\}. \end{aligned}$$

This observation concludes the proof. □

There are several questions remaining open, for instance, whether only inhibitors in the rules or only priorities in the rules are sufficient to yield $FIN \cup coNFIN$ with the asynchronous mode, too.

Conclusions We have shown that, like in case of catalytic P systems, for non-cooperative P systems with promoters and/or inhibitors (with or without priorities), determinism is a criterion drawing a borderline between universality and decidability. In fact, for non-cooperative P systems working in the maximally parallel or the asynchronous mode, we have computational completeness in the unrestricted case, and only all finite number sets and their complements in the deterministic case.

2.3 Determinism and Reversibility

We study reversibility and determinism aspects and the strong versions of these properties of sequential multiset processing systems and of maximally parallel systems, from the computability point of view. In the sequential case, syntactic criteria are established for both strong determinism and strong reversibility. In the parallel case, a criterion is established for strong determinism, whereas strong reversibility is shown to be decidable.

In the sequential case, without control all four classes – deterministic, strongly deterministic, reversible, strongly reversible – are not universal, whereas in the parallel case deterministic systems are universal. When allowing inhibitors, the first and the third class become universal in both models, whereas with priorities all of them are universal. In the maximally parallel case, strongly deterministic systems with both promoters and inhibitors are universal. We also present a few more specific results and conjectures.

If a fixed enumeration of the elements of the alphabet is assumed, then multisets are isomorphic to vectors. In that sense, sequential multiset processing corresponds to vector addition systems (see, e.g., [170]). Alternatively, adding and removing symbols can be viewed as incrementing and decrementing counters, i.e., vector addition systems may be viewed as a variant of stateless counter machines (as multitape non-writing Turing machines), where for every instruction it is specified for each counter which integer is to be added to it, yet not restricted to -1, 0 or 1; such a variant is also equivalent to multiset processing systems (in this case, testing for zero corresponds to using inhibitors).

The aim of this section is to consider such properties of multiset rewriting systems as reversibility and determinism as well as their strong versions.

Reversibility is an important property of computational systems. It has been well studied for circuits of logical elements ([165]), circuits of memory elements ([234]), cellular automata ([235]), Turing machines ([143], [237]), register machines ([236]). Reversibility as a syntactical property is closely related to the microscopic physical reversibility, and thus it assumes

better miniaturization possibilities for potential implementation. Moreover, reversibility essentially is backward determinism.

Sequential reversible P systems have been considered in [216], in the energy-based model, simulating Fredkin gates and thus reversible circuits. The maximally parallel case of reversible multiset rewriting systems has been considered in [106], [105]; such systems are universal with priorities or inhibitors; it follows that reversibility is undecidable in these cases. In [106], [105] also strong reversibility is considered, extending the requirement for reversibility in such a way that it does not depend on the initial configuration, and a characterization of strongly reversible systems without control is given. In [195] one shows that even for parallel multiset rewriting systems strong reversibility is decidable. Moreover, in [106], [105] strong determinism is defined, and a syntactic criterion for it is given showing that the power of strongly deterministic systems is weaker than that of deterministic systems. Naturally, similar questions were asked for *sequential* multiset rewriting systems, [64].

Most results on sequential systems are first obtained in [64]; we here compare the results obtained for sequential systems obtained in [64] with the results obtained for maximally parallel systems in [105]. While revisiting the results from [105] we also answer a few questions left open there. The structure of the presentation is as follows: we first give the relevant definitions and then illustrate reversibility and strong reversibility for the case of sequential multiset rewriting systems by some examples. We continue with elaborating the results for the computational power of these sequential multiset rewriting systems as well as with the results for determinism and strong determinism. We proceed with investigating the same questions for the maximally parallel case. We conclude with a summary of the results shown in this section and with the discussion of some open problems.

2.3.1 Sequential multiset rewriting

Most of the corresponding results have first been shown in [105] and [106], where the presentation has been done in terms of one-membrane symport/antiport P systems, with the environment containing an unbounded supply of all objects[2].

Due to the arguments above, we can lead the exposition of the definitions and results in this section in terms of multiset rewriting systems only, keeping in mind that the results are equivalent both to those of the one-membrane symport/antiport model as well as even to those of multi-region systems.

We now present a few examples to illustrate the definitions.

Example 2.5 *Any multiset rewriting system* $\Pi_1 = (O, w_0, \{u \to v\})$ *with only one rule is strongly deterministic: there is no choice. Moreover,* Π_1 *is strongly reversible: there is at*

[2] It is well-known that rules in symport/antiport systems can be represented as cooperative rewriting rules on objects of the form (object,region). It is also known that, in case the environment contains an unbounded supply of all objects, any symport/antiport rule u/v (meaning u goes out and v comes in) is equivalent to a multiset rewriting rule $u \to v$ (clearly, symport-in rules are not allowed). Therefore, such one-membrane systems with complete environment are a normal form for symport/antiport P systems, and they are behaviorally equivalent (bisimulation) to multiset rewriting systems. Moreover, these systems and the related transitions preserve such properties we consider later as determinism, reversibility and self-stabilization.

most one incoming transition (to obtain the preimage, remove v and add u). If both w_0 and v contain u, then no halting configuration is reachable, i.e., $\emptyset \in NR_sMR(coo)$. Otherwise, a singleton is generated; if w_0 does not contain u, the computation immediately halts, hence, the singleton w_0 is generated, i.e., $\{n\} \in NR_sMR(coo)$ for $n \in \mathbb{N}$.

Example 2.6 *Consider the system $\Pi_2 = (\{a, b, c\}, a, \{a \to ab,\ a \to c\})$. It generates the set of positive integers since the reachable halting configurations are cb^*, and it is reversible: there is at most one incoming transition into any reachable configuration (for the preimage, replace c with a or ab with a), but not strongly reversible (e.g., $aab \Rightarrow cab$ and $ca \Rightarrow cab$). Hence, $\mathbb{N}_+ \in NRMR(coo)$.*

Example 2.7 *Any multiset rewriting system containing some erasing rule $u \to \lambda$ is not reversible, unless other rules are never applicable.*

Example 2.8 *Any system containing two rules of the form $x_1 \to y$ and $x_2 \to y$ that may apply at least one of them in some computation is not reversible.*

2.3.2 Reversible sequential rewriting

Reversible multiset rewriting systems with either inhibitors or priorities are universal.

Theorem 2.10 $NRMR(coo, Pri)_T = NRMR(coo, inh)_T = NRE$.

Proof. We reduce the statement of the theorem to the claim that such multiset rewriting systems can simulate the work of any reversible register machine $M = (m, Q, I, q_0, q_f)$. Consider the multiset rewriting system

$$\begin{aligned}
\Pi &= (O, \{r_1\}, q_0, R), \text{ where } O = \{r_j \mid 1 \le j \le m\} \cup Q,\\
R &= \{q \to q' r_j, q \to q'' r_j \mid (q : [RjP], q', q'') \in I\}\\
&\cup \{q r_j \to q', q r_j \to q'' \mid (q : [RjM], q', q'') \in I\} \cup R_t,\\
R_t &= \{q \to q''|_{\neg r_j}, q r_j \to q' r_j \mid (q : \langle RjZ\rangle, q', q'') \in I\}.\\
&\text{Inhibitors can be replaced by priorities, redefining } R_t \text{ as follows:}\\
R_t &= \{q r_j \to q' r_j > q \to q'' \mid (q : \langle RjZ\rangle, q', q'') \in I\}.
\end{aligned}$$

The configurations of Π with one symbol from Q are in a bijection to configurations of M, so reversibility of Π follows from the correctness of the simulation, the reversibility of M and preservation of the number of symbols from Q by the transitions of Π. □

The universality in Theorem 2.10 leads to the following undecidability.

Corollary 2.8 *It is undecidable whether a system from the class of multiset rewriting systems with either inhibitors or priorities is reversible.*

Proof. Recall that the halting problem for register machines is undecidable. Add instructions $q_f \to F_1$, $q_f \to F_2$, $F_1 \to F$, $F_2 \to F$ to the construction presented above (F_1, F_2, F are new objects); the system now is reversible if and only if some configuration containing F is reachable, i.e., if the underlying register machine does not halt, which is undecidable. □

2.3.3 Strong reversibility

Deciding strong reversibility is much easier: it is necessary and sufficient that no two different rules are applicable to any configuration. Without restricting generality we only consider systems without useless rules, i.e., no rule is inhibited by some of its reactants.

Consider the case that inhibitors may be used, and let $r_1 : x_1 \to y_1 v$ and $r_2 : x_2 \to y_2 v$ be two rules of the system (possibly controlled by inhibitors), where v is the largest common submultiset of the right-hand sides of r_1 and r_2. Then both rules can be reversely applied to some configuration C if and only if it contains $y_1 v$ and $y_2 v$ and none of these two rules is inhibited. Now writing C as $y_1 y_2 vw$, we get the two possible transitions $x_1 y_2 w \Rightarrow y_1 v y_2 w$ and $x_2 y_1 w \Rightarrow y_2 v y_1 w$. The inhibitors should in particular forbid one of these transitions in the case $w = \lambda$ (from that, the general case for w follows immediately). Therefore, either r_1 should be inhibited by some object from y_2, or r_2 should be inhibited by some object from y_1. Clearly, this criterion is also sufficient, since for any two rules the competition for the backwards applicability is solved by inhibitors. We have just proved the following statement:

Theorem 2.11 *A sequential multiset rewriting system with inhibitors is strongly reversible if for any two rules r_1 and r_2, either r_1 is inhibited by $rhs(r_2) \setminus rhs(r_1)$ or r_2 is inhibited by $rhs(r_1) \setminus rhs(r_2)$.*

The case of priorities is slightly more involved. Let $r_1 : x_1 \to y_1 v$ and $r_2 : x_2 \to y_2 v$ be two rules of the system, where v is the largest common submultiset of the right sides of r_1 and r_2. Then both rules can be reversely applied to some configuration C if and only if it contains $y_1 y_2 v$ and none of these two rules is made inapplicable by rules of higher priority. Writing C as $y_1 y_2 vw$, we get the possible transitions $x_1 y_2 w \Rightarrow y_1 v y_2 w$ and $x_2 y_1 w \Rightarrow y_2 v y_1 w$. The priorities should in particular forbid one of these transitions in the case $w = \lambda$ (from that, the general case for w follows immediately). Therefore, either r_1 should be made inapplicable by some rule $r > r_1$ with left-hand side contained in $x_1 y_2$, or r_2 should be made inapplicable by some rule $r > r_2$ with left-hand side contained in $x_1 y_2$. Clearly, this criterion is also sufficient since the competition for reverse applicability between any two rules is eliminated. We have just proved the following result:

Theorem 2.12 *A sequential multiset rewriting system with priorities is strongly reversible if for any two rules r_1, r_2 there exists a rule r such that either $r > r_1$ and $lhs(r) \subseteq lhs(r_1)(rhs(r_2) \setminus rhs(r_1))$ or $r > r_2$ and $lhs(r) \subseteq lhs(r_2)(rhs(r_1) \setminus rhs(r_2))$.*

It is not difficult to see that the criterion of strong reversibility for systems that may use both inhibitors and priorities is obtained as a disjunction of the requirements from Theorem 2.11 and Theorem 2.12 for any two rules r_1 and r_2.

Corollary 2.9 *A sequential multiset rewriting system with priorities is strongly reversible if for any two rules r_1 and r_2 at least one of the following conditions holds:*

- *r_1 is inhibited by $rhs(r_2) \setminus rhs(r_1)$,*
- *r_2 is inhibited by $rhs(r_1) \setminus rhs(r_2)$,*

- *there exists a rule* $r > r_1$ *such that* $lhs(r) \subseteq lhs(r_1)(rhs(r_2) \setminus rhs(r_1))$,
- *there exists a rule* $r > r_2$ *such that* $lhs(r) \subseteq lhs(r_2)(rhs(r_1) \setminus rhs(r_2))$.

Corollary 2.10 *A multiset rewriting system without control is strongly reversible if and only if it only has exactly one rule.*

$$NR_sMR(coo)_T = \{\emptyset\} \cup \{\{n\} \mid n \in \mathbb{N}\}.$$

It is known that (see, e.g, [170]) the generative power of sequential multiset rewriting systems equals $PsMAT$, even without requiring additional properties like reversibility.

Corollary 2.11 *Reversible multiset rewriting systems without priorities and without inhibitors are not universal.*

It is an open problem to specify the exact generative power of this class.

We now return to controlled systems. Surprisingly, adding priorities turns degenerate computational systems into universal ones.

Theorem 2.13 $NR_sMR(coo, Pri)_T = NRE$.

Proof. Take the construction from Theorem 2.10, the case of priorities, and add rules R_d (q and q' may also be the same): $R_d = \{qq' \to qq' \mid q, q' \in Q\}$. Finally, extend the priority relation to a total order relation in such a way that every rule from R_d has priority over every rule not from R_d. The system remains universal, because throughout the simulation of register machines, exactly one object from Q is present, and rules from R_d are not applicable; for the same reason, rules with different states in the left side are never applicable simultaneously, so the extension of the priority relation has no effect on the simulation.

Obtained system is also strongly reversible. Indeed, all rules preserve the number of objects from Q. All configurations without objects from Q are immediately halting and have no preimage. Any configuration with multiple objects from Q evolves into itself (by applying a rule with the highest priority possible, trivially rewriting two state symbols), and its preimage is the configuration itself, with the incoming transition corresponding to the applicable rule with the highest priority. In case of one object from Q, the property follows from the strong reversibility of the simulated register machine. □

2.3.4 Deterministic sequential rewriting

The concept of determinism common to multiset rewriting systems as considered in the area of membrane computing essentially means that such a system, starting from any fixed configuration, has a unique computation. As it will be obvious later, this property is often not decidable. Of course, this section only deals with accepting systems.

First, it is well known that deterministic multiset rewriting systems with either priorities or inhibitors are universal, by simulating of any (deterministic accepting) register machine M. In fact, in this case the construction of Theorem 2.10 is both deterministic and reversible.

Corollary 2.12 $N_aD_sMR(coo, Pri)_T = NRE$.

Proof. For a strongly deterministic system, it suffices to take the construction with priorities in Theorem 2.10 (simulating any deterministic accepting register machine) and to extend the priorities to a total order. The total order relation can be defined by the relation specified by taking the union of R_t and the relation induced by an arbitrary fixed (e.g., lexicographical) order on the objects from Q as they appear in the left sides of rules. □

In general, if a certain class of non-deterministic systems is universal even in a deterministic way, then the determinism is undecidable for that class. This applies to multiset rewriting systems, similarly to Corollary 2.8.

Corollary 2.13 *It is undecidable whether a system from the class of multiset rewriting systems with either inhibitors or priorities is deterministic.*

Proof. For an arbitrary register machine M, there is a deterministic multiset rewriting system Π with either inhibitors or priorities simulating M. Without restricting generality we assume that an object q_f appears in the configuration of Π if and only if it halts. Add instructions $q_f \to F_1$ and $q_f \to F_2$ to the set of rules (F_1, F_2 are new objects); the system is now deterministic if and only if some configuration with q_f is reachable, i.e., when M does not halt, which is undecidable. □

2.3.5 Strong determinism

On the contrary, the strong determinism we now consider means that a system has no choice of transitions from any configuration. We now claim that this is a syntactic property.

Theorem 2.14 *A multiset rewriting system is strongly deterministic if and only if any two rules are either mutually excluded by an inhibitor (of one rule appearing on the left-hand side of another rule), or are in a priority relation.*

Proof. Any multiset rewriting system with exactly one rule is strongly deterministic.

The forward implication of the theorem holds true because mutually excluding reactant/inhibitor conditions eliminate all competing rules except one, and so does the priority relation. In the result, for any configuration at most one rule is applicable.

Conversely, assume that rules p and p' of the system are not in a priority, and are not mutually excluded by the reactant/inhibitor conditions. Let x and x' be the multisets of objects consumed by the rules p and p', respectively. Then, to the configuration $C = xx'$, it is possible to apply either rule, contradicting the condition of strong determinism. □

Corollary 2.14 *A multiset rewriting system without inhibitors and without priority is not strongly deterministic, unless it only has exactly one rule.*

We now characterize the power of strongly deterministic multiset rewriting systems without additional control: any multiset rewriting system without inhibitors or priorities accepts either the set of all non-negative integers, or the set of all numbers bounded by some number.

Theorem 2.15 $N_a D_s MR(coo) = \{\emptyset, \mathbb{N}\} \cup \{\{k \mid 0 \leq k \leq n\} \mid n \in \mathbb{N}\}$.

Proof. Any strongly deterministic system is of the form $(O, T, w_0, \{u \to v\})$. If u is not contained in v, then the system accepts all numbers. Otherwise, if v contains u, the computation starting from an initial configuration C is not accepting if and only if u is contained in both C and v. Hence, the system accepts all numbers k for which there exists an $x \in \Sigma^k$ such that w_0x does not contain the multiset u. The converse of the latter condition is monotone with respect to k, i.e., it is either never satisfied, or always satisfied, or there exists a number $n \in \mathbb{N}$ such that it holds if and only if $k > n$.

- System $(\{a\}, \{a\}, a, \{a \to a\})$ accepts the empty set $\emptyset$, because the computation from any configuration aw with $w \in \{a\}^*$ is an infinite loop;
- system $(\{a\}, \{a\}, \lambda, \{a \to \lambda\})$ accepts $\mathbb{N}$, i.e., any natural number, because it halts after erasing everything in n steps when starting with a^n, for any $n \in \mathbb{N}$; and
- for any $n \in \mathbb{N}$ there is a system $(\{a\}, \{a\}, \lambda, \{a^{n+1} \to a^{n+1}\})$ accepting $\{k \mid 0 \leq k \leq n\}$, because the system immediately halts in the initial configuration if and only if the input does not exceed n, and enters an infinite loop otherwise.

These examples show the converse implication of the theorem. □

Theorem 2.15 shows that the computational power of strongly deterministic seqential multiset rewriting systems without additional control is, in a certain sense, degenerate (it is sublinear). We now strengthen Theorem 2.15 from strongly deterministic systems to all deterministic ones.

Corollary 2.15 $N_aDMR(coo) = \{\emptyset, \mathbb{N}\} \cup \{\{k \mid 0 \leq k \leq n\} \mid n \in \mathbb{N}\}$.

Proof. Like in Theorem 2.15, the system accepts all numbers k for which there exists $x \in \Sigma^k$ such that the computation starting from w_0x halts. It suffices to recall that if $C \subseteq C'$ and a computation starting with C' halts, then a computation starting with C also halts. Indeed, since the computation starting from C' is deterministic, if it applies rule r in step s, then the computation starting from C cannot apply any other rule in the same step. The corresponding configurations of the two computations preserve any $\subseteq$ relation of the initial configurations. □

2.3.6 Maximally parallel multiset rewriting

We give a few examples to illustrate the definitions when the rules are applied in parallel.

Example 2.9 *Consider the P system* $\Pi_0 = (\{a, b\}, a, \{a \to ab\})$. *It is strongly reversible: there is at most one incoming transition (for a preimage, remove as many copies of* b *as there are copies of* a*, in case it is possible and there is at least one copy of* a*), but no halting configuration is reachable. Therefore,* $\emptyset \in NR_sOP_1(ncoo)$.

Example 2.10 *Consider the P system* $\Pi_2 = (\{a, b, c\}, a, \{a \to ab,\ a \to c\})$. *As the corresponding sequential multiset rewriting system, it generates the set of positive integers since the reachable halting configurations are* cb^**, and it is reversible because there is at most one incoming transition into any reachable configuration (for the preimage, replace* c *with* a *or* ab *with* b*), but not strongly reversible (e.g.,* $aa \Rightarrow cc$ *and* $ac \Rightarrow cc$*). Hence,* $\mathbb{N}_+ \in NROP(ncoo)$.

Example 2.11 *Consider the P system* $\Pi_2 = (\{a, b\}, aa, \{aa \to ab,\ ab \to bbb\})$. *It is reversible (the configuration* aa *has in-degree* 0*, while the configurations* ab *and* bbb *have in-degree* 1*, and no other configuration is reachable), but not strongly reversible (e.g.,* $aab \Rightarrow abbb$ *and* $aabb \Rightarrow abbb$*).*

Example 2.12 *Any P system containing a rule* $x \to \lambda$ *is not reversible. Therefore, erasing rules cannot be used in reversible P systems.*

Example 2.13 *Any P system containing two rules* $x_1 \to y$ *and* $x_2 \to y$ *such that at least one of them can be applied in some computation is not reversible.*

2.3.7 Reversible parallel rewriting

The next result is obtained by a construction identical to that in Theorem 2.10. Indeed, notice that every rule uses some symbol associated to Q, and there is exactly one such symbol present in the system all the time, so multiple rules are never applied in the same step.

Theorem 2.16 $NROP_1(coo, Pri)_T = NROP_1(coo, inh)_T = NRE$.

These universality results lead to the following undecidability results (by a construction identical to that in Corollary 2.8):

Corollary 2.16 *It is undecidable whether a system from the class of P systems with either inhibitors or priorities is reversible.*

The construction in Theorem 2.10 uses both cooperation and additional control. It is natural to ask whether both inhibitors and priorities can be avoided. Yet, consider the following situation: let $(p : i?, s, q''), (q : i?, q', s) \in I$. It is not unusual for reversible register machines to have this, since the preimage of a configuration containing a representation of instruction s depends on register i. Nevertheless, P systems with maximal parallelism without additional control can only implement a zero-test by a *try-and-wait-then-check* strategy. In this case, the object containing the information about the register p finds out the result of checking after a possible action of the object related to the register. Therefore, when the instruction represented in the configuration of the system changes to s, it obtains an erroneous preimage representing instruction q. This leads us to the following

Conjecture 2.1 *Reversible P systems without priorities and without inhibitors are not universal.*

2.3.8 Strong reversibility

Unlike reversibility itself, the more restricted property of strong reversibility is decidable, see [195], since checking that at most one incoming transition exists for any configuration is no longer related to reachability. However, the following theorem establishes a very serious limitation on such systems if no additional control is used.

Theorem 2.17 *In strongly reversible P systems without inhibitors and without priorities, every configuration is either halting or induces only infinite computation(s).*

Proof. If the right-hand side of every rule contains the left-hand side of some rule, then the claim obviously holds (only infinite computations if at least one rule is applicable to the initial configuration, otherwise no transition is possible and the computation immediately halts). On the other hand, assume $x \to y$ to be a rule of the system such that y does not contain the left-hand side of any rule (hence, $x \neq y$, too). Then $x \Rightarrow y$ and y is a halting configuration. It is not difficult to see that then two different preimages of yy exist, i.e., $xy \Rightarrow yy$ (the objects in y are idle) and $xx \Rightarrow yy$ (the rule can be applied twice). Therefore, such a system is not strongly reversible, which proves the theorem. □

Thus, the strongly reversible systems without additional control can only generate singletons, and only in a degenerate way, i.e., without doing any transition step:

Corollary 2.17 $NR_sOP_1(coo)_T = \{\{n\} \mid n \in \mathbb{N}\}$.

It turns out that the theorem above does not hold if inhibitors are used: consider th system $\Pi_4 = (\{q, f, a\}, q, \{q \to qaa|_{\neg f}\}, \{q \to f|_{\neg f}\})$. If at least one object f is present or no objects q are present, a configuration of Π_4 is a halting one. Otherwise, all objects q are used by the rules of the system. Therefore, the only possible transitions in the space of all configurations are of the form $q^{m+n}a^{p-2m} \Rightarrow q^m f^n a^p$, $m + n > 0$, $p \geq 2m$ and the system is strongly reversible. Notice that $N(\Pi_4) = \{2k+1 \mid k \geq 0\}$, since, in any halting computation, starting from q we apply the first rule for $k \geq 0$ steps and finally the second rule.

We now consider the controlled case. Once again, priorities change degenerate computational systems into universal ones.

Theorem 2.18 $NR_sOP_1(coo, Pri)_T = NRE$.

Proof. Consider the construction from Theorem 2.13. We recall that every rule uses at least one object from Q and preserves the number of objects from Q. When there are no multiple objects from Q in the system, multiple rules cannot be applied simultaneously, so the behavior is identical for the maximally parallel case, and the simulation is correct. Otherwise, due to the total priority, only the rule with the highest possible priority (rewriting two state symbols into themselves) will be applied, as many times as possible, yielding the same configuration. Hence, the only preimage of a configuration in this case is the configuration itself (and the only incoming transition consists of the maximal application of the rule with the highest priority among the applicable ones), hence, the same system is reversible in the maximally parallel case, too. □

Remark 2.4 *Notice that universality may still be obtained by simulating priorities, if the concept of inhibitors is generalized as follows: typically, inhibitors are defined as single objects. Occasionally, we may consider inhibitors that are multisets, in the sense that the rule is applicable if the inhibiting multiset is not a submultiset of the current configuration. A further possible generalization would be for a rule to have multiple inhibiting multisets, in the sense that the rule is applicable if it is not inhibited by any of these multisets, i.e., if none of them is a submultiset of the current configuration.*

For instance, a rule $a \to b|_{\neg cd, \neg ce}$ would be applicable to an object a if the current configuration does not simultaneously contain c and d, and also does not simultaneously contain

c and e. Indeed, priorities can then be simulated by including as inhibiting multisets the left-hand sides of all rules with higher priorities.

We conjecture that strongly reversible universality is impossible using just one singleton inhibitor in any rule. This remark also holds for the sequential case.

2.3.9 Strongly deterministic parallel rewriting

The concept of determinism common to membrane computing essentially means that such a system, starting from the fixed configuration, has a unique computation. This property often turns out undecidable. Of course, this subsection only deals with accepting systems.

First, we recall that if deterministic accepting register machines are simulated in Theorem 2.16, then the construction is both deterministic and reversible. The following corollary is obtained similarly to Corollary 2.12, i.e., by extending the priority relation to a total order.

Corollary 2.18 $N_aD_sMR(coo, Pri)_T = NRE$.

In general, if a certain class of non-deterministic P systems is universal even in a deterministic way, then the determinism is undecidable for that class. This also applies to the special model of P systems considered in this section (the proof is similar to that of Corollary 2.13).

Corollary 2.19 *It is undecidable if a given P system is deterministic.*

On the contrary, the strong determinism we now consider means that a system has no choice of transitions from *any* configuration. We claim that it is a syntactic property; to formulate the claim, we need the notion of the *domain* of a rule $x \to y$, $x \to y|_a$ or $x \to y|_{\neg a}$ is the set of objects in x (the multiplicities of objects in x are not relevant for the results here). We say that two rules are mutually excluded by promoter/inhibitor conditions if the inhibitor of one is either the promoter of the other rule, or is in the domain of the other rule.

Theorem 2.19 *A P system is strongly deterministic if and only if any two rules with intersecting domains are either mutually excluded by promoter/ inhibitor conditions, or are in a priority relation.*

Proof. Clearly, any P system with only one rule is strongly deterministic, because the degree of parallelism is defined by exhausting the objects from the domain of this rule.

The forward implication of the theorem holds because the rules with non-intersecting domains do not compete for the objects, while mutually excluding promoter/inhibitor conditions eliminate all competing rules except one, and so does the priority relation. As a result, for any configuration the set of objects is partitioned in disjoint domains of applicable rules, and the number of applications of different rules can be computed independently.

We now proceed with the converse implication. Assume that two rules p and p' of the system intersect in the domain, but are not in a priority relation and are not mutually excluded by the promoter/inhibitor conditions. Let x and x' be the multisets of objects to be rewritten by rules the rules p and p', respectively. Then consider the multiset C which is the minimal multiset including x and x', and the configuration C', defined as the minimal multiset including C' and the promoters of p and p', if any.

Starting from C', there are enough objects for applying either p or p'. Since the rules neither are mutually excluded nor are in a priority relation, both rules are applicable. However, both cannot be applied together because the rules intersect in the domain and thus the multiset C is strictly included in xx' (and C' is only different from C if a promoter of p or p' does not belong to C). In that way we get a contradiction to the definition of strong determinism and therefore proves the sufficiency of the condition stated in the theorem. □

Corollary 2.20 *A P system without promoters, inhibitors, and without priority is strongly deterministic if and only if the domains of all rules are disjoint.*

We now show an interesting property of strongly deterministic P systems without additional control. To define it, we use the following notion for deterministic P systems: let $C \Rightarrow^{\rho_1} C_1 \Rightarrow^{\rho_2} C_2 \cdots \Rightarrow^{\rho_n} C_n$, where ρ_i are multisets of applied rules, $1 \leq i \leq n$. We **define** the multiset of rules applied starting from configuration C in n steps as $m(C, n) = \bigcup_{i=1}^{n} \rho_i$.

We write $lhs(x \to y) = x$ and $rhs(x \to y) = y$, and extend this notation to the multiset of rules by taking the union of the corresponding multisets. For instance, if $C \Rightarrow^{\rho} C_1$, then $C_1 = (C \setminus lhs(\rho)) \cup rhs(\rho)$.

Lemma 2.12 *Consider a strongly deterministic P system Π without promoters, inhibitors and without priorities as well as two configurations C, C' with $C \subsetneq C'$ and a number n; then $m(C, n) \subseteq m(C', n)$.*

Proof. We prove the statement by induction. It holds for $n = 1$ step because strongly deterministic systems are deterministic, and if the statement did not hold, then the system even would not be deterministic.

Assume the statement holds for $n - 1$ steps, and

$$C \Rightarrow^{\rho_1} C_1 \Rightarrow^{\rho_2} C_2 \cdots \Rightarrow^{\rho_n} C_n,$$
$$C' \Rightarrow^{\rho'_1} C'_1 \Rightarrow^{\rho'_2} C'_2 \cdots \Rightarrow^{\rho'_n} C'_n.$$

Then, after $n-1$ steps the difference between the configurations can be described by $C'_{n-1} = (C_{n-1} \setminus D_0) \cup D_1 \cup D_2$, where

$$\begin{aligned} D_0 &= lhs(m(C', n-1) \setminus m(C, n-1)), \\ D_1 &= rhs(m(C', n-1) \setminus m(C, n-1)), \\ D_2 &= C' \setminus C. \end{aligned}$$

Therefore, $C'_{n-1} \setminus C_{n-1} \subsetneq D_0$. By the strong determinism property, these objects will either be consumed by some rules from $m(C', n-1) \setminus m(C, n-1)$, or remain idle. Therefore, $m(C_{n-1}, 1) \subseteq m(C'_{n-1}, 1) \cup (m(C', n-1) \setminus m(C, n-1))$, so $m(C, n) \subseteq m(C', n)$. □

Example 2.14 *Let $\Pi = (\{a\}, a, \{p : a^3 \to a\})$. Then $m(C', n) = p^6 \subset p^7 = m(C, n)$:*

$$\begin{aligned} C &= a^{15} \Rightarrow^{p^5} a^5 \Rightarrow^{p} a^4 \Rightarrow^{p} a, \\ C' &= a^{14} \Rightarrow^{p^4} a^6 \Rightarrow^{p^2} a^2. \end{aligned}$$

We now characterize the power of strongly deterministic P systems without additional control: any P system without promoters, inhibitors or priorities accepts either the set of all non-negative integers, or a finite set of all numbers bounded by some number.

Theorem 2.20 $N_aD_sOP_1(coo) = \{\emptyset, \mathbb{N}\} \cup \{\{k \mid 0 \leq k \leq n\} \mid n \in \mathbb{N}\}$.

Proof. A computation starting from a configuration C is not accepting if it does not halt, i.e., if $\lim_{n\to\infty} m(C, n) = \infty$. Due to Lemma 2.12, if the computation starting from C is accepting, then any computation starting from a submultiset $C' \subseteq C$ is accepting, too. This also implies that if the computation starting from C is not accepting, then neither is any computation starting from a multiset containing C. Therefore, the set of numbers accepted by a strongly deterministic P system without additional control can be identified by the largest number of input objects leading to acceptance, unless the system accepts all numbers or none.

The converse can be shown by the following P systems.

- System $(\{a\}, \{a\}, a, \{a \to a\})$ accepts $\emptyset$, because with any input the only computation is an infinite loop;
- system $(\{a\}, \{a\}, \lambda, \{a \to \lambda\})$ accepts $\mathbb{N}$, i.e., anything, because with any input it halts after erasing everything in one step; and
- for any $n \in \mathbb{N}$ there is a system $(\{a\}, \{a\}, \lambda, \{a^{n+1} \to a^{n+1}\})$ accepting $\{k \mid 0 \leq k \leq n\}$, because the system halts with the initial configuration if and only if the input does not exceed n, and enters an infinite loop otherwise. □

Theorem 2.20 shows that the computational power of strongly deterministic P systems without additional control is, in a certain sense, degenerate (it is subregular). We now show that the use of promoters and inhibitors leads to universality of even the strongly deterministic P systems.

Theorem 2.21 $N_aD_sOP_1(coo, pro, inh) = NRE$.

Proof. We reduce the statement of the theorem to the claim that such P systems simulate the work of any deterministic register machine $M = (m, Q, I, q_0, q_f)$. Without restricting generality, we assume that every subtracting instruction is preceded by the corresponding testing instruction. Consider a P system

$$\begin{aligned}
\Pi &= (O, \{r_1\}, q_0, R), \text{ where} \\
O &= \{r_j, d_j \mid 1 \leq j \leq n\} \cup \{q, q_1 \mid q \in Q\}, \\
R &= \{q \to q' r_j \mid (q : [RjP], q', q') \in I\} \\
&\cup \{q \to q_1 d_j, q_1 \to q',\ d_j r_j \to \lambda \mid (q : [RjM], q', q') \in I\} \\
&\cup \{q \to q'|_{r_j}, q \to q''|_{\neg r_j} \mid (q : \langle RjZ\rangle, q', q'') \in I\}.
\end{aligned}$$

All rules using objects q and q' have disjoint domains, except the ones in the last line, simulating the zero/non-zero test. However, they exclude each other by the same object which serves as promoter and inhibitor, respectively. Subtraction of register j is handled by

producing object d_j, which will "annihilate" (i.e., be deleted together with) r_j. Therefore, different instructions subtracting the same r_j are implemented by the same rule $d_j r_j \to \lambda$, hence all rules using objects d_j and r_j have different domains. It follows from Theorem 2.19 that the system is strongly deterministic. □

Remark 2.5 *In the theorem stated above, both promoters and inhibitors were used to eliminate conflicts between the rules for the zero test and for the positive test. By the reasons already explained in Remark 2.4, universality may be obtained by simulating priorities, if the inhibitors are generalized to multiple multisets.*

Notice that, since (unlike in the case of strong reversibility) no inhibiting multiset contains multiple copies of the same symbol, the condition of applicability is a function defined on the signature *of the configuration, i.e., it does not depend on the multiplicities; it is a conjunction of disjunctions of effects of atomic inhibitors (e.g., for a rule* $a \to b|_{\neg cd, \neg ce}$*, such a condition is* $(\neg c \vee \neg d) \wedge (\neg c \vee \neg e)$ *).*

We conjecture that strongly deterministic universality is impossible using just one singleton inhibitor in any rule (with or without promoters). This remark also holds for the sequential case.

Conclusions We have outlined the concepts of reversibility, strong reversibility, determinism, and strong determinism for sequential and maximally parallel multiset rewriting systems and established the results for the computational power of such systems, see Table 2.1 (U - universal, N - non-universal, L - sublinear, C - conjectured to be non-universal.):

Table 2.1: Properties of sequential(top) and maximally parallel (bottom) rewriting

Seq.	pure	Pri	inh	pro, inh
D(acc)	L(Cor.2.15)	U(Th.2.10)	U(Th.2.10)	
D_s(acc)	L(Th.2.15)	U(Cor.2.12)	C(Rem.2.5)	
R(gen)	N(Cor.2.11)	U(Th.2.10)	U(Th.2.10)	
R_s(gen)	L(Cor.2.10)	U(Th.2.13)	C(Rem.2.4)	
Max.par.	pure	Pri	inh	pro, inh
D(acc)	U	U	U	U
D_s(acc)	L(Th.2.20)	U(Cor.2.18)	C(Rem.2.5)	U(Th.2.21)
R(gen)	C(Conj.2.1)	U(Th.2.16)	U(Th.2.16)	U(Th.2.16)
R_s(gen)	L(Cor.2.17)	U(Th.2.18)	C(Rem.2.4)	C(Rem.2.4)

Sequential and maximally parallel multiset rewriting systems with priorities have shown to be universal for all four properties, i.e., for reversibility, strong reversibility, determinism, and strong determinism, whereas without control only deterministic maximally parallel systems are universal and all others are even sublinear except for the reversible classes (in the case of sequential systems, we have shown non-universality, for maximally parallel systems we have not been able to prove such a conjecture yet).

With inhibitors, deterministic and reversible systems are universal in both cases, too, whereas for strong determinism universality could only be shown for maximally parallel systems with inhibitors and promoters (which are of no use in the sequential case). All other remaining classes are conjectured to be non-universal.

Strongly reversible sequential multiset rewriting systems without control do not halt unless the starting configuration is halting, but this is no longer true with inhibitors. For systems with inhibitors or priorities, strong reversibility has also been characterized syntactically; moreover, we have given a syntactic characterization for the property of strong determinism. For sequential systems without control, the power of deterministic systems coincides with that of strongly deterministic systems: such a system without control either accepts all natural numbers, or a finite set of numbers; a similar result holds for strongly deterministic maximally parallel systems.

Note some interesting differences of the sequential case with respect to the parallel one:

- promoters are useless;
- the criterion of strong determinism is different;
- the criterion of strong reversibility is not only decidable, but has a concrete description and thus is easily testable;
- the power of deterministic uncontrolled systems is radically different: sublinear instead of universal.

The author has also investigated determinism and reversibility in circuits of reversible logic elements with memory, [239], [238], [240], [241], [242], and number-conservation in cellular automata, [200]. However, these computational models are rather different from multiset processing and string processing, so we skip the presentation of these results in order to keep the size and the scope of this thesis reasonable.

2.4 Self-stabilization

In this section we discuss a notion of self-stabilization, inspired from biology and engineering. Multiple variants of formalization of this notion are considered, and we discuss how such properties affect the computational power of multiset rewriting systems.

We will call a property dynamic if it depends on the behavior of a system and cannot be easily derived from its description (as opposed to syntactic properties). Given any finite computation, we assume that the property is easily verifiable. The two usual sources of undecidability are a) that we do not always know whether we are dealing with finite or infinite computations, and b) that some properties are defined on infinite number of computations (due to non-determinism, to the initial input or to some other parameter). In the case of this concept, another source of potential undecidability is the finite set to be reached as given in the definitions below. Since in this section we will deal with reachability issues, we would also like to mention the connection with temporal logic [164].

Self-stabilization is a known concept in conventional distributed computing, introduced by E. Dijkstra in [163], as well as in systems biology, but only considered in the framework of membrane computing in [22] and [21]. It has been recalled by Jacob Beal during the Twelfth Conference in Membrane Computing, CMC12, and an attempt to formalize it in the membrane computing framework has been done in [18]. The underlying idea is the tolerance

of natural and engineering systems to perturbations. The formulation from [290], says that A system is self-stabilizing if and only if:

1. Starting from any state, it is guaranteed that the system will eventually reach a correct state (convergence).

2. Given that the system is in a correct state, it is guaranteed to stay in a correct state, provided that no fault happens (closure).

In case of inherently non-deterministic systems, "with probability 1" should be added. Based on this concept, we propose a few formal properties, following the discussion below.

In this section we consider fully cooperative multiset rewriting, possibly with promoters/inhibitors/priorities, operating either in the maximally parallel or the sequential mode. We consider a single working region only, for two reasons. First, the properties of interest are unaffected by flattening the static membrane structure. Second, we would currently like to avoid the discussion about reachability related to "arbitrary configurations" with dynamic membrane structure.

2.4.1 Self-stabilization and related properties

Clearly, "a correct state" should be rephrased as "a configuration in the set of correct configurations". Moreover, we would like to eliminate the set of correct states, let us denote it by S, as a parameter. We say that our property holds if there exists some finite set S of configurations satisfying the conditions 1 and 2 above. Since membrane systems are inherently non-deterministic, we additionally propose two weaker degrees of such a property: possible (there exists a computation satisfying the conditions), almost sure (the conditions are satisfied with probability 1 with respect to non-determinism). Finally, if condition 2 is not required, we call the corresponding property (finite) set-convergence instead of self-stabilization. We now give the formal definitions from [18].

Definition 2.3 *A P system* Π *is possibly converging to a finite set* S *of configurations iff for every configuration* C *of* Π *there exists a configuration* $C' \in S$ *such that* $C \Rightarrow^* C'$.

Definition 2.4 *A P system* Π *is (almost surely) converging to a finite set* S *of configurations iff for every configuration* C *of* Π *the computations starting in* C *reach some configuration in* S *(with probability 1, respectively).*

Definition 2.5 *A P system* Π *is possibly closed with respect to a finite set* S *iff for every non-halting configuration* $C \in S$ *there exists a configuration* $C' \in S$ *such that* $C \Rightarrow C'$.

Definition 2.6 *A P system* Π *is closed with respect to a finite set* S *iff for every non-halting configuration* $C \in S$ $C \Rightarrow C'$ *implies* $C' \in S$.

We say that a system is **(possibly, almost surely) set-converging** if it is (possibly, almost surely, respectively) converging to some finite set of configurations.

We say that a system is **possibly self-stabilizing** if it is possibly converging to some finite set S of configurations and if it is possibly closed with respect to S.

We say that a system is **(almost surely) self-stabilizing** if it is (almost surely, respectively) converging to some finite set S of configurations and if it is closed with respect to S.

Examination of computational aspects of these properties motivates us to add "weakly" to the properties proposed in [18] – *(possibly, almost surely) converging, (possibly) closed, (possibly, almost surely) set-converging, (possibly, almost surely) self-stabilizing* – if the corresponding conditions over configurations C only span the **reachable non-halting** ones.

Another comment we can make on "almost sure" is that such a property may depend on how exactly the transition probability is defined. The easiest way is to assign equal probabilities to all transitions from a given configuration. Alternatively, to a transition via a multiset of rules $r_1^{n_1} \cdots r_m^{n_m}$ we may assign the weight of a multinomial coefficient $\binom{n_1+\cdots+n_m}{n_1,\cdots,n_m} = \frac{(n_1+\cdots+n_m)!}{n_1!\cdots n_m!}$, which will make the corner cases less probable than the average ones. There can be other ways to define transition probabilities, but we would like to discuss the properties of interest without fixing a specific way. We assume the transition probabilities in an independent subsystem are the same as if it were the entire system.

An important assumption we impose on the probability distribution is that *the probability of each transition is uniquely determined by the associated multiset of rules and by the set of all applicable multisets of rules*, yet it does not depend on the objects that cannot react, or by the previous history of the computation.

2.4.2 Accepting systems

For the following theorem we consider any computationally complete model of P systems as defined above, e.g., a model with maximally parallel multiset rewriting or with controlled sequential multiset rewriting.

Theorem 2.22 *If a model of P systems yields a computationally complete class, then the weakly self-stabilizing subclass accepts exactly $NREC$.*

The proof can be found in [21].

Strengthening this result by removing "weakly" is problematic, even if more powerful P systems are used. Indeed, self-stabilization also from unreachable configurations would need to handle not only the configurations without any state or with multiple states (which could be handled with the joint power of maximal parallelism and priorities), but also configurations representing a situation with only one state which is not the initial state of the underlying register machine. We have to leave this question open.

Theorem 2.23 *If a model of P systems yields a computationally complete class, then the weakly almost surely self-stabilizing P systems of this class accept exactly NRE.*

The proof can be found in [21].

Theorem 2.24 *If a model of P systems yields a computationally complete class, then the class of all almost surely self-stabilizing maximally parallel/sequential P systems with priorities accepts exactly NRE.*

Proof. Given a set L from NRE, we first construct a P system Π simulating a register machine M accepting L and then extend Π to a P system Π' even fulfilling the condition of almost surely self-stabilizing.

Let $M = (m, Q, I, q_0, q_f)$ a deterministic register machine accepting L. We now construct the P system $\Pi = (O, q_0, R, >)$ with priorities accepting L:

$$\begin{aligned} O &= Q \cup \{a_j \mid 1 \leq j \leq m\}, \\ R &= \{q_1 \to a_j q_2 \mid (q_1 : [RjP], q_2) \in I\} \\ &\cup \{a_j q_1 \to q_2, q_1 \to q_3 \mid (q_1 : \langle RjZM\rangle, q_2, q_3) \in I\} \\ > &= \{a_j q_1 \to q_2 > q_1 \to q_3 \mid (q_1 : \langle RjZM\rangle, q_2, q_3) \in I\}. \end{aligned}$$

The contents of a register j, $1 \leq j \leq m$, is represented by the number of symbols a_j in Π. The state q of the register machine is represented by the corresponding symbol q in Π, too. When M halts in q_f with all registers being empty, Π also halts with the configuration $\{q_f\}$. Obviously, Π accepts L, both in the sequential as well as in the maximally parallel mode.

To strengthen the result to even non-weak almost sure self-stabilization, we have to take into account the non-reachable configurations, too. The almost surely self-stabilizing P system $\Pi' = (O', q_0, R', >')$ with priorities accepting L is constructed as follows:

$$\begin{aligned} O' &= Q \cup \{a_j \mid 1 \leq j \leq m\} \cup \{e\}, \\ R' &= \{q_1 \to a_j q_2 \mid (q_1 : [RjP], q_2) \in I\} \\ &\cup \{a_j q_1 \to q_2, q_1 \to q_3 \mid (q_1 : \langle RjZM\rangle, q_2, q_3) \in I\} \\ &\cup \{a_j \to e \mid 1 \leq j \leq m\} \cup \{ex \to e \mid x \in O'\} \cup \{e \to e\} \\ &\cup \{q \to e \mid q \in Q \setminus \{q_f\}\} \cup \{qq' \to e \mid q, q' \in Q\}, \\ >' &= \{a_j q_1 \to q_2 > q_1 \to q_3 \mid (q_1 : \langle RjZM\rangle, q_2, q_3) \in I\} \\ &\cup \{ex \to e > r, qq' \to e > r \mid q, q' \in Q, x \in O', r \in R\} \\ &\cup \{q \to e > a_j \to e \mid q \in B \setminus \{q_f\}, 1 \leq i \leq m\} \\ &\cup \{r > e \to e \mid r \in R' \setminus \{e \to e\}\}. \end{aligned}$$

In addition to the idea of the construction in the proof of Theorem 2.23 using the exit e by applying a rule $q \to e$, $q \in Q \setminus \{q_f\}$, it suffices to self-stabilize from the configurations with no state and from the configurations with multiple states of the register machine. Multiple states can be reduced by the rules $qq' \to e$, $q, q' \in Q$. If no state symbol is present, then we may exit with one of the rules $a_j \to e$, $1 \leq j \leq m$. All remaining cases can be captured by the rules $ex \to e$, $x \in O'$. By construction, the self-stabilizing set S equals $\{\{q_f\}, \{e\}, \emptyset\}$. The whole construction again is valid for both sequential and maximally parallel mode. □

It is open whether priorities in Theorem 2.24 can be replaced by promoters or inhibitors.

2.4.3 Generating systems

Theorem 2.25 *Any finite set M of numbers can be generated by some self-stabilizing membrane system without control.*

Proof. Consider a P system $\Pi = (\{s, a\}, s, R)$, where

$$R = \{s \to a^n \mid n \in M\} \cup \{a^{\max(M)+1} \to \lambda, ss \to s\}.$$

It is not difficult to see that Π generates M and (taking $S = \{a^n \mid n \leq \max(M)\} \cup \{s\}$) it is self-stabilizing. □

Since self-stabilization implies set-convergence and closure, and relaxing either property (to possibly, almost surely and/or weakly) does not compromise the construction of the P system described in the proof of Theorem 2.25, the lower bound on the generative power of associated systems restricted to any property we have defined, is at least $NFIN$.

Lemma 2.13 *A possibly finite set-converging system only generates finite sets.*

Proof. By Definition 2.3, for a system possibly converging to a set S, S contains all halting configurations. Since S is finite, so is the set of all the halting configurations. Hence, at most $NFIN$ is generated. □

Theorem 2.26 *Any of the following classes* $\mathbf{dp}OP^{\mathbf{m}}(\mathbf{c})$ *generate exactly* $NFIN$*:*
d *is possibly/almost surely/ -*
p *is self-stabilizing/finite set-converging*
m *is maximally parallel/sequential*
c *is uncontrolled/with promoters/with inhibitors/with priorities.*

Proof. The claims directly follow from Theorems 2.25 and 2.26. □

We now proceed to weak properties of generative systems.

Theorem 2.27 *Weakly almost surely self-stabilizing P systems generate exactly* $NFIN$*.*

Proof. The lower bound is shown by Theorem 2.25. Now take a weakly self-stabilizing P system Π, and its associated set S from the definition of the property. Consider an arbitrary halting computation of Π. Let C be the configuration of Π one step before the halting. Interpreting finite set-convergence for C implies that the halting configuration belongs to S. Since the halting computation has been arbitrarily chosen, the set of all halting configurations is a subset of S, and hence it is finite. Therefore, the set generated by Π is finite, too. □

Theorem 2.28 *If a model of P system yields a computationally complete class, then weakly possibly self-stabilizing subclass generates* NRE*.*

Proof. Consider the construction from Theorem 2.23, but for a generative P system. The simulation of the underlying register machine is carried out until some point. Unless the P system has already halted, it always has a choice to self-stabilize and loop. □

Table 2.2: Results (letter F stands for "generate exactly $NFIN$")

Property	comput. complete	(sequ/maxpar) +pri	Thm
self stabilizing	acc. $?/F$		-/2.26
almost surely s.s.	acc. $?/F$	acc. NRE/F	2.24/2.26
possibly s.s.	acc. $?/F$	acc. NRE/F	2.24/2.26
weakly s.s.	acc. $NREC/F$		2.22/2.27
weakly almost surely s.s.	acc. NRE/F		2.23/2.27
weakly possibly s.s.	acc. NRE/gen. NRE		2.23/2.28

Conclusions We have presented some results (some of them summarized in Table 2.2) concerning the notion of self-stabilization, recently proposed for membrane computing. Its essence is reachability and closure of a finite set.

One of the questions we proposed is whether priorities may be replaced by promoters or inhibitors in Theorem 2.24. Another open question is the power of accepting with unrestricted self-stabilization, even if maximal parallelism is combined with priorities (a comment after Theorem 2.22 and the first question mark in the table above). The other open questions are also marked with question marks in the table above. Any system in the corresponding classes must (besides doing the actual computation) converge (definitely, in probability or possibly) to some finite set from anywhere, without using the joint power of maximal parallelism and control.

2.5 Membrane Creation

In this section, it is essential that not only multiset rewriting is distributed over a tree structure, but also that such a structure is dynamic. We present quite surprising results that non-cooperative rewriting reaches computational completeness when equipped with membrane creation and membrane dissolution. Intuitively, this is possible due to the cooperation between an object and a fact of existence of a membrane.

The aim of this section is to present the improvements of descriptive complexity parameters or properties of a few universality results. More precisely, we shall speak about object complexity (bounds in the starting configuration, in any configuration, in the alphabet) and also about membrane complexity.

It was shown in [17] that P systems with membrane creation generate $PsRE$, even when every region (except the environment) contains at most one object of every kind, but using unbounded membrane labels. We will show that they generate all recursively enumerable *languages*, and *two membrane labels* are sufficient (the same result holds for *accepting* all recursively enumerable vectors of non-negative integers). Moreover, at most *two objects* are present inside the system at any time in the generative case. On the other hand, using an unbounded membrane alphabet we can bound the symbol alphabet by $10+m$ objects, where m is the size of the output alphabet.

Finally, we show that P systems with *restricted* membrane creation (only of the *same kind* as parent) generate at least matrix languages, and so do P systems with membrane

creation having at most one object in the configuration (except the environment). Remarks and open questions are presented.

Figure 2.3 describes the membrane structure used in the theorems.

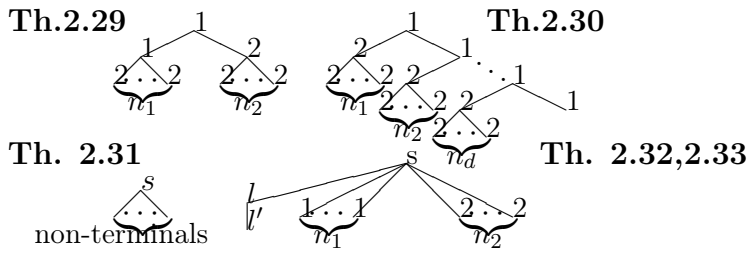


Figure 2.3: Membrane structures for membrane creation proofs

Generating The first theorem shows how recursively enumerable languages can be generated by P systems with a small number of objects inside the system and a small number of membrane labels.

Theorem 2.29 $LO_{1,2,*}P_{1,*,2}(ncoo, tar, mcre, \delta) = RE$.

Proof. Due to Proposition 1.4, we construct a P system simulating a register machine $M = (2, T, Q, I, q_0, q_f)$; I_- denotes the set of all SUB instruction labels.

$$\begin{aligned}
\Pi &= (O, H, [\ ]_1, w_1, R_1, R_2),\\
O &= T \cup \{a_1, a_2, C_1, C_2, g_0, g_1, g_2, t\} \cup P \cup \{q_i \mid q \in I_-,\ 1 \le i \le 7\},\\
H &= \{1, 2\},\ w_1 = g_0,\\
R_1 &= R_{1,I} \cup R_{1,A} \cup R_{1,S} \cup R_{1,D} \cup R_{1,Z} \cup R_{1,O},\\
R_2 &= R_{2,I} \cup R_{2,A} \cup R_{2,S} \cup R_{2,D} \cup R_{2,Z}.
\end{aligned}$$

For clarity, the rules are grouped in categories (initialization, add, subtract, decrement case, zero case, output).

$$\begin{aligned}
R_{1,I} &= \{g_0 \to [\ g_0\ ]_2,\ g_1 \to [\ g_2\ ]_1,\ g_2 \to (l_0)_{out}\},\\
R_{2,I} &= \{g_0 \to (g_1)_{out}\},\\
R_{1,O} &= \{q \to q' a_{out} \mid a \in T,\ (q : [SaW], q') \in I\},\\
R_{1,A} &= \{q \to q'(C_2)_{in_j},\ q \to q''(C_2)_{in_j} \mid (q : [RjP], q', q'') \in I,\ j \in \{1, 2\}\}\\
&\cup\ \{C_2 \to [\ t\ ]_2,\ t \to \lambda\},\\
R_{2,A} &= \{C_2 \to [\ t\ ]_2,\ t \to \lambda\},\\
R_{1,S} &= \{q \to (q_1 C_1)_{in_j} \mid (q : \langle RjZM\rangle, q', q'') \in I,\ j \in \{1, 2\}\}\\
&\cup\ \{C_1 \to [\ t\ \ ]_1\} \cup \{q_1 \to (q_2)_{in_2} \mid (q : \langle R1ZM\rangle, q', q'') \in I\},\\
R_{2,S} &= \{C_1 \to [\ 1t\ ]_1\} \cup \{q_1 \to (q_2)_{in_2} \mid (l : \langle R2ZM\rangle, l', l'') \in I\},
\end{aligned}$$

$$
\begin{aligned}
R_{1,D} &= \{q_4 \to q_5\delta \mid q \in I_-\} \\
&\cup \{q_3 \to (q_4)_{in_1},\ q_5 \to (q')_{out} \mid (q : \langle R1ZM\rangle, q', q'') \in I)\}, \\
R_{2,D} &= \{q_2 \to q_3\delta \mid q \in I_-\} \\
&\cup \{q_3 \to (q_4)_{in_1},\ q_5 \to (q')_{out} \mid (q : \langle R2ZM\rangle, q', q'') \in I\}, \\
R_{1,Z} &= \{q_6 \to q_7\delta \mid q \in I_-\} \\
&\cup \{q_1 \to (q_6)_{in_1},\ q_7 \to (q'')_{out} \mid (q : \langle R1ZM\rangle, q', q'') \in I\}, \\
R_{2,Z} &= \{q_1 \to (q_6)_{in_1}, q_7 \to (q'')_{out} \mid (q : \langle R2ZM\rangle, q', q'') \in I\}.
\end{aligned}
$$

Initially, by means of the auxiliary objects g_j, we create two membranes inside the skin region, labeled by 1 and 2, respectively. These membranes will be referred to as *cluster-membranes* (because they will contain inside them a number of elementary membranes). We finish the initialization phase by generating an object q_0 in the skin region.

The values of the two registers j, $j \in \{1, 2\}$, are represented by the number of elementary membranes labeled by 2 that occur inside the corresponding cluster-membrane i. The duty of the object C_j is to create membrane j. Object t is not needed for the computation, it is only used to keep the usual form of membrane creation rules and is immediately erased.

Writing an output symbol $a \in T$ is done by a non-cooperative rule changing the instruction label and producing a symbol a that is immediately sent out. To increment a register, a membrane labeled 2 is created in the corresponding cluster-membrane.

To simulate a subtraction of register j we send objects q_1 and C_1 into cluster-membrane j and then proceed as follows: while creating a membrane with label 1, object q_1 tries to enter some membrane with label 2 as q_2. If such a membrane exists (i.e., register j is not empty), then q_2 changes to q_3 and dissolves the membrane, thus being spilled back into the cluster-membrane. Before proceeding to the next label, we have to get rid of the auxiliary membrane 1 created inside the cluster-membrane by C_1. To this aim, q_3 enters membrane 1 as q_4 and dissolves it, thereby changing to q_5. Finally, q_5 sends object q' out to the skin region. *Overall*, q has been replaced by q' and the number of membranes with label 2 inside the cluster-membrane j has been reduced by 1. If, on the other hand, no membrane with label 2 exists in the cluster membrane, then q_1 waits for one step and then enters the newly created membrane 1 as q_6. Immediately afterwards, it changes to q_7 and dissolves the membrane. Finally, q_7 sends out object q'' into the skin region. *Overall*, in the absence of membranes with label 2 inside the cluster-membrane j, q has been replaced by q''.

Note that inside the system there can never be more than one copy of the same object. In fact, the number of objects inside the system never exceeds two (it is two after the first step of an ADD or a SUB instruction). □

Accepting Notice that the simulation of the register machine instructions in Theorem 2.29 is deterministic (the non-determinism arises from the non-determinism of the register machine program itself, not from the simulation).

Theorem 2.30 $DPs_aOP_{1,*,2}(ncoo, tar, mcre, \delta) = PsRE$.

The proof of this result can be found in [73].

Generating with one object Considering P systems only having one object inside the system during the whole computation, we realize that such P systems with one object work sequentially. Hence, the following holds:

Theorem 2.31 *$PsMAT$ is characterized by P systems $\Pi = (O, H, \mu, w_1, \cdots, w_m, R_1, \cdots, R_n)$, where (a) the initial membrane structure is limited by two levels (any membrane inside the skin is elementary) and label 1 of the skin membrane is unique, (b) exactly one of the multisets w_i consist of exactly one object, whereas all the other initial multisets are empty, (c) the rules in R_1 are of the forms $a \to bu_{out}$, $a \to b_{in_i}u_{out}$, $a \to b$, or $a \to [\ b\ ]_i$ with $a, b \in O$, $u \in O^*$, and $i \in H'$ where $H' = H \setminus \{s\}$, (d) the rules in R_i, $2 \leq i \leq n$, are of the forms $a \to b$, $a \to b_{out}$, or $a \to b\delta$.*

We do not present the proof here; an interested reader can find it in [73].

We conjecture that we need not restrict the membrane structure, i.e., $PsMAT = PsO_{1,1,*}P_{*,*,*}(ncoo, tar, mcre, \delta)$.

Number of Symbols The size of the alphabet in the computational completeness proofs usually depends on the complexity parameters of the simulated device. We now show that any recursively enumerable set of m-dimensional vectors of numbers can be generated by P systems with membrane creation and dissolution with the alphabet of $10 + m$ symbols.

Theorem 2.32 $L(m)O_{1,2,10+m}P_{2,*,*}(ncoo, tar, mcre, \delta) = RE(m)$.

Proof. We simulate a register machine $M = (2, T, Q, P, q_0, q_f)$ where $T = \{a_j \mid 1 \leq j \leq m\}$; let P_+ denote the set of all ADD instruction labels, and let P_- denote the set of all SUB instruction labels.

$$\begin{aligned}
\Pi &= (O, H, [\ [\ ]_I\ ]_s, \lambda, b, R_{q_0}, \cdots, R_D),\\
O &= T \cup \{b, c, d, e\} \cup \{r_+, r_-, r'_- \mid r \in \{1, 2\}\},\\
H &= P \cup \{I, 1, 2, s, D\},\\
R_i &= R_{i,I} \cup R_{i,O} \cup R_{i,A} \cup R_{i,S} \cup R_{i,D} \cup R_{i,Z} \cup R_{i,N},\ i \in H.
\end{aligned}$$

For clarity, the rules are grouped (initialization, output, add, subtract, decrement case, zero case, next instruction).

$$\begin{aligned}
R_{I,I} &= \{b \to [\ d\ ]_{q_0}\},\\
R_{i,I} &= \emptyset,\ i \in H \setminus \{I\},\\
R_{q,O} &= a \to a_{out} \mid (q : [SaW], q'),\ a \in T\},\\
&\cup\ \{a \to a\delta \mid a \in T\},\ q \in Q \cup \{I\},\\
R_{s,O} &= \{a \to a_{out}b,\ a \to a_{out}c \mid a \in T\},\\
R_{i,O} &= \emptyset,\ i \in H \setminus (P \cup \{I, s\}),
\end{aligned}$$

$$
\begin{aligned}
R_{q,A} &= \{d \to (r_+)_{out} \mid (q : ((q_r, [R, P], q_q)', q'') \in P,\ r \in \{1,2\}\} \\
&\cup \{r_+ \to r_+\delta \mid r \in \{1,2\}\},\ q \in P \cup \{I\}, \\
R_{s,A} &= \{r_+ \to [\ d\ ]_r \mid r \in \{1,2\}\}, \\
R_{r,A} &= \{d \to b_{out}, d \to c_{out}\},\ r \in \{1,2\}, \\
R_{D,A} &= \emptyset, \\
R_{q,S} &= \{d \to (r_-)_{out} \mid (q : \langle RrZM\rangle, q', q'') \in P,\ r \in \{1,2\}\} \\
&\cup \{r_- \to r_-\delta \mid r \in \{1,2\}\},\ q \in P \cup \{I\}, \\
R_{s,S} &= \{r_- \to r'_-d,\ d \to [\ d\ ]_D \mid r \in \{1,2\}\}, \\
R_{D,S} &= \{d \to \lambda\}, \\
R_{i,S} &= \emptyset,\ i \in H \setminus P \setminus \{I, s, D\}, \\
R_{s,D} &= \{r'_- \to (r'_-)_{in_r},\ e \to e_{in_D} \mid r \in \{1,2\}\}, \\
R_{r,D} &= \{r_- \to r'_-d,\ d \to [\ d\ ]_D\},\ r \in \{1,2\}, \\
R_{D,D} &= \{r'_- \to e_{out}\},\ r \in \{1,2\}, \\
R_{i,D} &= \emptyset,\ i \in H \setminus \{1, 2, s, D\}, \\
R_{s,Z} &= \{r'_- \to (r'_-)_{in_D}\} \mid r \in \{1,2\}\}, \\
R_{D,Z} &= \{r'_- \to b\delta \mid r \in \{1,2\}\}, \\
R_{i,D} &= \emptyset,\ i \in H \setminus \{s, D\}, \\
R_{s,N} &= \{b \to b_{in_q},\ c \to c_{in_q} \mid q \in P\}, \\
R_{q,N} &= \{b \to [\ a\ ]_{q'},\ c \to [\ a\ ]_{q''} \mid q : (OP, q', q'') \in P,\ OP \in \{[RrP], \langle RrZM\rangle\}\}.
\end{aligned}
$$

The instruction labels are encoded into membrane labels, and the values of the registers are encoded by the number of copies of some membranes associated with them. The proof mainly relies on the fact that the amount of information needed to be transmitted between the instructions and the registers is "small", i.e., the instructions tell us which operation (ADD or SUB, represented by r_+, r_-, $r \in \{1,2\}$) has to be applied and to which register r it has to be applied. The objects r'_-, $r \in \{1,2\}$, and e are used to implement the SUB instruction, and the object d here is used to organize a delay for the appearance checking, similar to the technique from Theorem 2.29, whereas otherwise it is used when the membranes already contain all the information needed.

After an operation is simulated, the next instruction is chosen from two variants, non-deterministically chosen in the ADD case and as well in the SUB case here depending on whether decrementing has been successful or not. These variants are represented by objects b, c. The transition to the next instruction is done as follows: object b in membrane q creates membrane q', or object c in membrane q creates membrane q''. After this, the object "memorizes" the next register to be operated on and the operation to be performed, and then membrane q is dissolved, leaving the newly created membrane in the skin. □

If we want to start with the simplest membrane structure, it suffices to use one more symbol, as is exhibited in the following:

Theorem 2.33 $L(m)O_{1,2,11+m}P_{1,*,*}(ncoo, tar, mcre, \delta) = RE(m)$.

Proof. We again simulate a register machine $M = (2, T, Q, P, q_0, q_f)$ as in the proof of the preceding theorem, but in the P system Π' we use an additional symbol a for an initial step starting in the skin membrane:

$$\begin{aligned}
\Pi' &= (O, H, [\ ]_s, a, R_{q_0}, \cdots, R_D), \\
O &= T \cup \{a, b, c, d, e\} \cup \{r_+, r_-, r'_- \mid r \in \{1, 2\}\}, \\
H &= P \cup \{I, 1, 2, s, D\}, \\
R_i &= R_{i,I} \cup R_{i,O} \cup R_{i,A} \cup R_{i,S} \cup R_{i,D} \cup R_{i,Z} \cup R_{i,N},\ i \in H. \\
R_{s,I} &= \{a \to [\ b\ ]_I\}, \\
R_{I,I} &= \{b \to [\ d\ ]_{q_0}\}, \\
R_{i,I} &= \emptyset,\ i \in H \setminus \{s, I\}.
\end{aligned}$$

Except for the initialization, the sets of rules are exactly the same as for the P system Π constructed in the preceding proof, which observation already completes this proof. □

Restricted Membrane Creation

We now consider restricted membrane creation: in region i it is only possible to create membranes with label i.

Theorem 2.34 $PsO_{1,1,*}P_{*,*,*}(ncoo, tar, mcre_r, \delta) \supseteq PsMAT$.

We do not present the proof here; an interested reader can find it in [73].

Conclusions and Open Problems We have shown that P systems with membrane creation generate RE, using two membrane labels and at most two objects present inside the system throughout the computation. Accepting any recursively enumerable language can also be done with two membrane labels. On the other hand, it is possible to bound the number of symbols by $m + 10$ and still generate $RE(m)$, provided that the number of membrane labels is unbounded.

We also have shown that RE is generated by P systems using four membranes and three labels or seven membranes and two labels in the initial configuration, where at most three objects are ever present in any halting computation.

Improving any complexity parameter greater than one (especially in the case of $*$) in any theorem is an open question. Moreover, the following questions are of interest:

- What is the power of P systems with membrane creation and *one object*?
- What is the power of P systems with *restricted* membrane creation?
- How can target indications be restricted in Theorem 2.29?
- What further restrictions cause a complexity trade-off?

2.6 Conclusions to Chapter 2

The family of languages generated by transitional P systems without cooperation and without additional control has been reconsidered. It was shown that one membrane is enough, and a characterization of this family was given via derivation trees of context-free grammars. Next, three normal forms were given for the corresponding grammars. It was than shown that the membrane systems language family lies between $REG \bullet \texttt{Perm}(REG)$ and context-sensitive semilinear polynomially parsable languages, and it is incomparable with linear and with context-free languages. An example of a considerably more "difficult" language was given than the lower bound mentioned above.

The membrane systems language family was shown to be closed under union, permutations, erasing/renaming morphisms. It is not closed under intersection, intersection with regular languages, complement, concatenation or taking the mirror image. The following are examples of questions that are still not answered: Does $LOP(ncoo, tar) \subseteq MAT$ hold? Is $LOP(ncoo, tar)$ closed under arbitrary morphisms? (Conjecture: no.) Characterize $LOP(ncoo, tar)$.

It has been shown that, for non-cooperative P systems with promoters and/or inhibitors (with or without priorities), determinism is a borderline criterion between universality and decidability. In fact, for non-cooperative P systems working in the maximally parallel or the asynchronous mode, we have computational completeness in the unrestricted case, and only all finite number sets and their complements in the deterministic case.

The concepts of reversibility, strong reversibility, determinism, and strong determinism have been outlined for sequential and maximally parallel multiset rewriting systems and established the results for the computational power of such systems, see Table 2.1.

Sequential and maximally parallel multiset rewriting systems with priorities have shown to be universal for all four properties, i.e., for reversibility, strong reversibility, determinism, and strong determinism, whereas without control only deterministic maximally parallel systems are universal and all others are even sublinear except for the reversible classes (in the case of sequential systems, we have shown non-universality, for maximally parallel systems we have not been able to prove such a conjecture yet).

With inhibitors, deterministic and reversible systems are universal in both cases, too, whereas for strong determinism universality could only be shown for maximally parallel systems with inhibitors and promoters (which are of no use in the sequential case). All other remaining classes are conjectured to be non-universal.

Strongly reversible sequential multiset rewriting systems without control do not halt unless the starting configuration is halting, but this is no longer true with inhibitors. For systems with inhibitors or priorities, strong reversibility has also been characterized syntactically; moreover, we have given a syntactic characterization for the property of strong determinism. For sequential systems without control, the power of deterministic systems coincides with that of strongly deterministic systems: such a system without control either accepts all natural numbers, or a finite set of numbers; a similar result holds for strongly deterministic maximally parallel systems.

