Distributed Algorithms in Membrane Systems

Yun-Bum Kim

under the supervision of

Dr. Radu Nicolescu and Dr. Michael J. Dinneen

A thesis submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy in Computer Science,
The University of Auckland, 2012.

Abstract

Membrane systems or P systems are distributed and parallel computing models, inspired by the interconnected structure and function of living cells.

This thesis investigates the adequacy of synchronous membrane systems for modelling a set of fundamental distributed algorithms. This thesis provides a coherent approach to specify and analyse a certain class of problems related to distributed algorithms. This thesis presents a set of P algorithms (i.e. algorithms that are implemented using membrane systems) that are *comparable* (i.e. not necessarily better, but not much worse) to the corresponding distributed algorithms, expressed as high-level pseudocodes, with respect to time complexity and program-size complexity.

This thesis introduces a new membrane system model. This new model incorporates features that allow for the reduction of inherent non-determinism (such as allowing for a total order priority on rule sets), allowing a better control of the rule execution strategy of the considered distributed algorithms, which are complex and require clear and well defined threads of execution. This thesis proves that this new model is Turing complete and universal.

This thesis presents several broadcast- and echo-based P algorithms, as modular building blocks, for building more complex algorithms. These broadcast- and echo-based P algorithms have time complexity of $e + k$ and $2e + k$, respectively, where e is the eccentricity of a source and k is an integer constant.

Next, this thesis presents two solutions to the *firing squad synchronization problem* (FSSP). The first FSSP solution synchronizes digraph-structured P systems in $3e+11$ steps, where e is the eccentricity of the general. The second FSSP solution synchronizes tree-structured P systems in $h + 2r + 5$ steps, where h and r are the height and radius of the tree, respectively.

Finally, this thesis presents solutions to two fundamental problems in graph theory: the *edge-* and *node-disjoint paths problems.* For a given membrane system with n cells and m links, these solutions find sets of edge- and node-disjoint paths of maximum cardinality in $O(mn)$ steps. These solutions are the first P algorithms for solving the edge- and node-disjoint paths problems in membrane systems.

Acknowledgements

I would like to thank my supervisors, Dr. Radu Nicolescu and Dr. Michael J. Dinneen, for providing me the opportunity to work under their guidance, and for their constant advice, encouragement and support throughout my PhD program.

I am grateful to Associate Professor John Morris for his advice on general guidelines on writing research papers and his reviews and comments.

I would like to thank my research colleagues during the period of my study for their friendship. I would like to also thank the Department of Computer Science of the University of Auckland for providing all the facilities necessary to conduct my research.

Finally, I would like to thank my family. This work would not have been possible without their patience, understanding and unconditional support.

Contents

1 Introduction **1**

1.1 Membrane systems 1

1.2 Motivation 3

1.3 Thesis outline 5

1.4 Publications 9

2 Membrane Systems **11**

2.1 Preliminaries 12

2.2 Transition P systems 14

2.3 Extensions and variants 18

2.4 Simple P systems 20

2.5 Summary 25

3 Turing completeness of simple P systems **27**

3.1 Register machines 27

3.1.1 Register machine instructions 28

3.1.2 Input data 29

3.1.3 Run-time errors 29

3.1.4 Register machine instructions for GCD algorithm 30

3.2 Universality results 33

3.2.1 The main cell 33
3.2.2 The provider cell 38
3.2.3 Handling of run-time errors 40
3.2.4 Analysis of system Π_M and remarks 41
3.2.5 Constructing system Π_{GCD} using direct approach 43
3.2.6 Construction of system Π'_{GCD} using indirect approach 44
3.3 Summary 49

4 Traversal Algorithms in Membrane Systems 51

4.1 Traversal algorithms 52
4.1.1 Preliminaries 52
4.2 Tree traversal algorithms 55
4.2.1 Algorithm: Broadcast with acknowledgement 56
4.2.2 Algorithm: Echo 60
4.2.3 Algorithm: Tree height 67
4.3 Graph traversal algorithms 73
4.3.1 Algorithm: Number of shortest paths 74
4.3.2 Algorithm: Distance parity 80
4.3.3 Algorithm: Echo for graphs 87
4.3.4 Algorithm: Cell heights 96
4.4 Summary 101

5 The Firing Squad Synchronization Problem 103

5.1 Synchronization 103
5.1.1 Classification of the FSSP 104
5.1.2 Formulation of the FSSP for simple P systems 104

5.1.3 Motivations 105
5.2 Phase-based decomposition of the FSSP 106
5.2.1 Algorithm: Phase II—Decrementing hop-counter 107
5.3 Static FSSP solution for graphs 113
5.3.1 Algorithm: Phase I—Compute general's eccentricity 115
5.3.2 Algorithm: Phase II—Propagation of the order 117
5.3.3 Summary of Phase I and Phase II 118
5.4 Adaptive FSSP solution for trees 121
5.4.1 Algorithm: Phase I—Find a tree centre 123
5.4.2 Algorithm: Phase II—Propagation of the order 129
5.4.3 Summary of Phase I and Phase II 129
5.4.4 Experimental work 132
5.5 Summary 134

6 The Disjoint Paths Problem 139

6.1 Disjoint paths in digraphs 140
6.1.1 Preliminaries 141
6.1.2 Edge-disjoint paths in digraphs 142
6.1.3 Node-disjoint paths in digraphs 145
6.1.4 Pointer management 146
6.1.5 Structural and search digraphs in membrane systems 147
6.2 Edge-disjoint paths solution 148
6.2.1 Phase I: Algorithm—Neighbourhood discovery 152
6.2.2 Phase II: Algorithm—Edge-disjoint paths discovery 157
6.3 Node-disjoint paths solution 161
6.4 Summary 163

7 Conclusions **165**

Bibliography **171**

List of Figures

2.1 A membrane structure represented as a tree and as a Venn diagram . 14

3.1 Symbol exchanges between the main and provider cells in the READ and HALT instructions 40

4.1 A table of graph node attributes 53

4.2 A sample rooted tree used as an algorithm illustration 56

4.3 Propagation of symbols in Algorithm 4.2.1 (Broadcast with acknowledgement) 60

4.4 Propagation of symbols in Algorithm 4.2.2 (Echo) 66

4.5 Propagation of symbols in Algorithm 4.2.3 (Tree height) 72

4.6 A table of membrane system cell attributes 74

4.7 Propagation of symbols in Algorithm 4.3.1 (Number of shortest paths) 77

4.8 Propagation of symbols in Algorithm 4.3.2 (Distance parity) 83

4.9 Propagation of symbols in Algorithm 4.3.3 (Graph echo) 95

4.10 Propagation of symbols in Algorithm 4.3.4 (Cell height) 100

5.1 A rooted tree and a graph used as an FSSP solution illustration . . . 107

5.2 Propagation of symbols in Algorithm 5.2.1 (Decrementing hop-counter) 110

5.3 Propagation of symbols in Algorithm 5.3.1 (Compute general's eccentricity) 119

5.4 Propagation of symbols in Algorithm 5.4.1 (Find a tree centre) 124

5.5 Propagation of symbols in Algorithm 5.4.1 (Find a tree centre) for locating a centre of a tree with two centres 125

5.6 An auxiliary figure for Proposition 5.14. 127

6.1 A maximum set of edge-disjoint paths 141

6.2 Edge- and node-disjoint paths of a graph 141

6.3 Residual digraphs and augmenting paths 143

6.4 The residual digraph of a given digraph 143

6.5 The node-splitting technique 145

6.6 A set of node-disjoint paths of a graph 146

6.7 Three virtual search digraphs for a simple P system 148

List of Tables

3.1 Comparing system Π_{GCD}, system Π'_{GCD} and GCD pseudo-code 50

4.1 Traces of Algorithm 4.2.1 (Broadcast with acknowledgement) 59

4.2 Traces of Algorithm 4.2.2 (Echo) 63

4.3 Traces of Algorithm 4.2.3 (Tree height) 69

4.4 Traces of Algorithm 4.3.1 (Number of shortest paths) 78

4.5 Traces of Algorithm 4.3.2 (Distance parity) 84

4.6 Traces of Algorithm 4.3.3 (Graph echo) 91

4.7 Traces of Algorithm 4.3.4 (Cell height) 98

4.8 Comparing a synchronous distributed broadcast pseudo-code against Algorithms 4.2.1 (Broadcast with acknowledgement) and 4.3.1 (Number of shortest paths) 101

4.9 Comparing a synchronous distributed broadcast pseudo-code against Algorithms 4.3.1 (Number of shortest paths) and 4.3.2 (Distance parity) 101

4.10 Comparing a synchronous distributed echo pseudo-code against Algorithms 4.2.2 (Echo) and 4.2.3 (Tree height) 102

4.11 Comparing a synchronous distributed echo pseudo-code against Algorithms 4.3.3 (Graph echo) and 4.3.4 (Cell heights) 102

5.1 Traces of Algorithm 5.2.1 (Decrementing hop-counter) for a tree . . . 109

5.2 Traces of Algorithm 5.2.1 (Decrementing hop-counter) for a graph . . 110

5.3 Traces of Algorithm 5.3.1 (Compute general's eccentricity) 117

5.4 Traces of Algorithms 5.3.1 and 5.3.2 (Static FSSP solution) 120

5.5 Traces of Algorithms 5.4.1 and 5.4.2 (Adaptive FSSP solution) 131

5.6 Observed reduction in synchronization time of the adaptive FSSP solution over the static FSSP solution 134

5.7 Comparing an FSSP solution of cellular automata against the static FSSP solution of Definition 5.8 137

5.8 Comparing an FSSP solution of cellular automata against the adaptive FSSP solution of Definition 5.13 137

5.9 Comparing a synchronous distributed broadcast pseudo-code against Algorithm 5.2.1 (Decrementing hop-counter) 138

5.10 Comparing a synchronous distributed echo pseudo-code against Algorithm 5.3.1 (Compute general's eccentricity) 138

6.1 A set of neighbour pointer symbols 153

6.2 Traces of Algorithm 6.2.1 (Neighbourhood discovery) 156

6.3 A representation of a maximum set of edge-disjoint paths 158

6.4 A representation of a maximum set of node-disjoint paths 162

7.1 Comparing a synchronous distributed broadcast pseudo-code against Algorithms 4.2.1 (Broadcast with acknowledgement) and 4.3.1 (Number of shortest paths) 166

7.2 Comparing a synchronous distributed broadcast pseudo-code against Algorithms 4.3.1 (Number of shortest paths), 4.3.2 (Distance parity) and 5.2.1 (Decrementing hop-counter) 166

7.3 Comparing a synchronous distributed echo pseudo-code against Algorithms 4.2.2 (Echo) and 4.2.3 (Tree height) 167

7.4 Comparing a synchronous distributed echo pseudo-code against Algorithms 4.3.3 (Graph echo), 4.3.4 (Cell heights) and 5.3.1 (Compute general's eccentricity) 167

7.5 Comparing an FSSP solution of cellular automata against the static FSSP solution of Definition 5.8 168

7.6 Comparing an FSSP solution of cellular automata against the adaptive FSSP solution of Definition 5.13 168

Chapter 1

Introduction

1.1 Membrane systems

Păun pioneered membrane computing [69], which aims to define computing models, called membrane systems (or P systems), that abstract computing ideas from the structure and functioning of a living cell.

Membrane computing is inspired by the observation that a eukaryotic cell is subdivided into several functionally distinct compartments. Further, cells are equipped with complex distribution systems that transport molecules (contained inside a compartment) between compartments. A cell can perform different metabolic activities simultaneously. Each compartment can perform its own specific function, without interference from other cell functions. Thus, important features that can be identified from this observation are:

- *distributed*, in that each compartment carries out its own specific functions,
- *intra-compartment-parallelism*, in that several reactions can be simultaneously carried out in each compartment and
- *inter-compartment-parallelism*, in that several compartments in a cell carry out their functions simultaneously.

These features are incorporated in the design of membrane systems.

A membrane system is a distributed and parallel computing model, which consists of a set of autonomous computing units, called *membranes* or cells. Typically, membranes in a membrane system are arranged in (i) a cell-like hierarchical structure, which can be represented as a rooted tree, or (ii) a tissue-like network structure, which can be represented as a graph or a directed graph. At each given time, each membrane contains a multiset of (alphabet of) *symbols*. Moreover, each membrane is associated with a finite set of *evolution rules* that can (i) transform the multiset of symbols (i.e. consume existing symbols and produce new symbols) and (ii) transfer multisets of symbols to other neighbouring membranes. A *P algorithm* is a set of evolution rules, which is directly *executable* on a membrane system simulator [33]. In principle, such a P algorithm should be automatically *verifiable* by a membrane system verifier [45]—if these become more practical.

Membrane systems evolve via a *non-deterministic maximally parallel* use of evolution rules. Membrane systems are typically *synchronized*, i.e. a global clock sends a signal to all membranes of the system, and in each time unit each membrane uses a maximal multiset of evolution rules. Starting from the initial configuration, a membrane system repeatedly evolves via the non-deterministic maximally parallel use of evolution rules, until it reaches a halting configuration, i.e. a configuration where no membrane can use a rule. Classically, the *result* of a computation is the final multiplicity of symbols in a designated membrane, called the output membrane. In our case, we consider *distributed results*, where each cell may contain a part of the overall solution.

Broadly speaking, research on membrane systems falls into one of the following three areas (as described by Nicolescu [56]). For a comprehensive list, refer to Păun et al.'s survey [70].

- **Theory:** such as computational completeness (universality) [69, 32, 6, 31, 44, 15, 17, 68], complexity classes (e.g. polynomial solutions to NP-hard problems [83, 47]) or relationships with other models (e.g. automata, grammar systems and formal languages).
- **Tools:** including designers, simulators and verifiers [33, 34, 35, 45].
- **Applications:** such as computational biology, economics, ecosystem, linguistics and distributed computing [66, 11, 46, 14, 13, 2].

1.2 Motivation

The goals of this thesis are to: (i) investigate the adequacy of synchronous membrane systems for modelling a set of synchronous distributed algorithms and (ii) obtain a library of fundamental distributed P algorithms. Many studies in membrane systems have typically focused on obtaining theoretical results. However, there have not been enough studies that focus on practical applications of membrane systems, especially on fundamental problems of distributed computing.

To achieve these goals, this thesis presents several P algorithms that are *comparable* (i.e. not necessarily better, but not much worse) to:

- the best-known algorithms, with respect to *time complexity*,
- their corresponding pseudo-code, with respect to *program-size complexity* (i.e. the number of instructions or evolution rules).

This thesis shows that, even at a detailed executable level, P algorithms can still compare favourably against pseudo-codes on the considered criteria.

Solving a distributed problem typically involves the following three phases:

1. Design phase: design a solution in an informal high-level *pseudo-code*. Pseudo-code is typically used to sketch an algorithm or program and it only contains details that are essential for understanding the algorithm or program. However, pseudo-codes are not executable and not formal.

2. Implementation phase: implement the pseudo-code into an *executable* solution, using programming languages. This phase adds many details (that were not considered in the design phase), which make the output solution long, complicated and difficult to understand.

3. Verification or testing phase: use formal models to verify that the executable solution satisfies the requirements and specifications. Many formal models are not directly executable and could be difficult to understand. Software designers use various types of testing, such as regression testing and performance testing.

Membrane systems are formal, executable and can be made comparable to pseudo-codes, with respect to time and program-size complexities. Most membrane system models are Turing complete. Hence, theoretically, there exists a P algorithm that can solve a given computable function. However, we want to look beyond the theoretical results (i.e. the mere existence of algorithms) and focus on finding practical P algorithms that are comparable to the best-known algorithms.

As typically defined, several versions of membrane systems, such as tissue P systems [51] and hyperdag P systems [58], can represent distributed systems with the following characteristics:

- communication is synchronous,
- all messages are transferred reliably, without being modified or lost,
- messages are transferred both one-to-one (i.e. unicast) and one-to-many (i.e. broadcast) manner and
- no collisions of messages that are sent at the same time over the same channel.

This may or may not be enough to efficiently use membrane systems as practical tools for modelling distributed systems and algorithms. This thesis provides substantial evidence to support positive answers to the following questions, provided that we adopt a few *extensions* to the basic membrane system framework, which make it more practical, without extending its theoretical power.

- Is it natural to model distributed systems using membrane systems?
- Can we use membranes to represent autonomous nodes in a distributed algorithm?
- Can we use multiset of symbols to practically model crucial data required by distributed algorithms?
- Can evolution rules play the role of instructions?
- Can we adequately model distributed message passing as multiset symbol transfer?

This thesis is also a first step towards creating a library of fundamental P algorithms that could be useful for building other P algorithms, i.e. providing software support (fundamental algorithms) to distributed systems (membrane systems).

1.3 Thesis outline

Chapter 2: Membrane system models

In the original membrane system [69], cells are arranged in a rooted tree structure and evolution rules are applied in a non-deterministic and maximal manner. These membrane system characteristics may not be suitable for designing deterministic distributed algorithms. Several extended features were introduced, which:

1. generalize the cell structure [51, 44], from rooted trees to directed graphs,
2. provide more dynamic ways of using evolution rules (e.g. rewriting modes [51, 23] and membrane polarization [64]) and
3. reduce the level of non-determinism in using evolution rules (e.g. priorities [63]).

This chapter starts with a survey of a membrane system (called a transition P system [64]) and discusses several extended features. This chapter continues to an introduction of a new membrane system, called a *simple P system*, which incorporates features, such as generalized cell structure, rewriting modes and priorities. Simple P systems will be used to study distributed algorithms in the following chapters.

Chapter 3: Turing completeness of simple P systems

In membrane systems, many studies have focused on obtaining small universal computing devices [6, 31, 44, 17, 68, 15]. Some of these universality results are obtained by constructing membrane systems that can simulate a universal register machine [31, 17]. Moreover, these results have used a register machine model that stores instructions and input data in its registers. In a `READ`-enhanced register machine model [9], the input data, which is not stored in the registers, is accessed using `READ` instructions.

This chapter proves that simple P systems, introduced in Chapter 2, are Turing complete (i.e. can compute anything a Turing machine [53, 42, 74] can compute) and universal (i.e. can simulate an arbitrary Turing machine with input), by showing that simple P systems can simulate all `READ`-enhanced register machines. There are two approaches for obtaining a simple P system, Π, that computes a given function:

1. **Direct approach:** Specify a set of evolution rules, symbols and states of system Π and prove its correctness.

2. **Indirect (or translated) approach:** First obtain a register machine, M (and its register machine instructions), that computes the function. Then, transform register machine M into a functionally equivalent system Π, such that Π can simulate M.

However, in general, using the indirect approach to produce a membrane system could have unwanted consequences. A register machine is a single computing unit that executes a list of instructions in sequence. However, membrane systems consist of several autonomous computing units that simultaneously use evolution rules in parallel. Hence, a membrane system, produced by transforming a register machine, may not use the maximal parallel processing capabilities. Further, this transformation may introduce additional auxiliary cells, symbols or evolution rules, which could increase the time complexity of the system without any benefit. Thus, considering the goals of this thesis, the direct approach is adopted in the succeeding chapters.

Chapter 4: Traversal algorithms in membrane systems

The study of distributed algorithms in membrane systems has been initiated by Ciobanu et al. [13, 14]. They present basic distributed algorithms, such as broadcast and convergecast, in pseudo-code, without explicit evolution rules.

This thesis extends Ciobanu et al.'s research by identifying and removing the following limitations. First, Ciobanu et al.'s distributed algorithms are restricted to tree structures, hence, due to additional challenges present in digraphs, these algorithms are not applicable to general digraph-structured membrane systems (such as tissue P systems [51] and hyperdag P systems [58, 57]). Second, presenting algorithms with pseudo-code (instead of explicit evolution rules) overlooks many of the challenges that are associated in design of the algorithms. Discovering and addressing the membrane system specific features that are used to overcome the challenges could highlight and demonstrate their adequacy. Moreover, we propose and introduce new membrane system features, when the existing features cannot overcome the challenges effectively.

This chapter presents several fundamental traversal algorithms, such as broadcast, echo and structure height, for tree- and digraph-structured simple P systems. Each of these traversal algorithms is completely defined with explicit evolution rules. Conceptually, these algorithms form a library of modular building blocks, used to build the more complex algorithms presented in Chapters 5 and 6. Modularity is a technique that solves a complex problem by: (i) subdividing the problem into simpler sub-problems, (ii) developing and testing the sub-problems separately and (ii) assembling the results of the sub-problems.

Chapter 5: The firing squad synchronization problem

The *firing squad synchronization problem* (FSSP), originally proposed by Myhill in 1957, is one of the most studied problems for cellular automata. The original problem involves constructing a one-dimensional cellular automaton, such that the left-most (or right-most) cell ("general") causes all the other cells ("soldiers") to enter a designated state (firing state), simultaneously and for the first time.

The FSSP has recently been studied in a framework of membrane systems, specifically for tree-structured membrane systems. Bernardini et al. [7] (2008) provided a first deterministic algorithm that synchronizes a *transition P systems with polarization and priority* in $4n + 2h$ steps, where h is the height of the underlying tree structure and n is the number of membranes. Later, Alhazov et al. [1] (2008) provided a deterministic algorithm that synchronizes a *transition P systems with promoters and inhibitors* in $3h + 3$ steps.

In both FSSP solutions, (i) the height of the tree is determined, then (ii) a decrementing counter (initially set to the tree height) is broadcast from the tree root to all tree nodes; the current value of the decrementing counter corresponds to the number of remaining steps before synchronization occurs. Bernardini et al. and Alhazov et al. used a depth-first search (DFS) and a breadth-first search (BFS) to determine the tree height, respectively. The FSSP solution of Alhazov et al. concluded that using a BFS approach (which uses the parallelism available in membrane systems) can provide a more efficient solution (than a DFS approach) to the FSSP.

However, both FSSP solutions did not consider finding a tree centre and sending out the decrementing counter (initially set to the height of this centre) from this centre; finding a centre of a given structure is a common strategy used in cellular automata for solving the FSSP. Further, both FSSP solutions are designed for tree structures, hence may not synchronize digraph-structured membrane systems, such as neural P systems [63] and tissue P systems [51].

This chapter presents three results on membrane systems synchronization [26, 27, 25, 21]. First a deterministic algorithm that synchronizes a digraph-structured simple P system Π in $3e+11$ steps, where e is the eccentricity of the general in the underlying graph of Π, is discussed. Next, two adaptive algorithms that delegate the role of the general towards a digraph centre, are discussed. The first adaptive algorithm synchronizes a tree-structured simple P system Π in $h + 2r + 5$ steps, where h and r are the height and radius of the tree of system Π. The second adaptive algorithm synchronizes a digraph-structured simple P system Π in $e + 2r' + 5$ steps, where r' is the radius of a spanning tree of the underlying graph of system Π and e is the eccentricity of the general in the underlying graph of system Π.

Chapter 6: Disjoint paths problem

Two fundamental problems in graph theory are:

1. The *edge-disjoint paths problem*: find the maximum number of paths from a source node to a target node, that have no edge in common.

2. The *node-disjoint paths problem:* find the maximum number of paths from a source node to a target node that have no other node in common, except the source and target nodes.

Maximum flow algorithms (with edge capacities of one), together with the *augmenting path* technique [75], was used to solve the edge-disjoint paths problem of a given network. Further, an edge-disjoint paths algorithm, together with the node splitting technique [77], was used to solve the node-disjoint paths problem.

This chapter presents native membrane system versions of the edge- and node-disjoint paths problems [22]. For a simple P system with n cells and m edges, the edge- and node-disjoint paths P algorithms presented in this chapter take $O(mn)$ steps. These P algorithms are based on the standard depth-first search maximum flow algorithms (by Ford and Fulkerson [48]) with additional constraints, such as (i) no structural information is available initially and (ii) each membrane system cell has to learn its immediate neighbours.

Chapter 7: Conclusions and future work

This chapter includes some final remarks and possible future work.

1.4 Publications

Journal articles:

1. Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu. Faster synchronization in P systems. Natural Computing, 11:107–115, 2012.

2. Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu. P systems and the Byzantine agreement. Journal of Logic and Algebraic Programming, 79(6):334–349, 2010.

3. Radu Nicolescu, Michael J. Dinneen, and Yun-Bum Kim. Towards structured modelling with hyperdag P systems. International Journal of Computers, Communications and Control, 2:209–222, 2010.

Conference proceedings and revised selected papers:

1. Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu. An adaptive algorithm for P system synchronization. In Marian Gheorghe, Gheorghe Păun, Grzegorz Rozenberg, Arto Salomaa, and Sergey Verlan, editors, Int. Conf. on

Membrane Computing, volume 7184 of Lecture Notes in Computer Science, pages 139–164. Springer, 2011.

2. Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu. Edge- and node-disjoint paths in P systems. Electronic Proceedings in Theoretical Computer Science, 40:121–141, 2010.

3. Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu. A faster P solution for the Byzantine agreement problem. In Marian Gheorghe, Thomas Hinze, and Gheorghe Păun, editors, Conference on Membrane Computing, volume 6501 of Lecture Notes in Computer Science, pages 175–197. Springer-Verlag, Berlin Heidelberg, 2010.

4. Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu. Synchronization in P modules. In Cristian S. Calude, Masami Hagiya, Kenichi Morita, Grzegorz Rozenberg, and Jon Timmis, editors, Unconventional Computation, volume 6079 of Lecture Notes in Computer Science, pages 32–44. Springer-Verlag, Berlin Heidelberg, 2010.

5. Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu. New solutions to the firing squad synchronization problems for neural and hyperdag P systems. Electronic Proceedings in Theoretical Computer Science, 11:107–122, 2009.

6. Radu Nicolescu, Michael J. Dinneen, and Yun-Bum Kim. Discovering the membrane topology of hyperdag P systems. In Gheorghe Păun, Mario J. Pérez-Jiménez, Agustín Riscos-Núñez, Grzegorz Rozenberg, and Arto Salomaa, editors, Workshop on Membrane Computing, volume 5957 of Lecture Notes in Computer Science, pages 410–435. Springer-Verlag, 2009.

Technical report:

1. Michael J. Dinneen and Yun-Bum Kim. A new universality result on P systems. Report CDMTCS-423, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, Auckland, New Zealand, July 2012.

Chapter 2

Membrane Systems

Membrane systems [64] are distributed and parallel computing models, abstracted from the functioning and structure of living cells. Several variant membrane systems were introduced [62, 51, 5, 44], which share the basic components of the original P system, but with conflicting definitions for their extended features. Moreover, the existing features do not seem to provide sufficient flexibility for constructing P algorithms to model various distributed algorithms, due to use of global and fixed multiset rewrite and transfer modes.

This chapter introduces a new membrane system model, called a *simple P system*, which incorporates several extended features that seem useful and necessary for modelling distributed algorithms. Simple P systems are used in the design and specification of distributed algorithms in the following chapters.

This chapter is organized as follows. Section 2.1 presents some basic definitions of graphs, strings and multisets. Section 2.2 presents a formal definition of transition P systems. Section 2.3 presents several extended features that are introduced in membrane systems. Section 2.4 introduces a formal definition of simple P systems (this model is derived from transition P systems and it incorporates several extended features, described in Section 2.3). Section 2.5 provides a summary of the new features of simple P systems.