Note the interesting differences of the sequential case with respect to the parallel one: promoters are useless; the strong determinism criterion is different; the strong reversibility

criterion is not only decidable, but is easily testable via a concrete description; the power of deterministic uncontrolled systems is radically different: sublinear instead of universal.

Some results have been presented (some of them summarized in Table 2.2) concerning the notion of self-stabilization, recently proposed for membrane computing. Its essence is reachability and closure of a finite set.

One of the questions we proposed is whether priorities may be replaced by promoters or inhibitors in Theorem 2.24. Another open question is the power of accepting with unrestricted self-stabilization, even if maximal parallelism is combined with priorities. The other open questions are also marked with question marks in the table above. Any system in the corresponding classes must (besides doing the actual computation) converge (definitely, in probability or possibly) to some finite set from anywhere, without using the joint power of maximal parallelism and control.

P systems with membrane creation have been shown to generate RE, using two membrane labels and at most two objects present inside the system throughout the computation. Accepting any recursively enumerable language can also be done with two membrane labels. On the other hand, it is possible to bound the number of symbols by $m+10$ and still generate $RE(m)$, provided that the number of membrane labels is unbounded.

We also have shown that RE is generated by P systems using four membranes and three labels or seven membranes and two labels in the initial configuration, where at most three objects are ever present in any halting computation.

Improving any complexity parameter greater than one (especially in the case of $*$) in any theorem is an open question. Moreover, the following issues are of interest: the power of P systems with membrane creation and *one object*; the power of P systems with *restricted* membrane creation; restricting target indications in Theorem 2.29; further restrictions that cause a complexity trade-off.

Section 2.1 is based on publications [43], [41], [44], [45], [46], [42], [47]. Section 2.2 is based on publications [56], [57], [58] (and we mention [135], [134], [133]). Section 2.3 is based on publications [65], [63], [64], [105], [106], and [104] (and we mention [239], [238], [240], [241], [242], [200]). Section 2.4 is based on publications [21], [22] and [18]. Section 2.5 is based on publications [73], [17], [74] and [75].

3. SYMPORT/ANTIPORT

This chapter is dedicated to showing the power of systems that only move objects between the nodes of the underlying tree, without changing the objects. Of course, the power of such systems is subregular unless there exists a region with unbounded supply of some objects.

In Section 3.1 we present a small universal antiport P system from [136], constructed by simulating the universal register machine U_{22} from [207], see Figure 1.1 in Subsection 1.1.5.

In Section 3.2 we investigate the power of communication – the P systems evolving by communicating objects between regions. Computational completeness can already be obtained with one membrane using antiport rules (objects are communicated in different directions) or symport rules (objects go together in the same direction) of size three, i.e., involving three objects. Applying the communication rules in the minimally parallel mode, we need two membranes to achieve computational completeness. Acceptance can even be performed by deterministic P systems with antiport or symport rules. Computational completeness can be obtained with a rather small number of objects. The author has a significant contribution to the results covered in this state-of-the art (including, but not limited to the publications covered in his Ph.D. thesis, see also the Bibliographic Notes in the end of the section).

Symport rules move multiple objects to a neighboring region. It is known that for P systems with symport rules of weight at most 3 and a single membrane, 7 superfluous symbols are enough for computational completeness, and 1 is necessary.

In Section 3.3 we present the improvements of the lower bounds on the generative power of P systems with symport of weight bounded by 3 and 4, in particular establishing that 6 and 2 extra symbols suffice, respectively. Besides maximally parallel P systems, we also consider sequential ones. In fact, all presented non-universality lower bound results, together with all upper bound results, hold also in this case, yielding the current state-of-the-art.

3.1 Universality with Small Number of Rules

In this section we present a small universal antiport P system from [136], constructed by simulating the universal register machine U_{22} from [207], see Figure 1.1 in Subsection 1.1.5.

Theorem 3.1 *There exists a universal antiport P system with 23 rules.*

The proof has been presented in terms of maximally parallel multiset rewriting systems. Indeed, a multiset rewriting system directly correponds to a one-membrane symport/antiport system with environment containing an unbounded supply of all objects, and rule $u \rightarrow v$ corresponds to rule $(u, out; v, in)$.

We now present the formal description of the system; the flowchart representing its finite state transition graph is illustrated by Figure 3.2:

$$\begin{aligned}
\gamma &= (O, R, \{R_1\}, \mathcal{I}, \mathcal{P}), \text{ where} \\
O &= R \cup \{C_3, C_5', C_6'\} \cup \{q_{16}, q_{27}\} \cup \{T, I, J, K, L, M, N, O, P, Q, T, X\}, \\
R &= \{R_i \mid 0 \leq i \leq 7\}, \\
\mathcal{I} &= LQLQJJNXXXR_0^{i_0} \cdots R_7^{i_7}.
\end{aligned}$$

Here $i_0, \cdots, i_7$ is the contents of registers 0 to 7 of U_{22} and $LQLQJJNXXX$ is the encoding of the initial state q_1C_1S. Table 3.1 gives the set $\mathcal{P}$ of rules.

Table 3.1: 23 rules of a universal antiport P system

phase	:	$XX \to XT$
$D0$	:	$IJKPQR_0 \to LQLQJJM$
$D1$	:	$LQLQJJNR_1 \to LPLPJJMR_7$
$D2$	:	$IIKPQR_2 \to JJKPQ$
$D3$	:	$q_{27}C_3R_3 \to JJKPQ$
$D4$	:	$JJKR_4 \to JJLLM$
$D5$	:	$JJOR_5 \to C_5'$
$D6$	:	$IJLR_6 \to C_6'$
$D7$	:	$IILQLQNR_7 \to IJLOR_1$
A	:	$ITT \to JXX$
B	:	$JJMTT \to JJNXX$
C	:	$LP \to LQ$
a	:	$LQLQJJNTT \to JJLOR_6XX$
b	:	$LC_5'TT \to JJLOR_6XX$
c	:	$OC_6'TT \to IILQLQNR_5XX$
d	:	$QLQNC_6'TT \to JJKQQR_6XX$
e	:	$q_{27}C_3TT \to LQLQJJNR_0XX$
f	:	$q_{16}JJOC_5'C_5'TT \to LQLQJJNR_2R_3XX$
g	:	$q_{16}C_5'C_5'C_5'TT \to q_{16}JJOJJOJJOXX$
1	:	$JJLOTT \to IJLOXX$
5	:	$JJKQQTT \to q_{16}JJOJJOJJOXX$
8	:	$q_{16}JJOJJOJJOTT \to IIKPQMXX$
12	:	$q_{16}JJOJJOC_5'TT \to q_{27}C_3XX$

In fact, by simulation all objects except $R_0, \cdots, R_7$ appear inside the system in bounded quantities, so the constructed system is explained by projections of configurations onto $O' = O \backslash \{R_0, \cdots, R_7\}$, yielding a finite transition graph. We refer to its nodes as *finite states*. The possibility of some transitions, however, depends on the availability of objects R_j, $0 \leq j \leq 7$. In [136] one thus speaks about *finite-state maximally parallel multiset rewriting systems* (FsMPMRSs).

Machine U_{22} may be simulated in a *straightforward* way, by rule $q \to R_iq_1$ for each instruction $(q, [R_kP], q_1)$ and by rules

$$q \to q'C_q,\ q' \to q'',\ C_qR_{i_q} \to C_q',\ q''C_q \to q_1,\ q''C_q' \to q_2$$

for each instruction $(q, \langle R_{i_q}ZM\rangle, q_1, q_2)$. This yields a universal P system with 73 symport/antiport rules, reported already in [157] (together with some optimizations). The number of rules is then decreased at the expense of their weight. The overall behavior eventually gets quite complicated, so flowcharts are used to describe it. A square represents a finite state (see the previous paragraph), and a circle attached to it represents a (possibly partial) application of rules; multiple circles may be drawn as one for simplicity.

Multiple techniques are used to decrease the number of rules. First, if one rule (e.g., increment) is always applied after another one, then they can be merged, *eliminating an intermediate state*. A state then typically contains a *checker* (object C with an index, possibly primed), verifying whether a specific register is present in the system (is non-zero); addition instructions and renaming rules are no longer present as separate rules. This increases the weight of rules to 5.

A very important optimization is *gluing*: the representation of the configurations is changed such that the effect of multiple rules is obtained by one rule. A general scheme

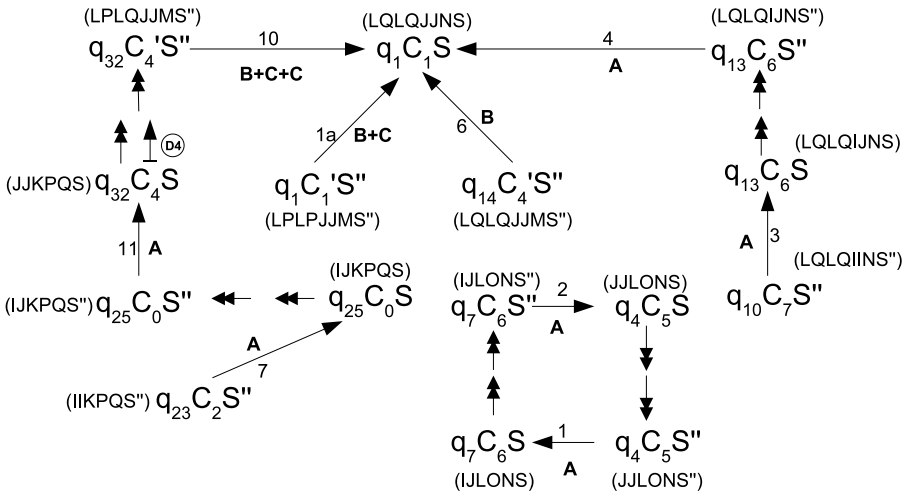


Figure 3.1: Part of the multiset rewriting flowchart of U_{22}: only glued rules and encoding

is the following: suppose we have rules $r_1 : c_1 \to c_2$ and $r_2 : d_1 \to d_2$. They both can be replaced by a rule $r : X \to Y$ if we transform the representation as follows: $c_1 = cX$, $c_2 = cY$, $d_1 = dX$, $d_2 = dY$. It is, however, needed that no state is a submultiset of another state.

We now proceed with two simple special cases of gluing. The first case is *phases*. Representing states q and q' by qS and qS' lets us glue all rules $q \to q'$ (waiting while the checker gets a chance to decrement a register) yielding a single rule $S \to S'$. Later, *three phases* help to further optimize the other rules, but the transitions $S \to S'$ and $S' \to S''$ are also glued by substitution $S = XXX$, $S' = XXT$, $S'' = XTT$ yielding a single rule $XX \to XT$. This phase rule is represented on flowcharts by a double-headed arrow.

The second simple special case of gluing is *unifying the checkers* that decrement the same register. Now the state typically contains a phase, a checker, and the rest of the state is currently a symbol q with an index, derived from U_{22}. We now proceed to the structural optimizations.

The first structural optimization is reducing the decoder block of U_{22}, responsible for dividing value of R_5 by three. Instead of three conditional decrement instructions, a loop decrementing three is replaced by one rule, and three other rules implement exits from this loop, depending on the remainder. One further rule acts on the register by the checker; it may be used up to 3 times in parallel.

The second structural optimization exploits the fact that registers 0, 1, 2, 3, 7 are only decremented by one instruction. The corresponding rules may be merged with the rules that follow them. However, rule $S \to S'$ is performed independently; this is solved by introducing the third phase (re-glued as described above; the phases on flowcharts are still represented by S, S' and S'' only for compactness), the move to the next state changes phase 3 into phase 1. For register 1, the rule cannot be combined with the next one, but it increments register 7 instead of the next rule.

We present the final *encoding optimization*: 3 rules $A : ITT \to JXX$, $B : JJMTT \to JJNXX$ and $C : LP \to LQ$ perform the effect of 9 rules, see Figure 3.1. This yields the system γ defined above; its flowchart is illustrated by Figure 3.2.

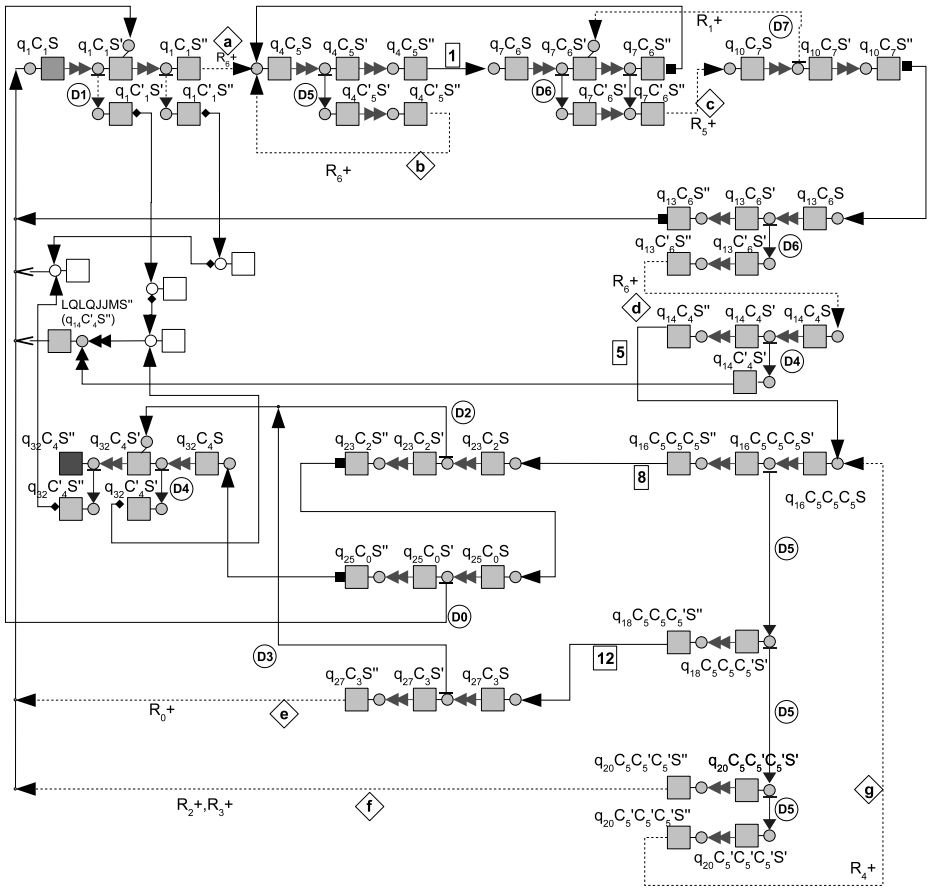


Figure 3.2: Multiset rewriting flowchart of U_{22} with glued rules

As described above, γ corresponds to a universal antiport system with 23 rules. It is still quite incredible that 23 rules are sufficient for such a simple computational model.

3.2 State of the Art

In this section we investigate the power of communication – the P systems we consider evolve by communicating objects between regions. Computational completeness can already be obtained with one membrane using antiport rules (objects are communicated in different directions) or symport rules (objects go together in the same direction) of size three, i.e., involving three objects. Applying the communication rules in the minimally parallel mode, we need two membranes to achieve computational completeness. Acceptance can even be performed by deterministic P systems with antiport or symport rules. We also consider tissue P systems, where the objects are communicated through channels between cells. Computational completeness can be obtained with a rather small number of objects and membranes or cells, in the case of tissue P systems even with copies of only one object.

Communication P systems are inspired by the idea of communicating substances through membrane channels of a cell. Molecules may go the same direction together – *symport* – or some of them may leave while at the same time other molecules enter the cell – *antiport*. Communicating objects between membrane regions is a powerful tool yielding computational

completeness with one membrane using antiport rules or symport rules of size three, i.e., involving three objects, in the maximally parallel mode. Yet even with the minimally parallel mode, we get computational completeness with two membranes. As register machines can be simulated deterministically, P systems with antiport rules or symport rules can accept any recursively enumerable set of (vectors of) natural numbers in a deterministic way.

In tissue P systems, the objects are communicated through channels between cells. In each transition step we apply only one rule for each channel, whereas at the level of the whole system we work in the maximally parallel way. Computational completeness can be obtained with a rather small number of objects and membranes or cells, in the case of tissue P systems even with copies of only one object. The computational power of P systems with antiport rules or symport rules involving copies of only one object remains as one of the most challenging open questions.

P systems with communication rules can also be used as language generators – we take the sequences of terminal objects sent out to the environment as the strings generated by the system.

We elucidate the computational power of the basic model of communication P systems, i.e., P systems using antiport and/or symport rules. Computational completeness can be obtained with rules of size (at most) 3 in only one membrane when working in the maximally parallel mode, whereas in the minimally parallel mode two membranes are needed. Using the sequential or the asynchronous mode, we only obtain Parikh sets of matrix languages.

The following simple examples show that at least when considering only one single object in one membrane, the generating power of P systems with antiport and/or symport rules differs from the accepting power:

Example 3.1 *With one object, we can only generate finite sets, i.e.,* $NO_1P_1(anti_*, sym_*) = NFIN$*: on the one hand, consider a P system* $\Pi = (\{b\}, \{b\}, E, [\ ]_1, w_1, R_1, 1)$*. Then* $N(\Pi)$ *is finite if* E *is empty, because no additional symbols can be brought in from the environment; if* $E = \{b\}$*, then by definition the rules in* R_1 *may only be of the form* (b^k, out) *or* $(b^k, out; b^m, in)$*. With symport rules* (b^k, out) *the number of objects in the skin membrane decreases, the same happens with antiport rules* $(b^k, out; b^m, in)$ *where* $k > m$*; yet these rules are the only ones which do not enforce infinite computations, as antiport rules* $(b^k, out; b^m, in)$ *where* $k \leq m$ *remain applicable as soon as the number of objects in the skin membrane is at least* k*. Hence, we conclude that in all cases* $N(\Pi) \in NFIN$*. On the other hand, any non-empty set* $M \in NFIN$ *is generated by the P system* $\Pi = (\{b\}, \{b\}, \{b\}, [\ ]_1, b^m, R_1, 1)$*, where* $m = \max(M) + 1$ *and* $R_1 = \{(b^m, out; b^j, in) \mid j \in M \setminus \{0\}\} \cup \{(b^m, out) \mid 0 \in M\}$*. The empty set is generated by the P system* $\Pi = (\{b\}, \{b\}, \{b\}, [\ ]_1, b, \{(b, out; b, in)\}, 1)$*.*

Example 3.2 *The infinite set* $\mathbb{N}$ *is accepted by the P system*

$$\Pi = (\{b\}, \{b\}, \{b\}, [\ ]_1, \lambda, \{(b, out)\}, 1).$$

Every computation for an input b^m *with* $m > 0$ *takes exactly one step in the maximally parallel mode and at most* m *steps in the other modes. Hence,* $\mathbb{N} \in N_aO_1P_1(sym_1, X)$ *for all* $X \in \{max, amin, asyn, sequ\}$*.*

With more than one membrane, we may even accept non-semilinear sets even when using copies of only one object:

Example 3.3 *The non-semilinear set* $\{2^n \mid n \in \mathbb{N}\}$ *is accepted by the P system with antiport*

$$\begin{aligned}
\Pi &= (\{b\}, \{b\}, \{b\}, [\,[\,[\,\,]_3\,]_2\,]_1, \lambda, \lambda, \lambda, R_1, R_2, R_3, 1),\\
R_1 &= \{(bb, out; b, in)\},\\
R_2 &= \{(b, in)\},\\
R_3 &= \{(bb, out), (bb, in)\}.
\end{aligned}$$

Using the rule $(bb, out; b, in)$ from R_1 the number m of objects b put into the skin membrane is divided by 2 in every maximally parallel transition step. If m has been of the form 2^n, then at the end of the computation the last symbol b enters membrane 2 by using (b, in). If this rule (b, in) was chosen or had to be used before that because of the contents of the skin membrane being an uneven number, then at the end of the computation at least two objects are in membrane two which starts an infinite computation with the rules from R_3. Hence, we conclude $\{2^n \mid n \in \mathbb{N}\} \in N_aO_1P_3(anti_2^3, sym_2)$.

3.2.1 Computational completeness

The following result shows that we only need one membrane to obtain computational completeness; in the case of accepting P systems, even deterministic systems are sufficient which contrasts the situation for catalytic P systems:

Theorem 3.2 $NO_{-1}P_1(anti_2^3) = N_aDOP_1(anti_2^3) = NRE$.

Proof. Let $M = (3, H, I, q_0, q_f)$ be a deterministic register machine. We construct a P system generating set $N_a(M)$ of numbers accepted by M:

$$\begin{aligned}
\Pi &= (O, T, O, [\,\,]_1, l_I, R_1, 1),\\
O &= \{p, p', p'', \tilde{p}, \bar{p} \mid p \in H\} \cup \{A_i \mid 1 \le i \le 3\} \cup \{l_I, l_{b_1}, b_1\},\\
T &= \{b_1\},\\
R_1 &= R_{1,I} \cup R_{1,A} \cup R_{1,S},\\
R_{1,I} &= \{(l_I, out; A_1 l_{b_1}, in),\ (l_{b_1}, out; b_1 l_I, in)\} \cup \{(l_I, out; q_0, in)\},\\
R_{1,A} &= \{(p, out; A_r q, in) \mid (p : [RrP], q, q) \in I\},\\
R_{1,S} &= \{(p, out; p'p'', in),\ (p''A_r, out; \bar{p}, in),\ (p', out; \tilde{p}, in),\\
&\quad (\tilde{p}\bar{p}, out; q, in), (\tilde{p}p'', out; s, in) \mid (p : \langle RrZM\rangle, q, s) \in I\}.
\end{aligned}$$

The contents of register r is represented by the corresponding number of symbols A_r, $1 \le r \le 3$. First, we generate an arbitrary number n of symbols b_1 and A_1 by n times applying the rules $(l_I, out; A_1 l_{b_1}, in)$ and then $(l_{b_1}, out; b_1 l_I, in)$. With applying the rule $(l_I, out; q_0, in)$ we then start the simulation of M which accepts n if and only if $n \in N_a(M)$. An add-instruction $p : (\mathtt{ADD}(r), q, q) \in I$ is simulated by using the rule $(p, out; A_r q, in)$. A subtract-instruction $p : (\mathtt{SUB}(r), q, s) \in I$ is simulated by using the rules from $R_{1,S}$ starting with applying $(p, out; p'p'', in)$: in the next step, $(p', out; \tilde{p}, in)$ is applied, and only if register r is not empty, $(p''A_r, out; \bar{p}, in)$ is applied in parallel; in the succeeding step, we either continue with $(\tilde{p}\bar{p}, out; q, in)$ for this non-empty case and with $(\tilde{p}p'', out; s, in)$ for the case that register r has been empty. When the final label q_f appears, the computation stops with the desired

output of n symbols b_1 being found in the skin membrane together with the only additional symbol q_f.

If we consider acceptance, then the input n is given by ${A_1}^n$ in the skin membrane. As the only non-determinism in the P system Π occurred in $R_{1,I}$, the P system $\Pi_a = (O_a, T_a, O_a, [\ ]_1, q_0, R_a, 1)$ with $O_a = O \setminus \{l_I, l_{b_1}, b_1\}$, $T_a = \{A_1\}$, and $R_a = R_{1,A} \cup R_{1,S}$ is deterministic and accepts $N_a(M)$. □

Adding only the very simple symport rule of size one (q_f, out) we can avoid the additional symbol q_f at the end of a computation of the P system Π constructed in the proof above, i.e., we obtain:

Corollary 3.1 $NOP_1(anti_2^3, sym_1) = NRE$.

The results established above for recursively enumerable sets of natural numbers can easily be extended to Parikh sets – we take $T = \{b_i \mid 1 \leq i \leq k\}$ and add l_{b_i}, b_i, $2 \leq i \leq k$, to O, and in $R_{1,I}$ we have to take all rules with b_i, $1 \leq i \leq k$, instead of b_1 only, i.e.,

$$R_{1,I} = \{(l_I, out; A_i l_{b_i}, in), (l_{b_i}, out; b_i l_I, in) \mid 1 \leq i \leq k\} \cup \{(l_I, out; q_0, in)\}.$$

Moreover, we then have to simulate a deterministic register machine $M = (k+2, H, I, q_0, q_f)$. In sum, we obtain:

Corollary 3.2 $PsO_{-1}P_1(anti_2^3) = Ps_aDOP_1(anti_2^3) = PsRE$.

In the results stated above we can restrict ourselves even to antiport rules of size exactly being 3. This result is already optimal with respect to the size of the rules, because with antiport rules of size 2, i.e., being of the form $(b, out; c, in)$ with b and c being single objects, the number of objects in the system cannot be changed.

We now turn our attention to P systems with only symport rules; again rules of size 3 in one membrane are sufficient to obtain computational completeness:

Theorem 3.3 $PsO_EP_1(sym_3) = Ps_aDOP_1(sym_3) = PsRE$.

Proof. Let $M = (k+2, H, I, q_0, q_f)$ be a deterministic register machine and construct the P system

$$\begin{aligned}
\Pi &= (O, T, E, [\ ]_1, w_1, R_1, 1),\\
O &= \{p, p', \tilde{p}, \tilde{p}', \tilde{p}'', \bar{p}, \bar{p}', \bar{p}'', Z_p \mid p \in H\} \cup \{A_i \mid 1 \leq i \leq k+2\}\\
&\cup \{X, l_I, l_I'\} \cup T,\\
T &= \{b_i \mid 1 \leq i \leq k\},\\
E &= O \setminus (\{X, l_I, l_I'\} \cup \{p', \tilde{p}', \bar{p}', Z_p \mid p \in H\},\\
w_1 &= \{X, l_I, l_I'\} \cup \{p', \tilde{p}', \bar{p}', Z_p \mid p \in H\},\\
R_1 &= R_{1,I} \cup R_{1,A} \cup R_{1,S},\\
R_{1,I} &= \{(l_I l_I' X, out), (q_0 l_I' X, in), (l_I X, in)\} \cup \{(l_I' A_i b_i, in) \mid 1 \leq i \leq k\},\\
R_{1,A} &= \{(pp', out), (A_r p' q, in) \mid (p : [RrP], q, q) \in I\},\\
R_{1,S} &= \{(pp', out), (p'\tilde{p}\bar{p}, in), (\bar{p}\bar{p}' A_r, out), (\bar{p}'\bar{p}'', in), (\tilde{p}\tilde{p}', out),\\
&\quad (\tilde{p}'\tilde{p}'', in), (\bar{p}''\tilde{p}'' Z_q, out), (\bar{p}\tilde{p}'' Z_s, out) \mid (p : \langle RrZM\rangle(r), q, s) \in I\}\\
&\cup \{(Z_p p, in) \mid p \in H\}.
\end{aligned}$$

Using the rules $(l_I l'_I X, out)$ as well as $(l_I X, in)$ and $(l'_I A_i b_i, in)$ from $R_{1,I}$, we are able to generate any number of symbols A_i and the same number of symbols b_i in the skin membrane. The application of the rule $(q_0 l'_I X, in)$ after $(l_I l'_I X, out)$ then starts the simulation of M. A deterministic add-instruction $p : (\texttt{ADD}(r), q, q) \in I$ is simulated by using the single copy of p' with the rules (pp', out) and $(A_r p' q, in)$. A subtract-instruction $p : (\texttt{SUB}(r), q, s) \in I$ is simulated by using the rules from $R_{1,S}$. The symbol p takes p' outside, which then returns together with $\tilde{p}, \bar{p}$; $\tilde{p}$ goes to the environment with $\tilde{p}'$ and returns transformed to $\tilde{p}''$. At the same time, $\bar{p}$ tries to decrement register r; if this is possible, it goes out together with $\bar{p}'$ and a copy of A_r and returns back as $\bar{p}''$ together with $\bar{p}'$. If $\tilde{p}''$ meets $\bar{p}''$, then the new state q has to be chosen by sending out Z_q; on the other hand, if the register has been empty, i.e., if no symbol A_r had been present, then $\bar{p}$ has remained in the skin membrane and the state s is chosen by sending out Z_s. When the final label q_f appears, the computation stops, with the garbage of symbols $(w_1 \setminus \{l_I\}) \cup \{q_f\}$ remaining in the skin membrane.

Obviously, omitting the rules from $R_{1,I}$ in the generating P system Π, we obtain the corresponding P system Π_a with $T_a = \{A_i \mid 1 \leq i \leq k\}$, which is deterministic and accepts $N_a(M)$. □

The number of garbage symbols remaining in the skin membrane at the end of a computation in the P system Π constructed in the proof above depends on the deterministic register machine to be simulated, yet using more sophisticated proof techniques, this number can be bounded by a constant (as shown in [76], this constant is at most seven; the improvements are presented in Section 3.3).

Corollary 3.3 $NO_E P_1(sym_3) = N_a DOP_1(sym_3) = NRE$.

Variants of transition

We can restrict ourselves to systems with one membrane by renaming the symbols in the different regions, yet now also taking into account the environment as region 0. In that way, we obtain a characterization of Parikh sets of matrix languages when using the sequential or the asynchronous mode:

Theorem 3.4 *For every* $X \in \{asyn, sequ\}$, $PsOP_*(anti_*, sym_*, X) =$

$$PsO_{-1}P_1(anti_2^3, X) = PsO_E P_1(sym_3, X) = PsMAT.$$

We skip the results specific for tissue P systems with symport and antiport; they can be found in [76].

In what follows we investigate the trade-off between several parameters in (tissue) P systems with antiport and symport rules, for example, between the number of membranes (cells) and the number of objects needed to obtain computational completeness.

3.2.2 Minimal antiport and minimal symport

When using antiport or symport rules, we needed rules of size three to obtain computational completeness in only one membrane (see Theorem 3.2 and Corollary 3.3). These results are already optimal for systems with only one membrane; the situation changes completely if

rules of size two, called *minimal antiport* or *minimal symport* rules, are considered – in one membrane or cell, we only get finite sets:

Theorem 3.5 $NO[t]P_1(anti_1, sym_1) \cup N[t]OP_1(sym_2) \subseteq NFIN$.

Yet with two membranes or cells, in order to get computational completeness, we may already restrict ourselves to minimal symport and/or minimal antiport rules:

Theorem 3.6 $NRE = NO[t]P_2(anti_1, sym_1) = NO[t]P_2(sym_2)$.

The proof significantly differs if tissue or tree-like P systems are considered. In the tissue case, the proof is based on the possibility to reach a membrane from another one by two roads – directly or via the environment. In this way, a temporal de-synchronization of pairs of objects is obtained and it can be used to simulate the instructions of a register machine. For the tree-like case, the result in the previous theorem only holds true if we do not require the output membrane to be elementary, otherwise it were inevitable that some object remains in the output membrane at the end of a successful computation.

Moreover, in the tissue case, we have a deterministic construction for the acceptance of recursively enumerable sets. In the tree-like case it is not possible to use a similar technique, because only the root is connected to the environment, which considerably restricts the accepting power of deterministic P systems:

Theorem 3.7 *For any deterministic P system with rules of type sym_2 and $anti_1$, the number of objects present in the initial configuration of the system cannot be increased during halting computations.*

However, if non-deterministic systems are considered, then it is possible to reach computational completeness for the accepting case with two membranes: an initial pumping phase is performed to introduce a sufficient number of working objects needed to carry out the computation (a non-deterministic guess for the number of working objects is done). After that, the system simulates a register machine thereby consuming the number of working objects. In sum, we obtain the following results:

Theorem 3.8

$$\begin{aligned} PsRE &= PsO[t]P_2(anti_1, sym_1) = PsO[t]P_2(sym_2) = Ps_aOP_2(anti_1, sym_1) \\ &= Ps_aOP_2(sym_2) = Ps_aDOtP_2(anti_1, sym_1) = Ps_aDOtP_2(sym_2). \end{aligned}$$

3.2.3 Number of symbols

Not taking care of the complexity of the rules, yet instead regarding the number of objects, the main results for P systems with antiport (and symport) rules can be summarized in the following table:

In Table 3.2, the class of P systems indicated by A generates exactly $NFIN$, the class indicated by B generates at least $NREG$, in the case of C at least $NREG$ can be generated and at least $NFIN$ can be accepted, while a class indicated by a number d can simulate any d-register machine. A box around a number indicates a known computational completeness

Table 3.2: Families NO_mP_n

	membranes					
objects	1	2	3	4	5	m
1	A	B	B	B	B	B
2	C	1	2 (U)	$\boxed{3}$	4	$m-1$
3	1	**2** (U)	$\boxed{4}$	6	8	$2m-2$
4	2 (U)	$\boxed{4}$	**6**	9	12	$3m-3$
5	$\boxed{3}$	6	9	**12**	16	$4m-4$
6	4	8	12	16	**20**	$5m-5$
s	$s-2$	$2s-4$	$3s-6$	$4s-8$	$5s-10$	$\max\{m(s-2),$ $(m-1)(s-1)\}$

bound, (U) indicates a known unpredictability bound, and a number in boldface shows the diagonal where $m(s-2)$ equals $(m-1)(s-1)$. The most interesting questions still remaining open are to characterize the families generated or accepted by P systems with only one symbol.

3.2.4 Number of rules

Another complexity parameter investigated in the literature is the number of rules in a universal P system with antiport and symport rules. Such a bound can be obtained if we simulate a universal device for which a bound on the number of rules is already known. Since P systems with antiport and symport rules can easily simulate register machines, it is natural to consider simulations of register machines having a small number of instructions. An example of such a machine is the register machine U_{32} described in [207], which has 22 instructions (9 increment and 13 decrement instructions). The table below summarizes the best results known on this topic, showing the trade-off between the number of antiport rules and their size:

Table 3.3: P systems with small numbers of antiport rules

number of rules	73	56	47	43	30	23
size of rules	3	5	6	7	11	19

3.2.5 Efficiency

If P systems with symport/antiport are additionally equipped with membrane division, then they can efficiently solve NP-complete problems. Already in [20] one constructed an $O(n)+O(\log m)$-time solution of SAT with n variables and m clauses by a uniform family of deterministic P systems with communication rules (of size at most 3 and weight at most 2) and membrane division rules (without polarization) and empty environment.

This was a development of [255], where membrane division rules were added to (tissue) P systems with symport and antiport, to solve SAT in a uniform way. The improvements were the following:

- Tissue P system was replaced by a ("usual") P system with tree-like structure;
- The P system gives the result in time which depends on the number of clauses logarithmically, not linearly;
- The computation was deterministic, not just confluent;
- The environment was empty.
- The size of communication rules was decreased from 5 to 3.
- Nothing was sent into environment except the result, just like P systems with active membranes ([253]).

The determinism can be reached due to the massive parallelism (what could happen in either order should happen simultaneously) and the system does not need resources (supply of objects) from the environment because the number of objects can grow via membrane division.

The problem about symport/antiport and membrane separation, posted in [20], has been later answered by Sevilla group, leading to a study with a different definition of separation than the original one ([80], see also [244], [79]). We do not go into details here.

We would like to make the following comments:

- The determinism heavily depends on the massive parallelism (not just in different membranes corresponding to different clauses, but also in each membrane for the same clause). Is massive parallelism of rules with respect to membranes (i.e., using for some membrane the number of rules not bounded by a constant independent of n and m) really needed in order to have a deterministic construction, or it is only needed for a construction that runs in logarithmic time with respect to the number of clauses?
- We expect that the number of starting membranes can be decreased. Is such a solution possible starting with two membranes?

We would like to mention an important observation. In the presented construction, the environment was not used, and we only had a two-level membrane structure. It is easy to see that this corresponds to a particular case of a tissue system (where all regions become cells). From here, it already follows that tissue P systems with communication rules of size at most 3 are computationally efficient. In the notation by Sevilla group, $\mathtt{SAT} \in PMC_{TDC(3)}$, and, since SAT is known to be NP-hard and the answer can be inverted by interchanging yes and no, it already follows that

$$\mathtt{NP} \cup \mathtt{co} - \mathtt{NP} \subseteq PMC_{TDC(3)}, \text{ and even} \tag{3.1}$$

$$\mathtt{NP} \cup \mathtt{co} - \mathtt{NP} \subseteq PMC_{\widehat{TDC}(3)} \tag{3.2}$$

(and, therefore, it is no longer relevant which NP-complete problem one uses to show the computational efficiency). The last result follows from the fact that we can avoid using the environment. However, (3.1) has been later reproved a number of times, for different NP-complete problems. Finally, the size of communication rules has been decreased down to 2. However, the result from [20] is still not superseded, because the communication graph there was a tree.

Conclusion The results presented in this section have elucidated the computational power of pure communication. P systems with antiport and/or symport rules are computationally complete with rules of size 3 applied in the maximally parallel mode in only one membrane or applied in the minimally parallel mode in two membranes.

Bibliographic notes

P systems with antiport and symport rules were introduced in [247] where the first results concerning computational completeness were established: $NRE = NOP_2(anti_2, sym_2) = NOP_5(anti_1, sym_2)$.

The computational completeness with one membrane independently was shown in [189] and [178] as well as in [175], where this result was obtained based on a more general model with channels through membranes. A deterministic simulation of register machines by P systems with antiport rules of weight two first was established in [180]. Tissue P systems were introduced in [230], and tissue-like P systems with channel states were investigated in [182]. P systems with minimal symport and antiport rules first were investigated in [142], where nine membranes were used to achieve computational completeness. This number was progressively decreased down and finally established to two membranes in [124]. A deterministic proof using three cells for the tissue case first was presented in [280] and improved to two cells in [78].

The descriptional complexity of P systems with respect to the number of objects first was considered in [254], where three objects were shown to be sufficient for obtaining computational completeness in four membranes, then in [61] five objects in one membrane were shown to be enough, and for tissue P systems even with only one object computational completeness was established in [177]. The main results listed in Subsection 3.2.3 were established in [67] for P systems and in [69] for tissue P systems. Based on the universal register machine U_{32} in [207], universal P systems with a small number of antiport rules were described in [157], [176], and [136].

Example 3.3 first was published in [198]. $NOtP_1(anti_1, sym_1) \subseteq NFIN$ and $NtOP_1(sym_2) \subseteq NFIN$ from Theorem 3.5 was shown in [130] and [188], respectively. Evolution-communication P systems were introduced in [149] and investigated especially in [15].

In [117], the generation of languages by P systems with minimal antiport and symport rules in two membranes was investigated.

The minimally parallel mode was introduced in [152], yet the concept was already considered in [252], p. 84, and called *minimal synchronization* there. A formal framework for P systems was developed in [185].

In [76], [77], the history as well as open problems for the computational power of P

systems with antiport and symport rules were described. The PhD thesis [5] investigates many variants of communication P systems, and a thorough survey is presented in [169].

3.3 Recent Symport Developments

Membrane systems (with symbol objects) are formal models of distributed parallel multiset processing. Symport rules move multiple objects to a neighboring region. It is known that for P systems with symport rules of weight at most 3 and a single membrane, 7 superfluous symbols are enough for computational completeness, and 1 is necessary.

We present the improvements of the lower bounds on the generative power of P systems with symport of weight bounded by 3 and 4, in particular establishing that 6 and 2 extra symbols suffice, respectively. Besides maximally parallel P systems, we also consider sequential ones. In fact, all presented non-universality lower bound results, together with all upper bound results, hold also in this case, yielding the current state-of-the-art.

Membrane systems (with symbol objects) are formal models of distributed parallel multiset processing. Symport rules move predefined groups objects to a neighboring region [247]. In maximally parallel mode (typical for membrane computing), this alone is sufficient to construct a computationally universal device, as long as the environment may contain an unbounded supply of some objects. The number of symbols specified in a symport rule is called its weight. The result of a computation is the total number of objects when the system halts. In some cases, however, for technical reasons the desired result may only be obtained alongside a small number of superfluous objects in the output region.

There were multiple papers improving the results on P systems with symport/antiport of small weight (an antiport rule moves objects between 2 regions in both directions, and its weight is the maximum of objects per direction), see [169] for a survey of results. Computational completeness is achieved even for minimal cooperation: either symport/antiport of weight 1, or symport of weight at most 2. This holds for 2 membranes, without superfluous objects if the output is considered in the skin, or with 1 superfluous object under the classical assumption of the output in the elementary membrane. In the tissue case, the accepting systems can even be made deterministic.

With cooperation of up to 3 objects, a single membrane suffices. The regions are called the skin and the environment, the latter contains an unbounded supply of some objects, while the contents of the former is always finite. With antiport-2/1 alone (i.e., exchanging 1 object against 2), the computational completeness is obtained with a single superfluous object. With symport-3 (i.e., symport rules only, of weight up to 3), one proved in [188] that 13 extra objects suffice for computational completeness. This result has been improved in [76] to 7 superfluous symbols. In the same paper it was shown that without any superfluous symbols such systems only generate finite sets. Although one-membrane symport-only systems are, in principle, universal, their exact characterization remains open, and narrowing the gap between 7 objects and 1 object presents an interesting combinatorics-style problem.

The computation of a P system consists of multiple, sometimes simultaneous, actions of two types: move objects from the skin to the environment, and move objects from the environment into the skin. It is obvious that trying to move all objects out in the environment

will activate the rules of the second type. Since, clearly, rules of the first type alone cannot generate more than finite sets, it immediately follows that the "garbage" is unavoidable. In [123] one obtains some partial results on the power of one-membrane systems with symport-3, concerning intermediate number of extra objects, both for maximally parallel and for sequential mode; we start it with a simple result for unbounded symport from [119]. We also recall the recent improvements from [120]: 6 extra objects are enough for symport of weight at most 3, and 2 objects are enough for symport of weight at most 4.

It is known that the power of one-membrane P systems with symport of weight at most 2 is quite limited:

$$\begin{aligned} NOP_1(sym_2) &\subseteq NFIN, [188] \\ NOP_1(sym_2) &\supseteq SEG_1 \cup SEG_2, [118] \end{aligned}$$

It is not difficult to see that the proofs of both bounds remain valid also for the sequential case, i.e., for $NOP_1^{sequ}(sym_2)$. We turn to symport with higher bounds on weight, e.g., unbounded, at most 3, and at most 4, starting from finite and regular classes, and then proceeding with the universality results.

3.3.1 Unbounded weight

We claim a simple new result: P systems with a single membrane and unbounded symport generate all finite sets of numbers without superfluous objects. Indeed, for an arbitrary non-empty finite set M of non-negative integers, consider the following system:

$$\begin{aligned} \Pi_\infty &= (O = \{s, a\}, E = \emptyset, \mu = [\ ]_1, w = sa^{\max(M)}, R), \\ R &= \{(sa^{\max(M)-x}, out) \mid x \in M\}. \end{aligned}$$

Clearly, all computations halt in one step, choosing an element in M and sending out everything *except* that many copies of a inside. The empty set case is shown in the next subsection.

Paying the price of one superfluous object, we can extend finite sets to regular sets of numbers. We now proceed by constructing a P system generating the length set of a language accepted by a finite automaton $A = (Q, \Sigma, \delta, q_0, F)$, where $Q = \{q_j \mid 0 \leq j \leq m\}$; without loss of generality we assume A satisfies the following property: there is at least one transition from every non-final state.

$$\begin{aligned} \Pi'_A &= (O = Q \cup Q' \cup \Sigma, E = Q' \cup \Sigma, \mu = [\ ]_1, w = q_0 \cdots q_m q'_0 M, R), \\ Q' &= \{q' \mid q \in Q\}, \\ R &= \{(q_j q'_j M, out) \mid 0 \leq j \leq m\} \cup \{(q_0 \cdots q_m q'_j, out) \mid q_j \in F\} \\ &\cup \{(q_j a q'_k M, in) \mid q_k \in \delta(q_j, a),\ a \in \Sigma\}. \end{aligned}$$

The computation of a finite automaton A is simulated, using object M in all rules. If the current state is accepting, then all state objects, primed and unprimed, may be sent out, and the computation halts with M and a correct number of objects a.

Based on the constructions in this subsection, $NOP_1(sym_*) \supseteq NFIN \cup N_1REG$. The constructions, however, used arbitrarily large symport rules. In the constructions in the rest of the subsection, the weight of symport rules is bounded by 3 or 4.

3.3.2 Few-element sets

We start with a simple system: $\Pi_0 = (O = \{a\}, E = \emptyset, \mu = [\ \ ]_1, w = a, R = \{(a, in), (a, out)\})$. System Π_0 perpetually moves a single object in and out, effectively generating the emptyset. For any $x \in \mathbb{N}$, setting $R = \emptyset$ and $w = a^x$ will lead to a system Π_1 which immediately halts, generating a singleton $\{x\}$.

We now proceed to arbitrary small-cardinality sets. To generate a multi-element set, the system must make at least one non-deterministic choice. Since we want to allow the difference between the elements to be arbitrarily large, such choice must be persistent, i.e., the decision information should not vanish, at least until multiple objects are moved accordingly. For any numbers $y > x$, consider the following P system:

$$\begin{aligned} \Pi_2 &= (O = \{a, b, i, p, q\}, E = \{q\}, \mu = [\ \ ]_1, w = a^x b^{y-x+1} ip, R), \\ R &= \{(i, out), (ip, out), (pq, in), (pqb, out)\}. \end{aligned}$$

There are two possible computations of Π_2: either i exits alone, halting with $a^x b^{y-x+1} p$, generating $y + 2$, or i exits with p, leading to a sequence of applications of the last two rules until no objects b remain in the skin, halting with $a^x pq$, generating $x + 2$. Therefore, Π_2 generates an arbitrary 2-element set with 2 extra objects.

This construction can be improved to generate higher-cardinality sets as follows. Let $m \geq 2$; for arbitrary $m + 1$ distinct numbers denote the largest one by y and the others by x_j, $1 \leq j \leq m$. We construct the following P system

$$\begin{aligned} \Pi_{m+1} &= (O, E = \{q_j \mid 1 \leq j \leq m\}, \mu = [\ \ ]_1, w, R), \\ O &= \{a_j \mid 1 \leq j \leq y + 1\} \cup \{i\} \cup \{p_j, q_j \mid 1 \leq j \leq m\}, \\ w &= i a_1 \cdots a_{y+1} p_1 \cdots p_m, \\ R &= \{(i, out)\} \cup \{(ip_j, out), (p_j q_j, in) \mid 1 \leq j \leq m\} \\ &\cup\ (p_j q_j a_k, out) \mid 1 \leq j \leq m,\ x_j + 1 \leq k \leq y + 1, j \neq k\}. \end{aligned}$$

Such system behaves like Π_2, except it also chooses among different objects p_j to send out symbols a_k for $k > x_j$. It halts either with $a_1 \cdots a_{y+1} p_1 \cdots p_m$ generating $y + m + 1$, or with $a_1 \cdots a_{x_j} q_j p_1 \cdots p_m$ generating $x_j + m + 1$. For $m = 2, 3, 4$ this leads to Π_3 generating $\{x_1 + 3, x_2 + 3, y + 3\}$, Π_4 generating $\{x_1 + 4, x_2 + 4, x_3 + 4, y + 4\}$, and Π_5 generating $\{x_1 + 5, x_2 + 5, x_3 + 5, x_4 + 5, y + 5\}$, i.e., any 3-, 4- or 5-element set with 3, 4 or 5 extra objects, respectively.

3.3.3 Straightforward regularity

We now proceed by constructing a P system generating the length set of a language accepted by a finite automaton $A = (Q, \Sigma, \delta, q_0, F)$, where $Q = \{q_j \mid 0 \leq j \leq m\}$; without loss of generality we assume A satisfies the following property: there is at least one transition from every non-final state.

$$\begin{aligned} \Pi_A &= (O = Q \cup Q' \cup \Sigma, E = Q' \cup \Sigma, \mu = [\ \ ]_1, w = q_0 \cdots q_m q'_0, R), \\ Q' &= \{q' \mid q \in Q\}, \\ R &= \{(q_j q'_j, out) \mid 0 \leq j \leq m\} \\ &\cup\ \{(q_j a q'_k, in) \mid q_k \in \delta(q_j, a),\ a \in \Sigma\} \cup \{(q'_j, out) \mid q_j \in F\}. \end{aligned}$$

Unfortunately, besides the needed number, the skin region at halting also contains the superfluous symbols, as many as there are states in A. Therefore, we have obtained all sets N_kREG_k.

The simplest examples of application of Π_A are the set of all positive numbers and the set of all positive even numbers.

Therefore, $NOP_1(sym_3)$ contains $NFIN_1 \cup \bigcup_{k=0}^{\infty}(N_kFIN_k \cup N_kREG_k)$.

3.3.4 Improved universality

We now revisit the symport-3 construction from [76]. The 7 extra objects were denoted $l_h, b, d, x_1, x_4, x_5, x_6$. In [120] one showed that all regular number sets can be generated with 6 extra objects. This result has been recently superseded by generating all recursively enumerable number sets with 6 extra objects, [120], so we recall the best known result:

Theorem 3.9 $NOP_1(sym_3) \supseteq N_6RE$.

Proof. Consider an arbitrary counter automaton M. We first transform it as follows: for each counter i, a conflicting counter $\bar{i}$ is introduced, initially containing value zero. The semantics of a counter machines is modified such that whenever counters i and $\bar{i}$ are non-zero, the computation is aborted without producing a result.

Then, all zero-test instructions for any counter i are performed by incrementing a conflicting counter $\bar{i}$, and then decrementing it. The counter automaton $M' = (Q, q_0, q_f, P, C)$ under the conflicting counter semantics is equivalent to the counter automaton M.

We construct a P system simulating counter automaton M':

$$
\begin{aligned}
\Pi &= (O, E, [\ ]_1, w, R, 1), \text{ where } O = E \cup alph(w),\\
E &= Q \setminus \{q_f\} \cup \{x_2, x_4, \#\} \cup \{a_i \mid i \in C\} \cup \{p_2 \mid p \in P\},\\
O' &= \{p_i \mid i \in \{1,3\}, p \in P\} \cup \{A_i \mid i \in C\},\\
w &= q_f o q_0 x_1 x_3 x_5 d s, \text{ where } s \text{ represents } O',\\
R &= \{1 : (x_1x_2, in), 2 : (x_2x_1x_3, out), 3 : (x_2d, out),\\
&\quad 4 : (x_3x_4, in), 5 : (x_4x_3x_5, out), 6 : (x_4d, out)\}\\
&\cup \{7 : (A_ia_ix_5, in), 8 : (a_ia_{\bar{i}}d, out) \mid i \in C\}\\
&\cup \{9 : (qp_1x_1, out), 10 : (p_1p_2x_1, in), 11 : (p_2p_3A_i, out),\\
&\quad 12 : (p_3q'x_3, in), 13 : (p_3\#q_f, in) \mid p : (q \to q', i+) \in P\}\\
&\cup \{14 : (qp_1x_3, out), 15 : (p_1p_2x_3, in), 16 : (p_2p_3a_i, out), 17 : (p_2p_3d, out),\\
&\quad 18 : (p_3q'x_5, in), 19 : (p_3\#q_f, in) \mid p : (q \to q', i-) \in P\}\\
&\cup \{20 : (q_fbx, out) \mid x \in O'\} \cup \{21 : (q_fb, in),\\
&\quad 22 : (\#d, out), 23 : (\#d, in), 24 : (oq_f, out), 25 : (od, out)\}.
\end{aligned}
$$

We now explain the "correct" work of Π. If non-determinism allows to apply a different multiset of rules, then one of rules from the group $T = \{3, 6, 8, 13, 17, 19, 25\}$ is also applied, and then the computation applied rules 22, 23 forever, without producing the result. Notice that even if multiple applications of rules from T happen, this would only add further

objects # to the skin, and the computation would still be unable to halt. In the "correct" computations of Π, rules from T are not applied, their role is only to ensure that symbols x_2, x_4, p_2, o or pair $(a_i, a_{\bar{i}})$ are never idle in the skin, as well as that symbols p_3 are never idle in the environment.

Rule 24 is applied in the beginning of the computation, in parallel with simulation of the first instruction of M'. If rule 21 is applied instead, then rule 25 forces an infinite computation. Throughout the simulation of M', object q_f stays in the environment (available in a single copy unlike other objects from Q).

Increment is performed by a sequence of multisets of rules 9, 1, 2, (10,4), (5,11), (7,12). We remark that the system seems to have a choice between rules 1 and 10. However, if rule 10 is applied immediately after rule 9, then rule 11 is applied, followed by rule 13, forcing a non-ending computation. Yet, if rule 1 is applied again immediately after rule 2, then rule 2 is no longer applicable, forcing rule 3 and a non-ending computation.

Decrement is performed by a sequence of multisets of rules 14,4,5,15,16,18. We remark that the system seems to have a choice between rules 4 and 15. However, if rule 15 is applied immediately after rule 14, then rule 16 or 17 is applied, followed by rule 19, forcing a non-ending computation. Yet, if rule 4 is applied again immediately after rule 5, then rule 5 is no longer applicable, forcing rule 6 and a non-ending computation. If decrement is attempted on a counter with a zero value, then rule 17 is applied instead of rule 16, forcing an infinite computation.

The conflicting counter semantics is ensured by rule 8.

Notice that once the simulation of M' arrives to q_f, symbols x_1, x_3, x_5, q_f stay in the skin, while symbols from $Q \setminus \{q_f\}$ stay in the environment. Hence, all the rules moving objects p_1, p_3 or A_i in, i.e., rules 7,10,12,15,18 could not be applied even if objects from O' are sent out, which is carried out by rules 20,21. Rules 13,19 temporary become applicable, but lead to an infinite computation; they are no longer applicable once all object from O' are taken out, and q_f stays in the skin. The computation halts with the skin containing the result (the desired number of copies of a_1) as well as 6 extra objects: x_1, x_3, x_5, d, q_f, b. □

3.3.5 Symport of weight at most 4

We now recall another improvement from [120]. If we allow up to 4 objects to participate in symport rules, then the number of extra objects can be decreased to 2.

Theorem 3.10 $NOP_1(sym_4) \supseteq N_2RE$.

Proof. Consider an arbitrary counter automaton M. We first transform it as follows: for each counter i, a conflicting counter $\bar{i}$ is introduced, initially containing value zero. The semantics of a counter machines is modified such that whenever counters i and $\bar{i}$ are non-zero, both are instantly decremented.

Then, all zero-test instructions for any counter i are performed by incrementing a conflicting counter $\bar{i}$, and then decrementing it. (nothing changes except the states if counter i has value zero. Otherwise, both counters are decremented, and then the decrement of counter $\bar{i}$ fails. Therefore, we have transformed M into a counter automaton $M' = (Q, q_0, q_f, P, C)$, which is equivalent under the conflicting counter semantics.

We construct a P system simulating a counter automaton M':

$$
\begin{aligned}
\Pi &= (O, E, [\ ]_1, w, R, 1), \text{ where} \\
O &= E \cup O' \cup \{T, N\}, \\
E &= \{a_i \mid i \in C\} \cup Q \cup \{p_2 \mid p \in P\}, \\
O' &= \{p_i \mid i \in \{1,3,4\}, p \in P\}, \\
w &= q_0\ T\ s\ c^{|O'|-1}\ N, \text{ where } s \text{ represents } O', \\
R &= \{1 : (qp_1T, out), 2 : (p_1q'a_iT, in) \mid p : (q \to q', i+) \in P\} \\
&\cup\ \{3 : (qp_1T, out), 4 : (p_1p_2T, in), 5 : (p_2p_3a_iT, out), 6 : (p_2p_4T, out), \\
&\quad\ 7 : (p_2p_4T, in), 8 : (p_3q'T, in) \mid p : (q \to q', i-) \in P\} \\
&\cup\ \{9 : (a_ia_{\bar{i}}, out) \mid i \in C\} \\
&\cup\ \{10 : (q_fbx, out \mid x \in O')\} \cup \{11 : (Nc, out), 12 : (q_fbN, in).
\end{aligned}
$$

The simulation of a transition in A is performed as follows:

- An increment instruction is performed by replacing q by q' in two steps with the help of object p_1, also brining an object a_i in.
- A decrement instruction is performed by replacing q by q' in four steps with the help of objects p_1, p_2, p_3, also brining an object a_i out. Moreover, rules 6,7 are optionally applied an arbitrary number of steps. If a_i is not present, then the computation never exits the loop of applying rules 6,7.
- The conflicting counter semantics is implemented by rule 9.
- Once the simulation of M is finished, object q_f enters the system. Each application of the group of rules 11,10,12 leads to removal from the skin of one copy of c and one object from O'. Notice that objects from O' cannot reenter the skin without object T. The first application of rule 11 actually happens in the first step of the computation, but the rest of this process takes place after the simulation of M' is finished. The multiplicity of objects c has been chosen to be $|O'| - 1$ so that objects q_f, b stop reentering the skin exactly when no more objects from O' remain there. The computation halts with the skin containing the result and objects T, N. □

Hence, the known bounds can be described as follows.

$$
\begin{aligned}
NOP_1(sym_2) &\supseteq SEG_1 \cup SEG_2. && (3.3)\\
NOP_1(sym_3) &\supseteq NFIN_1 \cup \bigcup_{k=0}^{5}(N_kFIN_k \cup N_kREG_k) \cup N_6RE. && (3.4)\\
NOP_1(sym_4) &\supseteq NFIN_0 \cup NFIN_1 \cup N_1REG_1 \cup N_2RE. && (3.5)\\
NOP_1(sym_*) &\supseteq NFIN \cup N_1REG \cup N_2RE. && (3.6)\\
\hline
NOP_1(sym_2) &\subseteq NFIN. && (3.7)\\
NOP_1(sym_*) &\subseteq NFIN \cup N_1RE. && (3.8)
\end{aligned}
$$

3.3.6 Sequential mode

Also for the sequential case, at least one superfluous object is unavoidable for generating any infinite set. The argument in the end of Introduction stays valid.

It is also not difficult to see that sequential P systems with symport cannot generate more than regular sets of numbers, because their translation in sequential multiset rewriting systems is straightforward, and the behavior of the latter is known to characterize matrix grammars. The claim follows from $NMAT = NREG$. All presented above in this section yields $NOP_1(sym_*) \subseteq NFIN \cup N_1REG$. As we explain later, this bound is exact.

An important observation is that all non-universality results and all upper bounds also hold for sequential systems. Indeed, in the constructions for (3.3) all rules are of type *out*, and the halting computations are just one step long; this is equivalent to sequential exhausting the contents of the skin by the same rules. Results in (3.7) and (3.8) essentially deal with the halting condition, and also hold. The subfiniteness and subregulatity results in (3.4) and (3.5) are based on the constructions that do not use parallelism, also for arbitrary values of k. Similarly, parallelism is not used in unbounded symport constructions for (3.6). The following relationships thus hold.

$$NFIN \supseteq NOP_1^{sequ}(sym_2) \supseteq SEG_1 \cup SEG_2. \tag{3.9}$$

$$NOP_1^{sequ}(sym_3) \supseteq NFIN_1 \cup \bigcup_{k=0}^{\infty}(N_kFIN_k \cup N_kREG_k) \tag{3.10}$$

$$NOP_1^{sequ}(sym_*) = NFIN \cup N_1REG. \tag{3.11}$$

Discussion We have improved the best known lower bounds of the computational power of one-membrane P systems with symport only. Using symport of weight at most 3, computational completeness holds with 6 extra objects. With symport of weight at most 4, 2 extra objects suffice. Notice that all finite sets can be generated if the weight of symport is not restricted. Since 1 extra object is known to be necessary to generate infinite sets by any symport-only P system, a particularly interesting open question is whether

- one extra object is also sufficient for computational completeness (and what symport weight bound is enough), or
- two extra objects are also necessary for computational completeness.

Another problem is to bridge or further (see (3.4)) decrease the gap between 6 and 1 extra objects for symport of weight at most 3.

For sequential mode, it is particularly interesting whether infinitely many additional objects are unavoidable for generation of finite or regular number sets (or if N_kFIN/N_kREG belong to $NOP_1^{sequ}(sym_s)$ for some $k, s \in \mathbb{N}$).

3.4 Conclusions to Chapter 3

A quite remarkable result has been described: a strongly universal symport/antiport P system with only 23 rules.

A general overview of the symport/antiport area has been given, containing multiple results by the author. (Overall, this chapter is shorter than the other chapters with results, because we do not zoom on the results already covered in the author's Ph.D. thesis [5]).

The improved known lower bounds of the computational power of one-membrane P systems with symport have been shown. Using symport of weight at most 3, computational completeness holds with 6 extra objects. With symport of weight at most 4, 2 extra objects suffice. Notice that all finite sets can be generated with symport of unrestricted weight.

Since 1 extra object is known to be necessary to generate infinite sets by any symport-only P system, a particularly interesting open question is whether one extra object is also sufficient for computational completeness (and what symport weight bound is enough), **or** two extra objects are also necessary for computational completeness. Another problem is to bridge or further (see (3.4)) decrease the gap between 6 and 1 extra objects for symport of weight at most 3.

For sequential mode, it is particularly interesting whether infinitely many additional objects are unavoidable for generation of finite or regular number sets (or if N_kFIN/N_kREG belong to $NOP_1^{sequ}(sym_s)$ for some $k, s \in \mathbb{N}$).

Section 3.1 is based on publications [136], [137], [138] and [268]. Section 3.2 is mainly based on publications [169], [121], [122], [128], [117], [124] and [125], and contains extensive bibliographical notes. Section 3.3 is based on publications [119], [120] and [123].

4. ACTIVE MEMBRANES. ENERGY

Like in Chapter 3, we consider the models where rules are associated to membranes (except Section 4.8). However, instead of direct cooperation between objects, we now use cooperation between one object and the information (called polarization or energy) stored in a membrane.

P systems with active membranes are parallel computation devices inspired by the internal working of biological cells. Their main features are a hierarchy of nested membranes, partitioning the cell into regions, and *multisets* of symbol-objects describing the chemical environment. The system evolves by applying rules such as non-cooperative multiset rewriting (i.e., objects are individually rewritten), communication rules that move the objects between adjacent regions, and membrane division rules that increase the number of membranes in the system. The membranes also possess an *electrical charge* that works as a local state, regulating the set of rules applicable during each computation step. The rules, in turn, may change the charge of the membrane where they take place.

In order to solve computational problems one usually employs *polynomial-time uniform families* of P systems with active membranes, consisting of a P system Π_n for each input length n (as for Boolean circuits) and a single Turing machine constructing Π_n from n in polynomial time. The actual input is then encoded as a multiset of objects, and placed inside an input membrane of Π_n. The space required by a family of P systems (in terms of number of membranes and objects) for solving a decision problem can then be analyzed as a function of n. It is already known that polynomial-space P systems and polynomial-space Turing machines are equal in computing power, but the proof of this result does not generalize to larger space bounds. In Section 4.1 we show the key ideas needed in order to prove the exponential-space analogue of that result by directly simulating deterministic exponential-space Turing machines using P systems.

In Section 4.2 we present the improvements, by using register machines, of some existing universality results for specific models of P systems. It is known that P systems with active membranes without polarization generate $PsRE$, working with an unbounded number of membranes. We show that they generate all recursively enumerable *languages*; *4 starting membranes* with 3 labels or 7 starting membranes with *2 labels* are sufficient.

In Section 4.3 we present an algorithm for deterministically deciding `SAT` in linear time by P systems with active membranes using only two polarizations and rules of types (a), (c), and (e). Moreover, various restrictions on the general form of the rules are considered. Several problems related to different combinations of these restrictions are also formulated.

Due to massive parallelism and exponential space in membrane systems, some intractable computational problems can be solved by P systems with active membranes in polynomial number of steps. In Section 4.4 we present a generalization of this approach from decisional

problems to the computational ones, by providing a solution of a #**P**-complete problem, namely to compute the permanent of a binary matrix. The implication of this result to the **PP** complexity class is discussed and compared to the known result of $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP}$.

It is known that the satisfiability problem (SAT) can be solved with a semi-uniform family of deterministic polarizationless P systems with active membranes with non–elementary membrane division. In Section 4.5 we present an improvement of this result by showing that the satisfiability of a *quantified* Boolean formula (QSAT) can be solved by a *uniform* family of P systems of the same kind.

In Section 4.6 we study P systems with active membranes without non-elementary membrane division, in minimally parallel way. The main question we address is the number of polarizations enough for an efficient computation depending on the types of rules used. In particular, we show that it is enough to have four polarizations, sequential evolution rules changing polarizations, polarizationless non-elementary membrane division rules and polarizationless rules of sending an object out. The same problem is solved with the standard evolution, sending an object out and polarizationless non-elementary membrane division, with six polarizations. It is open whether these numbers are optimal.

Section 4.7 considers a different, but related model: the number of states of membranes is now numeric and infinite; however, the rules can only increase the state and decrease it (provided it stays non-negative). In this way, we recall a variant of membrane systems introduced in [62], where the rules are directly assigned to membranes and, moreover, every membrane carries an energy value that can be changed during a computation by objects passing through the membrane. The result of a successful computation is considered to be the distribution of energy values carried by the membranes.

We show that for systems working in the sequential mode with a kind of priority relation on the rules we already obtain universal computational power. When omitting the priority relation, we obtain a characterization of the family of Parikh sets of languages generated by context-free matrix grammars. On the other hand, when using the maximally parallel mode, we do not need a priority relation to obtain computational completeness. Finally, we introduce the corresponding model of tissue P systems with energy assigned to the membrane of each cell and objects moving from one cell to another one in the environment as well as being able to change the energy of a cell when entering or leaving the cell. In each derivation step, only one object may pass through the membrane of each cell. When using priorities on the rules in the sequential mode (where in each derivation step only one cell is affected) as well as without priorities in the maximally parallel mode (where in each derivation step all cells possible are affected) we again obtain computational completeness, whereas without priorities on the rules in the sequential mode we only get a characterization of the family of Parikh sets of languages generated by context-free matrix grammars.

In Section 4.8 we consider the energy contained in regions, not membranes. Moreover, a conservation law is imposed on the rules of the system. In this way, we investigate the computational power of energy-based P systems, a model of membrane systems where a fixed amount of energy is associated with each object and the rules transform single objects by adding or removing energy from them. We answer recently proposed open questions about the power of such systems without priorities associated to the rules, for both sequential and

maximally parallel modes. We also conjecture that deterministic energy-based P systems are not computationally complete.

4.1 Simulating Turing Machines

P systems with active membranes [253] are parallel computation devices inspired by the internal working of biological cells. Their main features are a hierarchy of nested membranes, partitioning the cell into regions, and *multisets* of symbol-objects describing the chemical environment. The system evolves by applying rules such as non-cooperative multiset rewriting (i.e., objects are individually rewritten), communication rules that move the objects between adjacent regions, and membrane division rules that increase the number of membranes in the system. The membranes also possess an *electrical charge* that works as a local state, regulating the set of rules applicable during each computation step. The rules, in turn, may change the charge of the membrane where they take place.

In order to solve computational problems one usually employs *polynomial-time uniform families* of P systems with active membranes, consisting of a P system Π_n for each input length n (as for Boolean circuits) and a single Turing machine constructing Π_n from n in polynomial time. The actual input is then encoded as a multiset of objects, and placed inside an input membrane of Π_n. The space required by a family of P systems (in terms of number of membranes and objects) for solving a decision problem can then be analyzed as a function of n. It is already known that polynomial-space P systems and polynomial-space Turing machines are equal in computing power [263], but the proof of this result does not generalize to larger space bounds. In this section we show the key ideas needed in order to prove the exponential-space analogue of that result by directly simulating deterministic exponential-space Turing machines using P systems. For the full technical details of the results presented here we refer the reader to the paper [93].

Simulating Turing machines We describe how deterministic Turing machines working in exponential space can be simulated by P systems by means of an example. Let M be a Turing machine processing an input x of length $n = 2$ and requiring $2^n = 4$ auxiliary tape cells (the total length of the tape is then 6); assume that the alphabet of M consists of the symbols a and b. Suppose that the current configuration C of M is the one depicted on the left of Figure 4.1, and that the transition it performs leads it to the configuration C' on the right. In the figure, the tape cells of M are identified by a binary index.

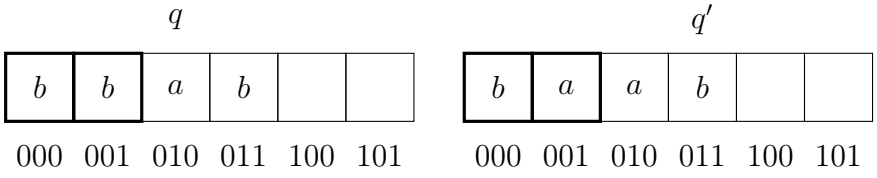


Figure 4.1: A transition of a Turing machine

We encode the configuration C of M as the following configuration of the P system Π simulating it, as shown on Figure 4.2:

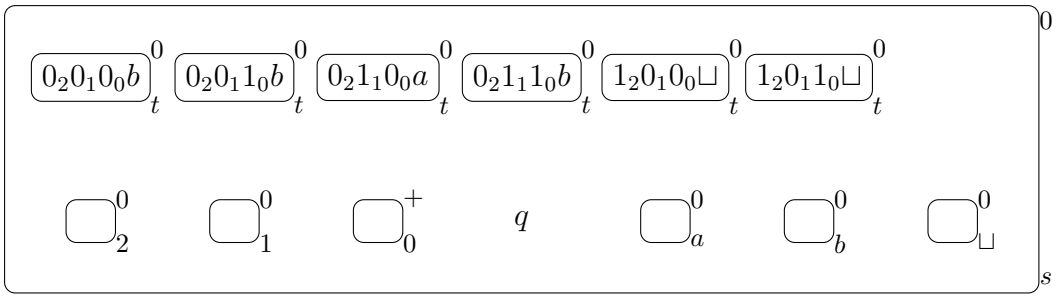


Figure 4.2: Representation of a TM by a P system

In this picture, the label of a membrane is indicated at its lower-right corner, while its electrical charge (+, 0, or −) is at its upper-right corner. The symbols located inside the membranes represent the objects in the configuration of Π. The P system, beside its external membrane s, possesses 6 membranes labeled by t (called the tape-membranes) corresponding to the tape cells of M; each tape-membrane contains 3 subscripted bit-objects encoding the index of the corresponding tape cell (the subscript represents the position of the bit in the index; for instance, the presence of bit-object 1_0 indicates that 1 is the least significant bit). Furthermore, each tape-membrane contains an object representing the symbol written in the corresponding tape cell of M, where $\sqcup$ represents a blank cell. Only one tape membrane is part of the initial configuration of Π, as it can be at most polynomial in size; the other ones are created by membrane division before simulating the first step of M.