2.1 Preliminaries

This section covers several key mathematical concepts that are used in this thesis, such as sets, strings, multisets, graphs and trees.

Strings and multisets

A *set* is a group of distinct objects called elements. An *alphabet* is a finite non-empty set with elements called *symbols*. A *string over* alphabet O is a finite sequence of symbols from O. The set of all strings over O is denoted by O^*. The length of a string $x \in O^*$, denoted by $|x|$, is the number of symbols in x. The number of occurrences of a symbol $o \in O$ in a string x over O is denoted by $|x|_o$. The *empty string* is denoted by λ.

A *multiset* is a set with multiplicities associated with its elements; we represent a multiset in the form of a string. The empty multiset is represented by λ. The *size* of a multiset v is denoted by $|v|$. The *multiplicity* of a symbol $o \in O$ in a multiset v is denoted by $|v|_o$. The *multiplicity* of a multiset $u \in O^*$ in a multiset v is denoted by $|v|_u$. Consider two multisets $v \in O^*$ and $w \in O^*$. We say that w is *included* in v, denoted by $w \subseteq v$, if, for all $o \in O$, $|w|_o \leq |v|_o$. The *union* of v and w, denoted by $v \cup w$, is a multiset x, such that, for all $o \in O$, $|x|_o = |v|_o + |w|_o$. The *intersection* of v and w, denoted by $v \cap w$, is a multiset x, such that, for all $o \in O$, $|x|_o = \mathtt{min}\{|v|_o, |w|_o\}$. If $w \subseteq v$, then the *difference* of v and w, denoted by $v - w$, is a multiset x, such that, for all $o \in O$, $|x|_o = |v|_o - |w|_o$. The *product* of v by $k \geq 1$, denoted by $v \otimes k$, is a multiset x, such that, for all $o \in O$, $|x|_o = |v|_o \cdot k$.

A *multiset rewriting rule* is of the form $u \to v$, where $u \in O^*$ and $v \in O^*$ are multisets. Given a multiset $w \in O^*$, a rule $u \to v$ is *applicable*, if $u \subseteq w$. One application of the rule $u \to v$ on multiset w transforms w to multiset w', such that, for all $o \in O$, $|w'|_o = |w|_o - |u|_o + |v|_o$.

Graphs

A (binary) *relation* R over two sets X and Y is a subset of their Cartesian product, $R \subseteq X \times Y$. For $A \subseteq X$ and $B \subseteq Y$, we set $R(A) = \{y \in Y \mid \exists x \in A, (x, y) \in R\}$,

$R^{-1}(B) = \{x \in X \mid \exists y \in B, (x, y) \in R\}$.

A *graph* is an ordered pair (V, E), where V is a finite set of elements called *nodes* (or *vertices*) and E is a set of unordered pairs of V called *edges*. The *order* of a graph (V, E) is $|V|$. The *size* of a graph (V, E) is $|E|$. A *path* of length $n - 1$ is a sequence of n nodes, $v_1, v_2, \ldots, v_n$, such that $\{(v_1, v_2), \ldots, (v_{n-1}, v_n)\} \subseteq E$. A *simple path* is a path that does not repeat any node. A *tree* is a graph in which, for every pair of nodes, there is exactly one simple path between them.

A graph is *connected*, if there exists a path between every pair of nodes. Given a connected graph $G = (V, E)$, the *eccentricity* of a node $v \in V$ is the maximum of the lengths of shortest paths between v and any other node. The *diameter* of a graph is the maximum eccentricity of all nodes in the graph. The *radius* of a graph is the minimum eccentricity of all nodes in the graph. A *central node* of a graph is a node that has its eccentricity equal to the radius of the graph. A *spanning tree* of a connected graph (V, E) is a tree $T = (V, E')$, where $E' \subseteq E$.

Consider graphs $G = (V, E)$ and $H = (V', E')$. An *isomorphism* of G to H is a bijection $\alpha : V \to V'$ matching up the nodes, so that $\{v, w\}$ is an edge of G if and only if $\{\alpha(v), \alpha(w)\}$ is an edge of H. Two graphs G and H are *isomorphic*, if there is an isomorphism α of one onto the other.

A *directed graph* (digraph) is a pair (V, A), where V is a finite set of elements called nodes (or vertices), and A is a set of ordered pairs of V called *arcs*, i.e. A is a binary relation on $V \times V$. Given a digraph $D = (V, A)$, for $v \in V$, the *parents* of v are $A^{-1}(v) = A^{-1}(\{v\}) = \{w \in V \mid \exists w \in V, (w, v) \in A\}$ and the *children* of v are $A(v) = A(\{v\}) = \{w \in V \mid \exists w \in V, (v, w) \in A\}$. A *directed path* of length $n - 1$ is a sequence of n nodes, $v_1, v_2, \ldots, v_n$, such that $\{(v_1, v_2), \ldots, (v_{n-1}, v_n)\} \subseteq A$. A *directed cycle* is a directed path, $v_1, v_2, \ldots, v_n$, where $n \geq 1$ and $(v_n, v_1) \in A$. The *underlying graph* of a given digraph is a graph, which is obtained by replacing every directed arc with an undirected edge. A directed graph is *strongly connected*, if there exists a directed path between every pair of nodes. A directed graph is *weakly connected*, if its underlying graph is connected.

A *directed acyclic graph* (DAG) is a digraph without directed cycles. A *rooted tree* is a DAG that has exactly one node without a parent (called the *root*) and every other node has a unique parent. A *leaf* in a rooted tree is a node without a child. An *internal* node in a rooted tree is a node that has a child. The *subtree* of node v in a rooted tree is the tree that consists of v and all the nodes that have a path from v. The *height* of node v in a rooted tree is the length of the longest directed path from v to a leaf.

2.2 Transition P systems

An essential feature of a membrane system is the *membrane structure*, a (cell-like) hierarchical arrangement of a set of membranes. A *region* of a membrane is a space delimited by the membrane, i.e. the interior of a simple closed curve (Jordan curves [10]). Regions represent various compartments in a cell and each region contains a multiset of *symbols* (i.e. contents).

Regions can contain several membranes inside. Assume that $\subset$ denotes strict inclusion and r_i denotes the region of a membrane m_i. A region r_i is *contained* in a region r_j, if $r_i \subset r_j$. Moreover, a region r_i is *directly contained* in a region r_j, if $r_i \subset r_j$ and there is no region r_k, such that $r_i \subset r_k \subset r_j$. If a membrane m_i is directly contained in a membrane m_j, then m_i is a *child membrane* of m_j and m_j is the *parent membrane* of m_i. A membrane without a parent membrane is called the *skin* membrane. A membrane without a child membrane is called an *elementary* membrane.

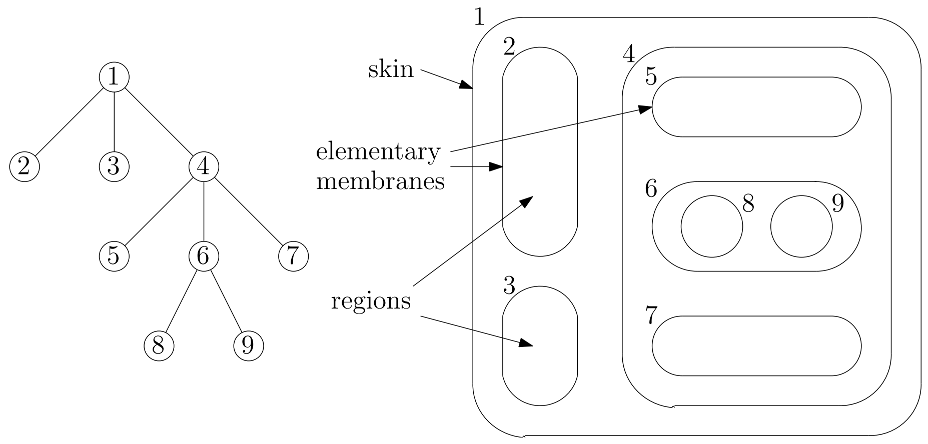


Figure 2.1: A membrane structure represented as a tree and as a Venn diagram.

Membrane structures are often represented as a Venn diagram and a rooted tree. Figure 2.1 [64] illustrates a membrane structure with nine membranes, labelled as

$1, 2, \ldots, 9$, both as a rooted tree and as a Venn diagram.

Consider a multiset rewriting rule of the form $u \to v$, where u and v are multisets. In the evolution rules of membrane systems, the communication between membranes, i.e. passing symbols between membranes, is incorporated in the multiset rewriting rules by specifying *target indications* (i.e. $\odot$, $\uparrow$, $\downarrow$) to each produced symbol of v, where $\odot$ indicates that the produced symbol remains in the current membrane, $\uparrow$ indicates that the produced symbol moves up to the parent membrane and $\downarrow$ indicates that the produced symbol moves down to a non-deterministically chosen child membrane. Multiset rewriting rules that are generalized with target indications are called *transition multiset rewriting rules.*

A *transition P system*, defined in Definition 2.1, is a membrane system based on transition multiset rewriting rules.

Definition 2.1. (Transition P system) A *transition P system* [63] (of order $n \geq 1$) is a construct of the form:

$$\Pi = (O, C, \mu, w_1, w_2, \ldots, w_n, R_1, R_2, \ldots, R_n, i_o)$$

where

1. O is the finite and non-empty alphabet of symbols;
2. $C \subset O$ is the set of catalysts;
3. μ is a membrane structure, a rooted tree consisting of n membranes, labelled with $\sigma_1, \sigma_2, \ldots, \sigma_n$;
4. $w_i \in O^*$, for $1 \leq i \leq n$, is a multiset of symbols present in σ_i, called the *contents*;
5. R_i, for $1 \leq i \leq n$, is a finite set of evolution rules that are associated with membrane σ_i; an evolution rule $r \in R_i$ is a transition multiset rewriting rule of the form $r = u \to v$, where $u \in O^+$ (denoted by $\texttt{LHS}(r)$) and $v \in (O \times \tau)^*$ (denoted by $\texttt{RHS}(r)$), where *target indication* $\tau \in \{\odot, \uparrow, \downarrow\}$;
6. i_o the label of membrane σ_i of μ that presents the output membrane, i.e. a membrane that can send multisets to the "environment".

In each time unit, membranes evolve by applying evolution rules in a *non-deterministic* and *maximally parallel* manner. First, symbols inside a region are assigned to (non-deterministically) chosen evolution rules, such that no further assignment can be made using the remaining unassigned symbols. Then, the assigned evolution rules are applied in parallel: two levels of parallelism during the execution of the step, where (i) all membranes simultaneously apply the selected evolution rules (ii) each membrane applies its selected evolution rules in parallel.

Definition 2.2. (A maximal multiset of evolution rules) For each membrane σ_i, $1 \le i \le n$, a maximal multiset of evolution rules is $M_i \in R_i^*$, such that

$$\bigcup_{r \in M_i} \mathtt{LHS}(r) \subseteq w_i$$

where:

1. For each rule $r \in M_i$, if $\Delta(i) = \emptyset$, then $(o, \downarrow) \notin \mathtt{RHS}(r)$, for all $o \in O$.

2. There is no rule $r' \in R_i$, such that

$$\mathtt{LHS}(r') \cup \bigcup_{r \in M_i} \mathtt{LHS}(r) \subseteq w_i$$

Definition 2.3. (Execution of evolution rules) For each membrane σ_i, $1 \le i \le n$, consider a maximal multiset of evolution rules, M_i, found according to Definition 2.2. For membrane σ_i, multisets U_i, V_i, $V_i^{\uparrow}$ and $V_i^{\downarrow k}$ (where σ_k is a child of σ_i) are defined as follows.

- $U_i = \bigcup_{r \in M_i} \mathtt{LHS}(r)$,

- $V_i = \bigcup_{r \in M_i} \bigcup_{(o, \odot) \in \mathtt{RHS}(r)} \{o\}$,

- $V_i^{\uparrow} = \bigcup_{r \in M_i} \bigcup_{(o, \uparrow) \in \mathtt{RHS}(r)} \{o\}$.

- $V_i^{\downarrow k} = \bigcup_{r \in M_i} \bigcup_{(o, \downarrow) \in \mathtt{RHS}(r)} \{o\}$, where σ_k is a child that σ_i non-deterministically chosen to send symbol o.

Each membrane σ_i, $1 \leq i \leq n$, transforms its content w_i to w'_i, where

$$w'_i = w_i - U_i \ \cup \ V_i \ \cup \ \bigcup_{h \in A(i)} V_h^{\uparrow} \ \cup \ \bigcup_{f \in A^{-1}(i)} V_f^{\downarrow_i}$$

A *configuration* of a membrane system is identified by (i) the membrane structure and (ii) the multiset of symbols available inside each membrane.

Definition 2.4. (Configuration) A *configuration* of a transition P system of order n is an n-tuple of the form $(w_1, w_2, \ldots, w_n)$, where, for $1 \leq i \leq n$, w_i is the current content of membrane σ_i.

A *transition* is a transformation of a configuration (in one time unit), obtained by applying the evolution rules inside each membrane in a non-deterministic and maximally parallel manner. A *computation* in a membrane system is a sequence of transitions, starting from an *initial configuration* (i.e. the membrane structure and the multisets of symbols available inside each membrane at the beginning of a computation).

Definition 2.5. (Transition) Consider two configurations of a transition P system Π of order n, $C' = (w'_1, w'_2, \ldots, w'_n)$ and $C'' = (w''_1, w''_2, \ldots, w''_n)$. A *transition* in system Π is a transformation from C' to C'' in one time unit, denoted by $C' \Rightarrow C''$, such that C'' is obtained from C'. A transition $C' \Rightarrow C''$ consists of two substeps (substep 1 and substep 2); all membranes simultaneously perform substep 2, after every membrane has finished substep 1.

1. **Substep 1:** Each membrane σ_i, $1 \leq i \leq n$, *finds* a (maximal) multiset of evolution rules, M_i, as defined in Definition 2.2.

2. **Substep 2:** Each membrane σ_i, $1 \leq i \leq n$, *executes* a multiset of evolution rules found in substep 1, M_i, as defined in Definition 2.3,

A computation *halts*, if it reaches a configuration (called the halting configuration), where no evolution rule can be applied to the existing symbols inside all membranes. The *output* of a halted computation is defined by the number of symbols present inside a designated membrane (called the output membrane) in the halting configuration.

Definition 2.6. (Halting and results) A transition P system Π *halts*, if no more transitions are possible in Π. For a halted transition P system Π, the *computational result* is the multiset of symbols that are contained inside the output membrane i_o.

2.3 Extensions and variants

This section presents several extensions and variants introduced in membrane systems.

Generalized membrane structure

Neural P systems [63] (and tissue P systems [51]) generalize the structure of a membrane system from a (cell-like) hierarchical arrangement of membranes to a (tissue-like) net of cells, where cells are placed in the nodes of an arbitrary digraph. In tissue P systems, the direction of structural arcs represent the direction of *unidirectional* communication; given a structural arc (σ_i, σ_j), cell σ_i can communicate (i.e. transfer symbols) to cell σ_j, but σ_j cannot communicate to σ_i.

Priority

This extension considers a total order $(\leq)$ on the set of evolution rules. There are two interpretations of priority: *strong priority* and *weak priority.*

- **Strong priority scheme:** In the *strong* interpretation of the priority, if a higher priority rule was applied, then a lower priority rule cannot be applied at all. That is, in every transition, each cell applies the applicable rule with the maximal priority. More precisely, in the first substep, the applicable rule with the maximal priority assigns as many symbols as possible. The remaining unassigned symbols remain unassigned in the current step, even if there are other applicable rules.

- **Weak priority scheme:** In the *weak* interpretation of the priority, rules are applied in decreasing order of their priorities—where a lower priority rule can only be applied after all higher priority rules have been applied (as required by the rewriting modes). More precisely, in the first substep, the applicable rule with the maximal priority assigns as many symbols as possible. We repeat the following on the remaining unassigned symbols, until there is no applicable rule. The remaining unassigned symbols are assigned, as many symbols as possible, by the next rule in the decreasing priority order.

Example 2.7. Assume that, at step t, a cell σ_i has content $w_i = a^4b^2c^3$ and has set of evolution rules with priorities $\{r_1, r_2, r_3\}$, where $r_1 = 1\ ab \to c$, $r_2 = 2\ a \to ab$ and $r_3 = 3\ cc \to b$. Under the strong priority scheme, at step $t+1$, cell σ_i applies rule r_1 twice and does not apply other rules; only the applicable rule with the highest priority is applied. After step $t+1$, σ_i ends with content $w_i = a^2c^5$. Under the weak priority scheme, at step $t+1$, cell σ_i applies: (i) rule r_1 twice, (ii) rule r_2 twice and (iii) rule r_3 once. After step $t+1$, σ_i ends with content $w_i = a^2b^3c^3$.

Cell states

Cell states are introduced in neural P systems [63] (and tissue P systems [51]). Each cell possesses a state, as well as contents and rules, which are used, together with contents, to control the applications of evolution rules. Consider an evolution rule of a transition P system, $r = u \to v$. An evolution rule r with cell states s and s' has the form

$$r = s\ u \to s'\ v$$

where s and s' are called the *source* and *target* states of rule r, respectively.

Presence of states in an evolution rule introduces additional conditions in the application of rules. A cell executing rule r must be in state s. Further, if several rules are executed together, then all these rules must have the same target state. A cell transforms its current cell state s to s' by executing rule r.

Example 2.8. Assume that, at step t, a cell σ_i is in state s, has content $w_i = a^4b^2c^3$ and has set of evolution rules $\{r_1, r_2, r_3\}$, where $r_1 = s\ ab \to s'\ c$, $r_2 = s\ a \to s''\ ab$ and $r_3 = s'\ cc \to s''\ b$. At step $t+1$, cell σ_i cannot apply rule r_3, because the source state of rule r_3 does not equal σ_i's current state. At step $t+1$, cell σ_i can either apply: (i) rule r_1 twice or (ii) rule r_2 four times; σ_i cannot apply rule r_1 together with rule r_2, because rules r_1 and r_2 have different target state. If cell σ_i applies rule r_1 twice at step $t+1$, then σ_i will end in state s' with content $w_i = a^2c^5$.

Rewriting operators

Neural P systems [63] (and tissue P systems [51]) introduced *rewriting operators* (`min`, `par`, `max`) that provide alternative ways to apply evolution rules. At the initial configuration of a system Π, one of these operators is chosen and is adopted by all cells of Π. Operator `min` restricts each cell to select one applicable evolution rule and apply it once. Operator `par` restricts each each cell to select one applicable evolution rule and apply it as many times as possible. Operator `max` allows each cell σ_i to select and apply evolution rules as defined in Definitions 2.2 and 2.3, respectively.

Hyperdag P systems [58] generalized these rewriting operators by specifying a rewriting operator for each cell. Then, P modules [24, 23] further generalized these rewriting operators by specifying a rewriting operator for each evolution rule; note, only `min` and `max` operators are considered. An evolution rule r with rewriting operators $\alpha \in \{\mathtt{min}, \mathtt{max}\}$ is of the form

$$r = u \rightarrow_\alpha v$$

where: (i) if $\alpha = \mathtt{min}$, then rule r is applied once, and (ii) if $\alpha = \mathtt{max}$, then rule r is applied as many times as possible.

Example 2.9. Assume that, at step t, a cell σ_i has content $w_i = a^4b^2c^3$ and set of evolution rules $\{r_1, r_2, r_3\}$, where $r_1 = ab \rightarrow_{\mathtt{max}} c$, $r_2 = a \rightarrow_{\mathtt{min}} ab$ and $r_3 = cc \rightarrow_{\mathtt{max}} b$. At step $t+1$, cell σ_i applies: (i) rule r_1 twice, (ii) rule r_2 once and (iii) rule r_3 once. Cell σ_i can only execute rule r_2 once at step $t+1$, because the rewriting mode of rule r_2 is set to `min`. After step $t+1$, cell σ_i ends with content $w_i = a^2b^2c^3$.

2.4 Simple P systems

Simple P systems incorporate several extended features described in Section 2.3, such as: (i) generalized cell structure, (ii) cell states, (iii) weak priority scheme and (iv) rewriting operators of P modules. The definition of a simple P system is given below.

Definition 2.10. (Simple P system) A *simple P system* of order n is a system $\Pi = (O, K, \Delta)$, where:

1. O is a finite non-empty alphabet of *symbols.*
2. $K = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ is a finite set of *cells*, where each cell $\sigma_i \in K$ is of the form:
$$\sigma_i = (Q_i, s_{i0}, w_{i0}, R_i)$$
where
   - Q_i is a finite set of *states*,
   - $s_{i0} \in Q_i$ is the *initial state* ($s_i \in Q_i$ denotes the *current state*),
   - $w_{i0} \in O^*$ is the *initial content* and ($w_i \in O^*$ denotes the *current content*),
   - R_i is a finite *linearly ordered* set of evolution rules (i.e. transition multiset rewriting rules with *priority* and *rewrite operator*). An evolution rule $r \in R_i$ has the form:
   $$r : \ j \ s \ u \rightarrow_\alpha s' \ v$$
   where:
     - $\alpha \in \{\mathtt{min}, \mathtt{max}\}$ is a *rewriting* operator of r, denoted by $\mathtt{rewrite}(r)$,
     - $j \in \mathbb{N}$ is the *priority* of r, denoted by $\mathtt{priority}(r)$, where the lower value j indicates higher priority,
     - $s, s' \in Q_i$, where s is the *source state* of r, denoted by $\mathtt{source}(r)$, and s' is the *target state* of r, denoted by $\mathtt{target}(r)$,
     - $u \in O^+$, denoted by $\mathtt{LHS}(r)$,
     - $v \in (O \times \tau)^*$, denoted by $\mathtt{RHS}(r)$, where $\tau \in \{\odot, \uparrow, \downarrow, \updownarrow\}$.

   The *initial configuration* of σ_i is denoted by (s_{i0}, w_{i0}) and the *current configuration* of σ_i is denoted by (s_i, w_i).
3. Δ is an irreflexive and asymmetric relation, representing a set of arcs between cells with *bidirectional* communication capabilities.

Definition 2.11. (Target indicator) The meaning of each element of $\{\odot, \uparrow, \downarrow, \updownarrow\}$ is as follows. For cell σ_i, consider an evolution rule $r \in R_i$ with $(o, \tau) \in \mathtt{RHS}(r)$, where $o \in O$.

- If $\tau = \odot$, then symbol o will remain in σ_i.
- If $\tau = \uparrow$, then symbol o will be replicated and sent to each $\sigma_j \in \Delta^{-1}(i)$, if any.
- If $\tau = \downarrow$, then symbol o will be replicated and sent to each $\sigma_j \in \Delta(i)$, if any.
- If $\tau = \updownarrow$, then symbol o will be replicated and sent to each $\sigma_j \in \Delta^{-1}(i) \cup \Delta(i)$, if any.

Definition 2.12. (Applications of an evolution rule) Given a multiset $w \in O^*$ and an evolution rule $r \in R$, where $\mathtt{LHS}(r) \subseteq w$, the number of applications of r over w is

$$\mathtt{apply}(r, w) = \begin{cases} 1 & \text{if } \mathtt{rewrite}(r) = \mathtt{min}, \\ |w|_{\mathtt{LHS}(r)} & \text{if } \mathtt{rewrite}(r) = \mathtt{max}. \end{cases}$$

Definition 2.13. (A maximal multiset of evolution rules) For cell σ_i, in state s_i with content w_i and a set of evolution rules R_i, a maximal multiset of evolution rules, M_i, is obtained by the procedure below.

```
**Input:** a set of evolution rules $R_i$ and a multiset $w := w_i$.
**Output:** a maximal multiset $M_i$.
$M_i := \emptyset$
for each $r_j \in R_i$, $1 \leq j \leq |R_i|$ (in an increasing priority order)
    if $(M_i = \emptyset \,||\, \forall r_k \in M_i\ (\mathtt{target}(r_j) = \mathtt{target}(r_k)))$ then
        if $(\mathtt{LHS}(r_j) \subseteq w$ and $\mathtt{source}(r_j) = s_i)$ then
            $m := \mathtt{apply}(r_j, w)$
            $M_i := M_i \cup \{r_j^m\}$
            $w := w - \mathtt{LHS}(r_j)^m$
        endif
    endif
endfor
```

Definition 2.14. (Execution of evolution rules) For each cell σ_i, $1 \leq i \leq n$, consider a maximal multiset of evolution rules, M_i, found according to Definition 2.13. For cell σ_i with the current content w_i, multisets U_i, V_i, $V_i^{\downarrow}$, $V_i^{\uparrow}$ and $V_i^{\updownarrow}$, for each $\sigma_k \in \Delta(i) \cup \Delta^{-1}(i)$, are defined as follow:

- $U_i = \bigcup_{rj \in M_i} \texttt{LHS}(r_j)$, denotes the multiset of symbols that will be consumed from w_i.
- $V_i = \bigcup_{rj \in M_i} \bigcup_{(o,\odot) \in \texttt{RHS}(r_j)} \{o\}$, denotes the multiset of symbols that will be produced and added to w_i.
- $V_i^{\downarrow} = \bigcup_{rj \in M_i} \bigcup_{(o,\downarrow) \in \texttt{RHS}(r_j)} \{o\}$, denotes the multiset of symbols that will be sent to each $\sigma_k \in \Delta(i)$.
- $V_i^{\uparrow} = \bigcup_{rj \in M_i} \bigcup_{(o,\uparrow) \in \texttt{RHS}(r_j)} \{o\}$, denotes the multiset of symbols that will be sent to each $\sigma_k \in \Delta^{-1}(i)$.
- $V_i^{\updownarrow} = \bigcup_{rj \in M_i} \bigcup_{(o,\updownarrow) \in \texttt{RHS}(r_j)} \{o\}$, denotes the multiset of symbols that will be sent to each $\sigma_k \in \Delta(i) \cup \Delta^{-1}(i)$.

For each cell σ_i in state s_i with content w_i:

- If $M_i = \emptyset$, then σ_i remains in state s_i with content w_i.
- Otherwise, σ_i transforms:
  - its current state to $s_i = \texttt{target}(r_f)$, where $r_f \in M_i$.
  - its current content w_i to w'_i, where

$$w'_i = w_i - U_i \;\cup\; V_i \;\cup \bigcup_{f \in \Delta^{-1}(i)} V_f^{\downarrow} \;\cup\; \bigcup_{g \in \Delta(i)} V_g^{\uparrow} \;\cup \bigcup_{h \in \Delta(i) \cup \Delta^{-1}(i)} V_h^{\updownarrow}$$

Definition 2.15. (Configuration) A *configuration* of a simple P system of order n is an n-tuple of the form $(s_1\ w_1, s_2\ w_2, \ldots, s_n\ w_n)$, where, for $1 \leq i \leq n$, s_i and w_i are the current state and content of cell σ_i, respectively.