A *state-object* q represents the current state of M; this object will also regulate the simulation of the next step of M. The position of the tape head is encoded in binary as the electrical charges of the membranes 0, 1, 2 (the *position-membranes*); the label of each membrane represents the position of the corresponding bit, while its charge the value of the bit: a neutral charge represents a 0, and a positive charge a 1. In the example above, the charges of membranes 2; 1; 0 are 0; 0;+, encoding the binary number 001 (decimal 1). Finally, the auxiliary membranes labeled by a, b, $\sqcup$ (the *symbol-membranes*) in the lower-right corner correspond to the tape symbols of M, and are used in order to read the symbol on the current tape cell. Figure 4.3 shows how the next step of M is simulated.

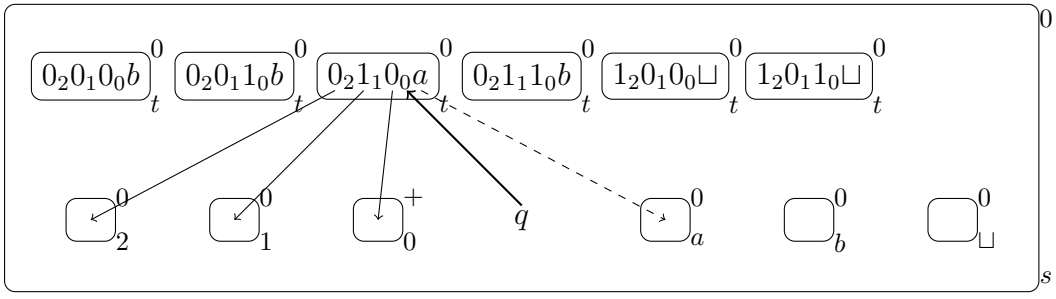


Figure 4.3: Beginning of simulation of a step of a TM by a P system

First, the object q non-deterministically guesses a tape-membrane t (all such membranes are indistinguishable from the outside) and enters it (thick arrow in the picture) while changing the charge to positive. The change of charge enables the bit-objects inside it to move to the corresponding position-membranes (along the thin arrows), where their values are com-

pared to the charges of the membranes; this allows us to check whether the tape-membrane we guessed is indeed the one under the tape head of M. In the meantime, object a is sent to the corresponding symbol-membrane (dashed arrow) to change the charge to positive.

Since in the example the tape-membrane that was chosen is not the correct one, an error-object is produced by one of the mismatched position-bits, and the configuration of Π is restored to the initial one, with the following exception: the charge of the tape-membrane is set to *negative*, so it will not be chosen again. The P system then proceeds by guessing another tape-membrane among the remaining (neutrally charged) ones. After a number of wrong guesses, the configuration of Π will be similar to the one shown on Figure 4.4.

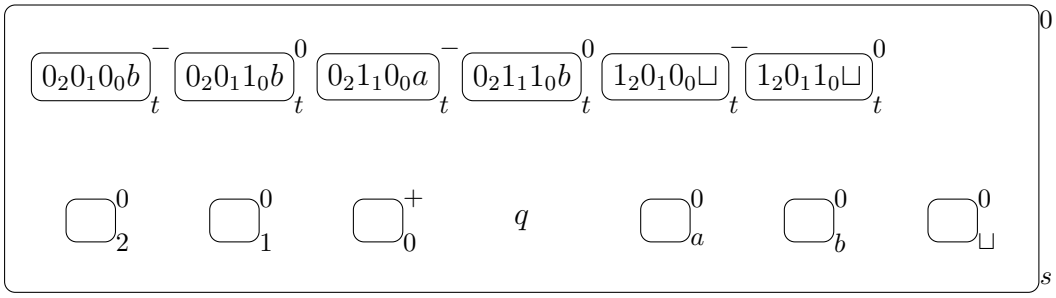


Figure 4.4: After some "wrong" guesses

When the tape-membrane corresponding to the current cell of M is finally guessed, we can perform the actual simulation of the computation step (updating the position of the head, the symbol on the tape, and the state of M). The state-object may first read the tape symbol by looking at the only positively charged symbol-membrane; it can then update the charges of the position-membranes (from the least to the most significant bit) in order to increment or decrement the binary number they encode, produce the new tape symbol (b in the example) and finally rewrite itself as the new state of M (q' in the example). The charges of all tape- and symbol-membranes are also reset to neutral by using auxiliary objects. The configuration of Π corresponding to the new configuration of M thus becomes the one shown in Figure 4.5:

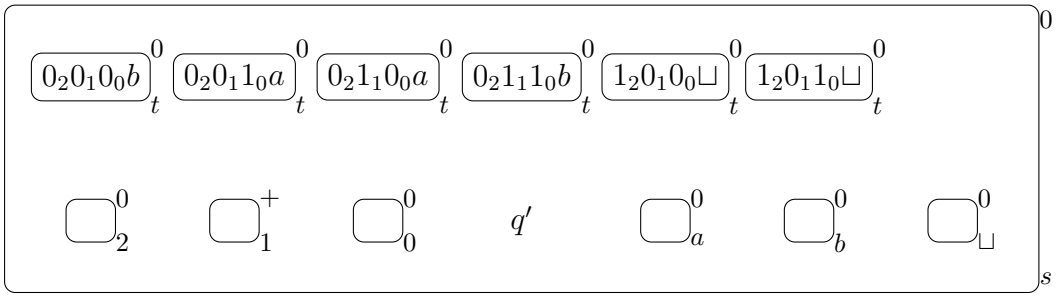


Figure 4.5: Finishing the simulation of a TM

Now the P system continues simulating the next steps of M, until an accepting (resp., rejecting) state is reached; when this happens, the P system produces a `YES` (resp., `NO`) object that is sent out from the outermost membrane as the result of the computation.

Conclusions and open problems The simulation described above can be carried out by a polynomial time uniform family of P systems with active membranes operating in space $O(s(n) \log s(n))$, where $s(n)$ is the space required by the simulated Turing machine on inputs of length n. Since an analogous result holds in the opposite direction ([263], Theorem 5), the two classes of devices solve exactly the same decision problems when working withing an exponential space limit. The techniques employed here do not carry over to the simulation of superexponential space Turing machines, since they would require a super-polynomial number of subscripted objects in order to encode tape positions; this amount of objects (and their associated rules) cannot be constructed using a polynomial-time uniformity condition. Novel techniques will be probably needed in order to prove that the equivalence of Turing machines and P systems also holds for larger space bounds.

4.2 Universality

We present the improvements, by using register machines, of some existing universality results for specific models of P systems. It is known from [17] that P systems with active membranes without polarization generate $PsRE$, working with an unbounded number of membranes. We show that they generate all recursively enumerable *languages*; *4 starting membranes* with 3 labels or 7 starting membranes with *2 labels* are sufficient.

As shown in [72], [71], P systems with 2 polarizations and rules of types (a) – rewriting – and (c) – sending an object out – generate $PsRE$ using 2 membranes or accept $PsRE$ using 1 membrane. In this section we will show that *deterministic* systems of this kind with 1 membrane accept $PsRE$. Figure 4.6 depicts the membrane structures of the P systems constructed in the succeeding proofs.

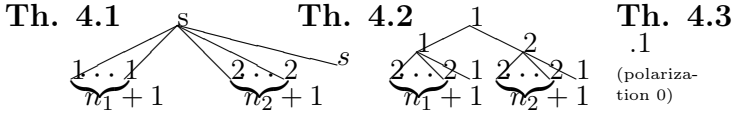


Figure 4.6: Membrane structures for active membrane proofs

4.2.1 One polarization

The theorem below provides a result, similar to that of Theorem 2.29, for P systems with active membranes with only one polarization (one usually calls them without polarizations). The construction gives no upper bound on the number of objects present inside the system in general, but during any halting computation the number of objects never exceeds 3.

Theorem 4.1 $LO_{1,3/*,*}P_{4,*,3}(active_1, a, b, c, d, e) = RE$.

Proof. In the description of the P system Π below, w_s describes the initial multiset for the skin membrane, whereas w'_s denotes the initial multiset in the elementary membrane having the same label as the skin membrane. We now simulate a register machine $M = (2, T, Q, I, q_0, q_f)$:

$$
\begin{aligned}
\Pi &= (O, \mu, w_s, w_1, w_2, w'_s, R),\\
\mu &= [\,[\,]_1[\,]_2[\,]_s\,]_s,\\
O &= T \cup \{a_i \mid 1 \le i \le 2\} \cup \{q, q_1, q_2 \mid q \in Q\} \cup \{b_1, b_2, t, d, \#\},\\
w_s &= q_0,\ w_1 = w_2 = w'_s = \lambda,\\
R &= R_O \cup R_A \cup R_S \cup R_D \cup R_Z.
\end{aligned}
$$

The rules are grouped in categories: output, add, subtract, decrement case and zero case.

$$
\begin{aligned}
R_O &= [\, q \to q'a\,]_s,\ [\, q \to q''a\ \,]_s,\ [\, a\,]_s \to [\ \,]_s a \mid (q : [SaW], q'),\ a \in T\},\\
R_A &= \{q[\ \,]_i \to [\, q\,]_i,\ [\, q\,]_i \to [\, q_1\,]_i[\, t\,]_i,\ [\, t \to \lambda\,]_i,\ [\, q_1\,]_i \to [\ \,]_i l',\\
&\quad [\, q_1\,]_i \to [\ \,]_i q'' \mid (q : [RiP], q', q''),\ i \in \{1, 2\}\},\\
R_S &= [\, q \to db_i q_1\,]_s,\ b_i[\ \,]_i \to [\, b_i\,]_i,\ [\, b_i\,]_i \to [\ \,]_i t,\ [\, t \to \lambda\,]_s,\\
&\quad [\, b_i \to \#\,]_s,\ [\, b_i \to \#\,]_i,\ [\, \# \to \#\,]_i \mid (q : \langle RiZM\rangle, q', q''),\ i \in \{1, 2\}\}\\
&\cup \{d[\ \,]_s \to [\, t\,]_s,\ [\, t \to \lambda\,]_s,\ [\, d \to \#\,]_s,\ [\, \# \to \#\,]_s\},\\
R_D &= \{q_1[\ \,]_i \to [\, q_1\,]_i,\ [\, q_1\,]_i \to q' \mid (q : \langle RiZM\rangle, q', q''),\ i \in \{1, 2\}\},\\
R_Z &= \{q_1[\ \,]_s \to [\, q_2\,]_s,\ [\, q_2\,]_s \to [\ \,]_s q'' \mid q \in I_-\}.
\end{aligned}
$$

Like in Theorem 2.29, we simulate a register machine with output tape and 2 registers; the values of registers $i \in \{1, 2\}$, are represented by the multiplicities of membranes i. However, since new membranes can only be created by dividing existing ones, one extra membrane is needed for every register. The duty of d is to "keep busy" the elementary membrane labeled s (otherwise # appears and the computation does not halt), and the use of the objects b_i is to " keep busy" one membrane labeled i for two steps. Object t is not needed for the computation, it is only used to keep the usual form of membrane division and communication rules, it is immediately erased.

Generating an output is done by a non-cooperative rule changing the instruction label and producing the corresponding symbol, which is then sent out. Incrementing a register $(q : [RiP], q', q'')$ is done in the following way: q enters membrane i (there is always at least one), dividing it. The object q_1 in one copy is sent to the skin as q' or q'', while the object t in the other copy is erased.

Subtracting with $(q : \langle RiZM\rangle, q', q''))$ is done by keeping busy the elementary membrane labeled s for one step and one membrane labeled i for two steps, while object q_1 tries to enter any membrane labeled i. If the register is not zero, then q_1 immediately enters one of the other membranes labeled i, dissolves it and changes to l'. Otherwise after waiting for one step object q_1 enters the elementary membrane labeled s and returns to the skin as q''.

In a correct simulation of a register machine run (in particular, during any halting computation) there are never > 3 objects inside the system. □

It is possible to reduce the number of membrane labels to two at the price of starting with seven membranes.

Theorem 4.2 $LO_{1,3/*,*}P_{7,*,2}(active_1, a, b, c, d, e) = RE$.

Proof. (sketch) In a way similar to the proof of Theorem 2.29, let us start with a membrane structure $[\,[\,[\,]_1[\,]_2\,]_1[\,[\,]_1[\,]_2\,]_2\,]_1$ (see also Figure 4.6), represent the values of a working register i by the number of elementary membranes with label 2, inside the membranes with labels i, minus one. The elementary membranes with label 1 will be used for delay, just like the elementary membrane labeled by s was used in the proof of Theorem 4.1, and the instructions are simulated accordingly. The main difference is that when simulating an ADD or a SUB instruction we have one additional initial step at the beginning choosing the "cluster membrane" representing the corresponding register (see the proof of Theorem 2.29). Obviously, at the end of the simulation of the instruction in the right "cluster membrane", we need an additional final step for moving the instruction label back to the skin. □

4.2.2 Two polarizations

The last theorem established in this section shows that with two polarizations we need only one membrane to simulate register machines deterministically:

Theorem 4.3 $DPs_aOP_{1,1,1}(active_2, a, c) = PsRE$.

Proof. We will simulate the actions of a deterministic register machine $M = (d, Q, I, q_0, q_f)$ with d registers by a deterministic P system with one membrane and two polarizations. For every instruction l, let us denote the register q acts on by $r(q)$ and the operation q carries out by $op(q)$.

$$\begin{aligned}
\Pi &= (O, E, [\,]_1^0, w_1, R),\\
O &= \{(a,i,j) \mid 1 \le i \le d,\ 0 \le j \le d+2\}\\
&\cup\ \{(l,i,j) \mid l \in P,\ 0 \le i \le 2,\ 1 \le j \le d+2\} \cup \{\#\},\\
E &= \{0,1\},\ w_1 = (l_0, 0, 0).
\end{aligned}$$

The system receives the input $(a,1,0)^{n_1} \cdots (a,d,0)^{n_d}$ in addition to w_1 in the skin. The set R contains the rules

$$[\ z\ ]_1^e \to [\,]_1^{1-e} z,\ e \in \{0,1\}, \tag{4.1}$$
$$[\ (a,i,j) \to (a,i,j+1)\ ]_1^0,\ 1 \le i \le d,\ 0 \le j \le d+1, \tag{4.2}$$
$$[\ (a,i,d+2) \to (a,i,0)\ ]_1^0,\ 1 \le i \le d, \tag{4.3}$$
$$[\ (a,i,j+1)\ ]_1^1 \to [\,]_1^0 (a,i,j+1),\ 1 \le i \le d, \tag{4.4}$$
$$[\ (q,0,j) \to (q,0,j+1)\ ]_1^0,\ q \in Q,\ 0 \le j < r(q) - 1, \tag{4.5}$$
$$[\ (q,i,j) \to (q,i,j+1)\ ]_1^0,\ q \in Q,\ i \in \{1,2\},\ r(q) \le j \le d+1, \tag{4.6}$$
$$[\ (q,1,d+2) \to (q',0,0)\ ]_1^0,\ q \in Q, \tag{4.7}$$
$$[\ (q,2,d+2) \to (q'',0,0)\ ]_1^0,\ q \in Q, \tag{4.8}$$
$$[\ (q,0,j) \to (q,1,j+1)(a,j+1,j+1)\ ]_1^0,\ q \in Q,$$
$$j = r(q) - 1,\ op(q) = [RP], \tag{4.9}$$
$$[\ (q,0,j) \to (q,0,j+1)z\ ]_1^0,\ q \in Q,\ j = r(q) - 1,\ op(q) = \langle RZM \rangle, \tag{4.10}$$
$$[\ (q,0,j) \to (q,0,j+1)\ ]_1^0,\ q \in Q,\ j = r(q),\ op(q) = \langle RZM \rangle, \tag{4.11}$$

$$[\ (q,0,j) \to (q,0,j+1)\ ]_1^1,\ q \in Q,\ j = r(q)+1,\ op(q) = \langle RZM \rangle, \tag{4.12}$$

$$[\ (q,0,j) \to (q,1,j+1)\ ]_1^0,\ q \in Q,\ j = r(q)+2,\ op(q) = \langle RZM \rangle, \tag{4.13}$$

$$[\ (q,0,j) \to (q,2,j)z\ ]_1^1,\ q \in Q,\ j = r(q)+2,\ op(q) = \langle RZM \rangle, \tag{4.14}$$

$$[\ (q,2,j) \to (q,2,j+1)\ ]_1^1,\ q \in Q,\ j = r(q),\ op(q) = \langle RZM \rangle, \tag{4.15}$$

$$[\ (q_f,0,0)\ ]_1^0 \to [\ \ ]_1^1(q_f,0,0),\ 1 \le i \le d. \tag{4.16}$$

The idea of this proof is similar to the one from [71], [72]: the symbols corresponding to the registers have states (second subscript) $0, \cdots, d+2$, and so do the symbols corresponding to the instructions of the register machine. The first subscript of the instruction symbols is 0 if the instruction has not yet been applied, and it is 1 if increment or decrement has been applied, and 2 if the decrement has failed.

Most of the time the polarization is 0; object z can reverse the polarization by (4.1). When the polarization is 0, the register symbols cycle through the states by (4.2), (4.3). Before the current instruction q is applied, the instruction symbols also cycle through the states until the state becomes $r(q)-1$, i.e., the index of the register (the instruction operates on) minus one. We will explain the details of the application below. After the instruction has been applied, the first subscript of the instruction symbol changes to 1 or 2 and it cycles through the states by (4.6), finally changing into q' by (4.7) or into l'' by (4.8).

Addition is done by rule (4.9). Decrement is done by a "diagonalization technique": polarization 1 when register i is in state $i+1$ signals a decrement attempt of register i by (4.4), and the polarization will change if and only if it has been successful. Thus, to apply $(q : \langle RiZM \rangle, q', q'')$, the instruction symbol in state $i-1$ additionally produces a symbol z. By the time z changes the polarization to 1 by (4.1), all other symbols reach state $i+1$. After one more step the state symbol checks whether the decrement has been successful, (4.13), or not, (4.14). After a successful decrement all symbols continue changing states with polarization 0 and state $i+1$. Otherwise, the instruction symbol additionally produces a symbol z and after one more step all symbols continue changing states with polarization 0 and state $i+1$.

After the simulation of M has reached the final label, the instruction symbol exits the system, changing the polarization to 1. Since the register symbols are in state 1, the system halts. □

Looking into the proof of the preceding theorem we realize that even a more general result is shown: the multisets remaining in the skin membrane at the end of a halting computation can be interpreted as the computation result:

Corollary 4.1 *Any partially recursive function can be computed by a deterministic P system with 1 membrane, 2 polarizations and internal output.*

Conclusions and Open Problems We have shown that RE is generated by P systems using four membranes and three labels or seven membranes and two labels in the initial configuration, where at most three objects are ever present in any halting computation.

It is known from [72], [71] that P systems with two polarizations and rules of types (a) and (c) generate $PsRE$ using two membranes, or accept $PsRE$ using one membrane. We

have proved in this section that *deterministic* systems of this kind with one membrane accept $PsRE$. Moreover, in the proof of this result (Theorem 4.3), the rules are global (there is only one membrane) and rules of type (c) are non-renaming (the contents of the environment does not matter).

Improving any complexity parameter greater than one (especially in the case of $*$) in any theorem is an open question. Moreover, the following questions are of interest:

- What is the power of *deterministic* P systems with membrane division (without polarizations, without changing labels, etc.)?
- How can the types of rules be restricted in Theorem 4.1?
- What further restrictions cause a complexity trade-off?
- What is the generative power of P systems without polarizations and m membranes, $m = 1, 2, 3$?
- What is the generative power of one-membrane P systems with two polarizations and external output?

4.3 Efficiency with Two Polarizations

We present an algorithm for deterministically deciding `SAT` in linear time by P systems with active membranes using only two polarizations and rules of types (a), (c), and (e). Moreover, various restrictions on the general form of the rules are considered: global, non-renaming, independent of the polarization, preserving it, changing it, producing two membranes with different polarizations, having exactly one or two objects in (each membrane of) the right-hand side, thus improving results from [72]. Several problems related to different combinations of these restrictions are formulated, too.

Membrane systems are biologically motivated theoretical models of distributed and parallel computing. The most interesting questions probably are completeness (solving every solvable problem) and efficiency (solving a hard problem in feasible time). We here address the latter problem, i.e., we shall give an algorithm how to decide `SAT` in linear time using only two polarizations in P systems with active membranes.

The question of removing the polarizations (charges $+, -, 0$ associated with the membranes) from P systems with active membranes without diminishing their computing power or their efficiency in solving computationally hard problems in a feasible time was formulated several times and was considered in various contexts (with the polarizations replaced by various other features, such as label changing – see, e.g., [107], [108]). Here, following [72], we present another way for improving previous results: the number of polarizations can be decreased to two, without introducing new features.

There are numerous results of solving such (mostly NP-complete) problems as `SAT`, `HPP`, Validity, Subset-Sum, Knapsack, Vertex Cover, Clique, `QBF-SAT` by P systems with active membranes with three polarizations (see, e.g., references in [60]). The ability of the systems to act depending on the membrane polarizations and to change them is a powerful control

feature, the use of which is not necessary if one pays the price of changing membrane labels. Another result is solving SAT in a semi-uniform manner, without polarizations and without changing labels, but also using membrane dissolution and non-elementary membrane division. Here we show that two polarizations are enough even when restricting the types of rules to (a), (c), and (e). It remains as an open question whether polarizations can be completely removed, and we *conjecture* that the answer is negative.

Moreover, we consider a few restrictions on the general form of the rules, under which it is still possible to solve SAT. The motivations of considering these restrictions are of three kinds: bringing the construction closer to biological cells (as "realistic" as possible); building a normal form (as restrictive as possible), for the possible future direct simulation results; and finding out which aspects of active membranes are essential for the efficiency.

We use the following notation for instances of the SAT problem:

We consider a propositional formula in conjunctive normal form:

$$\begin{aligned} \beta &= C_1 \wedge \cdots \wedge C_m, \\ C_i &= y_{i,1} \vee \cdots \vee y_{i,l_i},\ 1 \leq i \leq m, \text{ where} \\ y_{i,k} &\in \{x_j, \neg x_j \mid 1 \leq j \leq n\},\ 1 \leq i \leq m, 1 \leq k \leq l_i, \end{aligned}$$

i.e., n is the number of variables and m is the number of clauses, hence, to β the size (n, m) is associated. For arbitrary $(n, m) \in \mathbb{N}^2$, we denote the family of SAT problems of size (n, m) by $\mathtt{SAT}(n, m)$.

4.3.1 Using global rules

As it was shown in [72], $\mathtt{SAT}(n, m)$ can be decided in linear time (linear with respect to n and m, i.e., the algorithm has time complexity $O(n + m)$) by a uniform family of P systems with two polarizations, only using rules of types (a), (c), and (e). Throughout this section we will always restrict ourselves to restricted variants of these types of rules.

We first recall the theorem from [72], giving the construction of the proof and short explanations as well as repeating the example that illustrates the corresponding construction.

Theorem 4.4 $\mathtt{SAT}(n, m)$ *can be deterministically decided in linear (with respect to* n *and* m*) time by a uniform family of P systems with active membranes with two polarizations and global rules of types* (a)*,* (c)*, and* (e)*.*

Proof. An instance β of the $\mathtt{SAT}(n, m)$ problem as described above is encoded as a multiset over $V(n, m) = \{x_{i,j,j}, x'_{i,j,j} \mid 1 \leq i \leq m,\ 1 \leq j \leq n\}$.The object $x_{i,j,j}$ ($x'_{i,j,j}$) represents the variable x_j appearing in the clause C_i without (with) negation. Thus, the input multiset is

$$\begin{aligned} w &= \{x_{i,j,j} \mid x_j \in \{y_{i,k} \mid 1 \leq k \leq l_i\},\ 1 \leq i \leq m,\ 1 \leq j \leq n\} \\ &\cup \{x'_{i,j,j} \mid \neg x_j \in \{y_{i,k} \mid 1 \leq k \leq l_i\},\ 1 \leq i \leq m,\ 1 \leq j \leq n\}, \end{aligned}$$

which is placed into membrane 2 in addition to the initial symbol d_0 in the P system $\Pi(n, m)$ we will construct for any given $(n, m) \in \mathbb{N}^2$:

$$
\begin{aligned}
\Pi(n,m) &= (O(n,m), \{0,1\}, [\ [\ ]_2\ ]_1, t_0, d_0, 0, 0, R, 2),\\
O(n,m) &= \{x_{i,j,k}, x'_{i,j,k} \mid 1 \le i \le m,\ 0 \le k \le j \le n\} \cup \{z, o, \texttt{yes}, \texttt{no}\}\\
&\cup\ \{c_{i,j} \mid 0 \le i \le m, 0 \le j \le n\} \cup \{c_i \mid 0 \le i \le m\}\\
&\cup\ \{d_i \mid 0 \le i \le n+1\} \cup \{e_i \mid 0 \le i \le m+1\}\\
&\cup\ \{t_h \mid 0 \le h \le n+2m+4\};
\end{aligned}
$$

R contains the following rules (grouped by sub-tasks; see [72] for more explanations):
Global control in skin membrane

- $[\ t_h \to t_{h+1}\ ]^0$, $0 \le h \le n+2m+2$.

Generation phase

- $[\ d_j\ ]^e \to [\ d_{j+1}\ ]^0 [\ d_{j+1}\ ]^1$, $e \in \{0,1\}$, $0 \le j < n-1$;
- $[\ x_{i,j,k} \to x_{i,j,k-1}\ ]^e$,
  $[\ x'_{i,j,k} \to x'_{i,j,k-1}\ ]^e$, $e \in \{0,1\}$, $1 \le i \le m$, $1 \le k \le j \le n$;
- $[\ x_{i,j,0} \to \lambda\ ]^0$,
  $[\ x_{i,j,0} \to c_{i,j}\ ]^1$,
  $[\ x'_{i,j,0} \to c_{i,j}\ ]^0$,
  $[\ x'_{i,j,0} \to \lambda\ ]^1$, $1 \le i \le m$, $1 \le j \le n$;
- $[\ c_{i,j} \to c_{i,j+1}\ ]^e$, $e \in \{0,1\}$, $1 \le i \le m$, $1 \le j < n$;
- $[\ d_n \to d_{n+1} z\ ]^1$,
  $[\ d_n \to d_{n+1}\ ]^0$.

During each of the first n steps, every elementary membrane is duplicated, in order to examine all possible truth assignments to the variables $x_1, \cdots, x_n$.

In step j of the generation phase, one of membranes resulting from the application of rule $[\ d_j\ ]^e \to [\ d_{j+1}\ ]^0 [\ d_{j+1}\ ]^1$ gets polarization 0, corresponding to assigning the truth value `false` to x_j (and in this case the clauses where $\neg x_j$ appears are satisfied), and the other membrane gets polarization 1, corresponding to assigning the truth value `true` to x_j (and in this case those clauses where x_j appears without negation are satisfied). Due to the application of the rules $[\ x_{i,j,0} \to \lambda\ ]^0$, $[\ x_{i,j,0} \to c_{i,j}\ ]^1$, $[\ x'_{i,j,0} \to c_{i,j}\ ]^0$, $[\ x'_{i,j,0} \to \lambda\ ]^1$, only those variables "survive" which correspond to the correct truth assignment at the moment the last index has reached the ground level 0.

After the end of this first phase of the algorithm, 2^n elementary membranes (each of them with label 2) have been produced, each of them containing d_{n+1} and objects $c_{i,n}$ for all clauses C_i that are satisfied. Every membrane with polarization 1 also contains an object z. This procedure described so far in total takes $n+1$ step.

Transition phase

- $[\ z\ ]^1 \to [\ ]^0 o$;

- $[\ d_{n+1} \to e_1\ ]^e$, $e \in \{0,1\}$;
- $[\ c_{i,n} \to c_i\ ]^e$, $e \in \{0,1\}$, $1 \leq i \leq m$.

By the application of the rule $[\ z\ ]^1 \to [\ \ ]^0 o$ the polarization of the membranes polarized by 1 is reset to zero again by passing through the surrounding membrane, thereby also yielding the "garbage" symbol o within the skin membrane. After this single step of the transition phase all the elementary membranes now have the polarization 0 and contain e_1 as well as c_i for every satisfied clause C_i.

Checking phase

- $[\ c_1\ ]^0 \to [\ \ ]^1 o$;
- $[\ e_i \to e_{i+1} z\ ]^0$, $1 \leq i < m$;
- $[\ c_1 \to \lambda\ ]^1$;
- $[\ c_i \to c_{i-1}\ ]^1$, $2 \leq i \leq m$;
- $[\ e_m \to e_{m+1}\ ]^0$;
- $[\ e_{m+1}\ ]^1 \to [\ \ ]^1$ `yes`.

All clauses are satisfied if and only if all objects $c_1, \cdots, c_m$ are present in some membrane, and at the end all objects c_i, $1 \leq i \leq m$, have been sent out into the skin membrane. While checking the last clause, no object z (for resetting the polarization of the membrane as this is done in the preceding steps) is produced from e_m by applying the rule $[\ e_m \to e_{m+1}\ ]^0$, hence, e_{m+1} will be present in a membrane with polarization 1 thus allowing for the application of rule $[\ e_{m+1}\ ]^1 \to [\ \ ]^1$ `yes` indicating that the corresponding elementary membrane represented a solution of the given satisfiability problem. In total, this phase takes $2m$ steps.

Output phase

- $[\ $`yes`$\ ]^0 \to [\ \ ]^1$ `yes`;
- $[\ t_{n+2m+3}\ ]^0 \to [\ \ ]^1$ `no`.

Every elementary membrane which after the first $n+1$ steps had represented a solution of the given satisfiability problem, after $n+1+1+2m$ steps has sent a copy of `yes` into the skin membrane, and in the next step one of these copies exits into the environment by using rule $[\ $`yes`$\ ]^0 \to [\ \ ]^1$ `yes`, thus giving the positive result `yes` and changing the skin polarization to 1 in order to prevent further output. If, on the other hand, the given satisfiability problem has no solution, after $n+2m+3$ steps the polarization of the skin membrane will still be 0, hence, rule $[\ t_{n+2m+3}\ ]^0 \to [\ \ ]^1$ `no` sends out the correct answer `no`. □

The construction elaborated above is illustrated by an example, see Figure 4.7.

It is worth noticing that the rules are *global*: the same set of rules is valid for all membranes, i.e., in the rules, the labels of the membranes can be omitted. We also note that in this construction already elaborated in [72], the membrane division rules do not depend on the polarization (which therefore can be omitted in the meaning of "applicable for any membrane"), and the contents of membranes after division is identical, but the polarizations

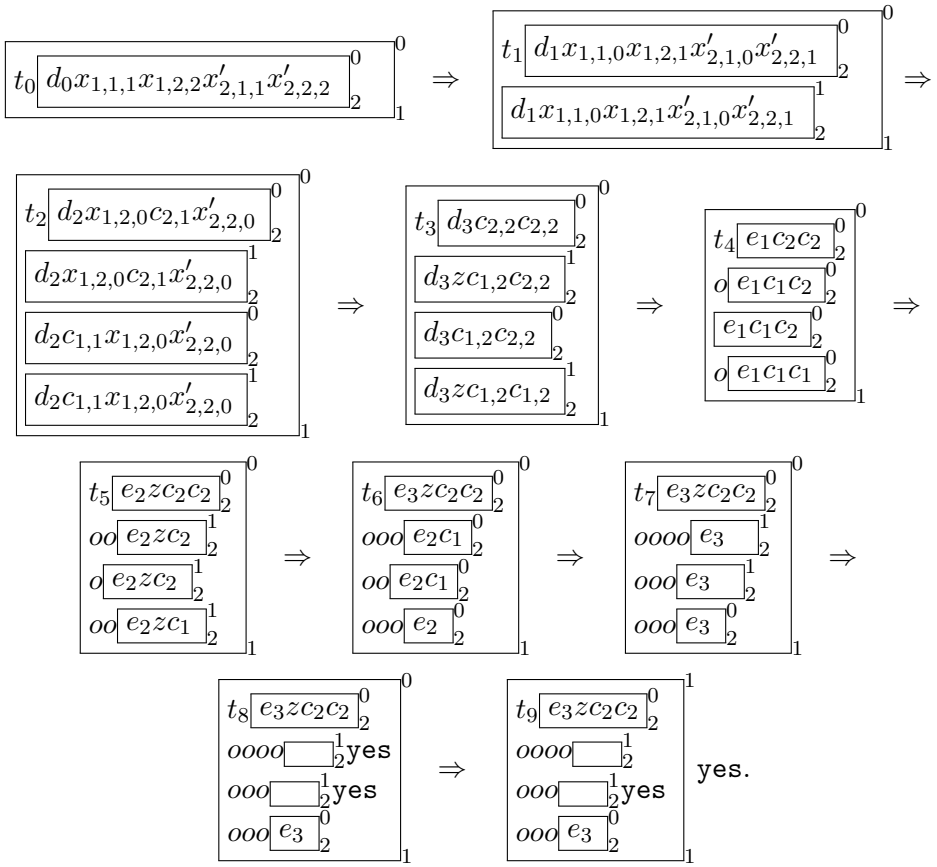


Figure 4.7: A run a P system deciding solvability of $\gamma = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$

are different. Finally, every rule of type (c) changes the polarization (the superscript $\neg$ will be used to denote this variant).

Thus, all rules used are even of the following restricted forms (where the interpretation of the subscripts g, $g1$, and $g2$ is explained in the Appendix A4; the superscript $\neg$ indicates that the polarization is changed):

(a_g) $[\ a \to v\ ]^i$,

(c_{g1}) $[\ a\ ]^i \to [\ \ ]^{\neg}\ b$,

(e_{g2}) $[\ a\ \ ] \to [\ b\ ]^0[\ b\ ]^1$,
where $a, b \in O$, $v \in O^*$, $h \in H$, $i \in \{0, 1\}$.

By the explanations above, we now have even proved a stronger result than that in [72]. The subsequent discussion on restricting rules is given in the Appendix A4.

Conclusions In Theorem A4.1 we have given an algorithm for deciding the NP-complete decision problem $\mathtt{SAT}(n, m)$ by a uniform family of P system with active membranes in linear time (with respect to nm) with only two polarizations and rules of types (a), (c), and (e), of specific restrictive types. Various other restrictions are summarized in Corollary A4.1, followed by the discussion.

The question remains whether further or other restrictions, respectively, of the general form of these rules are possible. For instance, can the problem be solved using only rules of

types (a), (c_{p0}), (e) (the rules of type (c) do not depend on the polarization and preserve it)? What about using only types (a_p), (c), (e) (the rules of type (a) do not depend on the polarization)?

Another interesting question is to study systems with rules of types (a_u), (b), (c), (d), (e); such systems can only increase the number of objects via membrane division. What is their generative power? Are they efficient?

4.4 Beyond NP and co-NP

Due to massive parallelism and exponential space in membrane systems, some intractable computational problems can be solved by P systems with active membranes in polynomial number of steps. In this section we present a generalization of this approach from decisional problems to the computational ones, by providing a solution of a #**P**-complete problem, namely to compute the permanent of a binary matrix. The implication of this result to the **PP** complexity class is discussed and compared to the known result of $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP}$.

Membrane systems are convenient for describing polynomial-time solutions to certain intractable problems in a massively parallel way. Division of membranes makes it possible to create exponential space in linear time, suitable for attacking problems in **NP** and even in **PSPACE**. Their solutions by P systems with active membranes have been investigated in a number of papers since 2001, later focusing on solutions by restricted systems.

The first efficient *semi–uniform solution* to `SAT` was given by Gh. Păun in [253], using division for non–elementary membranes and three electrical charges. This result was improved by Gh. Păun, Y. Suzuki, H. Tanaka, and T. Yokomori in [260] using only division for elementary membranes.

Different efficient *uniform solutions* have been obtained in the framework of recognizer P systems with active membranes, with polarizations and only using division rules for elementary membranes (see, e.g., references in [35]).

Let us cite [282] for additional motivation: While `3SAT` and the other problems in **NP**-complete are widely assumed to require an effort at least proportional to 2^n, where n is a measure of the size of the input, the problems in #**P**-complete are harder, being widely assumed to require an effort proportional to $n2^n$.

While attacking **NP** complexity class by P systems with active membranes have been often motivated by $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ problem, we recall the following fact from [288]:

If the permanent can be computed in polynomial time by any method, then **FP**=#**P** which is an even stronger statement than **P**= **NP**.

Here, by "any method" one understands "... on sequential computers" and **FP** is the set of polynomial-computable functions.

Finally, we recall the definition of **PP** (the probabilistic polynomial time complexity class) and present an approach to solving the problems in **PP**.

4.4.1 Permanent of a matrix

The complexity class #**P**, see [291], was first defined in [277] in a paper on the computation of the permanent.

Definition 4.1 *Let S_n be the set of permutations of integers from 1 to n, i.e., the set of bijective functions $\sigma : \{1, \cdots, n\} \to \{1, \cdots, n\}$. The permanent of a matrix $A = (a_{i,j})_{1 \leq i,j \leq n}$ is defined as $perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)}$.*

Informally, consider a combination of n matrix elements containing one element from every row and one element from every column. The permanent is the sum over all such combinations of the product of the combination's elements.

A matrix is binary if its elements are either 0 or 1. In this case, the permanent is the number of combinations of n matrix elements with value 1, containing one element from each row and one element from each column. For example, $perm \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} = 2$. Unlike the determinant of a matrix, the permanent cannot be computed by Gauss elimination.

Theorem 4.5 *The problem of computing a permanent of a binary matrix is solvable in polynomial time by a uniform family of deterministic P systems with active membranes with two polarizations and rules of types $(a), (c), (e)$.*

We give the proof in Appendix A5.

Note that requirement that the output region is the environment (typical for decisional problem solutions) has been dropped. This makes it possible to give non-polynomial answers to the permanent problem (which is a number between 0 and $n!$) in polynomial number of steps without having to recall from [108] rules sending objects out that work in parallel.

4.4.2 Attacking PP complexity class

The probabilistic polynomial complexity class **PP**, also called Majority-P, has been introduced in [191]. It is the class of **decision** problems solvable by a probabilistic Turing machine in polynomial time, with an error probability of less than $1/2$ for all instances, see also [289].

It is known that $\mathbf{PP} \supseteq \mathbf{NP} \cup \mathbf{co} - \mathbf{NP}$, and the inclusion is strict if $\mathbf{P} \neq \mathbf{NP}$. Therefore, showing a solution to a **PP**-complete problem by P systems with active membranes without division of non-elementary membranes and without membrane creation would improve the best known results ($\mathbf{NP} \cup \mathbf{co} - \mathbf{NP}$).

In this section we show a way to do this, paying a small price of post-processing. We recall that the framework of solving decisional problems by P systems with active membranes includes two encoding functions (computing the description of a P system from the size of the problem instance and computing the input multiset from the instance of the problem). Unlike a more general case of solving computational problems, there was no need for the decoding function, since the meaning of objects `yes` and `no` sent to the environment was linked with the answer. While the decoding function was necessary for extending the framework for the computational problems (computing the answer to the instance of the problem from the output multiset of a P system in polynomial time), we would like to underline that it is useful even for the decisional problems.

It is not difficult to see that the problem "given a matrix A of size n, is $Perm(A) > n!/2$?" is **PP**-complete. Hence, we only have to compare the result of the computation of the matrix

permanent with $n!/2$. Doing it by usual P systems with active membranes would need a non-polynomial number of steps. We can propose two approaches.

- Generalizing rules of type (a) to cooperative ones. It would then suffice to generate $n!/2$ copies of a new object z, then erase pairs of o and z and finally check if some object o remains. However, this class of P systems is not studied.
- Consider, as before, the number of objects o as the result of the computation of a P system. Use the decoding function $dec(x) = \begin{cases} \texttt{no} & , \ x \leq n!/2, \\ \texttt{yes} & , \ x > n!/2. \end{cases}$ The function dec can obviously computed in polynomial time.

Discussion In this section we presented a solution to the problem of computing a permanent of a binary matrix by P systems with active membranes, namely with two polarizations and rules of object evolution, sending objects out and membrane division. This problem is known to be #**P**-complete. The solution has been preceded by the framework that generalizes decisional problems to computing functions: now the answer is much more than one bit. This result suggests that P systems with active membranes without non–elementary membrane division still compute more than decisions of the problems in $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP}$. Indeed, paying the small price of using the decoding function also for decisional problem this approach allows to solve the class **PP**, which is strictly larger than that (assuming $\mathbf{P} \neq \mathbf{NP}$).

The main result presented in this section has been improved by the Milano group, by implementing decoding also by P systems themselves without additional features. Hence, P systems with active membranes without non–elementary membrane division compute at least **PP**. A subsequent paper by the Milano group, contains an attempt to implement solutions to $\mathbf{P}^{\mathbf{PP}}$ problems, implementing **PP** oracles as subsystems of a P system. Although $\mathbf{P}^{\mathbf{PP}}$ solves entire polynomial hierarchy (by Toda's Theorem), the question about whether P systems with active membranes without non–elementary membrane division solve the whole **PSPACE** class is open, in particular due to difficulties of understanding the uniformity criterion and whether such transformations are of a permitting (i.e., polynomial time) complexity.

4.5 Attacking PSPACE

It is known that the satisfiability problem (`SAT`) can be solved with a semi-uniform family of deterministic polarizationless P systems with active membranes with non–elementary membrane division. We present a double improvement of this result by showing that the satisfiability of a *quantified* Boolean formula (`QSAT`) can be solved by a *uniform* family of P systems of the same kind.

A particularly interesting model of membrane systems are the systems with active membranes, see [253], where membrane division can be used in order to solve computationally hard problems in polynomial or even linear time, by a space–time trade-off. The description of rules in this model involves membranes and objects; the typical types of rules are (a) object evolution, $(b), (c)$ object communication, (d) membrane dissolution, $(e), (f)$ membrane division. Since membrane systems are an abstraction of living cells, the membranes are

arranged hierarchically, yielding a tree structure. A membrane is called elementary if it is a leaf of this tree, i.e., if it does not contain other membranes.

The first efficient *semi–uniform solution* to SAT was given by Gh. Păun in [253], using division for non–elementary membranes and three electrical charges. This result was improved by Gh. Păun, Y. Suzuki, H. Tanaka, and T. Yokomori in [260] using only division for elementary membranes (in that paper also a semi–uniform solution to *HPP* using membrane creation is presented).

P. Sosík in [272] provides an efficient *semi–uniform solution* to QSAT (quantified satisfiability problem), a well known **PSPACE**–complete problem, in the framework of P systems with active membranes but using cell division rules for non–elementary membranes. A *uniform solution* for QSAT was presented in [100], while a *semi–uniform* **polarizationless** *solution* for SAT was presented in [108].

Different efficient *uniform solutions* have been obtained in the framework of recognizer P systems with active membranes, with polarizations and only using division rules for elementary membranes (see, e.g., references in [114]). Nevertheless, the polynomial complexity class associated with recognizer P systems with active membranes and with polarizations does not seem precise enough to describe classical complexity classes below **PSPACE**. Hence, it is challenging to investigate weaker variants of membrane systems able to characterize classical complexity classes. Here we work with a variant of these systems not using polarizations.

A uniform solution of QSAT In this section we extend the result (2) from Theorem 1, providing an uniform and linear time solution of QSAT (*quantified satisfiability*) problem, through a family of recognizer P systems using polarizationless active membranes, dissolution rules and division for elementary and non–elementary membranes.

Given a Boolean formula $\varphi(x_1, \ldots, x_n)$ in conjunctive normal form, with Boolean variables $x_1, \ldots, x_n$, the sentence $\varphi^* = \exists x_1 \forall x_2 \ldots Q_n x_n \varphi(x_1, \ldots, x_n)$ (where Q_n is $\exists$ if n is odd, and Q_n is $\forall$ otherwise) is said to be the (existential) *fully quantified* formula associated with $\varphi(x_1, \ldots, x_n)$. Recall that a sentence is a Boolean formula in which every variable is in scope of a quantifier. We say that φ^* is satisfiable if for each truth assignment, σ, over $\{i : 1 \leq i \leq n \ \wedge \ i \text{ even}\}$ there is an extension σ^* of σ over $\{1, \ldots, n\}$ such that the value of x_i only depends on the values of x_j, $1 \leq j < i$, verifying $\sigma^*(\varphi(x_1, \ldots, x_n)) = 1$.

The QSAT problem is the following: *Given the (existential) fully quantified formula φ^* associated with a Boolean formula $\varphi(x_1, \ldots, x_n)$ in conjunctive normal form, determine whether or not φ^* is satisfiable.* It is well known that QSAT is a **PSPACE**–complete problem [246].

Theorem 4.6 $\texttt{QSAT} \in \mathbf{PMC}_{\mathcal{AM}^0(+d,+ne)}$.

Proof. The solution proposed follows a brute force approach, in the framework of recognizer P systems with polarizationless active membranes where dissolution rules, and division for elementary and non–elementary membranes are permitted. The solution is grouped in stages:

Generation stage: using membrane division for elementary and non–elementary membranes, all truth assignments for the variables associated with the Boolean formula are produced. *Assignments stage*: in a special membrane we encode the clauses that are satisfied for each truth assignment. *Checking stage*: we determine what truth assignments make the Boolean formula evaluate to true. *Quantifier stage*: the universal and existential gates of

the fully quantified formula are simulated and its satisfiability is encoded by a special object in a suitable membrane. *Output stage*: The systems sends out to the environment the right answer according to the result of the previous stage.

Let us consider a propositional formula in the conjunctive normal form:

$$\begin{aligned} \varphi &= C_1 \wedge \cdots \wedge C_m, \\ C_i &= y_{i,1} \vee \cdots \vee y_{i,l_i},\ 1 \leq i \leq m, \text{ where} \\ y_{i,k} &\in \{x_j, \neg x_j \mid 1 \leq j \leq n\},\ 1 \leq i \leq m, 1 \leq k \leq l_i. \end{aligned}$$

We consider a normal form for `QSAT`: the number of variables is even ($n = 2n'$) and the quantified formula is $\varphi^* = \exists x_1 \forall x_2 \cdots \exists x_{n-1} \forall x_n\ \varphi(x_1, \ldots, x_n)$.

Let us consider the (polynomial time computable and bijective) pair function from $\mathbb{N}^2$ onto $\mathbb{N}$ defined by $\langle n, m\rangle = ((n+m)(n+m+1)/2)+n$. Depending on numbers m (of clauses) and n (of variables), we will consider a system $(\Pi(\langle n, m\rangle), \Sigma(\langle n, m\rangle), i_0)$, where $i_0 = 0$ is the input region and $\Sigma(\langle n, m\rangle) = \{v_{i,j}, v'_{i,j} \mid 1 \leq i \leq m,\ 1 \leq j \leq n\}$ is the input alphabet.

The problem instance φ will be encoded in the P system by a multiset containing one copy of each symbol from sets $X, X' \subseteq \Sigma(\langle n, m\rangle)$, corresponding to the clause-variable pairs such that the clause is satisfied by *true* and *false* assignment of the variable, defined below. We now construct the P system

$$\begin{aligned} X_\varphi &= \{v_{i,j} \mid x_j \in \{y_{i,k} \mid 1 \leq k \leq l_i\}, 1 \leq i \leq m,\ 1 \leq j \leq n\}, \\ X'_\varphi &= \{v'_{i,j} \mid \neg x_j \in \{y_{i,k} \mid 1 \leq k \leq l_i\}, 1 \leq i \leq m,\ 1 \leq j \leq n\}. \\ \Pi(\langle n, m\rangle) &= (O, H, \mu, w_0, \cdots, w_{m+5n+3}, R), \text{ with} \\ O &= \Sigma(\langle n, m\rangle) \cup \{u_{i,j}, u'_{i,j} \mid 1 \leq i \leq m,\ 1 \leq j \leq n\} \\ &\cup \{d_i \mid 0 \leq i \leq 2m + 7n + 2\} \cup \{a_i, t_i, f_i \mid 1 \leq i \leq n\} \\ &\cup \{c_i \mid 1 \leq i \leq m\} \cup \{t, f, z, z', T, T', \texttt{yes}, \texttt{no}\}, \\ \mu &= [\,[\cdots[\,[\,\,]_0\,]_1 \cdots\,]_{m+5n+2}\,]_{m+5n+3}, \\ w_0 &= w_{m+5n+1} = d_0, \\ w_{m+2n+3i} &= d_{m+5n},\ 1 \leq i \leq n, \\ w_i &= \lambda, i \notin \{0, m + 5n + 1\} \cup \{m + 2n + 3i \mid 1 \leq i \leq n\}, \\ H &= \{0, \cdots, m + 5n + 3\}, \end{aligned}$$

and the following rules (we also explain their use):

Generation stage

G1 $[\ d_{3i} \to a_{i+1} d_{3i+1}\ ]_0$,
$[\ d_{3i+1} \to d_{3i+2}\ ]_0$,
$[\ d_{3i+2} \to d_{3i+3}\ ]_0$, $0 \leq i < n$.
$[\ d_{3n+i} \to d_{3n+i+1}\ ]_0$, $0 \leq i < m + 2n$.

We count to $m + 5n$, which is the time needed for producing all 2^n truth assignments for the n variables, as well as membrane sub-structures which will examine the truth value of formula φ for each of these truth assignments; this counting is done in the central membrane; moreover during steps $3i - 2$, $1 \leq i \leq n$, symbols $a_1, \cdots, a_n$ are subsequently produced.

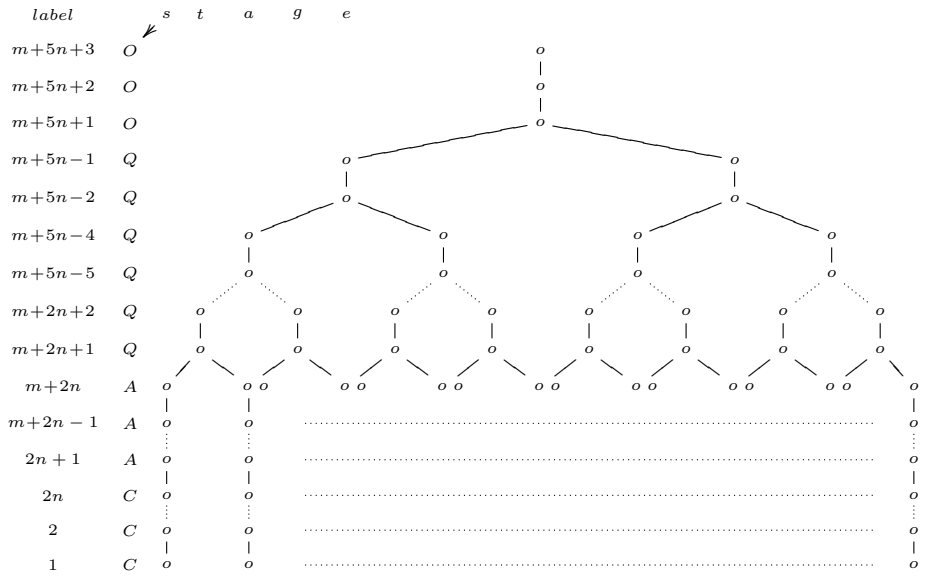


Figure 4.8: The membrane structure of the system Π after $m + 5n$ steps

G2 $[\ a_i\ ]_0 \to [\ t_i\ ]_0[\ f_i\ ]_0$, $1 \le i \le n$.

In membrane 0, we subsequently choose each variable x_i, $1 \le i \le n$, and both values *true* and $false$ are associated with it, in form of objects t_i and f_i, which are separated in two membranes with label 0. The division of membrane 0 is triggered by the objects a_i, which are introduced by the first rule from group G1 in steps $3i - 2$, $1 \le i \le n$; this is important in interleaving the use of these rules (hence the division of membrane 0) with the use of the rules of group G4, for dividing membranes placed above membrane 0.

G3 $[\ d_j \to d_{j-1}\ ]_{m+2n+3i}$, $1 \le j \le m + 5n - 2$, $1 \le i \le n$,
$[\ d_0\ ]_{m+2n+3i} \to z'$, $1 \le i \le n$.

After $m + 5n$ steps, dissolution rule is applied to membranes $m + 2n + 3i$.

G4 $[\ [\ \ ]_i[\ \ ]_i\ ]_{i+1} \to [\ [\ \ ]_i\ ]_{i+1}[\ [\ \ ]_i\ ]_{i+1}$, $0 \le i < m + 5n$.

These are division rules for membranes with label $0, 1, \cdots, m+5n$, to be used for the central membrane 0 in steps which follow the use of the first rule of type G1. The division of a membrane with label 1 is then propagated from lower levels to upper levels of the membrane structure and the membranes are continuously divided. The membrane division stops at the level where a membrane $m + 2n + 3i$ has been already dissolved by a rule from group G3. This results in the structure as shown in Fig. 4.8 after $m + 5n$ steps.

G5 $[\ d_{m+5n}\ ]_0 \to T$.

After $m + 5n$ steps, each copy of membrane with label 0 is dissolved and the contents is released into the surrounding membrane, which is labeled with 1.

Assignments stage

A1 $[\ t_i \to t'\ ]_{2i-1}$,
$[\ t'\ ]_{2i-1} \to z$,

$[\ f_i\ ]_{2i-1} \to f'$,
$[\ f' \to z\ ]_{2i}$,
$[\ z\ ]_{2i} \to z'$, $1 \leq i \leq n$.

Depending on the variable assignments, we need to determine what clauses are satisfied. For a variable x_i, this is done in membranes $2i-1$ and $2i$. The objects encoding the problem propagate through the membrane structure: object t_i dissolves membrane $2i-1$ after one step, and then it dissolves membrane $2i$ immediately, while object f_i dissolves membrane $2i-1$ immediately, and then it dissolves membrane $2i$ after one step.

A2 $[\ v_{i,j} \to u_{i,j}\ ]_{2i-1}$,
$[\ v'_{i,j} \to u'_{i,j}\ ]_{2i-1}$, $1 \leq i \leq n$, $1 \leq j \leq m$.

Once in membrane $2i-1$, objects $v_{i,j}$ and $v'_{i,j}$ wait for one step.

A3 $[\ u'_{i,j} \to \lambda\ ]_{2i-1}$,
$[\ u_{i,j} \to c_i\ ]_{2i-1}$,
$[\ u_{i,j} \to \lambda\ ]_{2i}$,
$[\ u'_{i,j} \to c_i\ ]_{2i}$, $1 \leq i \leq n$, $1 \leq j \leq m$.

If there is no membrane $2i-1$ in the meantime, then the objects encoding the instance of `SAT` assume the true value of x_i, otherwise, they assume the false value of x_i.

At the end of this routine (it takes $3n$ steps), a membrane with label $2n+1$ which contains all the symbols $c_1, \cdots, c_m$, corresponds to the truth assignment satisfying all clauses, hence it satisfies formula φ, and vice-versa.

Checking stage

C1 $[\ c_i\ ]_{2n+i} \to c_i$, $1 \leq i \leq m$.

A membrane with label $2n+i$, $1 \leq i \leq m$, is dissolved if and only if c_i appears in it (i.e., clause C_i is satisfied by the current truth assignment); if this is the case, the truth assignment associated with the membrane is released in the surrounding membrane. Otherwise, the truth assignment remains blocked in membrane $2n+i$ and never used at the next steps by the membranes placed above.

C2 $[\ T\ ]_{m+2n+1} \to T$.

The fact the object T appears in the membrane labeled $m+2n+1$ means that there is a truth assignment which satisfies formula φ. In this case, the membrane labeled $m+2n+1$ is dissolved and the contents are released into membrane labeled $m+2n+2$. Otherwise, the formula is not satisfiable, and the membrane labeled $m+2n+1$ will not dissolve.

Quantifier stage

Q1 $[\ T\ ]_{m+2n+6i+1} \to T'$,
$[\ T\ ]_{m+2n+6i+2} \to T$,
$[\ T' \to \lambda\ ]_{m+2n+6i+2}$, $1 \leq 2i \leq m$.

The universal gate of the formula is simulated by dissolution of two membranes: this happens if and only if two copies of T are present. One copy dissolves membrane $m + 2n + 6i + 1$ and is erased while the other copy dissolves membrane $m + 2n + 6i + 2$ and sends one copy of T outside; otherwise the computation in this gate stops without sending any object out. Recall that membrane $m + 2n + 6i + 3$ has been erased by rule from group G3.

Q2 $[\ T\ ]_{m+2n+6i+4} \to T'$,
$[\ T'\ ]_{m+2n+6i+5} \to T$,
$[\ T \to \lambda\ ]_{m+2n+6i+5}$, $1 \leq 2i \leq m$.

The existential gate of the formula is simulated by dissolution of two membranes: this happens if and only at least one copy of T is present. One copy dissolves membrane $m + 2n + 6i + 4$ and then it also dissolves membrane $m + 2n + 6i + 2$, (thus sending one copy of T outside) while the other copy (if exists) is erased; if no copy of T is present, no rule is applied, so the gate sends nothing outside. Recall that membrane $m + 2n + 6i + 6$ has been erased by rule from group G3.

Q3 $[\ d_i \to d_{i+1}\ ]_{m+5n+1}$, $0 \leq i \leq 2m + 8n + 1$.

At the same time as the membrane with label $m + 5n + 1$ is dissolved (at step $2m + 8n + 1$), the object $d_{2m+8n+1}$ evolves to $d_{2m+8n+2}$, and then released to the membrane with label $m + 5n + 2$.

Output stage

O1 $[\ d_{2m+8n+2}\ ]_{m+5n+2} \to \texttt{yes}$.

O2 $[\ a\ ]_{m+8n+3} \to [\ \ ]_{m+5n+3} a$, $a \in \{\texttt{yes}, \texttt{no}\}$.

In the next two steps, the object `yes` is produced, and sent to the environment.

O3 $[\ d_{2m+8n+2}\ ]_{m+5n+1} \to \texttt{no}$.

O4 $[\ \texttt{no}\ ]_{m+5n+2} \to \texttt{no}$.

If the formula is not satisfiable, then the object $d_{2m+8n+1}$ remains in the membrane with label $m + 5n + 1$, which produces the object *no*, ejecting it into the membrane with label $m + 5n + 2$, then into the membrane with label $m + 5n + 3$, finally into the environment.

Therefore, in $2m + 8n + 3$ the system halts and sends into the environment one of the objects `yes, no`, indicating whether or not the formula φ^* is satisfiable.

It is easy to see that the system Π can be constructed in a polynomial time starting from numbers m, n, and this concludes the proof. [1] □

[1] The systems constructed above are deterministic.

It is possible to speed up the system; the present construction is made for an easier explanation: the stages do not overlap in time.

The only rules of type (c_0) in the system are O2, executed in the last step. Hence, these rules are not important for deciding whether φ^* is satisfied; they are only needed to send the answer out of the skin membrane.

This result can be contrasted to one from [100] as follows: we used membrane dissolution instead of polarization. One of the techniques used to achieve this goal is: instead of modifying the (polarization of the) membrane and checking it later, we use two membranes and control the time when the inner membrane is dissolved. In this case checking the membrane polarization is replaced by checking whether it exists, i.e., checking the membrane label.

This is used in the Assignments stage (truth-value objects influence the input objects, rules A1 and A3). In the Checking stage the dissolution picks one object c_i, performing the "if" behavior. In the Output stage the dissolution makes it possible to send exactly one of objects `yes` and `no` out. in the Quantifier stage OR and AND are implemented by counting until one or two by dissolution.

Another way the dissolution is used in the construction is to stop (by rules G3) the propagation of the non-elementary division (rules G4) from the elementary membranes outwards, to obtain the structure on Figure 4.8, because the rule is more restricted then in the case with polarizations.

From Theorem 2, since the complexity class $\mathbf{PMC}_{\mathcal{AM}^0(+d,+ne)}$ is closed under polynomial time reductions, we have the following result.

Corollary 4.2 $\mathbf{PSPACE} \subseteq \mathbf{PMC}_{\mathcal{AM}^0(+d,+ne)}$.

Conclusions In this section we gave a polynomial time and uniform solution of `QSAT`, a well-known **PSPACE**–complete problem, through a family of recognizer P systems using polarizationless active membranes, dissolution rules and division for elementary and non–elementary membranes. It remain as an open question if the division for non–elementary membranes can be removed. Our result thus presents an interesting counterpart of the result from [100], compared to which the polarizations have been replaced by membrane dissolution.

4.6 Minimal Parallelism

It is known that the satisfiability problem (`SAT`) can be efficiently solved by a uniform family of P systems with active membranes with two polarizations working in a maximally parallel way. We study P systems with active membranes without non-elementary membrane division, working in minimally parallel way. The main question we address is what number of polarizations is sufficient for an efficient computation depending on the types of rules used.

In particular, we show that it is enough to have four polarizations, sequential evolution rules changing polarizations, polarizationless non-elementary membrane division rules and polarizationless rules of sending an object out. The same problem is solved with the standard evolution rules, rules of sending an object out and polarizationless non-elementary membrane division rules, with six polarizations. It is open whether these numbers are optimal.

An interesting class of membrane systems are those with active membranes (see [253]), where membrane division can be used for solving computationally hard problems in polynomial time. Let us mention a few results: A *semi–uniform* **solution** to `SAT` using three polarizations and division for non-elementary membranes, [253]. A *polarizationless* solution, [108]. Using only division for elementary membranes, with three polarizations, [260].

A *uniform* solution, with elementary membrane division, [261]. Using only *two polarizations*, in a uniform way, with elementary membrane division, [59], [60]. Computational **completeness** of P systems with three polarizations and three membranes, [252]. Using only *two polarizations* and two membranes, [72], [71]. Using only *one membrane*, with two polarizations, [75], [74], [73]. *Polarizationless* systems are complete, with no known bound on the number of membranes, [17]. Solving SAT in a **minimally parallel** way, using non-elementary membrane division (replicating both objects and inner membranes), [152]. Avoiding polarizations by using rules *changing membrane labels*. Using (up to the best author's knowledge) either cooperative rules or non-elementary division as above, [199].

Given a P system, a rule and an object, whether this rule is applicable to this object in some membrane might depend on both membrane label (that usually cannot be changed) and membrane polarization. Essentially, the number of polarizations is the number of states that can be encoded directly on the membrane.

Minimal parallelism provides less synchronization between the objects, so one might expect the need of a stronger control, i.e., more polarizations. It is not difficult to construct the system in such a way that the rules are global (i.e., the membrane labels are not distinguished), most likely without adding additional polarizations. In this way the results on the number of polarizations can be reformulated in terms of number of membrane labels (in that case, the systems have no polarizations, but the rules may modify membrane labels).

4.6.1 With sequential polarization-changing evolution

The three size parameters of the SAT problem are the number m of clauses, the number n of variables and the total number l of occurrences of variables in clauses (clearly, $l \leq mn$: without restricting generality, we could assume that no variable appears in the same clause more than once, with or without negation).

Theorem 4.7 *A uniform family of confluent minimally parallel P systems with rules* $(a''_s), (c_0), (e_0)$ *can solve* SAT *with four polarizations in* $O(l(m+n))$ *number of steps.*

Proof. The main idea of the construction is to implement a maximally parallel step sequentially. For this, a "control" object will be changing the polarization, and then an input object or a clause object will be restoring it. Since the input is encoded in l objects, changing and restoring polarization will happen for l times, the counting is done by the "control" object.

Consider a propositional formula in the conjunctive normal form:

$$\begin{aligned} \beta &= C_1 \vee \cdots \vee C_m, \\ C_i &= y_{i,1} \wedge \cdots \wedge y_{i,l_i},\ 1 \leq i \leq m, \text{ where} \\ y_{i,k} &\in \{x_j, \neg x_j \mid 1 \leq j \leq n\},\ 1 \leq i \leq m, 1 \leq k \leq l_i, \\ l &= \sum_{i=1}^{m} l_i. \end{aligned}$$

Let us encode the instance of β in the alphabet $\Sigma(\langle n, m, l \rangle)$ by multisets X, X' of the clause-variable pairs such that the variable appears in the clause without negation, with negation or neither:

$$
\begin{aligned}
\Sigma(\langle n,m,l\rangle) &= \{v_{j,i,1,s} \mid 1 \le j \le m,\ 1 \le i \le n,\ 1 \le s \le 2\},\\
X &= \{(v_{j,i,1,1},1) \mid x_i \in \{y_{j,k} \mid 1 \le k \le l_j\},\\
&\quad 1 \le j \le m,\ 1 \le i \le n\},\\
X' &= \{(v_{j,i,1,2},1) \mid \neg x_i \in \{y_{j,k} \mid 1 \le k \le l_j\},\\
&\quad 1 \le j \le m,\ 1 \le i \le n\}.\\
&\quad \text{We construct the following P system:}\\
\Pi(\langle n,m,l\rangle) &= (O,H,E,[\,[\ ]_2^0[\ ]_3^0\,]_1^0, w_1,w_2,w_3,R),\ \text{with}\\
O &= \{v_{j,i,k,s} \mid 1 \le j \le m,\ 1 \le i \le n,\\
&\quad 1 \le k \le m+n+1,\ 1 \le s \le 4\}\\
&\cup\ \{d_{i,k} \mid 1 \le i \le m+n+1,\ 1 \le k \le 2l\}\\
&\cup\ \{t_{i,k}, f_{i,k} \mid 1 \le i \le n,\ 1 \le k \le l\}\\
&\cup\ \{d_i \mid 1 \le i \le m+n+1\} \cup \{S,Z,\texttt{yes},\texttt{no}\}\\
&\cup\ \{z_k \mid 1 \le k \le (4l+3)n + m(4l+1) + 2\}\\
w_1 &= \lambda,\ w_2 = d_1,\ w_3 = z_0,\ H = \{1,2,3\},\ E = \{0,1,2,3\},
\end{aligned}
$$

and the rules are listed below. The computation consists of three stages.

1. Producing 2^n membranes labeled 2, corresponding to the possible assignments of variables $x_1, \cdots, x_n$ and selecting clauses satisfied for every assignment (groups A, C).

2. Checking for all assignments whether all clauses are satisfied (groups B and D of rules).

3. Generating `yes` from the positive answer, and sending it to the environment. Generating `no` from the timeout (during the first two stages the number of steps is counted in the object in membrane with label 3) and sending it to the environment if there was no positive answer (groups E and F of rules).

Stage 1 consists of n cycles and stage 2 consists of m cycles. Each cycle's aim is to process all l objects, i.e., each object counts the number of cycles completed, and in the first stage the clauses are evaluated while in the second stage the presence of each clause is checked.

In the case of maximal parallelism, a cycle could be performed in a constant number of (actually, one or two) steps, while the minimal parallelism cannot guarantee that all objects are processed. The solution used here is the following. A cycle consists of marking (setting the last index to 3 or 4) all l objects one by one while performing the necessary operation, and then unmarking (setting the last index to 1 or 2) all of them. Marking or unmarking an object happens in two steps: the control object changes the polarization from 0 to 1, 2 (to mark) or to 3 (to unmark), and then one of the objects that has not yet been (un)marked is processed, resetting the polarization to 0.

Control objects in membrane 2: select clauses

A1 (for variable i: divide)
$[\ d_i\ ] \to [\ t_{i,0}\ ][\ f_{i,0}\ ]$, $1 \le i \le n$

A2 (process and mark all l objects)
$[\ t_{i,k-1}\ ]^0 \to [\ t_{i,k}\ ]^1$, $1 \le i \le n$, $1 \le k \le l$
$[\ f_{i,k-1}\ ]^0 \to [\ f_{i,k}\ ]^2$, $1 \le i \le n$, $1 \le k \le l$

A3 (prepare to unmark objects)
$[\ t_{i,l}\ ]^0 \to [\ d_{i,0}\ ]^0$, $1 \le i \le n$
$[\ f_{i,l}\ ]^0 \to [\ d_{i,0}\ ]^0$, $1 \le i \le n$

A4 (unmark all l objects)
$[\ d_{i,k-1}\ ]^0 \to [\ d_{i,k}\ ]^3$, $1 \le i \le n$, $1 \le k \le l$

A5 (switch to the next variable)
$[\ d_{i,l}\ ]^0 \to [\ d_{i+1}\ ]^0$, $1 \le i \le n$

Control objects in membrane 2: check clauses

B1 (test if clause i is satisfied)
$[\ d_{n+i}\ ]^0 \to [\ d_{n+i,1}\ ]^2$, $1 \le i \le m$

B2 (process and mark the other $l-1$ objects)
$[\ d_{n+i,k-1}\ ]^0 \to [\ d_{n+i,k}\ ]^1$, $1 \le i \le m$, $1 \le k \le l$

B3 (unmark all l objects)
$[\ d_{n+i,l+k-1}\ ]^0 \to [\ d_{n+i,l+k}\ ]^3$, $1 \le i \le m$, $1 \le k \le l$

B4 (switch to the next clause)
$[\ d_{n+i,2l}\ ]^0 \to [\ d_{n+i+1}\ ]^0$, $1 \le i \le m$

B5 (send a positive answer)
$[\ d_{m+n+1}\ ] \to [\ \ ]S$

Input objects in membrane 2: select clauses

C1 (mark an object)
$[\ v_{j,i,k,s}\ ]^p \to [\ v_{j,i,k+1,s+2}\ ]^0$,
$1 \le i \le m$, $1 \le j \le n$, $1 \le k \le m$, $k \ne m$, $1 \le s \le 2$, $1 \le p \le 2$

C2 (a true variable present without negation or a false variable present with negation satisfies the clause)
$[\ v_{j,i,i,s}\ ]^s \to [\ v_{j,i,i+1,3}\ ]^0$, $1 \le i \le m$, $1 \le j \le n$, $1 \le s \le 2$

C3 (a true variable present with negation or a false variable present without negation does not satisfy the clause)
$[\ v_{j,i,i,3-s}\ ]^s \to [\ v_{j,i,i+1,4}\ ]^0$, $1 \le i \le m$, $1 \le j \le n$, $1 \le s \le 2$

C4 (unmark an object)
$[\ v_{j,i,k,s+2}\ ]^3 \to [\ v_{j,i,k,s}\ ]^0$,
$1 \le i \le m$, $1 \le j \le n$, $2 \le k \le m+1$, $1 \le s \le 2$

Input objects in membrane 2: check clauses

D1 (check if the clause is satisfied at least by one variable)
$[\ v_{j,i,m+j,1}\ ]^2 \to [\ v_{j,i,k+1,3}\ ]^0$, $1 \le i \le m$, $1 \le j \le n$, $1 \le s \le 2$

D2 (mark an object)
$[\ v_{j,i,m+k,s}\ ]^1 \to [\ v_{j,i,k+1,s+2}\ ]^0$,
$1 \le i \le m$, $1 \le j \le n$, $1 \le k \le n$, $1 \le s \le 2$

D3 (unmark an object)
$[\ v_{j,i,m+k,s+2}\ ]^3 \to [\ v_{j,i,k,s}\ ]^0$,
$1 \le i \le m$, $1 \le j \le n$, $2 \le k \le n+1$, $1 \le s \le 2$

Control objects in membrane 3

E1 (count)
$[\ z_{k-1}\ ]^0 \to [\ z_k\ ]^0$, $1 \le k \le N = (4l+3)n + m(4l+1) + 2$

E2 (send time-out object)
$[\ z_N\ ] \to [\ ]Z$

Control objects in the skin membrane

F1 (a positive result generates the answer)
$[\ S\ ]^0 \to [\ \texttt{yes}\ ]^1$

F2 (without the positive answer, the time-out generates the negative answer)
$[\ Z\ ]^0 \to [\ \texttt{no}\ ]^0$

F3 (send the answer)
$[\ \texttt{yes}\ ] \to [\ ]\texttt{yes}$
$[\ \texttt{no}\ ] \to [\ ]\texttt{no}$

Let us now explain how the system works in more details.

Like the input objects, the control objects keep track of the number of cycles completed. The control object also remembers whether marking or unmarking takes place, as well as the number of objects already (un)marked. Moreover, the control object is responsible to pass the "right" information to the objects via polarization: in stage 1, 1 if the variable is true, and 2 if the variable is false; in stage 2, 1 if the clause is already found, and 2 if the clause is being checked for.

During the first stage, an object $v_{j,i,1,s}$ is transformed into $v_{j,i,n+1,t}$, where $t = 1$ if variable x_j satisfies clause C_i, or $t = 2$ if not. The change of the last index from s to t happens when the third index is equal to i. Notice that although only information about what clauses are satisfied seems to be necessary for checking if β is true for the given assignment of the variables, the information such as the number of cycles completed is kept for synchronization purposes, and the other objects are kept so that their total number remains l. The control object d_1 is transformed into d_{n+1}. Stage 1 takes $(4l+3)n$ steps.

If some clause is not satisfied, then the computation in the corresponding membrane is "stuck" with polarization 2. Otherwise, during the second stage an object $v_{j,i,n+1,t}$ is

transformed into $v_{j,i,n+m+1,t}$, while the control object d_{n+1} becomes d_{m+n+1}. Stage 2 takes $m(4l+1)$ steps, plus one extra step to send objects S to skin, if any.

After stage 2 is completed, one copy of S, if any, is transformed into `yes`, changing the polarization of the skin membrane. In the same time `yes`, if it has been produced, is sent out, object Z comes to the skin from region 3. If the polarization of the skin remained 0, Z changes to `no`, which is then sent out. Depending on the answer, stage 3 takes 2 or 4 steps. In either case, the result is sent out in the last step of the computation. □

Notice that membrane labels are not indicated in the rules. This means that the system is organized in such a way that the rules are *global*, i.e., the system would work equally well starting with the configuration $\mu = [\ w_1[\ w_2\ ]_1^0[\ w_3\ ]_1^0\ ]_1^0$, the labels were only given for the simplicity of explanation.

Using the remark in the end of the Introduction, we obtain

Corollary 4.3 *A uniform family of confluent polarizationless minimally parallel P systems with rules* $(a'_{0s}), (c_0), (e_0)$ *can solve* `SAT` *with membrane labels of four kinds.*

The statement follows directly from the possibility of rewriting a global rule $[\ a\ ]^e \to [\ u\ ]^{e'}$ of type (a''_s) in a rule $[\ a\ ]_e \to [\ u\ ]_{e'}$ of type (a'_{0s}) (which is polarizationless but is able to change the membrane label).

Using Rules (a) An informal idea of this section is to replace rules of type (a''_s) with rules (a) producing additional objects, and rules (c), sending an additional object out to change the polarization.

Theorem 4.8 *A uniform family of confluent minimally parallel P systems with rules* $(a), (c), (e_0)$ *can solve* `SAT` *with six polarizations in* $O(l(m+n))$ *number of steps.*

We give the proof of this theorem in Appendix A6.

The rules of the system in the proof above are also *global*, so we again obtain the following

Corollary 4.4 *A uniform family of confluent polarizationless minimally parallel P systems with rules* $(a), (c'_0), (e_0)$ *can solve* `SAT` *with membrane labels of six kinds.*

Conclusions Since changing membrane polarization controls what rules can be applied, the number of polarizations corresponds to the number of states of this control. Moreover, almost the only way the objects of the system may interact is via changing membrane polarization. Hence, the number of polarizations is a complexity measure deserving attention.

For maximal parallelism it has been proved that two polarizations are sufficient for both universality (with one membrane) and efficiency, while one-polarization systems are still universal (with elementary membrane division and membrane dissolution), but are conjectured not to be efficient.

We proved that efficient solutions of computationally hard problems by P systems with active membranes working in minimally parallel way can be constructed avoiding both cooperative rules and non-elementary membrane division, thus improving results from [152],

[199]. For this task, it is enough to have four polarizations, sequential evolution rules changing polarizations, polarizationless elementary membrane division rules and polarizationless rules of sending an object out. The standard evolution and send-out rules can be used with polarizationless elementary membrane division rules; in this case, six polarizations suffice.

The first construction is "almost" deterministic: the only choices the system can make in each cycle is the order in which the input systems are processed. The second construction exhibits a more asynchronous behavior of the input objects, which, depending on the chosen degree of parallelism, might speed up obtaining the positive answer, but less than by 20% [2]. In this case, controlling polarizations by evolution is still faster than by communication.

A number of interesting problems related to minimal parallelism remain open. For instance, is it possible to decrease the number of polarizations/labels? Moreover, it presents an interest to study other computational problems in the minimally-parallel setting, for instance, the computational power of P systems with one active membrane working in the minimally parallel way.

4.7 Energy Assigned to Membranes

We recall a variant of membrane systems introduced in [62], where the rules are directly assigned to membranes and, moreover, every membrane carries an energy value that can be changed during a computation by objects passing through the membrane. The result of a successful computation is considered to be the distribution of energy values carried by the membranes. We show that for systems working in the sequential mode with a kind of priority relation on the rules we already obtain universal computational power. When omitting the priority relation, we obtain a characterization of the family of Parikh sets of languages generated by context-free matrix grammars. On the other hand, when using the maximally parallel mode, we do not need a priority relation to obtain computational completeness. Finally, we introduce the corresponding model of tissue P systems with energy assigned to the membrane of each cell and objects moving from one cell to another one in the environment as well as being able to change the energy of a cell when entering or leaving the cell. In each derivation step, only one object may pass through the membrane of each cell. When using priorities on the rules in the sequential mode (where in each derivation step only one cell is affected) as well as without priorities in the maximally parallel mode (where in each derivation step all cells possible are affected) we again obtain computational completeness, whereas without priorities on the rules in the sequential mode we only get a characterization of the family of Parikh sets of languages generated by context-free matrix grammars.

Considering the energy balancing of processes in a cell first was investigated in [259] and then in [166]. There the energies of all rules to be used in a given step in a membrane are summed up; if the total amount of energies is positive ([259]) or within a given range ([166]), then this multiset of rules can be applied if it is maximal with this property.

[2] The maximal total number of steps needed is slightly over $10l(m+n)$; the fastest computation happens if rules C2 are executed in parallel for all input objects, as well as rules C4, D2, D3, saving $lm-1$, $lm-1$, $ln-1$, $ln-1$ steps, respectively. Their total is $2l(m+n)-4$, which is less than (but asymptotically equal to) 1/5 of the worst time.

We here take another approach. In contrast to most models of P systems where the evolution rules are placed within a region, in this section we consider membrane systems where the rules are directly assigned to the membranes (as, for example, in [175]); moreover, each membrane carries an energy value. As long as the energy value of a membrane is non-negative, by the application of a rule, singleton objects can be renamed while passing through membranes, thereby consuming or producing energy that is added to or subtracted from the energy value of the respective membrane. We also consider a kind of priority relation on the rules assigned to the membranes by choosing first the rules that changes the energy value of the membrane under consideration in a maximal way. The result of a successful computation is stored in the final energy values of the membranes. We consider these systems working in different modes of derivation: either the rules are applied in a sequential way (for sequential variants of P systems see, for example, [167] and [168]), i.e., in each derivation step only one membrane is affected by one singleton object entering or leaving this membrane, or else we work in the maximally parallel mode, i.e., to each membrane we have to apply a rule if possible thereby obeying to the constraints given by the priority relation for the rules.

We present the results for the sequential case in Appendix A7.

Trading Priorities for Maximal Parallelism In this section we show that even without the priority feature P systems with unit rules and energy assigned to membranes can obtain universal computational power, but only when working in the maximally parallel mode and not in the sequential mode as this was originally defined for P systems with unit rules and energy assigned to membranes in [173].

Theorem 4.9 *Any partial recursive function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$ ($\alpha > 0$, $\beta > 0$) can be computed by a P system with unit rules and energy assigned to membranes with (at most)* $\max\{\alpha, \beta\}+4$ *membranes working in the maximally parallel mode without priorities.*

The proof is quite long and technical, so we refer the reader to [62].

The following results are immediate consequences of Theorem 4.9 as Corollaries A7.1 and A7.2 were immediate consequences of Theorem A7.1; we therefore omit the proofs:

Corollary 4.5 *Each recursively enumerable set $L \subseteq PsRE(\beta)$ can be accepted by a P system with unit rules and energy assigned to membranes with (at most) $\beta + 4$ membranes in the maximally parallel mode without priorities on the rules.*

Again we want to point out that in the proofs of Theorem 4.9 and Corollary 4.5, the simulation of a deterministic register machine by a P systems with unit rules and energy assigned to membranes in the maximally parallel mode can be carried out in a deterministic way, i.e., for a given input only one computation (halting or not) exists. In the generative case (which corresponds to taking $\alpha = 0$ in the proof of Theorem 4.9), the non-determinism is an inherent feature (again we now have to use the non-deterministic variant of ADD-instructions), although the simulation of each single step of the underlying register machine is carried out in a deterministic way, too:

Corollary 4.6 *Each recursively enumerable set $L \subseteq PsRE(\alpha)$ can be generated by a P system with unit rules and energy assigned to cells with (at most) $\alpha + 4$ membranes in the maximally parallel mode without priorities on the rules.*

The Tissue-like Variant

We briefly recall from [62] the associated results in case when the communication across every edge of a graph is sequential[3].

Corollary 4.7 *Each partial recursive function* $f : \mathbb{N}^{\alpha} \rightarrow \mathbb{N}^{\beta}$ $(\alpha > 0,\ \beta > 0)$ *can be computed by a tissue-like P system with unit rules and energy assigned to cells with (at most)* $\max\{\alpha, \beta\} + 2$ *cells in the sequential mode with priorities on the rules.*

Corollary 4.8 *Each language in* $PsRE(d)$ *can be accepted/generated by a tissue-like P system with unit rules and energy assigned to cells with (at most)* $d+2$ *cells in the sequential mode with priorities on the rules.*

Corollary 4.9 *Each partial recursive function* $f : \mathbb{N}^{\alpha} \rightarrow \mathbb{N}^{\beta}$ $(\alpha > 0,\ \beta > 0)$ *can be computed by a tissue-like P system with unit rules and energy assigned to cells with (at most)* $\max\{\alpha, \beta\} + 3$ *cells in the maximally parallel mode without priorities on the rules.*

Corollary 4.10 *Each language in* $PsRE(d)$ *can be accepted/generated by a tissue-like P system with unit rules and energy assigned to cells with (at most)* $d+3$ *cells in the maximally parallel mode without priorities on the rules.*

Corollary 4.11 $PstPE_*(unit) = PsPE_*(unit) = PsMAT$.

$PstPE_*(unit)$ denotes the family of sets of Parikh vectors generated (or accepted) by tissue-like P systems with unit rules and energy assigned to cells without priorities, with arbitrary number of cells.

4.8 Energy Assigned to Regions

We investigate the computational power of energy-based P systems, a model of membrane systems where a fixed amount of energy is associated with each object and the rules transform single objects by adding or removing energy from them. We answer recently proposed open questions about the power of such systems without priorities associated to the rules, for both sequential and maximally parallel modes. We also conjecture that deterministic energy-based P systems are not computationally complete.

In this section we consider *energy-based P systems* [259], [217, 216, 215], a model of computation in the framework of Membrane Computing in which a given amount of energy

[3] We should like to point out that we have considered a quite restricted variant of tissue-like P systems, i.e., the cells only communicate with the environment. A more general model (e.g., compare with that one introduced in [173]) would also allow connections between cells themselves; in that way, we could change the energy value of two cells at the same time by the application of one unit rule, but then in the case of the maximally parallel derivation mode there might occur conflicts between different rules applicable at the same time to a specific cell, which in the maximal case now might have connections to the environment and every other cell. As computational completeness can be obtained quite easily already with the restricted model defined at the beginning of this section (with each cell being able to communicate only with the environment), we do not investigate the more general model in this section.

is associated to each object, and the energy manipulated during computations is taken into account by means of conservative rules.

Note there has been other attempts in the literature to incorporate certain conservation laws in membrane computing. One is purely communicative models, of which the most thoroughly studied is P systems with symport/antiport [247]. In these systems the computation is carried out by moving objects between the regions in groups. To reach computational completeness, the workspace is increased by bringing (some types of) objects from the environment, where they can be found in an unbounded supply. Another model is *conformon P systems* [187], where computations are performed by redistributing energy between objects, that can also be renamed and moved. A feature of these systems is that a different amount of energy may be embedded in the same object at different time steps. Yet another approach is to assign energy to membranes, as in P systems with Unit Rules and Energy assigned to Membranes (*UREM P systems*, for short) [62]. Here the computations are performed by rules renaming and moving an object across a membrane, possibly modifying the energy assigned to that membrane. It has been proved in [62] that UREM P systems working in the sequential mode characterize $PsMAT$, the family of Parikh sets generated by matrix grammars without appearance checking (and with *erasing rules*), and that their power is increased to $PsRE$ (the family of recursively enumerable Parikh sets) if priorities are assigned to the rules or the mode of applying the rules is changed to maximally parallel.

As stated above, in this section we consider *energy-based P systems*, in which energy is assigned to objects in a way that each object from the alphabet is assigned a specific value. Instances of a special symbol are used to denote *free energy units* occurring inside the regions of the system. The computations are carried out by rules renaming and possibly moving objects, which may consume or release free energy in the region, respecting the energy conservation law (that is, the total amount of energy associated with the objects that appear in the left side of a rule is the same as the energy occurring in the right side). The result of a computation may be interpreted in many ways: for example, as the amount of free energy units in a designated output region. Also for this model, to give the possibility to reach computational completeness it is necessary (but not sufficient, as we will see) that there may be an unbounded amount of free energy in (at least one) specified region of the system. In [215] it is proved that energy-based P systems working in the sequential way and using a form of local priorities associated to the rules are computationally complete. Without priorities, their behavior can be simulated by vector addition systems, and hence are not universal. However, in [215] the precise characterization of the computational power of energy-based P systems without priorities is left open. A related open question was whether energy-based P systems can reach computational completeness by working in the maximally parallel mode, without priorities, as it happens with UREM P systems [62].

In this section we answer these questions, by showing that the power of energy-based P systems containing infinite free energy and without priorities is exactly $PsMAT$ when working in the sequential mode, and $PsRE$ when working in the maximally parallel mode. Nonetheless we will end with another open question: what is the power of energy-based P systems under the restriction of determinism? We conjecture non-universality for this case.

Characterizing the Power of Energy-based P Systems In this section we characterize the computational power of energy-based P systems without priorities associated to the rules and with an unbounded amount of free energy units. As stated in the previous section, we use energy-based P systems as generating devices; the extension to the computing and accepting cases (concerning computational completeness) is easy to obtain.

We start with systems working in the sequential mode; with the next two theorems we prove that they characterize $PsMAT$. We first prove the inclusion $PsMAT \subseteq PsOP_*^{seq}(energy_*)$, obtained by simulating partially blind register machines.

Theorem 4.10 $PsOP_*^{seq}(energy_*) \supseteq PsMAT$.

Proof. Consider a partially blind register machine $M = (m, Q, I, q_0, q_f)$, generating β-dimensional vectors. We recall that we assume registers $\beta + 1, \cdots, m$ are empty in the final configuration of successful computations, corresponding to an implicit final zero-test.

We construct a corresponding energy-based P system Π_M, containing an infinite amount of free energy units in the skin, as follows:

$$
\begin{aligned}
\Pi &= (A, \varepsilon, \mu, e, w_s, \cdots, w_f, R_s, \cdots, R_f, i_0), \text{ where} \\
i_0 &= \{1, \cdots, \beta\} \text{ are the output membranes,} \\
A &= \{q, q', q'' \mid q \in Q\} \cup \{t_j, T_j \mid \beta + 1 \leq j \leq m\} \cup \{t, T, H, H'\}, \\
\varepsilon(q) &= 1,\ \varepsilon(q') = 2,\ \varepsilon(q'') = 0,\ q \in Q, \\
\varepsilon(t_j) &= 0,\ \varepsilon(T_j) = 2,\ \beta + 1 \leq j \leq m, \\
\varepsilon(t) &= 0,\ \varepsilon(T) = 1,\ \varepsilon(H) = 2m + 1,\ \varepsilon(H') = 2\beta + 2, \\
\mu &= [\,[\ ]_1 \cdots [\ ]_m [\ ]_f\,]_s, \\
w_s &= q_0,\ w_j = \lambda,\ 1 \leq j \leq m,\ w_f = t_{\beta+1} \cdots t_m T, \\
R_s &= \{q_1 e \to (q_1', in(j)) \mid (q_1 : [RjP], q_2, q_3) \in I\} \\
&\cup \{q_1 \to (q_1'', in(j))\, e \mid (q_1 : \langle RjZM \rangle, q_2) \in I\} \\
&\cup \{T \to (T, in(f)),\ l_h e^{2m} \to (H, in(f))\} \\
&\cup \{T_j \to (t, in(j))\, e^2 \mid \beta + 1 \leq j \leq m\}, \\
R_j &= \{q_1' \to (q_2, out)\, e,\ q_1' \to (q_3, out)\, e \mid (q_1 : [RjP], q_2, q_3) \in I\} \\
&\cup \{q_1'' e \to (q_2, out) \mid (q_1 : \langle RjZM \rangle, q_2) \in I\} \cup R_j',\ 1 \leq j \leq m, \\
R_j' &= \emptyset,\ 1 \leq j \leq \beta, \\
R_j' &= \{T \to te,\ te \to T\},\ \beta + 1 \leq j \leq m, \\
R_f &= \{T \to te,\ te \to T,\ H \to H' e^{2(m-\beta)-1}\} \\
&\cup \{t_j e^2 \to (T_j, out) \mid \beta + 1 \leq j \leq m\}.
\end{aligned}
$$

Note that if $\beta = m$, then we replace $H \to H' e^{-1}$ in R_f by $He \to H'$. The sets of rules R_j and R_j', $1 \leq j \leq m$, are both intended to be associated with region j, and hence should be joined. As explained below, the rules from R_j' are used to produce infinite loops $T \leftrightarrow te$ in the regions corresponding to non-output registers of M if such registers are non-empty when the computation halts.

The simulation consists of a few parts. Every object associated with an instruction label q_1 embeds 1 unit of energy. To simulate an increment instruction $(q_1 : [RjP], l_2, l_3)$, the

corresponding object q_1 consumes 1 more unit of energy, enters the region associated with register j as q_1', releases e there, and returns to the skin either as the object q_2 or as q_3 (the choice being made in a non-deterministic way), indicating the next instruction of M to be simulated. To simulate a decrement instruction $(q_1 : \langle RjZM\rangle, q_2)$, the corresponding object q_1 releases e in the skin and enters as q_1'' the region associated with register j. There it consumes 1 unit of energy, and returns to the skin as the object q_2 associated with the next instruction of M to be simulated. If the register was empty then this process is blocked.

Meanwhile, another process takes place in region f; the order of execution of the two processes is non-deterministic, but both must finish in order for the system to terminate the computation and produce a result. The aim of this latter process is indeed to make Π_M "compute" forever if the above simulation of the $\langle RZM\rangle$ instruction gets blocked when trying to decrement an empty register, so that no result is produced in this case. The process consists of object T cyclically releasing e and capturing it, generating a possibly infinite loop. The only way to stop the loop is to alter the free energy occurring in region m. This is done when the simulation of M is finished, leading to object H releasing $e^{2(m-\beta)-1}$. If $\beta = m$ this consumes 1 unit of energy, thus stopping the $T \leftrightarrow te$ loop. Otherwise it releases enough energy for objects t_j, $\beta + 1 \leq j \leq m$ to leave the region, and the last one of them stops the $T \leftrightarrow te$ loop. The objects t_j are used to ensure that registers $\beta + 1 \leq j \leq m$ are empty, otherwise causing a $T \leftrightarrow te$ loop in the corresponding region. □

The opposite inclusion, $PsOP_*^{seq}(energy_*) \subseteq PsMAT$, is proved by simulating energy-based P systems by matrix grammars.

Theorem 4.11 $PsOP_*^{seq}(energy_*) \subseteq PsMAT$.

Proof. Let $\Pi = (A, \varepsilon, \mu, e, w_1, \cdots, w_m, R_1, \cdots, R_m, i_{out})$ be an energy-based P system containing an infinite amount of free energy units that applies its rules in the sequential mode, and let $Ps(\Pi)$ be the set of vectors generated by Π. Each rule of Π can be simulated by a corresponding rewriting rule on multisets of object-region pairs, ignoring those pairs involving energy objects in the regions containing infinite free energy. Such a multiset rewriting rule can be written as a matrix, yielding a matrix grammar.

Formally, let $R_i' = \{a \to (b,p) \mid ae^k \to (b,p) \in R_i \text{ or } a \to (b,p)e^k \in R_i\}$ if region i contains infinite free energy, and $R_i' = R_i$ otherwise. We construct a matrix grammar $(G = N, T, S, M)$, where

$$\begin{aligned}
T &= \{(a,i)' \mid a \in A \cup \{e\},\ 1 \leq i \leq m\},\\
N &= \{(a,i) \mid a \in A \cup \{e\},\ 1 \leq i \leq m\} \cup \{S\},\\
M &= \{((a,i) \to (b,p), \underbrace{(e,i) \to \lambda, \cdots, (e,i) \to \lambda}_{k \text{ times}}) \mid ae^k \to (b,p) \in R_i'\}\\
&\cup \{((a,i) \to (b,p)(e,i)^k) \mid a \to (b,p)e^k \in R_i\}\\
&\cup \{(a,i) \to (a,i)' \mid (a,i) \in T\} \cup \{S \to u\},\ \text{where}
\end{aligned}$$

u is a string containing a copy of symbol (a,i) for every copy of object a in w_i.

Grammar G generates the representations of all reachable configurations of Π. We now proceed with specifying the halting condition by a regular language. The applicability of every rule $r \in R_i'$ corresponds to the presence of some $(a,i)'$ as well as k copies of $(e,i)'$ for

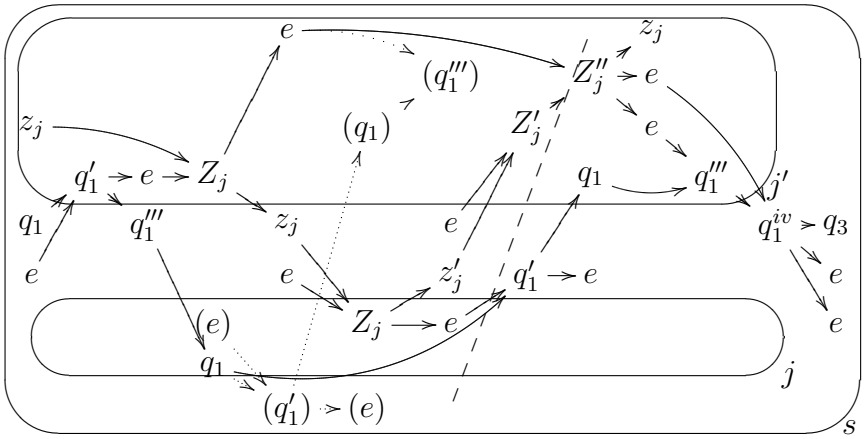


Figure 4.9: Simulation of the zero-test of $(q_1 : \langle RjZM\rangle, q_2, q_3)$ in Theorem 4.12

some $k \geq 0$: $App(r) = (a,i)'\sqcup\!\sqcup(e,i)'^k\sqcup\!\sqcup T^*$, which is regular. Then the halting condition is $H = T^* \setminus \bigcup_{r\in R_i',\ 1\leq i\leq m} App(r)$, which is also regular. Since the family of matrix languages is closed under intersections with regular languages, $L(G)\cap H$ is also a matrix language.

Finally, the result consists of symbols $(e,i)'$, $i \in i_{out}$, i.e., energy in the output region(s) of Π. We define a morphism h by $h((e,i)') = e_i$, $i \in i_{out}$, and $h((a,i)') = \lambda$ otherwise. Since matrix languages are closed under morphisms, also $h(L(G)\cap H) \in MAT$ holds. It follows from $Ps(h(L(G)\cup H)) = Ps(\Pi)$ that $Ps(\Pi) \in PsMAT$. □

By joining the two inclusions proved in Theorems 4.10 and 4.11 we obtain our characterization of $PsMAT$ by energy-based P systems working in the sequential mode with an unbounded amount of free energy and without priorities associated to the rules:

Corollary 4.12 *$PsOP_*^{seq}(energy_*) = PsMAT$.*

Running energy-based P systems in the maximally parallel mode allows them to reach computational completeness without using priorities, as shown in the next theorem.

Theorem 4.12 *$Ps(\beta)RE = Ps(\beta)OP_{\beta+6}(energy_*)$ for all integers $\beta \geq 1$.*

Proof. We start by noticing that the construction from Theorem 4.10 produces the same result when the P system works in the maximally parallel mode. Thus, it suffices to only add a simulation of zero-test instructions without disrupting the existing machinery. The new system non-deterministically chooses between decrement and zero-test, and blocks the simulation process if the zero-test fails.

We thus add membranes $[\ \ ]_{1'}, [\ \ ]_{2'}, \cdots, [\ \ ]_{m'}$ and the following sets of rules to the energy-based P system Π_M mentioned in the proof of Theorem 4.10:

$$\begin{aligned} R_s^0 \ = \ & \{1 : q_1 e \to (q_1', in(j')),\ 3a : q_1''' \to (q_1, in(j)),\\ & 5a : q_1' \to (q_1, in(j'))\, e,\ 5b : z_j e \to (Z_j, in(j))\, e,\\ & 7b : z_j' e \to (Z_j', in(j')),\ 12a : q_1^{(iv)} \to q_3 e^2\\ & \mid (q_1 : \langle RjZM\rangle, q_2, q_3) \in P\}, \end{aligned}$$

$$
\begin{aligned}
R_j^0 &= \{4a : q_1 e \to (q_1', out),\ 6b : Z_j \to (z_j', out)\, e \in R_j'' \\
&\quad \mid (q_1 : \langle RjZM \rangle, q_2, q_3) \in P\}, \\
R_{j'}^0 &= \{2 : q_1' \to (q_1''', out)\, e,\ 3b : z_j e \to Z_j,\ 4b : Z_j \to (z_j, out), \\
&\quad 8b : Z_j' e \to Z_j'',\ 9b : Z_j'' \to z_j e^2,\ 10a : q_1 e \to q_1''', \\
&\quad 11a : q_1''' e \to (q_1^{(iv)}, out) \mid (q_1 : \langle RjZM \rangle, q_2, q_3) \in P\}.
\end{aligned}
$$

The case of correct simulation of a zero-test is illustrated in Figure 4.9. The case when the register is not zero is shown by dotted lines and symbols in parentheses, and the computation stops before the dashed line. Indeed, if region j does not contain any object e, then the following sequence of multisets of rules is applied: 1, 2, $(3a, 3b)$, $(4a, 4b)$, $(5a, 5b)$, $6b$, $7b$, $8b$, $9b$, $10a$, $11a$, $12a$. In this way, l_1 is transformed to l_3 while the other objects used (energy and z_j) are reproduced. On the other hand, if region j contains some object e, corresponding to a non-zero value of the corresponding register, then the sequence of multisets of rules applied is 1, 2, $(3a, 3b)$, $(4a, 4b)$, $(5a, 5b)$, $(6b, 10a)$, $7b$, and the simulation process is blocked. We recall that blocking the simulation process leads to an infinite computation due to the $T \leftrightarrow te$ loop in region f. In words, because of free energy in region j, object l_1 left that region three steps earlier. As a result, instead of object Z_j' consuming 1 unit of energy and then releasing 2 units needed for l_1, the existing unit of energy has been consumed by l_1, leaving the computation unfinished.

The full P system, defined using components of the construction from Theorem 4.10, is given below:

$$
\begin{aligned}
\Pi'' &= (A'', \varepsilon'', \mu'', e, w_s'', \cdots, w_f'', R_s'', \cdots, R_f'', i_o ut), \text{ where} \\
A'' &= A \cup \{q''' \mid q \in Q\} \cup \{z_j, z_j', Z_j, Z_j', Z_j'' \mid 1 \le j \le m\}, \\
\varepsilon''(x) &= \varepsilon(x),\ \forall\, x \in A,\ \varepsilon(q''') = 1,\ \varepsilon(q^{(iv)}) = 2,\ \forall\, q \in Q, \\
\varepsilon''(z_j) &= \varepsilon''(z_j') = 0,\ \varepsilon''(Z_j) = \varepsilon''(Z_j') = 1,\ \varepsilon''(Z_j'') = 2,\ 1 \le j \le m, \\
\mu'' &= [\,[\ ]_1 \cdots [\ ]_m [\ ]_{1'} \cdots [\ ]_{m'} [\ ]_f\,]_s, \\
w_s'' &= w_s,\ w_j'' = w_j,\ 1 \le j \le m,\ w_f'' = w_f, \\
w_{j'}'' &= z_j,\ 1 \le j \le m, \\
R_s'' &= R_s \cup R_s^0,\ R_j'' = R_j \cup R_j^0,\ 1 \le j \le m, \\
R_{j'}'' &= R_{j'}^0,\ 1 \le j \le m,\ R_f'' = R_f.
\end{aligned}
$$

As we can see, system Π'' uses the skin membrane, one membrane to control the halting, and two membranes for each of the m registers, for a total of $2m + 2$ membranes.

Since it is known (see Proposition 1.3) that $m = \beta + 2$ registers suffice to generate any recursively enumerable set $L \subseteq Ps(\beta)RE$ of vectors of non-negative integers by non-deterministic register machines, we would obtain $2\beta + 6$ membranes. However, as recalled above, when using register machines as generating devices we can assume without loss of generality that only $[RP]$ instructions are applied to the output registers. So only one membrane is needed for each output register, thus reducing the total number of membranes to $\beta + 2(m - \beta) + 2 = \beta + 6$. □

By putting $\beta = 1$ in the above theorem we obtain a characterization of $\mathbb{N}RE$:

Corollary 4.13 $\mathbb{N}OP_7(energy_*) = \mathbb{N}RE$,

whereas if we make the union of all classes $Ps(\beta)OP_{\beta+6}(energy_*)$ for β ranging through the set of non-negative integers we obtain a characterization of $PsRE$:

Corollary 4.14 $PsOP_*(energy_*) = PsRE$.

As stated above, these results can be easily generalized to the cases in which energy-based P systems are used as accepting devices or as devices computing partial recursive functions. First of all note that the energy-based P systems built in the proofs of Theorems 4.10 and 4.12 can be easily modified to simulate deterministic register machines. Considering the computing case, we know from Proposition 1.1 that $m = \max\{\alpha, \beta\} + 2$ registers suffice to compute any partial recursive function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$. To simulate such a register machine we would obtain $2\max\{\alpha, \beta\} + 6$ membranes for the system Π'' built in the proof of Theorem 4.12. However, this number can be reduced to $\alpha + \max\{\alpha, \beta\} + 6$ by considering that:

- as stated above, we can assume that only ADD instructions are applied to the output registers. This means that only one membrane (instead of two) is needed to simulate the behavior of each output register;
- in general some input registers may also be used as output ones. Yet, any "primed" membrane j' associated with an input register, $1 \leq j' \leq \alpha$, cannot be used also as a membrane associated to an output register, due to object z_j residing in the membrane. Hence, with α inputs and β outputs we need α primed membranes plus $\max\{\alpha, \beta\}$ non-primed membranes. Adding two membranes for each of the 2 additional registers of M, plus membranes f and s, we obtain $\alpha + \max\{\alpha, \beta\} + 6$ membranes.

As particular cases, we need $2\alpha + 6$ membranes for the accepting case and $\beta + 6$ membranes for the generating case.

Conclusions and Discussion on Determinism In this section we have considered *energy-based P systems*, a model of membrane systems with energy assigned to objects. We have answered two questions about their computational power, and we have thus proved that it matches Parikh images of matrix languages when the rules of the P systems are applied in the sequential mode, whereas there is computational completeness in the maximally parallel case.

As a direction for future research, we propose the following problem: What is the computational power of *deterministic* energy-based P systems? We conjecture that they are not universal. The question originates from the fact that in [217, 216] energy-based P systems are used to simulate Fredkin gates and Fredkin circuits, respectively; however, the simulation is performed in a non-deterministic way, relying on the fact that sooner or later the simulation will choose the correct sequence of rules. Note that if the wrong rules are chosen the simulation is not aborted; the state of the system is "rolled back" so that a new non-deterministic choice can be made, hopefully the correct one. Clearly this situation could produce infinite loops; this is why one would like instead to have a *deterministic* simulation.

Here we can only give an *informal* justification for our conjecture. Notice that objects only interact indirectly, via releasing free energy units in a region or consuming them. Consider a dependency graph whose nodes are identified by object-region pairs. Two nodes are connected if the corresponding objects are present in the associated regions in some rule. A system is deterministic if no branching can be effectively used in its computations, so removing unusable rules would lead to a dependency graph where each node has out-degree at most one. Hence, any object occurring in the initial configuration of the system has some predetermined evolution path, and one of the following cases must happen: 1)the path is finite, and the object evolves until there are no associated rules, 2)the path leads to a cycle, and the object evolves forever (the computation yields no result), or 3)the evolution is "frozen" because there is not enough energy for the associated rule.

In energy-based P systems, the only way one object can influence the behavior of another object is by manipulating energy, leading to freezing or unfreezing the computational path of another object. There is no deterministic way to set an object to two different paths. If a "frozen" object receives enough energy to continue its evolution, then its computational path is the same as if it was never frozen.

So the information that can be passed from an object to another one is quite limited: giving the latter energy, as opposed to letting it freeze forever. However, every time this happens, some object must stop evolving forever. Since the initial number of objects is fixed and cannot increase, the communication complexity is bounded and this should imply non-universality.

However, even if deterministic energy-based P systems were not universal, they could nonetheless be able to simulate Fredkin gates. This should be doable if leaving some "garbage" into the system at the end of the computation is allowed. Indeed, the active objects could unfreeze the desired ones, producing the needed result. More difficult would be designing an energy-based P system that can be reused to simulate a Fredkin gate as many times as desired. We expect the reusable construction to be impossible, for the same reasons as exposed above.

4.9 Conclusions to Chapter 4

A simulation of exponential-space Turing machines by P systems with active membranes has been presented. The simulation can be carried out by a uniform family of polynomial-time P systems with active membranes operating in space $O(s(n) \log s(n))$, where $s(n)$ is the space required by the simulated Turing machine on inputs of length n. Since the converse result holds, the two classes of devices solve exactly the same decision problems when working withing an exponential space limit. The techniques employed here do not carry over to the simulation of superexponential space Turing machines, since they would require a super-polynomial number of subscripted objects in order to encode tape positions; this amount of objects (and their associated rules) cannot be constructed using a polynomial-time uniformity condition. Novel techniques will be probably needed in order to prove that the equivalence of Turing machines and P systems also holds for larger space bounds.

The computational completeness of P systems with active membranes without polariza-

tions has been established. Specifically, we have shown that RE is generated by P systems using four membranes and three labels or seven membranes and two labels in the initial configuration, where at most three objects are ever present in any halting computation. We have also proved that *deterministic* P systems with two polarizations and rules of types (a) and (c) accept $PsRE$ using *one membrane*. Moreover, we can require the rules to be global and rules of type (c) to be non-renaming.

Improving any complexity parameter greater than one (especially in the case of $*$) in any theorem is an open question. Moreover, the following issues are of interest: the power of *deterministic* P systems with membrane division (without polarizations, without changing labels, etc.); restricting the types of rules in Theorem 4.1; further restrictions causing a complexity trade-off; the generative power of P systems without polarizations and m membranes, $m = 1, 2, 3$; the generative power of one-membrane P systems with two polarizations and external output.

An algorithm has been given for deciding the NP-complete decision problem $\texttt{SAT}(n, m)$ by a uniform family of P system with active membranes in linear time (with respect to nm) with only two polarizations and rules of types (a), (c), and (e), of specific restrictive types. Various other restrictions are summarized in Corollary A4.1, followed by the discussion.

The question remains whether further or other restrictions, respectively, of the general form of these rules are possible. For instance, can the problem be solved using only rules of types (a), (c_{p0}), (e) (the rules of type (c) do not depend on the polarization and preserve it)? What about using only types (a_p), (c), (e) (the rules of type (a) do not depend on the polarization)? Another interesting question is to study systems with rules of types (a_u), (b), (c), (d), (e); such systems can only increase the number of objects via membrane division. What is their generative power? Are they efficient?

A solution has been presented to a known $\#\mathbf{P}$-complete problem of computing a permanent of a binary matrix by P systems with active membranes, namely with two polarizations and rules of object evolution, sending objects out and membrane division. The solution has been preceded by the framework that generalizes decisional problems to computing functions: now the answer is much more than one bit. This result suggests that P systems with active membranes without non–elementary membrane division still compute more than decisions of the problems in $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP}$.

The main result presented in Section 4.4 has been later developed by the Milano group, by implementing decoding also by P systems themselves without additional features. Hence, P systems with active membranes without non–elementary membrane division compute at least **PP**. A subsequent paper by the Milano group, contains an attempt to implement solutions to $\mathbf{P}^{\mathbf{PP}}$ problems, implementing **PP** oracles as subsystems of a P system.

A uniform polynomial-time solution of QSAT, a well-known **PSPACE**–complete problem has been given through a family of recognizer P systems using polarizationless active membranes, dissolution rules and division for elementary and non–elementary membranes. It remain as an open question if the division for non–elementary membranes can be removed. Our result thus presents an interesting counterpart of the result from [100], compared to which the polarizations have been replaced by membrane dissolution.

It has been showed that efficient solutions of computationally hard problems by P systems

with active membranes working in minimally parallel way can be constructed avoiding both cooperative rules and non-elementary membrane division. For this task, it is enough to have 4 polarizations, sequential evolution rules changing polarizations, polarizationless elementary membrane division rules and polarizationless rules of sending an object out. One can use the standard evolution and send-out rules, as well as polarizationless elementary membrane division rules; in this case, 6 polarizations suffice.

The first construction is "almost" deterministic: the only choices the system can make in each cycle is the order in which the input systems are processed. The second construction exhibits a more asynchronous behavior of the input objects, which, depending on the chosen degree of parallelism, might speed up obtaining the positive answer, but less than by 20%. In this case, controlling polarizations by evolution is still faster than controlling polarizations by communication.

A number of interesting problems related to minimal parallelism remain open. For instance, is it possible to decrease the number of polarizations/labels? Moreover, it presents an interest to study other computational problems in the minimally-parallel setting, for instance, the computational power of P systems with one active membrane working in the minimally parallel way.

The results on two models of P systems with energy have been presented. In the first case, energy is a kind of a membrane polarization that can take infinitely many values, but is used only by adding and subtracting one to/from it. In the second case, energy is associated to objects and regions, requiring also a conservativity condition.

For the latter case, called *energy-based P systems*, we have answered two questions about their computational power, proving that it matches Parikh images of matrix languages when the rules of the P systems are applied in the sequential mode, whereas there is computational completeness in the maximally parallel case.

As a direction for future research, we propose the following problem: What is the computational power of *deterministic* energy-based P systems? We conjecture that they are not universal.

However, even if deterministic energy-based P systems were not universal, they could nonetheless be able to simulate Fredkin gates. This should be doable if leaving some "garbage" into the system at the end of the computation is allowed. Indeed, the active objects could unfreeze the desired ones, producing the needed result. More difficult would be designing an energy-based P system that can be reused to simulate a Fredkin gate as many times as desired. We expect the reusable construction to be impossible, for the same reasons as exposed above.

Section 4.1 is based on publications [92] and [93]. Section 4.2 is based on publications [73], [17], [74] and [75], as well as [71] and [72]. Section 4.3 is based on publications [60], [59] and [72]. Section 4.4 is mainly based on publications [35] and [34]. Section 4.5 is based on publications [114], [115], [108] and [107]. Section 4.6 is based on publications [9], [8] and [10]. Sections 4.7 and 4.8 are based on publications [23], [24] and [62].

5. STRING-OBJECT MODELS

In this chapter we take strings instead of symbols as objects. Hence, we already have linear order as structure. We mostly consider very restricted variants of the string rewriting rules, such as symbol insertion, symbol deletion, symbol substitution (we speak about splicing in Section 5.4). It is particularly interesting how one can enforce interaction between different symbols of the string (and interaction between different strings in cases of splicing and later in Section 6.2).

In Section 5.1 we consider networks of evolutionary processors (NEPs), which are are distributed word rewriting systems, as language generators (these models have inspirations similar to P systems, but are not considered a part of membrane computing). Each node contains a set of words, a set of operations (typically insertion, deletion or rewriting of one symbol with another one), an input filter and an output filter. The purpose of this section is to overview existing models of NEPs, their variants and developments.

In particular, besides the basic model, hybrid networks of evolutionary processors (HNEPs) have been extensively studied. In HNEPs, operations application might be restricted to specific end of the string, but the filters are random-context conditions (they were regular in the basic model). We will also cover the literature on the so-called obligatory HNEPs, i.e., ones where the operations are obligatory: the string that cannot be rewritten is not preserved. Some specific aspects that we pay attention to are: computational universality and completeness, the topology of the underlying graph, the number of nodes, the power of filters.

In Section 5.2 we consider insertion-deletion P systems with priority of deletion over the insertion. We show that such systems with one symbol context-free insertion and deletion rules are able to generate Parikh sets of all recursively enumerable languages ($PsRE$). If one-symbol one-sided context is added to insertion or deletion rules, then all recursively enumerable languages can be generated. The same result holds if a deletion of two symbols is permitted. We also show that the priority relation is very important and in its absence the corresponding class of P systems is strictly included in the family of matrix languages (MAT).

In Section 5.3 we consider insertion-deletion P systems where the place where operations are applied is restricted to the ends of the string. We prove the computational completeness in case of priority of deletion over insertion. This result presents interest since the strings are controlled by a tree structure only, and because insertion and deletion of one symbol are the simplest string operations.

To obtain a simple proof, we use a new variant (CPM5) of circular Post machines (Turing machines moving one-way on a circular tape): those with instructions changing a state and

either reading one symbol or writing one symbol. CPM5 is a simple, yet useful tool. In the last part of the section, we return to the case without priorities. We give a lower bound on the power of such systems, which holds even for one-sided operations only.

In Section 5.4 we mention a group of results on splicing, an abstraction of a well-known biological operation, mostly the small universal systems based on splicing. Note that basic splicing has subregular behavior, so additional control is needed to make such systems possible.

One approach is to view splicing systems in a distributed framework, yielding splicing P systems. The second case is the so-called test tube systems based on splicing. The third way is to allow rules to change, in the framework of time-varying distributed H systems (TVDH systems). The fourth possibility is to consider a class of H systems which can be viewed as a counterpart of the matrix grammars in the regulated rewriting area.

Finally, we recall a related model, motivated by the gene assembly in ciliates. Ciliate operations are considered in membrane systems framework, establishing the computational completeness of the intermolecular model. One performs analysis of pointers in actual living ciliates, and one considers the problem of complexity of the graph-based model of gene assembly in ciliates. Moreover, ciliate operations provide a framework of describing the solution to the Hamiltonian Path Problem.

5.1 Networks of Evolutionary Processors

Networks of evolutionary processors (NEPs) are distributed word rewriting systems typically viewed as language generators. Each node contains a set of words, a set of operations (typically insertion, deletion or rewriting of one symbol with another one), an input filter and an output filter. The purpose of this section is to overview existing models of NEPs, their variants and developments.

In particular, besides the basic model, hybrid networks of evolutionary processors (HNEPs) have been extensively studied. In HNEPs, operations application might be restricted to specific end of the string, but the filters are random-context conditions (they were regular in the basic model). We will also cover the literature on the so-called obligatory HNEPs, i.e., ones where the operations are obligatory: the string that cannot be rewritten is not preserved.

Some specific aspects that we pay attention to are: computational universality and completeness, the topology of the underlying graph, the number of nodes, the power of filters.

Insertion, deletion, and substitution are fundamental operations in formal language theory, their power and limits have obtained much attention during the years. Due to their simplicity, language generating mechanisms based on these operations are of particular interest. *Networks of evolutionary processors* (NEPs, for short), introduced in [148], are proper examples for distributed variants of these constructs. In this case, an evolutionary processor (a rewriting system which is capable to perform an insertion, a deletion, and a substitution of a symbol) is located at every node of a virtual graph which may operate over sets or multisets of words. The system functions by rewriting the collections of words present at the nodes and then re-distributing the resulting strings according to a communication protocol

defined by a filtering mechanism. The language determined by the network is defined as the set of words which appear at some distinguished node in the course of the computation. These architectures also belong to models inspired by cell biology, since each processor represents a cell performing point mutations of DNA and controlling its passage inside and outside the cell through a filtering mechanism. The evolutionary processor corresponds to the cell, the generated word – to a DNA strand, and the operations insertion, deletion, and substitution of a symbol – to the point mutations. It is known that, by using an appropriate filtering mechanism, NEPs with a very small number of nodes are computationally complete computational devices, i.e. they are as powerful as the Turing machines (see, for example [103], [101], [102]).

Basic model Motivated by some models of massively parallel computer architectures, networks of language processors have been introduced in [160]. Such a network can be considered as a graph, where the nodes are sets of productions and at any moment of time a language is associated with a node. In a derivation step, any node derives from its language all possible words as its new language. In a communication step, any node sends those words to other nodes that satisfy an output condition given as a regular language, and any node takes those words sent by the other nodes that satisfy an input condition also given by a regular language. The language generated by a network of language processors consists of all (terminal) words which occur in the languages associated with a given node.

Inspired by biological processes, a special type of networks of language processors was introduced in [148], called networks with evolutionary processors, because the allowed productions model the point mutation known from biology. The sets of productions have to be substitutions of one letter by another letter or insertions of letters or deletion of letters; the nodes are then called substitution node or insertion node or deletion node, respectively. Results on networks of evolutionary processors can be found e. g. in [148], [147], [146], [103]. In [147] it was shown that networks of evolutionary processors are universal in that sense that they can generate any recursively enumerable language, and that networks with six nodes are sufficient to get all recursively enumerable languages. In [103] the latter result has been improved by showing that networks with three nodes are sufficient.

In [103] one presents the proof of the computational completeness with two nodes, additionally employing a morphism. In [55] one shows that NEPs with two nodes (one insertion node and one deletion node) generate all recursively enumerable languages (in intersection with a monoid), avoiding the need for a morphism. The same paper shows that insertion and substitution characterize context-sensitive languages, while deletion and substitution characterize finite languages.

Hybrid model Particularly interesting variants of these devices are the so-called *hybrid networks of evolutionary processors* (HNEPs), where each language processor performs only one of the above operations on a certain position of the words in that node. Furthermore, the filters are defined by some variants of random-context conditions, i.e., they check the presence/absence of certain symbols in the words. These constructs can be considered both language generating and accepting devices, i.e., generating HNEPs (GHNEPs) and accepting HNEPS (AHNEPs). The notion of an HNEP, as a language generating device, was

introduced in [229] and the concept of an AHNEP was defined in [224].

In [159] it was shown that, for an alphabet V, GHNEPs with $27 + 3 \cdot card(V)$ nodes are computationally complete. A significant improvement of the result can be found in [51], [52], where it was proved that GHNEPs with 10 nodes (irrespectively of the size of the alphabet) obtain the universal power. For accepting HNEPs, in [222] it was shown that for any recursively enumerable language there exists a recognizing AHNEP with 31 nodes; the result was improved in [221] where the number of necessary nodes was reduced to 24. Furthermore, in [221] the authors demonstrated a method to construct for any NP-language L an AHNEP with 24 nodes which decides L in polynomial time.

At last in [53] it was proved that any recursively enumerable language can be generated by a GHNEP having 7 nodes (thus, the result from [51], [52] is improved) and in [54] the same authors showed that any recursively enumerable language can be accepted by an AHNEP with 7 nodes (thus, the result from [221] is improved significantly). An improvement of the accepting result to 6 nodes has been obtained in [220], by simulating Tag systems. In [54] also it was showed that the families of GHNEPs and AHNEPs with 2 nodes are not computationally complete.

In [159] it was demonstrated that a GHNEP with one node can generate only regular language, while in [116] a precise form of the generated language was presented, also considering one case omitted in the previous proof. Tasks of characterization of languages generated by a GHNEP with two nodes and languages accepting by an AHNEP with two nodes are still open.

Obligatory operations A variant of HNEPs, called Obligatory HNEPs (OHNEP for short) was introduced in [29]. The differences between HNEP and OHNEP are the following:

1. in deletion and substitution: a node discards a string if no operations in the node are applicable to the string (in HNEP case, this string remains in the node),

2. the underlying graph is a directed graph (in HNEP case, this graph is undirected); this second difference disappears when we consider complete networks.

These differences make OHNEPs universal [29] with 1 operation per node, no filters and only left insertion and right deletion.

In [27] complete OHNEPs were considered, i.e., OHNEPs with complete underlying graph. One may now regard complete OHNEP as a set of very simple evolutionary processors "swimming in the environment" (i.e., once a string leaves a node, it is not essential for the rest of the computation which node it left). In [27] it is proved that the complete OHNEPs with very simple evolutionary processors, i.e., evolutionary processors with only one operation (obligatory deletion, obligatory substitution and insertion) and filters containing not more than 3 symbols are computationally complete. We recall that the filters are either single symbols or empty sets, while the **sum** of weights has been counted.

In [26] one considers OHNEPs without substitution. It is not difficult to notice that in complete OHNEPs without substitution there is no control on the number of insertion or deletion of terminal symbols (i.e., those symbols which appear in output words). Therefore, the definition of OHNEPs needed to be modified in order to increase their computational

power. In [26] one shows that it is possible to avoid substitution using modified operations of insertion and deletion in evolutionary processors similar to "matrix" rules in formal language theory. By using such techniques a small universal complete OHNEP with 182 nodes without substitution is constructed.

Several open questions were posed in [26], in particular the question about the minimal total complexity of filters of evolutionary processor in computationally complete OHNEPs and the question about universal complete OHNEP without substitution with the minimal number of nodes. In [25] one considers a model of OHNEP allowing the use of all three molecular operations: insertion, deletion and substitution, and provides a very unexpected result. OHNEPs are computationally complete even if the total power of the filters of each node does not exceed 1! This means that in any node, all four filters are empty, except possibly one, being a single symbol.

We would like to point out that there is no interaction, direct or indirect, between the words of the network. Hence, the generated language is a union of languages, generated by the same system, but starting with only one word.

As for the replication, i.e., the possibility of applying multiple rules or the same rule in multiple ways, producing many words from one word, this could be viewed as a non-deterministic evolution of *one* word. In this case, distributivity simply means assigning a state to the word. Summing up, the language generated by a parallel deterministic word rewriting system may be viewed as a (union of) language(s) generated by a non-deterministic one-word rewriting system with states (without any other parallelism or distributivity).

Furthermore, the nature of the model (except the obligatory variant) leads to many cases of the "shadow" computations, in the following sense. If one carefully considers the definitions, and constructs a faithful simulation of the model, one would notice that a lot of computation in the system consists of repeatedly recomputing the same steps. This is due to the fact that if some operation $\pi \in Sub_V$ or $\rho \in Del_V$ of a node is not applicable to some word w in that node, the result is w. Taking the union over all operations of a node yields a set containing w, even if some other operation was applicable to w. Clearly, in the next step, the words obtainable from w in the same node will be recomputed. However, the system is deterministic, so nothing *new* is obtained in this way.

A careful examination reveals that, in some circumstances, the shadow computations can be avoided, modifying the definition while yielding the same generated language! Indeed, preserving w is useless (everything that is possible to derive from w in that node is derived immediately) unless w exits the node. However, at least in the case of complete networks, if w enters a node and exits it unchanged, this does not do anything new either (if w is an initial word, it can be copied to all nodes that it can reach unchanged in communication step).

The above reason lets us claim that, e.g., any result for complete OHNEPs holds also for the usual complete HNEPs, and the associated computational burden of the simulation may be greatly reduced. If the network is not complete, then a heuristic still may be used by a simulator, by preserving unchanged words only in case if they actually move from a node into some different node.

5.1.1 NEPs with two nodes

Theorem 5.1 *For any recursively enumerable language L, there are a set T and a network N of evolutionary processors with exactly one insertion node and exactly one deletion node such that $L = L(N) \cap T^*$.[55]*

Proof. (sketch) We consider a type-0 grammar $G = (N, T, P, S)$ with $L(G) = L$. Then all rules of P have the form $u \to v$ with $u \in N^+$ and $v \in (N \cup T)^*$. Let $X = N \cup T, X' = \{a, a' \mid a \in N \cup T\}$, $T' = \{a, a' \mid a \in T\}$ and $P' = \{p_i \mid p \in P, 1 \leq i \leq 4\}$. We define a morphism $\mu : X^* \to (X')^*$ by $\mu(a) = aa'$ for $a \in X$ and set $W = \{\mu(w) \mid w \in X^*\}$. We construct the following network $\mathcal{N} = (V, (M_1, A_1, I_1, O_1), (M_2, A_2, I_2, O_2), E, 2)$ of evolutionary processors with

$$
\begin{aligned}
V &= P' \cup X', \\
M_1 &= \{\lambda \to p_i \mid p_i \in P',\ 1 \leq i \leq 4\} \cup \{\lambda \to a \mid a \in X'\}, \\
A_1 &= \{\mu(S)\},\ I_1 = W \setminus (T')^*,\ O_1 = V * \backslash (WR_{1,1}W), \\
M_2 &= \{p_i \to \lambda \mid p_i \in P',\ 1 \leq i \leq 4\} \cup \{a \to \lambda \mid a \in X' \setminus T\}, \\
A_2 &= \emptyset,\ I_2 = WR_{1,2}W,\ O_2 = V^* \setminus (WR_{2,2}W \cup (T')^*), \\
E &= \{(1,2),(2,1)\},\ \text{where} \\
R_{1,1} &= \bigcup_{p:u\to v \in P} (\{p_1\mu(u), p_1\mu(u)p_3, p_1p_2\mu(u)p_3, p_1p_2\mu(u)p_3p_4\} \\
&\cup\ \{p_1p_2\mu(u)\}PPref(\mu(v))\{p_3p_4\}) \\
&\quad \backslash \{p_1p_2\mu(u)p_3p_4 \mid p : u \to v \in P\}, \\
R_{1,2} &= \{p_1p_2\mu(uv)p_3p_4 \mid p : u \to v \in P\}, \\
R_{2,2} &= \bigcup_{p:u\to v \in P} (\{p_1p_2\}PSuf(\mu(u))\{\mu(v)p_3p_4\} \\
&\cup\ \{p_2\mu(v)p_3p_4, p_2\mu(v)p_4, \mu(v)p_4\}).
\end{aligned}
$$

The output and input filters are defined in order to remove the garbage and communicate the strings that should change the type of operation, keeping only the strings that should continue to evolve by operations of the same type. Since the morphism $\mu(a) = aa'$ is introduced, the strings obtained by applying rules to the left or to the right of the place of application of the current rule are no longer kept in the node by the filter, and are not accepted by either node (recall that $W = aa'|a \in \{N \cup T^*)$, so they leave the system. Claim: $L(\mathcal{N}) \setminus T^* = L$. The "correct" simulation of an application of a production $p : a_1 \cdots a_s \to b_1 \cdots b_t$ to a sentential form $\alpha a_1 \cdots a_s \beta$ and with $x = \mu(\alpha)$ and $y = \mu(\alpha)$ has the following form. In N_1 we have

$$
\begin{aligned}
xa_1a_1' \cdots a_sa_s'y &\Rightarrow^{\lambda\to p_1} xp_1a_1a_1' \cdots a_sa_s'y \\
\Rightarrow^{\lambda\to p_3} xp_1a_1a_1' \cdots a_sa_s'p_3y &\Rightarrow^{\lambda\to p_2} xp_1p_2a_1a_1' \cdots a_sa_s'p_3y \\
\Rightarrow^{\lambda\to p_4} xp_1p_2a_1a_1' \cdots a_sa_s'p_3p_4y &\Rightarrow^{\lambda\to b_1} xp_1p_2a_1a_1' \cdots a_sa_s'b_1p_3p_4y \\
\Rightarrow^{\lambda\to b_1'} xp_1p_2a_1a_1' \cdots a_sa_s'b_1b_1'p_3p_4y &\Rightarrow^{*} xp_1p_2a_1a_1' \cdots a_sa_s'b_1b_1' \cdots b_tp_3p_4y \\
\Rightarrow^{\lambda\to b_t'} & xp_1p_2a_1a_1' \cdots a_sa_s'b_1b_1' \cdots b_tb_t'p_3p_4y,
\end{aligned}
$$

and in N_2 we have $xp_1p_2a_1a'_1 \cdots a_sa'_sb_1b'_1 \cdots b_tb'_tp_3p_4y \Rightarrow^{a_1 \to \lambda}$

$$
\begin{array}{lll}
xp_1p_2a'_1 \cdots a_sa'_sb_1b'_1 \cdots b_tb'_tp_3p_4y & & \Rightarrow^{a'_1 \to \lambda} \\
xp_1p_2a_2 \cdots a_sa'_sb_1b'_1 \cdots b_tb'_tp_3p_4y & \Rightarrow^* & xp_1p_2a_sa'_sb_1b'_1 \cdots b_tb'_tp_3p_4y \\
\Rightarrow^{a_s \to \lambda} xp_1p_2a'_sb_1b'_1 \cdots b_tb'_tp_3p_4y & \Rightarrow^{a'_s \to \lambda} & xp_1p_2b_1b'_1 \cdots b_tb'_tp_3p_4y \\
\Rightarrow^{p_1 \to \lambda} xp_2b_1b'_1 \cdots b_tb'_tp_3p_4y & \Rightarrow^{p_3 \to \lambda} & xp_2b_1b'_1 \cdots b_tb'_tp_4y \\
\Rightarrow^{p_2 \to \lambda} xb_1b'_1 \cdots b_tb'_tp_4y & \Rightarrow^{p_4 \to \lambda} & xb_1b'_1 \cdots b_tb'_ty.
\end{array}
$$

Notice that if a production can be applied to the same sentential form in different ways (multiple productions and/or multiple places to apply them), then the corresponding number of strings is produced in the first step (inserting marker p_1 associated to the production, to the left of the application place). The rest of the simulation is "deterministic" in the following sense: starting from $xp_1a_1a'_1 \cdots a_sa'_sy$, the result $xb_1b'_1 \cdots b_tb'_ty$ is obtained according to the derivations above, while all other strings are discarded. The strings that leave one node and enter another one belong to the sets $O_1 \setminus I_2 = I_2$ and $O_2 \setminus I_1 = I_1$. All other strings that leave a node do not enter anywhere. With $p \in P$, $a \in X'$ and $A \in X' \setminus T$, Table 5.1 illustrates the behavior of a string (the numbers give the situation which is obtained by using the rule in question and n/a refers to non-applicability of the rule).

Table 5.1: Strings in 2-node NEP

n	Shape in $N1$	$\lambda \to p_1$	$\lambda \to p_3$	$\lambda \to p_2$	$\lambda \to p_4$	$\lambda \to a$
1	W	2	out	out	out	out
2	$Wp_1\mu(u)W$out	3	out	out	out	
3	$Wp_1\mu(u)p_3W$	out	out	4	out	out
4	$Wp_1p_2\mu(u)p_3W$	out	out	out	5	out
5	$Wp_1p_2\mu(u)\cdot$ $PPref(\mu(v))p_3p_4W$	out	out	out	out	5,6
n	Shape in $N2$	$p_1 \to \lambda$	$p_3 \to \lambda$	$p_2 \to \lambda$	$p_4 \to \lambda$	$A \to \lambda$
6	$Wp_1p_2\cdot$ $NSuf(\mu(u))\mu(v)p_3p_4W$	out	out	out	out	6,7
7	$Wp_1p_2\mu(v)p_3p_4W$	8	out	out	out	out
8	$Wp_2\mu(v)p_3p_4W$	n/a	9	out	out	out
9	$Wp_2\mu(v)p_4W$	n/a	n/a	10	out	out
10	$W\mu(v)p_4W$	n/a	n/a	n/a	1,11	out
11	$(T')^*$	n/a	n/a	n/a	n/a	11

Table 5.1 illustrates the fact that if a symbol is inserted or deleted in a way that does not follow the "correct" simulation, than the string leaves the system. Finally, consider $L(\mathcal{N}) \cap (T')^*$. It is the set of all strings obtained in N_2 without nonterminal symbols, without markers and without pre-terminals (i. e., primed versions of terminals). Hence, all of them are obtained from shape 5 of N_2 by deleting the marker p_4, reaching shape 6 if the string only has terminals and pre-terminals. It is easy to see that in several computation steps all pre-terminal symbols will be deleted. This exactly corresponds to the set of terminal strings w produced by the underlying grammar G, all letters being represented by a double repetition, i. e., encoded by μ. Such strings remain in N_2 and all pre-terminals are deleted, obtaining w from $\mu(w)$. □

5.1.2 HNEPs with one node

The following theorem states the regularity result for GHNEPs with one node. Although this has already been stated in [159], their proof is certainly incomplete. They stated that while GHNEPs without insertion only generate finite languages, GHNEPs with one insertion node only generate languages I^*C_0, C_0I^*, $C_0 \sqcup\!\sqcup I^*$, for the mode $l, r, *$, respectively. In the theorem below we present a precise characterization of languages generated by GHNEP with one node and consider the case omitted in [159], when the underlying graph G has a loop.

Theorem 5.2 *One-node GHNEPs only generate regular languages.[116]*

Proof. As finite languages are regular, the statement holds for GHNEPs without insertion nodes. We now proceed with the case of one insertion node. Consider such a GHNEP $\Gamma = (V, G, N_1, C_0, \alpha, \beta, 1)$, where

$$N_1 = (M, PI, FI, PO, FO).$$

Let us introduce a few notations. Inserting a symbol from I in a language C yields a language $\mathsf{ins}_I(C)$. Depending on whether $\alpha = l$, $\alpha = r$ or $\alpha = *$, $\mathsf{ins}_I(C)$ is one of IC, CI, $C \sqcup\!\sqcup I$, respectively. For inserting an arbitrary number of symbols from a set I in a language C, $\mathsf{ins}^*_I(C)$ is one of I^*C, CI^*, $C \sqcup\!\sqcup I^*$. Clearly, ins^*_I preserves regularity.

We denote the set of symbols inserted in N_1 by $I = \{a \mid \lambda \to a \in M\}$. The configuration of N_1 after one step is $C_1 = \mathsf{ins}_I(C_0)$. Assume that $\beta = 2$ (a case, when $\beta = 1$, can be considered analogously), then the conditions of passing permitting and forbidding output filter can be specified by regular languages $\pi = V^*POV^*$ and $\varphi = (V - FO)^*$, respectively. For instance, the set of words of C_1 that pass the forbidding output filter but do not pass the forbidding input filter is $C_1' = C_1 \cap \varphi \setminus \pi$. Notice that inserting symbols that belong to neither PO nor FO does not change the behavior of the filters; we denote the corresponding language by $B = \mathsf{ins}^*_{I \setminus (PO \cup FO)}(C_1)$.

Consider the case when the graph G consists of one node and no edges. Then, Γ generates the following language

$$\begin{aligned} L_1 = L_1(\Gamma) &= C_0 \cup C_1 \cup \mathsf{ins}^*_I(C_1 \setminus \varphi) \cup B \\ &\quad \cup \mathsf{ins}_{I \cap PO \setminus FO}(B) \cup \mathsf{ins}^*_I(\mathsf{ins}_{I \cap FO}(B)), \qquad (5.1) \\ B &= \mathsf{ins}^*_{I \setminus (PO \cup FO)}(C_1), \\ C_1 &= \mathsf{ins}_I(C_0). \end{aligned}$$

Indeed, this is a union of six languages:

1. initial configuration,
2. configuration after one insertion,
3. all words that can be obtained from a word from C_1 if it is trapped in N_1 by the forbidding filter,
4. B represents the words that pass the forbidding filter but not the permitting filter,

5. words obtained by inserting one permitting and not forbidden symbol into B, and

6. words obtained by inserting one forbidden symbol into B, and then by arbitrary insertions.

Consider the case when the graph G has a loop. The set of words leaving the node (for the first time) is $D = (C_1 \cap \varphi \cap \pi) \cap \mathsf{ins}_{I \cap PO \setminus FO}(B)$. The conditions of the permitting and forbidding input filters can be specified by regular languages $\pi' = V^*PIV^*$ and $\varphi' = (V - FI)^*$, respectively. Some of words from D return to N_1, namely $D \cap \pi' \cap \varphi'$. Notice that further insertion of symbols that belong neither to FO nor to FI causes the words to continuously exit and reeenter N_1. The associated language is $B' = \mathsf{ins}^*_{I \setminus (FO \cup FI)}(D \cap \pi' \cap \varphi')$. Finally, we give the complete presentation of the language generated by Γ in this case:

$$\begin{aligned} L_1' = L_1(\Gamma) &= L_1 \cup B' \cup \mathsf{ins}^*_I(\mathsf{ins}_{I \cap FO}(B')) \cup \mathsf{ins}_{I \cap FI \setminus FO}(B'), \\ B' &= \mathsf{ins}^*_{I \setminus (FO \cup FI)}(D \cap \pi' \cap \varphi'), \\ D &= (C_1 \cap \varphi \cap \pi) \cap \mathsf{ins}_{I \cap PO \setminus FO}(B), \\ C_1 &= \mathsf{ins}_I(C_0). \end{aligned} \tag{5.2}$$

Indeed, this is a union of four languages:

1. words that never reenter N_1, as in the case when G has no edges,

2. B' represents the words that once leave and reenter N_1, and keep doing so after subsequent insertions,

3. words obtained by inserting a symbol from FO into B', and then by arbitrary insertions,

4. words obtained by inserting a symbol from $FI \setminus FO$ into B'. □

5.1.3 HNEPs with 7 nodes

Theorem 5.3 *Any recursively enumerable language can be generated by a complete HNEP of size 7. [53], [54]*

Proof. Let $L \subseteq T^*$ be a language generated by a type-0 grammar $G = (N, T, S, P)$ in Kuroda normal form.

We construct a complete HNEP $\Gamma = (V, H, \mathcal{N}, C_0, \alpha, \beta, 7)$ of size 7 which simulates the derivations in G and only that, by using the so-called rotate-and-simulate method. The rotate-and-simulate method means that the words in the nodes are involved in either the rotation of their leftmost symbol (the leftmost symbol of the word is moved to the end of the word) or the simulation of a rule of P. In order to indicate the end of the word when rotating its symbols and thus to guarantee the correct simulation, a marker symbol, #, different from any element of $(N \cup T)$ is introduced. Let $N \cup T \cup \{\#\} = A = \{A_1, A_2, \dots A_n\}$, $I = \{1, 2, \dots, n\}$, $I' = \{1, 2, \dots, n-1\}$, $I'' = \{2, 3 \dots, n\}$, $I_0 = \{0, 1, 2, \dots, n\}$, $I_0' = \{0, 1, 2, \dots, n-1\}$, $B_0 = \{B_{j,0} \mid j \in I\}$, $B_0' = \{B_{j,0}' \mid j \in I\}$, $\# = A_n$, $T' = T \cup \#$. Let us define the alphabet $V = A \cup B \cup B' \cup C \cup C' \cup D \cup D' \cup E \cup E' \cup F \cup G \cup \{\lambda'\}$ of Γ

Table 5.2: A universal HNEP with 7 nodes.