Definition 2.16. (Transition) Consider two configurations of a simple P system Π of order n, $C' = (s'_1\ w'_1, s'_2\ w'_2, \ldots, s'_n\ w'_n)$ and $C'' = (s''_1\ w''_1, s''_2\ w''_2, \ldots, s''_n\ w''_n)$. A *transition* in system Π is a transformation from C' to C'' in one time unit, denoted by $C' \Rightarrow C''$, such that C'' is obtained from C'. A transition $C' \Rightarrow C''$ consists of two substeps (substep 1 and substep 2); first, all cells synchronously perform substep 1, then, after all cells have finished substep 1, all cells synchronously perform substep 2.

1. **Substep 1:** Each cell σ_i, $1 \le i \le n$, *finds* a maximal multiset of evolution rules, M_i, as defined in Definition 2.13.

2. **Substep 2:** Each cell σ_i, $1 \le i \le n$, *executes* this maximal multiset of rules found in substep 1, M_i, as defined in Definition 2.14.

Remark 2.17. An *evolution trace table* is a table that contains the history of transitions of a considered system, i.e. the history of configurations. Note, trace tables do not indicate the multiset of applicable rules used to obtain the resulting contents. Trace tables are a good representation to show the system evolutions. A Sevilla carpet [16, 18] provides a graphical presentation that indicates the number of rule applications in a considered system.

Definition 2.18. (Computation) A simple P system Π *halts*, if no more transitions are possible in Π. The *results* of a halted simple P system is the final configuration, i.e. the configuration of a halted system.

Definition 2.19. (Deterministic system) A simple P system is considered a *deterministic system*, if for every transition $C \Rightarrow C'$, C' is unique.

Definition 2.20. (Time complexity) The *time complexity* of a simple P system Π is the number of steps needed for Π to halt.

Definition 2.21. (Message complexity) The *message complexity* of a simple P system Π is the total number of symbols that are transferred between cells until Π halts.

2.5 Summary

In this chapter, I introduced a new membrane system, called a simple P system, which will be used to design distributed P algorithms in Chapters 4, 5 and 6.

The properties of simple P systems that are relevant to distributed algorithms are described below.

1. System Π is synchronous, i.e. Π assumes a global clock, which indicates the time and transition steps for all cells.
2. Cells can only communicate with their neighbours. Any communication to a remote cell, i.e. a non-neighbour, is relayed through intermediate cells.
3. Transferred symbols are always guaranteed to arrive, without being modified or lost.
4. Cells receive every incoming symbol, i.e. cells do not have a choice whether to receive the incoming symbols or not.
5. Initially, cells do not have the knowledge of the global topology, i.e. membrane structure, and local neighbourhood, i.e. the number and existence of neighbours.

Additional properties of a simple P system Π, that are typically considered in the P algorithm design, are listed below.

1. All cells share the same alphabet, cell states sets and set of evolution rules.
2. System Π terminates after a finite number of steps.
3. All cells start from a quiescent state.
4. The source is the only cell with an applicable evolution rule from the initial quiescent state.

Each P system will be presented and analysed according to the following sequence.

1. **Overview:** The overview describes the main idea of the algorithm. We include a *visual description* that illustrates the manner in which cells communicate (i.e. symbol transfers) during the algorithm.

2. **Precondition:** Denotes the initial configuration of cells.

3. **Postcondition:** Denotes the final configuration of cells, after applying the evolutions rules according to the rule description.

4. **Symbols and states (optionally, where required):** Explain the meaning of symbols and states used, i.e. what each symbol and state represent in the algorithm.

5. **Evolution rules:** Provide a set of evolution rules that enable cells to achieve the algorithm objective.

6. **Rules description:** Informally describes the effect of the evolution rules. We provide an *evolution trace table*, using an example system, that shows the history of cells' evolutions during each algorithm.

7. **Time and message complexities:** Indicate (i) the number of steps needed to terminate an algorithm and (ii) the number of symbols exchanged between cells during an algorithm.

Chapter 3

Turing completeness of simple P systems

This chapter shows that simple P systems, introduced in Chapter 2, are *Turing complete* (i.e. can compute anything a Turing machine [42, 74] can compute) and *universal* (i.e. can simulate an arbitrary Turing machine with input), by proving that simple P systems can simulate all `READ`-enhanced register machines [9]. This register machine model allows input data (denoted as a sequence of bits or characters), which are accessed using `READ` instructions that return the next unread bit. Several universality results in membrane systems have used register machines that need to store input data in the registers, e.g. [49].

This chapter is organized as follows. Section 3.1 recalls the definition of `READ`-enhanced register machines [9]. Section 3.2 presents simple P systems that simulate `READ`-enhanced register machines (with binary input data). Additionally, using the greatest common divisor (GCD) function, this section provides two simple P systems, obtained via direct and indirect approaches, respectively. Finally, a summary of this chapter is given in Section 3.3.

3.1 Register machines

A register machine, as presented in [9], has $n > 1$ instructions and $m > 0$ registers, where each register may contain an arbitrarily large non-negative integer. Register machines are Turing-complete and universal. A register machine program consists of a finite list of instructions, `EQ`, `SET`, `ADD`, `READ` and `HALT`, with the restriction that the `HALT` instruction appears only once, as the last instruction of the list, followed by input data. The first instruction of a program is indexed by the value 0. Here, a program is presented in symbolic instruction form.

3.1.1 Register machine instructions

A set of instructions of a register machine M [9] is listed below. In the instructions below, r_1, r_2 and r_3 denote registers and k denotes a non-negative binary integer constant.

1. **Instruction** (`EQ` r_1 r_2 r_3) **or** (`EQ` r_1 k r_3)**:**
   Assume that j denotes the content of r_3. If the content of r_1 equals (i) the content of r_2 or (ii) the constant k, then the execution of M continues at the j-th instruction. If the content of r_1 does not equal (i) the content of r_2 or (ii) the constant k, then the execution of M continues at the next instruction in the sequence.

2. **Instruction** (`SET` r_1 r_2) **or** (`SET` r_1 k)**:**
   The content of r_1 is replaced by (i) the content of r_2 or (ii) the constant k.

3. **Instruction** (`ADD` r_1 r_2) **or** (`ADD` r_1 k)**:**
   The content of r_1 is replaced by (i) the sum of the contents of r_1 and r_2 or (ii) the sum of the contents of r_1 and constant k.

4. **Instruction** (`READ` r_1)**:**
   One bit is read into r_1, so the numerical value of r_1 becomes either 0 or 1. Any attempt to read past the last data-bit results in a run-time error.

5. **Instruction** (`HALT`)**:**
   This is the last instruction of a register machine program.

3.1.2 Input data

In a register machine program, its input data, denoted as a sequence of bits (or characters), follows immediately after the halt instruction. Note, some programs may not have input data and it is up to the program to know how to process the data in the chosen encoding format. A variation of the following two encoding formats is used to represent register machine input data within a simple P system.

1. A sequence of non-negative integers, $a_1, a_2, \ldots, a_k$, $a_i \in \mathbb{Z}$, $1 \leq i \leq k$, is encoded as $10^{a_1}10^{a_2}1 \ldots 10^{a_k}1$. Following [19] as an example, the sequence $[3, 0, 2]$ is represented by the integer $281_{(10)} = 100011001_{(2)}$.
2. A self-delimiting representation of a sequence of bits $b_1 b_2 \cdots b_k$, $b_i \in \{0, 1\}$, $1 \leq i \leq k$, is encoded as $1b_1 1b_2 \cdots 1b_k 0$. Note, if used to represent a positive integer n in base 2, then we need only $O(k) = O(\log_2 n)$ bits, but twice the number of 'real' bits.

3.1.3 Run-time errors

A register machine program, with $n \geq 1$ instructions, can encounter the following run-time errors:

1. **Illegal branch error:** This error occurs when an `EQ` instruction is executed where the value indicated by its third register is greater or equal to n.
2. **Under-read error:** This error occurs if a register machine halts with unread input data, i.e. when the `HALT` instruction is encountered and there exist unread input data.
3. **Over-read error:** This error occurs if a register machine attempts to read past the last data-bit, i.e. when a `READ` instruction is encountered and the entire input data has already been read.

3.1.4 Register machine instructions for GCD algorithm

The Euclidean algorithm, based on modular operation, for computing greatest common divisor (GCD) function is given in Definition 3.1.

Definition 3.1. (GCD based on modular operation (%))

Input: $x \geq 1$ and $y \geq 1$.
Output: the final value of $y = GCD(x, y)$.

```
1.  function GCD($x, y$)
2.      if ($y == 0$)
3.          return $x$
4.      else
5.          return GCD($y, x\%y$)
6.      endif
7.      return $x$
```

Note, the chosen register machine model [9] does not provide modular operation as a primitive operation. Modular operation can be implemented as subroutine, using `SET`, `ADD` and `EQ` operations. The Euclidean algorithm, based on subtraction operation, for computing GCD function is given in Definition 3.2.

Definition 3.2. (GCD based on subtraction)

Input: $x \geq 1$ and $y \geq 1$.
Output: the final value of $y = GCD(x, y)$.

```
1.  function GCD($x, y$)
2.      while ($x \neq 0$)
3.          if ($x < y$)
4.              $z := x$
5.              $x := y$
6.              $y := z$
7.          endif
8.          $x := x - y$
9.      endwhile
10.     return $x$
```

Register machine instructions, in symbolic instruction forms, that correspond to the steps of the GCD algorithm of Definition 3.2 is given below. Assume that, the input data is of form 0^x10^y1, which represents a sequence of two integers x and y. This register machine implementation has exactly 37 instructions, where it reads any input values x and y from its data.

- Initialize registers $a, b, c, d, e, f, g, h, i$ with the instruction line numbers (to be used as targets in branching `EQ` instructions).

Line number	Symbolic instruction
0	`SET` a $\mathtt{L_a}$
1	`SET` b $\mathtt{L_b}$
2	`SET` c $\mathtt{L_c}$
3	`SET` d $\mathtt{L_d}$
4	`SET` e $\mathtt{L_e}$
5	`SET` f $\mathtt{L_f}$
6	`SET` g $\mathtt{L_g}$
7	`SET` h $\mathtt{L_h}$
8	`SET` i $\mathtt{L_i}$

- Initialize registers x and y, using auxiliary register z, with the first and the second values of the input data, respectively.

Line number	Symbolic instruction
9	$\mathtt{L_a}$: `READ` z
10	`EQ` z 1 b
11	`ADD` x 1
12	`EQ` a a a
13	$\mathtt{L_b}$: `READ` z
14	`EQ` z 1 c
15	`ADD` y 1
16	`EQ` b b b

- Check the condition $x = 0$.

Line number	Symbolic instruction
17	$\mathtt{L_c}$: `EQ` x 0 i

- Check the condition $x < y$.

Line number	Symbolic instruction
18	`SET` z x
19	`SET` w y
20	$\mathtt{L_d}$: `EQ` z y f
21	`ADD` z 1
22	`ADD` w 1
23	`EQ` x w f
24	`EQ` y z e
25	`EQ` d d d
26	$\mathtt{L_e}$: `SET` y x
27	`SET` x z

- Execute $x := x - y$.

Line number	Symbolic instruction
28	$\mathtt{L_f}$: `SET` z y
29	`SET` w 0
30	$\mathtt{L_g}$: `EQ` x z h
31	`ADD` z 1
32	`ADD` w 1
33	`EQ` g g g
34	$\mathtt{L_h}$: `SET` x w
35	`EQ` c c c

- Halt.

Line number	Symbolic instruction
36	$\mathtt{L_i}$: `HALT`

3.2 Universality results

For a given `READ`-enhanced register machine M [9] with $n \geq 1$ instructions, $m \geq 0$ registers and input data bits $\beta = b_1 b_2 \cdots b_\nu$, build a simple P system $\Pi_M = (O, K, \Delta)$ that simulates M, where:

1. $K = \{\sigma_m, \sigma_p\}$, where σ_m is called the *main* cell and σ_p is called the *provider* cell. The descriptions of these cells are given in Sections 3.2.1 and 3.2.2.
2. $\Delta = \{(\sigma_m, \sigma_p)\}$.
3. $O = \{r_i \mid 0 \leq i < k\} \cup \{\delta, \gamma, \mu, \phi, \pi, \theta\}$, where symbols $r_0, r_1, \ldots, r_{k-1}$ represent the registers of M. Symbols π, γ and θ are auxiliary symbols, used only by the main cell, for executing the evolution rules that correspond to `HALT` and `EQ` instructions. Symbols μ, ϕ and δ are used as *communication messages*, i.e. requests and responses, between the main and provider cells during the execution the evolution rules that correspond to `READ` and `HALT` instructions—Section 3.2.3 describes the manner in which these symbols are exchanged.

3.2.1 The main cell

The role of the main cell is to simulate every instruction of M, except data-bit extraction—this operation is handled by the provider cell. The main cell, σ_m, is of the form $\sigma_m = (Q_m, s_{m0}, w_{m0}, R_m)$, where:

- $Q_m = \{s_i, s'_i, s''_i \mid 0 \leq i < n\} \cup \{s\}$, where states s_i, s'_i and s''_i, $0 \leq i < n$, represent the i-th instruction of M, state s_{n-1} represents the "halting" state and state s represents the "springboard jump" state. From state s, cell σ_m transits to state s_{j-1}, where $j \leq n$ is the multiplicity of symbol t that σ_m currently contains; if $j > n$, then σ_m remains at state s.
- $s_{m0} = s_0$, indicates the first instruction, i.e. 0-th instruction.
- $w_{m0} = \{r_i \mid 0 \leq i < k\} \cup \{\pi\}$, indicates the initial content of cell σ_m, where the value of each register r_i, $0 \leq i < k$, corresponds to the multiplicity of r_i minus one, i.e. $|w_m|_{r_i} - 1$.

- R_m corresponds to the instructions of M, described next.

Evolution rules for a `SET` instruction

An i-th instruction of the form, either ($\texttt{SET}\ r_{i_1}\ r_{i_2}$) or ($\texttt{SET}\ r_{i_1}\ k_i$), is translated into the following evolution rules. The rules below first consume, all but one, copies of symbol r_{i_1} and then produce j additional copies of symbol r_{i_1}, where: (i) j is the multiplicity of symbol r_{i_2}, i.e. the value of the register r_{i_2}, or (ii) $j = k_i$, i.e. the value of constant k_i.

Instruction	Corresponding evolution rules
($\texttt{SET}\ r_{i_1}\ r_{i_2}$)	1 $s_i\ r_{i_1} \rightarrow_{\texttt{max}} s_{i+1}$
	2 $s_i\ r_{i_2} \rightarrow_{\texttt{max}} s_{i+1}\ r_{i_1}\ r_{i_2}$

Using the rules above, the main cell consumes all copies of symbol r_{i_1}, i.e. set the value of register r_{i_1} to 0. At the same time, the main cell rewrites every copy of symbol r_{i_2} into multiset $r_{i_1} r_{i_2}$, i.e. set the value of register r_{i_1} to the value of register r_{i_2}.

Instruction	Corresponding evolution rules
($\texttt{SET}\ r_{i_1}\ k_i$)	1 $s_i\ r_{i_1} \rightarrow_{\texttt{min}} s_{i+1}\ r_{i_1}^{k_i+1}$
	2 $s_i\ r_{i_1} \rightarrow_{\texttt{max}} s_{i+1}$

Using the rules above, the main cell: (i) rewrites one copy of symbol r_{i_1} into $k_i + 1$ copies of symbol r_{i_1} and (ii) consumes all the remaining copies of symbol r_{i_1}, i.e. set the value of register r_{i_1} to constant k_i.

Evolution rules for an `ADD` instruction

An i-th instruction of the form, either ($\texttt{ADD}\ r_{i_1}\ r_{i_2}$) or ($\texttt{ADD}\ r_{i_1}\ k_i$), is translated into the following evolution rules. The rules below produce j additional copies of symbol r_{i_1}, where: (i) j is the multiplicity of symbol r_{i_2}, i.e. the value of register r_{i_2} or (ii) $j = k_i$, i.e. the value of the constant k_i.

Instruction	Corresponding evolution rules
$(\texttt{ADD}\ r_{i_1}\ r_{i_2})$	$1\ s_i\ r_{i_2} \rightarrow_{\texttt{min}} s_{i+1}\ r_{i_2}$
	$2\ s_i\ r_{i_2} \rightarrow_{\texttt{max}} s_{i+1}\ r_{i_1}\ r_{i_2}$

Using the rules above, except one copy of symbol r_{i_2}, the main cell rewrites every copy of symbol r_{i_2} into multiset $r_{i_1}r_{i_2}$, i.e. set the value of register r_{i_1} to the sum of values of registers r_{i_1} and r_{i_2}.

Instruction	Corresponding evolution rule
$(\texttt{ADD}\ r_{i_1}\ k_i)$	$1\ s_i\ r_{i_1} \rightarrow_{\texttt{min}} s_{i+1}\ r_{i_1}^{k_i+1}$

Using the rules above, the main cell rewrites one copy of symbol r_{i_1} into $k_i + 1$ copies of symbol r_{i_1}, i.e. set the value of register r_{i_1} to the sum of the value of register r_{i_1} and constant k_i.

Evolution rules for an `EQ` instruction

An i-th instruction of the form, $(\texttt{EQ}\ r_{i_1}\ r_{i_1}\ r_{i_3})$, $(\texttt{EQ}\ r_{i_1}\ r_{i_2}\ r_{i_3})$ or $(\texttt{EQ}\ r_{i_1}\ k_i\ r_{i_3})$, is translated into the following evolution rules. If the first two indicated registers (or the first register and the constant) have the same value, then the main cell produces j copies of symbol θ, where j corresponds to the value of register r_{i_3}, and transits to the “springboard” state. Otherwise, the main cell transits to state s_{i+1}, i.e. the next instruction. Note, in the rules for instruction $(\texttt{EQ}\ r_{i_1}\ r_{i_2}\ r_{i_3})$, the multiplicity of symbol γ corresponds to the difference in the values of registers r_{i_1} and r_{i_2}.

Instruction	Corresponding evolution rule
$(\texttt{EQ}\ r_{i_1}\ r_{i_1}\ r_{i_3})$	$1\ s_i\ r_{i_3} \rightarrow_{\texttt{max}} s\ r_{i_3}\ \theta$

Instruction	Corresponding evolution rules
(`EQ` r_{i_1} r_{i_2} r_{i_3})	**Rules for state s_i:**
	1 $s_i\ r_{i_1}\ r_{i_2} \rightarrow_{\texttt{max}} s_i'\ r_{i_1}\ r_{i_2}$
	2 $s_i\ r_{i_1} \rightarrow_{\texttt{max}} s_i'\ r_{i_1}\ \gamma$
	3 $s_i\ r_{i_2} \rightarrow_{\texttt{max}} s_i'\ r_{i_2}\ \gamma$
	Rules for state s_i':
	1 $s_i'\ \gamma \rightarrow_{\texttt{max}} s_{i+1}$
	2 $s_i'\ r_{i_3} \rightarrow_{\texttt{max}} s\ r_{i_3}\ \theta$

Instruction	Corresponding evolution rules
(`EQ` r_{i_1} k_i r_{i_3})	1 $s_i\ r_{i_1}^{k_i+2} \rightarrow_{\texttt{min}} s_{i+1}\ r_{i_1}^{k_i+2}$
	2 $s_i\ r_{i_1}^{k_i+1} \rightarrow_{\texttt{min}} s\ r_{i_1}^{k_i+1}$
	3 $s_i\ r_{i_1} \rightarrow_{\texttt{min}} s_{i+1}\ r_{i_1}$
	4 $s_i\ r_{i_3} \rightarrow_{\texttt{max}} s\ r_{i_3}\ \theta$

The "springboard" state mimics the manner in which a register machine performs a "GOTO" operation to move to l-th instruction, $0 \le l \le n-1$. The springboard state contains one rule designated for each value $1, 2, \ldots, n$, such that $1 \le j \le n$ copies of symbol θ will lead the main cell to transit to state s_{j-1}. Additionally, the springboard state contains one extra rule designated for all values greater than n, such that $j > n$ copies of symbol θ will lead the main cell to infinite loop state transitions.

Rules for the "springboard" state s:
1 $s\ \theta^{n+1} \rightarrow_{\texttt{min}} s\ \theta^{n+1}$
2 $s\ \theta^{n} \rightarrow_{\texttt{min}} s_{n-1}$
3 $s\ \theta^{n-1} \rightarrow_{\texttt{min}} s_{n-2}$
$\vdots$
$n+1$ $s\ \theta \rightarrow_{\texttt{min}} s_0$

Evolution rules for a `READ` instruction

An *i*-th instruction of the form (`READ` r_{i_1}) is translated into the following evolution rules in the main cell. The rules below set the multiplicity of symbol r_{i_1} to either *one* or *two* (indicating a data-bit of value 0 or 1, respectively).

Setting the multiplicity of symbol r_{i_1} as described above requires interactions with the provider cell. Symbol μ represents the "last unread data-bit" request from the main cell to the provider cell. Symbols ϕ and δ represent the provider cell's responses, where (i) symbol ϕ indicates that the entire data has been read and (ii) symbol δ indicates that the last unread data-bit is 1. The provider cell's other response is not to send any symbol—this response indicates that the last unread data-bit is 0.

According to the provider cell's three possible responses, the main cell sets the multiplicity of symbol r_{i_1} as follows. Note, the main cell initially contains one copy of symbol r_{i_1}. If the main cell receives: (i) symbol δ, then the main cell rewrites the received symbol δ into symbol r_{i_1}, such that the final multiplicity of symbol r_{i_1} is two, or (ii) symbol ϕ, then the main cell enters infinite loop state transitions. Otherwise, the main cell remains idle, such that the final multiplicity of symbol r_{i_1} remains at one.

Instruction	Corresponding evolution rules
(`READ` r_{i_1})	**Rules for state s_i:**
	1 $s_i\ r_{i_1} \rightarrow_{\texttt{min}} s'_i\ r_{i_1}\ (\mu, \downarrow)$
	2 $s_i\ r_{i_1} \rightarrow_{\texttt{max}} s'_i$
	Rules for state s'_i:
	1 $s'_i\ r_{i_1} \rightarrow_{\texttt{min}} s''_i\ r_{i_1}$
	Rules for state s''_i:
	1 $s''_i\ \phi \rightarrow_{\texttt{min}} s''_i\ \phi$
	2 $s''_i\ \delta \rightarrow_{\texttt{min}} s_{i+1}\ r_{i_1}$
	3 $s''_i\ r_{i_1} \rightarrow_{\texttt{min}} s_{i+1}\ r_{i_1}$

Evolution rules for a HALT instruction

The last instruction must be HALT, which is translated into the following evolution rules. According to the rules below, the main cell reaches either: (i) a halting configuration or (ii) an infinite loop configuration. The rules below involve the interaction between the main and provider cells, where the responses from the provider cell are as described in the READ instruction. If the provider cell's response is symbol ϕ, then the main cell reaches a halting configuration. Otherwise, the main cell enters an infinite loop configuration.

Instruction	Corresponding evolution rules
(HALT)	**Rules for state s_{n-1}:**
	1 $s_{n-1}\ \pi \rightarrow_{\mathtt{min}} s'_{n-1}\ \pi\ (\mu, \downarrow)\ (\mu, \downarrow)$
	Rules for state s'_{n-1}:
	1 $s'_{n-1}\ \pi \rightarrow_{\mathtt{min}} s''_{n-1}\ \pi$
	Rules for state s''_{n-1}:
	1 $s''_{n-1}\ \phi\ \pi \rightarrow_{\mathtt{min}} s_{n-1}$
	2 $s''_{n-1}\ \pi \rightarrow_{\mathtt{min}} s''_{n-1}\ \pi$

3.2.2 The provider cell

The role of the provider cell is to obtain and send the last unread bit to the main cell. For input data of ν bits, $\beta = b_1 b_2 \cdots b_\nu$, where $b_i \in \{0, 1\}$ and $1 \le i \le \nu$, the provider cell initially contains the multiset δ^k, where k is the value of the binary-encoded integer $\beta' = 1 b_\nu b_{\nu-1} \cdots b_1$. The bit of value 1 at the first position of β', which is not part of β, ensures that if the last $k \ge 1$ bits of β, i.e. $b_{\nu-k} b_{\nu-k+1} \cdots b_\nu$, are of value 0, then we do not lose them.

Let $x > 1$ denote the current multiplicity of symbol δ in the provider cell, σ_p. At the i-th READ instruction, cell σ_p performs "mod 2" operation on value x to obtain the i-th bit. Then σ_p performs a "div 2" operation on the value of x to prepare for the next READ instruction, if any. For example, if cell σ_p has $x = 11_{(10)} = 1011_{(2)}$ copies of symbol δ, then the next three successive bits returned to the main cell are 1, 1 and 0.

The provider cell, σ_p, is of the form $(Q_p, s_{p0}, w_{p0}, R_p)$, where:

- $Q_p = \{s, s'\}$.
- $s_{p0} = s$.
- $w_{p0} = \{\delta^{\beta'}\}$, where $\beta' = 1b_nb_{n-1}\cdots b_{1(2)}$.
- R_p is the following rules, which correspond to the `READ` and `HALT` instructions of M.

Rules for state s:

1 $s\ \delta\ \delta\ \mu \rightarrow_{\texttt{min}} s'\ \delta$

2 $s\ \delta\ \mu \rightarrow_{\texttt{min}} s'\ \delta\ (\phi, \uparrow)$

3 $s\ \delta \rightarrow_{\texttt{min}} s\ \delta$

4 $s\ \delta\ \delta \rightarrow_{\texttt{max}} s'\ \delta$

5 $s\ \delta \rightarrow_{\texttt{min}} s'\ (\delta, \uparrow)$

Rules for state s':

1 $s'\ \delta\ \mu\ \rightarrow_{\texttt{min}} s$

2 $s'\ \delta \rightarrow_{\texttt{min}} s\ \delta$

The provider cell receives at most two copies of symbol μ from the main cell in a single step. Using the first copy of symbol μ, if any, the provider cell performs the "`mod` 2" and "`div` 2" operations on the current multiplicity of symbol δ as described above. Using the second copy of symbol μ, if any, the provider cell reaches a halting configuration.

Let j denote the current multiplicity of symbol δ. The provider cell notifies the results of a "`mod` 2" operation on the value j to the main cell by: (i) not sending any symbol to indicate that j `mod` $2 = 0$ (ii) sending one copy of symbol δ to indicate that j `mod` $2 = 1$, or (iii) sending one copy of symbol ϕ to indicate that $j = 1$, i.e. the all data-bits $b_1b_2\cdots b_\nu$ have been read.

3.2.3 Handling of run-time errors

Section 3.1.4 described the run-time errors of a register machine program. This section describes the manner in which the cells of system Π_M detect and handle the run-time errors. Figure 3.1 illustrates the provider cell's three possible responses, upon receiving a data-bit request (i.e. symbol μ) from the main cell. From these responses, the main cell detects the over-read and under-read errors.

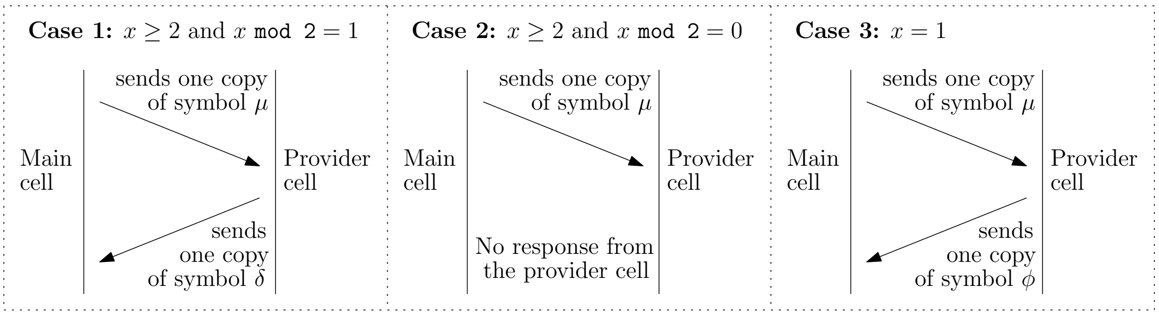

Case 1: $x \geq 2$ and x mod $2 = 1$
Main cell
sends one copy of symbol μ
Provider cell
sends one copy of symbol δ
Case 2: $x \geq 2$ and x mod $2 = 0$
Main cell
sends one copy of symbol μ
Provider cell
No response from the provider cell
Case 3: $x = 1$
Main cell
sends one copy of symbol μ
Provider cell
sends one copy of symbol ϕ

Figure 3.1: Symbol exchanges between the main and provider cells in the `READ` and `HALT` instructions, where x indicates the current multiplicity of symbol δ in the provider cell.

Over-read error

This error can be detected in a `READ` instruction. When the main cell encounters a `READ` instruction, it sends one copy of symbol μ to the provider cell. The purpose of this symbol μ is to request the next unread bit. The main cell interprets the provider cell's responses as follows.

- One copy of symbol δ indicates that the next bit is 1.
- "No response", i.e. case 2, indicates that the next bit is 0.
- One copy of symbol ϕ indicates that the entire input data has been read, i.e. an over-read error. In this case, the main cell enters an infinite loop, hence system Π_M does not halt.

Under-read error

This error can be detected in the `HALT` instruction. When the main cell encounters the `HALT` instruction, it sends one copy of symbol μ to the provider cell. The purpose of this symbol μ is to confirm that the entire input data have been read. The main cell interprets provider's responses as follows.

- One copy of symbol ϕ indicates that the entire input data has been read. In this case, both the main and provider cell do not apply rules any more, hence system Π_M halts.
- One copy of symbol δ or "no response" (i.e. case 2) indicates that there are unread input data, i.e. an under-read error. In this case, the main cell enters an infinite loop, hence system Π_M does not halt.

Illegal branching error

This error can be detected in an `EQ` instruction. If the values of the first two indicated registers (or the first register and the constant) of an `EQ` instruction are the same, then the main cell produces $j+1$ copies of symbol θ, where j is the value of the third register of the `EQ` instruction. Let n denote the number of instructions. If $j+1 \geq n$ then the value of the third register must be greater or equal to n. Thus, the main cell detects (in the "springboard" state s) an illegal branching error and enters an infinite loop. Hence, system Π_M does not halt.

3.2.4 Analysis of system Π_M and remarks

When system Π_M halts, the configuration of (i) the provider cell is $(s, \emptyset)$ and (ii) the main cell is (s_{n-1}, w_m), where $w_m = \{r_i \mid 0 \leq i < n\}^*$. The computational results of a halted system Π_M is the final content of the main cell, where the multiplicity of symbol r_i minus one, $0 \leq i < n$, indicates the value of the register r_i of register machine M.

Theorem 3.3. Simple P systems are universal.

Proof. Section 3.2 presented the details of building a simple P system Π_M that simulates any register machine M [9] (with input data). Thus, simple P systems are universal. $\square$

The number of evolution rules in system Π_M, which simulates a given machine M with n instructions, is proportional to n. The lower and upper bounds on the number of evolution rules of system Π_M are indicated in Proposition 3.4.

Proposition 3.4. For register machine M with n instructions, there are $n_{\texttt{SET}}$ `SET` instructions, $n_{\texttt{ADD}}$ `ADD` instructions, $n_{\texttt{EQ}}$ `EQ` instructions, $n_{\texttt{READ}}$ `READ` instructions and one `HALT` instruction, such that $n = n_{\texttt{EQ}} + n_{\texttt{SET}} + n_{\texttt{ADD}} + n_{\texttt{READ}} + 1$. The corresponding system Π_M contains $n_{\texttt{total}}$ evolution rules, where $n_{\texttt{EQ}} + 2 \cdot n_{\texttt{SET}} + n_{\texttt{ADD}} + 6 \cdot n_{\texttt{READ}} + (n_{\texttt{EQ}}/n_{\texttt{EQ}}) \cdot (n+1) + 12 \leq n_{\texttt{total}} \leq 5 \cdot n_{\texttt{EQ}} + 2 \cdot n_{\texttt{SET}} + 2 \cdot n_{\texttt{ADD}} + 6 \cdot n_{\texttt{READ}} + (n_{\texttt{EQ}}/n_{\texttt{EQ}}) \cdot (n+1) + 12$.

Proof. Recall the set of evolution rules given in Sections 3.2.1 and 3.2.2.

There are two evolution rules for each of instructions (`SET` r_{i_1} r_{i_2}) and (`SET` r_{i_1} k_i). Thus, for $n_{\texttt{SET}}$ number of `SET` instructions, there are $2 \cdot n_{\texttt{SET}}$ evolution rules.

There are two and one evolution rules for each of instructions (`ADD` r_{i_1} r_{i_2}) and (`ADD` r_{i_1} k_i), respectively. If all $n_{\texttt{ADD}}$ instructions are of type (`ADD` r_{i_1} r_{i_2}), then there are $n_{\texttt{ADD}}$ number of evolution rules. If all $n_{\texttt{ADD}}$ instructions are of type (`ADD` r_{i_1} k_i), then there are $2 \cdot n_{\texttt{ADD}}$ number of evolution rules.

There are one, five and four evolution rules for instructions (`EQ` r_{i_1} r_{i_1} r_{i_3}), (`EQ` r_{i_1} r_{i_2} r_{i_3}) and (`EQ` r_{i_1} k_i r_{i_3}), respectively, and $n+1$ additional evolution rules for the "springboard" state. If all $n_{\texttt{EQ}}$ instructions are of type (`EQ` r_{i_1} r_{i_1} r_{i_3}), then there are $n_{\texttt{EQ}}+n+1$ number of evolution rules. If all $n_{\texttt{EQ}}$ instructions are of type (`EQ` r_{i_1} r_{i_2} r_{i_3}), then there are $5 \cdot n_{\texttt{EQ}} + n + 1$ number of evolution rules. If all $n_{\texttt{EQ}}$ instructions are of type (`EQ` r_{i_1} k_i r_{i_3}), then there are $4 \cdot n_{\texttt{EQ}} + n + 1$ number of evolution rules.

The main cell contains (i) six evolution rules for each `READ` instruction and (ii) four evolution rules for the `HALT` instruction. The provider cell contains seven evolution rules, which are associated with `READ` and `HALT` instructions. $\square$

3.2.5 Constructing system Π_{GCD} using direct approach

Definition 3.5 provides the details of a simple P system, obtained via direct approach, which computes the GCD function.

Definition 3.5. (Simple P system for GCD function—using direct implementation) A simple P system, obtained via direct approach, that computes the GCD function is $\Pi_{GCD} = (O, K, \Delta)$, where

1. $O = \{a, b\}$, where the multiplicities of symbols a and b correspond to the GCD algorithm's input values x and y, respectively.
2. $\Delta = \emptyset$.
3. $K = \{\sigma_m\}$. Cell $\sigma_m = (Q, s_{m0}, w_{m0}, R)$, where:
   - $Q = \{s_0\}$.
   - $s_{m0} = s_0$, indicates the initial state.
   - $w_{m0} = \{a^x\ b^y\}$, where the multiplicities of symbols a and b correspond to the GCD algorithm's input values x and y, respectively.
   - R is a set of evolution rules below.

     Rules for state s_0:

     1 $s_0\ ab \rightarrow_{\texttt{max}} s_0\ b$

     2 $s_0\ b \rightarrow_{\texttt{max}} s_0\ a$

Precondition of system Π_{GCD}

The input of Π_{GCD} is the values of x and y, which are represented by the multiplicity of symbols a and b, respectively, i.e. the initial content of σ_m is $a^x\ b^y$.

Postcondition of system Π_{GCD}

The result of Π_{GCD} is the final multiplicity of symbol a in cell σ_m.

Overview of system Π_{GCD}

Starting from the initial configuration, described in Precondition, cell σ_m repeatedly performs the following two steps, until all copies of symbol b are consumed.

1. Rewrite every pair of symbols a and b, into one copy of symbol b (rule 0.1).

2. Rewrite every remaining copy of symbol b into one copy of symbol a (rule 0.2).

Correctness of system Π_{GCD}

Cell σ_m initially contains $x > 0$ copies of symbol a and $y > 0$ copies of symbol b. Assume that, at the end of step t, cell σ_m contains $x' \leq x$ copies of symbol a and $y' \leq y$ copies of symbol. Let $k = \mathtt{min}\{x', y'\}$. At step $t+1$, σ_m rewrites: (i) k copies of symbol a and k copies of symbol b into k copies of symbol b and (ii) $y' - k$ copies of symbol b into $y' - k$ copies of symbol a. Hence, at the end of step $t+1$, cell σ_m ends with $(x'-k)+(y'-k)$ copies of symbol a and k copies of symbol b, which correspond to the computations in lines $2, 4, \ldots, 8$ of the GCD algorithm of Definition 3.2.

3.2.6 Construction of system Π'_{GCD} using indirect approach

The formal definition of a simple P system Π'_{GCD} that simulates the GCD register machine instructions (given in Section 3.1.4), is the system described in the proof of Theorem 3.3 with the following evolution rules. For each instruction line number j, the following tables contain: (i) j-th register machine instruction and (ii) corresponding evolution rules to j-th instruction.

- The following rules in "springboard" state, s, are needed for `EQ` instructions.

  1 $s\ \theta^{30} \rightarrow_{\mathtt{min}} s\ t^{30}$

  2 $s\ \theta^{29} \rightarrow_{\mathtt{min}} s_{28}$

  3 $s\ \theta^{28} \rightarrow_{\mathtt{min}} s_{27}$

  $\vdots$

  30 $s\ \theta \rightarrow_{\mathtt{min}} s_0$

- Rules for initializing registers $a, b, c, d, e, f, g, h, i$ with the instruction line numbers (to be used as targets in branching `EQ` instructions).

Line number	Symbolic instruction	Evolution rules
0	SET a $\mathtt{L_a}$	1 $s_0\ a \rightarrow_{\mathtt{min}} s_1\ a^{10}$ 2 $s_0\ a \rightarrow_{\mathtt{max}} s_1$
1	SET b $\mathtt{L_b}$	1 $s_1\ b \rightarrow_{\mathtt{min}} s_2\ b^{14}$ 2 $s_1\ b \rightarrow_{\mathtt{max}} s_2$
2	SET c $\mathtt{L_c}$	1 $s_2\ c \rightarrow_{\mathtt{min}} s_3\ c^{18}$ 2 $s_2\ c \rightarrow_{\mathtt{max}} s_3$
3	SET d $\mathtt{L_d}$	1 $s_3\ d \rightarrow_{\mathtt{min}} s_4\ d^{21}$ 2 $s_3\ d \rightarrow_{\mathtt{max}} s_4$
4	SET e $\mathtt{L_e}$	1 $s_4\ e \rightarrow_{\mathtt{min}} s_5\ e^{27}$ 2 $s_4\ e \rightarrow_{\mathtt{max}} s_5$
5	SET f $\mathtt{L_f}$	1 $s_5\ f \rightarrow_{\mathtt{min}} s_6\ f^{29}$ 2 $s_5\ f \rightarrow_{\mathtt{max}} s_6$
6	SET g $\mathtt{L_g}$	1 $s_6\ g \rightarrow_{\mathtt{min}} s_7\ g^{31}$ 2 $s_6\ g \rightarrow_{\mathtt{max}} s_7$
7	SET h $\mathtt{L_h}$	1 $s_7\ h \rightarrow_{\mathtt{min}} s_8\ h^{35}$ 2 $s_7\ h \rightarrow_{\mathtt{max}} s_8$
8	SET i $\mathtt{L_i}$	1 $s_8\ i \rightarrow_{\mathtt{min}} s_9\ i^{37}$ 2 $s_8\ i \rightarrow_{\mathtt{max}} s_9$

- Rules for initializing registers x and y, using auxiliary register z, with the first and the second values of the input data, respectively.

Line number	Symbolic instruction	Evolution rules
9	$\mathtt{L_a}$: READ z	1 $s_9\ z \rightarrow_{\mathtt{min}} s_9'\ z\ (\mu, \downarrow)$ 2 $s_9\ z \rightarrow_{\mathtt{max}} s_9'$ 1 $s_9'\ z \rightarrow_{\mathtt{min}} s_9''\ z$ 1 $s_9''\ \phi \rightarrow_{\mathtt{min}} s_9''\ \phi$ 2 $s_9''\ \delta \rightarrow_{\mathtt{min}} s_{10}\ z$ 3 $s_9''\ z \rightarrow_{\mathtt{min}} s_{10}\ z$

Line number	Symbolic instruction	Evolution rules
10	`EQ` z 1 b	1 $s_{10}\ z^3 \rightarrow_{\texttt{min}} s_{11}\ z^3$ 2 $s_{10}\ z^2 \rightarrow_{\texttt{min}} s\ z^2$ 3 $s_{10}\ z \rightarrow_{\texttt{min}} s_{11}\ z$ 4 $s_{10}\ b \rightarrow_{\texttt{max}} s\ b\ \theta$
11	`ADD` x 1	1 $s_{11}\ x \rightarrow_{\texttt{min}} s_{12}\ x^2$
12	`EQ` a a a	1 $s_{12}\ a \rightarrow_{\texttt{max}} s\ a\ \theta$
13	$\mathtt{L_b}$: `READ` z	1 $s_{13}\ z \rightarrow_{\texttt{min}} s'_{13}\ z\ (\mu, \downarrow)$ 2 $s_{13}\ z \rightarrow_{\texttt{max}} s'_{13}$ 1 $s'_{13}\ z \rightarrow_{\texttt{min}} s''_{13}\ z$ 1 $s''_{13}\ \phi \rightarrow_{\texttt{min}} s''_{13}\ \phi$ 2 $s''_{13}\ \delta \rightarrow_{\texttt{min}} s_{14}\ z$ 3 $s''_{13}\ z \rightarrow_{\texttt{min}} s_{14}\ z$
14	`EQ` z 1 c	1 $s_{14}\ z^3 \rightarrow_{\texttt{min}} s_{15}\ z^3$ 2 $s_{14}\ z^2 \rightarrow_{\texttt{min}} s\ z^2$ 3 $s_{14}\ z \rightarrow_{\texttt{min}} s_{15}\ z$ 4 $s_{14}\ c \rightarrow_{\texttt{max}} s\ c\ \theta$
15	`ADD` y 1	1 $s_{15}\ y \rightarrow_{\texttt{min}} s_{16}\ y^2$
16	`EQ` b b b	1 $s_{16}\ b \rightarrow_{\texttt{max}} s\ b\ \theta$

- Rules for checking the condition $x = 0$.

Line number	Symbolic instruction	Evolution rules
17	$\mathtt{L_c}$: `EQ` x 0 i	1 $s_{17}\ x^2 \rightarrow_{\texttt{min}} s_{18}\ x^2$ 2 $s_{17}\ x^1 \rightarrow_{\texttt{min}} s\ x^1$ 3 $s_{17}\ x \rightarrow_{\texttt{min}} s_{18}\ x$ 4 $s_{17}\ i \rightarrow_{\texttt{max}} s\ i\ \theta$

- Rules for checking the condition $x \geq y$.

Line number	Symbolic instruction	Evolution rules
18	SET z x	1 $s_{18}\ z \rightarrow_{\texttt{max}} s_{19}$ 2 $s_{18}\ x \rightarrow_{\texttt{max}} s_{19}\ z\ x$
19	SET w y	1 $s_{19}\ w \rightarrow_{\texttt{max}} s_{20}$ 2 $s_{19}\ y \rightarrow_{\texttt{max}} s_{20}\ w\ y$
20	$\mathsf{L_d}$: EQ z y f	1 $s_{20}\ z\ y \rightarrow_{\texttt{max}} s'_{20}\ z\ y$ 2 $s_{20}\ z \rightarrow_{\texttt{max}} s'_{20}\ z\ \gamma$ 3 $s_{20}\ y \rightarrow_{\texttt{max}} s'_{20}\ z\ \gamma$ 1 $s'_{20}\ \gamma \rightarrow_{\texttt{max}} s_{21}$ 2 $s'_{20}\ f \rightarrow_{\texttt{max}} s\ f\ \theta$
21	ADD z 1	1 $s_{21}\ z \rightarrow_{\texttt{min}} s_{22}\ z^2$
22	ADD w 1	1 $s_{22}\ w \rightarrow_{\texttt{min}} s_{23}\ w^2$
23	EQ x w f	1 $s_{23}\ y\ z \rightarrow_{\texttt{max}} s'_{23}\ y\ z$ 2 $s_{23}\ y \rightarrow_{\texttt{max}} s'_{23}\ y\ \gamma$ 3 $s_{23}\ z \rightarrow_{\texttt{max}} s'_{23}\ y\ \gamma$ 1 $s'_{23}\ \gamma \rightarrow_{\texttt{max}} s_{24}$ 2 $s'_{23}\ e \rightarrow_{\texttt{max}} s\ e\ \theta$
24	EQ y z e	1 $s_{24}\ x\ w \rightarrow_{\texttt{max}} s'_{24}\ x\ w$ 2 $s_{24}\ x \rightarrow_{\texttt{max}} s'_{24}\ x\ \gamma$ 3 $s_{24}\ w \rightarrow_{\texttt{max}} s'_{24}\ x\ \gamma$ 1 $s'_{24}\ \gamma \rightarrow_{\texttt{max}} s_{25}$ 2 $s'_{24}\ f \rightarrow_{\texttt{max}} s\ f\ \theta$
25	EQ d d d	1 $s_{25}\ d \rightarrow_{\texttt{max}} s\ d\ \theta$
26	$\mathsf{L_e}$: SET y x	1 $s_{26}\ y \rightarrow_{\texttt{max}} s_{27}$ 2 $s_{26}\ x \rightarrow_{\texttt{max}} s_{27}\ y\ x$
27	SET x z	1 $s_{27}\ x \rightarrow_{\texttt{max}} s_{28}$ 2 $s_{27}\ z \rightarrow_{\texttt{max}} s_{28}\ x\ z$

- Rules for executing $x := x - y$.

Line number	Symbolic instruction	Evolution rules
28	$\mathtt{L_f}$: SET z y	1 $s_{28}\ z \rightarrow_{\texttt{max}} s_{29}$ 2 $s_{28}\ y \rightarrow_{\texttt{max}} s_{29}\ z\ y$
29	SET w 0	1 $s_{29}\ w \rightarrow_{\texttt{min}} s_{30}\ w^1$ 2 $s_{29}\ w \rightarrow_{\texttt{max}} s_{30}$
30	$\mathtt{L_g}$: EQ x z h	1 $s_{30}\ x\ z \rightarrow_{\texttt{max}} s'_{30}\ x\ z$ 2 $s_{30}\ x \rightarrow_{\texttt{max}} s'_{30}\ x\ \gamma$ 3 $s_{30}\ z \rightarrow_{\texttt{max}} s'_{30}\ x\ \gamma$ 1 $s'_{30}\ \gamma \rightarrow_{\texttt{max}} s_{31}$ 2 $s'_{30}\ h \rightarrow_{\texttt{max}} s\ h\ \theta$
31	ADD z 1	1 $s_{31}\ z \rightarrow_{\texttt{min}} s_{32}\ z^2$
32	ADD w 1	1 $s_{32}\ w \rightarrow_{\texttt{min}} s_{33}\ w^2$
33	EQ g g g	1 $s_{33}\ g \rightarrow_{\texttt{max}} s\ g\ \theta$
34	$\mathtt{L_h}$: SET x w	1 $s_{34}\ x \rightarrow_{\texttt{max}} s_{35}$ 2 $s_{34}\ w \rightarrow_{\texttt{max}} s_{35}\ x\ w$
35	EQ c c c	1 $s_{35}\ c \rightarrow_{\texttt{max}} s\ c\ \theta$

- Rules for the HALT.

Line number	Symbolic instruction	Evolution rules
36	$\mathtt{L_i}$: HALT	1 $s_{36}\ \pi \rightarrow_{\texttt{min}} s'_{36}\ \pi\ (\mu, \downarrow)\ (\mu, \downarrow)$ 1 $s'_{36}\ \pi \rightarrow_{\texttt{min}} s''_{36}\ \pi$ 1 $s''_{36}\ \phi\ \pi \rightarrow_{\texttt{min}} s_{36}$ 2 $s''_{36}\ \pi \rightarrow_{\texttt{min}} s''_{36}\ \pi$

3.3 Summary

This chapter presented a new universality result [20] in P systems, by providing the details of building a simple P system Π_M that simulates any `READ`-enhanced register machine M [9] (with input data). Each constructed system Π_M has the following properties:

- There are two cells—the purpose of having two cells is to designate specific functions to different cells, i.e. the provider cell handles IO operations on the input data and the main cell handles all other operations.
- The number of states and evolution rules are proportional to the number of instructions of a given register machine.
- The number of P system steps required for each register machine instruction during the simulation is constant.

This Turing completeness result indicates that we can compute any computable function using simple P systems. Hence, we have two approaches for obtaining a simple P system, Π, that computes a given computable function:

1. **Direct approach:** Specify a set of evolution rules, symbols and states of system Π and prove its correctness.
2. **Indirect (or translated) approach:** First obtain a register machine, M, that computes the given function. Then, transform machine M into a functionally equivalent system Π.

Using the GCD function, Table 3.1 compares:

- system Π_{GCD}, obtained via direct approach (Section 3.2.5),
- system Π'_{GCD}, obtained via indirect approach (Section 3.2.6) and
- GCD pseudo-code (Definition 3.1).

The comparison uses the following criteria:

- for pseudo-code: the number of lines,
- for simple P systems: the number of evolution rules, states and symbols.

Based on the statistics of Table 3.1, this GCD example seems to support that the direct approach is more suitable than the indirect approach for obtaining a simple P system that computes a given function, with respect to the considered criteria.

Table 3.1: Comparing system Π_{GCD}, system Π'_{GCD} and GCD pseudo-code.

	System Π_{GCD} (Section 3.2.5)	System Π'_{GCD} (Section 3.2.6)	GCD pseudo-code (Definition 3.1)
Number of evolution rules or pseudo-code lines	2 rules	121 rules	7 lines
Number of states	1	48	-
Number of symbols	2	19	-

Chapter 4

Traversal Algorithms in Membrane Systems

Complex problems in distributed computing can typically be subdivided into smaller parts called *phases*; each phase performs a specific task, where the result of this phase is an input for the subsequent phase. Presenting an algorithm as a sequence of phases enables us to analyse each phase separately, which makes it easier to describe the computations performed within an algorithm and to verify the correctness of the algorithm.

In this chapter, several basic traversal P algorithms are presented, which form components of a library of fundamental algorithms in membrane systems. The presented P algorithms are variant algorithms to the standard broadcast and echo algorithms. These P algorithms are compared to broadcast and echo algorithms, with respect to time complexity and program-size complexity (i.e. the number of evolution rules or pseudo-code lines).

Section 4.1 describes the characteristics of the considered traversal algorithms, and indicates the presentation organization of the traversal algorithms. Sections 4.2 and 4.3, present tree and graph traversal algorithms, respectively. Finally, Section 4.4 provides a summary that includes the comparison between P algorithms against broadcast and echo algorithms pseudo-codes, with respect to time and program-size complexities.

4.1 Traversal algorithms

We can consider traversal algorithms on the underlying graph of a given weakly-connected digraph as follows. A traversal algorithm starts from one node of the (underlying) graph, called the *source*, and visit all nodes that are reachable from the source. If a given digraph is a rooted tree, then the source is the *tree root*, otherwise, the source is an *arbitrary* digraph node.

Simple P systems in this chapter satisfy the following constraints: (i) system halts in a finite number of steps, (ii) all cells start from a designated quiescent state, where the source is the only cell with an applicable evolution rule using its initial content, and (iii) all cells have the same set of evolution rules and cell states.

4.1.1 Preliminaries

In the following definitions, for a given digraph (V, Δ), the source is denoted by $s \in V$.

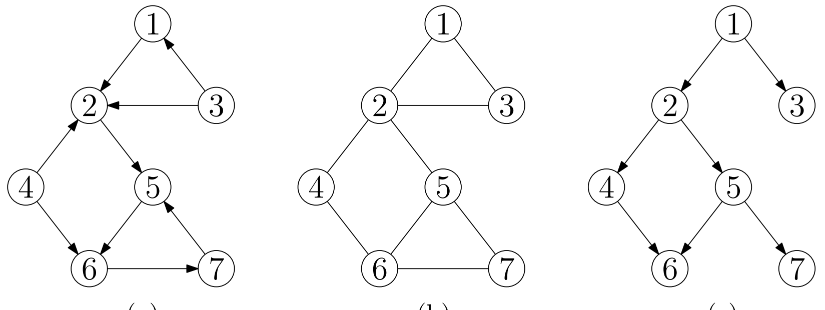
Given a digraph (V, Δ), for $v \in V$, $\texttt{Neighbour}(v) = \Delta(v) \cup \Delta^{-1}(v)$. The relation $\texttt{Neighbour}$ is always symmetric and defines a graph structure, which will be here called the virtual *communication graph* defined by Δ.

The *depth* of a node v, denoted by $\texttt{depth}_s(v) \in \mathbb{N}$, is the length of any shortest path between s and v, over the $\texttt{Neighbour}$ relation. The *eccentricity* of a node v is denoted by $\texttt{ecc}(v)$. Note, the eccentricity of the source $s \in V$ is $\texttt{ecc}(s) = \max\{\texttt{depth}_s(v) \mid v \in V\}$.

Given nodes v and w, (i) v is a *predecessor* of w and (ii) w is a *successor* of v, if $w \in \texttt{Neighbour}(v)$ and $\texttt{depth}_s(w) = \texttt{depth}_s(v) + 1$; a pair (v, w) is called a *depth-increasing arc*. Similarly, a node x is a *peer* of v, if $x \in \texttt{Neighbour}(v)$ and $\texttt{depth}_s(x) = \texttt{depth}_s(v)$. A *terminal* is a node without a successor. For node v, $\texttt{Pred}_s(v) = \{w \mid w \text{ is a predecessor of } v\}$, $\texttt{Peer}_s(v) = \{w \mid w \text{ is a peer of } v\}$ and $\texttt{Succ}_s(v) = \{w \mid w \text{ is a successor of } v\}$. Note, for node v, $\texttt{Pred}_s(v)$, $\texttt{Peer}_s(v)$ and $\texttt{Succ}_s(v)$ are disjoint.