$N, \alpha, \beta, C_0,$	M, PI, FI, PO, FO
$1, *, (2),$ $\{A_1 B_{n,0}\}$	$\{\mathbf{1.1} : A_i \to C'_i \mid i \in I\} \cup \{\mathbf{1.2} : A_i \to \lambda' \mid i \in I',\ A_i \to \lambda\} \cup$ $\{\mathbf{1.3} : B_{j,0} \to B_{s,0} \mid A_j \to A_s,\ j, s \in I'\} \{\mathbf{1.4} : C_i \to C'_{i-1},\ \mathbf{1.5} : B_{j,0} \to B'_{j,0},$ $\mathbf{1.6} : B_{j,k} \to B'_{j,k+1} \mid i \in I'',\ j \in I,\ k \in I'\} \cup \{\mathbf{1.7} : C_1 \to \lambda'\} \cup$ $\{\mathbf{1.8} : E'_{j,k} \to E_{j,k-1},\ \mathbf{1.9} : D'_i \to D_{i+1},\ \mathbf{1.10} : E'_{j,1} \to F_j \mid i \in I', j \in I, k \in I''\}$ $PI = \{A_n, B_{n,0}\} \cup C \cup E',\ FI = C' \cup E \cup D \cup F \cup G \cup \{\lambda'\}$ $PO = C' \cup B' \cup D \cup F \cup \{\lambda'\},\ FO = B \cup C \cup D' \cup E'$
$2, *, (2), \emptyset$	$\{\mathbf{2.1} : C'_i \to C_{i-1},\ \mathbf{2.2} : B'_{j,k} \to B_{j,k+1} \mid i \in I'',\ j \in I,\ k \in I'_0\} \cup \{\mathbf{2.3} : C'_1 \to \lambda'\} \cup$ $\{\mathbf{2.4} : E_{j,k} \to E'_{j,k-1},\ \mathbf{2.5} : D_i \to D'_{i+1},\ \mathbf{2.6} : E_{j,1} \to F_j \mid i \in I'_0, j \in I, k \in I''\} \cup$ $\{\mathbf{2.7} : A_n \to \lambda'\} \cup \{\mathbf{2.8} : B_{j,0} \to A_j \mid A_j \in T\}$ $PI = \{B_{j,0} \mid A_j \in T\} \cup C' \cup E,\ FI = \{B \setminus B_{j,0} \mid A_j \in T\} \cup C \cup\ D' \cup E' \cup F \cup G \cup \{\lambda'\}$ $PO = C \cup D' \cup F \cup \{\lambda'\},\ FO = \{B_{j,0} \mid A_j \in T\} \cup B' \cup C' \cup\ E \cup D$
$3, r, (2), \emptyset$	$\{\mathbf{3.1} : \lambda \to D_0\}$ $PI = B \setminus B_0 \cup B' \setminus B'_0 \cup G,\ FI = C \cup C' \cup B_0 \cup \{D_0\},\ PO = \{D_0\},\ FO = \emptyset$
$4, *, (2), \emptyset$	$\{\mathbf{4.1} : B_{j,k} \to E_{j,k},\ \mathbf{4.2} : B'_{j,k} \to E_{j,k} \mid j, k \in I\} \cup \{\mathbf{4.3} : B_{j,k} \to E_{s,t},$ $\mathbf{4.4} : B'_{j,k} \to E_{s,t} \mid j, k, s, t \in I', A_j A_k \to A_s A_t\} \cup \{\mathbf{4.5} : G_{j,k} \to E_{j,k} \mid j, k \in I'\}$ $PI = \{D_0\},\ FI = E,\ PO = E,\ FO = B \cup B' \cup G$
$5, *, (2), \emptyset$	$\{\mathbf{5.1} : D_j \to B_{j,0},\ \mathbf{5.2} : D'_j \to B_{j,0} \mid j \in I\} \cup \{\mathbf{5.3} : F_j \to A_j \mid j \in I\} \cup$ $\{\mathbf{5.4} : D_j \to G_{s,t},\ \mathbf{5.5} : D'_j \to G_{s,t} \mid A_j \to A_s A_t, j, s, t \in I'\}$ $PI = D \setminus \{D_0\} \cup D',\ FI = E \cup E' \cup \{D_0\} \cup C \cup C' \cup \{\lambda'\}\ PO = \emptyset,\ FO = D \cup D' \cup F$
$6, l, (2), \emptyset$	$\{\mathbf{6.1} : \lambda' \to \lambda\}$ $PI = \{\lambda'\},\ FI = B \setminus B_0 \cup B' \cup C \cup C' \cup F \cup (D \setminus \{D_0\}),\ PO = \emptyset,\ FO = \{\lambda'\}$
$7, *, (2), \emptyset$	$\emptyset$ $PI = T,\ FI = V \setminus T,\ PO = \emptyset,\ FO = T$

as follows: $B = \{B_{i,j} \mid i \in I,\ j \in I_0\}$, $B' = \{B'_{i,j} \mid i \in I, j \in I_0\}$, $C = \{C_i \mid i \in I\}$, $C' = \{C'_i \mid i \in I\}, D = \{D_i \mid i \in I_0\}$, $D' = \{D'_i \mid i \in I\}$, $E = \{E_{i,j} \mid i, j \in I\}$, $E' = \{E'_{i,j} \mid i, j \in I\}$, $F = \{F_j \mid j \in I\}$, $G = \{G_{i,j} \mid i, j \in I\}$.

Let H be a complete graph with 7 nodes, let $\mathcal{N}, C_0, \alpha, \beta$ be presented in Table 5.2, and let node 7 be the output node of HNEP Γ.

A sentential form (a configuration) of grammar G is a word $w \in (N \cup T)^*$. When simulating the derivations in G, each sentential form w of G corresponds to a string of Γ in node 1 and having one of the forms $wB_{n,0}$ or $w''A_n w' B_{i,0}$, where $A_n = \#$, $w, w', w'' \in (N \cup T)^*$ and $w = w' A_i w''$. The start symbol $S = A_1$ of G corresponds to an initial word $A_1\#$, represented as $A_1 B_{n,0}$ in node 1 of HNEP Γ, the other nodes do not contain any word. The simulation of the application of a rule of G to a substring of a sentential form of G is done in several evolution and communication steps in Γ, through rewriting the leftmost symbol and the two rightmost or the rightmost symbol of strings. This is the reason why we need the symbols to be rotated.

In the following we describe how the rotation of a symbol and the application of an arbitrary rule of grammar G are simulated in HNEP Γ.

Rotation.

Let $A_{i_1} A_{i_2} \dots A_{i_{k-1}} B_{i_k,0} = A_{i_1} w B_{i_k,0}$ be a word found at node 1, and let $w, w', w'' \in A^*$. Then, by applying rule `1.1` we obtain $A_{i_1} A_{i_2} \dots A_{i_{k-1}} B_{i_k,0} = A_{i_1} w B_{i_k,0} \xrightarrow{1.1} \{C'_{i_1} w B_{i_k,0},\ A_{i_1} w' C'_{i_t} w'' B_{i_k,0}\}$.

We note that during the simulation symbols C'_i should be transformed to λ', and this symbol can only be deleted from the left-hand end of the string (node 6). So, the replacement of C_{i_t} by its primed version in a string of the form $A_{i_1}w'C_{i_t}w''B_{i_k,0}$ results in a word that will stay in node 6 forever; thus, in the sequel, we will not consider strings with C'_i not in the leftmost position. In the communication step following the above evolution step, string $C'_{i_1}wB_{i_k,0}$ cannot leave node 1 and stays there for the next evolution step:

$$C'_{i_1}wB_{i_k,0} \xrightarrow{1.5} C'_{i_1}wB'_{i_k,0}.$$

Observe that rules `1.1` and `1.5` may be applied in any order. After then, string $C'_{i_1}wB'_{i_k,0}$ can leave node 1 and can enter only node 2. In the following steps of the computation, in nodes 1 and 2, the string is involved in evolution steps followed by communication:

$C_{i_1-t}wB_{i_k,t} \xrightarrow{1.4} C'_{i_1-(t+1)}wB_{i_k,t} \xrightarrow{1.6} C'_{i_1-(t+1)}wB'_{i_k,t+1}$ (in node 1),

$C'_{i_1-t}wB'_{i_k,t} \xrightarrow{2.1} C_{i_1-(t+1)}wB'_{i_k,t} \xrightarrow{2.2} C_{i_1-(t+1)}wB_{i_k,t+1}$ (in node 2).

We note that during this phase of the computation rules `1.2`: $A_i \to \lambda'$ or `2.7`: $A_n \to \lambda'$ may be applied in nodes 1 and 2. In this case, the string leaves node 1 or 2, but cannot enter any node. So, this case will not be considered in the sequel.

The process continues in nodes 1 and 2 until subscript i of C_i or that of C'_i is decreased to 1. In this case, either rule `1.7` : $C_1 \to \lambda'$ in node 1 or rule `2.3` : $C'_1 \to \lambda'$ in node 2 will be applied and the obtained string $\lambda'wB'_{i_k,i_1}$ or $\lambda'wB_{i_k,i_1}$ is communicated to node 3. (Notice that the string is able to leave the node either if both C and B are primed or both of them are unprimed.) Then, in node 3, depending on the form of the string, either evolution step $\lambda'wB'_{i_k,i_1} \xrightarrow{3.1} \lambda'wB'_{i_k,i_1}D_0$ or evolution step $\lambda'wB_{i_k,i_1} \xrightarrow{3.1} \lambda'wB_{i_k,i_1}D_0$ is performed. Strings $\lambda'wB'_{i_k,i_1}D_0$ or $\lambda'wB_{i_k,i_1}D_0$ can enter only node 4, where (depending on the form of the string) either evolution step $\lambda'wB_{i_k,i_1}D_0 \xrightarrow{4.1} \lambda'wE_{i_k,i_1}D_0$ or evolution step $\lambda'wB'_{i_k,i_1}D_0 \xrightarrow{4.2} \lambda'wE_{i_k,i_1}D_0$ follows. The obtained word, $\lambda'wE_{i_k,i_1}D_0$, can enter only node 6, where evolution step $\lambda'wE_{i_k,i_1}D_0 \xrightarrow{6.1} wE_{i_k,i_1}D_0$ is performed. Then the string leaves the node and enters node 2.

Then, in nodes 2 and 1, a sequence of computation steps is performed, when the string is involved in evolution steps followed by communication as follows:

$wE_{i_k,i_1-t}D_t \xrightarrow{2.4} wE'_{i_k,i_1-(t+1)}D_t \xrightarrow{2.5} wE'_{i_k,i_1-(t+1)}D'_{t+1}$ (in node 2).

$wE'_{i_k,i_1-t}D'_t \xrightarrow{1.8} wE_{i_k,i_1-(t+1)}D'_t \xrightarrow{1.9} wE_{i_k,i_1-(t+1)}D_{t+1}$ (in node 1),

The process continues in nodes 1 and 2 until the second subscript of $E'_{i,j}$ or that of $E_{i,j}$ is decreased to 1. In this case, either rule `1.10` : $E'_{i_k,1} \to F_{i_k}$ in node 1 or rule `2.6` : $E_{i_k,1} \to F_{i_k}$ in node 2 is applied and the new string, $wF_{i_k}D_{i_1}$ or $wF_{i_k}D'_{i_1}$, will be present in node 5. Notice that applying rules `1.1, 1.2` and `2.7` we obtain strings that cannot enter nodes 3 – 7 and stay in nodes 1 or 2.

The next evolution steps that take place in node 5 are as follows:

$wF_{i_k}D_{i_1}(wF_{i_k}D'_{i_1}) \xrightarrow{5.1(5.2)} wF_{i_k}B_{i_1,0} \xrightarrow{5.3} wA_{i_k}B_{i_1,0}$.

In the following communication step, string $wA_{i_k}B_{i_1,0}$ can enter either node 1 or node 2 (if $A_{i_1} \in T$). In the first case, the rotation of symbol A_{i_1} has been successful. Let us consider the second case. Then string $wA_{i_k}B_{i_1,0}$ appears in node 2.

- Suppose that the word $wA_{i_k}B_{i_1,0}$ does not contain any nonterminal symbol except A_n. Let $wA_{i_k}B_{i_1,0} = A_nw'A_{i_k}B_{i_1,0}$, where $w = A_nw'$. So, $w'A_{i_k}A_{i_1}$ is a result and it appears in node 7. Notice that if $w = w'A_nw''$ and $w' \neq \lambda$, then word $w'A_nw''A_{i_k}B_{i_1,0}$ leads to a string which will stay in node 6 forever (if rule `2.7` was applied). So, we consider the following evolution of the word $wA_{i_k}B_{i_1,0} = A_nw'A_{i_k}B_{i_1,0}$: $A_nw'A_{i_k}B_{i_1,0} \xrightarrow{2.7} \lambda'w'A_{i_k}B_{i_1,0} \xrightarrow{2.8} \lambda'w'A_{i_k}A_{i_1}$. Then, string $\lambda'w'A_{i_k}A_{i_1}$ will appear in node 6, where symbol λ' will be deleted by rule `6.1`. Finally, the resulted word $w'A_{i_k}A_{i_1}$ will enter node 7. This is a result.

- Suppose now that the word $wA_{i_k}B_{i_1,0}$ contains at least one nonterminal symbol different from A_n and $A_{i_1} \in T$.

  Consider the evolution of the word $wA_{i_k}B_{i_1,0} = w'A_nw''A_{i_k}B_{i_1,0}$ in node 2:

  $$w'A_nw''A_{i_k}B_{i_1,0} \xrightarrow{2.8} w'A_nw''A_{i_k}A_{i_1} \xrightarrow{2.7} w'\lambda'w''A_{i_k}A_{i_1}.$$

  Now, string $w'\lambda'w''A_{i_k}A_{i_1}$ will enter node 6 and either it will not be able to leave it (if $w' \neq \lambda$) or it will not be able to enter any of the other nodes (if $w' = \lambda$).

In the following we will explain how the application of the rules of G are simulated in Γ.
Rule $A_i \longrightarrow \lambda$**.** Suppose that string $A_iwB_{j,0}$ is in node 1 and let $w, w', w'' \in A^*$. Then, by evolution, we obtain $A_iwB_{j,0} \xrightarrow{1.2} \lambda'wB_{j,0}$ or $A_tw'A_iw''B_{j,0} \xrightarrow{1.2} A_tw'\lambda'w''B_{j,0}$ which can enter only node 6. String $A_tw'\lambda'w''B_{j,0}$ will stay in node 6 forever. By evolution $\lambda'wB_{j,0} \xrightarrow{6.1} wB_{j,0}$ and the resulting string, $wB_{j,0}$, enters in node 1 (and node 2, if $A_j \in T$). Thus, the application of rule $A_i \longrightarrow \lambda$ in G was correctly simulated.
Rule $A_i \longrightarrow A_j$**.** The evolution step performed at node 1 is $wB_{i,0} \xrightarrow{1.3} wB_{j,0}$. Since string $wB_{j,0}$ is in node 1, the simulation of the rule $A_i \longrightarrow A_j$ of grammar G was done in a correct manner.
Rule $A_j \longrightarrow A_sA_t$**.** At the end of the simulation of the rotation of a symbol, in node 5 instead of applying rule $D_j \to B_{j,0}$ ($D'_j \to B_{j,0}$) a rule $D_j \to G_{s,t}$ ($D'_j \to G_{s,t}$) is applied. That is, in node 5, either evolution step $wD_j \xrightarrow{5.4} wG_{s,t}$ or evolution step $wD'_j \xrightarrow{5.5} wG_{s,t}$ is performed. The new string $wG_{s,t}$ can enter only node 3, where, by evolution, $wG_{s,t} \xrightarrow{3.1} wG_{s,t}D_0$. String $wG_{s,t}D_0$ can enter only node 4, where evolution step $wG_{s,t}D_0 \xrightarrow{4.5} wE_{s,t}D_0$ follows. The process continues as above, in the case of simulating rotation, and in several computation steps the string wF_sD_t or $wF_sD'_t$ will enter node 5. After evolution in this node, the resulting string $wA_sB_{t,0}$ will enter node 1 (and node 2, if $A_t \in T$). Thus, the application of rule $A_j \longrightarrow A_sA_t$ of G is correctly simulated.
Rule $A_iA_j \longrightarrow A_sA_t$**.** The evolutionary processor in node 4 has rules `4.3` : $B_{i,j} \to E_{s,t}$ or `4.4` : $B'_{i,j} \to E_{s,t}$. As in the case of simulating rotation, above, we will obtain string $wA_sB_{t,0}$ in node 1 (and in node 2, if $A_t \in T$).

We have demonstrated how the rotation of a symbol and the application of rules of G are simulated by Γ. By the constructions, the reader can verify that G and Γ generate the same language. □

5.1.4 Obligatory HNEPs

As described in the second section, obligatory operations were considered in HNEPs. The result of the evolution step now consists of all strings produced from the current ones by the operations of insertion, deletion and substitution (the current strings are no longer preserved, even if some operation is not applicable to them). Not only this yields a simpler and a more uniform definition, but also the following result is obtained.

Theorem 5.4 *Any CPM0 P can be simulated by an OHNEP P', where obligatory evolutionary processors are with empty input and output filters and only insertion and obligatory deletion operations in right and left modes are used (without obligatory substitution operations). [29], [28]*

Proof. Let us consider a CPM0 P with symbols $a_j \in \Sigma,\ j \in J = \{0, 1 \ldots, n\}$, $a_0 = 0$ is the blank symbol, and states, $q_i \in Q$, $i \in I = \{1, 2, \ldots, f\}$, where q_1 is the initial state and the only terminal state is $q_f \in Q$. We suppose that P stops in the terminal state q_f on every symbol, i.e., there are instructions $q_f a_j \to Halt$, $a_j \in J$. (Notice that it is easy to transform any CPM0 P into a CPM0 P' that stops on every symbol in the final state.)

So, we consider CPM0 P with the set R of instructions of the forms $q_i a_j \longrightarrow q_l$, $q_i a_j \longrightarrow a_k q_l$, $q_i 0 \longrightarrow a_k q_m 0$, $q_f a_j \longrightarrow Halt$, where $q_i \in Q \setminus \{q_f\},\ q_l, q_m \in Q,\ a_j, a_k \in \Sigma$. We do not consider case $q_m = q_f$ in instruction $q_i 0 \longrightarrow a_k q_m 0$. Notice that it is easy to modify the program of P such that it only halts by instructions of other types.

A configuration $w = q_i a_j W$ of CPM0 P describes that P in state $q_i \in Q$ considers symbol $a_j \in \Sigma$ to the left of $W \in \Sigma^*$.

Now we construct an OHNEP P' simulating P. To simplify its description, we use $\langle q_f a_j \rangle$ and $\langle q_f a_j \rangle_1$, $j \in J$ as aliases of $\langle out \rangle$. Let $v \in Q$ and let $u \in Q \cup Q \cdot \{0\}$.

$$
\begin{aligned}
P' &= (V, G, N, C_0, \alpha, \beta, i_0),\ V = \{q_1\} \cup \Sigma,\ G = (X_G, E_G),\\
X_G &= \{\langle init \rangle, \langle out \rangle\} \cup \{\langle q_i a_j \rangle \mid (q_i a_j \to v) \in R\} \cup \{\langle q_i a_j \rangle_1 \mid (q_i a_j \to a_k u) \in R\},\\
E_G &= \{(\langle init \rangle, \langle q_1 a_j \rangle) \mid j \in J\} \cup \{(\langle q_i a_j \rangle, \langle q_l a_k \rangle) \mid (q_i a_j \to q_l) \in R,\ k \in J\}\\
&\cup\ \{(\langle q_i a_j \rangle, \langle q_i a_j \rangle_1) \mid (q_i a_j \to a_k u) \in R\}\\
&\cup\ \{(\langle q_i a_j \rangle_1, \langle q_l a_s \rangle) \mid (q_i a_j \to a_k q_l) \in R,\ s \in J\}\\
&\cup\ \{(\langle q_i 0 \rangle_1, \langle q_l 0 \rangle_1) \mid (q_i 0 \to a_k q_l 0), (q_l 0 \to a_s u) \in R, s \in J\},\\
&\cup\ \{(\langle q_i 0 \rangle_1, \langle q_p a_s \rangle) \mid (q_i 0 \to a_k q_l 0), (q_l 0 \to q_p) \in R, s \in J\},\\
C_0(x) &= \{q_1 W\},\ \text{if } x = \langle init \rangle,\ \text{where } W \text{ is the input of } P,\\
C_0(x) &= \emptyset,\ x \in X_G \setminus \{\langle init \rangle\},\ \beta(x) = 2,\ x \in X_G,\\
N(x) &= (M_x, \emptyset, \emptyset, \emptyset, \emptyset),\ x \in X_G,\ M_{\langle init \rangle} = \{q_1 \to \lambda\},\\
M_x &= \{a_j \to \lambda\}, x = \langle q_i a_j \rangle,\\
M_x &= \{\lambda \to a_k\}, x = \langle q_i a_j \rangle_1,\ \text{where } (q_i a_j \to a_k u) \in R,\\
\alpha(x) &= l,\ \text{if } M_x = \{a \to \lambda\},\ \alpha(x) = r,\ \text{if } M_x = \{\lambda \to a\} \text{ or } M_x = \emptyset.
\end{aligned}
$$

OHNEP P' will simulate every computation step performed by CPM0 P by a sequence of computation steps in P'.

Let $q_1a_jW_0$ be the initial configuration of CPM0 P. We represent this configuration in node $\langle init\rangle$ of OHNEP P' as a word $q_1a_jW_0$. Obligatory evolutionary processor associated with this node is $N(\langle init\rangle) = (\{(q_1 \to \lambda)^l\}, \emptyset, \emptyset, \emptyset, \emptyset)$. Since all other nodes also have empty filters, in the following we will skip the complete description of obligatory evolutionary processors, and will present only their obligatory evolutionary operations. The word a_jW_0 will be passed from node $\langle init\rangle$ to nodes $\langle q_1a_j\rangle,\ j \in J$.

If the computation in P is finite, then the final configuration q_fW of P will appear at node $\langle out\rangle$ of P' as a string W, moreover, any string W that can appear at node $\langle out\rangle$ corresponds to a final configuration q_fW of P. In the case of an infinite computation in P, no string will appear in node $\langle out\rangle$ of P' and the computation in P' will never stop.

Now we describe nodes of OHNEP P', connections between them and obligatory evolutionary operations, associated with these nodes. Let $I' = I \setminus \{f\}$.

1. Node $\langle q_ia_j\rangle$ with operation $(a_j \to \lambda)^l,\ i \in I',\ j \in J$.
Let word $a_tW,\ t \in J,\ W \in \Sigma^*$ appear in this node. If $j \neq t$, then this word a_tW will be discarded, and in the next communication step node $\langle q_ia_j\rangle$ will send nothing. If $j = t$, then the node sends W to nodes $\{\langle q_la_k\rangle \mid k \in J\}$ or $\langle q_ia_j\rangle_1$.

- Instruction of P is $q_ia_j \longrightarrow q_l,\ i \in I',\ j \in J,\ l \in I$. Node $\langle q_ia_j\rangle$ is connected with nodes $\{\langle q_la_k\rangle \mid k \in J\}$.

- Instructions of P are $q_ia_j \longrightarrow a_kq_l$ or $q_i0 \longrightarrow a_kq_l0,\ i \in I',\ j,k \in J,\ l \in I$. Node $\langle q_ia_j\rangle$ is connected with node $\langle q_ia_j\rangle_1$.

2. Node $\langle q_ia_j\rangle_1,\ i \in I',\ j \in J$ with operation $(\lambda \to a_k)^r$ receives word W and sends word Wa_k to nodes $\{\langle q_la_s\rangle \mid s \in J\}$ or $\langle q_l0\rangle_1$.

- Instructions of P are $q_ia_j \longrightarrow a_kq_l,\ i \in I',\ j,k \in J,\ l \in I$. Node $\langle q_ia_j\rangle_1$ is connected with nodes $\{\langle q_la_s\rangle \mid s \in J\}$.

- Instruction of P is $q_i0 \longrightarrow a_kq_l0,\ i \in I',\ k \in J,\ l \in I$. Node $\langle q_i0\rangle_1$ is connected with nodes $\{\langle q_pa_s\rangle \mid s \in J\}$ if there exists an instruction of P $q_l0 \longrightarrow q_p$, $p \in I$; and with node $\langle q_l0\rangle_1$ in other cases.

We repeat that in all cases, we mean $\langle out\rangle$ whenever we write $\langle q_fa_j\rangle$ or $\langle q_fa_j\rangle_1$, $j \in J$.

Now we describe simulation of instructions of CPM0 P by OHNEP P'.

Instruction $q_ia_j \longrightarrow q_l$**:** $q_ia_jW \xrightarrow{P} q_lW$**.**

Let word a_tW, where $t \in J,\ W \in \Sigma^*,\ i \in I'$ appear in node $\langle q_ia_j\rangle$. If $t \neq j$, string a_tW will be discarded; if $t = j$, string W will be passed to nodes $\{\langle q_la_s\rangle \mid s \in J\}$. If $l = f$, the final configuration q_fW of P will appear in the output node $\langle out\rangle$ as W. This is the result. So, we simulated instruction $q_ia_j \longrightarrow q_l$ in a correct manner.

Instruction $q_ia_j \longrightarrow a_kq_l$**:** $q_ia_jW \xrightarrow{P} q_lWa_k$**.**

Let word a_tW, where $t \in J,\ W \in \Sigma^*,\ i \in I'$ appear in node $\langle q_ia_j\rangle$. If $t \neq j$, string a_tW will be discarded; if $t = j$ string W will be passed to node $\langle q_ia_j\rangle_1$. Node $\langle q_ia_j\rangle_1$ receives this word and sends word Wa_k to nodes $\langle q_la_s\rangle,\ s \in J$. If $l = f$, the final configuration q_fWa_k of P will appear in the output node $\langle out\rangle$ as Wa_k. This is the result. So, we simulated instruction $q_ia_j \longrightarrow a_kq_l$ in a correct manner.

Instruction $q_i 0 \longrightarrow a_k q_l 0$**:** $q_i 0W \xrightarrow{P} q_l 0Wa_k$**.**

Let word $a_t W$, where $t \in J,\ W \in \Sigma^*,\ i \in I'$ appear in node $\langle q_i 0 \rangle$. If $a_t \neq 0$, string $a_t W$ will be discarded; if $a_t = 0$, string W will be passed to node $\langle q_i 0 \rangle_1$. It receives this word and sends word Wa_k to nodes $\langle q_p a_s \rangle, s \in J$ if there is instruction of P $q_l 0 \longrightarrow q_p, p \in I'$. If there are instructions $q_l 0 \longrightarrow a_s q_p$ or $q_l 0 \longrightarrow a_s q_p 0$, then node $\langle q_i 0 \rangle_1$ is connected with node $\langle q_l 0 \rangle_1$. Thus, word Wa_k will be passed to node $\langle q_l 0 \rangle_1$, which corresponds to the configuration of P which has "just read" symbol 0 in state q_l. So, we simulated instruction $q_i 0 \longrightarrow a_k q_l 0$ in a correct manner.

So, CPM0 P is correctly modeled. We have demonstrated that the rules of P are simulated in P'. The proof that P' simulates only P comes from the construction of the rules in P', we leave the details to the reader. □

Conclusion We have described the networks of evolutionary processors, their models and variants, together with the associated results. A few selected results were presented in more details. For instance, NEPs with two nodes are already computationally complete modulo the terminal alphabet. HNEPs with one node are given the precise regular characterization, HNEPs with two nodes are not computationally complete, while seven nodes are enough to reach the computational completeness of HNEPs, even with a complete graph. We should mention that a network over a complete graph (with loops, although it is not important for the last proof) may be viewed as number of agents in a common environment, acting "independently" without explicitly enforcing any transition protocol, where a computationally complete behavior still emerges.

A particularly interesting variant is obligatory HNEPs (OHNEPs). Using a power of the underlying graph, computational completeness is obtained even without the filters. In case of a complete graph, OHNEPs are still computationally complete. Moreover, it suffices that the sum of numbers of symbols in filters of each node does not exceed one. The last proof has been obtained in [25], using a variant of circular Post machines, CPM5, introduced in [84]. Some details of definitions and associated results for CPM5 can be found in Appendix A3.

5.2 Insertion-Deletion P Systems

In this section we consider insertion-deletion P systems with priority of deletion over the insertion. We show that such systems with one symbol context-free insertion and deletion rules are able to generate Parikh sets of all recursively enumerable languages ($PsRE$). If one-symbol one-sided context is added to insertion or deletion rules, then all recursively enumerable languages can be generated. The same result holds if a deletion of two symbols is permitted. We also show that the priority relation is very important and in its absence the corresponding class of P systems is strictly included in the family of matrix languages (MAT).

The insertion and the deletion operations originate from the language theory, where they where introduced mainly with linguistic motivation. In general form, an insertion operation means adding a substring to a given string in a specified (left and right) context, while

a deletion operation means removing a substring of a given string from a specified (left and right) context. A finite set of insertion-deletion rules, together with a set of axioms provide a language generating device: starting from the set of initial strings and iterating insertion-deletion operations as defined by the given rules we get a language.

Insertion systems, without using the deletion operation, were first considered in [190], however the idea of the context adjoining was exploited long before by [223]. Both insertion and deletion operations were first considered together in [206] and related formal language investigations can be found in several places; we mention only [204], [231] and [251]. In the last few years, the study of these operations has received a new motivation from molecular computing, see, for example, [161], [205], [256], [275], because, from the biological point of view, insertion-deletion operations correspond to mismatched annealing of DNA sequences.

As expected, insertion-deletion systems are quite powerful, leading to characterizations of recursively enumerable languages. This is not quite surprising as the corresponding device contains two important ingredients needed for the universality: the context dependency and the erasing ability. However, as it was shown in [225], the context dependency may be replaced by insertion and deletion of strings of sufficient length, in a context-free manner. If the length is not sufficient (less than two) then such systems are decidable and a characterization of them was shown in [278]. Similar investigations were continued in [232] and [210] on insertion-deletion systems with one-sided contexts, i.e. where the context dependency is present only from the left (right) side of all insertion and deletion rules. These articles also give some combinations of rule parameters that lead to systems which are not computationally complete. However, if these systems are combined with the distributed computing framework of P systems [252], then their computational power may strictly increase, see [211], [209].

In this section we study P systems with context-free insertion and deletion rules of one symbol. We show that this family is strictly included in MAT, however some non-context-free languages may be generated. If Parikh vectors are considered, then the corresponding family equals to the family of Parikh sets of matrix languages ($PsMAT$). When a priority of deletion over insertion is introduced, $PsRE$ can be characterized, but in terms of language generation such systems cannot generate a lot of languages because there is no control on the position of an inserted symbol. If one-sided contextual insertion or deletion rules are used, then this can be controlled and all recursively enumerable languages can be generated. The same result holds if a context-free deletion of two symbols is allowed.

5.2.1 Minimal insertion-deletion P systems

When a membrane structure is added to minimal insertion-deletion systems without context, their computational power is increased.

Theorem 5.5 $PsStP_*(ins_1^{0,0}, del_1^{0,0}) = PsMAT$.

Proof. It is not difficult to see that dropping the requirement of the uniqueness of the instructions with the same label, the power of partially blind register machines does not change, see, e.g., [170]. We use this fact for the proof.

The inclusion $PsStP_*(ins_1^{0,0}, del_1^{0,0}) \subseteq PsMAT$ follows from the simulation of minimal context-free insertion-deletion P systems by partially blind register machines, which are

known to characterize $PsMAT$ [170]. Indeed, any rule $(\lambda, a, \lambda; q)_a \in R_p$ is simulated by instructions $(p : [RaP], q)$. Similarly, rule $(\lambda, a, \lambda; q)_e \in R_p$ is simulated by instructions $(p : [RaM], q)$.

The output region i_0 is associated to the final state, while the halting is represented by the absence of the corresponding symbols (final zero-test) as follows. We assume that R_{i_0} has no insertion rules ($\emptyset$ can be generated by a trivial partially blind register machine), and the output registers correspond to those symbols that cannot be deleted by rules from R_{i_0}.

The converse inclusion follows from the simulation of partially blind register machines by P systems. Indeed, with every instruction p of the register machine we associate a cell. Instruction $(p : [RA_kP], q)$ is simulated by rule $(\lambda, A_k, \lambda; q)_a \in R_p$, and instruction $(p : [RA_kM], q)$ by $(\lambda, A_k, \lambda; q)_e \in R_p$. Final zero-tests: rules $(\lambda, A_k, \lambda; \#)_e \in R_h$, $k \geq m$, should be inapplicable ($R_\# = \emptyset$). □

As the membrane structure is a tree, one-way inclusion follows.

Corollary 5.1 $PsSP_*(ins_1^{0,0}, del_1^{0,0}) \subseteq PsMAT$.

In terms of the generated language the above systems are not too powerful, even with priorities. Like in the case of insertion-deletion systems there is no control on the position of insertion. Hence, the language $L = \{a^*b^*\}$ cannot be generated, for insertion strings of any size. Hence we obtain:

Theorem 5.6 $REG \backslash LStP_*(ins_n^{0,0} < del_1^{0,0}) \neq \emptyset$, *for any* $n > 0$.

However, there are non-context-free languages that can be generated by such P systems (even without priorities and deletion).

Theorem 5.7 $LStP_*(ins_1^{0,0}, del_0^{0,0}) \setminus CF \neq \emptyset$.

Proof. It is easy to see that the language $\{w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c\}$ is generated by such a system with 3 nodes, inserting consecutively a, b and c. □

For the tree case the language $\{w \in \{a, b\}^* : |w|_a = |w|_b\}$ can be generated in a similar manner.

We show a more general inclusion:

Theorem 5.8 $ELStP_*(ins_n^{0,0}, del_1^{0,0}) \subset MAT$, *for any* $n > 0$.

Proof. As in [210] we can suppose that there are no deletions of terminal symbols. We also suppose that there is only one initial string in the system, because there is no interaction between different evolving strings and the result matches the union of results for the systems with only one string. Consider a tissue P system Π with alphabet O, terminal symbols T, the set H of unique cell labels and the initial string w in cell labeled p_0. Such a system can be simulated by the following matrix grammar $G = (O \cup H, T, S, P)$.

For insertion instruction $(\lambda, a_1 \cdots a_n, \lambda; q)_a$ in cell p, the matrix $\{p \to q, D \to Da_1D \cdots Da_nD\} \in P$. For any deletion instruction $(\lambda, A, \lambda; q)_e$ in cell p, the matrix $\{p \to q, A \to \lambda\} \in P$. Three additional matrices $\{h \to \lambda\}$, $\{D \to \lambda\}$ and $\{S \to q_0Da_1D \cdots Da_mD\}$ $(w = a_1 \cdots a_m)$ shall be also added to P.

The above construction correctly simulates the system Π. Indeed, symbols D represent placeholders for all possible insertions. The first rule in the matrix simulates the navigation between cells. □

Nevertheless, minimal context-free insertion-deletion systems with priorities do generate $PsRE$. This is especially clear for the tissue P systems: jumping to an instruction corresponds to sending a string to the associated region, and the entire construction is a composition of graphs shown in Figure 5.1. The decrement instruction works correctly because of priority of deletion over insertion.

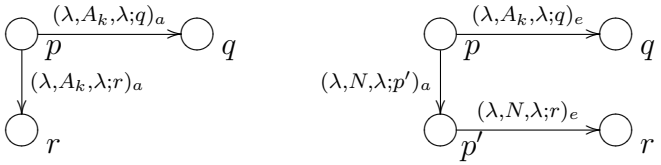


Figure 5.1: Simulating $(p : [RkP], q, r)$(left) and $(p : \langle RkZM\rangle, q, r)$ (right)

We now give a more sophisticated proof for the tree-like membrane structure.

Theorem 5.9 $PsSP_*(ins_1^{0,0} < del_1^{0,0}) = PsRE$.

Proof. The proof is done by showing that for any register machine $\mathcal{M} = (d, Q, I, q_0, q_f)$ there is a P system $\Pi \in PsSP_*(ins_1^{0,0} < del_1^{0,0})$ with $Ps(\Pi) = Ps(\mathcal{M})$. Then the existence of register machines generating $PsRE$ implies $PsRE \subseteq PsSP_*(ins_1^{0,0} < del_1^{0,0})$.

Let Q_+ (Q_-) be the sets of labels of increment (conditional decrement, respectively) instructions of a register machine, and let $Q = Q_+ \cup Q_- \cup \{q_f\}$ represent all instructions. Consider a P system with alphabet $Q \cup \{A_i \mid 1 \le i \le d\} \cup \{Y\}$ and the following structure (illustrated in Figure 5.2, the structures in the dashed rectangles are repeated for every instruction of the register machine):

$$\begin{aligned}
\mu &= [\,[\,[\prod_{p\in Q_+} \mu_{\langle p+\rangle} \prod_{p\in Q_-} \mu_{\langle p-\rangle}\,]_3[\ ]_4\,]_2\,]_1, \text{ where}\\
\mu_{\langle p+\rangle} &= [\,[\,[\ ]_{p_3^+}\,]_{p_2^+}\,]_{p_1^+},\ p- \text{ increment},\\
\mu_{\langle p-\rangle} &= [\,[\,[\ ]_{p_3^-}\,]_{p_2^-}\,[\,[\ ]_{p_3^0}\,]_{p_2^0}\,]_{p_1^-},\ p- \text{ conditional decrement}.
\end{aligned}$$

Initially there is a single string q_0 in membrane 3. The rules are the following.

$$\begin{aligned}
R_1 = \{ &\quad 1 : (\lambda, Y, \lambda; out)_e\},\\
R_2 = \{ &\quad 2.1 : (\lambda, Y, \lambda; out)_a, \quad 2.2 : (\lambda, Y, \lambda; in_4)_e\},\\
R_3 = \{ &\quad 3.1 : (\lambda, p, \lambda; in_{p_1^+})_e \mid p \in Q_+\} \cup \{3.2 : (\lambda, p, \lambda; in_{p_1^-})_e \mid p \in Q_-\}\\
\cup \{ &\quad 3.3 : (\lambda, Y, \lambda; here)_e, \quad 3.4 : (\lambda, h, \lambda; out)_e\},
\end{aligned}$$

For any rule $(p : [RkP], q, s)$, $R_{p_3^+} = \emptyset$ and

$$\begin{aligned}
R_{p_1^+} = \{ &\quad a.1.1 : (\lambda, A_k, \lambda; in_{p_2^+})_a, \quad a.1.2 : (\lambda, Y, \lambda; out)_a\},\\
R_{p_2^+} = \{ &\quad a.2.1 : (\lambda, q, \lambda; out)_a, \quad a.2.1' : (\lambda, s, \lambda; out)_a,\\
&\quad a.2.2 : (\lambda, q, \lambda; in_{p_3^+})_e, \quad a.2.2' : (\lambda, s, \lambda; in_{p_3^+})_e\},
\end{aligned}$$

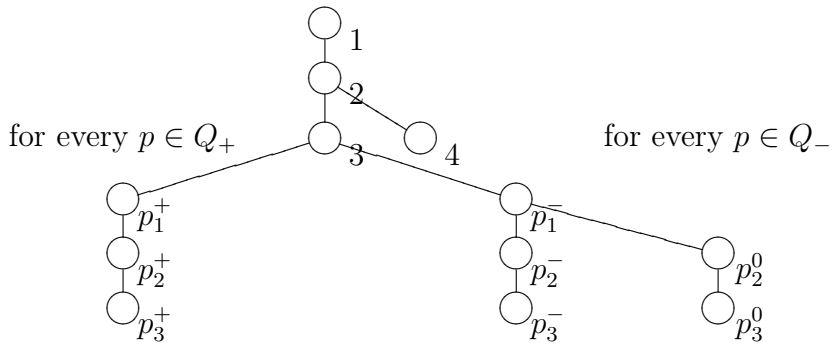


Figure 5.2: Membrane structure for Theorem 5.9

For any rule $(p : \langle RkZM \rangle, q, s)$, $R_{p_3^-} = R_{p_3^0} = \emptyset$ and

$$\begin{aligned}
R_{p_1^-} = \{ \quad & e.1.1 : (\lambda, A_k, \lambda; in_{p_2^-})_e, \quad e.1.2 : (\lambda, Y, \lambda; in_{p_2^0})_a, \quad e.1.3 : (\lambda, Y, \lambda; out)_e \}, \\
R_{p_2^-} = \{ \quad & e.2.1 : (\lambda, q, \lambda; out)_a, \quad e.2.2 : (\lambda, q, \lambda; in_{p_3^-})_e, \\
& e.2.3 : (\lambda, s, \lambda; in_{p_3^-})_e, \quad e.2.4 : (\lambda, Y, \lambda; here)_a \}, \\
R_{p_2^0} = \{ \quad & e.3.1 : (\lambda, s, \lambda; out)_a, \quad e.3.2 : (\lambda, q, \lambda; in_{p_3^0})_e, \quad e.3.3 : (\lambda, s, \lambda; in_{p_3^0})_e \}.
\end{aligned}$$

In membrane 3 configurations $(p, x_1, \cdots, x_n)$ of $\mathcal{M}$ are encoded by strings

$$Perm(pA_1^{x_1} \cdots A_n^{x_n} Y^t),\ t \geq 0.$$

We say that such strings have a *simulating* form. Clearly, in the initial configuration the string is already in the simulating form.

To prove that system Π correctly simulates $\mathcal{M}$ we prove the following claims:

1. For any transition $(p, x_1 \cdots x_n) \Longrightarrow (q, x_1', \cdots, x_n')$ in $\mathcal{M}$ there exist a computation in Π from the configuration containing $Perm(pA_1^{x_1} \cdots A_n^{x_n} Y^t)$ in membrane 3 to the configuration containing $Perm(qA_1^{x_1'} \cdots A_n^{x_n'} Y^{t'})$, $t' \geq 0$ in membrane 3 such that during this computation membrane 3 is empty on all intermediate steps and, moreover, this computation is unique.

2. For any successful computation in Π (yielding a non-empty result), membrane 3 contains only strings of the above form.

3. The result $(x_1, \cdots, x_n)$ in Π is obtained if and only if a string of form

   $$Perm(hA_1^{x_1} \cdots A_n^{x_n})$$

   appears in membrane 3.

Now we prove each claim from above. Consider a string $Perm(pA_1^{x_1} \cdots A_n^{x_n} Y^t)$, $t \geq 0$ in membrane 3 of Π. Take an instruction $p : (ADD(k), q, s) \in P$. The only applicable rule in Π is from the group 3.1 (in the future we simply say rule 3.1) yielding the string $Perm(A_1^{x_1} \cdots A_n^{x_n} Y^t)$ in membrane p_1^+. After that rule $a.1.1$ is applied yielding string $Perm(A_1^{x_1} \cdots A_k^{x_k+1} \cdots A_n^{x_n} Y^t)$ in membrane p_2^+. After that one of rules $a.2.1$ or $a.2.1'$ is applied; then rule $a.1.2$ yields one of strings $Perm(zA_1^{x_1} \cdots A_k^{x_k+1} \cdots A_n^{x_n} Y^{t+1})$, $z \in \{q, s\}$, is in the simulating form.

Now suppose that there is an instruction $(p : \langle RkZM \rangle, q, s) \in P$. Then the only applicable rule in Π is 3.2 which yields the string $Perm(A_1^{x_1} \cdots A_n^{x_n} Y^t)$ in membrane p_1^-. Now if $x_k > 0$, then, due to the priority, rule $e.1.1$ will be applied followed by application of rules $e.2.4$, $e.2.1$ and $e.1.3$ which yields the string $Perm(qA_1^{x_1} \cdots A_k^{x_k - 1} \cdots A_n^{x_n} Y^{t'})$ that is in the simulating form. If $x_k = 0$, then rule $e.1.2$ will be applied (provided that all symbols Y were previously deleted by rule 3.3), followed by rules $e.3.1$ and $e.1.3$ which leads to the string $Perm(sA_1^{x_1} \cdots A_n^{x_n})$ that is in the simulating form.

To show that membrane 3 is empty during the intermediate steps, we prove the following invariant:

Invariant 5.1 *During a successful computation, any visited membrane p_1^+ or p_1^- is visited an even number of times as follows: first a string coming from membrane 3 is sent to an inner membrane (p_2^+, p_2^- or p_2^0) and after that a string coming from an inner membrane is sent to membrane 3.*

Indeed, since there is only one string in the initial configuration, it is enough to follow only its evolution. Hence, a sting may visit the node p_1^+ or p_1^- only if on the previous step symbol p was deleted by one of rules 3.1 or 3.2. If one of rules $a.1.2$ or $e.1.3$ is applied, then membrane 3 will contain a string of form $Perm(A_1^{x_1} \cdots A_n^{x_n} Y^t)$ which cannot evolve anymore because all rules in membrane 3 imply the presence of symbol from the set Q. Hence, the string is sent to an inner membrane. In the next step the string will return from the inner membrane by one of rules $a.2.1$, $a.2.1'$, $e.2.1$ or $e.3.1$ inserting a symbol from Q. If the string enters an inner membrane again, then it will be sent to a trap membrane (p_3^+, p_3^- or p_3^0) by rules deleting symbols from Q. Hence the only possibility is to go to membrane 3 (a string that visited membrane p_2^- will additionally use rule $e.2.4$).

For the second claim, it suffices to observe that the invariant above ensures that in membrane 3 only one symbol from Q can be present in the string.

The third claim holds since a string may move to membrane 2 if and only if the final label h of $\mathcal{M}$ appears in membrane 3. Then, the string is checked for the absence of symbols Y by rule 2.2 (note that symbols Y can be erased in membrane 3 by rule 3.3) and sent to the environment by rules 2.1 and 1. Hence, the string sent to the environment will contain only occurrences of symbols A_i, $1 \leq i \leq d$.

By induction on the number of computational steps we obtain that Π simulates any computation in $\mathcal{M}$. Claim 1 and 2 imply it is not possible to generate other strings and Claim 3 implies that the same result is obtained. Hence, $Ps(\mathcal{M}) \subseteq Ps(\Pi)$.

The converse inclusion $Ps(\Pi) \subseteq Ps(\mathcal{M})$ follows from above considerations as well. Since Π can compute only configurations corresponding to configurations in $\mathcal{M}$, a computation in $\mathcal{M}$ can be reconstructed from a successful computation in Π. This concludes the proof. □

We remark that an empty string may be obtained during the proof. This string can still evolve using insertion rules. If we would like to forbid such evolutions, it suffices to use a new symbol, e.g. X, in the initial configuration, add new surrounding membrane and a rule that deletes X from it.

5.2.2 Small contextual insertion-deletion P systems

Although Theorem 5.9 shows that the systems from the previous section are quite powerful, they cannot generate RE without control on the place where a symbol is inserted. Once we allow a context in insertion or deletion rules, they can.

Theorem 5.10 $LSP_*(ins_1^{0,1} < del_1^{0,0}) = RE.$

The result is proved by simulating register machines with an output tape. The proof can be found in [88]. Taking M in the left context yields the mirror language. Since RE is closed with respect to the mirror operation we get the following corollary:

Corollary 5.2 $LSP_*(ins_1^{1,0} < del_1^{0,0}) = RE.$

A similar result holds if contextual deleting operation is allowed.

Theorem 5.11 $LSP_*(ins_1^{0,0} < del_1^{1,0}) = RE.$

The result is proved by simulating register machines with an output tape. The proof can be found in [88].

Since RE is closed with respect to the mirror operation we obtain:

Corollary 5.3 $LSP_*(ins_1^{0,0} < del_1^{0,1}) = RE.$

We remark that the contextual deletion was used only to check for erroneous evolutions. Therefore we can replace it by a context-free deletion of two symbols.

Theorem 5.12 $LSP_*(ins_1^{0,0} < del_2^{0,0}) = RE.$

The proof can be found in [88]. We mention that the counterpart of Theorem 5.12 obtained by interchanging parameters of the insertion and deletion rules is not true, see Theorem 5.6.

Conclusions We showed several results concerning P systems with insertion and deletion rules of small size. Surprisingly, systems with context-free rules inserting and deleting only one symbol are quite powerful and generate $PsRE$ if the priority of deletion over insertion is used. From the language generation viewpoint such systems are not too powerful and no language specifying the order of symbols can be generated. To be able to generate more complicated languages we considered systems with one-symbol one-sided insertion or deletion contexts. In both cases we obtained that any recursively enumerable language can be generated. The same result holds if a context-free deletion of two symbols is allowed. The counterpart of the last result is not true, moreover Theorem 5.6 shows that the insertion of strings of an arbitrary size still cannot lead to generating languages like a^*b^*.

We also have considered one-symbol context-free insertion-deletion P systems without the priority relations and we showed that in terms of Parikh sets these systems characterize $PsMAT$ family. However, in terms of the generated language such systems are strictly included in MAT.

We remark that context-free insertions and deletions of one symbol correspond to random point mutations from an evolutionary point of view. We think that the obtained results could give a frame for the modeling of evolutionary processes.

Most of the results above were obtained using rules with target indicators. It is interesting to investigate the computational power of systems with non-specific target indicators *in* or *go*. Another open problem is to replace the priority relation by some other mechanism from the P systems area without decreasing the computational power.

5.3 (Exo) Insertion-Deletion Operations

The main aim of this section is to consider the operations applied at the ends of the string, and prove the computational completeness in case of priority of deletion over insertion. This result presents interest since the strings are controlled by a tree structure only, and because insertion and deletion of one symbol are the simplest string operations.

To obtain a simple proof, we use a new variant (CPM5) of circular Post machines (Turing machines moving one-way on a circular tape): those with instructions changing a state and either reading one symbol or writing one symbol. We believe CPM5 deserves attention as a simple, yet useful tool.

In the last part of the section, we return to the case without priorities. We give a lower bound on the power of such systems, which holds even for one-sided operations only.

Insertion and deletion are fundamental string operations, introduced in the formal language theory mainly with linguistic motivation. The biological motivation is that these operations correspond to mismatched annealing of DNA sequences. They are also present in the evolution processes as point mutations as well as in RNA editing, see [141], [271] and [256].

In general, insertion/deletion means adding/removing a substring to/from a given string in a specified (left and right) context. A language-generating device can be defined by a finite set of insertion-deletion rules together with a set of axioms: the language is obtained by iterating the operations, as the set of terminal strings from the closure of the axioms under the operations.

We mention some sources concerning the topic of this section; see [190] for insertion systems, [223] for the idea of context adjoining, [206] for insertion and deletion. Related formal language theoretic research can be found in e.g., [204], [231] and [251], and the biologically motivated studies can be found in e.g., [161], [205], [256] and [275].

As expected, insertion-deletion systems characterize recursively enumerable languages, which is not surprising since the systems the context dependency and the erasing ability. However, the contexts may be replaced by insertion and deletion of strings of sufficient length, in a context-free manner, [225]. If the length is not sufficient (at most two symbols), then such systems are decidable and their characterization was shown in [278].

Similar research continued in [232] and [210] on insertion-deletion systems with one-sided contexts, i.e., where the context dependency is present only from the left (right) side of all insertion and deletion rules. The behavior of such variants is between those of context-free and context-dependent insertion-deletion systems. Indeed, like in the context-free systems, an insertion or deletion may be performed any number of times, but like in the contextual variant, their application site is not arbitrary. The above papers give several computational completeness results depending on the size of parameters of insertion or deletion rules, i.e.,

the bounds on the lengths of strings in the description of rules, *(inserted string, its left context, its right context; deleted string, its left context, its right context)*; some combinations do not lead to computational completeness and there are languages that cannot be generated by such devices.

In [208] one-sided insertion-deletion systems with insertion and deletion rules of at most two symbols were considered. This corresponds to systems of size $(1, 1, 0; 1, 1, 0)$, $(1, 1, 0; 1, 0, 1)$, $(1, 1, 0; 2, 0, 0)$ and $(2, 0, 0; 1, 1, 0)$, where the first three numbers represent the maximal length of the inserted string and the maximal length of the left and right contexts, while the last three numbers represent the same information for deletion rules. A characterization in terms of context-free grammars of the class of insertion-deletion systems of size $(1, 1, 0; 1, 1, 0)$ was presented. It was also shown that such systems generate some non-regular context-free languages even without deletion. The remaining classes are not computationally complete; the language $(ba)^+$ cannot be generated by such systems.

In [208] one also considered insertion-deletion operations in P systems framework, which is a maximally parallel distributed computing device inspired by the structure and the functioning of a living cell. We refer to [252], [257], [186] and [284] for more details about P systems. It was shown that such additional control permits to increase the computational power up to computationally completeness results for all four cases, improving the results from [211] and [209]. However, the framework of P systems cannot increase the computational power to such extent in all cases, namely it was shown that if context-free insertion and deletion rules using at most two symbols are considered, i.e. systems of size $(2, 0, 0; 2, 0, 0)$, then the corresponding P systems are still not computationally complete [208], [278]. It is thus interesting to consider conditions that would allow such systems to reach computational completeness.

In [87] one considers insertion-deletion P systems with insertion and deletion operations applied at the ends of the string (called exo-operations). Such systems with insertion of one symbol and deletion of up to two symbols and systems with insertion of up to two symbols and deletion of one symbol are computationally complete. The question about the computational power of insertion-deletion P systems with one-symbol insertion and one-symbol deletion at the ends of string has been left open (except the computational completeness holds in tissue case).

We should also mention the research in [203]: one considers insertion at one end of a string coupled with deletion at the other end. Even when the pairing is not prescribed, the universality is still achieved. However, the size of the inserted and deleted substrings is not bounded.

In this section we deal with one-symbol exo-insertion and one-symbol exo-deletion with priority of the latter. We also introduce a new variant (CPM5) of circular Post machines and use it for constructing a simpler proof; we believe CPM5 presents interest in itself, as a convenient tool for similar proofs. Finally, we give a lower bound on the power of systems without priorities, by a construction with one-sided operations.

It has been shown in [214] that any Turing machine can be simulated by a CPM0. To obtain, e.g., a language generating device, it suffices to introduce non-determinism and perform minor post-processing (such as removing the blanks at the ends of the halting configuration; doing this by a CPM is an easy exercise for the reader).

We now define another string-processing system that is computationally complete, is suitable for easy simulation by P systems with exo-insertion and exo-deletion, and has a low descriptional complexity of the rule types.

The following contents makes a use of a specific variant of circular Post machines, (N)CPM5, where instructions either read one symbol, or write one symbol, but not both. The corresponding definitions and associated results are presented in Appendix A3.

5.3.1 P systems with priority of exo-deletion

Theorem 5.13 *Any NCPM5 P can be simulated by an eIDP* Π *of size* $(1, 0, 0; 1, 0, 0)$ *with priority of deletion rules over insertion rules.*

The proof is rather involved. We do not present it here, referring the reader to [84]. We note that a similar result has been later obtained in [183], [184], but without priority of deletion over insertion. Although the former result is not a direct consequence of the latter, we recommend the reader to study the latter one first.

Corollary 5.4 *Notice also that the proof does not use deletion at the right.*
$LSP_*(e - ins_1^{0,0} < e - del_1^{0,0}) = LSP_*(e - ins_1^{0,0} < l - del_1^{0,0}) = RE.$

The power of the systems with deletion only at the left and insertion only at the right is an interesting *open* question.

5.3.2 One-sided insertion/deletion without priorities

We emphasize the main open problem we try to attack:

> P: Find a characterization of P systems with exo-insertion of weight one and exo-deletion of weight one without contexts?

A number of related results is known (the references are given in the introduction, we repeat them here in a compact formulation for easy comparison):

- Not computationally complete if operations (even both with weight two) are performed anywhere in the string.
- Computationally complete if insertion has weight two.
- Computationally complete if deletion has weight two.
- Computationally complete for tissue P systems.
- Computationally complete if deletion has priority over insertion (even without deletion on the right).

In this subsection we start show a lower bound, by a construction that only uses one-sided operations.

Lemma 5.1 *Any deterministic finite automaton P can be simulated by an eIDP Π of size $(1,0,0;1,0,0)$ (without priorities) and with insertions and deletions at the right only.*

Proof. Consider a deterministic finite automaton $P = (\Sigma, Q, q_1, F, \sigma)$ with symbols Σ, states Q transition function σ, initial state q_1 and final states $F \subseteq Q$.

We now construct an eIDP $\Pi = (V, \Sigma, \mu, M_{i_s}, \ldots, M_{i_f}, R_{i_s}, \ldots, R_{i_f})$:

$$\begin{aligned}
V &= \Sigma \cup Q \cup \{m_i \mid q_i \in Q\} \\
&\cup \{n_{ij} \mid \sigma(q_i, a_j) = q_l\} \cup \{S\}, \\
\mu &= [\prod_{q_i \in Q} \left([\prod_{\sigma(q_i,a_j)=q_l} [\ ]_{n_{ij}}\ ]_{m_i} \right) [\ ]_{i_f}\ ]_{i_s}, \\
M_{i_s} &= \{q_1\}, \\
M_i &= \emptyset,\ i \neq i_s, \text{ and the rules are given and explained below.}
\end{aligned}$$

The membrane structure of Π consists of the skin membrane i_s, output membrane i_f, inner membranes m_i and n_{ij}, where $q_i \in Q$, $\sigma(q_i, a_j) = q_l$.

$$\begin{aligned}
R_{i_s} &= \{1_i : del_r(q_i, m_i) \mid q_i \in Q\} \cup \{2 : del_r(S, here)\} \\
&\cup \{3_k : del_r(q_k, i_f) \mid q_k \in F\}, \\
R_{m_i} &= \{1_j : ins_r(a_j, n_{ij})\} \cup \{2 : ins_r(S, out)\},\ \sigma(q_i, a_j) = q_l, \\
R_{n_{ij}} &= \{1 : ins_r(q_l, out)\},\ \sigma(q_i, a_j) = q_l.
\end{aligned}$$

We claim that $L(P) = L(\Pi)$: Π generates exactly the language accepted by P. Indeed, eIDP Π starts the computation from a string q_1 in the skin i_s (other regions of Π are empty).

A transition rule of P $\sigma(q_i, a_j) = q_l$, $q_i, q_l \in Q, a_j \in \Sigma$ is simulated as follows: $q_i W \overset{((i_s,1_i),m_i)}{\Longrightarrow} W \overset{((m_i,1_j),n_{ij})}{\Longrightarrow} a_j W \overset{((n_{ij},1),m_i)}{\Longrightarrow} q_l a_j W \overset{((m_i,2_i),i_s)}{\Longrightarrow} S q_l a_j W \overset{((i_s,2),i_s)}{\Longrightarrow} q_l a_j W$. At the end of computation, $q_k \in F$ is removed by rule $3_k : del_r(q_k, i_f)$ and the resulted string appears in the output membrane i_f : $q_k W \overset{((i_s,3_k),i_f)}{\Longrightarrow} W$.

Notice, that in region m_i rule $1_j : ins_r(a_j, n_{ij})$ must be applied (otherwise string SW appears in the skin by rule $2 : ins_r(S, out)$ and will stay there forever). Moreover, this rule 1_j must be applied exactly once, after that rule 2 applies and string $S q_l a_k W$ is sent to the skin. At this point rule $2 : del_r(S, i_s)$ deletes S. Now this string is ready to continue the evolution. Notice that if rule 1_j were applied more than once, a string of the form $S q_l a_k q_l a_k W'$ appears in the skin and the symbols q_l at the right cannot be further deleted. By the given construction, the system Π generates exactly the words accepted by P. Moreover, if a derivation of Π differs from the pattern shown above, then it will contain nonterminals $q_i \notin \Sigma$ that cannot be removed and, hence, such word is not in $L(\Pi)$. Therefore, $L(P) = L(\Pi)$. □

Corollary 5.5 *Since the family of regular languages is closed with respect to taking the mirror image, the following relations hold:*
$ELSP(e - ins_1^{0,0}, e - del_1^{0,0}) \supseteq ELSP(r - ins_1^{0,0}, r - del_1^{0,0}) \supseteq REG.$
$ELSP(e - ins_1^{0,0}, e - del_1^{0,0}) \supseteq ELSP(l - ins_1^{0,0}, l - del_1^{0,0}) \supseteq REG.$

Conclusions In this section we presented a tool — a simple computationally complete string rewriting model, NCPM5 (non-deterministic circular Post machines), for a purpose of allowing as simple simulation as possible by systems with insertion and deletion of prefixes and suffixes. Then we focus on distributed systems inserting and deleting substrings at the ends of the string, called P systems with exo-insertion and exo-deletion without contexts (eIDPs).

The focus of study is the maximal size of inserted and deleted substrings. It is known from [87] that eIDPs are computationally complete with either 1-symbol insertion and 2-symbol deletion, or with 2-symbol insertion and 1-symbol deletion, or even with 1-symbol insertion and 1-symbol deletion in the tissue case. The general problem about 1-symbol insertion and 1-symbol deletion has been partially answered, by showing that the computational completeness holds if we impose the priority of deletion rules over insertion rules. The definite solution of the above mentioned general problem has been presented in [183], [184].

We also gave a number of remarks related to the variants of CPMs and to the interpretations of computational completeness. With respect to the general problem above, in the final part of the section we showed that the family of regular languages is the lower bound, by a construction with one-sided operations.

The previous section is based on publications [88], [89], [90] and [86]. This section is based on publications [84], [87] and [85].

5.4 Splicing

Head splicing systems (H systems) [194] were one of the first theoretical models of biomolecular computing (DNA-computing). The molecules from biology are replaced by words over a finite alphabet and the chemical reactions are replaced by the *splicing* operation. An H system specifies a set of rules used to perform a splicing and a set of initial words or axioms. The computation is done by applying iteratively the rules to the set of words until no more new words can be generated. This corresponds to a bio-chemical experiment where one has enzymes (splicing rules) and initial molecules (axioms) which are put together in a tube and one waits until the reaction stops.

From the formal language theory point of view, the computational power of the obtained model is rather limited, only regular languages can be generated. Various additional control mechanisms were proposed in order to "overcome" this obstacle and to generate all recursively enumerable languages. An overview of such mechanisms can be found in [256].

The main goal of this section is to recall several small universal systems based on splicing. The number of rules is a measure of the size of the system. This approach is coherent with investigations related to small universal Turing machines, e.g. [267].

One of the first ideas to increase the computational power of splicing systems is to consider them in a distributed framework. Such a framework introduces test tubes, corresponding to H systems, which are arranged in a communicating network. The computation is then performed as a repeated sequence of two steps: computation and communication. During the computational step, each test tube evolves as an ordinary H system in an independent manner. During the communication step, the words at each test tube are redistributed

among other tubes according to some communication protocol.

Test tube systems based on splicing, introduced in [156], communicate through redistribution of the contents of the test tubes via filters that are simply sets of letters (in a similar way to the *separate* operation of Lipton-Adleman [218], [1]). These systems, with finite initial contents of the tubes and finite sets of splicing rules associated to each component, are computationally complete, they characterize the family of recursively enumerable languages. The existence of universal splicing test tube distributed systems was obtained on this basis, hence the theoretical proof of the possibility to design universal programmable computers with the structure of such a system. After a series of results, the number of tubes sufficient to achieve this result was established to be 3 [264]. The computational power of splicing test tube systems with two tubes is still an open question. The descriptional complexity for such kind of systems was investigated in [83] where it was shown that there exist universal splicing test tube systems with 10 rules. The best known result shows that there exist universal splicing test tube system with 8 rules [129].

A simple possibility to turn splicing-based systems into computationally complete devices are time-varying distributed H systems (TVDH systems). Such systems work like H systems, but on each computational step the set of active rules is changed in a cycle. These sets are called *components*. It was shown [256] that 7 components are enough for the computational completeness; further this number was reduced to 1 [226], [227]. This last result shows a fine difference between the definitions of a computational step in H systems. If one iterates the splicing operation while keeping all generated strings, then such systems are regular. If only the result of each splicing step is kept, then the resulting systems are computationally complete. An overview of results on TVDH systems may be found in [228]. Recently one constructed very small universal TVDH systems with two components and 15 rules and with one component and 17 rules [83].

Another extension of H systems was done using the framework of P systems [252], see also [186] and [257]. In a formal way, splicing P systems can be considered like a graph, whose nodes contain sets of strings and sets of splicing rules. Every rule permits to perform a splicing and to send the result to some other node. Since splicing P systems generate any recursively enumerable language, it is clear that there are universal splicing P systems. Like for small universal Turing machines, we are interested in such universal systems that have a small number of rules. A first result was obtained in [265] where a universal splicing P system with 8 rules was shown. Recently a new construction was presented in [126], [127] for a universal splicing P system with 6 rules. The best known result [129] shows that there exists a universal splicing P system with 5 rules and this result is presented in this section. Notice, that this result (5 rules) is the best known for "classical" approach to construct small universal devices.

We also consider a class of H systems which can be viewed as a counterpart of the matrix grammars in the regulated rewriting area. These systems are called double splicing extended H systems [256]. In [129] one obtains an unexpected result: 5 rules are enough for such kind of H systems in order to be universal.

The following series of results claims existence of universal devices of very small size. Thus, there exist the following universal devices:

- A double splicing extended H system with 5 rules [129],

- An extended splicing test tube system with 3 tubes with 8 rules [129],
- A TVDH system with two components and 15 rules [83],
- A TVDH system with one component and 17 rules [83],
- A splicing P system with 5 rules [129].

We skip all the technicalities and the actual constructions from the above mentioned results, referring the interested reader to [268], [129], [126], [83] and [127], as well as the corresponding references within this section.

Ciliates Another model of rewriting of (circular) strings is that of *ciliate operations*, inspired from the gene assembly in *Ciliated Protozoa*. It can be viewed as specific synchronized splicing, but does not provide as much control as the models described above. Here we will only list a few results with associated references, without actually describing them.

In [3] and [4] one considers ciliate operations in membrane systems framework, establishing the computational completeness of the intermolecular model, without context, even in the case where the pointers (the key concept used to define the operations) consist of a single symbol. In [281], [113] one performs analysis of pointers in actual living ciliates. Publications [94], [95] and [112] focus on the complexity of the graph-based model of gene assembly in ciliates, while works [109], [110] and [111] attack *Hamiltonian Path Problem*, a well-known NP-complete problem, with ciliate operations.

5.5 Conclusions to Chapter 5

The networks of evolutionary processors and their models and variants have been described, together with the associated results. A few selected results were presented in more details. For instance, NEPs with two nodes are already computationally complete modulo the terminal alphabet. HNEPs with one node are given the precise regular characterization, HNEPs with two nodes are not computationally complete, while seven nodes are enough to reach the computational completeness of HNEPs, even with a complete graph. We should mention that a network over a complete graph (with loops, although it is not important for the last proof) may be viewed as number of agents in a common environment, acting "independently" without explicitly enforcing any transition protocol, where a computationally complete behavior still emerges.

A particularly interesting variant is obligatory HNEPs (OHNEPs). Using a power of the underlying graph, computational completeness is obtained even without the filters. In case of a complete graph, OHNEPs are still computationally complete. Moreover, it suffices that the sum of numbers of symbols in filters of each node does not exceed one.

Several results were showed concerning P systems with insertion and deletion rules of small size. Surprisingly, systems with context-free rules inserting and deleting only one symbol are quite powerful and generate $PsRE$ if the priority of deletion over insertion is used. From the language generation viewpoint such systems are not too powerful and no language specifying the order of symbols can be generated. To be able to generate more complicated

languages we considered systems with one-symbol one-sided insertion or deletion contexts. In both cases we obtained that any recursively enumerable language can be generated. The same result holds if a context-free deletion of two symbols is allowed. The counterpart of the last result is not true, moreover, the insertion of strings of an arbitrary size still cannot lead to generating languages like a^*b^*.

We also have considered one-symbol context-free insertion-deletion P systems without the priority relations and we showed that in terms of Parikh sets these systems characterize $PsMAT$ family. However, in terms of the generated language such systems are strictly included in MAT. We remark that context-free insertions and deletions of one symbol correspond to random point mutations from an evolutionary point of view. We think that the obtained results could give a frame for the modeling of evolutionary processes.

Most of the results above were obtained using rules with target indicators. It is interesting to investigate the computational power of systems with non-specific target indicators *in* or *go*. Another open problem is to replace the priority relation by some other mechanism from the P systems area without decreasing the computational power.

Section 5.1 is based on publications [6], [25], [26], [27], [28], [29], [54], [53], [55], [101], [102], [103], [116], [51] and [52]. Section 5.2 is based on publications [88], [89], [90], [86]. Section 5.3 is based on publications [84], [87] and [85]. Section 5.4 is based on publications [129], [126], [83], [127], as well as the ciliate direction, based on publications [3], [4], [281], [113], [94], [95], [112], [109], [110] and [111].

6. APPLICATIONS

A few applications of membrane systems (sorting, solving NP-complete problem and modeling bi-stable catalysts by protons) are described in my Ph.D. thesis [5]. A larger number of applications is presented in the book containing [132]. In this chapter we present some more applications obtained in the recent author's publications.

In Section 6.1 we present a formalization of inflection process for the Romanian language using the model of P systems with cooperative string replication rules, which will make it possible to automatically build the morphological lexicons as a base for different linguistic applications. We also mention the solution to the problem of annotating affixes, by the process of reverse derivation.

In Section 6.2 we describe the work with the prefix tree by P systems with strings and active membranes. We present the algorithms of searching in a dictionary and updating it implemented as membrane systems. The systems are constructed as reusable modules, so they are suitable for using as sub-algorithms for solving more complicated problems.

In Section 6.3 we consider the problem of synchronizing the activity of all membranes of a P system. After pointing at the connection with a similar problem in the field of cellular automata where the problem is called the *firing squad synchronization problem*, $FSSP$ for short, we focus on the deterministic algorithm to solve this problem. Our algorithm works in $3h + 3$, where h is the height of the tree describing the membrane structure.

In Section 6.4 we present a variant of the multiset rewriting model where the rules of every region are defined by the contents of interior regions, rather than being explicitly specified in the description of the system. This idea is inspired by the von Neumann's concept of "program is data" and also related to the research direction proposed by Gh. Păun about the cell nucleus.

The author has published a number of other works on application of P systems and other formal computing models. In Section 6.5 we list some of them here, without describing them in detail. These include performing logical inference by membrane systems, called "chaining", encoding numbers by multisets and discussion of the questions of compactness of the representation and of the efficiency of performing the arithmetic operations in different multiset representations, right self-assembly of a double-stranded molecule from a single-stranded words of bounded length, cellular automata that possess a specific invariant, a so-called *number-conservativity*, and reversible logical elements with memory.

6.1 Inflections

The aim of this section is to present a formalization of inflection process for the Romanian language using the model of P systems with cooperative string replication rules, which will make it possible to automatically build the morphological lexicons as a base for different linguistic applications.