If a given digraph (V, Δ) is a rooted tree and the root is the source, then, for each node $v \in V$, v's children correspond to $\texttt{Succ}_s(v)$ and v's parent corresponds to $\texttt{Pred}_s(v)$.

The depth-increasing arcs form a *spanning breadth-first search DAG* (spanning BFS DAG) rooted at the source s, where each path from s to a node v is a shortest path, over the `Neighbour` relation. The height of v in the spanning BFS DAG is denoted by $\texttt{height}_s(v)$ and the number of shortest paths from s to v in the spanning BFS DAG is denoted by $\texttt{paths}_s(v)$.



(a) (b) (c)

Node	`Neighbours`	$\texttt{Pred}_s$	$\texttt{Peer}_s$	$\texttt{Succ}_s$	$\texttt{depth}_s$	$\texttt{height}_s$	$\texttt{paths}_s$
1	2, 3	–	–	2, 3	0	3	1
2	1, 3, 4, 5	1	3	4, 5	1	2	1
3	1, 2	1	2	–	1	0	1
4	2, 6	2	–	6	2	1	1
5	2, 7, 8	2	–	7, 8	2	1	1
6	4, 5	4, 5	–	–	3	0	2
7	5, 6	5	6	–	3	0	1

(d)

Figure 4.1: (a) A digraph G with the source cell $\sigma_s = \sigma_1$. (b) The communication graph of G, i.e. the underlying graph of G. (c) The spanning BFS DAG of G. (d) A table of node attributes introduced in this section.

Figure 4.1 (a) illustrates a digraph G with the source $s = 1$. Figure 4.1 (b) illustrates the communication graph of G. Figure 4.1 (c) illustrates the spanning BFS DAG of G. For each node v, Figure 4.1 (d) indicates $\texttt{Neighbour}(v)$, $\texttt{Pred}_s(v)$, $\texttt{Peer}_s(v)$, $\texttt{Succ}_s(v)$, $\texttt{depth}_s(v)$, $\texttt{height}_s(v)$ and $\texttt{paths}_s(v)$.

Given a simple P system $\Pi = (O, K, \Delta)$ with the source cell $\sigma_s \in K$, for each cell $\sigma_i \in K$, $\texttt{Pred}_s(i) = \texttt{Pred}_{\sigma_s}(\sigma_i)$, $\texttt{Succ}_s(i) = \texttt{Succ}_{\sigma_s}(\sigma_i)$, $\texttt{Peer}_s(i) = \texttt{Peer}_{\sigma_s}(\sigma_i)$, $\texttt{depth}_s(i) = \texttt{depth}_{\sigma_s}(\sigma_i)$, $\texttt{height}_s(i) = \texttt{height}_{\sigma_s}(\sigma_i)$, $\texttt{paths}_s(i) = \texttt{paths}_{\sigma_s}(\sigma_i)$.

The standard synchronous distributed broadcast [3] and echo [79] algorithms are given in Definitions 4.1 and 4.2, respectively, as pseudo-codes. The time and program-size complexities of these algorithms will be compared against the P algorithms of this chapter.

Definition 4.1. (Spanning tree broadcast algorithm [3])

```
1.  Initially $M$ is in transit from $p_r$ to all its children in the spanning tree.
2.  Code for $p_r$:
3.  1: Upon receiving no message: //first computation event by $p_r$
4.  2:          terminate
5.  Code for $p_i$, $o \leq i \leq n-1$, $i \neq r$:
6.  3: Upon receiving $M$ from parent:
7.  4:          send $M$ to all children
8.  5:          terminate
```

Definition 4.2. (The echo algorithm [79])

```
1.   var $rec_p$ : integer init 0; (*Counts number of received messages *)
2.        $father_p$ : $\mathbb{P}$ init $udef$;
3.   For the initiator:
4.        begin forall $q \in Neigh_p$ do send ⟨tok⟩ to $q$;
5.             while $rec_p < \#Neigh_p$ do
6.                  begin receive ⟨tok⟩; $rec_p := rec_p + 1$ end;
7.             decide
8.   For non-initiators:
9.        begin receive ⟨tok⟩ from neighbour $q$; $father_p := q$; $rec_p := rec_p + 1$;
10.            forall $q \in Neigh_p$, $q \neq father_p$ do send ⟨tok⟩ to $q$;
11.            while $rec_p < \#Neigh_p$ do
12.                 begin receive ⟨tok⟩; $rec_p := rec_p + 1$ end;
13.            send ⟨tok⟩ to $father_p$
14.       end
```

For synchronous distributed algorithms, the broadcast algorithm of Definition 4.1, Attiya and Welch [3], has time complexity of d, where d is the depth of a rooted spanning tree, and the echo algorithm of Definition 4.2 [79], has time complexity of $2d$ [12], where d is the depth of a tree.

4.2 Tree traversal algorithms

This section presents tree traversal algorithms. First, two basic algorithms in distributed computing are presented: a broadcast and an echo algorithm. Then, additional algorithms are presented, derived from the broadcast and echo algorithms, which find structural characteristics of a tree, such as the tree height.

Simple P systems that implement the tree traversal algorithms of this section differ only by their evolution rules. Hence, the *framework* of a simple P system, Ψ, of Definition 4.3, describes all the components of a simple P system, except a set of evolution rules. For each algorithm, a formal description of the corresponding simple P system Π is presented, by completing the framework Ψ (of Definition 4.3) with a set of evolution rules that implements the algorithm.

Definition 4.3. (Framework of a simple P system for tree traversal algorithms) The framework of a simple P system (of order n), for designing the tree traversal algorithms, is $\Psi = (O, K, \Delta)$, where:

1. $O = \{a, b, c, e, h, t, v\}$.

2. $K = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, where $\sigma_s \in K$ is the root cell, i.e. the source cell.

   Each cell $\sigma_i \in K$ has an initial form $\sigma_i = (Q, s_{i0}, w_{i0}, R)$, where:

   - $Q = \{s_0, s_1, s_2\}$, where s_0 is the quiescent state.
   - $s_{i0} = s_0$, is the initial state.
   - $w_{i0} = \begin{cases} b & \text{if } \sigma_i = \sigma_s, \\ \emptyset & \text{if } \sigma_i \neq \sigma_s, \end{cases}$ is the initial content.
   - R is a set of evolution rules, which is given in each algorithm of this section.

3. Δ is a rooted tree.

Figure 4.2 illustrates the tree that will be used for the visual description figures and evolution trace tables.

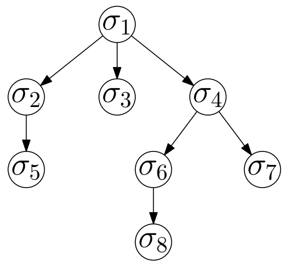


Figure 4.2: A sample rooted tree used as an algorithm illustration.

4.2.1 Algorithm: Broadcast with acknowledgement

This algorithm performs a broadcast operation, initiated from the root cell, which visits all cells and determines the number of children of every cell. A set of evolution rules that transforms the framework Ψ (of Definition 4.3) into a simple P system Π of Algorithm 4.2.1 is given below. Additionally, the description and analysis of system Π are provided.

Precondition of system Π

Each cell $\sigma_i \in K$ starts with the initial configuration described in Definition 4.3.

Postcondition of system Π

When Π halts, the configuration of cell $\sigma_i \in K$ is (s_2, w_i), where:

- $|w_i|_a = 1$.
- $|w_i|_c = |\mathtt{Succ}_s(i)|$, i.e. the number of σ_i's children.

States and symbols of system Π

Symbol b represents broadcast symbol and symbol c represents acknowledgement symbol. Symbol a indicates the “visited” status. State s_0 is a quiescent state, where a cell remains idle until it receives symbol b. State s_1 is an auxiliary state and state s_2 is a terminal state.

Evolution rules of system Π

0. Rules for state s_0:

    1. $s_0\ b \rightarrow_{\texttt{min}} s_1\ a\ (c, \uparrow)\ (b, \downarrow)$

1. Rules for state s_1:

    1. $s_1\ a \rightarrow_{\texttt{min}} s_2\ a$

Overview of system Π

Initially, the root cell σ_s has one copy of broadcast symbol b. At step 1, σ_s: (i) sends down one copy of symbol b to each of its children and (ii) rewrites the symbol b into symbol a (rule 0.1). After cell $\sigma_j \neq \sigma_s$ receives symbol b, σ_j: (i) *acknowledges* the sender (i.e. its parent) by sending up one copy of acknowledgement symbol c, (ii) *forwards* one copy of symbol b to each of its children, if any, and (iii) rewrites the received symbol b into symbol a (rule 0.1).

Cell σ_i receives one copy of symbol b from its parent at step $\texttt{depth}_s(i)$ (Proposition 4.4). Further, cell σ_i receives $|\texttt{Succ}_s(i)|$ copies of symbol c at step $\texttt{depth}_s(i) + 2$ (Proposition 4.6). A cell that does not receive any symbol c at step $\texttt{depth}_s(i) + 2$ is a tree leaf (Proposition 4.7).

Proposition 4.4. At step $\texttt{depth}_s(i)$, cell $\sigma_i \neq \sigma_s$ receives one copy of symbol b from its parent. At step $\texttt{depth}_s(i)+1$, σ_i sends one copy of symbol b to each of its children.

Proof. Proof by induction on $m = \texttt{depth}_s(i) \geq 1$. At step 1, the root σ_s sends down one copy of symbol b to each of its children. Hence, at step 1, each cell σ_f in depth 1 receives one copy of symbol b. Then, at step 2, cell σ_f sends (i) one copy of symbol c to its parent and (ii) one copy of symbol b to each of its children.

Assume that the induction hypothesis holds for each cell σ_j at depth m. Consider cell σ_k in depth $m' = m + 1$. By induction hypothesis, at step $m + 1$, each $\sigma_j \in \texttt{Pred}_s(k)$ sends (i) one copy of symbol c to its parent and (ii) one copy of symbol b to each of its children (which includes σ_k). Thus, at step $m' = m + 1$, cell σ_k receives one copy of symbol b from σ_j. Then, at step $m' + 1$, cell σ_k sends (i) one copy of symbol c to its parent and (ii) one copy of symbol b to each of its children, if any. $\square$

Proposition 4.5. At step $\mathtt{depth}_s(i)+1$, cell σ_i makes transition $s_0 \Rightarrow s_1$, and reaches configuration (s_1, a).

Proof. As indicated in Proposition 4.4, at step $\mathtt{depth}_s(i)$, cell σ_i obtains one copy of symbol b and reaches configuration (s_0, b). At step $\mathtt{depth}_s(i) + 1$, cell σ_i:

- Rewrites one copy of symbol b, received at step $\mathtt{depth}_s(i)$, into one copy of symbol a.
- Enters state s_1, from the transition $s_0 \Rightarrow s_1$.

□

Proposition 4.6. At step $\mathtt{depth}_s(i)+2$, cell σ_i makes transition $s_1 \Rightarrow s_2$, and reaches configuration (s_2, ac^k), where $k = |\mathtt{Succ}_s(i)|$.

Proof. As indicated in Proposition 4.5, at step $\mathtt{depth}_s(i)+1$, cell reaches configuration (s_1, a). At step $\mathtt{depth}_s(i) + 2$, cell σ_i:

- Receives one copy of symbol c from each of its children, i.e. each $\sigma_h \in \mathtt{Succ}_s(i)$, such that, in total, σ_i receives $|\mathtt{Succ}_s(i)|$ copies of symbol c.

  From Proposition 4.4, at step $\mathtt{depth}_s(h)+1$, each cell $\sigma_h \in \mathtt{Succ}_s(i)$ sends up one copy of symbol c to each its parent. Hence, at step $\mathtt{depth}_s(h)+1 = \mathtt{depth}_s(i)+2$, σ_i receives one copy of symbol c from each $\sigma_h \in \mathtt{Succ}_s(i)$.
- Enters state s_2, from the transition $s_1 \Rightarrow s_2$.

□

Proposition 4.7. If cell σ_i does not receive a copy of symbol c at step $\mathtt{depth}_s(i) + 2$, then σ_i is a tree leaf.

Proof. Follows from Proposition 4.6. □

Table 4.1 contains the traces of system Π of Figure 4.2. Figure 4.3 provides a visual description of system Π of Figure 4.2. In Table 4.1 and Figure 4.3, the final multiplicity of symbol c in each cell represents the number of that cell's children.

Table 4.1: The traces of system Π of Algorithm 4.2.1 of Figure 4.2. The final multiplicity of symbol c in cell σ_i, $1 \leq i \leq 8$, represents the number of σ_i's children. For example, cell σ_4 has two children, cells σ_6 and σ_7. Hence, at step 5, σ_4 contains two copies of symbol c.

Step	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8
0	$s_0\ b$	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	$s_1\ a$	$s_0\ b$	$s_0\ b$	$s_0\ b$	s_0	s_0	s_0	s_0
2	$s_2\ ac^3$	$s_1\ a$	$s_1\ a$	$s_1\ a$	$s_0\ b$	$s_0\ b$	$s_0\ b$	s_0
3	$s_2\ ac^3$	$s_2\ ac$	$s_2\ a$	$s_2\ ac^2$	$s_1\ a$	$s_1\ a$	$s_1\ a$	$s_0\ b$
4	$s_2\ ac^3$	$s_2\ ac$	$s_2\ a$	$s_2\ ac^2$	$s_2\ a$	$s_2\ ac$	$s_2\ a$	$s_1\ a$
5	$s_2\ ac^3$	$s_2\ ac$	$s_2\ a$	$\boldsymbol{s_2\ ac^2}$	$s_2\ a$	$s_2\ ac$	$s_2\ a$	$s_2\ a$

Correctness and complexity of system Π

Proposition 4.8 indicates the correctness and time complexity of system Π. Proposition 4.9 indicates the message complexity of system Π.

Proposition 4.8. System Π halts in $\texttt{height}_s(s)+2$ steps and the final configuration of each cell corresponds to the postcondition.

Proof. At step $\texttt{depth}_s(i) + 2$, cell $\sigma_i \in K$ reaches the configuration (s_2, ac^k), where $k = |\texttt{Succ}_s(i)|$, as indicated in Proposition 4.6.

There are no rules in state s_2, hence, cell σ_i cannot evolve once it enters state s_2. For a farthest cell σ_f (with respect to the source cell σ_s), $\texttt{depth}_s(f) \geq \texttt{depth}_s(g)$, for all $\sigma_g \in K$, such that $\texttt{depth}_s(f) = \texttt{height}_s(s)$. Cell σ_f enters state s_2 at step $\texttt{depth}_s(f) + 2 = \texttt{height}_s(s) + 2$.

Therefore, system Π halts at step $\texttt{height}_s(s) + 2$, and the final configuration of each cell $\sigma_i \in K$ is (s_2, ac^k), where $k = |\texttt{Succ}_s(i)|$. □

Proposition 4.9. The total number of symbols that are transferred between cells of system Π is $2 \cdot |\Delta|$.

Proof. In each tree arc $(\sigma_j, \sigma_k) \in \Delta$, (i) cell σ_j sends down one copy of symbol b to σ_k and (ii) cell σ_k sends up one copy of symbol c to σ_j. Thus, the total number of

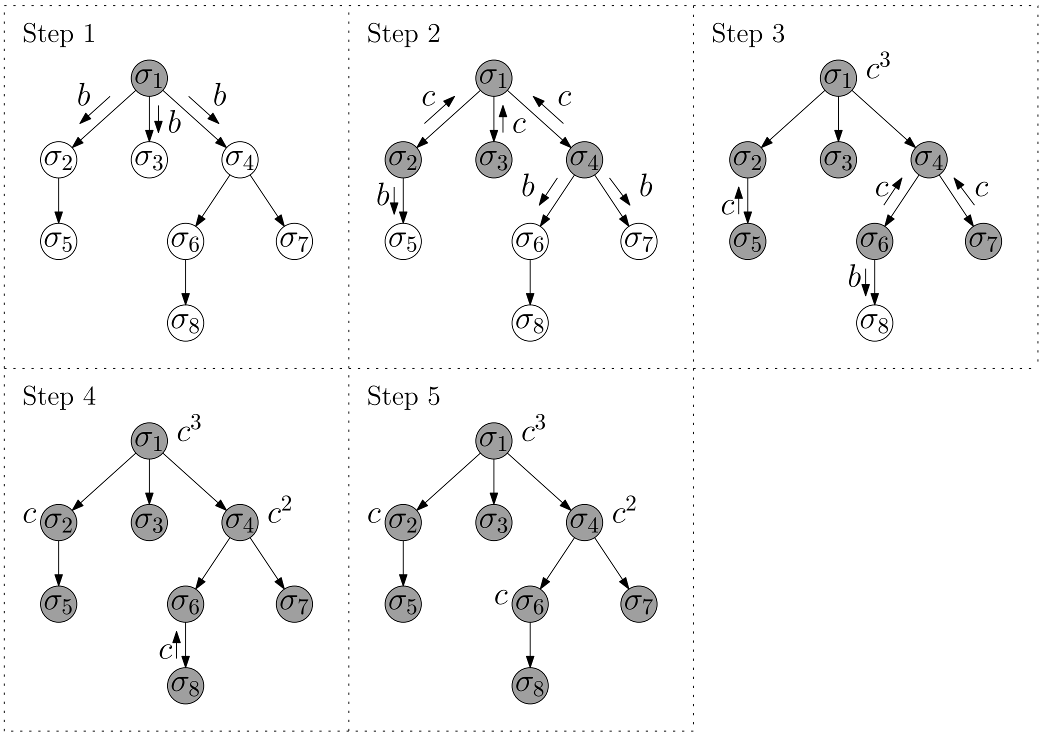


Figure 4.3: The propagation of symbols b and c in Algorithm 4.2.1, for the tree of Figure 4.2. Cells that have received symbol b (i.e. reached by the broadcast) are shaded. The multiplicity of symbol c in cell σ_i, $1 \le i \le 8$, represents the number of σ_i's children.

symbols b and c that are transferred between cells is $2 \cdot |\Delta|$. □

4.2.2 Algorithm: Echo

In this echo algorithm, the root initiates a search that visits all cells of a tree and returns back to the root, in the reverse order in which the cells are visited. A set of evolution rules that transforms the framework Ψ (of Definition 4.3) into a simple P system Π of Algorithm 4.2.2 is present below. Additionally, the description and analysis of system Π are provided.

Precondition of system Π

Each cell $\sigma_i \in K$ starts with the initial configuration described in Definition 4.3.

Postcondition of system Π

When Π halts, the configuration of cell σ_i is $C = (s_3, \emptyset)$. Further, each cell reaches configuration C, after all its children have reached C.

States and symbols of system Π

Symbols b and c represent the broadcast and acknowledgement symbols (in Phase I), respectively, and symbol e represents the convergecast symbol (in Phase II). Symbol a is used to check if a cell has received symbol e from all its children (in Phase II). Symbol t indicates that a cell has sent symbol e to its parent (in Phase II). In state s_2, each cell *checks*, if it has received symbol e from all its children. State s_3 is a terminal state. The meaning of states s_0 and s_1 are described in Algorithm 4.2.1.

Evolution rules of system Π

Rules used in Phase I (Algorithm 4.2.1):

0. Rules for state s_0:
    1. $s_0\ b \rightarrow_{\texttt{min}} s_1\ a\ (c, \uparrow)\ (b, \downarrow)$
1. Rules for state s_1:
    1. $s_1\ a \rightarrow_{\texttt{min}} s_2\ a$

Rules used in Phase II (convergecast):

2. Rules for state s_2:
    1. $s_2\ t \rightarrow_{\texttt{min}} s_3$
    2. $s_2\ ce \rightarrow_{\texttt{max}} s_2$
    3. $s_2\ ac \rightarrow_{\texttt{min}} s_2\ ac$
    4. $s_2\ a \rightarrow_{\texttt{min}} s_2\ t\ (e, \uparrow)$

Overview of system Π

Each cell of system Π progresses through two conceptual phases, Phase I (Algorithm 4.2.1) and Phase II (convergecast), independent of other cells.

Phase I (Algorithm 4.2.1):

- The source cell (i.e. tree root) starts Phase I at the beginning of the algorithm. Each non-source cell starts Phase I, when it receives broadcast symbol b.

- The root cell σ_s (i) sends down one copy of symbol b to each of its children and (ii) produces one copy of symbol a (rule 0.1).
- When cell $\sigma_j \neq \sigma_s$ receives symbol b, σ_j: (i) sends up one copy of acknowledgement symbol c to its parent, (ii) sends down one copy of symbol b to each of its children, if any, and (iii) rewrites the received symbol b into symbol a (rule 0.1).
- At the end of Phase I, each cell contains one copy of symbol a and $|\texttt{Succ}_s(i)|$ copies of symbol c, as indicated in the postcondition of Algorithm 4.2.1. Note, each leaf cell has zero copies of symbol c.

Phase II (convergecast):

- Each cell starts Phase II, immediately after completing Phase I.
- Each leaf cell (i.e. a cell that has not received any symbol c in Phase I) (i) sends up one copy of convergecast symbol e to its parent and (ii) rewrites symbol a into symbol t (rule 2.4).
- Each cell consumes one copy of symbol c with one copy of symbol e (rule 2.2). When a non-leaf cell σ_j consumes all copies of symbol c (i.e. σ_j has received symbol e from all its children in Phase II), σ_j: (i) sends one copy of symbol e to its parent, if any, and (ii) rewrites symbol a into symbol t (rule 2.4).
    - In cell σ_i, having at least one copy of symbol c indicates that σ_i has not received symbol e from all its children.
    - In each step, rule 2.3 (which has higher priority than rule 2.4) rewrites multiset ac into multiset ac, such that, if at least one copy of symbol c exists, then symbol a will not be available for rule 2.4 in the current step.
    - Therefore, in each step, rule 2.3 prevents each cell σ_i from sending symbol e to its parent, until σ_i receives symbol e from all its children.
- Each cell consumes symbol t and enters the terminal state s_3 (rule 2.1).

Table 4.2 contains the traces of system Π of Figure 4.2. Figure 4.4 provides a visual description of system Π of Figure 4.2.

Table 4.2: The traces of the system Π of Algorithm 4.2.2, for the tree of Figure 4.2. For each column of σ_i, $1 \leq i \leq 8$, the first shaded table cell indicates the start of Phase I, the second shaded table cell indicates the start of Phase II (equivalently, the end of Phase I) and the third shaded table cell indicates the end of Phase II.

Step	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8
0	$s_0\ b$	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	$s_1\ a$	$s_0\ b$	$s_0\ b$	$s_0\ b$	s_0	s_0	s_0	s_0
2	$s_2\ ac^3$	$s_1\ a$	$s_1\ a$	$s_1\ a$	$s_0\ b$	$s_0\ b$	$s_0\ b$	s_0
3	$s_2\ ac^3$	$s_2\ ac$	$s_2\ a$	$s_2\ ac^2$	$s_1\ a$	$s_1\ a$	$s_1\ a$	$s_0\ b$
4	$s_2\ ac^3e$	$s_2\ ac$	$s_2\ t$	$s_2\ ac^2$	$s_2\ a$	$s_2\ ac$	$s_2\ a$	$s_1\ a$
5	$s_2\ ac^2$	$s_2\ ace$	s_3	$s_2\ ac^2e$	$s_2\ t$	$s_2\ ac$	$s_2\ t$	$s_2\ a$
6	$s_2\ ac^2e$	$s_2\ t$	s_3	$s_2\ ac$	s_3	$s_2\ ace$	s_3	$s_2\ t$
7	$s_2\ ac$	s_3	s_3	$s_2\ ace$	s_3	$s_2\ t$	s_3	s_3
8	$s_2\ ace$	s_3	s_3	$s_2\ t$	s_3	s_3	s_3	s_3
9	$s_2\ t$	s_3	s_3	s_3	s_3	s_3	s_3	s_3
10	s_3	s_3	s_3	s_3	s_3	s_3	s_3	s_3

Proposition 4.10 indicates the step in which each cell receives convergecast symbols from all its children.

Proposition 4.10. By step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 2$, each non-leaf cell σ_i receives $|\texttt{Succ}_s(i)|$ copies of convergecast symbol e from all its children. At step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 3$, each non-source cell σ_i sends up one copy of symbol e to its parent.

Proof. Proof by induction, on cell σ_i's height, $h = \texttt{height}_s(i)$.

In the base case, where $h = 0$, cell σ_i is a leaf cell. Clearly, cell σ_i has no children and therefore does not receive any acknowledgement symbol c at step $\texttt{depth}_s(i) + 2$. Thus, at step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 3 = \texttt{depth}_s(i) + 3$, cell σ_i sends one copy of convergecast symbol e to its parent.

Assume that the induction hypothesis holds for each cell σ_k at height $\texttt{height}_s(k) \leq h$. Consider a non-leaf cell σ_i at height $\texttt{height}_s(i) = h + 1$. Each σ_i's children,

σ_j, has height $\texttt{height}_s(j) \leq h$, thus it satisfies the induction hypothesis. Cell σ_j sends up one copy of symbol e to cell σ_i at step $\texttt{depth}_s(j) + 2 \cdot \texttt{height}_s(j) + 3$. For cell σ_i, $\texttt{depth}_s(i) = \texttt{depth}_s(j) - 1$ for each child σ_j, and $\max\{\texttt{height}_s(j) \mid \sigma_j \text{ is a child of } \sigma_i\} = \texttt{height}_s(i) - 1$. Thus, by step $(\texttt{depth}_s(i) + 1) + 2(\texttt{height}_s(i) - 1) + 3 = \texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 2$, cell σ_i receives $k_i \geq 1$ copies of symbol e from its children, where k_i corresponds to the number of σ_i's children. Then, at step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 3$, cell σ_i sends up one copy of symbol e to its parent. □

Proposition 4.11. *From step $\texttt{depth}_s(i) + 3$ until step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 2$, cell σ_i makes transition $s_2 \Rightarrow s_2$ and reaches configuration $(s_2, ac^h e^f)$, where $1 \leq h \leq |\texttt{Succ}_s(i)|$ and $0 \leq f \leq h$.*

Proof. From Proposition 4.6, at step $\texttt{depth}_s(i) + 2$, cell σ_i reaches configuration (s_2, ac^k), where $k = |\texttt{Succ}_s(i)|$. From step $\texttt{depth}_s(i) + 3$, cell σ_i consumes each copy of symbol c together with one copy of symbol e.

As indicated in Proposition 4.10, cell σ_i receives, in total, $|\texttt{Succ}_s(i)|$ copies of symbol e by step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 2$.

Therefore, until step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 2$, cell σ_i reaches configuration $(s_2, ac^h e^f)$, where $1 \leq h \leq |\texttt{Succ}_s(i)|$ and $0 \leq f \leq h$. □

Proposition 4.12. *At step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 3$, cell σ_i makes transition $s_2 \Rightarrow s_2$ and reaches configuration (s_2, t).*

Proof. From Proposition 4.10, cell σ_i receives $|\texttt{Succ}_s(i)|$ copies of symbol e by step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 2$. At step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 3$, cell σ_i:

- Consumes all $|\texttt{Succ}_s(i)|$ copies of symbol c, received at step $\texttt{depth}_s(i)+2$ (Proposition 4.6), together with $|\texttt{Succ}_s(i)|$ copies of symbol e.
- Rewrites one copy of symbol a into one copy of symbol t.
- Remains at state s_2, from the transition $s_2 \Rightarrow s_2$.

□

Proposition 4.13. *At step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 4$, cell σ_i makes transition $s_2 \Rightarrow s_3$, and reaches configuration $(s_3, \emptyset)$.*

Proof. From Proposition 4.12, at step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 3$, cell σ_i reaches configuration (s_2, t). At step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 4$, cell σ_i:

- Consumes one copy of symbol t, produced at step $\texttt{depth}_s(i)+2\cdot\texttt{height}_s(i)+3$.
- Enters state s_3, from the transition $s_2 \Rightarrow s_3$.

□

Correctness and complexity of system Π

Proposition 4.14 indicates the correctness and time complexity of system Π. Proposition 4.15 indicates the message complexity of system Π.

Proposition 4.14. System Π halts in $2 \cdot \texttt{height}_s(s) + 4$ steps and the final configuration of each cell corresponds to the postcondition.

Proof. At step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 4$, cell $\sigma_i \in K$ reaches the configuration $C = (s_3, \emptyset)$, as indicated in Proposition 4.13. For each cell $\sigma_j \in \texttt{Succ}_s(i)$, $\texttt{depth}_s(i) + 1 = \texttt{depth}_s(j)$. Thus, cell σ_i reaches configuration C, after all its children have reaches configuration C.

There are no rules in state s_3, hence, cell σ_i cannot evolve once it enters state s_3. For a farthest cell σ_f (with respect to the source cell σ_s), $\texttt{depth}_s(f) \geq \texttt{depth}_s(g)$, for all $\sigma_g \in K$, such that $\texttt{depth}_s(f) = \texttt{height}_s(s)$. Cell σ_f enters state s_3 at step $\texttt{depth}_s(f) + 2 \cdot \texttt{height}_s(f) + 4 = 2 \cdot \texttt{depth}_s(f) + 4 = 2 \cdot \texttt{height}_s(s) + 4$. Therefore, system Π halts at step $2 \cdot \texttt{height}_s(s) + 4$. □

Proposition 4.15. The total number of symbols that are transferred between cells of Π is $3 \cdot |\Delta|$.

Proof. In each tree arc $(\sigma_j, \sigma_k) \in \Delta$, (i) cell σ_j sends down one copy of symbol b to σ_k in Phase I, (ii) cell σ_k sends up one copy of symbol c to σ_j in Phase I and (iii) cell σ_k sends up one copy of symbol e to σ_j in Phase II. Thus, the total number of symbols that are transferred between cells is $3 \cdot |\Delta|$. □

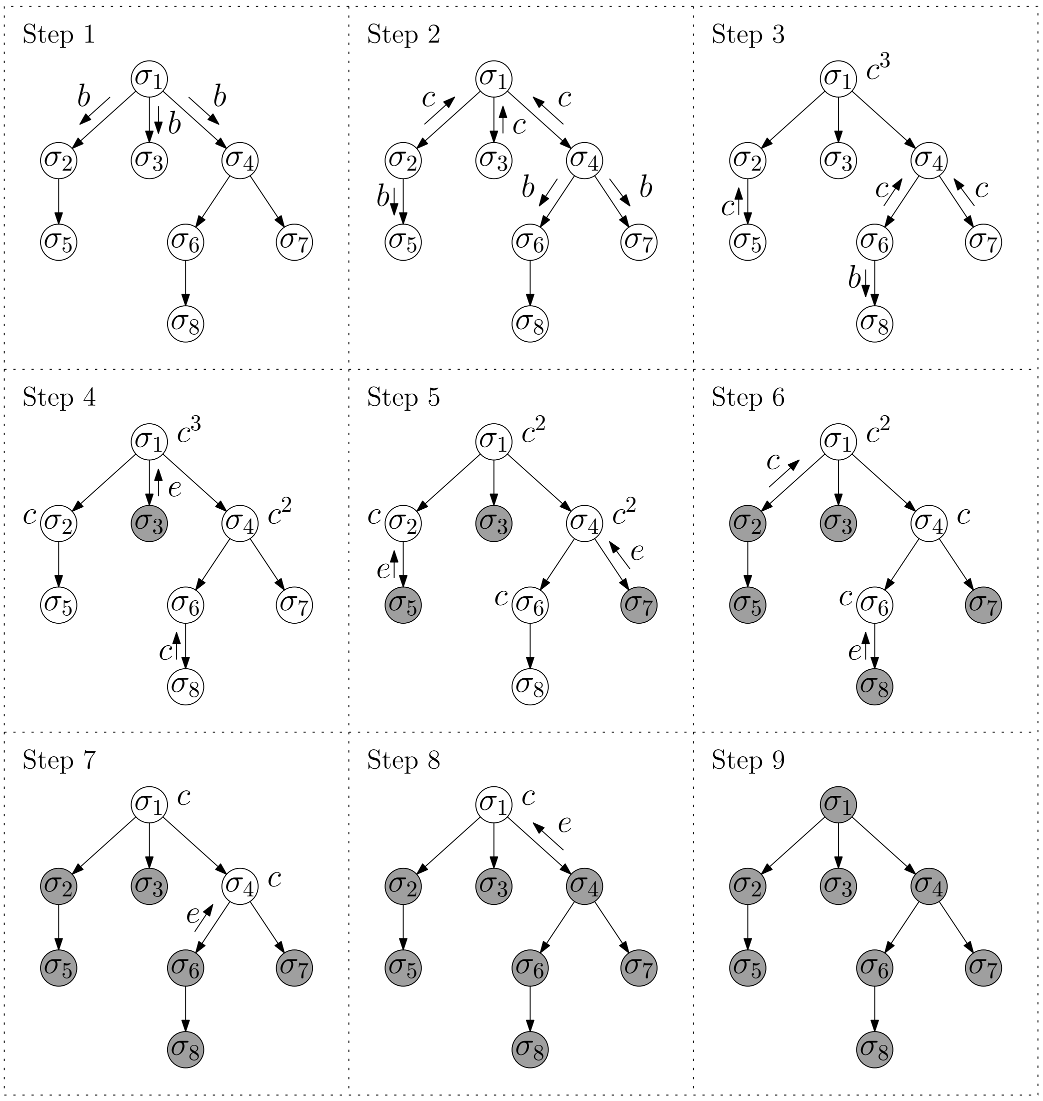


Figure 4.4: The propagation of symbols b, c and e in Algorithm 4.2.2, for the tree of Figure 4.2. The multiplicity of symbol c (that initially incremented according to acknowledgement symbols received from children in Phase I) decrements according to the number of the received convergecast symbol e in Phase II; thus, the multiplicity of symbol c represents the number of remaining children that have not sent a convergecast symbol. Cells that have received symbol e from all their children, if any, are shaded. A cell sends up one copy of symbol e to its parent, after it consumes all copies of symbol c.

4.2.3 Algorithm: Tree height

This algorithm, derived from Algorithm 4.2.2, computes the height of each cell in a rooted tree. A set of evolution rules that transforms the framework Ψ (of Definition 4.3) into a simple P system Π of Algorithm 4.2.3 is presented. Then, the description and analysis of system Π are provided.

Precondition of system Π

Each cell $\sigma_i \in K$ starts with the initial configuration described in Definition 4.3.

Postcondition of system Π

When Π halts, the configuration of cell $\sigma_i \in K$ is (s_3, w_i), where:

- $|w_i|_h = \texttt{height}_s(i)$, the height of σ_i.

States and symbols of system Π

The final multiplicity of symbol h in a cell indicates the height of that cell. Symbol v is an auxiliary symbol used to produce symbol h. The meaning of the remaining states and symbols are described Algorithm 4.2.2.

Evolution rules of system Π

The set of evolution rules below computes the height of each cell $\sigma_i \in K$. Note, the evolution rules below are the evolution rules used in Algorithms 4.2.2 with the following changes: rule 2.3 is modified to accumulate $2 \cdot \texttt{height}_s(i)$ copies of symbol v, and rule 2.5 is added to rewrite every two copies of symbol v into one copy of symbol h.

Rules used in Phase I:

0. Rules for state s_0:
   1. $s_0\ b \rightarrow_{\texttt{min}} s_1\ a\ (c, \uparrow)\ (b, \downarrow)$
1. Rules for state s_1:
   1. $s_1\ a \rightarrow_{\texttt{min}} s_2\ a$

Rules used in Phase II:

2. Rules for state s_2:
   1. $s_2\ t \rightarrow_{\texttt{min}} s_3$
   2. $s_2\ ce \rightarrow_{\texttt{max}} s_2$
   3. $s_2\ ac \rightarrow_{\texttt{min}} s_2\ acv$
   4. $s_2\ a \rightarrow_{\texttt{min}} s_2\ t\ (e, \uparrow)$
   5. $s_2\ vv \rightarrow_{\texttt{max}} s_3\ h$

Overview of system Π

Recall the Phase I and Phase II of Algorithm 4.2.2. For cell σ_i, consider two steps, t_i and t'_i, where:

- $t_i = \texttt{depth}_s(i)+2$, i.e. two steps after σ_i receives the broadcast symbol in Phase I (Proposition 4.4),
- $t'_i = \texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 2$, i.e. the step in which σ_i has received the convergecast symbols from all its children in Phase II (as indicated in Proposition 4.10).

Note that, $t'_i = t_i + 2 \cdot \texttt{height}_s(i)$, where $\texttt{height}_s(i)$ is the height of cell σ_i.

Cell $\sigma_i \in K$ obtains $\texttt{height}_s(i)$ copies of symbol h in the following manner. First, cell σ_i accumulates $2 \cdot \texttt{height}_s(i)$ copies of symbol v by producing one copy of symbol h (rule 2.3) in each step, from step $t_i + 1$ until step t'_i (both inclusive). Then, cell σ_i rewrites every two copies of symbol v into one copy of symbol h (rule 2.5).

Table 4.3 contains the trace of system Π, for the tree of Figure 4.2. Figure 4.5 provides a visual description of Π, for the tree of Figure 4.2. In Table 4.3 and Figure 4.5, the final multiplicity of symbol h in each cell represents its tree height.

Proposition 4.16. From step $\texttt{depth}_s(i) + 3$ until step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 2$, cell $\sigma_i \in K$ remains in state s_2 and produces one copy of symbol v in each step.

Table 4.3: The traces of the system Π of Algorithm 4.2.3 that computes the height of a cell, for the tree of Figure 4.2. The final multiplicity of symbol h in cell σ_i, $1 \leq i \leq 8$, corresponds to the height of σ_i. The height of σ_1 is three. Thus, at step 10, σ_1 contains three copies of symbol h.

Step	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8
0	$s_0\ b$	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	$s_1\ a$	$s_0\ b$	$s_0\ b$	$s_0\ b$	s_0	s_0	s_0	s_0
2	$s_2\ ac^3$	$s_1\ a$	$s_1\ a$	$s_1\ a$	$s_0\ b$	$s_0\ b$	$s_0\ b$	s_0
3	$s_2\ ac^3v$	$s_2\ ac$	$s_2\ a$	$s_2\ ac^2$	$s_1\ a$	$s_1\ a$	$s_1\ a$	$s_0\ b$
4	$s_2\ ac^3ev^2$	$s_2\ acv$	$s_2\ t$	$s_2\ ac^2v$	$s_2\ a$	$s_2\ ac$	$s_2\ a$	$s_1\ a$
5	$s_2\ ac^2v^3$	$s_2\ acev^2$	s_3	$s_2\ ac^2ev^2$	$s_2\ t$	$s_2\ acv$	$s_2\ t$	$s_2\ a$
6	$s_2\ ac^2ev^4$	$s_2\ tv^2$	s_3	$s_2\ acv^3$	s_3	$s_2\ acev^2$	s_3	$s_2\ t$
7	$s_2\ acv^5$	$s_3\ h$	s_3	$s_2\ acev^4$	s_3	$s_2\ tv^2$	s_3	s_3
8	$s_2\ acev^6$	$s_3\ h$	s_3	$s_2\ tv^4$	s_3	$s_3\ h$	s_3	s_3
9	$s_2\ tv^6$	$s_3\ h$	s_3	$s_3\ h^2$	s_3	$s_3\ h$	s_3	s_3
10	$\boldsymbol{s_3\ h^3}$	$s_3\ h$	s_3	$s_3\ h^2$	s_3	$s_3\ h$	s_3	s_3

Proof. At step $\texttt{depth}_s(i) + 2$, each cell σ_i enters state s_2 with $|\texttt{Succ}_s(i)|$ copies of symbol c (Proposition 4.6). Cell σ_i consumes each copy of symbol c together with one copy of symbol e.

From step $\texttt{depth}_s(i) + 3$, if σ_i contains at least one copy of symbol c, then σ_i: (i) remains in state s_2 and (ii) produces one copy of symbol v in each step.

As indicated in Proposition 4.10, by step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 2$, each cell σ_i receives, in total, $|\texttt{Succ}_s(i)|$ copies of symbol e from its children.

Thus, from step $\texttt{depth}_s(i) + 3$ until step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 2$, σ_i remains in state s_2 and produces one copy of symbol v in each step. □

Proposition 4.17. At step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 3$, cell σ_i makes transition $s_2 \Rightarrow s_2$, and reaches configuration (s_2, tv^k), where $k = 2 \cdot \texttt{height}_s(s)$.

Proof. From Proposition 4.12, at step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 2$, cell σ_i reaches configuration $C = (s_2, ac^he^f)$, where $1 \leq h \leq |\texttt{Succ}_s(i)|$ and $0 \leq f \leq h$. Note, cell σ_i

receives one copy of symbol e from all its children by step $\texttt{depth}_s(i)+2\cdot\texttt{height}_s(i)+2$ (Proposition 4.10). Hence, $C = (s_2, ac^he^h)$. Further, from Proposition 4.16, by step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 2$, cell σ_i accumulates $2 \cdot \texttt{height}_s(s)$ copies of symbol v. Thus, at step $\texttt{depth}_s(i)+2\cdot\texttt{height}_s(i)+2$, cell σ_i reaches configuration $(s_2, ac^he^hv^k)$, where $k = 2 \cdot \texttt{height}_s(s)$.

At step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 3$, cell σ_i:

- Consumes h copies of symbol c together with h copies of symbol e.
- Rewrites one copy of symbol a into one copy of symbol t.
- Remains at state s_2, from the transition $s_2 \Rightarrow s_2$.

□

Proposition 4.18. At step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 4$, cell σ_i makes transition $s_2 \Rightarrow s_3$ and reaches configuration (s_3, h^j), where $j = \texttt{height}_s(s)$.

Proof. From Proposition 4.17, at step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 3$, cell σ_i reaches configuration (s_2, tv^k), where $k = 2\cdot\texttt{height}_s(s)$. At step $\texttt{depth}_s(i)+2\cdot\texttt{height}_s(i)+4$, cell σ_i:

- Consumes one copy of symbol t.
- Rewrites every two copies of symbol v into one copy of symbol h.
- Enters state s_3, from the transition $s_2 \Rightarrow s_3$.

□

Correctness and complexity of system Π

Proposition 4.19 indicates the correctness and time complexity of system Π. Proposition 4.20 indicates the message complexity of system Π.

Proposition 4.19. System Π halts in $2 \cdot \texttt{height}_s(s) + 4$ steps and the final configuration of each cell corresponds to the postcondition.

Proof. At step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 4$, cell $\sigma_i \in K$ reaches the configuration $C = (s_3, h^j)$, where $j = \texttt{height}_s(s)$, as indicated in Proposition 4.18.

There are no rules in state s_3, hence, cell σ_i cannot evolve once it enters state s_3. For a farthest cell σ_f (with respect to the source cell σ_s), $\texttt{depth}_s(f) \geq \texttt{depth}_s(g)$, for all $\sigma_g \in K$, such that $\texttt{depth}_s(f) = \texttt{height}_s(s)$. Cell σ_f enters state s_3 at step $\texttt{depth}_s(f) + 2 \cdot \texttt{height}_s(f) + 4 = 2 \cdot \texttt{depth}_s(f) + 4 = 2 \cdot \texttt{height}_s(s) + 4$. Therefore, system Π halts at step $2 \cdot \texttt{height}_s(s) + 4$. □

Proposition 4.20. The total number of symbols that are exchanged between cells of Π is $3 \cdot |\Delta|$.

Proof. This algorithm is derived from Algorithm 4.2.2 and cells in this algorithm do not send any additional symbols. Thus, this proof follows from Proposition 4.15. □

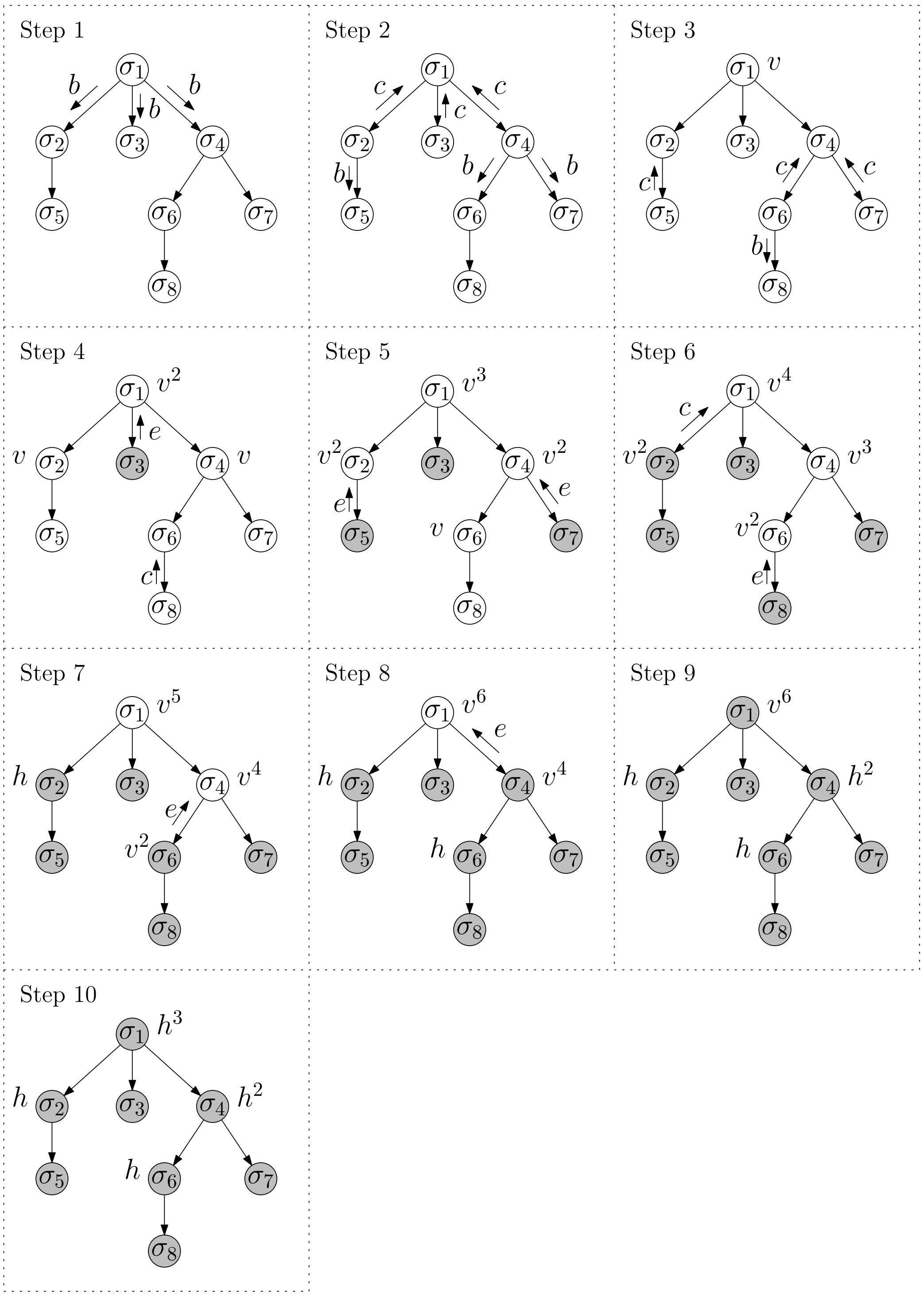


Figure 4.5: The propagation of symbols b, c and e in Algorithm 4.2.3 that determines the height of cells in a rooted tree, for the tree of Figure 4.2. In each cell σ_i, $1 \leq i \leq 8$, the final multiplicity of symbol h represents the height of σ_i (i.e. $\texttt{height}_s(i)$).

4.3 Graph traversal algorithms

This section presents graph traversal algorithms, such as (i) an algorithm that determines the number of shortest paths from the source cell to every cell in Section 4.3.1, (ii) an algorithm that determines distance parity of cells (i.e. even- or odd-distances from the source cell) in Section 4.3.2, (iii) an echo algorithm in Section 4.3.3 and (iv) an algorithm that determines the heights of cells in the spanning BFS DAG (rooted at the source cell) in Section 4.3.4.

In general, the algorithms presented in this section differ from the tree traversal algorithms of Section 4.2, in that (i) the source cell is located at an arbitrary node of a graph (instead of the tree root) and (ii) transfer operator $\updownarrow$ is used in evolution rules (instead of $\uparrow$ and $\downarrow$).

Simple P systems that implement the algorithms of this section differ only by their evolution rules. Hence, the *framework* of a simple P system, Ψ, presented in Definition 4.21, describes all the components of a simple P system Π, except a set of evolution rules. For each algorithm, a formal description of the corresponding system Π is presented by, completing the framework Ψ (of Definition 4.21) with a set of evolution rules that implements the algorithm.

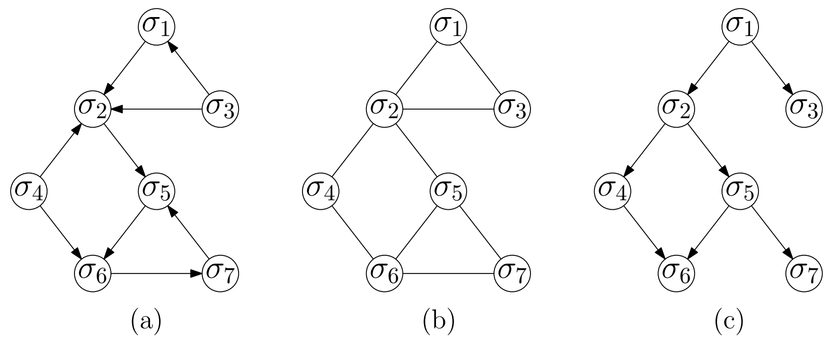
Definition 4.21. (Framework of a simple P system for graph traversal algorithms) The framework of a simple P system (of order n), for designing the traversal algorithms of this section, is $\Psi = (O, K, \Delta)$, where:

1. $O = \{a, c, e, h, o, v, x, s, t\}$.
2. $K = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, where $\sigma_s \in K$ is the *source* cell.

   Each cell $\sigma_i \in K$ has an initial form (Q, s_{i0}, w_{i0}, R), where:

   - $Q = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$, where s_0 is the initial quiescent state.
   - $s_{i0} = s_0$, is the initial state.
   - $w_{i0} = \begin{cases} s & \text{if } \sigma_i = \sigma_s, \\ \emptyset & \text{if } \sigma_i \neq \sigma_s, \end{cases}$ is the initial content.
   - R is a set of evolution rules, which is given in each algorithm of this section.
3. Δ forms a connected (communication) graph.

In Figure 4.6, (a) is a digraph G with the source cell $\sigma_s = \sigma_1$, (b) is the communication graph of G, which will be used for the visual description figures and evolution trace tables, (c) is the spanning BFS DAG of G and (d) is a table that contains the number of shortest paths ($\texttt{paths}_s(i)$) and height ($\texttt{height}_s(i)$), for each cell σ_i, $1 \leq i \leq 7$.



Cell	$\texttt{paths}_s$	$\texttt{height}_s$
σ_1	1	3
σ_2	1	2
σ_3	1	0
σ_4	1	1
σ_5	1	1
σ_6	2	0
σ_7	1	0

(d)

Figure 4.6: (a) A digraph G with the source cell $\sigma_s = \sigma_1$. (b) The communication graph of G, i.e. the underlying graph of G. (c) The spanning BFS DAG of G. (d) A table that contains the number of shortest paths ($\texttt{paths}_s(i)$) and height ($\texttt{height}_s(i)$), for each cell σ_i, $1 \leq i \leq 7$.

4.3.1 Algorithm: Number of shortest paths

This algorithm determines the number of the *shortest paths* from the source cell to each cell. Further, each cell determines the total number of shortest paths from the source to all its peers and successors. A set of evolution rules that transforms the framework Ψ (of Definition 4.21) into a simple P system Π of Algorithm 4.3.1 is presented. Then, the description and analysis of system Π are provided.

Precondition of system Π

Each cell $\sigma_i \in K$ starts with the initial configuration described in Definition 4.21.

Postcondition of system Π

When Π halts, the configuration of cell $\sigma_i \in K$ is (s_3, w_i), where:

- $|w_i|_u = \texttt{paths}_s(i)$, is the number of the shortest paths from σ_s to σ_i.

- $|w_i|_y = \sum_{\sigma_j \in \mathtt{Peer}_s(i)} \mathtt{paths}_s(j)$, is the number of the shortest paths from σ_s to all σ_i's peers.
- $|w_i|_z = \sum_{\sigma_j \in \mathtt{Succ}_s(i)} \mathtt{paths}_s(j)$, is the number of the shortest paths from σ_s to all σ_i's successors.

States and symbols of system Π

State s_0 is an initial quiescent state, where cells remain until they receive a broadcast symbol. States s_0, s_1 and s_2 represent the states, where a cell expects to receive symbols from its predecessors, peers and successors, respectively, if any. State s_3 represents the terminal state. Symbol a is the broadcast symbol. The final multiplicity of symbol u in cell σ_i indicates the number of shortest paths from the source cell to σ_i. The final multiplicities of symbols y and z in cell σ_i indicate the number of shortest paths from the source cell to σ_i's peers and successors, respectively.

Evolution rules of system Π

The set of evolution rules below determines the number of the shortest paths from the source to each cell $\sigma_i \in K$.

0. Rules for state s_0:
   1. $s_0 \ s \rightarrow_{\mathtt{min}} s_1 \ u \ (a, \updownarrow)$
   2. $s_0 \ a \rightarrow_{\mathtt{max}} s_1 \ u \ (a, \updownarrow)$
1. Rules for state s_1:
   1. $s_1 \ u \rightarrow_{\mathtt{min}} s_2 \ u$
   2. $s_1 \ a \rightarrow_{\mathtt{max}} s_2 \ y$
2. Rules for state s_2:
   1. $s_2 \ u \rightarrow_{\mathtt{min}} s_3 \ u$
   2. $s_2 \ a \rightarrow_{\mathtt{max}} s_3 \ z$

Overview of system Π

In this algorithm, broadcast symbol a is propagated from the source cell to all other cells in the following manner.