Natural language processing has a wide range of applications, the spectrum of which varies from a simple spell-check up to automatic translation, text and speech understanding, etc. The development of appropriate technology is extremely difficult due to the specific feature of multidisciplinarity of the problem. This problem involves several fields such as linguistics, psycholinguistics, computational linguistics, philosophy, computer science, artificial intelligence, etc.

As in many other fields, solving of a complex problem is reduced to finding solutions for a set of simpler problems. In our case among the items of this set we find again many traditional compartments of the language grammar. The subject of our interest is the morphology, and more specifically, its inflectional aspect.

The inflectional morphology studies the rules defining how the inflections of the words of a natural language are formed, i.e., the aspect of form variation (of the inflection, which is the action of words modification by gender, number, mood, time, person) for various expressing grammatical categories.

In terms of natural language typology the morphological classification can be *analytical* and *synthetic*. Of course, this classification is a relative one, having, however, some irrefutable poles: Chinese, Vietnamese, as typical representatives of the analytical group, and Slavic and Romance languages serving as examples of synthetic ones. The English language, with a low degree of morpheme use, is often among the analytical ones, sometimes is regarded as synthetic, indicating however that it is "less synthetic" than other languages from the same group. It is evident that it is the inflectional morphology of synthetic languages that presents special interest, being a problem more complex than the analytical class.

The object of our studies is the Romanian language, which belongs to the category of synthetic flective languages. The last notion stresses the possibility to form new words by declension and conjugation. Moreover, the Romanian language is considered a highly inflectional language, because the number of word-forms is big enough.

The inflection simplicity in English makes that the majority of researchers in the field of computational linguistics neglect the inflection morphology. For efficient processing of other natural languages, including Romanian, it is necessary to develop suitable computational models of morphology of each language.

A model of inflection process is a formalism, which should include two processes:

- making the alternation in the root, and
- concatenation of a flective.

In general case, from a whole variety of inflection groups, we can identify two classes:

– without alternations, and

– with alternations.

In the first case the inflection is made in the following manner. Let $\Im$ be a set formed from lists of flectives, $F = \{f_1, f_2, \cdots, f_n\}$, $w = w'\alpha$ is a word-lemma, where $|\alpha| \geq 0$. In the

simplest case the inflected words will be those of the form $w'f_i$, $f_i \in F$, $(i = 1, \cdots, n)$.

General case: Let $w = w_1a_1w_2a_2 \cdots w_m\alpha$. The inflected words will be of the form:

$$\begin{array}{lccccccc} w^{(1)} = w_1 & a_1 & w_2 & a_2 & \cdots & w_m f_{i_1}, \\ w^{(2)} = w_1 & u_1^{(2)} & w_2 & u_2^{(2)} & \cdots & w_m f_{i_2}, \\ \cdots & & & & & \\ w^{(s)} = w_1 & u_1^{(s)} & w_2 & u_2^{(s)} & \cdots & w_m f_{i_s}, \end{array}$$

where w_i, $a_i \in V^+$, $u_i^{(j)} \in V^*$, $f_{i_1} \in F^{(1)}, \ldots, f_{i_s} \in F^{(s)}$, and $F^{(1)} \cup \ldots \cup F^{(s)}$ forms a complete paradigm. Note: the analysis of inflection rules allowed us to ascertain that for the Romanian language $m \leq 4$, $s \leq 3$.

Modern dictionaries contain hundreds of thousands of words–lemma. Their forms of inflexion (the amount of which exceeds millions) are needed for developing various applications based on natural language: from the spell-checker up to the systems understanding the speech. Obviously, to solve the problem of creating a dictionary with a morphologically representative coverage, as well as to build various applications based on it, effective mechanisms are needed, especially those that allow parallel processing. One of the possible ways to perform parallel computation is based on biological models.

To formalize the inflection process for the Romanian language the model of cooperative membrane P systems with replication will be used [252].

Describing the inflection process by P systems Let us define the P system performing the inflection process. Let L be the set of words which form opened productive classes. We will start by assuming that the words in L are divided into groups of inflection, i.e. for each $w \in L$ the number of inflection group is known [219]. The inflection group is characterized by the set $G = \{\alpha, R_G, F_G\}$, where $|\alpha| \geq 0$ is the length of ending which is reduced in the process of inflection, F_G is the set of the lists of flectives, the assembly of which forms complete paradigm, R_G is the set of the rules, which indicate vowel/consonant alternation of type $a \rightarrow u$, $a \in V^+$, $u \in V^*$, and also the conformity of the roots obtained by the lists of flectives from F_G. To each group of inflexion a membrane system Π_G will be put into correspondence.

As it was mentioned earlier, we will investigate two cases:

– without alternations, and

– with vowel/consonant alternation.

The first model is very simple. For any group $G = (\alpha, \emptyset, \{f_{1_G}, f_{2_G}, \cdots, f_{n_G}\})$ of inflection without alternation,

$$\begin{array}{rcl} \Pi_G & = & (O, \Sigma, [\ ]_1, \emptyset, R_1, 1), \text{ where} \\ O & = & \Sigma = V \cup \{\#\}, \\ V & = & \{\mathrm{a}, \cdots, \mathrm{z}\} \text{ is the alphabet of the Romanian language, and} \\ R_1 & = & \{\alpha\# \rightarrow (f_{1_G}, out)||(f_{2_G}, out)|| \cdots ||(f_{n_G}, out)\} \end{array}$$

If this system receives as an input the words $w'\alpha\#$, where $w'\alpha$ corresponds to the inflection group G, then it sends all its inflected words out of the system in one step. Clearly, Π_G is

non-cooperative if $\alpha = \lambda$, but non-cooperativeness is too restrictive in general, since then the system would not be able to distinguish the termination to be reduced from any other occurrence of α.

The general model will require either a more complicated structure, or a more sophisticated approach. Let G be an arbitrary inflection group, with $m-1$ alternations $a_1 = a_1^{(1)} a_2^{(1)} \cdots a_{n_1}^{(1)}, \cdots, a_m = a_1^{(m)} a_2^{(m)} \cdots a_{n_m}^{(m)}$. Let the set of flectives consist of s subsets, and for subset $F_{k_G} = \{f_1^{(k)}, \cdots, f_{p_1}^{(k)}\}$, $1 \leq k \leq s$, the following alternations occur: $a_1 \to u_1^{(k)}$, $\cdots$, $a_m \to u_m^{(k)}$ (the alternations are fictive for $k = 1$), and $\bigcup_{k=1}^{s} F_{k_G}$ corresponds to a complete paradigm. For instance, Example 2 corresponds to $s = 2$ sublists (singular and plural), and $m - 1 = 2$ alternations.

The associated P system should perform the computation

$$w\# = \prod_{j=1}^{m-1} (w_j a_j)\, w_m \alpha\# \Rightarrow^* \left\{ \prod_{j=1}^{m-1} \left(w_j u_j^{(k)} \right) w_m f_{i_k} \mid 1 \leq k \leq s,\ f_{i_k} \in F^{(k)} \right\},$$

where $u_j^{(1)} = a_j$, $1 \leq j \leq m$.

The first method assumes the alternating subwords a_j are present in the input word in just one occurrence, or marked. Moreover, we assume that carrying out previous alternations does not introduce more occurrences of the next alternations.

For modeling such process of inflection for the group G we define the following P system with $1 + (s-1)m$ membranes

$$\begin{aligned}
\Pi'_G &= (O, \Sigma, \mu, \emptyset, \cdots, \emptyset, R_1, \cdots, R_{1+(s-1)m}, 1), \text{ where} \\
\Sigma &= V \cup \{\#\},\ O = \Sigma \cup E, \\
\mu &= [\,[\ ]_2[\ ]_3 \cdots [\ ]_{1+(s-1)m}\,]_1, \\
E &= \{\#_k \mid 2 \leq k \leq s\} \cup \{A_{k,j} \mid 1 \leq k \leq s,\ 1 \leq j \leq m\}, \\
V &= \{\mathrm{a}, \cdots, \mathrm{z}\} \text{ is the alphabet of the Romanian language,}
\end{aligned}$$

(V can be extended by marked letters if needed), and the rules are given below.

$$\begin{aligned}
R_1 &= \{\alpha\# \to A_{1,m} || (\#_2, in_2) || \cdots || (\#_s, in_s)\} \\
&\cup\ \{A_{k,j} \to (\lambda, in_{k+(s-1)j}) \mid 2 \leq k \leq s, 1 \leq j \leq m-1\} \\
&\cup\ \{A_{k,m} \to (f_1^{(k)}, out) || \cdots || (f_{p_m}^{(k)}, out) \mid 1 \leq k \leq s\}, \\
R_{k+(s-1)(j-1)} &= \{a_j \to (u_j^{(k)} A_{k,j}, out)\},\ 2 \leq k \leq s,\ 1 \leq j \leq m-1, \\
R_{k+(s-1)(m-1)} &= \{\#_k \to (A_{k,m}, out)\},\ 2 \leq k \leq s.
\end{aligned}$$

The work of P system Π'_G is the following. First, s copies of the string are made, and the first one stays in the skin, while others enter regions $2, \cdots, s$. Each copy in region k is responsible to handle the k-th subset of inflections. The first one simply performs a replicative substitution in the end, and sends the results out, in the same way as Π_G works. Consider a copy of the input in region k, $2 \leq k \leq s$. When j-th alternation is carried out, the string returns to the skin, and symbol $A_{k,j}$ is produced. This symbol will be used to send the string in the corresponding region to carry out alternation $j+1$. Finally, if $j = m$, then the system performs a replicative substitution in the end, and sends the results out.

Assuming $s \geq 2$, the system halts in $2m+1$ steps, making an efficient use of scattered rewriting with parallel processing of different inflection subsets. For instance, the inflection group from Example 2 would transform into a P systems with 4 membranes, halting in 7 steps. Notice that this system is non-cooperative if $\alpha = \lambda$ and $|a_j| = 1$, $1 \leq j \leq m$. It is also worth noticing that it is possible to reduce the time to $m+1$ steps by using tissue P systems with parallel channels.

The second method avoids the limiting assumptions of the first methods. More exactly, it performs the first alternation at its leftmost occurrence, the second alternation at its leftmost occurrence which is to the right of the first one, etc. Formally, such a P system discovers the representation of the input string as $\prod_{j=1}^{m-1} (w_j a_j)\, w_m \alpha$, where a_j has no other occurrences inside $w_j a_j$ except as a suffix.

A theoretical note: overlapping occurrences or occurrences with context can be handled by rules with a longer left-hand side. A different order of occurrences of the alternations can be handled by renumbering the alternations. Should the specification of a group require, e.g., second-leftmost occurrence for $a \to u$, this can be handled by inserting a fictive substitution $a \to a$ before $a \to u$, etc. Therefore, this is the most general method.

We construct the following P system, which takes the input in the form

$$\#_l w \#_r = \#_l \prod_{j=1}^{m-1} (w_j a_j)\, w_m \alpha \#_r.$$

$$\begin{aligned}
\Pi''_G &= (O, \Sigma, [\ ]_1, \emptyset, R_1, 1), \text{ where } \Sigma = V \cup \{\#_l, \#_r\},\ O = \Sigma \cup E,\\
E &= \{A_{k,j} \mid 1 \leq k \leq s,\ 0 \leq j \leq m\},\\
V &= \{a, \cdots, z\} \text{ is the Romanian alphabet,}
\end{aligned}$$

$$\begin{aligned}
R_1 &= \{\#_l \to A_{1,0} || \cdots || A_{s,0}\} && (6.1)\\
&\cup \{A_{k,j-1}\gamma \to \gamma A_{k,j-1} \mid \gamma \in V \setminus \{a_1^{(j)}\},\\
&\quad 1 \leq k \leq s,\ 1 \leq j \leq m\} && (6.2)\\
&\cup \{A_{k,j-1} a_1^{(j)} v\gamma \to a_1^{(j)} A_{k,j-1} v\gamma \mid a_1^{(j)} v \in Pref(a_j),\\
&\quad |v| < |a_j| - 1, \gamma \in V \setminus \{a_1^{(|v|+2)}\},\ 1 \leq k \leq s, 1 \leq j \leq m\} && (6.3)\\
&\cup \{A_{k,j-1} a_j \to u_j^{(k)} A_{k,j} \mid 1 \leq k \leq s,\ 1 \leq j \leq m\} && (6.4)\\
&\cup \{\alpha A_{k,m} \#_r \to (f_1^{(k)}, out) || \cdots || (f_{p_m}^{(k)}, out) \mid 1 \leq k \leq s\}. && (6.5)
\end{aligned}$$

The rules are presented as a union of 5 sets. The rule in the first set replicates the input for carrying out different inflection subsets. The symbol $A_{k,j}$ is a marker that will move through the string. Its index k corresponds to the inflection subset, while index j tells how many alternations have been carried out so far.

The rules in the second set allow the marker to skip a letter if it does not match the first letter needed for the current alternation. The rules in the third set allow the marker to skip one letter if some prefix of the needed subword is found, followed by a mismatch. The rules in the fourth set carry out an alternation, and the last set of rules perform the replicative substitution of the flectives.

This system halts in at most $|w| + 2$ steps.

For the question of determining the inflection group, we refer the interested readers to [154], [153] and [31].

Annotating affixes A more complicated problem is, given a derived word, to produce its lexical decomposition. An attempt to attack this problem using membrane systems is made in [48].

Conclusions The membrane system to describe the inflectional process when the inflectional morphological model is known is investigated in this section.

In the case when the model is not known in advance, it can be determined by using the algorithm from [153]. The membrane systems presented in this section can be also adapted for other natural languages with high level of inflection, such as Italian, French, Spanish etc., having structured morphological dictionaries, similar to the Romanian one.

6.2 Dictionary

In this section we describe the work with the prefix tree by P systems with strings and active membranes. We present the algorithms of searching in a dictionary and updating it implemented as membrane systems. The systems are constructed as reusable modules, so they are suitable for using as sub-algorithms for solving more complicated problems.

Solving most problems of natural language processing is based on using certain linguistic resources, represented by corpora, lexicons, etc. Usually, these collections of data constitute an enormous volume of information, so processing them requires much computational resources. A reasonable approach for obtaining efficient solutions is that based on applying parallelism; this idea has been promoted already in 1970s. Many of the stages of text processing (from tokenization, segmentation, lematizing to those dealing with natural language understanding) can be carried out by parallel methods. This justifies the interest to the methods offered by the biologically inspired models, and by membrane computing in particular.

However, there are some issues that by their nature do not allow complete parallelization, yet exactly they are often those "computational primitives" that are inevitably used during solving major problems, like the elementary arithmetic operations are always present in solving difficult computational problems. Among such "primitives" in the computational linguistics we mention handling of the dictionaries, e.g., dictionary lookup and dictionary update. Exactly these problems constitute the subject of the present section. In our approach we speak about dictionary represented by a prefix tree.

P (membrane) systems are a convenient framework of describing computations on trees. Since membrane systems are an abstraction of living cells, the membranes are arranged hierarchically, yielding a tree structure.

Dictionary Dictionary search represents computing a string-valued function $\{u_i \longrightarrow v_i \mid 1 \leq i \leq d\}$ defined on a finite set of strings.

We represent such a dictionary by the skin membrane containing the membrane structure corresponding to the prefix tree of $\{u_i \mid 1 \leq i \leq d\}$, with strings $\$v_i\$'$ in regions corresponding to the nodes associated to u_i. Let A_1, A_2 be the source and target alphabets: $u_i \in A_1^*$, $v_i \in A_2^*$, $1 \leq i \leq d$. Due to technical reasons, we assume that for every $l \in A_1$, the skin contains a membrane with label l. We also suppose that the source words are non-empty.

For instance, the dictionary $\{\text{bat} \longrightarrow \text{flying}, \text{bit} \longrightarrow \text{stored}\}$ is represented by

$$[\ [\ ]_a^0[\ [\ [\ \$flying\$'\ ]_t^0\ ]_a^0[\ [\ \$stored\$'\ ]_t^0\ ]_i^0\ ]_b[\ \ ]_c^0 \cdots [\ \ ]_z^0\ ]_0^0$$

Consider a P system corresponding to the given dictionary:

$$\begin{array}{rcl}
\Pi & = & \big(O, \Sigma, H, E, \mu, M_1, \cdots, M_p, R, i_0\big), \\
O & = & A_1 \cup A_2 \cup \{?, ?', \$, \$', \$_1, \$_2, \texttt{fail}\} \\
 & \cup & \{?_i \mid 1 \leq i \leq 11\} \cup \{!_i \mid 1 \leq i \leq 4\}, \\
\Sigma & = & A_1 \cup A_2 \cup \{?, ?', !, \$, \$'\}, \\
H & = & A_1 \cup \{0\},\ E = \{0, +, -\},\ i_0 = 1, \\
\mu & \text{and} & \text{sets } M_i,\ 1 \leq i \leq p, \text{ are defined as described above.}
\end{array}$$

So only the rules and input semantics still have to be defined.

6.2.1 Dictionary search

To translate a word u, input the string $?u?'$ in region 1. Consider the following rules.

S1 $?l[\ \ ]_l^0 \to [\ ?\ ]_l^0,\ l \in A_1$

Propagation of the input into the membrane structure, reaching the location corresponding to the input word.

S2 $[\ ??'\ ]_l^0 \to [\ \ ]_l^-\emptyset,\ l \in A_1$

Marking the region corresponding to the source word.

S3 $[\ \$ \to \$_1||\$_2\ ]_l^-,\ l \in A_1$

Replicating the translation.

S4 $[\ \$_2\ ]_l^e \to [\ \ ]_l^0\$_2,\ l \in H,\ e \in \{-, 0\}$

Sending one copy of the translation to the environment.

S5 $[\ \$_1 \to \$\ ]_l^0,\ l \in A_1$

Keeping the other copy in the dictionary.

The system will send the translation of u in the environment. This is a simple example illustrating search. If the source word is not in the dictionary, the system will be blocked without giving an answer. The following subsection shows a solution to this problem.

6.2.2 Search with fail

The set of rules below is considerably more involved than the previous one. However, it handles 3 cases: a) the target word is found, b) the target word is missing in the target location, c) the target location is unreachable.

F1 $[\ ?\rightarrow ?_1||?_2\ ]_0^0$

Replicate the input.

F2 $[\ ?_2\rightarrow ?_3\ ]_0^0$

Delay the second copy of the input for one step.

F3 $?_1 l[\ \ ]_l^0 \rightarrow [\ ?_1\ ]_l^+,\ l\in A_1$

Propagation of the first copy towards the target location, changing the polarization of the entered membrane to $+$.

F4 $?_3 l[\ \ ]_l^+ \rightarrow [\ ?_3\ ]_l^0,\ l\in A_1$

Propagation of the second copy towards the target location, restoring the polarization of the entered membrane.

F5 $[\ ?_1 l \rightarrow [\ ?_4\ ]_l^-\ ]_k^0,\ l,k\in A_1$

If a membrane corresponding to some symbol of the source word is missing, then the first copy of the input remains in the same membrane, while the second copy of the input restores its polarization. Creating a membrane to handle the failure.

F6 $[\ ?_1?'\rightarrow ?_7\ ]_l^0,\ l\in A_1$

Target location found, marking the first input copy.

F7 $[\ ?_7\ ]_l^0 \rightarrow [\ \ ]_l^-\emptyset,\ l\in A_1$

Marking the target location.

In either case, some membrane has polarization $-$. It remains to send the answer out, or fail if it is absent. The membrane should be deleted in the fail case.

F8 $[\ \$\rightarrow \$_1||\$_2\ ]_l^-,\ l\in A_1$

Replicating the translation.

F9 $[\ \$_2\ ]_l^e \rightarrow [\ \ ]_l^0\$_2,\ l\in H,\ e\in\{0,-\}$

Sending one copy of the translation out.

F10 $[\ \$_1\rightarrow \$\ ]_l^0,\ l\in A_1$

Keeping the other copy in the dictionary.

F11 $[\ ?_3\rightarrow ?_5\ ]_l^-,\ l\in A_1$

The second copy of input will check if the translation is available in the current region.

F12 $?_3 l[\ \ ]_l^- \to [\ ?_5\ ]_l^-,\ l \in A_1$

The second copy of input enters the auxiliary membrane with polarization $-$.

By now the second copy of the input is in the region corresponding to either the search word, or to its maximal prefix plus one letter (auxiliary one).

F13 $[\ ?_5 \to ?_6\ ]_l^-,\ l \in A_1$

It waits for one step.

F14 $[\ ?_6 \to \emptyset\ ]_l^0,\ l \in A_1$

If the target word has been found, the second copy of the input is erased.

F15 $[\ ?_6\ ]_l^- \to [\ \ ]_l^0 ?_8,\ l \in A_1$

If not, the search fails.

F16 $[\ ?_8\ ]_l^0 \to [\ \ ]_l^0 ?_8,\ l \in A_1$

Sending the fail notification to the skin.

F17 $[\ ?_8 l \to ?_8\ ]_0^0$

Erasing the remaining part of the source word.

F18 $[\ ?_8 ?'\ ]_0^0 \to [\ \ ]_0^0 \texttt{fail}$

Answering fail.

F19 $[\ ?_4 \to ?_9\ ]_l^-,\ l \in A_1$

F20 $[\ ?_9 \to ?_{10}\ ]_l^-,\ l \in A_1$

F21 $[\ ?_{10} \to ?_{11}\ ]_l^-,\ l \in A_1$

If the target location was not found, the first input copy waits for 3 steps while the membrane with polarization $-$ handles the second input copy.

F22 $[\ ?_{11}\ ]_l^0 \to \emptyset,\ l \in A_1$

Erasing the auxiliary membrane.

6.2.3 Dictionary update

To add an entry $u \longrightarrow v$ to the dictionary, input the string $!u\$v\$'$ in region 1. Consider the following rules.

U1 $[\ ! \rightarrow !_1 || !_2\ ]_0^0$

Replicate the input.

U2 $[\ !_2 \rightarrow !_3\ ]_0^0$

Delay the second copy of the input for one step.

U3 $!_1 l[\ \ ]_l^0 \rightarrow [\ !_1\ ]_l^+,\ l \in A_1$

Propagation of the first copy towards the target location, changing the polarization of the entered membrane to +.

U4 $!_3 l[\ \ ]_l^+ \rightarrow [\ !_3\ ]_l^0,\ l \in A_1$

Propagation of the second copy towards the target location, restoring the polarization of the entered membrane.

U5 $[\ !_1 \rightarrow !_4\ ]_l^0,\ l \in A_1$

If a membrane corresponding to some symbol of the source word is missing, then the first copy of the input remains in the same membrane, while the second copy of the input restores its polarization. Marking the fist copy of the input for creation of missing membranes.

U6 $[\ !_4 l \rightarrow [\ !_4\ ]_l^+\ ]_k^0,\ l, k \in A_1$

Creating missing membranes.

U7 $[\ !_4\$ \rightarrow \$\ ]_l^0,\ l \in A_1$

Releasing the target word in the corresponding location.

U8 $[\ !_3\$ \rightarrow \emptyset\ ]_l^0,\ l \in A_1$

Erasing the second copy of the input.

We underline that the constructions presented above also hold in a more general case, i.e., when the dictionary is a multi-valued function. Indeed, multiple translations can be added to the dictionary as multiple strings in the region associated to the input word. The search for a word with multiple translations will lead to all translations sent to the environment. The price to pay is that the construction is no longer deterministic, since the order of application of rules S4 or F9 to different translations is arbitrary. Nevertheless, the constructions remain "deterministic modulo the order in which the translations are sent out". All constructions work in linear time with respect to the length of the input. The parallelism is vital for checking for the absence of a needed submembrane, or for checking for the absence of a translation of a given word; sending multiple translation results out is also parallel.

Discussion In this section we presented the linear-time algorithms of searching in a dictionary and updating it implemented as membrane systems. We underline that the systems are constructed as reusable modules, so they are suitable for using as sub-algorithms for solving more complicated problems.

The scope of handling dictionaries is not limited to the dictionaries in the classical sense. Understanding a dictionary as introduced in Subsection 6.2, i.e., a string-valued function defined on a finite set of strings, leads to direct applicability of the proposed methods to handle alphabets, lexicons, thesauruses, dictionaries of exceptions, and even databases. At last, it is natural to consider these algorithms together with morphological analyzer and morphological synthesizer.

We have presented a few applications of abstract computing models in linguistics.

6.3 Synchronization

Consider the problem of synchronizing the activity of all membranes of a P system. After pointing at the connection with a similar problem dealt with in the field of cellular automata where the problem is called the *firing squad synchronization problem*, $FSSP$ for short, we provide two algorithms to solve this problem. One algorithm is non-deterministic and works in $2h + 3$, the other is deterministic and works in $3h + 3$, where h is the height of the tree describing the membrane structure.

The synchronization problem can be formulated in general terms with a wide scope of application. We consider a system constituted of explicitly identified elements and we require that starting from an initial configuration where one element is distinguished, after a finite time, all the elements which constitute the system reach a common feature, which we call **state**, all at the same time and the state was never reached before by any element.

This problem is well known for cellular automata, where it was intensively studied under the name of the *firing squad synchronization problem* (FSSP): a line of soldiers have to fire at the same time after the appropriate order of a general which stands at one end of the line, see, e.g., references in [97], [98]. The first solution of the problem was found by Goto, see [192]. It works on any cellular automaton on the line with n cells in the minimal time, $2n-2$ steps, and requiring several thousands of states. A bit later, Minsky found his famous solution which works in $3n$, see [233] with a much smaller number of states, 13 states. Then, a race to find a cellular automaton with the smallest number of states which synchronizes in $3n$ started. See the above papers for references and for the best results and for generalizations to the planar case, see [276] for results and references.

The synchronization problem appears in many different contexts, in particular in biology. As P systems model the work of a living cell constituted of many micro-organisms, represented by its membranes, it is a natural question to raise the same issue in this context. Take as an example the meiosis phenomenon, it probably starts with a synchronizing process which initiates the division process. Many studies have been dedicated to general synchronization principles occurring during the cell cycle; although some results are still controversial, it is widely recognized that these aspects might lead to an understanding of general biological principles used to study the normal cell cycle, see [273].

We may translate FSSP in P systems terms as follows. Starting from the initial configuration where all membranes, except the root, contain same objects the system must reach a configuration where all membranes contain a distinguished symbol, F. Moreover, this symbol must appear in all membranes only during at the synchronization time.

The synchronization problem as defined above was studied in [145] for two classes of P systems: transitional P systems and P systems with priorities and polarizations. In the first case, a non-deterministic solution to FSSP was presented and for the second case a deterministic solution was found. These solutions need time $3h$ and $4n + 2h$ respectively, where n is the number of membranes and h is the depth of the membrane tree.

In this section we recall a significant improvement of the previous results in the non-deterministic case. In the deterministic case, another type of P system was considered and this permitted to improve the parameters. The new algorithms synchronize the corresponding P systems in $2h + 3$ and $3h + 3$ respectively.

In the sequel, we will use transitional P systems without a distinguished compartment as an output, i_0, as this is not relevant for FSSP.

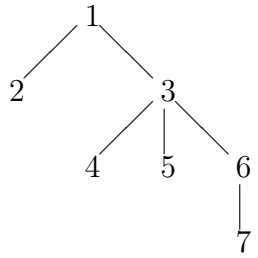
We translate the FSSP to P systems as follows:

Problem 6.1 *For a class of P systems $\mathcal{C}$ find two multisets $\mathcal{W}, \mathcal{W}' \in O^*$, and two sets of rules $\mathcal{R}$, $\mathcal{R}'$ such that for any P system $\Pi \in \mathcal{C}$ of degree $n \geq 2$ having*

$w_1 = \mathcal{W}'$, $R_1 = \mathcal{R}'$, $w_i = \mathcal{W}$ and $R_i = \mathcal{R}$ for all i in $\{2..n\}$, assuming that the skin membrane has the number 1, the following hold:

- *If the skin membrane is not taken into account, then the initial configuration of the system is stable (cannot evolve by itself).*
- *If the system halts, then all membranes contain the designated symbol F which appears only at the last step of the computation.*

Example 1 We present now an example and discuss the functioning of the system on it. Consider a system Π having 7 membranes with the following membrane structure:



Although non-deterministic solution is an interesting exercise, we do not present it here to keep the presentation reasonably concise. An interested reader can find it in [97] and [98].

6.3.1 Deterministic case

Consider now the deterministic case. We take the class of P systems with promoters and inhibitors and solve Problem 1 for this class.

The idea of the algorithm is very simple. A symbol C_2 is propagated down to the leaves and at each step, being at a inner node, it sends back a signal C. At the root a counter starts to compute the height of the tree and it stop if and only if there are no more signals C. It is easy to compute that the last signal C will arrive at time $2h-1$ (there are h inner nodes, and the last signal will continue for $h-1$ steps). At the same time the height is propagated down the tree as in the non-deterministic case.

Below is the formal description of the system.

The P system $\Pi = (O, \mu, w_1, \cdots, w_n, R_1, \cdots, R_n)$ for deterministic synchronization is present below. We consider that μ is an arbitrary membrane structure. The set of objects is $O = \{S_1, S_2, S_3, S_4, S, \bar{S}, S', S'', S''', C_1, C_2, C, a, a', b, F\}$, the initial contents of the skin is $w_1 = \{S_1\}$, the other membranes are empty. The set of rules $R_1, \ldots, R_n$ are identical, they are presented below.

Start:

$$S_1 \to S_2; C_2'; S, in!; C_1, in! \tag{6.6}$$

Propagation of S:

$$S \to \bar{S}; S, in! \tag{6.7}$$

Propagation of C (height computing signal):

$$C_1 \to C_1, in! \qquad C_2 \to C; C_2, in!; C, out \tag{6.8}$$

$$C_1 C_2 \to \lambda \qquad C_2' \to C; C_2, in! \tag{6.9}$$

$$C \to C, out \tag{6.10}$$

Root counter:

$$S_2 \to S_3 \qquad S_3 \to S_3'; b; a, in! \mid_C \tag{6.11}$$

$$C \to \lambda \mid_{S_3} \qquad S_3' \to S_3 \mid_C \tag{6.12}$$

$$C \to \lambda \mid_{S_3'} \qquad S_3' \to S_4; a', in! \mid_{\neg C} \tag{6.13}$$

Propagation of a:

$$\bar{S}a \to S' \qquad a \to b; a, in! \mid_{S'} \tag{6.14}$$

End propagate of a:

$$a'S' \to S''; a', in! \qquad a'Sa \to S''' \tag{6.15}$$

Decrement:

$$S''b \to S'' \qquad S'''a \to S''' \tag{6.16}$$

$$S'' \to F \mid_{\neg b} \qquad S''' \to F \mid_{\neg a} \tag{6.17}$$

Root decrement:

$$S_4 b \to S_4 \qquad S_4 \to F \mid_{\neg b} \tag{6.18}$$

We now give a structural explanation of the system. Rule (6.6) produces four objects. Similar to the system from the previous section, the propagation of object S by (6.7) leads to marking the intermediate nodes by $\bar{S}$ and the leaves by S. While objects C_1, C_2 propagate down the tree structure and send a continuous stream of objects C up to the root by (6.8)-(6.10), object S_2 counts, producing by rules (6.11)-(6.13) an object b every other step.

When the counting stops, there will be exactly h copies of object b in the root. Similar to the construction from the previous section, objects a are produced together with objects b by the second rule from (6.11). Objects a are propagated down the structure and decremented by one at every level by (6.14).

After the counting stops in the root (the last rule from (6.13)), object a' is produced. It propagates down the tree structure by (6.15), leading to the appearance of objects S'' in the intermediate nodes and S''' in the leaves. These two objects perform the countdown and the corresponding nodes fire by (6.16). The root behaves in a similar way by (6.18).

The correctness of the construction can be argued as follows. It takes $h+1$ steps for a symbol C_2 to reach all leaves. All this time, symbols C are sent up the tree. It takes further $h-1$ steps for all symbols C to reach the root node, and one more step until symbols C disappear. Therefore, symbols b appear in the root node every odd step from step 3 until step $2h+1$, so h copies will be made. Together with the production of b^h in the root node, this number propagates down the tree, being decremented by one at each level. For the depth i, the number $h-i$ is represented, during propagation, by the multiplicity of symbols a (one additional copy of a is made) in the leaves and by the multiplicity of symbols b in non-leaf nodes. After $2h+2$ steps, the root node starts the propagation of the countdown (i.e., decrement of symbols a or b). For a node of depth i, it takes i steps for the countdown signal (a') to reach it, another $h-i$ steps to eliminate symbols a or b, so every node fires after $2h+2+i+(h-i)+1 = 3h+3$ steps after the synchronization has started.

Example 2 Consider a P system having the membrane structure given in Example 1. We present below the evolution of the system in this case.

Step	w_1	w_2	w_3	w_4	w_5	w_6	w_7
0	S_1						
1	S_2C_2'	SC_1	SC_1				
2	S_3C	SC_1C_2	$\bar{S}C_2$	SC_1	SC_1	SC_1	
3	$S_3'bC$	Sa	$\bar{S}aC$	SC_1C_2	SC_1C_2	$\bar{S}C_2$	SC_1
4	S_3bC	Sa	$S'C$	S	S	$\bar{S}C$	SC_1C_2
5	$S_3'bbC$	Saa	$S'aC$	S	S	$\bar{S}$	S
6	S_3bbC	Saa	$S'b$	Sa	Sa	$\bar{S}a$	S
7	$S_3'bbb$	$Saaa$	$S'ba$	Sa	Sa	S'	S
8	S_4bbb	$a'Saaa$	$a'S'bb$	Saa	Saa	$S'a$	S
9	S_4bb	$S'''aa$	$S''bb$	$a'Saa$	$a'Saa$	$a'S'b$	Sa
10	S_4b	$S'''a$	$S''b$	$S'''a$	$S'''a$	$S'b$	$a'Sa$
11	S_4	S'''	S''	S'''	S'''	S'	S'''
12	F	F	F	F	F	F	F

Conclusions In this section we presented two algorithms that synchronize two given classes of P systems. The first one is non-deterministic and it synchronizes the class of transitional P systems (with cooperative rules) in time $2h + 3$, where h is the depth of the membrane tree. The second algorithm is deterministic and it synchronizes the class of P systems with promoters and inhibitors in time $3h + 3$.

It is worth to note that the first algorithm has the following interesting property. After $2h$ steps either the system synchronizes and the object F is introduced, or an object # will be present in some membrane. This property can be used during an implementation in order to cut off failure cases.

The results obtained in this article rely on a rather strong target indication, $in!$, which sends an object to all inner membranes. Such a synchronization was already considered in neural-like P systems where it corresponds to the target *go*. It would be interesting to investigate what happens if such target is not permitted. However, we conjecture that a synchronization would be impossible in this case.

The study of the synchronization algorithms for different classes of P systems is important as it permits to implement different synchronization strategies which are important for such a parallel device as P systems. In particular, with such approach it is possible to simulate P systems with multiple global clocks by P systems with one global clock. It is particulary interesting to investigate the synchronization problem for P systems which cannot create new objects, for example for P systems with symport/antiport.

This section is based on publications [97] and [98].

6.4 Polymorphism

In this section we present a variant of the multiset rewriting model where the rules of every region are defined by the contents of interior regions, rather than being explicitly specified in the description of the system. This idea is inspired by the von Neumann's concept of "program is data" and also related to the research direction proposed by Gh. Păun about the cell nucleus.

We present yet another, relatively powerful, extension for the framework of P systems, which allows the system to dynamically change the set of rules, not limited to some finite prescribed set of candidates. There are three motives for this extension. First, our experience shows that "practical" problems need "more" computing potential than just computational completeness. Second, we attempt to import a very important computational ingredients into P systems, this time from the conventional computer science. Third, this extension correlates with the biological idea that different actions are carried out by different objects, which too can be acted upon. (This last idea was also considered in, e.g., [167] and [2], but there one represented each rule by a single object, therefore all rules were still prescribed, though not their multiplicities.) Let us first explain these motives.

Most papers of the field belong to the following categories: 1) introducing different models and variants, 2) studying the computational power of different models depending on what ingredients are allowed and on the descriptional complexity parameters, 3) studying the computational efficiency of solving intractable problems (supercomputing potential) depending

on the ingredients, 4) using membrane computing to represent and model various processes and phenomena, including but not limited to biology, 5) other applications.

There is a surprisingly big gap between the sets of ingredients needed to fulfill requirements in directions 2, 3, and the sets of ingredients demanded by other applications. For instance, very weak forms of cooperation between objects are often enough for the computational completeness, but many "practical" problems cannot be solved in a satisfactory way under the same limitations. This leads to the following question.

What is implicitly required in most "practical" problems? We will mention just a few of these requirements below.

- **Determinism** or at least confluence. Clearly, the end user wants to obtain the answer to the specified problem in a single run of a system instead of examining infinitely many computations. This is a strong constraint, e.g., catalytic P systems and P systems with minimal symport/antiport are universal, while in the deterministic case non-universality is published for the first ones and claimed for the latter ones. Informally speaking, less computational power is needed to just compute the result than it is to also enforce choice-free behavior of the system.

- **Input/output**. Most of the universality results are formulated as generating languages or accepting sets of vectors, or in an even more restricted setup. There is no need to deal with input in the first case, and in the latter case the final configuration itself is irrelevant (except yes or no in case of the efficiency research). On the other side, both input and output are critical for most applications.

- **Representation**. Clearly, any kind of discrete information can be encoded in a single integer in some consistent way. However, a much more transparent data representation is typically required; even the intermediate configurations in a computation are expected to reflect a state of the object in the problem area.

- **Efficiency**. Suppose numbers are represented by multiplicities of certain objects. The number of steps needed to multiply two numbers by plain (cooperative) multiset processing is proportional to the result. If the multiset processing can be controlled by promoters/inhibitors/priorities, then the number of steps needed for multiplication is proportional to one of the arguments. However, many applications would ask for a multiplication to be performed in a constant number of steps. Similar problems appear for string processing.

- **Data structures**. Membrane computing deals with multisets distributed over a graph, while conventional computers provide random memory access and pointer operations, allowing much more complex structures to be built.

Some of these implicit requirements originate because the user wants a solution which is at least as good as the one that can be provided by conventional computers. We hope that the explanations of the above list have convinced the reader that this is often a challenge.

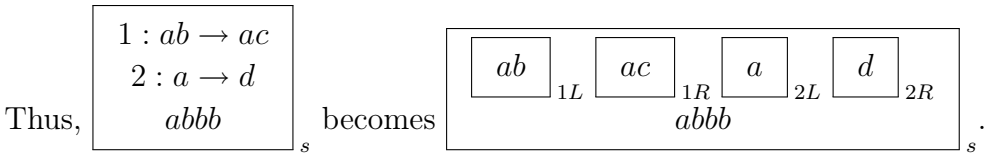


Figure 6.1: Graphical representation of a polymorphic P system

Program is data. Cell nucleus This time the new feature considered within the P systems framework is not of a biological inspiration, but rather is from the area of conventional computing. Suppose we want to be able to manipulate the rules of the system during its computation. A number of papers has been written in this direction (see, e.g., GP systems [167], rule creation [140], activators [2], inhibiting/deinhibiting rules [151] and symport/antiport of rules [150]), but in most of them the rules are predefined in the description of the system.

The most natural way to manipulate the rules is to represent them as data, treat this data as rules, and manipulate it as usual in P systems, in the spirit of von Neumann's approach. In membrane systems, the data consists of multisets, so objects should be treated as description of the rules. Informally, a rule j in a region i can be represented by the contents of membranes jL and jR inside i.

Changing the contents of regions jL and jR results in the corresponding change of the rule j. The next section illustrates this effect in Figure 6.2 and gives the formal definitions. We call such P systems polymorphic, by analogy with polymorphic, or self-modifying computer programs.

At the same time, if a membrane system is an abstraction inspired by the biological cell, one can view inner regions as an abstraction inspired by the cell nucleus; their contents correspond to the genes encoding the enzymes performing the reactions of the system. The simplicity of the proposed model is that we consider the natural encoding, i.e., no encoding at all: the multisets describing the rules are represented by exactly themselves. Therefore, we are addressing a problem informally stated by Gh. Păun in Section "Where Is the Nucleus?" of [248] by proposing a computational variant based on one simple difference: the rules are taken from the current configuration rather than from the description of the P system itself.

The idea of a nucleus was also considered in [274], but such a presentation had the following drawbacks. First, one described the dynamics of the rules in a high-level programming language (good for simulators, but otherwise too powerful extension having the power of conventional computers directly built into the definition). Second, this dynamics of the rules did not depend on the actual configuration of the membrane system (no direct feedback from objects to rules). In the model presented in this section, the dynamics of rules is defined by exactly the same mechanism as the standard dynamics of objects.

We illustrate the definitions by the following example.

Example 6.1 *A P system with a superexponential growth.*

$$\begin{aligned} \Pi_1 &= (\{a\}, \{a\}, \mu, a, a, a, a, a, a, aa, \varphi, 1), \text{ where} \\ \mu &= [\,[\,]_{1L}[\,[\,]_{2L}[\,[\,]_{3L}[\,]_{3R}\,]_{2R}\,]_{1R}\,]_s, \\ \varphi(i) &= here,\ 1 \le i \le 3. \end{aligned}$$

If the number of objects a in regions $3R$, $2R$ $1R$, s at step n is (x_n, y_n, z_n, t_n), respectively,

then $(x_0, y_0, z_0, t_0) = (2, 1, 1, 1)$ and $(x_{n+1}, y_{n+1}, z_{n+1}, t_{n+1}) = (x_n, y_n x_n, z_n y_n, t_n z_n)$. Following just this quadruple, the computation can be represented as $(2, 1, 1, 1) \Rightarrow (2, 2, 1, 1) \Rightarrow (2, 4, 2, 1) \Rightarrow (2, 8, 8, 2) \Rightarrow (2, 16, 64, 16) \Rightarrow (2, 32, 1024, 1024) \Rightarrow (2, 64, 32768, 1048576) \Rightarrow \cdots$.

The exponents of the closed form formula $(2, 2^n, 2^{n(n-1)/2}, 2^{n(n-1)(n-2)/6})$ can be verified as follows. $n+1 = n+1$, $(n+1)n/2 = n(n-1)/2 + n$, $(n+1)n(n-1)/6 = n(n-1)(n-2)/6 + n(n-1)/2$.

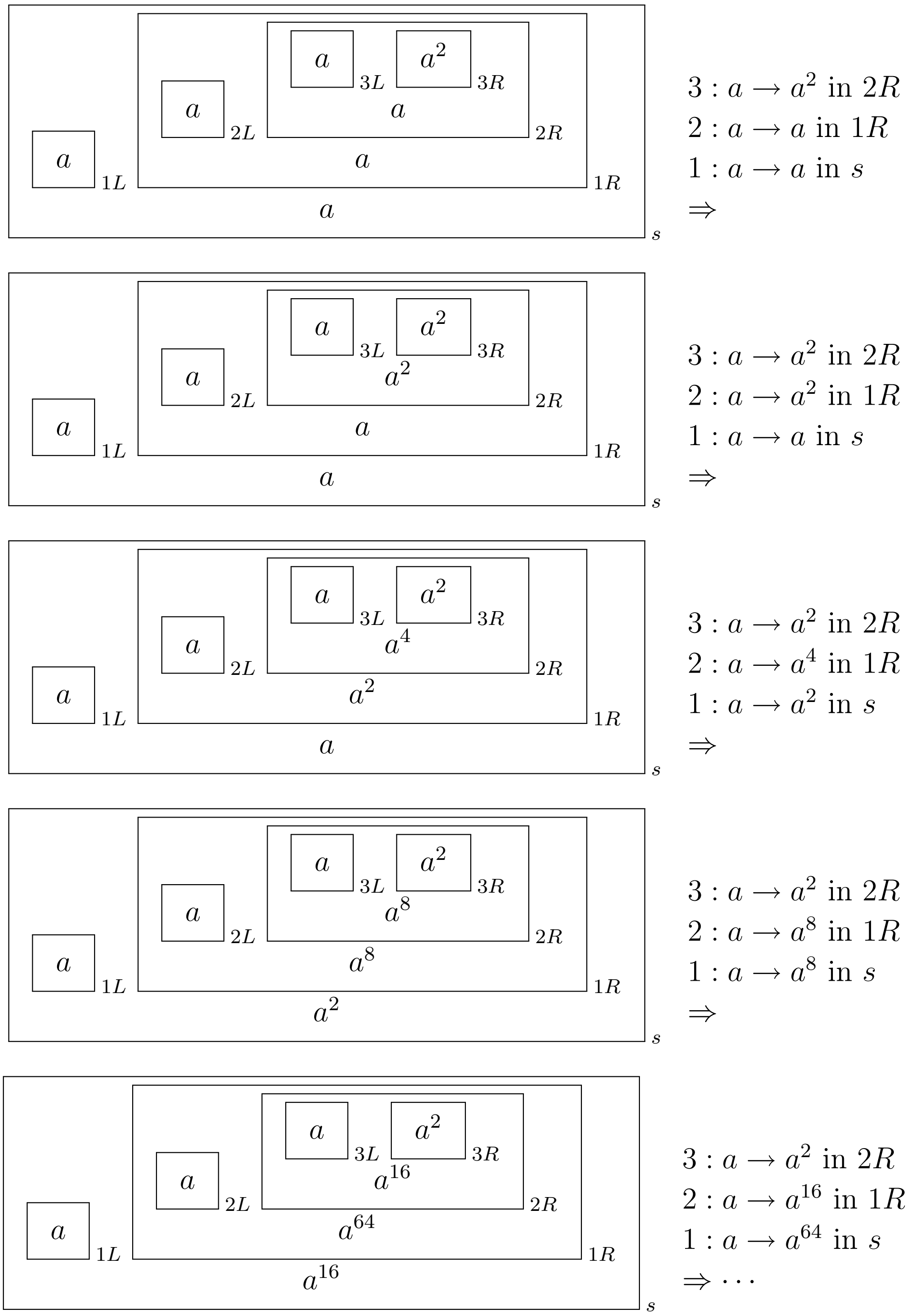


Figure 6.2: The polymorphic computation from Example 6.1

Naturally, contents of membranes $1L$, $2L$, $3L$ is never changed because they are elementary and no rules have the corresponding target indications, and their initial contents is a,

so the system is non-cooperative, and the rules are never disabled. Since only one rule acts in each of the regions s, $1R$, $2R$, the system is deterministic. From all above we conclude that $\Pi_1 \in DOP_7(polym_{-d}(ncoo))$, a quite restricted class.

This system never halts. Its interesting aspect, however, is the growth of the number of objects in the skin. We claim that at step n the skin contains $2^{n(n-1)(n-2)/6}$ objects, so the growth function is an exponential of a polynomial. Indeed, this is not difficult to see by starting from the elementary membranes and going outside.

The contents of $3R$ is aa and it never changes. Region $2R$ initially contains a and undergoes rule $a \to aa$ every step, so its contents at step n is a^{2^n}. Region $1R$ initially contains a and undergoes rule $a \to a^{2^n}$ at step n, so its contents at step n is $a^{2^{n(n-1)/2}}$. The skin originally contains a and at step n rule $a \to a^{2^{n(n-1)/2}}$ is applied, so its contents at step n is $a^{2^{n(n-1)(n-2)/6}}$, see Figure 6.2 for the actual illustration of the computation and for the proof of the result.

This growth is faster than that of any non-polymorphic P systems, which is bounded by the exponential Ic^n, where I is the initial number of objects in the system and c is the maximum ratio for all rules of the right side size and its left side size. It is not difficult to see that the growth function of a polymorphic P system without target indications is bounded by $Ic^p(n)$, where I and c are defined as above and p is a polynomial whose degree equals the depth of the membrane structure minus one.

6.4.1 The power of polymorphism

As long as full cooperation is allowed, the universality of polymorphic P system is not difficult to obtain, even without the actual polymorphism (i.e. without ever modifying rules) and without the use of target indications. The upper bound on the number of membranes needed is one plus twice the number of rules, because in the polymorphic P systems the rules can only be represented by pairs of membranes. We recall that in [136] one presents a strongly universal P system with 23 rules. Hence, the following theorem holds.

Theorem 6.1 *$NOP_{47}(polym_{-d}(coo)) = NRE$.*

Proof. The claim is fulfilled by taking the one-membrane construction from the main result in [136] and replacing each of the 23 rules by two membranes containing the left-hand side and the right-hand side of that rule. □

In the rest of the section we focus on the efficiency of computations performed by polymorphic P systems, using the time complexity terms. We devote special attention to fast generating and deciding factorials, because they best illustrate constant-time multiplication where the factors are not known in advance and are even changing during the computation. First, we present a non-cooperative system generating "slightly" more than factorials, using target indications. It is a bit more complicated than Π_1 because, firstly, we need to multiply by numbers that grow linearly, and secondly, we want the system to halt.

Example 6.2 *A polymorphic P system from $OP_{13}(polym_{-d}(ncoo), tar)$ which generates $\{n! \cdot$*

$n^k \mid n \geq 1, k \geq 0\}$.

$$\begin{aligned}
\Pi_2 &= (\{a,b,c,d\}, \{a\}, \mu, ab, a, a, a, a, a, c, b, bd, b, \lambda, d, a, \varphi, 1), \ \textit{where} \\
\mu &= [\,[\,[\,]_{2L}[\,]_{2R}[\,]_{3L}[\,]_{3R}\,]_{1L}[\,]_{1R}[\,]_{4L}[\,]_{4R}[\,]_{5L}[\,]_{5R}[\,]_{6L}[\,]_{6R}\,]_s, \\
\varphi(i) &= \textit{here}, \ 1 \leq i \leq 5, \ \varphi(6) = in_{1R}.
\end{aligned}$$

The initial configuration can be graphically represented as shown in Figure 6.3. In fact, such a graphical representation gives a complete description of Π_2 except the output alphabet and the output region. The target indication of a rule (here rule 6 in $1R$) may be indicated by an arrow, in this case from $6R$ to $1R$ (keeping in mind that the reactants of the rule are taken from the parent region of the membranes describing the rule, in this case, from region 1). At the right we give a simplified representation of the same system by replacing pairs of membranes with constant contents by the rules written explicitly (this is just a different representation, so-called "syntactic sugar", and we still count such rules as pairs of membranes). Rule 1 is not written with the rule syntax because the contents of both $1L$ and $1R$ will change.

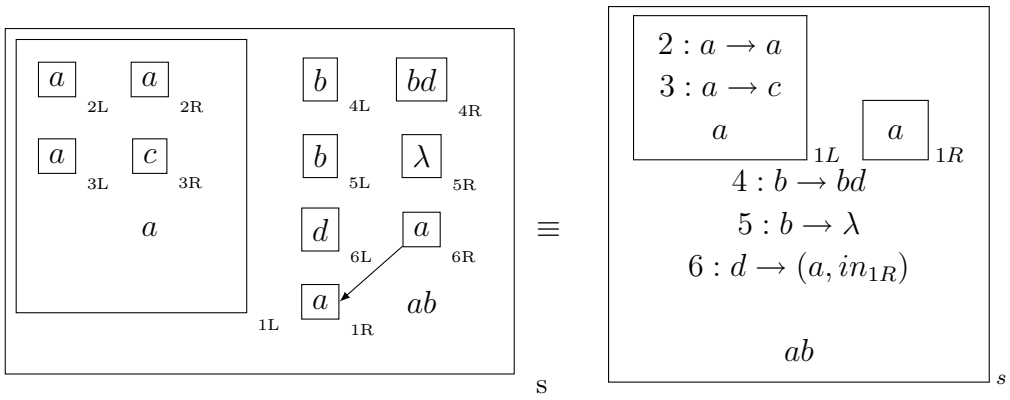


Figure 6.3: A non-cooperative polymorphic system giving a complicated result

The essence of the functioning of Π_2 is the following. Rules 4 and 6 lead to incrementation of the number of copies of a in $1R$ (the number of copies of a in the skin does not change during the first two steps). The system will apply rule 4 for $n - 1 \geq 0$ times and then rule 5 (applying rule 5 is necessary for the system to halt). Suppose that all this time rule 2 has been applied in region $1L$. Then, the number of objects in region $1R$ will grow linearly, and subsequent applications of a dynamic rule $1 : a \to a^i$, $1 \leq i \leq n$ will produce $a^{n!}$ in the skin. After that, the number of objects a in the skin will be multiplied by n until rule 3 is applied, because $1 : c \to a^n$ will be no longer applicable, halting with the skin only containing objects a their number being an arbitrary number of the form $n! \cdot n^k$. Now assume that rule 3 has been applied earlier, effectively stopping the multiplication of the number of objects a in the skin before the incrementation of objects a in $1R$ is finished. In that case the multiplicity of objects a in the skin will be just a factorial of a smaller number, and the system will evolve by application of rules 4, 6 until rule 5 is applied, without affecting the result. Notice that the time complexity (understood as the shortest computation producing the corresponding result) of generating $n! \cdot n^k$ is only $n + k + 1$.

To generate exactly $\{n! \mid n \geq 1\}$ we need to stop the multiplication when we stop the increment. This seems impossible without cooperative rules.

Example 6.3 *A P system from* $OP_9(polym_{-d}(coo), tar)$ *generating* $\{n! \mid n \geq 1\}$.

$$\begin{aligned} \Pi_3 &= (\{a,b,c,d\}, \{a\}, \mu, ab, a, a, b, bd, b, c, d, a, \varphi, 1), \text{ where} \\ \mu &= [\,[\,[\,]_{1L}[\,]_{1R}[\,]_{2L}[\,]_{2R}[\,]_{3L}[\,]_{3R}[\,]_{4L}[\,]_{4R}\,]_s, \\ \varphi(i) &= here,\ 1 \leq i \leq 2,\ \varphi(3) = in_{1L},\ \varphi(4) = in_{1R}. \end{aligned}$$

This system is very similar to Π_2. There are only the following differences. First, rules $a \to a$ and $a \to c$ are removed from region $1L$. Second, instead of erasing b in the skin, the corresponding rule sends object c to region $1L$, which stops both increment (b is erased) and multiplication ($1 : ac \to a^n$ is not applicable in the skin). Ironically, this system never applies any non-cooperative rule, but the non-cooperative feature seems unavoidable in order to stop the computation in the synchronized way. A compact graphical representation of Π_3 is given in Figure 6.4.

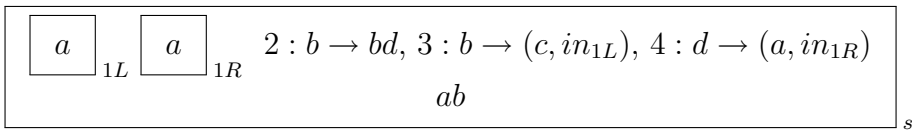


Figure 6.4: Illustration of a polymorphic system generating factorials

Now we proceed to describing a P system generating $\{2^{2^n} \mid n \geq 0\}$ in $O(n)$ steps. Since the growth of polymorphic P systems without target indications is bounded by exponential of polynomials, the system below grows faster than any of them. Moreover, it produces the above mentioned result by halting.

It is also worth noting that even polymorphic P systems cannot grow faster than exponential of exponential in linear time, because if a system has $n + n + 1 > 3$ objects at some step, then it cannot have more than $n^2 + n + 1$ objects in the next step. Indeed, consider that some rule r is applied for n times; let its left side contain x objects and let its right side contain y objects. Then, $x+y$ objects are needed to describe the rule and they transform nx other objects into ny objects. It is not difficult to see that the growth is maximal if $x = 1$ and $y = n$. Since $n^2 + n + 1$ is less than the square of $n + n + 1$, and iterated squaring yields the growth which is exponential of exponential, it is not possible to grow faster. The system below grows three times slower than this bound.

Example 6.4 *A P system from* $OP_{15}(polym_{-d}(ncoo), tar)$ *generating numbers from* $\{2^{2^n} \mid n \geq 0\}$ *in* $3n + 2$ *steps.*

$$\begin{aligned} \Pi_4 &= (\{a,b,a',b',c\}, \{a\}, \mu, b^2, a, \lambda, a, a, a, c, b, \lambda, b, a'b', a', a, b', b, \varphi, 1), \\ \mu &= [\,[\,[\,]_{2L}[\,]_{2R}[\,]_{3L}[\,]_{3R}\,]_{1L}[\,[\,]_{4L}[\,]_{4R}\,]_{1R} \prod_{i=5}^{7} ([\,]_{iL}[\,]_{iR})\,]_s, \\ &\quad \varphi(i) = here,\ 1 \leq i \leq 6,\ \varphi(7) = in_{1R}. \end{aligned}$$

The desired effect is obtained by iterated squaring. By rules 5, 6, 7, in two steps each copy of b in the skin changes into a and also sends a copy of b in region $1R$. In the next step, if region $1L$ still contains an a, each copy of a in the skin is replaced by the contents of region $1R$, and the process continues. Therefore, if we had b^k in the skin at some step, then in two steps we will have a^k in the skin and rule 1 will be of the form $a \to b^k$, yielding b^{k^2} in the third step. The iteration continues while rule 2 is being applied in region $1L$. When rule 3 is applied, the cycle stops because rule $1 : c \to b^k$ will not be applicable, and the result is given as the multiplicity of objects a in the skin. Clearly, $2 = 2^{2^0}$ and $2^{2^{n+1}} = (2^{2^n})^2$, so the systems generates 2^nth powers of 2. We underline that no cooperation was used in this case. A compact graphical representation of this system is shown in Figure 6.5.

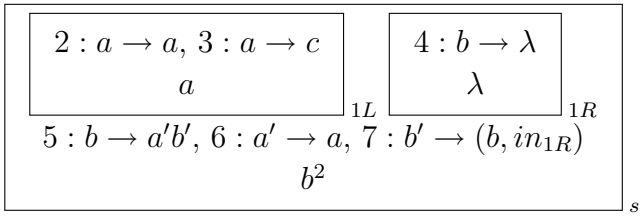


Figure 6.5: A polymorphic system generating 2^{2^n}

We remind the reader that the picture above represents a system with 15 membranes because the rules notation is simply a compact way to represent pairs of membranes. Note that one rule could have been saved if the right side of the rule were allowed to have objects with different target indications, but this issue does not affect the computational power, only the number of rules, whereas the definitions are much simpler. Another rule could be saved at the price of using a cooperative rule to stop the computation instead of rules 2 and 3, like in the previous example.

We now proceed to tasks which are more difficult than generating, namely, deciding a set of numbers or computing a function in a deterministic way. We illustrate the first case by modifying the previous example. We use an additional ingredient compared to the previous systems: we rely on disabling a rule by emptying the region describing its left side. Although we expect that this ingredient does not change the computational power of the systems, we use it in order to have smaller constructions.

Example 6.5 *A deterministic P system from $OP_{15}(polym_{+d}(coo), tar)$ computing the function $n \longrightarrow 2^{2^n}$ in $3n + 2$ steps.*

$$\begin{aligned}
\Pi_5 &= (\{a, b, a', b', c, d, d'\}, \{d\}, \{a\}, \mu, cb^2, \lambda, \lambda, a, \lambda, b, \lambda, \\
&\quad b, a'b', a', a, cd, c'd', c', c'', c'', c, d', a, b', b, \varphi, 1, 1), \ \textit{where} \\
\mu &= [\,[\,[\,[\,\,]_{2L}[\,\,]_{2R}\,]_{1L}[\,[\,\,]_{3L}[\,\,]_{3R}\,]_{1R} \prod_{i=4}^{10} ([\,\,]_{iL}[\,\,]_{iR})\,]_s, \\
&\quad \varphi(i) = here, \ 1 \leq i \leq 8, \ \varphi(9) = in_{1L}, \ \varphi(10) = in_{1R}.
\end{aligned}$$

This system works like Π_4 from the previous example. We only focus on the differences. The previous system used non-deterministic choice between rules 2 and 3 to continue the computation or to stop it. In this case, squaring stops by itself due to the rule $2 : a \to \lambda$,

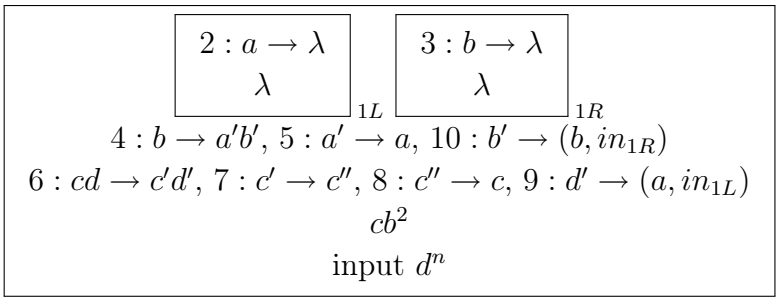


Figure 6.6: A polymorphic system computing a superpower function

so producing object a in region $1L$ activates one squaring. The most important difference is that the number n is given as input into the skin, by the multiplicity of objects d. Moreover, besides two copies of b the skin initially contains an object c, responsible for counting until n by consuming objects d and activating the squaring routine the corresponding number of times. The cycle takes 3 steps, see rules 6, 7, 8, 9. When object c has no more copies of d to consume, the result is obtained as the multiplicity of objects a in the skin. We show a compact graphical representation of Π_5 in Figure 6.6.

Note that this system uses cooperation for counting and disabling the rules for easier control. We leave it as an exercise for the reader to construct a P system Π'_5 computing the same function without disabling rules. Hint: as long as objects a only appear in the skin every third step, there is no need to disable rule 1 while the computation is in progress. Object c can deterministically subtract d and perform its appearance checking. Finally, when there are no copies of d in the skin, moving c into $1L$ will make rule 1 inapplicable without the need to disable it by emptying its left side.

Now we give an example of a P system deciding a set of numbers. It works deterministically and produces an object `yes` or `no` in the skin, depending on whether the input number belongs to the specified set. We also emphasize its time complexity.

Example 6.6 *A deterministic P system from $OP_{37}(polym_{-d}(coo), tar)$ deciding the set $\{n! \mid n \geq 1\}$. A number $k \leq n!$ is decided in at most $4n$ steps, i.e., in a sublogarithmic time with respect to k.*

$$\begin{aligned}
\Pi_6 \;=\;& (\{a, b, c_0, c_1, c_2, A, A', B, B', p_0, p_1, p_2, p_3, \texttt{yes}, \texttt{no}\}, \{a\}, \{\texttt{yes}, \texttt{no}\}, \mu,\\
& p_0c_0, a^2, b, b, a, c_1, c_2, c_2, \lambda, p_0, AABp_1, Aa, A'a, Bb, B'b, p_1, p_2,\\
& p_2B'AA, p_3d, p_2B'A'A, f\texttt{no}, p_2B'A'A', f\texttt{no}, p_2BAA, f\texttt{no},\\
& p_2BA'A, f\texttt{yes}, p_2BA'A', f\texttt{no}, p_3, p_0c_0, c_0, c_1, d, a, f, f, \varphi, 1, 1), \;\; \textit{where}\\
\mu \;=\;& [\,[\;]_{1L}[\;[\;]_{3L}[\;]_{3R}[\;]_{4L}[\;]_{4R}]_{2L}[\;]_{2R} \prod_{i=5}^{18} ([\;]_{iL}[\;]_{iR}) \;]_s,\\
& \varphi(i) = here, \;\; 1 \leq i \leq 15, \;\; \varphi(16) = in_{2L}, \;\; \varphi(17) = \varphi(18) = in_{1L}.
\end{aligned}$$

The work of Π_6 consists of iterated division of the input a^k. Each cycle consists of 4 steps. The role of object c_0 is to enter into $2L$ by rule 16, thus preventing rule $2 : b \to a$ to work during the second and the third step of the cycle ($bc_1 \to a$ is not applicable, changing by rule 3 to $bc_2 \to a$, which is also not applicable, and then being restored by rule 4).

Object p_0 marks the steps and produces the necessary objects for checking some numbers, and finally produces symbols to increment the divisor or to modify the dividing rule to stop the computation, and give the answer, as follows. Suppose that the input is a^k. In the first step, p_0 changes into p_1, also producing checkers AAB. In the same time, the number k will be divided by n (initially $n = 2$) by rule $1 : a^n \to b$, changing a^k into $b^x a^y$, where x is the quotient and y is the remainder.

In the second step, p_1 changes into p_2, waiting for the checkers. The role of the checking rules $6 : Aa \to A'a$ and $7 : Bb \to B'b$ is to test the multiplicity of the remainder and the quotient, respectively. Hence, object B will be primed if $x > 0$. Notice that since there are two copies of A in the system, the number of symbols A that will be primed is $\min(y, 2)$. Thus, there are 6 combinations of symbols A and B, primed or not.

In the third step, we distinguish two special cases. If $x > 0$ and $y = 0$, then the input is a multiple of the currently computed factorial, and we proceed to the next iteration by rule $9 : p_2 B'AA \to p_3 d$. If $x = 0$ and $y = 1$, then the input is equal to the previously computed factorial, and the system gives the positive answer by the rule $13 : p_2 BA'A \to f\texttt{yes}$. Four other combinations correspond to detecting that the input is not equal to a factorial of any number (two cases correspond to non-zero quotient and non-zero remainder, the third case corresponds to the input being zero, and the last case corresponds to a multiple of some factorial which is smaller than the next factorial), so $f\texttt{no}$ is produced.

In the fourth step, rule $2 : b \to a$ is used, so the quotient is ready to be divided again. Object f is used to stop the computation by rule 18, since rule $1 : a^n f \to b$ is not applicable. In case we proceed to the next iteration, the role of object d is to increment the multiplicity n of objects a in region $1L$, and object p_3 changes back to p_0 and produces a new copy of c_0 for the next cycle.

Figure 6.7 shows a compact graphical representation of Π_6.

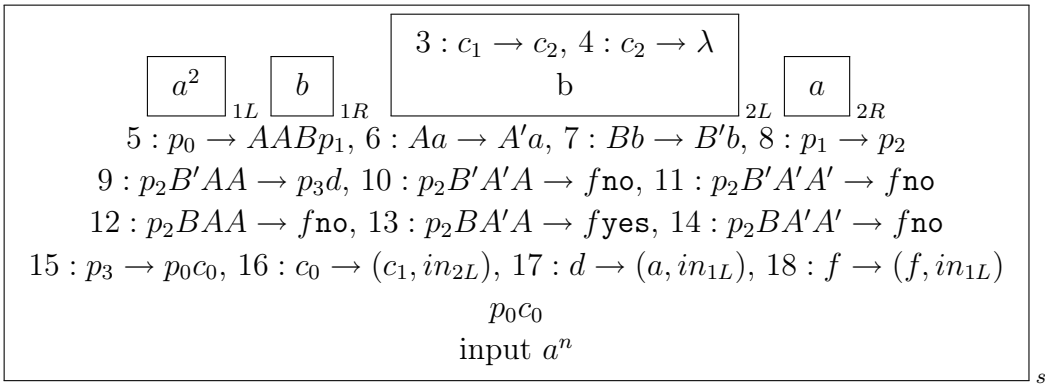


Figure 6.7: A polymorphic P system deciding factorials

We summarize some of the results we obtained as follows.

Theorem 6.2 *There exist*

- *A strongly universal P system from* $OP_{47}(polym_{-d}(coo))$*;*
- *A P system* $\Pi_1 \in DOP_7(polym_{-d}(ncoo))$ *with a superexponential growth;*

- *A P system* $\Pi_2 \in OP_{13}(polym_{-d}(ncoo), tar)$ *such that* $N(\Pi_2) = \{n! \cdot n^k \mid n \geq 1,\ k \geq 0\}$ *and the time complexity of generating* $n! \cdot n^k$ *is* $n + k + 1$*;*

- *A P system* $\Pi_3 \in OP_9(polym_{-d}(coo), tar)$ *such that* $N(\Pi_3) = \{n! \mid n \geq 1\}$ *and the time complexity of generating* $n!$ *is* $n + 1$*;*

- *A P system* $\Pi_4 \in OP_{15}(polym_{-d}(ncoo), tar)$ *such that* $N(\Pi_4) = \{2^{2^n} \mid n \geq 0\}$ *and the time complexity of generating* 2^{2^n} *is* $3n + 2$*;*

- *A P system* $\Pi'_5 \in DOP_*(polym_{-d}(coo), tar)$ *such that* $f(\Pi_5) = (n \longrightarrow 2^{2^n})$ *and the time complexity of computing* $n \longrightarrow 2^{2^n}$ *is* $O(n)$*;*

- *A P system* $\Pi_6 \in DOP_*(polym_{-d}(coo), tar)$ *such that* $N_d(\Pi_6) = \{n! \mid n \geq 1\}$ *and the complexity of deciding any number* k*,* $k \leq n!$ *does not exceed* $4n$*.*

Moreover, polymorphic P systems can grow faster than any non-polymorphic P systems, whereas even non-cooperative polymorphic P systems with targets can grow faster than any polymorphic P systems without targets.

Discussion We proposed a variant of the rewriting model of P systems where the rules are represented by objects of the system itself and thus can dynamically change. This yields a mechanism whose idea is similar to the idea of the functioning of the cell nucleus (i.e., DNA represent the proteins performing certain functions on the objects including DNA), except our formalism is more elegant mathematically because of its simplicity and because we only used a trivial encoding (which is no encoding at all, except the left and right parts of the rule are given in dedicated membranes).

This variant also has a number of connections to the conventional computing, since the "program" can be changed by manipulating data (cf. von Neumann architecture vs Harvard architecture). A number of possible extensions is suggested in the Definition section of the section.

Polymorphic P systems are universal (with 47 membranes) because non-polymorphic P systems are universal. While the growth of non-polymorphic P systems is bounded by exponential, polymorphic P systems without target indications can grow faster, bounded by an exponential of polynomials, and polymorphic P systems with target indications can grow even faster, bounded by an exponential of exponentials.

Non-cooperative polymorphic P systems can generate non-context-free sets of numbers. Cooperative polymorphic P systems can multiply numbers in constant time and generate factorials of n or exponentials of exponentials of n in time $O(n)$, which is a very important advantage over non-polymorphic P systems.