1. The source cell starts this algorithm by sending one copy of symbol a to each of its neighbours (rule 0.1).
2. A non-source cell σ_j participates in this algorithm, when it receives symbol a. For every copy of symbol a that cell σ_j receives, σ_j sends one copy of symbol a to each of its neighbours (rule 0.2).

As indicated in Propositions 4.22, each cell σ_i receives $\texttt{paths}_s(i)$ copies of symbol a from its predecessors, which corresponds to the number of shortest paths from the source cell. Further, cell σ_i receives $\sum_{\sigma_j \in \texttt{Peer}_s(i)} \texttt{paths}_s(j)$ copies of symbol a from its peers and $\sum_{\sigma_j \in \texttt{Succ}_s(i)} \texttt{paths}_s(j)$ copies of symbol a from its successors, as indicated in Propositions 4.23 and 4.24, respectively. Cell σ_i stores the number of the shortest paths from the source cell:

- to itself, by rewriting all copies of symbol a received from its predecessors (at step $\texttt{depth}_s(i)$) into symbol u (rule 0.2); the source cell obtains one copy of symbol u by rewriting the initial symbol s into u (rule 0.1).
- to all its peers, by rewriting all copies of symbol a received from its peers (at step $\texttt{depth}_s(i) + 1$) into symbol y (rules 1.2).
- to all its successors, by rewriting all copies of symbol a received from its successors (at step $\texttt{depth}_s(i) + 2$) into symbol z (rules 2.2).

Table 4.4 contains the traces of system Π, for the graph of Figure 4.6 (b). Figure 4.7 provides a visual description of Π, for the graph of Figure 4.6 (b).

Proposition 4.22. At step $\texttt{depth}_s(i)$, cell $\sigma_i \neq \sigma_s$ receives $\texttt{paths}_s(i)$ copies of symbol a from its predecessors. At step $\texttt{depth}_s(i)+1$, cell σ_i sends $\texttt{paths}_s(i)$ copies of symbol a to each of its successors.

Proof. Proof by induction, on $m = \texttt{depth}_s(i) \geq 1$. At step 1, the source cell sends one copy of symbol a to each of its neighbours. Hence, at step 1, each cell σ_i in depth $\texttt{depth}_s(i) = 1$ receives one copy of symbol a. Then, at step 2, σ_i sends one copy of symbol a to each of its neighbours.

Assume that the induction hypothesis holds for each cell σ_j at depth m. Consider cell σ_i in $m' = m + 1$. By induction hypothesis, at step $m + 1$, each $\sigma_j \in \texttt{Pred}_s(i)$ sends $\texttt{paths}_s(j)$ copies of symbol a to all its neighbours. Thus, at step $m+1 = m'$, σ_i receives $\sum_{\sigma_j \in \texttt{Pred}_s(i)} \texttt{paths}_s(j) = \texttt{paths}_s(i)$ copies of symbol a. At step $\texttt{depth}_s(i)+1$, σ_i sends $\texttt{paths}_s(i)$ copies of symbol a to each of its neighbours (which include its successors). □

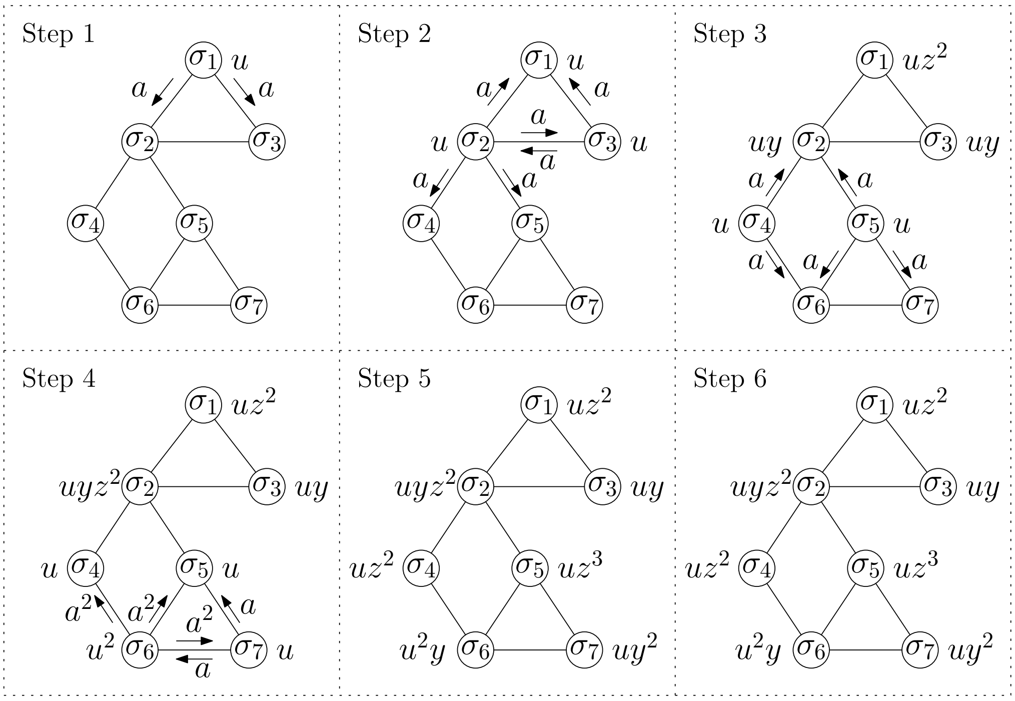


Figure 4.7: The propagation of symbol a in Algorithm 4.3.1 that determines the number of shortest paths from the source to each cell, for the graph of Figure 4.6 (b), where σ_1 is the source cell. In each cell σ_i, $1 \leq i \leq 7$, the final multiplicity of symbol u corresponds to the number of shortest paths from σ_s to σ_i. Further, the final multiplicities of symbols y and z correspond to the number of shortest paths from σ_s to σ_i's peers and successors, respectively.

Proposition 4.23. At step $\texttt{depth}_s(i) + 1$, cell σ_i makes transition $s_0 \Rightarrow s_1$, and reaches configuration (s_1, w_i), where $|w_i|_u = \sum_{\sigma_j \in \texttt{Pred}_s(i)} \texttt{paths}_s(j)$ and $|w_i|_a = \sum_{\sigma_k \in \texttt{Peer}_s(i)} \texttt{paths}_s(k)$.

Proof. As indicated in Proposition 4.22, at step $\texttt{depth}_s(i)$, cell σ_i receives $\texttt{paths}_s(i)$ copies of symbol a from its predecessors, and σ_i reaches configuration (s_0, w_i), where $|w_i|_a = \sum_{\sigma_j \in \texttt{Pred}_s(i)} \texttt{paths}_s(j)$.

At step $\texttt{depth}_s(i) + 1$, cell σ_i:

- Receives $\texttt{paths}_s(k)$ copies of symbol a from each $\sigma_k \in \texttt{Peer}_s(i)$.

  From Proposition 4.22, at step $\texttt{depth}_s(k) + 1$, each cell $\sigma_k \in \texttt{Peer}_s(i)$ sends $\texttt{paths}_s(k)$ copy of symbol a to each of its neighbours. Hence, at step $\texttt{depth}_s(k) + 1 = \texttt{depth}_s(i) + 1$, σ_i receives $\texttt{paths}_s(i)$ copies of symbol a from each $\sigma_k \in$

Table 4.4: The traces of the system Π of Algorithm 4.3.1, for the graph of Figure 4.6 (b), where σ_1 is the source cell. In each cell σ_i, $1 \le i \le 7$, the final multiplicities of symbols u, y and z correspond to $\texttt{paths}_1(i)$, $\sum_{\sigma_j \in \texttt{Peer}_1(i)} \texttt{paths}_1(j)$ and $\sum_{\sigma_j \in \texttt{Succ}_1(i)} \texttt{paths}_1(j)$, respectively. For example, cell σ_2 has one predecessor, one peer and two successors. Cell σ_2 has one shortest path from σ_1 and each of σ_2's peer and successors has one shortest path from σ_1. Thus, at step 6, σ_2 contains multiset uyz^2.

Step	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7
0	$s_0\ s$	s_0	s_0	s_0	s_0	s_0	s_0
1	$s_1\ u$	$s_0\ a$	$s_0\ a$	s_0	s_0	s_0	s_0
2	$s_2\ a^2u$	$s_1\ au$	$s_1\ au$	$s_0\ a$	$s_0\ a$	s_0	s_0
3	$s_3\ uz^2$	$s_2\ a^2uy$	$s_2\ uy$	$s_1\ u$	$s_1\ u$	$s_0\ a^2$	$s_0\ a$
4	$s_3\ uz^2$	$s_3\ uyz^2$	$s_3\ uy$	$s_2\ a^2u$	$s_2\ a^3u$	$s_1\ au^2$	$s_1\ a^2u$
5	$s_3\ uz^2$	$s_3\ uyz^2$	$s_3\ uy$	$s_3\ uz^2$	$s_3\ uz^3$	$s_2\ u^2y$	$s_2\ uy^2$
6	$s_3\ uz^2$	$\boldsymbol{s_3\ uyz^2}$	$s_3\ uy$	$s_3\ uz^2$	$s_3\ uz^3$	$s_3\ u^2y$	$s_3\ uy^2$

$\texttt{Peer}_s(i)$. Hence, in total, cell σ_i receives $\sum_{\sigma_k \in \texttt{Peer}_s(i)} \texttt{paths}_s(k)$ copies of symbol a.

- Rewrites each symbol a, received at step $\texttt{depth}_s(i)$, into one copy of symbol u, to obtain $\sum_{\sigma_j \in \texttt{Pred}_s(i)} \texttt{paths}_s(j)\}$ copies of symbol u.
- Enters state s_1, from the transition $s_0 \Rightarrow s_1$.

□

Proposition 4.24. At step $\texttt{depth}_s(i) + 2$, cell σ_i makes transition $s_1 \Rightarrow s_2$, and reaches configuration (s_2, w_i), where $|w_i|_u = \sum_{\sigma_j \in \texttt{Pred}_s(i)} \texttt{paths}_s(j)$, $|w_i|_y = \sum_{\sigma_k \in \texttt{Peer}_s(i)} \texttt{paths}_s(k)$ and $|w_i|_a = \sum_{\sigma_h \in \texttt{Peer}_s(i)} \texttt{paths}_s(h)$.

Proof. As indicated in Proposition 4.23, at step $\texttt{depth}_s(i)+1$, cell σ_i reaches configuration (s_1, w_i), where $|w_i|_u = \sum_{\sigma_j \in \texttt{Pred}_s(i)} \texttt{paths}_s(j)$ and $|w_i|_a = \sum_{\sigma_k \in \texttt{Peer}_s(i)} \texttt{paths}_s(k)$.

At step $\texttt{depth}_s(i) + 2$, cell σ_i:

- Receives $\texttt{paths}_s(h)$ copies of symbol a from each $\sigma_h \in \texttt{Succ}_s(i)$.

  From Proposition 4.22, at step $\texttt{depth}_s(h) + 1$, each cell $\sigma_h \in \texttt{Succ}_s(i)$ sends $\texttt{paths}_s(h)$ copy of symbol a to each of its neighbours. Hence, at step $\texttt{depth}_s(h) + 1 = \texttt{depth}_s(i) + 2$, σ_i receives $\texttt{paths}_s(h)$ copies of symbol a from each $\sigma_h \in \texttt{Peer}_s(i)$. Hence, in total, cell σ_i receives $\sum_{\sigma_h \in \texttt{Succ}_s(i)} \texttt{paths}_s(h)$ copies of symbol a.

- Rewrites each symbol a, received at step $\texttt{depth}_s(i) + 1$, into one copy of symbol y, to obtain $\sum_{\sigma_k \in \texttt{Peer}_s(i)} \texttt{paths}_s(k)$ copies of symbol y.

- Enters state s_2, from the transition $s_1 \Rightarrow s_2$.

□

Proposition 4.25. At step $\texttt{depth}_s(i) + 3$, cell σ_i makes transition $s_2 \Rightarrow s_3$, and reaches configuration (s_3, w_i), where $|w_i|_u = \sum_{\sigma_j \in \texttt{Pred}_s(i)} \texttt{paths}_s(j)$,
$|w_i|_y = \sum_{\sigma_k \in \texttt{Peer}_s(i)} \texttt{paths}_s(k)$ and $|w_i|_z = \sum_{\sigma_h \in \texttt{Peer}_s(i)} \texttt{paths}_s(h)$.

Proof. As indicated in Proposition 4.24, at step $\texttt{depth}_s(i) + 2$, cell σ_i reaches configuration (s_2, w_i), where $|w_i|_u = \sum_{\sigma_j \in \texttt{Pred}_s(i)} \texttt{paths}_s(j)$, $|w_i|_y = \sum_{\sigma_k \in \texttt{Peer}_s(i)} \texttt{paths}_s(k)$ and $|w_i|_a = \sum_{\sigma_h \in \texttt{Peer}_s(i)} \texttt{paths}_s(h)$.

At step $\texttt{depth}_s(i) + 3$, cell σ_i:

- Rewrites each symbol a, received at step $\texttt{depth}_s(i) + 2$, into one copy of symbol z, to obtain $\sum_{\sigma_h \in \texttt{Succ}_s(i)} \texttt{paths}_s(h)$ copies of symbol z.

- Enters state s_3, from the transition $s_2 \Rightarrow s_3$.

□

Correctness and complexity of system Π

Proposition 4.26 indicates the correctness and time complexity of system Π. Moreover, Proposition 4.27 indicates the message complexity of system Π.

Proposition 4.26. System Π halts at step $\texttt{ecc}(s) + 3$ and the configuration of each cell corresponds to the postcondition.

Proof. As indicated in Proposition 4.25, at step $\texttt{depth}_s(i)+3$, cell $\sigma_i \in K$ reaches configuration (s_3, w_i), where $|w_i|_u = \sum_{\sigma_j \in \texttt{Pred}_s(i)} \texttt{paths}_s(j)$, $|w_i|_y = \sum_{\sigma_k \in \texttt{Peer}_s(i)} \texttt{paths}_s(k)$ and $|w_i|_z = \sum_{\sigma_h \in \texttt{Peer}_s(i)} \texttt{paths}_s(h)$.

There are no rules in state s_3, hence, cell σ_i cannot evolve once it enters state s_3. For a farthest cell σ_f (with respect to the source cell σ_s), $\texttt{depth}_s(f) \geq \texttt{depth}_s(g)$, for all $\sigma_g \in K$, such that $\texttt{depth}_s(f) = \texttt{ecc}(s)$. Cell σ_f enters state s_3 at step $\texttt{depth}_s(i) + 3 = \texttt{ecc}(s) + 3$.

Therefore, system Π halts at step $\texttt{ecc}(s) + 3$, and the final configuration of each cell corresponds to the postcondition of Algorithm 4.3.1. $\square$

Proposition 4.27. The total number of symbols that are transferred between cells of Π is $\sum_{\sigma_i \in K}(\texttt{paths}_s(i) \cdot |\texttt{Neighbour}(i)|)$.

Proof. From Proposition 4.22, each cell σ_i receives $\texttt{paths}_s(i)$ copies of symbol a and sends $\texttt{paths}_s(i)$ copies of symbol a to each of its neighbours. Hence, σ_i sends $\texttt{paths}_s(i) \cdot |\texttt{Neighbour}(i)|$ copies of symbol a. Therefore, in total, $\sum_{\sigma_i \in K}(\texttt{paths}_s(i) \cdot |\texttt{Neighbour}(i)|)$ copies of symbol a are transferred between cells of Π. $\square$

4.3.2 Algorithm: Distance parity

In this algorithm, each cell determines its *distance parity* with respect to the source cell, i.e. even or odd distance from the source cell. Further, each cell determines the number of its predecessors, peers and successors. A set of evolution rules that transforms the framework Ψ (of Definition 4.21) into a simple P system Π of Algorithm 4.3.2 is presented. Then, the description and analysis of system Π are provided.

For cell $\sigma_i \in K$, symbols μ_i and $\bar{\mu}_i$ are defined as follow. If σ_i is an odd-parity cell, then $\mu_i = o$ and $\bar{\mu}_i = x$. If σ_i is an even-parity cell, then $\mu_i = x$ and $\bar{\mu}_i = o$.

Precondition of system Π

Each cell $\sigma_i \in K$ starts with the initial configuration described in Definition 4.21.

Postcondition of system Π

When Π halts, the configuration of cell $\sigma_i \in K$ is (s_i, w_i), where:

- $s_i = \begin{cases} s_3 & \text{if } \texttt{depth}_s(i) \text{ is even,} \\ s_4 & \text{if } \texttt{depth}_s(i) \text{ is odd.} \end{cases}$
- $|w_i|_a = 1$.
- $|w_i|_b = |\texttt{Pred}_s(i)|$, is the number of σ_i's predecessors.
- $|w_i|_c = |\texttt{Peer}_s(i)|$, is the number of σ_i's peers.
- $|w_i|_d = |\texttt{Succ}_s(i)|$, is the number of σ_i's successors.

States and symbols of system Π

State s_0 is an initial quiescent state, where each cell remains idle until it receives a broadcast symbol. States s_0, s_1 and s_2 represent the states where each cell receives broadcast symbols from its predecessors, peers and successors, respectively, if any. States s_3 and s_4 represent the terminal state for even- and odd-parity cells, respectively. Symbols o and x are the broadcast symbols for even- and odd-parity cells, respectively. Symbols a and e are produced when a cell in state s_0 receives symbols x and o, respectively. Symbol e is later used to guide an odd-parity cell to end in state s_4; at the same time, symbol e is rewritten into symbol a. Symbols b, c and d are produced from the broadcast symbol (x or o) received from predecessors, peers and successors, respectively.

Evolution rules of system Π

The set of evolution rules below determines the distance parity of cell $\sigma_i \in K$.

0. Rules for state s_0:
   1. $s_0\ s \rightarrow_{\texttt{min}} s_1\ a\ (o, \updownarrow)$
   2. $s_0\ x \rightarrow_{\texttt{min}} s_1\ ab\ (o, \updownarrow)$
   3. $s_0\ o \rightarrow_{\texttt{min}} s_1\ eb\ (x, \updownarrow)$
   4. $s_0\ x \rightarrow_{\texttt{max}} s_1\ b$
   5. $s_0\ o \rightarrow_{\texttt{max}} s_1\ b$
1. Rules for state s_1:
   1. $s_1\ a \rightarrow_{\texttt{min}} s_2\ a$
   2. $s_1\ e \rightarrow_{\texttt{min}} s_2\ e$
   3. $s_1\ o \rightarrow_{\texttt{max}} s_2\ c$
   4. $s_1\ x \rightarrow_{\texttt{max}} s_2\ c$
2. Rules for state s_2:
   1. $s_2\ a \rightarrow_{\texttt{min}} s_3\ a$
   2. $s_2\ e \rightarrow_{\texttt{min}} s_4\ a$
   3. $s_2\ x \rightarrow_{\texttt{max}} s_3\ d$
   4. $s_2\ o \rightarrow_{\texttt{max}} s_4\ d$

Overview of system Π

In this algorithm, broadcast symbols (o and x) are propagated from the source cell to all other cells, where (i) the initial broadcast symbol is o and (ii) at every step, the current broadcast symbol alternates between symbols o and x. The details are given below.

1. The source cell starts this algorithm by sending one copy of broadcast symbol o to each of its neighbours.
2. A non-source cell σ_j participates in this algorithm, when it receives a broadcast symbol μ_i (either symbol o or x) for the first time.
   - If $\mu_i = o$, then σ_j sends one copy of symbol $\bar{\mu}_i = x$ to each of its neighbours.
   - If $\mu_i = x$, then σ_j sends one copy of symbol $\bar{\mu}_i = o$ to each of its neighbours.

Propositions 4.28, 4.29 and 4.30 indicate the broadcast symbols that each cell receives from its predecessors, peers and successors, respectively.

Specifically, each cell σ_i, $1 \leq i \leq n$,

1. sends one copy of symbol $\bar{\mu}_i$ to each of its neighbours (rule 0.1 for an even-parity cell; rule 0.2 for an odd-parity cell).
2. stores the number of its predecessors, peers and successors by rewriting:
   - every copy of symbol μ_i (received from its predecessors) into one copy of symbol b (rules 0.2 and 0.4 for an even-parity cell; rules 0.3 and 0.5 for an odd-parity cell),

- every copy of symbol $\bar{\mu}_i$ (received from its peers) into one copy of symbol c (rule 1.3 for an even-parity cell; rule and 1.4 for an odd-parity cell),
- every copy of symbol μ_i (received from its successors) into one copy of symbol d (rule 2.3 for an even-parity cell; rule 2.4 for an odd-parity cell).

Table 4.5 contains the traces of system Π of Figure 4.6 (b). Figure 4.8 provides a visual description of Π of Figure 4.6 (b).

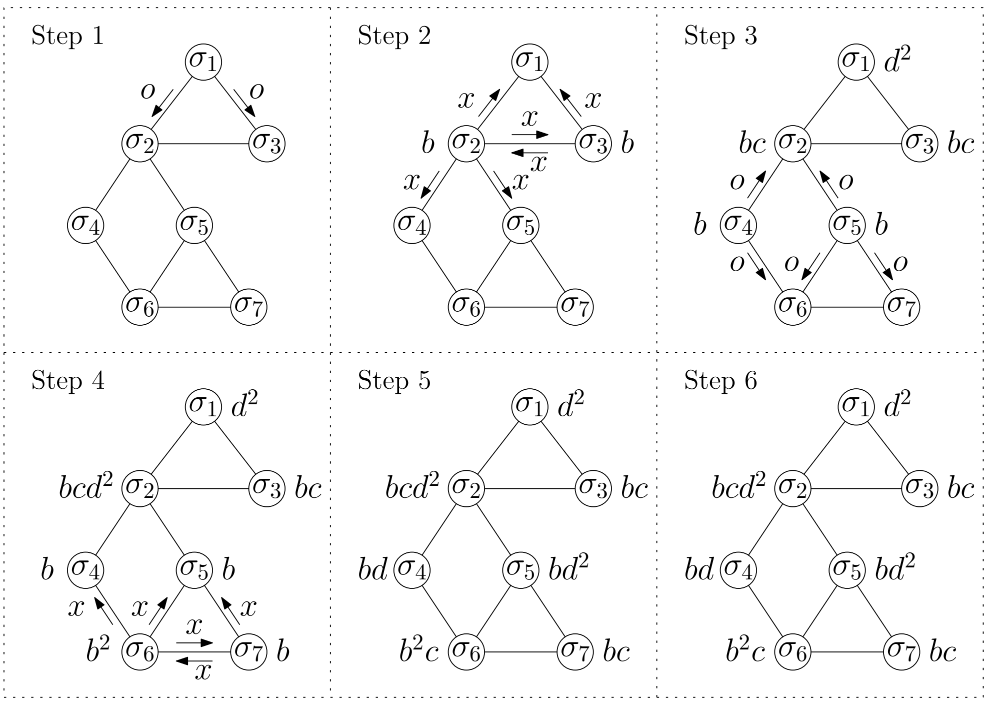


Figure 4.8: The propagation of symbols o and x in Algorithm 4.3.2 that determines the distance parity of cells, for the graph of Figure 4.6 (b), where σ_1 is the source cell. Even- and odd-parity cells receive symbols x and o from their predecessors, respectively. In each cell σ_i, $1 \le i \le 7$, the final multiplicities of symbols b, c and d represent the number of σ_i's predecessors, peers and successors, respectively.

Proposition 4.28. At step $\texttt{depth}_s(i)$, cell $\sigma_i \neq \sigma_s$ receives $|\texttt{Pred}_s(i)|$ copies of symbol μ_i from its predecessors. At step $\texttt{depth}_s(i) + 1$, cell σ_i sends one copy of symbol $\bar{\mu}_i$ to each of its successors.

Proof. Proof by induction, on $m = \texttt{depth}_s(i) \ge 1$. At step 1, the source σ_s sends one copy of symbol o to each of its neighbours. Hence, at step 1, each cell σ_i in depth 1

Table 4.5: The traces of the system Π of Algorithm 4.3.2, which determines the distance parity of cells with respect to the source cell, for the graph of Figure 4.6 (b), where σ_1 is the source cell. Even- and odd-parity cells end in states s_3 and s_4, respectively. In each cell σ_i, $1 \le i \le 7$, the final multiplicities of symbols b, c and d correspond to $|\texttt{Pred}_1(i)|$, $|\texttt{Peer}_1(i)|$ and $|\texttt{Succ}_1(i)|$, respectively. For example, σ_2 has one predecessor, one peer and two successors. Thus, at step 6, σ_2 contains multiset bcd^2.

Step	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7
0	$s_0\ s$	s_0	s_0	s_0	s_0	s_0	s_0
1	$s_1\ a$	$s_0\ o$	$s_0\ o$	s_0	s_0	s_0	s_0
2	$s_2\ ax^2$	$s_1\ bex$	$s_1\ bex$	$s_0\ x$	$s_0\ x$	s_0	s_0
3	$s_3\ ad^2$	$s_2\ bceo^2$	$s_2\ bce$	$s_1\ ab$	$s_1\ ab$	$s_0\ o^2$	$s_0\ o$
4	$s_3\ ad^2$	$s_4\ abcd^2$	$s_4\ abc$	$s_2\ abx$	$s_2\ abx^2$	$s_1\ b^2ex$	$s_1\ bex$
5	$s_3\ ad^2$	$s_4\ abcd^2$	$s_4\ abc$	$s_3\ abd$	$s_3\ abd^2$	$s_2\ b^2ce$	$s_2\ bce$
6	$s_3\ ad^2$	$\boldsymbol{s_4\ abcd^2}$	$s_4\ abc$	$s_3\ abd$	$s_3\ abd^2$	$s_4\ ab^2c$	$s_4\ abc$

receives one copy of symbol $o = \mu_i$. Then, at step 2, cell σ_i sends one copy of symbol $x = \bar{\mu}_i$ to each of its neighbours.

Assume that the induction hypothesis holds for each cell σ_j at depth m. Consider cell σ_i in $m' = m + 1$. By induction hypothesis, at step $m + 1$, each $\sigma_j \in \texttt{Pred}_s(i)$ sends one copy of symbol $\bar{\mu}_j$ to each its neighbours. Thus, in step $m + 1 = m'$, σ_i receives $|\texttt{Pred}_s(i)|$ copies of symbol $\bar{\mu}_j = \mu_i$. At step $\texttt{depth}_s(i) + 1$, cell σ_i sends one copy of symbol $\bar{\mu}_i$ to each of its neighbours (which includes its successors). $\square$

Proposition 4.29. At step $\texttt{depth}_s(i) + 1$, cell σ_i makes transition $s_0 \Rightarrow s_1$, and reaches configuration: (i) $(s_1, ab^j\bar{\mu}_i^k)$, if σ_i is an even-parity cell, where $j = |\texttt{Pred}_s(i)|$ and $k = |\texttt{Peer}_s(i)|$, and (ii) $(s_1, eb^j\bar{\mu}_i^k)$, if σ_i is an odd-parity cell.

Proof. From Proposition 4.28, cell σ_i receives $|\texttt{Pred}_s(i)|$ copies of symbol μ_i and reaches configuration (s_0, μ_i^j), where $j = |\texttt{Pred}_s(i)|$.