An especially interesting case is that of deciding if the input belongs to a given set, e.g., $\{n! \mid n \geq 1\}$. While non-polymorphic P systems cannot even grow with factorial speed, not to speak about halting or verifying the input, we have shown that polymorphic P systems can decide factorials in time $O(n)$. This implies that there exist infinite sets of numbers that are accepted in a time which is sublinear with respect to the size of the input in binary representation (without cheating by only examining a part of the input to accept).

Many questions are left open, we mention three questions here. First, we find it particularly interesting what is the exact characterization of the most restricted classes we defined, like $OP_*(polym_{-d}(ncoo))$. On the other hand, it seems interesting how the (general classes of) polymorphic P systems can solve the problems of real applications which non-polymorphic P system are not suitable for. Another question is whether the polymorphic P systems can effectively use superexponential growth and dynamics of rule description to solve intractable problems in polynomial time without dividing or creating membranes. Conjecture: no, because the total number of rules (counting rules in different regions as different) cannot grow.

6.5 Other Applications

The author has published a number of other works on application of P systems and other formal computing models. We list some of them here, without describing them in detail.

In [201], [202] one considers performing logical inference by membrane systems, called there "chaining".

In [33] one considers encoding numbers by multisets and discusses the questions of compactness of the representation and of the efficiency of performing the arithmetic operations in different multiset representations.

In [36], [37] one considers right self-assembly of a double-stranded molecule from a single-stranded words of bounded length. If this subregular generative mechanism is equipped with the idea of computing and observing, we have a computationally complete device.

In [200] one considers cellular automata that possess a specific invariant, a so-called *number-conservativity*. It is established that even with the minimal (called radius-1/2, and sometimes referred to as one-way) neighborhood (the focus cell and one neighbor), number conservative cellular automata are computationally complete. It is established that 1057 states are sufficient.

Reversible logical elements with memory In [239], [238], [240], [241] and [242] one considers different reversible computing models, in particular reversible logic elements with (1-bit) memory of degree k (written k-RLEMs). These are primitives with k inputs, k outputs and 2 states, defined by a bijection from input-state pairs onto state-output pairs.

It was well-known that a specific 4-RLEM called *rotary element*, denoted RE is computationally universal, in the sense that any computation device, e.g., a reversible Turing machine, can be constructed from REs, connected by wires. On the other hand, RE has an implementation in the so-called billiard ball model.

An RLEM is called non-degenerate if it is not equivalent to RLEM(s) of smaller degrees and/or wires. It is known that there must be some input that changes the state, there must be some input leading to different outputs depending on the state, and no input can lead to the same output in both states without changing the state. Moreover, these three requirements are sufficient for an RLEM to be non-degenerate.

It has been shown (see the references in the beginning of the subsection) that any non-degenerate k-RLEM is computationally universal if $k \geq 3$. In particular, all k-RLEMs are

thoroughly studied for $2 \leq k \leq 4$, and necessary simulations of some elements are given as circuits of other ones.

6.6 Conclusions to Chapter 6

Membrane systems were presented describing the inflectional process.

The linear-time algorithms were given for searching in a dictionary and updating it implemented as membrane systems. We underline that the systems are constructed as reusable modules, so they are suitable for using as sub-algorithms for solving more complicated problems.

The scope of handling dictionaries is not limited to the dictionaries in the classical sense. Understanding the definition of a dictionary, i.e., a string-valued function defined on a finite set of strings, leads to direct applicability of the proposed methods to handle alphabets, lexicons, thesauruses, dictionaries of exceptions, and even databases. At last, it is natural to consider these algorithms together with morphological analyzer and morphological synthesizer.

Two algorithms were outlined that synchronize two given classes of P systems. The first one is non-deterministic and it synchronizes the class of transitional P systems (with cooperative rules) in time $2h + 3$, where h is the depth of the membrane tree. The second algorithm is deterministic and it synchronizes the class of P systems with promoters and inhibitors in time $3h + 3$.

The results presented here rely on a rather strong target indication, $in!$, which sends an object to all inner membranes. Such a synchronization was already considered in neural-like P systems where it corresponds to the target go. It would be interesting to investigate what happens if such target is not permitted. However, we conjecture that a synchronization would be impossible in this case.

The study of the synchronization algorithms for different classes of P systems is important as it permits to implement different synchronization strategies which are important for such a parallel device as P systems. In particular, with such approach it is possible to simulate P systems with multiple global clocks by P systems with one global clock. It is particularly interesting to investigate the synchronization problem for P systems which cannot create new objects, for example for P systems with symport/antiport.

A variant of the rewriting model of P systems was proposed where the rules are represented by objects of the system itself and thus can dynamically change. This yields a mechanism whose idea is similar to the idea of the functioning of the cell nucleus (i.e., DNA represent the proteins performing certain functions on the objects including DNA), except our formalism is more elegant mathematically because of its simplicity and because we only used a trivial encoding (which is no encoding at all, except the left and right parts of the rule are given in dedicated membranes).

This variant also has a number of connections to the conventional computing, since the "program" can be changed by manipulating data (cf. von Neumann architecture vs Harvard architecture). A number of possible extensions is suggested in the Definition section of the section.

Polymorphic P systems are universal (with 47 membranes) because non-polymorphic P systems are universal. While the growth of non-polymorphic P systems is bounded by exponential, polymorphic P systems without target indications can grow faster, bounded by an exponential of polynomials, and polymorphic P systems with target indications can grow even faster, bounded by an exponential of exponentials.

Non-cooperative polymorphic P systems can generate non-context-free sets of numbers. Cooperative polymorphic P systems can multiply numbers in constant time and generate factorials of n or exponentials of exponentials of n in time $O(n)$, which is a very important advantage over non-polymorphic P systems.

An especially interesting case is that of deciding if the input belongs to a given set, e.g., $\{n! \mid n \geq 1\}$. While non-polymorphic P systems cannot even grow with factorial speed, not to speak about halting or verifying the input, we have shown that polymorphic P systems can decide factorials in time $O(n)$. This implies that there exist infinite sets of numbers that are accepted in a time which is sublinear with respect to the size of the input in binary representation (without cheating by only examining a part of the input to accept).

Many questions are left open, we mention three questions here. First, we find it particularly interesting what is the exact characterization of the most restricted classes we defined, like $OP_*(polym_{-d}(ncoo))$. On the other hand, it seems interesting how the (general classes of) polymorphic P systems can solve the problems of real applications which non-polymorphic P system are not suitable for. Another question is whether the polymorphic P systems can effectively use superexponential growth and dynamics of rule description to solve intractable problems in polynomial time without dividing or creating membranes. Conjecture: no, because the total number of rules (counting rules in different regions as different) cannot grow.

Other applicative directions are mentioned, such as chaining, encoding numbers by multisets for their processing with membrane systems, right self-assembly of a double-stranded molecule from a single-stranded words of bounded length, cellular automata that are number-conservative, and reversible logical elements with memory.

Section 6.1 is based on publications [31], [32] and [48], and mentions publication [48]. Section 6.2 is based on works [49] and [50]. Section 6.3 is based on publications [97] and [98]. Section 6.4 is based on publications [81] and [82]. Section 6.5 is based on works [201], [202], [33], [36], [37], [200], as well as the publications on reversible logical elements with memory [239], [238], [240], [241] and [242].

GENERAL CONCLUSIONS AND RECOMMENDATIONS

The present habilitation thesis contains an overview and the obtained results on small computational devices, mainly in distributed parallel processing of multisets and strings. In our world, with rapid development of technologies, we see that, despite their miniaturization, the computational devices become increasingly complex, to the unimaginable limits. In this perspective it strongly contrasts to the results here, showing that, at least in principle, a computational device does not really have to be complicated just to be able to *compute anything computable*. In the abstract framework, interpretation and execution of stored programs can be done by systems of very small descriptional complexity. Clearly, we are also interested in *efficient* computations. Hence, we also pay attention to the simple computing devices that attack intractable (for computers with bounded parallelism) problems by using parallelism and distributivity.

We have studied a number of models, variants, features, working modes, their restrictions, properties, computational power, efficiency, and certainly, their descriptional complexity parameters. The underlying research is classified as Theoretical computer science, but it also interconnects with multiple disciplines, like Algebra, Formal language theory, Computability, Computational complexity, Logic and Biocomputing, to name a few. The following results can be considered as the most important theoretical ones.

1. **Small number of rules**. In Section 3.1 we presented a universal P system with 23 antiport rules. Clearly, it may also be viewed as a universal maximally parallel system with 23 multiset rewriting rules. Multiple optimization techniques and strategies were used to achieve behavior of many rules with only a few. It is truly remarkable that such a small number of such simple rules makes it possible to interpret a stored program and yield a computationally complete behavior. The number of computational branches of the machine used as a starting point for the simulation was higher than that. Although the author has shown that 5 rules already suffice for universality in case of P systems with string-objects and splicing operations (this result is only mentioned in the section on Splicing, but its presentation is not included in the thesis), the symbol-object model is limited to a much simpler data structure.

2. **Determinism and reversibility**. In Section 2.3 we presented a study of deterministic and reversible P systems and the strong variants of these properties. The power (from sublinear to universal) of uncontrolled and of controlled systems satisfying these properties is summarized in Table 2.1. We also presented a syntactic criterion of strong reversibility. A few cases remain open, and some interesting conjectures were formulated. We mention other computational models where the author considered reversibility and determinism.

3. **Membrane creation** in transitional P systems, together with membrane dissolution,

makes it possible to obtain the computational complete systems! Note that no dynamic information is present on the membrane itself, besides the fact whether it exists or not. The only possible interaction between different objects is via creating and dissolving membranes. A completely untraditional approach has been used to obtain this result, see Section 2.5: the information of the simulated computation is stored in the multiplicities of membranes, rather than in the multiplicities of objects. We have also addressed the questions of generating languages, deterministic acceptance of sets of vectors, object and membrane complexity.

4. P systems with **active membranes without polarizations** are computationally complete! The idea is related to the previous most important result, except that the working mode of these P systems is different. It is possible here to create new membranes only by dividing the existing ones, so even zero has to be encoded by one, otherwise it cannot be incremented. Again, the information of the simulated computation is stored in the multiplicities of membranes, not objects, see Section 4.2. We have also addressed the questions of generating languages, object complexity, and membrane complexity. The technique used heavily relies on non-determinism, and we conjecture that such deterministic systems are not universal. Note that with two polarizations (storing 1 bit on a membrane), computational completeness can be obtained in a single membrane and deterministically.

5. **PSPACE**-complete problems can be solved by P systems with active membranes even without polarizations! We exploited timing as the way of interacting of the objects (dissolving a membrane earlier or later by one object, and checking this by another object), see Section 4.5. In contrast to the two previous most important results, we now use a limited number of such interactions, corresponding to the number of levels of membrane hierarchy. Note that with two polarizations (storing 1 bit on a membrane), one has solved problems in P^{PP} without non-elementary division.

6. We have improved the state of the art for the model of **hybrid networks of evolutionary processors**, see Section 5.1. Namely, HNEPs are universal with 7 nodes, while HNEPs with 1 node are subregular and have been characterized. Note that in the basic model of networks of evolutionary processors, the filters are much more powerful, and computational completeness has been obtained already with 2 nodes. We remark that NEPs with a complete graph demonstrate complicated (computationally universal) behavior emerging as a joint effect of very simple components, acting independently. We have also introduced a new variant of HNEPs: those with obligatory operations, its research being quite a perspective direction.

7. **Deterministic controlled non-cooperative** systems only accept finite sets and their complements, see Section 2.2. This implies that determinism establishes a frontier between the computational completeness and decidability for non-cooperative multiset rewriting with control. Note that this holds for the maximally parallel mode and for the asynchronous mode, while surprisingly the sequential mode turns out to be more powerful here, yielding computational completeness.

8. **Energy-based** P systems have been shown to be computationally complete in the maximally parallel mode, see Section 4.8. This model has a very weak form of interaction between objects: all they can do is to be renamed, releasing or consuming free energy of the region. Moreover, energy conservation law is a built-in feature of the model. We conjecture that deterministic systems are not universal, because by modifying free energy of the region

we can only toggle the behavior of an object between three possibilities, namely its evolution in the pre-defined way, waiting forever or waiting before resuming that evolution. Although the technique is quite different, the results are similar to those for P systems with unit rules and energy assigned to membranes.

9. **Insertion-deletion** P systems are computationally complete in the maximally parallel mode even with inserting and deleting one symbol without context (with priority of deletion over insertion), see Section 5.2. This is easy to see for graphs, but requires a more difficult technique for trees. Similar results have been established for insertion and deletion on the ends of the string.

Besides the theoretical results mentioned above, we have considered a number of **applications**. For instance, we presented new results on **synchronization** in P systems, see Section 6.3. This is a problem similar to Firing Squad Synchronization, except that the underlying structure is a tree (of height h, not known in advance), and the model is membrane systems, not cellular automata. Deterministic systems synchronize in $3h + 3$ steps.

Another application is polymorphic P systems. Its use is providing a framework where rules can dynamically change during the process of computation, which is important for problems of **symbolic computation** and computer algebra.

Other applications deal with **linguistics**. We proposed an efficient implementation of dictionaries by membrane systems, using membrane (tree) structure to represent the prefix tree of the dictionary. We discovered suitability of P systems for performing inflections of words in the Romanian language. We also proposed P systems annotating affixes of the Romanian language, using a model that accounts for complex derivation steps that may consist of multiple affixes, changing terminations and/or alternations in the root.

More detailed conclusions can be found at the end of Chapters 2, 3, 4, 5 and 6. Technical conclusions and more detailed open problems are given in the end of sections of these chapters. The potential new directions that complete and continue investigation of the corresponding formal models, can serve as the basis for further work. The present thesis can be also regarded as a didactic work. It provides a detailed coverage of multiset processing, transitional P systems, P systems with symport/antiport, P systems with string-objects and networks of evolutionary processors. Many results proved are optimal: either the minimization is already at the lowest possible parameter, or a characterization has been obtained.

Minimization problems lead to simplification of potential implementations performed by other researchers. Comparison and characterization of different models, variants and features lead to simplification of choice what to implement, depending on whether the corresponding task falls under the obtained characterization. Connection to different models advances the general theory of computation as information processing.

Recommendations The author recommends to continue research of parallel distributed processing of multisets and strings, in particular paying attention to the following problems that remained open:

1) What is the exact characterization of the family of languages generated by non-cooperative transitional P systems?

2) Verify the conjectures of non-universality in Table 2.1 on determinism and reversibility.

3) Answer unsolved cases in Table 2.2 on self-stabilization.

4) What is the power of P systems with membrane creation and one object ($PsMAT$ is conjectured)?

5) What is the power of symport systems with only one extra object?

6) What is the power of deterministic P systems with polarizationless membrane division?

7) Investigate P systems with active membranes where rules of type (a) have at most one object in their right side.

8) Can PSPACE-complete problems be solved by P systems with active membranes without non-elementary division?

9) What is the minimal number of membranes in efficient P systems with active membranes under minimal parallelism?

10) What is the power of deterministic energy-based P systems (conjecture: degenerate).

11) Characterizing restricted polymorphic P systems.

Direct applications and implementations The discussion below concerns investigations that are **NOT** part of the present thesis. However, the author has been overwhelmed with questions "why?" and "how?", so the corresponding explanations are due. Since most research presented here make part of membrane computing, it is the membrane computing that will be used as a representative member of unconventional computing area.

Applications of membrane systems include machine learning, modeling of biological processes (photosynthesis, certain signaling pathways, quorum sensing in bacteria, cell-mediated immunity), and computer science applications such as computer graphics, public-key cryptography, approximation and sorting algorithms, as well as analysis of various computationally hard problems.

Many variant models have been studied, and interest has focused on proving computational universality for systems with a small number of membranes, for the purpose of solving NP-complete problems such as Boolean satisfiability (SAT) problems and the traveling salesman problem (TSP). The P systems may trade space and time complexities and less often use models to explain natural processes in living cells. The studies devise models that may at least theoretically be implemented on hardware. To date, the P systems are nearly all theoretical models that have never been reduced to practice, although a practical system is given in the US Patent Application No.: 12/289,735.

Although inspired by biology, the primary research interest in P systems is concerned with their use as a computational model, rather than for biological modeling, although this is also being investigated. Most variants of membrane systems have been proved to be Turing-complete and computationally efficient, i.e., able to solve computationally hard problems in polynomial time. Although most research in P systems concentrates on computational powers, lately they have been used to model biological phenomena.

As P systems are inspired from the structure and functioning of the living cell, it is natural to consider them as modeling tools for biological systems, within the framework of Systems Biology, being an alternative and complementary to more classical approaches like Ordinary Differential Equations (ODEs), Petri nets and π-calculus.

Indirect applications. To give the reader just a few ideas about possible applications of membrane computing, we list the chapters of the book containing [132]. (I)Bio-applications: 2)P System Models for Mechanosensitive Channels, 3)P Systems for Biological Dynamics, 4)Modeling Respiration in Bacteria and Respiration/Photosynthesis Interaction in Cyanobacteria Using a P System Simulator, 5)Modeling Cell-Mediated Immunity by Means of P Systems, 6)A Membrane Computing Model of Photosynthesis, 7)Modeling

p53 Signaling Pathways by Using Multiset Processing; (II)Computer Science Applications: 8)Static Sorting P Systems, 9)Membrane-Based Devices Used in Computer Graphics, 10)An Analysis of a Public Key Protocol with Membranes, 11)Membrane Algorithms: Approximate Algorithms for NP-Complete Optimization Problems, 12)Computationally Hard Problems Addressed Through P Systems; (III)Applications to Linguistics: 13)Linguistic Membrane Systems and Applications, 14)Parsing with P Automata, (IV)Membrane Software: 15)Available Membrane Computing Software.

Let us return to the unconventional computing in general. The most obvious explanations of importance of unconventional computing are the miniaturization as the aim of devising small massively parallel computers or information storages, and the Moore's law, predicting the increase of speed, energy efficiency and compactness of future computers.

Looking deeper, one can easily see a number of other reasons to focus on unconventional computing, without directly having applications in mind. For instance, 1)developing new methods of algorithm design for conventional computers, 2)new perspective insights into the fundamental Physics laws, 3)new measures of information, relevant for non-standard data structures, 4)the interdisciplinary research bridging Classical computability, Information theory, Number theory, Biology, Physics, etc.

The author has published over 160 papers. Besides writing about 20 single-author publications, he has also collaborated with over 50 coauthors from Austria, China, Finland, France, Germany, Hungary, Italy, Japan, Kazakhstan, Macedonia, Moldova, Mongolia, Romania and Spain. The author has publications in the most prestigious journals in Theoretical computer science, e.g., *Theoretical Computer Science* and *Information Processing Letters* by Elsevier, *Natural Computing* and *Acta Informatica* by Springer, *Fundamenta Informaticae* by the Polish Mathematical Society, *International Journal of Foundations of Computer Science* by World Scientific, and *New Generation Computing* by Ohmsha Ltd. and Springer Japan. He has over 30 articles in *Lecture Notes of Computer Science* published by Springer, and chapters in monographs *The Oxford Handbook of Membrane Computing* by the Oxford University Press, *Bio-Inspired Models for Natural and Formal Languages* by the Cambridge Scholars Publishing, *Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory* by Imperial College Press/World Scientific, and *Applications of Membrane Computing* in the Natural Computing Series by Springer.

By the completion of this thesis, DBLP [286] has shown 37 journal papers and 37 conference ones, and Google Scholar [287] has reported the author's **h-index** of 16 and **i10-index** of 30, having registered over 880 **citations**. Out of over 200 researchers working in the areas mentioned below, the author is one of the main contributors to, e.g., the following research: maximally parallel multiset rewriting, non-cooperative P systems with/without control, transitional P systems, symport/antiport, evolution-communication P systems, active membranes, reversibility in membrane computing, and the networks of evolutionary processors. The author has produced some exotic results, e.g., on tatami tilings, and on solving the Hamiltonian path problem by ciliate gene assembly (with potential implementation in a laboratory), as well as a number of **applied** results, e.g., on the multicriterial bottleneck transportation problem, sorting, synchronization, chaining in logic, polymorphism, inflections in the Romanian language, and annotating affixes in the Romanian language.

Bibliography

1. Adleman L. Molecular Computation of Solutions to Combinatorial Problems. *Science* **226**, 1994, p. 1021–1024.
2. Alhazov A. A Note on P Systems with Activators. *Third Brainstorming Week on Membrane Computing*, RGNC report **01/2005**, Sevilla: Fénix Editora, 2005, p. 16–19.
3. Alhazov A. Ciliate Operations without Context in a Membrane Computing Framework. *Romanian Journal of Information Science and Technology* **10**, 4, 2007, p. 315–322.
4. Alhazov A. Ciliate Operations without Context in a Membrane Computing Framework. Technical Report **855**, Turku: *Turku Centre for Computer Science*, 2007.
5. Alhazov A. *Communication in Membrane Systems with Symbol Objects*. PhD Thesis, Rovira i Virgili University, 2006, 218 p.
6. Alhazov A. Developments in Networks of Evolutionary Processors. *Computer Science Journal of Moldova* **21**, 1(61), 2013, p. 3–35.
7. Alhazov A. Maximally Parallel Multiset-Rewriting Systems: Browsing the Configurations. *Third Brainstorming Week on Membrane Computing*, RGNC report **01/2005**, Univ. Seville, Sevilla: Fénix Editora, 2005, p. 1–10.
8. Alhazov A. Minimal Parallelism and Number of Membrane Polarizations. *Computer Science Journal of Moldova* **18**, 2(53), 2010, p. 149–170.
9. Alhazov A. Minimal Parallelism and Number of Membrane Polarizations. *Triangle. Language, Literature, Computation* **6**, Languages: Bioinspired Approaches, Tarragona: Publicacions URV, 2011, p. 1–18.
10. Alhazov A. Minimal Parallelism and Number of Membrane Polarizations. Preproc. *7th Int'l Workshop on Membrane Computing*, WMC7, Leiden, 2006, p. 74–87.
11. Alhazov A. Minimizing Evolution-Communication P Systems and Automata. *New Generation Computing* **22**, 4, 2004, p. 299–310.
12. Alhazov A. Minimizing Evolution-Communication P Systems and EC P Automata. *Brainstorming Week on Membrane Computing*, Technical Report **26/03**, Tarragona: Rovira i Virgili University, 2003, p. 23–31.
13. Alhazov A. Number of Protons/Bi-stable Catalysts and Membranes in P Systems. Time-Freeness. Membrane Computing, 6th Int'l Workshop, WMC 2005, Vienna, *Lecture Notes in Computer Science* **3850**, Springer, 2006, p. 79–95.
14. Alhazov A. Number of Protons/Bi-stable Catalysts and Membranes in P Systems. Time-Freeness. Preproc. *6th Int'l Workshop on Membrane Computing*, WMC6, Vienna, 2005, p. 102–122.
15. Alhazov A. On Determinism of Evolution-Communication P Systems. *Journal of Universal Computer Science* **10**, 5, 2004, p. 502–508.
16. Alhazov A. On the Power of Deterministic EC P Systems. *Third Brainstorming Week on Membrane Computing*, RGNC report **01/2005**, Univ. Seville, Sevilla: Fénix Editora, 2005, p. 11–15.
17. Alhazov A. P Systems without Multiplicities of Symbol-Objects. *Information Processing Letters* **100**, 3, 2006, p. 124–129.

18. Alhazov A. Properties of Membrane Systems. Membrane Computing - 12th International Conference, CMC12, Fontainebleau, *Lecture Notes in Computer Science* **7184**, 2012, p. 1–13.
19. Alhazov A. Properties of Membrane Systems. Preproc. *Twelfth Conference on Membrane Computing*, CMC12, Fontainebleau, 2011, p. 3–14.
20. Alhazov A. Solving SAT by Symport/Antiport P Systems with Membrane Division. *Cellular Computing (Complexity Aspects)*, ESF PESC Exploratory Workshop, Sevilla: Fénix Editora, 2005, p. 1–6.
21. Alhazov A., Antoniotti M., Freund R., Leporati A., Mauri G. Self-Stabilization in Membrane Systems. *Computer Sc. Journal of Moldova* **20**, 2(59), 2012, p. 133–146.
22. Alhazov A., Antoniotti M., Freund R., Leporati A., Mauri G. Self-Stabilization in Membrane Systems. RGNC report **1/2012**, *10th Brainstorming Week on Membrane Computing*, vol. I, Sevilla: Fénix Editora, 2012, p. 1–10.
23. Alhazov A., Antoniotti M., Leporati A. Characterizing the Computational Power of Energy-Based P Systems. *International Journal of Computer Mathematics* **90**, 4, 2013, p. 789–800.
24. Alhazov A., Antoniotti M., Leporati A. Characterizing the Computational Power of Energy-Based P Systems. RGNC report **1/2012**, *10th Brainstorming Week on Membrane Computing*, vol. I, Sevilla: Fénix Editora, 2012, p. 11–24.
25. Alhazov A., Bel-Enguix G., Epifanova I., Rogozhin Yu. About Two Models of Complete Obligatory Hybrid Networks of Evolutionary Processors. In preparation.
26. Alhazov A., Bel-Enguix G., Krassovitskiy A., Rogozhin Yu. About Complete Obligatory Hybrid Networks of Evolutionary Processors without Substitution. Advances in Computational Intelligence, 11th Int'l Work-Conference on Artificial Neural Networks, IWANN 2011, Málaga, *Lecture Notes in Computer Science* **6691**, 2011, p. 441–448.
27. Alhazov A., Bel-Enguix G., Krassovitskiy A., Rogozhin Yu. Complete Obligatory Hybrid Networks of Evolutionary Processors. Highlights in Practical Applications of Agents and Multiagent Systems, Salamanca, *Advances in Intelligent and Soft Computing* **89**, 2011, p. 275–282.
28. Alhazov A., Bel-Enguix G., Rogozhin Yu. About a New Variant of HNEPs: Obligatory Hybrid Networks of Evolutionary Processors. *Bio-Inspired Models for Natural and Formal Languages*, Cambridge Scholars Publishing, 2011, p. 191–204.
29. Alhazov A., Bel-Enguix G., Rogozhin Yu. Obligatory Hybrid Networks of Evolutionary Processors. *International Conference on Agents and Artificial Intelligence*, Porto, 2009, INSTICC Press, 613-618.
30. Alhazov A., Boian E., Ciubotaru C., Cojocaru S., Colesnicov A., Malahova L., Rogozhin Yu. Application of P System Models in Computer Linguistics. Proceedings of the *International Workshop on Intelligent Information Systems*, IIS2011, Chişinău, 2011, p. 101–104.
31. Alhazov A., Boian E., Cojocaru S., Rogozhin Yu. Modelling Inflections in Romanian Language by P Systems with String Replication. *Computer Science Journal of Moldova* **17**, 2(50), 2009, p. 160–178.
32. Alhazov A., Boian E., Cojocaru S., Rogozhin Yu. Modelling Inflections in Romanian Language by P Systems with String Replication. Preproc. *Tenth Workshop on Membrane Computing*, WMC10, Curtea de Argeş, RGNC report **3/2009**, University of Seville, 2009, p. 116–128.
33. Alhazov A., Bonchiş C., Ciobanu G., Izbaşa C. Encodings and Arithmetic Operations in P Systems. *4th Brainstorming Week on Membrane Computing*, RGNC report **02/2006**, Univ. Seville, vol. I, Sevilla: Fénix Ed., 2006, p. 1–28.

34. Alhazov A., Burtseva L., Cojocaru S., Rogozhin Yu. Computing Solutions of #P-complete Problems by P Systems with Active Membranes. Preproceedings of the *Ninth Workshop on Membrane Computing*, WMC9, Edinburgh, 2008, p. 59–70.
35. Alhazov A., Burtseva L., Cojocaru S., Rogozhin Yu. Solving PP-Complete and #P-Complete Problems by P Systems with Active Membranes. Membrane Computing - 9th International Workshop, WMC 2008, Edinburgh, *Lecture Notes in Computer Science* **5391**, Springer, 2009, p. 108–117.
36. Alhazov A., Cavaliere M. Computing by Observing Bio-Systems: the Case of Sticker Systems. DNA Computing: 10th International Workshop on DNA Computing, DNA10, Milan, Revised Selected Papers, *Lecture Notes in Computer Science* **3384**, Springer, 2005, p. 1–13.
37. Alhazov A., Cavaliere M. Computing by Observing Bio-Systems: the Case of Sticker Systems. *Tenth International Meeting on DNA Computing*, DNA10, Milan: University of Milano-Bicocca, 2004, p. 324–333.
38. Alhazov A., Cavaliere M. Evolution-Communication P Systems: Time-freeness. *Third Brainstorming Week on Membrane Computing*, RGNC report **01/2005**, Univ. Seville, Sevilla: Fénix Editora, 2005, p. 11–18.
39. Alhazov A., Cavaliere M. Proton Pumping P Systems. Membrane Computing, Int'l Workshop, WMC 2003, Tarragona, *Lecture Notes in Computer Science* **2933**, Springer, 2004, p. 1–18.
40. Alhazov A., Cavaliere M. Proton Pumping P Systems. Preproc. *Workshop on Membrane Computing*, TR **28/03**, Tarragona: Rovira i Virgili Univ., 2003, p. 1–16.
41. Alhazov A., Ciubotaru C., Ivanov S., Rogozhin Yu. Membrane Systems Languages Are Polynomial-Time Parsable. *The Computer Science Journal of Moldova* **18**, 2(53), 2010, 139-148.
42. Alhazov A., Ciubotaru C., Rogozhin Yu., Ivanov S. Introduction to the Membrane Systems Language Class. *Telecommunications, Electronics and Informatics*, 3rd International Conference, ICTEI 2010, Proceedings, vol. II, Chişinău, 2010, p. 19–24.
43. Alhazov A., Ciubotaru C., Rogozhin Yu., Ivanov S. The Family of Languages Generated by Non-Cooperative Membrane Systems. Membrane Computing - 11th Int'l Conference, CMC11, Jena, *Lecture Notes in Computer Science* **6501**, 2011, p. 65–79.
44. Alhazov A., Ciubotaru C., Rogozhin Yu., Ivanov S. The Family of Languages Generated by Non-Cooperative Membrane Systems. Preproc. *Eleventh Conference on Membrane Computing*, CMC11, Jena, Berlin: Verlag ProBusiness, 2010, p. 37–51.
45. Alhazov A., Ciubotaru C., Rogozhin Yu., Ivanov S. The Membrane Systems Language Class. LA symposium, *Mathematical Foundation of Algorithms and Computer Science*, RIMS Kôkyûroku Series **1691**, Kyoto University, 2010, p. 44–50.
46. Alhazov A., Ciubotaru C., Rogozhin Yu., Ivanov S. The Membrane Systems Language Class. RGNC report **1/2010**, *Eight Brainstorming Week on Membrane Computing*, Sevilla: Fénix Editora, 2010, p. 23–35.
47. Alhazov A., Ciubotaru C., Rogozhin Yu., Ivanov S. The Membrane Systems Language Class. LA sympos., Kyoto Univ., 2010, p. 12-1 – 12-9.
48. Alhazov A., Cojocaru S., Colesnicov A., Malahov L., Petic M. A P system for annotation of Romanian affixes. 14th International *Conference on Membrane Computing*, Chişinău, submitted, 2013.
49. Alhazov A., Cojocaru S., Malahova L., Rogozhin Yu. Dictionary Search and Update by P Systems with String-Objects and Active Membranes. *International Journal of Computers, Communications and Control* **3**, 2009, p. 206–213.

50. Alhazov A., Cojocaru S., Malahova L., Rogozhin Yu. Dictionary Search and Update by P Systems with String-Objects and Active Membranes. *7th Brainstorming Week on Membrane Computing*, RGNC report **1/2009**, vol. I, Sevilla: Fénix Editora, 2009, p. 1–8.
51. Alhazov A., Csuhaj-Varjú E., Martín-Vide C., Rogozhin Yu. About Universal Hybrid Networks of Evolutionary Processors of Small Size. Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Revised Papers, *Lecture Notes in Computer Science* **5196**, Springer, 2008, p. 28–39.
52. Alhazov A., Csuhaj-Varjú E., Martín-Vide C., Rogozhin Yu. About Universal Hybrid Networks of Evolutionary Processors of Small Size. Preproc. *2nd International Conference on Language and Automata Theory and Applications*, LATA 2008, Tarragona, 2008, p. 43–54.
53. Alhazov A., Csuhaj-Varjú E., Martín-Vide C., Rogozhin Yu. Computational Completeness of Hybrid Networks of Evolutionary Processors with Seven Nodes. Preproc. *10th Int'l Workshop on Descriptional Complexity of Formal Systems*, Charlottetown, Canada, 2008, p. 38–47.
54. Alhazov A., Csuhaj-Varjú E., Martín-Vide C., Rogozhin Yu. On the Size of Computationally Complete Hybrid Networks of Evolutionary Processors. *Theoretical Computer Science* **410**, 35, 2009, p. 3188–3197.
55. Alhazov A., Dassow J., Martín-Vide C., Rogozhin Yu., Truthe B. On Networks of Evolutionary Processors with Nodes of Two Types. *Fundamenta Informaticae* **91**, 1, 2009, p. 1–15.
56. Alhazov A., Freund R. Asynchronous and Maximally Parallel Deterministic Controlled Non-cooperative P Systems Characterize NFIN and coNFIN. Membrane Computing - 13th Int'l Conference, CMC13, Budapest, *Lecture Notes in Computer Science* **7762**, 2013, p. 101–111.
57. Alhazov A., Freund R. Asynchronous and Maximally Parallel Deterministic Controlled Non-cooperative P Systems Characterize NFIN and coNFIN. Preproc. *13th Int'l Conference on Membrane Computing*, CMC13, Budapest, 2012, p. 87–98.
58. Alhazov A., Freund R. Asynchronous and Maximally Parallel Deterministic Controlled Non-Cooperative P Systems Characterize NFIN U coNFIN. RGNC report **1/2012**, *10th Brainstorming Week on Membrane Computing*, vol. I, Sevilla: Fénix Editora, 2012, p. 25–34.
59. Alhazov A., Freund R. On Efficiency of P Systems with Active Membranes and Two Polarizations. Preproc. *Fifth Workshop on Membrane Computing*, WMC5, Milan: Univ. Milano-Bicocca, 2004, p. 81–94.
60. Alhazov A., Freund R. On the Efficiency of P Systems with Active Membranes and Two Polarizations. Membrane Computing, 5th International Workshop, WMC 2004, Milan, Revised Selected and Invited Papers, *Lecture Notes in Computer Science* **3365**, Springer, 2005, p. 146–160.
61. Alhazov A., Freund R. P Systems with One Membrane and Symport/ Antiport Rules of Five Symbols are Computationally Complete. *Third Brainstorming Week on Membrane Computing*, RGNC report **01/2005**, Sevilla: Fénix Editora, 2005, p. 19–28.
62. Alhazov A., Freund R., Leporati A., Oswald M., Zandron C. (Tissue) P Systems with Unit Rules and Energy Assigned to Membranes. *Fundamenta Informaticae* **74**, 4, 2006, p. 391–408.
63. Alhazov A., Freund R., Morita K. Determinism and Reversibility in P Systems with One Membrane. "Kasseler Informatikschriften" (KIS), Tagungsband zum 20. *Theo-*

rietag der GI-Fachgruppe "Automaten und Formale Sprachen", Technical report, urn:nbn:de:hebis:34-2010110534894, 2010, p. 39–44.
64. Alhazov A., Freund R., Morita K. Reversibility and Determinism in Sequential Multiset Rewriting. Unconventional Computation 2010, Tokyo, *Lecture Notes in Computer Science* **6079**, 2010, p. 21–31.
65. Alhazov A., Freund R., Morita K. Sequential and Maximally Parallel Multiset Rewriting: Reversibility and Determinism. *Natural Computing* **11**, 1, 2012, p. 95–106.
66. Alhazov A., Freund R., Oswald M. Cell / Symbol Complexity of Tissue P Systems with Symport / Antiport Rules. *International Journal of Foundations of Computer Science* **17**, 1, 2006, p. 3–26.
67. Alhazov A., Freund R., Oswald M. Symbol/Membrane Complexity of P Systems with Symport/Antiport Rules. Membrane Computing, 6th Int'l Workshop, WMC 2005, Vienna, Revised Sel. and Inv. Papers, *Lecture Notes in Computer Science* **3850**, Springer, 2006, p. 96–113.
68. Alhazov A., Freund R., Oswald M. Symbol / Membrane Complexity of P Systems with Symport / Antiport Rules. Preproc. *6th Int'l Workshop on Membrane Computing*, WMC6, Vienna, 2005, p. 123–146.
69. Alhazov A., Freund R., Oswald M. Tissue P Systems with Antiport Rules and Small Numbers of Symbols and Cells. Developments in Language Theory: 9th Int'l Conference, DLT 2005, Palermo, Proceedings, *Lecture Notes in Computer Science* **3572**, Springer, 2005, p. 100–111.
70. Alhazov A., Freund R., Oswald M. Tissue P Systems with Antiport Rules and Small Number of Symbols and Cells. *Cellular Computing (Complexity Aspects)*, ESF PESC Exploratory Workshop, Sevilla: Fénix Editora, 2005, p. 7–22.
71. Alhazov A., Freund R., Păun Gh. Computational Completeness of P Systems with Active Membranes and Two Polarizations. Machines, Computations, and Universality, International Conference, MCU 2004, Saint Petersburg, Revised Selected Papers, *Lecture Notes in Computer Science* **3354**, Springer, 2005, p. 82–92.
72. Alhazov A., Freund R., Păun Gh. P Systems with Active Membranes and Two Polarizations. *Third Brainstorming Week on Membrane Computing*, RGNC report **01/2005**, Univ. Seville, Sevilla: Fénix Editora, 2005, p. 20–36.
73. Alhazov A., Freund R., Riscos-Núñez A. Membrane Division, Restricted Membrane Creation and Object Complexity in P Systems. *Int'l Journal of Computer Mathematics* **83**, 7, 2006, p. 529–548.
74. Alhazov A., Freund R., Riscos-Núñez A. One and Two Polarizations, Membrane Creation and Objects Complexity in P Systems. *7th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, SYNASC 05, EEE Computer Society, 2005, p. 385–394.
75. Alhazov A., Freund R., Riscos-Núñez A. One and Two Polarizations, Membrane Creation and Objects Complexity in P Systems. Technical Report **05-11**, Institute e-Austria, Timişoara, Romania, *1st Int'l Workshop on Theory and Application of P Systems*, TAPS, 2005, p. 9–18.
76. Alhazov A., Freund R., Rogozhin Yu. Computational Power of Symport/Antiport: History, Advances and Open Problems. Membrane Computing, 6th Int'l Workshop, WMC 2005, Vienna, Revised Sel. and Inv. Papers, *Lecture Notes in Computer Science* **3850**, Springer, 2006, p. 1–30.
77. Alhazov A., Freund R., Rogozhin Yu. Computational Power of Symport / Antiport: History, Advances and Open Problems. Preproc. *6th Int'l Workshop on Membrane Computing*, WMC6, Vienna, 2005, p. 44–78.

78. Alhazov A., Freund R., Rogozhin Yu. Some Optimal Results on Symport/Antiport P Systems with Minimal Cooperation. *Cellular Computing (Complexity Aspects)*, ESF PESC Exploratory Workshop, Sevilla: Fénix Editora, 2005, p. 23–36.
79. Alhazov A., Ishdorj Ts.-O. Membrane Operations in P Systems with Active Membranes. *Triangle. Language, Literature, Computation* **6**, Languages: Bioinspired Appr., Tarragona: Publ. URV, 2011, p. 19–28.
80. Alhazov A., Ishdorj Ts.-O. Membrane Operations in P Systems with Active Membranes. *Third Brainstorming Week on Membrane Computing*, RGNC report **01/2005**, Univ. Seville, Sevilla: Fénix Editora, 2005, p. 37–44.
81. Alhazov A., Ivanov S., Rogozhin Yu. Polymorphic P Systems. Membrane Computing - 11th Int'l Conference, CMC11, Jena, *Lecture Notes in Computer Science* **6501**, 2011, p. 80–93.
82. Alhazov A., Ivanov S., Rogozhin Yu. Polymorphic P Systems. Preproc. *Eleventh Conference on Membrane Computing*, CMC11, Jena, Verlag ProBusiness Berlin, 2010, p. 53–66.
83. Alhazov A., Kogler M., Margenstern M., Rogozhin Yu., Verlan S. Small Universal TVDH and Test Tube Systems. *International Journal of Foundations of Computer Science* **22**, 1, 2011, p. 143–154.
84. Alhazov A., Krassovitskiy A., Rogozhin Yu. Circular Post Machines and P Systems with Exo-insertion and Deletion. Membrane Computing - 12th Int'l Conference, CMC12, Fontainebleau, *Lecture Notes in Computer Science* **7184**, 2012, p. 73–86.
85. Alhazov A., Krassovitskiy A., Rogozhin Yu. Circular Post Machines and P Systems with Exo-insertion and Deletion. Preproc. *Twelfth Conference on Membrane Computing*, CMC12, Fontainebleau, 2011, p. 63–76.
86. Alhazov A., Krassovitskiy A., Rogozhin Yu., Verlan S. A Note on P Systems with Small-Size Insertion and Deletion. Preproc. *Tenth Workshop on Membrane Computing*, WMC10, Curtea de Argeş, RGNC report **3/2009**, University of Seville, 2009, p. 534–537.
87. Alhazov A., Krassovitskiy A., Rogozhin Yu., Verlan S. P Systems with Insertion and Deletion Exo-operations. *Fundamenta Informaticae* **110**, 1, 2011, p. 13–28.
88. Alhazov A., Krassovitskiy A., Rogozhin Yu., Verlan S. P Systems with Minimal Insertion and Deletion. *Theoretical Computer Science* **412**, 1-2, 2011, p. 136–144.
89. Alhazov A., Krassovitskiy A., Rogozhin Yu., Verlan S. P Systems with Minimal Insertion and Deletion. *7th Brainstorming Week on Membrane Computing*, RGNC report **1/2009**, vol. I, Sevilla: Fénix Editora, 2009, p. 9–21.
90. Alhazov A., Krassovitskiy A., Rogozhin Yu., Verlan S. Small Size Insertion and Deletion Systems. *Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory*, Imperial College Press, 2010, p. 459–524.
91. Alhazov A., Kudlek M., Rogozhin Yu. Nine Universal Circular Post Machines. *The Computer Science Journal of Moldova* **10**, 3(30), Chişinău, 2002, p. 247–262.
92. Alhazov A., Leporati A., Mauri G., Porreca A.E., Zandron C. Simulating EXPSPACE Turing Machines Using P Systems with Active Membranes. *13th Italian Conference on Theoretical Computer Science*, ICTCS 2012, Varese, 2012, 4p.
93. Alhazov A., Leporati A., Mauri G., Porreca A.E., Zandron C. The Computational Power of Exponential-Space P Systems with Active Membranes. RGNC report **1/2012**, *10th Brainstorming Week on Membrane Computing*, vol. I, Sevilla: Fénix Editora, 2012, p. 35–60.
94. Alhazov A., Li C., Petre I. Computing the Graph-Based Parallel Complexity of Gene Assembly. *Theoretical Computer Science* **411**, 25, 2010, p. 2359–2367.

95. Alhazov A., Li C., Petre I. Computing the graph-based parallel complexity of gene assembly. Technical Report **859**, Turku: *Turku Centre for Computer Science*, 2007.
96. Alhazov A., Margenstern M., Rogozhin V., Rogozhin Yu., Verlan S. Communicative P Systems with Minimal Cooperation. Membrane Computing, 5th International Workshop, WMC 2004, Milan, Revised Selected and Invited Papers, *Lecture Notes in Computer Science* **3365**, Springer, 2005, p. 161–177.
97. Alhazov A., Margenstern M., Verlan S. Fast Synchronization in P Systems. Membrane Computing - 9th International Workshop, WMC 2008, Edinburgh, *Lecture Notes in Computer Science* **5391**, Springer, 2009, p. 118–128.
98. Alhazov A., Margenstern M., Verlan S. Fast Synchronization in P Systems. Preproc. *Ninth Workshop on Membrane Computing*, WMC9, Edinburgh, 2008, p. 71–84.
99. Alhazov A., Martín-Vide C., Pan L. Solving Graph Problems by P Systems with Restricted Elementary Active Membranes. Aspects of Molecular Computing, *Lecture Notes in Computer Science* **2950**, Springer, 2004, p. 1–22.
100. Alhazov A., Martín-Vide C., Pan L. Solving a PSPACE-Complete Problem by P Systems with Restricted Active Membranes. *Fundamenta Informaticae* **58**, 2, 2003, p. 67–77.
101. Alhazov A., Martín-Vide C., Rogozhin Yu. Networks of Evolutionary Processors with Two Nodes Are Unpredictable. Technical Report **818**, Turku: *Turku Centre for Computer Science*, 2007.
102. Alhazov A., Martín-Vide C., Rogozhin Yu. Networks of Evolutionary Processors with Two Nodes Are Unpredictable. *Language and Automata Theory and Applications*, Technical Report **35/07**, Tarragona: Rovira i Virgili University, 2007, p. 521–527.
103. Alhazov A., Martín-Vide C., Rogozhin Yu. On the Number of Nodes in Universal Networks of Evolutionary Processors. *Acta Informatica* **43**, 5, 2006, p. 331–339.
104. Alhazov A., Morita K. A Short Note on Reversibility in P Systems. *7th Brainstorming Week on Membrane Computing*, RGNC report **1/2009**, vol. I, Sevilla: Fénix Editora, 2009, p. 23–28.
105. Alhazov A., Morita K. On Reversibility and Determinism in P Systems. Membrane Computing, 10th Int'l Workshop, WMC 2009, Curtea de Argeş, *Lecture Notes in Computer Science* **5957**, 2010, p. 158–168.
106. Alhazov A., Morita K. On Reversibility and Determinism in P Systems. Preproc. *Tenth Workshop on Membrane Computing*, WMC10, Curtea de Argeş, RGNC report **3/2009**, University of Seville, 2009, p. 129–140.
107. Alhazov A., Pan L. Polarizationless P Systems with Active Membranes. *Grammars* **7**, 2004, p. 141–159.
108. Alhazov A., Pan L., Păun Gh. Trading Polarizations for Labels in P Systems with Active Membranes. *Acta Informatica* **41**, 2-3, 2004, p. 111–144.
109. Alhazov A., Petre I., Rogojin V. Solutions to Computational Problems through Gene Assembly. *Natural Computing* **7**, 3, 2008, p. 385–401.
110. Alhazov A., Petre I., Rogojin V. Solutions to Computational Problems through Gene Assembly. 13th Int'l Meeting on DNA Computing, DNA13, Memphis, TN, Revised Selected Papers, *Lecture Notes in Computer Science* **4848**, Springer, 2008, p. 36–45.
111. Alhazov A., Petre I., Rogojin V. Solutions to Computational Problems through Gene Assembly. Preproceedings of the *13th Int'l Workshop on DNA Computing*, 2007.
112. Alhazov A., Petre I., Rogojin V. The Parallel Complexity of Signed Graphs: Some Decidability Results and an Improved Algorithm. *Theoretical Computer Science* **410**, 24-25, 2009, p. 2308–2315.

113. Alhazov A., Petre I., Verlan S. A Sequence-Based Analysis of the Pointer Distribution of Ciliate Genes. Tech. Rep. **902**, Turku: *Turku Centre for Computer Science*, 2008.
114. Alhazov A., Pérez-Jiménez M.J. Uniform Solution of QSAT using Polarizationless Active Membranes. Machines, Computations, and Universality, 5th International Conference, MCU 2007, Orléans, *Lecture Notes in Computer Science* **4664**, Springer, 2007, p. 122–133.
115. Alhazov A., Pérez-Jiménez M.J. Uniform Solution to QSAT Using Polarizationless Active Membranes. *4th Brainstorming Week on Membrane Computing*, RGNC report **02/2006**, Univ. Seville, vol. I, Sevilla: Fénix Ed., 2006, p. 29–40.
116. Alhazov A., Rogozhin Yu. About Precise Characterization of Languages Generated by Hybrid Networks of Evolutionary Processors with One Node. *Computer Sci. J. of Moldova* **16**, 3, 2008, p. 364–376.
117. Alhazov A., Rogozhin Yu. Generating Languages by P Systems with Minimal Symport/Antiport. *The Computer Science Journal of Moldova* **14**, 3(42), 2006, p. 299–323.
118. Alhazov A., Rogozhin Yu. Minimal Cooperation in Symport /Antiport P Systems with One Membrane. *Third Brainstorming Week on Membrane Computing*, RGNC report **01/2005**, Univ. Seville, Sevilla: Fénix Editora, 2005, p. 29–34.
119. Alhazov A., Rogozhin Yu. One-Membrane Symport with Few Extra Symbols. *Int'l J. of Computer Mathematics* **90**, 4, 2013, p. 750–759.
120. Alhazov A., Rogozhin Yu. One-Membrane Symport P Systems with Few Extra Symbols. Preproc. *13th Int'l Conference on Membrane Computing*, CMC13, Budapest, 2012, p. 115–124.
121. Alhazov A., Rogozhin Yu. Skin Output in P Systems with Minimal Symport/Antiport and Two Membranes. Membrane Computing, 8th Int'l Workshop, WMC 2007, Thessaloniki, *Lecture Notes in Computer Science* **4860**, 2007, p. 97–112.
122. Alhazov A., Rogozhin Yu. Skin Output in P Systems with Minimal Symport/Antiport and Two Membranes. Preproc. *Eighth Workshop on Membrane Computing*, WMC8, Thessaloniki, 2007, p. 99–110.
123. Alhazov A., Rogozhin Yu. The Power of Symport-3 with Few Extra Symbols. RGNC report **1/2012**, *10th Brainstorming Week on Membrane Computing*, vol. I, Sevilla: Fénix Editora, 2012, p. 61–68.
124. Alhazov A., Rogozhin Yu. Towards a Characterization of P Systems with Minimal Symport/Antiport and Two Membranes. Membrane Computing, 7th Int'l Workshop, WMC 2006, Leiden, *Lecture Notes in Computer Science* **4361**, Springer, 2006, p. 135–153.
125. Alhazov A., Rogozhin Yu. Towards a Characterization of P Systems with Minimal Symport/Antiport and Two Membranes. Preproc. *7th Int'l Workshop on Membrane Computing*, WMC7, Leiden, 2006, p. 102–117.
126. Alhazov A., Rogozhin Yu., Verlan S. A Small Universal Splicing P System. Membrane Computing - 11th Int'l Conference, CMC11, Jena, *Lecture Notes in Computer Science* **6501**, 2011, p. 95–102.
127. Alhazov A., Rogozhin Yu., Verlan S. A Small Universal Splicing P System. Preproc. *Eleventh Conference on Membrane Computing*, CMC11, Jena, Berlin: Verlag ProBusiness, 2010, p. 67–74.
128. Alhazov A., Rogozhin Yu., Verlan S. Minimal Cooperation in Symport/Antiport Tissue P Systems. *Int'l Journal of Foundations of Computer Science* **18**, 1, 2007, p. 163–180.
129. Alhazov A., Rogozhin Yu., Verlan S. On Small Universal Splicing Systems. *Int'l J. Foundations of Comp. Sci.* **23**, 7, 2012, p. 1423–1438.

130. Alhazov A., Rogozhin Yu., Verlan S. Symport/Antiport Tissue P Systems with Minimal Cooperation. *Cellular Computing (Complexity Aspects)*, ESF PESC Exploratory Workshop, Sevilla: Fénix Editora, 2005, p. 37–52.
131. Alhazov A., Sburlan D. Static Sorting Algorithms for P Systems. Preproc. *Workshop on Membrane Computing*, TR **28/03**, Tarragona: Rovira i Virgili Univ., 2003, p. 17–40.
132. Alhazov A., Sburlan D. Static Sorting P Systems. In: Ciobanu G., Păun Gh., Pérez-Jiménez M.J.: *Applications of Membrane Computing*, Nat. Computing Ser., Springer, 2005, p. 215–252.
133. Alhazov A., Sburlan D. (Ultimately Confluent) Parallel Multiset-Rewriting Systems with Context. *Third Brainstorming Week on Membrane Computing*, RGNC report **01/2005**, Univ. Seville, Sevilla: Fénix Editora, 2005, p. 45–52.
134. Alhazov A., Sburlan D. (Ultimately Confluent) Parallel Multiset-Rewriting Systems with Permitting Contexts. Preproc. *Fifth Workshop on Membrane Computing*, WMC5, Milan: Univ. Milano-Bicocca, 2004, p. 95–103.
135. Alhazov A., Sburlan D. Ultimately Confluent Rewriting Systems. Parallel Multiset-Rewriting with Permitting or Forbidding Contexts. Membrane Computing, 5th International Workshop, WMC 2004, Milan, Revised Selected and Invited Papers, *Lecture Notes in Computer Science* **3365**, Springer, 2005, p. 178–189.
136. Alhazov A., Verlan S. Minimization Strategies for Maximally Parallel Multiset Rewriting Systems. *Theoretical Computer Science* **412**, 17, 2011, p. 1581–1591.
137. Alhazov A., Verlan S. Minimization Strategies for Maximally Parallel Multiset Rewriting Systems. Report arXiv:1009.2706v1 [cs.FL], 2010.
138. Alhazov A., Verlan S. Minimization Strategies for Maximally Parallel Multiset Rewriting Systems. Technical Report **862**, Turku: *Turku Centre for Computer Science*, 2008.
139. Alhazov A., Verlan S. Sevilla Carpets of Deterministic Non-cooperative P Systems. *Cellular Computing (Complexity Aspects)*, ESF PESC Exploratory Workshop, Sevilla: Fénix Editora, 2005, p. 53–60.
140. Arroyo F., Baranda A., Castellanos J., Păun Gh. Membrane Computing: The Power of (Rule) Creation. *Journal of Universal Computer Science* **8**, 3, 2002, p. 369–381.
141. Beene R. RNA-editing: The Alteration of Protein Coding Sequences of RNA. Ellis Horwood Series in Molecular Biology, Chichester, West Sussex, 1993.
142. Bernardini F., Gheorghe M. On the Power of Minimal Symport/Antiport. Preproc. *Workshop on Membrane Computing*, TR **28/03**, Tarragona: Rovira i Virgili Univ., 2003, p. 72–83.
143. Bennett C.H. Logical Reversibility of Computation. *IBM Journal of Research and Development* **17**, 1973, p. 525–532.
144. Bernardini F., Gheorghe M. Language Generating by Means of P Systems with Active Membranes. *Brainstorming Week on Membrane Computing*, Technical Report **26/03**, Tarragona: Rovira i Virgili University, 2003, p. 46–60.
145. Bernardini F., Gheorghe M., Margenstern M., Verlan S. How to Synchronize the Activity of All Components of a P System? Proc. *International Workshop Automata for Cellular and Molecular Computing*, Budapest: MTA SZTAKI, 2007, p. 11–22.
146. Castellanos J., Leupold P., Mitrana V. On the Size Complexity of Hybrid Networks of Evolutionary Processors. *Theoretical Computer Science* **330(2)**, 2005, p. 205–220.
147. Castellanos J., Martín-Vide C., Mitrana V., Sempere J. Networks of Evolutionary processors. *Acta Informatica* **38**, 2003, p. 517–529.
148. Castellanos J., Martín-Vide C., Mitrana V., Sempere J. *Solving NP-complete Problems with Networks of Evolutionary Processors.* IWANN 2001, *Lecture Notes in Computer Science* **2084**, Springer, 2001, p. 621–628.

149. Cavaliere M. Evolution-Communication P systems. Membrane Computing. Int'l Workshop, WMC 2002, Curteă de Argeş, Revised Papers, *Lecture Notes in Computer Science* **2597**, Springer, 2003, p. 134–145.
150. Cavaliere M., Genova D. P Systems with Symport/Antiport of Rules. *Journal of Universal Computer Science* **10**, 5, 2004, p. 540–558.
151. Cavaliere M., Ionescu M., Ishdorj Ts.-O. Inhibiting/De-inhibiting Rules in P Systems. Preproc. *Fifth Workshop on Membrane Computing*, WMC5, Milan: Univ. Milano-Bicocca, 2004, p. 174–183.
152. Ciobanu G., Pan L., Păun Gh., Pérez-Jiménez M.J. P Systems with Minimal Parallelism. *Theor. Computer Science* **378**, 2007, p. 117–130.
153. Cojocaru S. The Ascertainment of the Inflection Models for Romanian. *Computer Science Journal of Moldova* **14**, 1(40), 2006, p. 103–112.
154. Cojocaru S., Boian E. Determination of Inflexional Group using P Systems. *Computer Science Journal of Moldova* **18**, 1(52), 2010, p. 70–81.
155. Csuhaj-Varjú E. P Automata. Membrane Computing, 5th International Workshop, WMC 2004, Milan, Revised Selected and Invited Papers, *Lecture Notes in Computer Science* **3365**, Springer, 2005, p. 19–35.
156. Csuhaj-Varjú E., Kari L., Păun Gh. Test Tube Distributed Systems Based on Splicing. *Computers and Artificial Intelligence* **15(2-3)**, 1996, p. 211–232.
157. Csuhaj-Varjú E., Margenstern M., Vaszil Gy., Verlan S. Small Computationally Complete Symport/Antiport P systems. *Theoretical Computer Science* **372** (2-3), 2007, p. 152–164.
158. Csuhaj-Varjú E., Ibarra O.H., Vaszil Gy. On the Computational Complexity of P Automata. *Natural Computing* **5**, 2, 2006, p. 109–126.
159. Csuhaj-Varjú E., Martín-Vide C., Mitrana V. Hybrid Networks of Evolutionary Processors are Computationally Complete. *Acta Informatica* **41(4-5)**, 2005, p. 257–272.
160. Csuhaj-Varjú E., Salomaa A. Networks of Parallel Language Processors. *New Trends in Formal Language Theory*, *Lecture Notes in Computer Science* **1218**, Springer, 1997, p. 299–318.
161. Daley M., Kari L., Gloor G., Siromoney R. Circular Contextual Insertions/Deletions with Applications to Biomolecular Computation. In Proc. of 6th Int. Symp. on *String Processing and Information Retrieval*, SPIRE'99, Cancun, p. 47–54.
162. Dassow J., Păun Gh. *Regulated Rewriting in Formal Language Theory*. EATCS Monographs in Theor. Computer Science **18**, Springer, 1989.
163. Dijkstra E.W. Self-stabilizing Systems in Spite of Distributed Control. *Communication of the ACM* **17(11)**, 1974, p. 643–644.
164. Emerson E.A. Temporal and Modal Logic. *Handbook of Theoretical Computer Science*, Chapter 16, the MIT Press, 1990.
165. Fredkin E., Toffoli T. Conservative Logic. *International Journal of Theoretical Physics* **21**, 1982, p. 219–253.
166. Freund R. Energy-controlled P Systems. Membrane Computing. Int'l Workshop, WMC 2002, Curteă de Argeş, Revised Papers, *Lecture Notes in Computer Science* **2597**, Springer, 2003, p. 247–260.
167. Freund R. Generalized P-Systems, Proceedings of Fundamentals of Computation Theory, *Lecture Notes in Computer Science* **1684**, Springer, 1999, p. 281–292.
168. Freund R. Sequential P-Systems, *Romanian Journal of Information Science and Technology* **4**, 1-2, 2001, p. 77–88.
169. Freund R., Alhazov A., Rogozhin Yu., Verlan S. Communication P Systems. Oxford Handbook of Membrane Computing, 2010, p. 118–143.

170. Freund R., Ibarra O.H., Păun Gh., Yen H.-C. Matrix Languages, Register Machines, Vector Addition Systems. *Third Brainstorming Week on Membrane Computing*, RGNC report **01/2005**, Univ. Seville, Sevilla: Fénix Editora, 2005, p. 155–168.
171. Freund R., Kari L., Oswald M., Sosík P. Computationally Universal P Systems without Priorities: Two Catalysts are Sufficient. *Theoretical Computer Science* **330**, 2005, p. 251–266.
172. Freund R., Kogler M., Oswald M. A General Framework for Regulated Rewriting Based on the Applicability of Rules. *Computation, Cooperation, and Life*, *Lecture Notes in Computer Science* **6610**, Springer, 2011, p. 35–53.
173. Freund R., Leporati A., Oswald M., Zandron C. Sequential P Systems with Unit Rules and Energy Assigned to Membranes. Machines, Computations, and Universality, International Conference, MCU 2004, Saint Petersburg, Revised Selected Papers, *Lecture Notes in Computer Science* **3354**, Springer, 2005, p. 200–210.
174. Freund R., Oswald M. GP Systems with Forbidding Context. *Fundamenta Informaticae* **49**, 1-3, 2002, p. 81–102.
175. Freund R., Oswald M. P Systems with Activated/Prohibited Membrane Channels. Membrane Computing. Int'l Workshop, WMC 2002, Curteă de Argeş, Revised Papers, *Lecture Notes in Computer Science* **2597**, Springer, 2003, p. 261–268.
176. Freund R., Oswald M. Small Universal Antiport P Systems and Universal Multiset Grammars. *4th Brainstorming Week on Membrane Computing*, RGNC report **03/2006**, Univ. Seville, vol. II, Sevilla: Fénix Editora, 2006, p. 51–64.
177. Freund R., Oswald M. Tissue P Systems with Symport/Antiport Rules of One Symbol are Computationally Universal. *Cellular Computing (Complexity Aspects)*, ESF PESC Exploratory Workshop, Sevilla: Fénix Editora, 2005, p. 187–200.
178. Freund R., Păun A. Membrane Systems with Symport/Antiport: Universality Results. Membrane Computing. Int'l Workshop, WMC 2002, Curteă de Argeş, Revised Papers, *Lecture Notes in Computer Science* **2597**, Springer, 2003, p. 270–287.
179. Freund R., Păun Gh. From Regulated Rewriting to Computing with Membranes: Collapsing Hierarchies. *Theoretical Computer Science* **312**, 2004, p. 143–188.
180. Freund R., Păun Gh. On Deterministic P Systems, 2003, see [284].
181. Freund R., Păun Gh. On the Number of Non-terminals in Graph-controlled, Programmed, and Matrix Grammars. 3rd Int'l Conf. *Machines, Computations, and Universality*, *Lecture Notes in Computer Science* **2055**, Springer, 2001, p. 214–225.
182. Freund R., Păun Gh., Pérez-Jiménez M.J. Tissue-like P Systems with Channel States. *Theoretical Computer Science* **330**, 2005, p. 101–116.
183. Freund R., Rogozhin Yu., Verlan S. P Systems with Minimal Left and Right Insertion and Deletion. Unconventional Computation and Natural Computation, 11th International Conference, Orléans, *Lecture Notes in Computer Science* **7745**, Springer, 2012, p. 82–93.
184. Freund R., Rogozhin Yu., Verlan S. Generating and Accepting P Systems with Minimal Left and Right Insertion and Deletion. *Natural Computing*, submitted.
185. Freund R., Verlan S. A Formal Framework for Static (Tissue) P systems. Membrane Computing, 8th Int'l Workshop, WMC 2007, Thessaloniki, *Lecture Notes in Computer Science* **4860**, 2007, p. 271–284.
186. Frisco P. *Computing with Cells: Advances in Membrane Computing*, Oxford University Press, 2009.
187. Frisco P. The Conformon-P system: A Molecular and Cell Biology-Inspired Computability Model. *Theoretical Computer Science* **312**(2-3), 2004, p. 295–319.

188. Frisco P., Hoogeboom H.J. P Systems with Symport/Antiport Simulating Counter Automata. *Acta Informatica* **41**, 2004, p. 145–170.
189. Frisco P., Hoogeboom H.J. Simulating Counter Automata by P Systems with Symport/Antiport. Membrane Computing. Int'l Workshop, WMC 2002, Curteă de Argeş, Revised Papers, *Lecture Notes in Computer Science* **2597**, Springer, 2003, p. 288–301.
190. Galiukschov B.S. Semicontextual Grammars. *Matematika Logica i Matematika Linguistika*, Tallin University, 1981, p. 38–50 (in Russian).
191. Gill J. Computational Complexity of Probabilistic Turing Machines, *SIAM Journal on Computing* **6** (4), 1977, p. 675–695.
192. Goto E. A Minimum Time Solution of the Firing Squad Problem. *Course Notes for Applied Mathematics* **298**, Harvard University, 1962.
193. Gruska J. *Foundations of Computing*. International Thompson Computer Press, 1997.
194. Head T. Formal Language Theory and DNA: An Analysis of the Generative Capacity of Recombinant Behaviors. *Bulletin of Mathematical Biology* **49**, 1987, p. 737–759.
195. Ibarra O.H. On Strong Reversibility in P Systems and Related Problems. *International Journal of Foundations of Computer Science* **22** (1), 2011, p. 7–14.
196. Ibarra O.H., Păun Gh. Characterizations of Context-Sensitive Languages and Other Language Classes in Terms of Symport/Antiport P Systems. *Theoretical Computer Science* **358**, 1, 2006, p. 88–103.
197. Ibarra O.H., Yen H.-C. Deterministic Catalytic Systems are Not Universal. *Theoretical Computer Science* **363**, 2006, p. 149–161.
198. Ibarra O.H., Woodworth S. On Symport/ Antiport P Systems with One or Two Symbols. Proc. *7th International Symposyum SYNASC*, Timişoara, 2005, Washington: IEEE Computer Society, p. 431–439.
199. Ishdorj Ts.-O. Power and Efficiency of Minimal Parallelism in Polarizationless P Systems. *Journal of Automata, Languages, and Combinatorics* **11**, 3, 2006, p. 299–320.
200. K. Imai, Alhazov A. On Universality of Radius 1/2 Number-Conserving Cellular Automata. Unconventional Computation 2010, Tokyo, *Lecture Notes in Computer Science* **6079**, 2010, p. 45–55.
201. Ivanov S., Alhazov A., Rogojin V., Gutiérrez-Naranjo M.Á. Forward and Backward Chaining with P Systems. *International Journal of Natural Computing Research* (IJNCR) **2**, 2, 2011, p. 56–66.
202. Ivanov S., Alhazov A., Rogojin V., Gutiérrez-Naranjo M.Á. Forward and Backward Chaining with P Systems. *9th Brainstorming Week on Membrane Computing*, RGNC report **1/2011**, Sevilla: Fénix Ed., 2011, p. 221–236.
203. Karhumäki J., Kunc M., Okhotin A. Computational Power of Two Stacks with Restricted Communication. *Information and Computation* **208(9)**, 2010, p. 1060–1089.
204. Kari L. *On Insertion and Deletion in Formal Languages*, PhD Thesis, University of Turku, 1991.
205. Kari L., Păun Gh., Thierrin G., Yu S. At the Crossroads of DNA Computing and Formal Languages: Characterizing RE using Insertion-Deletion Systems. *DNA* Philadelphia, 1997, p. 318–333.
206. Kari L., Thierrin G. Contextual Insertion/Deletion and Computability. *Information and Computation* **131**, 1, 1996, p. 47–61.
207. Korec I. Small Universal Register Machines. *Theoretical Computer Science* **168**, 1996, p. 267–301.
208. Krassovitskiy A., Rogozhin Yu., Verlan S. Computational Power of Insertion-Deletion P Systems with Rules of Size Two. *Natural Computing* **10(2)**, Springer, 2011, p. 835–852.

209. Krassovitskiy A., Rogozhin Yu., Verlan S. Computational Power of P Systems with Small Size Insertion and Deletion Rules. *CSP* 2008, Cork, p. 137–148.
210. Krassovitskiy A., Rogozhin Yu., Verlan S. Further Results on Insertion-Deletion Systems with One-Sided Contexts. *LATA*, *Lecture Notes in Computer Science* **5196**, 2008, p. 347–358.
211. Krassovitskiy A., Rogozhin Yu., Verlan S. One-Sided Insertion and Deletion: Traditional and P Systems Case. *CBM* 2008, Vienna, p. 53–64.
212. Krishna S.-N. *Languages of P Systems. Computability and Complexity.* PhD thesis, Madras: Indian Institute of Technology, 2002.
213. Kudlek M., Rogozhin Yu. New Small Universal Circular Post Machines. Proc. FCT 2001, *Lecture Notes in Computer Science* **2138**, Springer, 2001, p. 217–227.
214. Kudlek M., Rogozhin Yu. Small Universal Circular Post Machines. *Computer Science Journal of Moldova* **9(1)**, 2001, p. 34–52.
215. Leporati A., Besozzi D., Cazzaniga P., Pescini D., Ferretti C. Computing with Energy and Chemical Reactions. *Natural Computing* **9**, 2010, p. 493–512.
216. Leporati A., Zandron C., Mauri G. Reversible P Systems to Simulate Fredkin Circuits. *Fundamenta Informaticae* **74** (4), 2006, p. 529–548.
217. Leporati A., Zandron C., Mauri G. Simulating the Fredkin Gate with Energy-Based P Systems. *Journal of Universal Computer Science* **10**(5), 2004, p. 600–619.
218. Lipton R.J. DNA Solution of Hard Computational Problems. *Science* **268**, 1995, p. 542–545.
219. Lombard A., Gâdei C. *Morphological Romanian Dictionary (Dictionnaire morphologique de la langue roumaine).* Bucureşti, Editura Academiei, 1981 (in French).
220. Loos R., Manea F., Mitrana V. Small Universal Accepting Hybrid Networks of Evolutionary Processors. *Acta Informatica* **47**, 2, 2010, p. 133–146.
221. Manea F., Martín-Vide C., Mitrana V. All NP-problems can be Solved in Polynomial Time by Accepting Hybrid Networks of Evolutionary Processors of Constant Size. *Information Processing Letters* **103**, 2007, p. 112–118.
222. Manea F., Martín-Vide C., Mitrana V. On the Size Complexity of Universal Accepting Hybrid Networks of Evolutionary Processors. *Mathematical Structures in Computer Science* **17(4)** 2007, p. 753–771.
223. Marcus S. *Contextual Grammars. Rev. Roum. Math. Pures Appl.* **14**, 1969, p. 1525–1534.
224. Margenstern M., Mitrana V., Pérez-Jiménez M.J. Accepting Hybrid Networks of Evolutionary Processors. DNA 10, *Lecture Notes in Computer Science* **3384**, Springer, 2005, p. 235–246.
225. Margenstern M., Păun Gh., Rogozhin Yu., Verlan S. Context-Free Insertion-Deletion Systems. *Theoretical Computer Science* **330**, 2005, p. 339–348.
226. Margenstern M., Rogozhin Yu. A Universal Time-Varying Distributed H System of Degree 1. *Lecture Notes in Computer Science* **2340**, 2002, p. 371–380.
227. Margenstern M., Rogozhin Yu., Verlan S. Time-Varying Distributed H Systems with Parallel Computations: the Problem is Solved. *Lecture Notes in Computer Science* **2943**, 2004, p. 48–53.
228. Margenstern M., Verlan S., Rogozhin Yu. Time-Varying Distributed H Systems: an Overview. *Fundamenta Informaticae* **64**, 2005, p. 291–306.
229. Martín-Vide C., Mitrana V., Pérez-Jiménez M.J., Sancho-Caparrini F. Hybrid Networks of Evolutionary Processors. Proc. of GECCO 2003, *Lecture Notes in Computer Science*, **2723**, Springer, 2003, p. 401–412.

230. Martín-Vide C., Pazos J., Păun Gh., Rodríguez-Patón A. Tissue P systems. *Theoretical Computer Science* **296**, 2003, p. 295–326.
231. Martín-Vide C., Păun Gh., Salomaa A. Characterizations of Recursively Enumerable Languages by means of Insertion Grammars. *Theoretical Computer Science* **205**, 1-2, 1998, p. 195–205.
232. Matveevici A., Rogozhin Yu., Verlan S. Insertion-Deletion Systems with One-Sided Contexts. Machines, Computations, and Universality, 5th Int'l Conference, MCU 2007, Orléans, *Lecture Notes in Computer Science* **4664**, Springer, 2007, p. 205–217.
233. M.L. Minsky *Computation: Finite and Infinite Machines*, Englewood Cliffs NJ: Prentice Hall, 1967.
234. Morita K. A Simple Reversible Logic Element and Cellular Automata for Reversible Computing. 3rd International Conference on *Machines, Computations, and Universality*, *Lecture Notes in Computer Science* **2055**, Springer, 2001, p. 102–113.
235. Morita K. Simple Universal One-Dimensional Reversible Cellular Automata. *Journal of Cellular Automata* **2**, 2007, p. 159–165.
236. Morita K. Universality of a Reversible Two-Counter Machine. *Theoretical Computer Science* **168**, 1996, p. 303–320.
237. Morita K., Yamaguchi Y. A Universal Reversible Turing Machine. Machines, Computations, and Universality, 5th International Conference, MCU 2007, Orléans, *Lecture Notes in Computer Science* **4664**, Springer, 2007, p. 90–98.
238. Morita K., Ogiro Ts., Alhazov A., Tanizawa Ts. Universality of Reversible Logic Elements with 1-Bit Memory. *Mathematical Foundations and Applications of Computer Science and Algorithms*, RIMS Kôkyûroku Series **1744**, Kyoto University, 2011, p. 77–84.
239. Morita K., Ogiro Ts., Alhazov A., Tanizawa Ts. Non-degenerate 2-State Reversible Logic Elements with Three or More Symbols Are All Universal. *Multiple-Valued Logic and Soft Comp.* **18**, 1, 2012, p. 37–54.
240. Morita K., Ogiro Ts., Alhazov A., Tanizawa Ts. Non-degenerate 2-State Reversible Logic Elements with Three or More Symbols Are All Universal. Proc. 2nd Workshop on Reversible Computation, Bremen, 2010, p. 27–34.
241. Ogiro Ts., Alhazov A., Tanizawa Ts., Morita K. Universality of 2-State 3-Symbol Reversible Logic Elements - A Direct Simulation Method of a Rotary Element. *Natural Computing, PICT 2*, Springer Japan, 2010, p. 252–259.
242. Ogiro Ts., Alhazov A., Tanizawa Ts., Morita K. Universality of 2-State 3-Symbol Reversible Logic Elements - A Direct Simulation Method of a Rotary Element. *4th International Workshop on Natural Computing*, Himeji, 2009, p. 220–227.
243. Pan L., Alhazov A. Solving HPP and SAT by P Systems with Active Membranes and Separation Rules. *Acta Inf.* **43**, 2, 2006, p. 131–145.
244. Pan L., Alhazov A., Ishdorj Ts.-O. Further Remarks on P Systems with Active Membranes, Separation, Merging, and Release Rules. *Soft Computing* **9**, 9, 2005, p. 686–690.
245. Pan L., Alhazov A., Ishdorj Ts.-O. Further Remarks on P Systems with Active Membranes, Separation, Merging, and Release Rules. *Third Brainstorming Week on Membrane Computing*, RGNC report **01/2005**, Sevilla: Fénix Editora, 2005, p. 316–324.
246. Papadimitriou C.H. *Computational Complexity*, Addison–Wesley, 1995.
247. Păun A., Păun Gh. The Power of Communication: P Systems with Symport/ Antiport. *New Generation Computing* **20**, 2002, p. 295–305.
248. Păun Gh. 2006 Research Topics in Membrane Computing. *4th Brainstorming Week on Membrane Computing*, RGNC report **02/2006**, vol. II, Sevilla: Fénix Ed., 2006, p. 235–252.

249. Păun Gh. Computing with Membranes. *Journal of Computer and System Sciences* **61**, 2000, p. 108–143.
250. Păun Gh. Computing with Membranes. Technical Report **208**, Turku: *Turku Centre for Computer Science*, 1998.
251. Păun Gh. *Marcus Contextual Grammars*. Kluwer, Dordrecht, 1997.
252. Păun Gh. *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.
253. Păun Gh. P Systems with Active Membranes: Attacking NP-complete Problems. *Journal of Automata, Languages and Combinatorics* **6**(1), 2001, p. 75–90.
254. Păun Gh., Pazos J., Pérez-Jiménez M.J., Rodríguez-Patón A. Symport/Antiport P Systems with Three Objects are Universal. *Fundamenta Informaticae* **64**, 2005, p. 1–4.
255. Păun Gh., Pérez-Jiménez M., Riscos-Núñez A. Tissue P Systems with Cell Division. *Third Brainstorming Week on Membrane Computing*, RGNC report **01/2005**, Univ. Seville, Sevilla: Fénix Editora, 2005, p. 380–386.
256. Păun Gh., Rozenberg G., Salomaa A. *DNA Computing. New Computing Paradigms*. Berlin: Springer, 1998.
257. Păun Gh., Rozenberg G., Salomaa A. (Eds.) *Handbook of Membrane Computing*, Oxford University Press, 2010.
258. Păun Gh., Rozenberg G., Salomaa A. Membrane Computing with an External Output. *Fundamenta Informaticae* **41**, 3, 2000, p. 313–340.
259. Păun Gh., Suzuki Y., Tanaka H. P Systems with Energy Accounting. *International Journal of Computer Mathematics* **78**, 3, 2001, p. 343–364.
260. Păun Gh., Suzuki Y., Tanaka H., Yokomori T. On the Power of Membrane Division in P Systems. *Theoretical Computer Science* **324**, 1, 2004, p. 61–85.
261. Pérez-Jiménez M.J., Romero-Jiménez A., Sancho-Caparrini F. Complexity Classes in Cellular Computing with Membranes. *Natural Computing* **2**, 3, 2003, p. 265–285.
262. Pérez-Jiménez M.J., Romero-Jiménez A., Sancho-Caparrini F. Solving VALIDITY Problem by Active Membranes with Input. *Brainstorming Week on Membrane Computing*, Tech. Rep. **26/03**, Tarragona: Rovira i Virgili University, 2003, p. 279–290.
263. Porreca A.E., Leporati A., Mauri G., Zandron C. P Systems with Active Membranes working in Polynomial Space. *International Journal of Foundations of Computer Science* **22**(1), 2011, p. 65–73.
264. Priese L., Rogozhin Yu., Margenstern M. Finite H-systems with 3 Test Tubes are not Predictable. Proceedings of *Pacific Simposium on Biocomputing*, Singapore: World Scientific Publishing, 1998, p. 545–556.
265. Rogozhin Yu., Verlan S. On the Rule Complexity of Universal Tissue P Systems. *Lecture Notes in Computer Science* **3850**, 2006, p. 356–362.
266. Qi Z., You J., Mao H. P Systems and Petri Nets. Membrane Computing, Int'l Workshop, WMC 2003, Tarragona, *Lecture Notes in Computer Science* **2933**, Springer, 2004, p. 286–303.
267. Rogozhin Yu. Small Universal Turing Machines. *Theoretical Computer Science* **168** (2), 1996, p. 215–240.
268. Rogozhin Yu., Alhazov A. Turing Computability and Membrane Computing. Membrane Computing - 13th Int'l Conference, CMC13, Budapest, *Lecture Notes in Computer Science* **7762**, 2013, p. 56–77.
269. Rozenberg G., Salomaa A. (Eds.) *Handbook of Formal Languages*, vol. 1-3, Springer, 1997.
270. Salomaa A. *Formal Languages*. New York: Academic Press, 1973.

271. Smith W. DNA computers in Vitro and in Vivo. Proc. *DIMACS* workshop, Providence: American Mathematical Society, 1996, p. 121–185.
272. Sosík P. The Computational Power of Cell Division. *Natural Computing* **2**, 3, 2003, p. 287–298.
273. Spellman P.T., Sherlock G. Reply Whole-cell Synchronization- effective Tools for Cell Cycle Studies. *Trends in Biotechnology* **22**(6), 2004, p. 270–273.
274. Ştefănescu G., Şerbănuţa T., Chira C., Roşu G. P Systems with Control Nuclei. Preproc. *Tenth Workshop on Membrane Computing*, WMC10, Curtea de Argeş, RGNC report **3/2009**, University of Seville, 2009, p. 361–365.
275. Takahara A., Yokomori T. On the Computational Power of Insertion-Deletion Systems. *DNA* 2002, Sapporo, *Lecture Notes in Computer Science* **2568**, 2003, p. 269–280.
276. Umeo H., Maeda M., Fujiwara N. An Efficient Mapping Scheme for Embedding any One-dimensional Firing Squad Synchronization Algorithm onto Two-dimensional Arrays. *ACRI 2002*, *Lecture Notes in Computer Science* **2493**, 2002, p. 69–81.
277. Valiant L.G. The Complexity of Computing the Permanent. *Theoretical Computer Science* **8**, Elsevier, 1979, p. 189–201.
278. Verlan S. On Minimal Context-Free Insertion-Deletion Systems. *J. Automata, Languages and Combinatorics* **12**, 1/2, 2007, p. 317–328.
279. Verlan S. *Study of Language-Theoretic Computational Paradigms Inspired by Biology.* Habilitation thesis, Créteil Val de Marne: Université Paris Est, 2010.
280. Verlan S. Tissue P Systems with Minimal Symport/Antiport. *Developments in Language Theory*, DLT 2004, *Lecture Notes in Computer Science* **3340**, Springer, p. 418–430.
281. Verlan S., Alhazov A., Petre I. A Sequence-Based Analysis of the Pointer Distribution of Stichotrichous Ciliates. *Biosystems* **101**, 2, 2010, p. 109–116.
282. Williams R.M., Wood D.H. Exascale Computer Algebra Problems Interconnect with Molecular Reactions and Complexity Theory, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **44**, 1999, p. 267–275.
283. Wood D. *Theory of Computation.* Harper and Row, New York, 1987.
284. P systems webpage. `http://ppage.psystems.eu`
See also `http://psystems.disco.unimib.it`,
`http://ppage2010.psystems.eu/` and `http://www.gcn.us.es/?q=workshops`
285. Publication page of Artiom Alhazov. `http://aartiom.50webs.com/pub_aa.html`
286. DBLP page of Artiom Alhazov.
`http://www.informatik.uni-trier.de/~ley/pers/hd/a/Alhazov:Artiom.html`
287. Google Scholar page of Artiom Alhazov.
`http://scholar.google.com/citations?sortby=pubdate&user=M8LdW5kAAAAJ`
288. `http://en.wikipedia.org/wiki/Permanent`
289. `http://en.wikipedia.org/wiki/PP_(complexity)`
290. `http://en.wikipedia.org/wiki/Self-stabilization`
291. `http://en.wikipedia.org/wiki/Sharp-P`

APPENDICES

A1. Context-free grammars and time-yield

Consider a non-terminal A in a grammar $G = (N, T, S, P)$. We denote by G_A the grammar (N, T, A, P) obtained by considering A as axiom in G.

A derivation tree in a context-free grammar is an ordered rooted tree with leaves labeled by terminals and all other nodes labeled by non-terminals. Rules of the form $A \to \lambda$ cause a problem, which can be solved by allowing to also label leaves by λ, or by transformation of the corresponding grammar. Note: by derivation trees we only mean finite ones. Consider a derivation tree τ. The following notion describes the sequence of terminal symbols at a particular depth of a derivation tree:

The n-th level yield $\texttt{yield}_n$ of τ can be defined as follows:

> We define $\texttt{yield}_0(\tau) = a$ if τ has a single node labeled by $a \in T$, and $\texttt{yield}_0(\tau) = \lambda$ otherwise.
>
> Let k be the number of children nodes of the root of τ, and $\tau_1, \cdots, \tau_k$ be the subtrees of τ with these children as roots. We define $\texttt{yield}_{n+1}(\tau) = \texttt{yield}_n(\tau_1) \bullet \texttt{yield}_n(\tau_2) \bullet \cdots \bullet \texttt{yield}_n(\tau_k)$.

We now define the time yield L_t of a context-free grammar derivation tree τ, as the usual yield except the order of terminals is vertical from root instead of left-to-right, and the order of terminals at the same distance from root is arbitrary. We use $\prod$ to denote concatenation in the following formal definition:

$$L_t(\tau) = \prod_{n=0}^{\texttt{height}(\tau)} (\texttt{Perm}(\texttt{yield}_n(\tau))).$$

The time yield $L_t(G)$ of a grammar G is the union of time yields of all its derivation trees. The corresponding family of languages is

$$L_t(CF) = \{L_t(G) \mid G \text{ is a context-free grammar}\}.$$

Example A1.1 *Consider a grammar* $G_1 = (\{S, A, B, C\}, \{a, b, c\}, S, P)$*, where* $P =$

$$\{S \to SABC, S \to ABC, A \to A, B \to B, C \to C, A \to a, B \to b, C \to c\}.$$

We now show that $L_t(G_1) = \{w \in \{a, b, c\}^* \mid |w|_a = |w|_b = |w|_c > 0\} = L$*. Indeed, all derivations of* A *are of the form* $A \Rightarrow^* A \Rightarrow a$*. Likewise, symbols* B, C *are also trivially rewritten an arbitrary number of times and then changes into a corresponding terminal. Hence,* $L_t(G_{1A}) = \{a\}$*,* $L_t(G_{1B}) = \{b\}$*,* $L_t(G_{1C}) = \{c\}$*. For inclusion* $L_t(G) \subseteq L$ *it suffices to note that* S *always generates the same number of symbols* A, B, C*.*

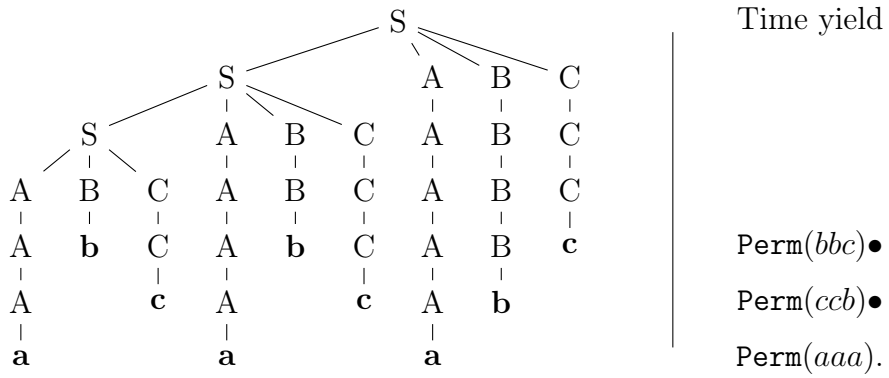


Figure A1.1: An example of a derivation of the grammar from Example A1.1

The converse inclusion follows from the following simulation: given a word $w \in L$, generate $|w|/3$ copies of A, B, C, and then apply their trivial rewriting in such way that the timing when the terminal symbols appear corresponds to their order in w.

Corollary A1.1 $L_t(CF) \not\subseteq CF$.

Derivation trees of context-free grammars

Theorem A1.1 $L_t(CF) = LOP(ncoo, tar)$.

Proof. By Lemma 2.1, the statement is equivalent to $L_t(CF) = LOP_1(ncoo, out)$. Consider a P system $\Pi = (O, [\ \ ]_1, w, R, 0)$. We construct a context-free grammar $G = (O' \cup \{S\}, O, S, P \cup \{S \to w\})$, where S is a new symbol, $'$ is a morphism from O into new symbols and

$$\begin{aligned} P &= \{a' \to u'v \mid (a \to u\ v_{out}) \in R,\ a \in O,\ u, v \in O^*\} \\ &\cup \{a' \to \lambda \mid \neg\exists (a \to u\ v_{out}) \in R\}. \end{aligned}$$

Here v_{out} are those symbols on the right-hand side of the rule in R which are sent out into the environment, and u are the remaining right-hand side symbols.

The computations of Π are identical to parallel derivations in G, except the following:

- Unlike G, Π does not keep track of the left-to-right order of symbols. This does not otherwise influence the derivation (since rules are context-free) or the result (since the order of non-terminals produced in the same step is arbitrary, and the timing is preserved).
- The initial configuration of Π is produced from the axiom of G in one additional step.
- The objects of Π that cannot evolve are erased in G, since they do not contribute to the result.

It follows that $L_t(CF) \supseteq LOP(ncoo, tar)$. To prove the converse inclusion, consider an arbitrary context-free grammar $G = (N, T, S, P)$. We construct a P system $\Pi = (N \cup$

$T, [\ ]_1, S, R, 0)$, where $R = \{a \to h(u) \mid (a \to u) \in R\}$, where h is a morphism defined by $h(a) = a$, $a \in N$ and $h(a) = a_{out}$, $a \in T$. The computations in Π correspond to parallel derivations in G, and the order of producing terminal symbols in G corresponds to the order of sending them to the environment by Π, hence the theorem statement holds. □

We now present a few normal forms for the context-free grammars in the context of the time yield. Note that these normal forms incorporate a number of similarities with both L systems and standard CF grammars, because level-by-level derivation in context-free grammars corresponds to the evolution in L systems. However, it is not possible to simply take existing normal forms, because the result must be preserved, and the result is defined in a different way. We omit the proofs; the reader can find them in [43].

Lemma A1.1 *(First normal form) For a context-free grammar G there exists a context-free grammar G' such that $L_t(G) = L_t(G')$ and in G':*

- *the axiom does not appear in the right-hand side of any rule, and*
- *if the left side is not the axiom, then the right-hand side is not empty.*

The First normal form shows that erasing can be limited to the axiom.

Lemma A1.2 *(Binary normal form) For a context-free grammar G there exists a context-free grammar G' such that $L_t(G) = L_t(G')$ and in G':*

- *the First normal form holds,*
- *the right-hand side of any production is at most 2.*

The Binary normal form shows that productions with right-hand side longer than two are not necessary.

Lemma A1.3 *(Third normal form) For a context-free grammar G there exists a context-free grammar G' such that $L_t(G) = L_t(G')$ and in G':*

- *the Binary normal form holds,*
- *$G' = (N, T, S, P')$ and every $A \in N$ is reachable,*
- *either $G' = (\{S\}, T, S, \{S \to S\})$, or $G' = (N, T, S, P')$ and for every $A \in N$, $L_t(G'_A) \neq \emptyset$.*

The Third normal form shows that never ending derivations are only needed to generate the empty language.

A2. Advanced control in one region

Flattening the membrane structure is a well-known technique transforming a P system into a one-region P system, representing each object a in each region i by an object a_i in the single region of the new system. A configuration of a membrane system (with a fixed structure) is the tuple of multisets contained in each region. We say that a system is deterministic if at every step, there is (at most) one multiset of applicable rules. Since flattening the membrane structure of a membrane system preserves both determinism and the model, in the following we restrict ourselves to consider membrane systems as one-region multiset rewriting systems.

A *(one-region) membrane system (P system)* is a tuple $\Pi = (O, \Sigma, w, R')$ where O is a finite alphabet, $\Sigma \subseteq O$ is the input subalphabet, $w \in O^*$ is a string representing the initial multiset, and R' is a set of rules of the form $r : u \to v$, $u \in O^+$, $v \in O^*$.

A configuration of the system Π is represented by a multiset of objects from O contained in the region, the set of all configurations over O is denoted by $\mathbb{C}(O)$. A rule $r : u \to v$ is applicable if the current configuration contains the multiset specified by u. Furthermore, applicability may be controlled by *context conditions*, specified by pairs of sets of multisets.

Definition A2.1 *Let P_i, Q_i be (finite) sets of multisets over O, $1 \leq i \leq m$. A* rule with context conditions $(r, (P_1, Q_1), \cdots, (P_m, Q_m))$ *is applicable to a configuration C if r is applicable, and there exists some $j \in \{1, \cdots, m\}$ for which*

- *there exists some $p \in P_j$ such that $p \subseteq C$ and*
- *$q \not\subseteq C$ for all $q \in Q_j$.*

In words, context conditions are satisfied if there exists a pair of sets of multisets (called *promoter set* and *inhibitor set*, respectively) such that at least one multiset in the promoter set is a submultiset of the current configuration, and no multiset in the inhibitor set is a submultiset of the current configuration.

Definition A2.2 *A* P system with context conditions and priorities on the rules *is a construct $\Pi = (O, \Sigma, w, R', R, >)$ where (O, Σ, w, R') is a (one-region) P system as defined above, R is a set of rules with context conditions and $>$ is a priority relation on the rules in R; if rule r' has priority over rule r, denoted by $r' > r$, then r cannot be applied if r' is applicable.*

We will use the word *control* to mean that at least one of these features is allowed (context conditions or promoters or inhibitors only and eventually priorities).

In the *sequential mode* (*sequ*), a computation step consists of the non-deterministic application of one applicable rule r, replacing its left-hand side ($lhs(r)$) with its right-hand side

$(rhs\,(r))$. In the *maximally parallel mode* $(maxpar)$, multiple applicable rules may be chosen non-deterministically to be applied in parallel to the underlying configuration to disjoint submultisets, possibly leaving some objects idle, under the condition that no further rule is simultaneously applicable to them (i.e., no supermultiset of the chosen multiset is applicable to the underlying configuration). Maximal parallelism is the most common computation mode in membrane computing, see also Definition 4.8 in [185]. In the *asynchronous mode* $(asyn)$, any positive number of applicable rules may be chosen non-deterministically to be applied in parallel to the underlying configuration to disjoint submultisets. The computation step between two configurations C and C' is denoted by $C \Rightarrow C'$, thus yielding the binary relation $\Rightarrow: \mathbb{C}\,(O) \times \mathbb{C}\,(O)$. A computation halts when there are no rules applicable to the current configuration (*halting configuration*) in the corresponding mode.

The computation of a *generating* P system starts with w, and its result is $|x|$ if it halts, an *accepting* system starts with wx, $x \in \Sigma^*$, and we say that $|x|$ is its result – is accepted – if it halts. The set of numbers generated/accepted by a P system working in the mode α is the set of results of its computations for all $x \in \Sigma^*$ and denoted by $N_g^{\alpha}(\Pi)$ and $N_a^{\alpha}(\Pi)$, respectively. The family of sets of numbers generated/accepted by a family of (one-region) P systems with context conditions and priorities on the rules with rules of type β working in the mode α is denoted by $N_{\delta}OP_1^{\alpha}\left(\beta, (pro_{k,l}, inh_{k',l'})_d, pri\right)$ with $\delta = g$ for the generating and $\delta = a$ for the accepting case; d denotes the maximal number m in the rules with context conditions $(r, (P_1, Q_1), \cdots, (P_m, Q_m))$; k and k' denote the maximum numbers of promoters/inhibitors in the P_i and Q_i, respectively; l and l' indicate the maximum of weights of promoters and inhibitors, respectively. If any of these numbers k, k', l, l' is not bounded, we replace it by $*$. As types of rules we here will distinguish between cooperative ($\beta = coo$) and non-cooperative (i.e., the left-hand side of each rule is a single object; $\beta = ncoo$) ones.

In the case of accepting systems, we also consider the idea of determinism, which means that in each step of any computation at most one (multiset of) rule(s) is applicable; in this case, we write D after N_a. It follows that, for any given input, the system has only one computation.

In the literature, we find a lot of restricted variants of P systems with context conditions and priorities on the rules, e.g., we may omit the priorities or the context conditions completely. If in a rule $(r, (P_1, Q_1), \cdots, (P_m, Q_m))$ we have $m = 1$, we say that $(r, (P_1, Q_1))$ is a rule with a *simple context condition*, and we omit the inner parentheses in the notation. Moreover, context conditions only using promoters are denoted by $r|_{p_1,\cdots,p_n}$, meaning $(r, \{p_1, \cdots, p_n\}, \emptyset)$, or, equivalently, $(r, (p_1, \emptyset), \cdots, (p_n, \emptyset))$; context conditions only using inhibitors are denoted by $r|_{\neg q_1,\cdots,\neg q_n}$, meaning $(r, \lambda, \{q_1, \cdots, q_n\})$, or $r|_{\neg\{q_1,\cdots,q_n\}}$. Likewise, a rule with both promoters and inhibitors can be specified as a rule with a simple context condition, i.e., $r|_{p_1,\cdots,p_n,\neg q_1,\cdots,\neg q_n}$ stands for $(r, \{p_1, \cdots, p_n\}, \{q_1, \cdots, q_n\})$. Finally, promoters and inhibitors of weight one are called *atomic*.

In what follows, when speaking about the effect of rules, we mean the behavior induced by them. Hence, two sets of rules have the same effect if substituting one of them by the other one does not change the computations of the system.

Remark A2.1 *If we do not consider determinism, then (the effect of) the rule*

$$(r, (P_1, Q_1), \cdots, (P_m, Q_m))$$

is equivalent to (the effect of) the collection of rules $\{(r, P_j, Q_j) \mid 1 \leq j \leq m\}$*, no matter in which mode the P system is working (obviously, the priority relation has to be adapted accordingly, too).*

Remark A2.2 *Let* $(r, \{p_1, \cdots, p_n\}, Q)$ *be a rule with a simple context condition; then we claim that (the effect of) this rule is equivalent to (the effect of) the collection of rules* $\{(r, \{p_j\}, Q \cup \{p_k \mid 1 \leq k < j\}) \mid 1 \leq j \leq m\}$ *even in the the case of a deterministic P system: If the first promoter is chosen to make the rule r applicable, we do not care about the other promoters; if the second promoter is chosen to make the rule r applicable, we do not allow* p_1 *to appear in the configuration, but do not care about the other promoters* p_3 *to* p_m*; in general, when promoter* p_j *is chosen to make the rule r applicable, we do not allow* p_1 *to* p_{j-1} *to appear in the configuration, but do not care about the other promoters* p_{j+1} *to* p_m*; finally, we have the rule* $\{(r, \{p_m\}, Q \cup \{p_k \mid 1 \leq k < m\})\}$*. If adding* $\{p_k \mid 1 \leq k < j\}$ *to Q has the effect of prohibiting the promoter* p_j *from enabling the rule r to be applied, this makes no harm as in this case one of the promoters* p_k*,* $1 \leq k < j$*, must have the possibility for enabling r to be applied. By construction, the domains of the new context conditions now are disjoint, so this transformation does not create (new) non-determinism. In a similar way, this transformation may be performed on context conditions which are not simple. Therefore, without restricting generality, the set of promoters may be assumed to be a singleton. In this case, we may omit the braces of the multiset notation for the promoter multiset and write* (r, p, Q)*.*

A3. A new variant of circular Post machines

We introduce a new variant of Circular Post Machines.

Definition A3.1 *A CPM5 is a tuple* $(\Sigma, Q, \mathbf{q}_1, \mathbf{q}_f, R)$ *with instructions of the following types (we use the notation* $Q' = Q \setminus \{\mathbf{q}_f\}$*):*

- $\mathbf{px} \to \mathbf{q}, p \in Q', q \in Q, x \in \Sigma$*, the same type as in CPM0.*
  The corresponding computational step is $pxW \overset{px\to q}{\Longrightarrow} qW,\ W \in \Sigma^*$.
- $\mathbf{p} \to \mathbf{yq}, p \in Q', q \in Q, y \in \Sigma$*. This is the new type of rule. Notice it does not consume a symbol. The corresponding computational step is* $pW \overset{p\to yq}{\Longrightarrow} qWy,\ W \in \Sigma^*$.

Theorem A3.1 *Any CPM0 P can be simulated by CPM5 P′.*

Proof. Consider a CPM0 $P = (\Sigma, Q, \mathbf{q}_1, \mathbf{q}_f, R)$ with $0 \in \Sigma$, and $Q' = Q \setminus \{q_f\}$.
We construct CPM5 $P' = (\overline{\Sigma}, \overline{Q}, \mathbf{q}_1, \mathbf{q}_f, \overline{R})$ simulating of CPM0 P:

$$\begin{aligned}
\overline{\Sigma} &= \Sigma \cup \{a_q \mid q \in \Sigma\},\\
\overline{Q} &= Q \cup \{p_x \mid (px \to qy) \in Q\} \cup \{p_0, p_0' \mid (p0 \to yq0) \in R\} \cup \{r', r\},\\
\overline{R} &= \{px \to q \mid (px \to q) \in R\} \cup \{px \to p_x,\ p_x \to yq \mid (px \to yq) \in R\}\\
&\cup \{p0 \to p_0,\ p_0 \to yp_0',\ p_0' \to a_q r',\ ra_q \to q \mid (p0 \to yq0) \in R\}\\
&\cup \{r' \to 0r\} \cup \{rx \to r_x,\ r_x \to xr \mid x \in \Sigma\}.
\end{aligned}$$

We claim P' is equivalent to P (it produces the same result). Let $pxW \overset{px\to yq}{\Longrightarrow} qWy$ be a computational step in P. The corresponding derivation in P' is $pxW \overset{px\to p_x}{\Longrightarrow} p_xW \overset{p_x\to yq}{\Longrightarrow} qWy$.

Consider the second case. Let $p0xW \overset{p0\to yq0}{\Longrightarrow} q0xWy$ be a computational step in P. The corresponding derivation in P' is

$$\begin{aligned}
p0xW &\overset{p0\to p_0}{\Longrightarrow} p_0xW \overset{p_0\to yp_0'}{\Longrightarrow} p'0xWy \overset{p_0'\to a_qr'}{\Longrightarrow} r'xWya_q \overset{r'\to 0r}{\Longrightarrow} rxWya_q0\\
&\overset{rx\to r_x}{\Longrightarrow} r_xWya_q0 \overset{r_x\to xr}{\Longrightarrow} rWya_q0x \Longrightarrow \cdots \Longrightarrow rya_q0xW\\
&\overset{ry\to r_y}{\Longrightarrow} r_ya_q0xW \overset{r_y\to yr}{\Longrightarrow} ra_q0xWy \overset{ra_q\to q}{\Longrightarrow} q0xWy.
\end{aligned}$$

It remains to notice that P' does not do anything more than the simulation of P. Indeed, even in the non-deterministic case, within the simulation of some computation step of P, P' is in some state $\overline{Q} \setminus Q$, whose behavior is bound to such simulation by construction. □

Remark A3.1 *If we do not require the simulation to go through all configurations of the computation, then a faster simulation of any CPM0 P by a CPM5 P′ is possible. Instructions* $px \to q$ *are performed in one step, and instructions* $px \to yq$ *are performed in two steps, so it suffices to consider instructions* $p0 \to yq0$.

Like in the proof above we may start with $p0 \to p_0$, $p_0 \to yp'_0$. Informally p'_0 corresponds to the state q, except symbol 0 has not been introduced, i.e., to the situation where it has "already been read" in state q. Hence, depending on the next instruction of P we sometimes replace p'_0 with an appropriate state, as follows. If P has $q0 \to s$, then it suffices to set $p'_0 = s$. If P has $q0 \to zs$, then it suffices to have a rule $p'_0 \to zs$. Finally, if P has $q0 \to zs0$, then it suffices to set $p'_0 = q_0$, so the simulation of the first instruction continues by simulation of the next one without the first step.

Consider the non-deterministic case. In fact, the following restriction of non-determinism suffices:

Definition A3.2 *An NCPM5 is a tuple $(\Sigma, Q, \mathbf{q}_1, \mathbf{q}_f, R)$, where*

- *$Q \setminus \{q_f\} = Q_1 \cup Q_2$, where $Q_1 \cap Q_2 = \emptyset$,*
- *For every $p \in Q_1$ and every $x \in \Sigma$, there is exactly one instruction of the form $px \to q$,*
- *For every $p \in Q_2$, there are two instructions of the form $p \to yq_1$, $p \to yq_2$ (and the machine is deterministic if $q_1 = q_2$ for every pair of instructions $p \to yq_1$, $p \to yq_2$).*

Corollary A3.1 *The class of (N)CPM5 is computationally complete.*

Proof. The statement is a trivial consequence of the theorem above in the sense of completeness as simulating Turing machines or as computing partial recursive functions. Let us prove the claim of the corollary also in the sense of generating languages.

We claim that NCPM5 generates all recursively enumerable languages. Indeed, for every $L \in RE$ there exists a deterministic Turing machine M that, starting with configurations $q_2 1^+$ generates exactly L. It is enough to consider an NCPM5 with rules $q_1 \to 1q_1$, $q_1 \to 1q_2$, where the rest of the machine starts with q_2 and is a deterministic CPM5 that simulates a CPM0 simulating M. □

A4. Two polarizations - "normal form"

In this subsection we now consider the following forms (particular cases) of the types (a), (c), (e) of rules (where $a, b, c \in O$, $h \in H$, $i \in \{0, 1\}$):

(a_{gb}) $[\ a \to bc\ ]^i$ (global split rule)

(a_{gu}) $[\ a \to b\ ]^i_h$ (rename only)

(c_{np1}) $[\ a\ ]_h \to [\ \ ]^\neg_h\ a$ (exit only, polarization switched)

(c_{gp1}) $[\ a\ ] \to [\ \ ]^\neg\ b$ (global exit rule, polarization switched)

(c_{gny}) $[\ \texttt{yes}\ ]^0 \to [\ \ ]^1\ \texttt{yes}$ (a special rule for ejecting the result)

(e_{gp0}) $[\ a\ ] \to [\ b\ ][\ c\ ]$ (global polarizationless division rule)

(e_{gp2}) $[\ a\ ] \to [\ b\ ]^0[\ c\ ]^1$ (global polarization-independent division rule, producing membranes of different polarizations)

In the subscripts of the rules, we write g if the rule is global (does not depend on the label of the membrane), n if the rule is not-renaming (the object(s) in (each membrane of) the right-hand side is(are) the same as the object in the left-hand side), p if the rule does not depend on the polarization, 0 if the rule preserves it, 1 if the rule changes it, 2 if the rule produces two membranes with different polarizations, and b (u) if the number of the objects in (each membrane of) the right-hand side is two (one, respectively). Finally, y is used if the rule acts on the object `yes`.

The main idea of the possible restrictions is the following: to try to make rules of types (c) and (e) independent of the polarization by remembering the needed value in a corresponding object, and then decoding it by generating copies of z if needed (using such an approach, the computation slows down by a constant factor). In the same time, other restrictions are put on the general form of the rules, leading to the following theorem:

Theorem A4.1 $\texttt{SAT}(n, m)$ *can be deterministically decided in linear time (with respect to* nm*) by a uniform family of P systems with active membranes with two polarizations and rules of the forms* (a_{gb}), (c_{np1}), (c_{gny}), *and* (e_{gp0}).

We do not present the proof here. It can be found in [60].

Remarks and other variants

Some definitions of decisional P systems require that the result is ejected into the environment only in the last step of the computation. Our construction can be easily adjusted to fulfill this property by remembering, in the objects $e_{i,j}$, also the number l of times the number of steps the membrane had polarization 1 during the checking phase, and then "keeping them busy" for $2(mn - l)$ steps. Then, all elementary membranes with positive answers will stop evolving at the same time by sending `yes` into the skin, and those with negative answers will stop evolving earlier.

In the construction given above, the time for giving a positive answer is actually bounded by $2K + 4n + 3m + 4$, where K is the number of occurrences of the variables in β. Thus, if the size of the problem is given as (n, m, K), then (adjusting the counter in the skin) the time can be made at most $2K + 4n + 3m + 5$.

On the other hand, we do not believe that the rule sending object `yes` into the skin can be made independent of the polarization; otherwise, multiple answers are given and the halting time is no longer polynomial. This can easily be avoided for the price of using membrane dissolution (rules of type (d_{gp})) and one more membrane: a copy of the "witness" of the positive result dissolves the middle membrane, releasing a unique object `yes` into the skin, otherwise object `no` is ejected to the skin, as it was done in the proof of Theorem 9 in [108].

Finally, we mention alternative variants of restrictions:

Using the generation phase similar to that from the proof of Theorem 4.4 and making relevant adjustments to the global control, one can quite easily replace the rules of type (e_{gp0}) by rules of type (e_{gp2}).

By replacing the rule $[\ z\ ]_2 \to [\ \ ]_2^{\neg}\ z$ by $[\ z\ ] \to [\ \ ]^{\neg}\ o$, one can remove type (c_{np1}) for the price of introducing type (c_{gp1}).

Corollary A4.1 *For* $\mathbf{t} \in \{n, g\}$ *and* $\mathbf{k} \in \{0, 2\}$, $\mathtt{SAT}(n, m)$ *can be decided in linear time (with respect to* nm*) by a uniform family of P systems with active membranes with two polarizations and rules of the forms* (a_{gb}), $(c_{\mathbf{t}p1})$, *and* $(e_{gp\mathbf{k}})$.

A5. Computing the Permanent

We now present the proof of Theorem 4.5: The problem of computing a permanent of a binary matrix is solvable in polynomial time by a uniform family of deterministic P systems with active membranes with two polarizations and rules of types $(a), (c), (e)$. *Proof.* Let $A = (a_{i,j})$ be an $n \times n$ matrix. We define $N = \lceil \log_2(n) \rceil$, and $n' = 2^N < 2n$ is the least power of two not smaller then n. The input alphabet is $\Sigma(n) = \{\langle i, j\rangle \mid 1 \leq i \leq n,\ 1 \leq j \leq n\}$, and the matrix A is given as a multiset $w(A)$ containing for every element $a_{i,j} = 1$ of the matrix one symbol $\langle i, j\rangle$. Let the output alphabet be $T = \{o\}$, we will present a P system $\Pi(n)$ giving $o^{perm(A)}$ as the result when given input $w(A)$ in region $i_{\Pi(n)} = 2$.

$$\begin{aligned}
\Pi(n) &= \big(O, T, H, E, \mu, w_1, w_2, R, 1\big),\\
O &= \Sigma(n) \cup T \cup \{c\} \cup \{d_i, a_i \mid 0 \leq i \leq Nn\} \cup \{D_i \mid 0 \leq i \leq n+1\}\\
&\cup \{\langle i, j, k, l\rangle \mid 0 \leq i \leq Nn-1,\ 0 \leq j \leq n-1,\ 0 \leq k \leq Nn-1,\\
&\quad 0 \leq l \leq n'-1\},\\
\mu &= [\,[\,\,]_2^0\,]_1^0,\ H = \{1, 2\},\ E = \{0, 1\},\\
w_1 &= \lambda,\ w_2 = d_0.
\end{aligned}$$

and the rules are presented and explained below.

A1 $[\ \langle i, j\rangle \to \langle Ni-1, j-1, Nn-1, 0\rangle\ ]_2^0$, $1 \leq i \leq n$, $1 \leq j \leq n$

Preparation of the input objects: tuple representation. Informal meaning of the tuple components is 1) number of steps remaining until row i is processed, 2) column number, starting from 0, 3) number of steps remaining until all rows are processed, 4) will be used for memorizing the chosen column.

A2 $[\ d_i\ ]_2^e \to [\ d_{i+1}\ ]_2^0[\ d_{i+1}\ ]_2^1$, $0 \leq i \leq Nn-1$, $e \in E$

Division of the elementary membrane for Nn times.

A3 $[\ \langle i, j, k, l\rangle \to \langle i-1, j, k-1, 2l+e\rangle\ ]_2^e$,
$0 \leq i \leq Nn-1$, i is not divisible by N,
$0 \leq j \leq n-1$, $1 \leq k \leq Nn-1$, $0 \leq l \leq (n-1-e)/2$, $e \in E$

For i times, during $N-1$ steps input objects corresponding to row i memorize the polarization history. The binary representation of the chosen column for the current row corresponds to the history of membrane polarizations during N steps.

A4 $[\ \langle i,j,k,l\rangle \to \lambda\ ]_2^e$,
$0 \le i \le Nn-1$, $0 \le j \le n-1$, $1 \le k \le Nn-1$,
$(n-1-e)/2 \le l \le n'/2-1$, $e \in E$

Erase all input objects if the chosen column is invalid, i.e., its number exceeds $n-1$.

A5 $[\ \langle i,j,k,l\rangle \to \langle i-1,j,k-1,0\rangle\ ]_2^e$,
$1 \le i \le Nn-1$, $0 \le j \le n-1$, $j \ne 2l+e$,
$0 \le k \le Nn-1$, $0 \le l \le (n-1-e)/2$, $e \in E$

If element's row is not reached and element's column is not chosen, proceed to the next row.

A6 $[\ \langle i,j,k,l\rangle \to \lambda\ ]_2^e$,
$1 \le i \le Nn-1$, $0 \le j \le n-1$, $j = 2l+e$,
$0 \le k \le Nn-1$, $0 \le l \le (n-1-e)/2$, $e \in E$

Erase the chosen column, except the chosen element.

A7 $[\ \langle 0,j,k,l\rangle \to \lambda\ ]_2^e$,
$0 \le j \le n-1$, $j \ne 2l+e$,
$0 \le k \le Nn-1$, $0 \le l \le (n-1-e)/2$, $e \in E$

Erase the chosen row, except the chosen element.

A8 $[\ \langle 0,j,k,l\rangle \to a_{k-1}\ ]_2^e$,
$0 \le j \le n-1$, $j = 2l+e$,
$0 \le k \le Nn-1$, $0 \le l \le (n-1-e)/2$, $e \in E$

If chosen element is present (i.e., it has value 1 and its column has not been chosen before), produce object a_{k-1}.

A9 $[\ a_k \to a_{k-1}\ ]_2^e$, $1 \le k \le Nn-1$, $e \in E$

Objects a_k wait until all rows are processed. Then a membrane represents a solution if n copies of a_0 are present.

B1 $[\ d_{Nn} \to D_{1-e}c^{n+e}\ ]_2^e$, $e \in E$

If polarization is 0, produce n copies of object c and a counter D_1. Otherwise, produce one extra copy of c and set the counter to D_0; this will reduce to the previous case in one extra step.

B2 $[\ c\ ]_2^1 \to [\ \ ]_2^0 c$

B3 $[\ a_0\ ]_2^0 \to [\ \ ]_2^1 a_0$

B4 $[\ D_i \to D_{i+1}\ ]_2^1$, $0 \le i \le n$

Each object a_0 changes polarization to 1, the counter D_i counts this, and then object c resets the polarization to 0.

B5 $[\ D_{n+1}\ ]_2^1 \to [\ \ ]_2^0 o$

If there are n chosen elements with value 1, send one object o out.

The system is deterministic. Indeed, for any polarization and any object (other than d_i, $i < Nn$, c, a_0 or D_{n+1}), there exist at most one rule of type (a) and no other associated rules. As for the objects in parentheses above, they have no rules of type (a) associated with them and they cause a well-observed deterministic behavior of the system: division rules are applied during the first Nn steps; then, depending on the polarization, symbols a_0 or c are sent out; finally, wherever D_{n+1} is produced, it is sent out.

The system computes the permanent of a matrix in at most $n(2+N)+1 = O(n \log n)$ steps. Indeed, first Nn steps correspond to membrane divisions corresponding to finding all permutations of S_n, see Definition 4.1, while the following steps correspond to counting the number of non-zero entries of the matrix associated to these permutations (there are at most $2n+1$ of them since the system counts to at most n and each count takes two steps; one extra step may be needed for a technical reasons: to reset to 0 the polarization of membranes that had polarization 1 after the first Nn steps). □

A6. Minimal parallelism - 6 polarizations

We present the proof of Theorem 4.8: A uniform family of confluent P systems with rules $(a), (c), (e_0)$ working in minimally parallel way can solve SAT with six polarizations in $O(l(m+n))$ number of steps.

Proof. The strategy used in the construction below is similar to that of the previous theorem. However, since the application of the evolution rules no longer changes the polarization of the membrane, the control symbols $d_{i,k}$, $t_{i,k}$, $f_{i,k}$ no longer "operate" in polarization 0, but rather in polarization that toggles between 0 (for even k) and 5 (for odd k), to prevent multiple applications of evolution rules in a row in the same membrane. Moreover, the input objects are actually allowed to evolve in parallel (and the degree of parallelism is chosen non-deterministically), but in the end of both halves of a cycle it is possible to count the number of extra objects produced, to make sure that all l objects have been processed.

For the same propositional formula

$$\begin{aligned} \beta &= C_1 \vee \cdots \vee C_m, \\ C_i &= y_{i,1} \wedge \cdots \wedge y_{i,l_i},\ 1 \leq i \leq m, \text{ where} \\ y_{i,k} &\in \{x_j, \neg x_j \mid 1 \leq j \leq n\},\ 1 \leq i \leq m, 1 \leq k \leq l_i, \\ l &= \sum_{i=1}^{m} l_i. \end{aligned}$$

and the same encoding of the instance of β in the alphabet $\Sigma(\langle n, m, l\rangle)$ by multisets X, X',

$$\begin{aligned} \Sigma(\langle n, m, l\rangle) &= \{v_{j,i,1,s} \mid 1 \leq j \leq m,\ 1 \leq i \leq n,\ 1 \leq s \leq 2\}, \\ X &= \{(v_{j,i,1,1}, 1) \mid x_i \in \{y_{j,k} \mid 1 \leq k \leq l_j\},\ 1 \leq j \leq m,\ 1 \leq i \leq n\}, \\ X' &= \{(v_{j,i,1,2}, 1) \mid \neg x_i \in \{y_{j,k} \mid 1 \leq k \leq l_j\},\ 1 \leq j \leq m,\ 1 \leq i \leq n\}. \end{aligned}$$

We construct the following P system:

$$\begin{aligned} \Pi(\langle n, m, l\rangle) &= (O, H, E, [\,[\,\,]_2^0[\,\,]_3^0\,]_1^0, w_1, w_2, w_3, R), \text{ with} \\ O &= \{v_{j,i,k,s} \mid 1 \leq j \leq m,\ 1 \leq i \leq n,\ 1 \leq k \leq m+n+1,\ 1 \leq s \leq 4\} \\ &\cup \{d_{i,k} \mid 1 \leq i \leq m+n+1,\ 1 \leq k \leq 2l\} \cup \{t_{i,k}, f_{i,k} \mid 1 \leq i \leq n,\ 1 \leq k \leq l\} \\ &\cup \{d_i \mid 1 \leq i \leq m+n+1\} \cup \{S, Z, \texttt{yes}, \texttt{no}\} \\ &\cup \{z_k \mid 1 \leq k \leq (4l+3)n + m(4l+1) + 2\} \\ &\cup \{o_{i,j} \mid 0 \leq i \leq 5,\ 0 \leq j \leq 5\} \\ w_1 &= \lambda,\ w_2 = d_1,\ w_3 = z_0,\ H = \{1,2,3\},\ E = \{0,1,2,3,4,5\}, \end{aligned}$$

and the rules are listed below. The computation stages are the same as in the previous proof.

1. Producing 2^n membranes corresponding to the possible variables assignments; selecting satisfied clauses (groups A and C).

2. Checking whether all clauses are satisfied (groups B and D).

3. Generating the answer and sending it to the environment. (groups E and F).

Stage 1 consists of n cycles and stage 2 consists of m cycles. Each cycle's aim is to process all l objects, i.e., each object counts the number of cycles completed, and in the first stage the clauses are evaluated while in the second stage the presence of each clause is checked.

A cycle consists of marking (setting the last index to 3 or 4) all l objects one by one while performing the necessary operation, and then unmarking (setting the last index to 1 or 2) all of them. Marking or unmarking an object generally happens in five steps:

1. the control object produces two "polarization changers",

2. one of them changes the polarization from 0 or 5 to 1, 2 (to mark) or to 3 (to unmark),

3. one of the objects that has not yet been (un)marked is processed, producing a "witness" — yet another "polarization changer",

4. the "witness" switches the polarization to 4,

5. the second "changer" produced in step 1 of this routine changes the polarization to 5 or 0.

Notice, however, that "step" 3 might actually take more than one step (more objects can be (un)marked in parallel, or even in a row, creating a supply of "witnesses"). Step 4 might actually be executed in parallel with the last step of "step" 3 (sending out a previous "witness" while producing more). Finally, "step" 3 might even be skipped if a previous "witness" is already there. What matters is that the whole (un)marking routine takes at most $5l$ steps.

Changing polarization of membrane 2

O1 (change from i to j)
$[\ o_{i,j}\ ]^i \to [\ \ ]^j\ o_{4,5}$, $0 \le i \le 5$, $0 \le j \le 5$

O2 ("witnesses" of D2 are "compatible" with "witnesses" of D1; this does not interfere with the rest of the computation)
$[\ o_{1,4}\ ]^2 \to [\ \ ]^4\ o_{4,5}$

Control objects in membrane 2: select clauses

A1 (for variable i: divide)
$[\ d_i\ ] \to [\ t_{i,0}\ ][\ f_{i,0}\ ]$, $1 \le i \le n$

A2 (process and mark all l objects)
$[\ t_{i,k-1} \to t_{i,k} o_{0,1} o_{4,5}\ ]^0$, $1 \le i \le n$, $1 \le k \le l$, k is odd
$[\ f_{i,k-1} \to f_{i,k} o_{0,2} o_{4,5}\ ]^0$, $1 \le i \le n$, $1 \le k \le l$, k is odd
$[\ t_{i,k-1} \to t_{i,k} o_{5,1} o_{4,0}\ ]^5$, $1 \le i \le n$, $1 \le k \le l$, k is even
$[\ f_{i,k-1} \to f_{i,k} o_{5,2} o_{4,0}\ ]^5$, $1 \le i \le n$, $1 \le k \le l$, k is even

A3 (prepare to unmark objects)
$[\ t_{i,l} \to d_{i,0}\ ]^0$, $1 \le i \le n$, if l is even
$[\ f_{i,l} \to d_{i,0}\ ]^0$, $1 \le i \le n$, if l is even
$[\ t_{i,l} \to d_{i,0}o_{5,0}\ ]^5$, $1 \le i \le n$, if l is odd
$[\ f_{i,l} \to d_{i,0}o_{5,0}\ ]^5$, $1 \le i \le n$, if l is odd

A4 (unmark all l objects)
$[\ d_{i,k-1} \to d_{i,k}o_{0,3}o_{4,5}\ ]^0$, $1 \le i \le n$, $1 \le k \le l$, k is odd
$[\ d_{i,k-1} \to d_{i,k}o_{5,3}o_{4,0}\ ]^0$, $1 \le i \le n$, $1 \le k \le l$, k is even

A5 (switch to the next variable)
$[\ d_{i,l} \to d_{i+1}\ ]^0$, $1 \le i \le n$, if l is even
$[\ d_{i,l} \to d_{i+1}o_{5,0}\ ]^5$, $1 \le i \le n$, if l is odd

Control objects in membrane 2: check clauses

B1 (test if clause i is satisfied)
$[\ d_{n+i} \to d_{n+i,1}o_{0,2}o_{4,5}\ ]^0$, $1 \le i \le m$

B2 (process and mark the other $l-1$ objects)
$[\ d_{n+i,k-1} \to d_{n+i,k}o_{0,1}o_{4,5}\ ]^0$, $1 \le i \le m$, $1 \le k \le l$, k is odd
$[\ d_{n+i,k-1} \to d_{n+i,k}o_{5,1}o_{4,0}\ ]^0$, $1 \le i \le m$, $1 \le k \le l$, k is even

B3 (unmark all l objects)
$[\ d_{n+i,l+k-1} \to d_{n+i,l+k}o_{0,3}o_{4,5}\ ]^0$, $1 \le i \le m$, $1 \le k \le l$, $l+k$ is odd
$[\ d_{n+i,l+k-1} \to d_{n+i,l+k}o_{5,3}o_{4,0}\ ]^0$, $1 \le i \le m$, $1 \le k \le l$, $l+k$ is odd

B4 (switch to the next clause)
$[\ d_{n+i,2l} \to d_{n+i+1}\ ]^0$, $1 \le i \le m$

B5 (send a positive answer)
$[\ d_{m+n+1}\ ]^0 \to [\ \ ]^0 S$

Input objects in membrane 2: select clauses

C1 (mark an object)
$[\ v_{j,i,k,s} \to v_{j,i,k+1,s+2}o_{p,4}\ ]^p$,
$1 \le i \le m$, $1 \le j \le n$, $1 \le k \le m$, $k \ne m$, $1 \le s \le 2$, $1 \le p \le 2$

C2 (a true variable present without negation or a false variable present with negation satisfies the clause)
$[\ v_{j,i,i,s} \to v_{j,i,i+1,3}o_{s,4}\ ]^s$, $1 \le i \le m$, $1 \le j \le n$, $1 \le s \le 2$

C3 (a true variable present with negation or a false variable present without negation does not satisfy the clause)
$[\ v_{j,i,i,3-s} \to v_{j,i,i+1,4}o_{s,4}\ ]^s$, $1 \le i \le m$, $1 \le j \le n$, $1 \le s \le 2$

C4 (unmark an object)
$[\ v_{j,i,k,s+2} \to v_{j,i,k,s}o_{3,4}\ ]^3$,
$1 \le i \le m$, $1 \le j \le n$, $2 \le k \le m+1$, $1 \le s \le 2$

Input objects in membrane 2: check clauses

D1 (check if the clause is satisfied at least by one variable)
$[\ v_{j,i,m+j,1} \to v_{j,i,k+1,3} o_{1,4}\ ]^2$, $1 \le i \le m$, $1 \le j \le n$, $1 \le s \le 2$

D2 (mark an object)
$[\ v_{j,i,m+k,s} \to v_{j,i,k+1,s+2} o_{1,4}\ ]^1$,
$1 \le i \le m$, $1 \le j \le n$, $1 \le k \le n$, $1 \le s \le 2$

D3 (unmark an object)
$[\ v_{j,i,m+k,s+2} \to v_{j,i,k,s} o_{3,4}\ ]^3$,
$1 \le i \le m$, $1 \le j \le n$, $2 \le k \le n+1$, $1 \le s \le 2$

Control objects in membrane 3

E1 (count)
$[\ z_{k-1} \to z_k\ ]^0$, $1 \le k \le N = (10l+5)n + m(10l+1) + 2$

E2 (send time-out object)
$[\ z_N\ ]^0 \to [\ \ ]^0 Z$

Control objects in the skin membrane

F1 (the first positive result sends the answer)
$[\ S\ ]^0 \to [\ \ ]^1$`yes`

F2 (without the positive result, the time-out sends the negative answer)
$[\ Z\ ]^0 \to [\ \ ]^0$`no`

Let us now explain how the system works in more details. The control objects keep track of the number of cycles completed, whether marking or unmarking takes place, as well as the number of objects already (un)marked. Moreover, the control object is responsible to pass the "right" information to the objects via polarization: in stage 1, by generating $o_{0,1}$ or $o_{5,1}$ if the variable is true, and $o_{0,2}$ or $o_{5,2}$ if the variable is false; in stage 2, $o_{0,1}$ or $o_{5,1}$ if the clause is already found, and $o_{0,2}$ or $o_{5,2}$ if the clause is being checked for.

During the first stage, an object $v_{j,i,1,s}$ is transformed into $v_{j,i,n+1,t}$, where $t = 1$ if variable x_j satisfies clause C_i, or $t = 2$ if not. The change of the last index from s to t happens when the third index is equal to i. The control object d_1 is transformed into d_{n+1}. Stage 1 takes at most $(10l+5)n$ steps (at most $(10l+3)n$ in the case when l is even).

If some clause is not satisfied, then the computation in the corresponding membrane is "stuck" with polarization 2. Otherwise, during the second stage an object $v_{j,i,n+1,t}$ is transformed into $v_{j,i,n+m+1,t}$, while the control object d_{n+1} becomes d_{m+n+1}. Stage 2 takes at most $m(10l+1)$ steps, plus one extra step to send objects S to skin, if any.

After stage 2 is completed, one copy of S, if any, is sent out as `yes`, changing the polarization of the skin membrane. After this time has passed, object Z comes to the skin from region 3. If the polarization of the skin remained 0, Z is sent out as `no`. □

A7. Sequential UREM P systems

Sequential Mode with Priorities The following theorem establishes computational completeness for P systems with unit rules and energy assigned to membranes, when working in the sequential mode with priorities on the rules:

Theorem A7.1 *Each partial recursive function* $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta \quad (\alpha > 0,\ \beta > 0)$ *can be computed by a P system with unit rules and energy assigned to membranes with (at most)* $\max\{\alpha, \beta\} + 3$ *membranes.*

The proof of this theorem can be found in [62].

When taking $\beta = 0$ in the preceding proof, we get the accepting variant of P systems with unit rules and energy assigned to membranes:

Corollary A7.1 *For any* $L \in PsRE\,(\alpha)$ *there exists a P system with unit rules and energy assigned to membranes with (at most)* $(\alpha + 3)$ *membranes accepting* L.

The proof uses the constructions from the previous theorem; it can be found in [62].

As can immediately be seen from the constructions given in the proofs of Theorem A7.1 and Corollary A7.1, the simulation of a deterministic register machine by a P systems with unit rules and energy assigned to membranes can be carried out in a deterministic way, i.e., for a given input only one computation (halting or not) exists.

We now turn to the non-deterministic case of generating vectors of non-negative integers: when taking $\alpha = 0$ in the proof of Theorem A7.1 and the non-deterministic variant of ADD-instructions, we get the generative variant of P systems with unit rules and energy assigned to membranes:

Corollary A7.2 *For any* $L \in PsRE\,(\beta)$ *there exists a P system with unit rules and energy assigned to membranes with (at most)* $(\beta + 3)$ *membranes generating* L.

The proof uses the constructions from the previous theorem; it can be found in [62].

Sequential Mode without Priorities When omitting the priority feature, we do not get systems with computational completeness anymore. Let $PsPE_*(unit)$ denote the family of sets of Parikh vectors generated by P systems with unit rules and energy assigned to membranes without priorities and with an arbitrary number of membranes. The following two lemmas prove that $PsPE_*(unit) = PsMAT$, i.e., we get a characterization of $PsMAT$ by the new family $PsPE_*(unit)$.

Lemma A7.1 $PsPE_*(unit) \supseteq PsMAT$.

This result is proved by simulating matrix grammars. The proof is rather long; it can be found in [62].

Lemma A7.2 $PsPE_* (unit) \subseteq PsMAT$.

This result is proved by simulating UREM P systems by matrix grammars. The proof is rather long; it can be found in [62].

If we now combine the two previous lemmas we get the following characterization of $PsMAT$:

Theorem A7.2 $PsPE_*(unit) = PsMAT$.

Due to the construction in Lemma A7.1 we not only have obtained a characterization of $PsMAT$ by P systems with unit rules and energy assigned to membranes but also a normal form for this kind of P systems, i.e., only one symbol moving through a membrane structure is already sufficient (which of course is the minimal resource needed to obtain reasonable results). Moreover, without giving a proof we should like to mention that $PsPE_*(unit)$ also equals the family of sets of Parikh vectors accepted by P systems with unit rules and energy assigned to membranes without priorities and with an arbitrary number of membranes.

The results obtained so far in this section are already optimal with respect to the size of the multisets transported through a membrane, as in all proofs we needed only one object to be present in the system. Yet the optimal numbers of membranes necessary for obtaining computational completeness or for characterizing $PsMAT$ still remain open problems (although we conjecture that the number of membranes needed in the universality results is already optimal).

A8. List of selected results in formulas

$$LOP_*(ncoo, tar) = LOP_1(ncoo, out) = L_t(CF) \tag{A8.1}$$

$$REG \bullet \texttt{Perm}(REG) \subseteq LOP(ncoo, tar) \subseteq CS \cap SLIN \cap \mathbf{P} \tag{A8.2}$$

$$N_aDOP_1^{sequ}(ncoo, pro_{1,1}, inh_{1,1}) = NRE \tag{A8.3}$$

$$NFIN \cup coNFIN = N_aDOP_1(ncoo, (pro_{*,*}, inh_{*,*})_*, pri) \tag{A8.4}$$

$$N_aDOP_1^{asyn}(ncoo, (pro_{*,*}, inh_{*,*})_*, pri) \tag{A8.5}$$

$$NFIN \cup coNFIN = N_aDOP_1(ncoo, pro_{1,*}) = N_aDOP_1(ncoo, pro_{1,*}) \tag{A8.6}$$

$$N_aDOP_1^{asyn}(ncoo, pro_{1,*}, inh_{1,*}) \tag{A8.7}$$

$$NRMR(coo, pri)_T = NRMR(coo, inh)_T = NRE \tag{A8.8}$$

$$NR_sMR(coo)_T = \{\emptyset\} \cup \{\{n\} \mid n \in \mathbb{N}\} \tag{A8.9}$$

$$NR_sMR(coo, pri)_T = NRE \tag{A8.10}$$

$$N_aD_sMR(coo, pri)_T = NRE \tag{A8.11}$$

$$N_aD_sMR(coo) = \{\emptyset, \mathbb{N}\} \cup \{\{k \mid 0 \leq k \leq n\} \mid n \in \mathbb{N}\} \tag{A8.12}$$

$$N_aDMR(coo) = \{\emptyset, \mathbb{N}\} \cup \{\{k \mid 0 \leq k \leq n\} \mid n \in \mathbb{N}\} \tag{A8.13}$$

$$NROP_1(coo, pri)_T = NROP_1(coo, pri)_T = NRE \tag{A8.14}$$

$$NR_sOP_1(coo)_T = \{\{n\} \mid n \in \mathbb{N}\} \tag{A8.15}$$

$$NR_sOP_1(coo, pri)_T = NRE \tag{A8.16}$$

$$N_aD_sMR(coo, pri)_T = NRE \tag{A8.17}$$

$$N_aD_sOP_1(coo) = \{\emptyset, \mathbb{N}\} \cup \{\{k \mid 0 \leq k \leq n\} \mid n \in \mathbb{N}\} \tag{A8.18}$$

$$N_aD_sOP_1(coo, pro, inh) = NRE \tag{A8.19}$$

$$LO_{1,2,*}P_{1,*,2}(ncoo, tar, mcre, \delta) = RE \tag{A8.20}$$

$$DPs_aOP_{1,*,2}(ncoo, tar, mcre, \delta) = PsRE \tag{A8.21}$$

$$L(m)O_{1,2,10+m}P_{2,*,*}(ncoo, tar, mcre, \delta) = RE(m) \tag{A8.22}$$

$$L(m)O_{1,2,11+m}P_{1,*,*}(ncoo, tar, mcre, \delta) = RE(m) \tag{A8.23}$$

$$NDOP_1(23anti) = NDMR_1(23coo) = NRE \tag{A8.24}$$

$$NFIN_1 \cup \bigcup_{k=0}^{5}(N_kFIN_k \cup N_kREG_k) \cup N_6RE \subseteq NOP_1(sym_3) \tag{A8.25}$$

$$NFIN_0 \cup NFIN_1 \cup N_1REG_1 \cup N_2RE \subseteq NOP_1(sym_4) \tag{A8.26}$$

$$NFIN \cup N_1REG \cup N_2RE \subseteq NOP_1(sym_*) \subseteq NFIN \cup N_1RE \tag{A8.27}$$

$$SEG_1 \cup SEG_2 \subseteq NOP_1^{sequ}(sym_2) \subseteq NFIN \tag{A8.28}$$

$$NFIN_1 \cup \bigcup_{k=0}^{\infty}(N_kFIN_k \cup N_kREG_k) \subseteq NOP_1^{sequ}(sym_3) \tag{A8.29}$$

$$NOP_1^{sequ}(sym_*) = NFIN \cup N_1REG \tag{A8.30}$$
$$LO_{1,3/*,*}P_{4,*,3}(active_1, a, b, c, d, e) = RE \tag{A8.31}$$
$$LO_{1,3/*,*}P_{7,*,2}(active_1, a, b, c, d, e) = RE \tag{A8.32}$$
$$DPs_aOP_{1,1,1}(active_2, a, c) = PsRE \tag{A8.33}$$
$$NP \subseteq co - NP \subseteq PMC_{OP(active_2,a,c,e)} \tag{A8.34}$$
$$PSPACE \subseteq PMC_{OP(active_1,a,c,d,e,f)} \tag{A8.35}$$
$$NP \subseteq co - NP \subseteq PMC_{OP(active_4,a_s'',c_0,e_0)} \tag{A8.36}$$
$$NP \subseteq co - NP \subseteq PMC_{OP(active_6,a,c,e_0)} \tag{A8.37}$$
$$PsOP_*^{sequ}(energy_*) = PsMAT \tag{A8.38}$$
$$Ps(m)OP_{m+6}(energy_*) = Ps(m)RE \tag{A8.39}$$
$$PsStP_*(ins_1^{0,0}, del_1^{0,0}) = PsMAT \tag{A8.40}$$
$$PsSP_*(ins_1^{0,0} < del_1^{0,0}) = PsRE \tag{A8.41}$$
$$LSP_*(ins_1^{0,1} < del_1^{0,0}) = LSP_*(ins_1^{1,0} < del_1^{0,0}) = RE \tag{A8.42}$$
$$LSP_*(ins_1^{0,0} < del_1^{1,0}) = LSP_*(ins_1^{0,0} < del_1^{0,1}) = RE \tag{A8.43}$$
$$LSP_*(ins_1^{0,0} < del_2^{0,0}) = RE \tag{A8.44}$$
$$LSP_*(e - ins_1^{0,0} < l - del_1^{0,0}) = RE \tag{A8.45}$$
$$REG \subseteq ELSP_*(r - ins_1^{0,0}, r - del_1^{0,0}) \tag{A8.46}$$
$$REG \subseteq ELSP_*(l - ins_1^{0,0}, l - del_1^{0,0}) \tag{A8.47}$$
$$NOP_{47}(polym_d(coo)) = NRE \tag{A8.48}$$

List of Tables

List of Figures

DECLARATION OF ASSUMING RESPONSIBILITY

Subsemnatul, declar pe răspundere personală că materialele prezentate în teza de doctorat sunt rezultatul propriilor cercetări şi realizări ştiinţifice. Conştientizez că, în caz contrar, urmează să suport consecinţele în conformitate cu legislaţia în vigoare.

Alhazov Artiom

.....

Dată: .....

CURRICULUM VITAE

Numele: Alhazov
Prenumele: Artiom
Data şi locul naşterii: 11 octombrie 1979, Chişinău, RM
Cetăţenia: Republica Moldova
Studii superioare:
1996–2001, Universitatea de Stat din Moldova, Chişinău,
Specialitatea: Matematica şi Informatica
Studii de doctorat:
2001–2004, Institutul de Matematică şi Informatică
al A.Ş.M., Chişinău
Specialitatea: 01.05.01 – Bazele teoretice ale informaticii;
programarea calculatoarelor
2002–2006, Universitatea "Rovira i Virgili", Tarragona, Spania
Diploma nr. 784408 Seria 1-BC , eliberată la 05.05.2006
echivalată la 20.12.2007 cu gradul de doctor în informatică,
în specialitatea: 01.05.01 - Bazele teoretice ale informaticii, programarea calculatoarelor.
Studii de postdoctorat: 2007, Åbo Akademi, Turku, Finlanda, 12 luni
2008–2010, Hiroshima University, Higashi Hiroshima, Japonia, 24 luni
2011-2012, Università degli Studi di Milano-Bicocca, Milano, Italia, 18 luni
Domenii de interes ştiinţific: Informatica teoretică, Teoria limbajelor formale,
Calcule biomoleculare, Matematica
Activitatea profesională: 2005-prezent, laboratorul Sisteme de programare,
Institutul de Matematică şi Informatică al A.Ş.M.
(inginer programator, cercetător ştiinţific, cercetător ştiinţific superior)
Participari in proiecte ştiinţifice naţionale si internaţionale:
Indicate în "acknowledgements" (p. 4)
şi în "acknowledgements" în [5],
`http://psystems.disco.unimib.it/download/Alhazovthesis.zip`
Participări la foruri ştiinţifice: peste 35 (co-autor),
peste 20 (participat personal, peste hotare)
Lucrari stiintifice publicate: peste 160,
`http://www.aartiom.50webs.com/pub_aa.html`,
`http://www.math.md/people/alhazov-artiom/`
Premii principale: 2006, Premiul Naţional pentru Tineret
în Domeniile Ştiinţei, Tehnicii, Literaturii şi Artelor,
2010, Premiul Academiei de Ştiinţe a Moldovei
pentru realizări ştiinţifice ale tinerilor cercetători,
2012, Premiu al Institutului de Matematică şi Informatică pentru tineri savanţi.
Cunoaşterea limbilor: rusa, româna, engleza (fluent),
spaniola, italiana (bine), catalana (cu dicţionarul)
Date de contact: tel. 022727483, email: artiom (at) math.md