At step $\texttt{depth}_s(i) + 1$, cell σ_i:

- Rewrites one copy of symbol μ_i, received at step $\texttt{depth}_s(i)$, into one copy of:

- ∘ symbol a, if σ_i is an even-parity cell or
- ∘ symbol e, if σ_i is an odd-parity cell.

- Rewrites each remaining copy of symbol μ_i, received at step $\texttt{depth}_s(i)$, into one copy of symbol b.

- Receives $|\texttt{Peer}_s(i)|$ copies of symbol $\bar{\mu}_i$ from its peers.

  From Proposition 4.28, each cell $\sigma_j \in \texttt{Peer}_s(i)$ sends one copy of symbol $\bar{\mu}_j$ to each of its neighbours at step $\texttt{depth}_s(j) + 1$. Hence, at step $\texttt{depth}_s(j) + 1 = \texttt{depth}_s(i) + 1$, cell σ_i receives one copy of symbol $\bar{\mu}_j = \bar{\mu}_i$ from cell σ_j. Thus, at step $\texttt{depth}_s(i) + 1$, cell σ_i receives $|\texttt{Peer}_s(i)|$ copies of symbol $\bar{\mu}_i$.

- Enters state s_1 from the transition $s_0 \Rightarrow s_1$.

□

Proposition 4.30. At step $\texttt{depth}_s(i) + 2$, cell σ_i makes transition $s_1 \Rightarrow s_2$, and reaches configuration: (i) $(s_2, ab^j c^k \mu_i^h)$, if σ_i is an even-parity cell, where $j = |\texttt{Pred}_s(i)|$, $k = |\texttt{Peer}_s(i)|$ and $h = |\texttt{Succ}_s(i)|$, or (ii) $(s_2, eb^j c^k \mu_i^h)$, if σ_i is an odd-parity cell.

Proof. From Proposition 4.29, at step $\texttt{depth}_s(i) + 1$: (i) an even-parity cell reaches configuration $(s_1, ab^j \bar{\mu}_i^k)$, where $j = |\texttt{Pred}_s(i)|$ and $k = |\texttt{Peer}_s(i)|$, and (ii) an odd-parity cell reaches configuration $(s_1, eb_j \bar{\mu}_i^k)$.

At step $\texttt{depth}_s(i) + 2$, cell σ_i:

- Rewrites each copy of symbol $\bar{\mu}_i$, received at step $\texttt{depth}_s(i) + 1$, into one copy of symbol c.

- Receives $|\texttt{Succ}_s(i)|$ copies of symbol μ_i from its successors.

  From Proposition 4.28, each cell $\sigma_j \in \texttt{Succ}_s(i)$ sends one copy of symbol $\bar{\mu}_j$ to each of its neighbours at step $\texttt{depth}_s(j) + 1$. Hence, at step $\texttt{depth}_s(j) + 1 = \texttt{depth}_s(i) + 2$, cell σ_i receives one copy of symbol $\bar{\mu}_j = \mu_i$ from cell σ_j. Thus, at step $\texttt{depth}_c(i) + 2$, cell σ_i receives $|\texttt{Succ}_s(i)|$ copies of symbol μ_i.

- Enters state s_2 from the transition $s_1 \Rightarrow s_2$.

□

Proposition 4.31. At step $\texttt{depth}_s(i) + 3$, an even-parity cell σ_i makes transition $s_2 \Rightarrow s_3$ and reaches configuration $(s_3, ab^j c^k d^h)$, where $j = |\texttt{Pred}_s(i)|$, $k = |\texttt{Peer}_s(i)|$ and $h = |\texttt{Succ}_s(i)|$.

Proof. From Proposition 4.30, at step $\texttt{depth}_s(i) + 2$, an even-parity cell σ_i reaches configuration $(s_2, ab^j c^k \mu_i^h)$, where $j = |\texttt{Pred}_s(i)|$, $k = |\texttt{Peer}_s(i)|$ and $h = |\texttt{Succ}_s(i)|$.

At step $\texttt{depth}_s(i) + 3$, cell σ_i:

- Rewrites each copy of symbol μ_i into one copy of symbol d.
- Enters state s_3 from the transition $s_2 \Rightarrow s_3$.

□

Proposition 4.32. At step $\texttt{depth}_s(i) + 3$, an odd-parity cell σ_i makes transition $s_2 \Rightarrow s_4$ and reaches configuration $(s_4, ab^j c^k d^h)$, where $j = |\texttt{Pred}_s(i)|$, $k = |\texttt{Peer}_s(i)|$ and $h = |\texttt{Succ}_s(i)|$.

Proof. From Proposition 4.30, at step $\texttt{depth}_s(i) + 2$, an odd-parity cell σ_i reaches configuration $(s_2, eb^j c^k \mu_i^h)$, where $j = |\texttt{Pred}_s(i)|$, $k = |\texttt{Peer}_s(i)|$ and $h = |\texttt{Succ}_s(i)|$.

At step $\texttt{depth}_s(i) + 3$, cell σ_i:

- Rewrites one copy of symbol e into one copy of symbol a.
- Rewrites each copy of symbol μ_i into one copy of symbol d.
- Enters state s_4 from the transition $s_2 \Rightarrow s_4$.

□

Proposition 4.33. Cell σ_i is a terminal, if σ_i does not receive a copy of symbol μ_i at step $\texttt{depth}_s(i) + 2$.

Proof. Follows from Proposition 4.30. □

Correctness and complexity of system Π

Proposition 4.34 indicates the correctness of system Π. Proposition 4.35 indicates the message complexity of system Π.

Proposition 4.34. System Π halts in $\texttt{ecc}(s) + 3$ steps and the final configuration of each cell corresponds to the postcondition.

Proof. As indicated in Propositions 4.31 and 4.32, at step $\texttt{depth}_s(i) + 3$, cell $\sigma_i \in K$ reaches the configuration $(s_i, ab^j c^k d^h)$, where $j = |\texttt{Pred}_s(i)|$, $k = |\texttt{Peer}_s(i)|$ and $h = |\texttt{Succ}_s(i)|$, and $s_i = s_3$ if σ_i is an even-parity cell or $s_i = s_4$ if σ_i is an odd-parity cell.

There are no rules in states s_3 and s_4, hence, cell σ_i cannot evolve once it enters state s_3 or s_4. For a farthest cell σ_f (with respect to the source cell σ_s), $\texttt{depth}_s(f) \geq \texttt{depth}_s(g)$, for all $\sigma_g \in K$, such that $\texttt{depth}_s(f) = \texttt{ecc}(s)$. Cell σ_f enters state s_3 at step $\texttt{depth}_s(i) + 3 = \texttt{ecc}(s) + 3$.

Therefore, system Π halts at step $\texttt{ecc}(s) + 3$ and the final configuration of each cell corresponds to the postcondition of Algorithm 4.3.2. □

Proposition 4.35. The total number of symbols that are transferred between cells of Π is $2 \cdot |\Delta|$.

Proof. Each cell sends one copy of the broadcast symbol to each of its neighbours. Hence, two copies of the broadcast symbol are sent over each arc of Δ. Thus, in total, $2 \cdot |\Delta|$ copies of symbols are transferred between cells of Π. □

4.3.3 Algorithm: Echo for graphs

This algorithm is an extension of Algorithm 4.2.2 for graphs, where the source is not restricted to a distinguished node, such as the tree root. Specifically, (i) a broadcast operation, initiated from the source cell, travels along the arcs of the spanning BFS DAG (rooted at the source cell), followed by (ii) a convergecast operation, initiated from the spanning BFS DAG leaves, that backtracks along the arcs of the BFS DAG towards the source cell.

A set of evolution rules that transforms the framework Ψ (of Definition 4.21) into a simple P system Π of Algorithm 4.3.3 is presented. Then, the description and analysis of system Π are provided.

For cell σ_i, $1 \leq i \leq n$, symbols μ_i and $\bar{\mu}_i$ are defined as follow. If σ_i is an odd-parity cell, then $\mu_i = o$ and $\bar{\mu}_i = x$. If σ_i is an even-parity cell, then $\mu_i = x$ and $\bar{\mu}_i = o$.

Precondition of system Π

Each cell $\sigma_i \in K$ starts with the initial configuration described in Definition 4.21.

Postcondition of system Π

When Π halts, the configuration of cell $\sigma_i \in K$ is $(s_6, \emptyset)$.

States and symbols of system Π

In each cell, the multiplicity of: (i) symbol b indicates the number of its predecessors, (ii) symbol c indicates the number of its peers and (iii) symbol d indicates the number of its successors. Symbols o and x represent both the broadcast symbols (in Phase I) and the convergecast symbols in (Phase II), for even- and odd-parity cells, respectively. Symbol t is used to indicate that a cell has sent convergecast to its predecessors. In states s_3 and s_4, even- and odd-parity cells, respectively, check if the current number of convergecast symbols received in Phase II equals the number of successors (determined in Phase I). In state s_5, each cell discards symbols b and c together with the convergecast symbols that are received from its peers and predecessors. State s_6 is the terminal state, indicating that a cell has completed Phase II. The meaning of states s_0, s_1, s_2 are described in Algorithm 4.3.2.

Evolution rules of system Π

The set of evolution rules below corresponds to Algorithm 4.3.3.

Rules used in Phase I (Algorithm 4.3.2):

0. Rules for state s_0:
   1. $s_0\ s \rightarrow_{\texttt{min}} s_1\ a\ (o, \updownarrow)$
   2. $s_0\ x \rightarrow_{\texttt{min}} s_1\ ab\ (o, \updownarrow)$
   3. $s_0\ o \rightarrow_{\texttt{min}} s_1\ eb\ (x, \updownarrow)$
   4. $s_0\ x \rightarrow_{\texttt{max}} s_1\ b$
   5. $s_0\ o \rightarrow_{\texttt{max}} s_1\ b$
1. Rules for state s_1:
   1. $s_1\ a \rightarrow_{\texttt{min}} s_2\ a$
   2. $s_1\ e \rightarrow_{\texttt{min}} s_2\ e$
   3. $s_1\ o \rightarrow_{\texttt{max}} s_2\ c$
   4. $s_1\ x \rightarrow_{\texttt{max}} s_2\ c$
2. Rules for state s_2:
   1. $s_2\ a \rightarrow_{\texttt{min}} s_3\ a$
   2. $s_2\ e \rightarrow_{\texttt{min}} s_4\ a$
   3. $s_2\ x \rightarrow_{\texttt{max}} s_3\ d$
   4. $s_2\ o \rightarrow_{\texttt{max}} s_4\ d$

Rules used in Phase II (convergecast):

3. Rules for state s_3:
   1. $s_3\ t \rightarrow_{\texttt{min}} s_5$
   2. $s_3\ xd \rightarrow_{\texttt{max}} s_3$
   3. $s_3\ ad \rightarrow_{\texttt{min}} s_3\ ad$
   4. $s_3\ a \rightarrow_{\texttt{min}} s_3\ at\ (o, \updownarrow)$
4. Rules for state s_4:
   1. $s_4\ t \rightarrow_{\texttt{min}} s_5$
   2. $s_4\ od \rightarrow_{\texttt{max}} s_4$
   3. $s_4\ ad \rightarrow_{\texttt{min}} s_4\ ad$
   4. $s_4\ a \rightarrow_{\texttt{min}} s_4\ at\ (x, \updownarrow)$
5. Rules for state s_5:
   1. $s_5\ ob \rightarrow_{\texttt{max}} s_5$
   2. $s_5\ xb \rightarrow_{\texttt{max}} s_5$
   3. $s_5\ oc \rightarrow_{\texttt{max}} s_5$
   4. $s_5\ xc \rightarrow_{\texttt{max}} s_5$
   5. $s_5\ ab \rightarrow_{\texttt{min}} s_5\ ab$
   6. $s_5\ ac \rightarrow_{\texttt{min}} s_5\ ac$
   7. $s_5\ a \rightarrow_{\texttt{min}} s_6$

Overview of system Π

In this echo algorithm, each cell progresses through two conceptual phases, Phase I (Algorithm 4.3.2) and Phase II (convergecast), independent of other cells.

1. **Phase I (Algorithm 4.3.2):**
   - The source cell starts Phase I at the beginning of the algorithm. Each non-source cell starts Phase I, when it receives a broadcast symbol.
   - In Phase I, each cell σ_i sends one copy of broadcast symbol $\bar{\mu}_i$ to each of its neighbours, as described in Algorithm 4.3.2.
   - At the end of Phase I, each cell σ_i contains one copy of symbol a, $\texttt{Pred}_s(i)$ copies of symbol b, $\texttt{Peer}_s(i)$ copies of symbol c and $\texttt{Succ}_s(i)$ copies of

symbol d, as indicated in the postcondition of Algorithm 4.3.2. Note, a cell that does not contain any copy of symbol d is a terminal.

2. **Phase II (convergecast):**

   - Each cell starts Phase II, immediately after completing Phase I.
   - In Phase II, each cell sends one copy of convergecast symbol to each of its neighbours, after it receives one copy of convergecast symbol from each of its successors. Each cell can receive the convergecast symbols from its successors and peers. Hence, each cell needs to *distinguish* the convergecast symbols that are originated from its successors, from the convergecast symbols that are originated from its peers.

     The convergecast symbol each cell σ_i sends (in Phase II) is $\bar{\mu}_i$. That is, for each cell, the broadcast symbol (sent in Phase I) and the convergecast symbol (sent in Phase II) are the same. Hence, the convergecast symbol that cell σ_i receives from its: (i) predecessors is μ_i (Proposition 4.28), (ii) peers is $\bar{\mu}_i$ (Proposition 4.29) and (iii) successors is μ_i (Proposition 4.30). Thus, each cell σ_i can *recognize* that it has received convergecast symbols from all its successors, if σ_i has received the first $|\texttt{Succ}_s(i)|$ copies of symbol μ_i (Proposition 4.37).
   - At the end of Phase I, each cell σ_i contains $|\texttt{Succ}_s(i)|$ copies of symbol d. Cell σ_i consumes each copy of symbol d with one copy of symbol μ_i, such that, having zero copies of symbol d indicates that σ_i has received symbol μ_i from all its successors. Cell σ_j uses symbol a to check whether all copies of symbol d are consumed (rule 3.3 for an even-parity cell; rule 4.3 for an odd-parity cell).

     Cell σ_i sends one copy of symbol $\bar{\mu}_i$ to each of its neighbours (rule 3.4 for an even-parity cell; rule 4.4 for an odd-parity cell), after it receives $|\texttt{Succ}_s(i)|$ copies of symbol μ_i from its successors (Proposition 4.38). Each terminal cell σ_t has zero copies of symbol d, thus, σ_t sends one copy of symbol $\bar{\mu}_t$ at the start of Phase II.
   - In Phase II, each cell receives convergecast symbols from its peers and predecessors. At the end of Phase I, each cell σ_i contains $|\texttt{Pred}_s(i)|$ copies of symbol b and $|\texttt{Succ}_s(i)|$ copies of symbol c.

Cell σ_i receives $|\texttt{Pred}_s(i)|$ copies of symbol μ_i from its predecessors and $|\texttt{Peer}_s(i)|$ copies of symbol $\bar{\mu}_i$ from its peers (Proposition 4.39). Cell σ_i consumes symbols b and c together with symbols μ_i and $\bar{\mu}_i$ that are received from its predecessors and peers, respectively (rules 5.1, 5.2, 5.3, 5.4). Cell σ_j uses symbol a to check whether all copies of symbols b and c are consumed (rules 5.5 and 5.6).

Cell σ_i enters terminal state s_6, after it consumes all copies of symbols b and c (rule 5.7); at the same time, σ_i consumes one copy of symbol a. Once cell σ_i enters state s_6, σ_i does not receive any further symbol.

Table 4.6 contains the traces of system Π of Figure 4.6 (b). Figure 4.9 provides a visual description of Π of Figure 4.6 (b).

Table 4.6: The traces of the system Π of Algorithm 4.3.3, for the graph of Figure 4.6 (b) with the source cell σ_1. In each column of cell σ_i, $1 \leq i \leq 7$, the first shaded table cell indicates the start of Phase I, the second shaded table cell indicates the start of Phase II (equivalently, the end of Phase I) and the third shaded table cell indicates the end of Phase II.

Step	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7
0	$s_0\ s$	s_0	s_0	s_0	s_0	s_0	s_0
1	$s_1\ a$	$s_0\ o$	$s_0\ o$	s_0	s_0	s_0	s_0
2	$s_2\ ax^2$	$s_1\ bex$	$s_1\ bex$	$s_0\ x$	$s_0\ x$	s_0	s_0
3	$s_3\ ad^2$	$s_2\ bceo^2$	$s_2\ bce$	$s_1\ ab$	$s_1\ ab$	$s_0\ o^2$	$s_0\ o$
4	$s_3\ ad^2$	$s_4\ abcd^2$	$s_4\ abc$	$s_2\ abx$	$s_2\ abx^2$	$s_1\ b^2ex$	$s_1\ bex$
5	$s_3\ ad^2x$	$s_4\ abcd^2x$	$s_4\ abct$	$s_3\ abd$	$s_3\ abd^2$	$s_2\ b^2ce$	$s_2\ bce$
6	$s_3\ ad$	$s_4\ abcd^2x$	$s_5\ abc$	$s_3\ abd$	$s_3\ abd^2$	$s_4\ ab^2c$	$s_4\ abc$
7	$s_3\ ad$	$s_4\ abcd^2x$	$s_5\ abc$	$s_3\ abdx$	$s_3\ abd^2x^2$	$s_4\ ab^2ctx$	$s_4\ abctx$
8	$s_3\ ad$	$s_4\ abcd^2o^2x$	$s_5\ abc$	$s_3\ abt$	$s_3\ abt$	$s_5\ ab^2co^2x$	$s_5\ abcox$
9	$s_3\ adx$	$s_4\ abctx$	$s_5\ abcx$	$s_5\ abx$	$s_5\ abx$	$s_5\ a$	$s_5\ a$
10	$s_3\ at$	$s_5\ abcox$	$s_5\ aco$	$s_5\ a$	$s_5\ a$	s_6	s_6
11	$s_5\ a$	$s_5\ a$	$s_5\ a$	s_6	s_6	s_6	s_6
12	s_6	s_6	s_6	s_6	s_6	s_6	s_6

Proposition 4.36. By step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 3$, each non-leaf cell σ_i receives $|\texttt{Succ}_s(i)|$ copies of symbol μ_i from all its successors. At step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 4$, each cell σ_i sends one copy of symbol $\bar{\mu}_i$ to each of its neighbours.

Proof. Proof by induction, on cell σ_i's height, $h = \texttt{height}_s(i)$.

In the base case, where $h = 0$, cell σ_i is a terminal cell. Clearly, cell σ_i has no successors and therefore does not receive any symbol μ_i at step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 3 = \texttt{depth}_s(i) + 3$. Thus, at step $\texttt{depth}_s(i) + 4$, cell σ_i sends one copy of symbol $\bar{\mu}_i$ to each of its neighbours.

Assume that the induction hypothesis holds for each cell σ_k at height $\texttt{height}_s(k) \leq h$. Consider a non-terminal cell σ_i at height $\texttt{height}_s(i) = h + 1$. Each cell $\sigma_j \in \texttt{Succ}_s(i)$ has height $\texttt{height}_s(j) \leq h$, thus it satisfies the induction hypothesis. Cell $\sigma_j \in \texttt{Succ}_s(i)$ sends one copy of symbol $\bar{\mu}_j = \mu_i$ to cell σ_i at step $\texttt{depth}_s(j)+2 \cdot \texttt{height}_s(j)+4$. For cell σ_i, $\texttt{depth}_s(i) = \texttt{depth}_s(j)-1$ for each $\sigma_j \in \texttt{Succ}_s(i)$, and $\max\{\texttt{height}_s(j) \mid \sigma_j \in \texttt{Succ}_s(i)\} = \texttt{height}_s(i)-1$. Thus, by step $(\texttt{depth}_s(i)+1)+2(\texttt{height}_s(i)-1)+4 = \texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 3$, cell σ_i receives $|\texttt{Succ}_s(i)|$ copies of symbol μ_i from all its successors. Then, at step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 4$, cell σ_i sends one copy of symbol $\bar{\mu}_i$ to each $\sigma_z \in \texttt{Neighbour}(i) = \texttt{Pred}_s(i) \cup \texttt{Peer}_s(i) \cup \texttt{Succ}_s(i)$. □

Proposition 4.37. The first $|\texttt{Succ}_s(i)|$ copies of symbol μ_i that cell σ_i receives are sent from σ_i's successors.

Proof. Each $\sigma_k \in \texttt{Pred}_s(i)$ needs convergecast symbol $\bar{\mu}_i$ from σ_i, before it can send its convergecast symbol $\bar{\mu}_k$. Thus, the first $|\texttt{Succ}_s(i)|$ copies of symbol μ_i that σ_i receives must have originated from σ_i's successors. □

Proposition 4.38. A non-terminal cell σ_i (i) receives $|\texttt{Succ}_s(i)|$ copies of symbol μ_i from its successors by step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 2$ and (ii) sends one copy of symbol $\bar{\mu}_i$ to each of its predecessors, peers and successors at step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 3$.

Proof. Proof by induction, on cell σ_i's height, $h = \texttt{height}_s(i)$.

In the base case, where $h = 0$, cell σ_i is a terminal cell. Clearly, cell σ_i has no successors and therefore does not receive any symbol μ_i at step $\texttt{depth}_s(i) + 2$. Thus, at step $\texttt{depth}_s(i)+2\cdot\texttt{height}_s(i)+3 = \texttt{depth}_s(i)+3$, cell σ_i sends one copy of symbol $\bar{\mu}_i$ to each of its neighbours.

Assume that the induction hypothesis holds for each cell σ_k at height $\texttt{height}_s(k) \leq h$. Consider a non-terminal cell σ_i at height $\texttt{height}_s(i) = h+1$. Each cell $\sigma_j \in \texttt{Succ}_s(i)$ has height $\texttt{height}_s(j) \leq h$, thus it satisfies the induction hypothesis. Cell $\sigma_j \in \texttt{Succ}_s(i)$ sends one copy of symbol $\bar{\mu}_j = \mu_i$ to cell σ_i at step $\texttt{depth}_s(j)+2\cdot\texttt{height}_s(j)+3$. For cell σ_i, $\texttt{depth}_s(i) = \texttt{depth}_s(j)-1$ for each $\sigma_j \in \texttt{Succ}_s(i)$, and $\max\{\texttt{height}_s(j) \mid \sigma_j \in \texttt{Succ}_s(i)\} = \texttt{height}_s(i)-1$. Thus, by step $(\texttt{depth}_s(i)+1)+2(\texttt{height}_s(i)-1)+3 = \texttt{depth}_s(i)+2\cdot\texttt{height}_s(i)+2$, cell σ_i receives $|\texttt{Succ}_s(i)|$ copies of symbol μ_i from all its successors. Then, at step $\texttt{depth}_s(i)+2\cdot\texttt{height}_s(i)+3$, cell σ_i sends one copy of symbol $\bar{\mu}_i$ to each $\sigma_z \in \texttt{Neighbour}(i) = \texttt{Pred}_s(i) \cup \texttt{Peer}_s(i) \cup \texttt{Succ}_s(i)$. □

Proposition 4.39. In Phase II, each cell σ_i receives: (i) $|\texttt{Pred}_s(i)|$ copies of symbol μ_i from its predecessors and (ii) $|\texttt{Peer}_s(i)|$ copies of symbol $\bar{\mu}_i$ from its peers.

Proof. For each $\sigma_j \in \texttt{Peer}_s(i)$, $\texttt{depth}_s(j) = \texttt{depth}_s(i)$, hence $\mu_j = \mu_i$ and $\bar{\mu}_j = \bar{\mu}_i$. For each $\sigma_k \in \texttt{Pred}_s(i)$, $\texttt{depth}_s(k) = \texttt{depth}_s(i)-1$, hence $\mu_k = \bar{\mu}_i$ and $\bar{\mu}_k = \mu_i$.

In Phase II, each cell σ_i sends one copy of symbol $\bar{\mu}_i$ to each of its neighbours. Thus, cell σ_i receives $|\texttt{Pred}_s(i)|$ copies of symbol μ_i from its predecessors and $|\texttt{Peer}_s(i)|$ copies of symbol $\bar{\mu}_i$ from its peers. □

Correctness and complexity of system Π

Proposition 4.40 indicates the correctness of system Π.

Proposition 4.40. When system Π halts, the configuration of each cell corresponds to the postcondition.

Proof. At the end of Phase I, each cell σ_i contains one copy of symbol a, $\texttt{Pred}_s(i)$ copies of symbol b, $|\texttt{Peer}_s(i)|$ copies of symbol c and $|\texttt{Succ}_s(i)|$ copies of symbol d. In Phase II, cell σ_i consumes: (i) all copies of symbol b with symbol μ_i received from its predecessors (Proposition 4.39), (ii) all copies of symbol c with symbol $\bar{\mu}_i$ received from its peers (Proposition 4.39) and (iii) all copies of symbol d with symbol μ_i received from its successors (Proposition 4.38). Further, after all copies of symbols b, c and d are consumed, cell σ_i consumes symbol a to enter state s_6. Thus, σ_i ends in configuration $(s_6, \emptyset)$. □

Time and message complexities of system Π

The time and message complexities of system Π of Algorithm 4.3.3 are indicated in Propositions 4.41 and 4.42, respectively.

Proposition 4.41. System Π halts in $2 \cdot \texttt{ecc}(s) + 6$ steps.

Proof. Phase I takes $\texttt{ecc}(s) + 3$ steps (Proposition 4.34). Propagating the convergecast to the source cell σ_s from its farthest cell takes $\texttt{ecc}(s)$ steps. After cell σ_s receives the convergecast symbols from all its successors, σ_s takes three additional steps to enter the end state s_6. Thus, in total, Π halts in $2 \cdot \texttt{ecc}(s) + 6$ steps. □

Proposition 4.42. The total number of symbols that are transferred between cells of Π is $4 \cdot |\Delta|$.

Proof. The total number of transferred symbols in Phase I is $2 \cdot |\Delta|$ (Proposition 4.35). In Phase II, each cell sends one copy of the convergecast symbol to each of its neighbours, i.e. two copies of the convergecast symbols are sent over each arc of Δ. Thus, in Phase II, $2 \cdot |\Delta|$ copies of symbols are transferred between cells of Π. Therefore, the total number of transferred symbols in Phase I and Phase II is $4 \cdot |\Delta|$. □

Remark 4.43. The purpose of distinguishing cells according to their distance parity is to *identify* which of the received convergecast symbols have originated from the successors. The convergecast symbols a cell σ_i receives from (i) its successors are $\bar{\mu}_i$ and (ii) its peers are μ_i. Thus, in Phase II, each cell can correctly detect that it has received convergecast symbols from all its successors, and ignore the convergecast symbols received from its peers. For example, in Figure 4.9, when cell σ_2 receives symbol $x = \mu_2$ from its peer σ_3 at step 5, σ_2 can correctly identify that this symbol x is a convergecast symbol originated from one of its peers.

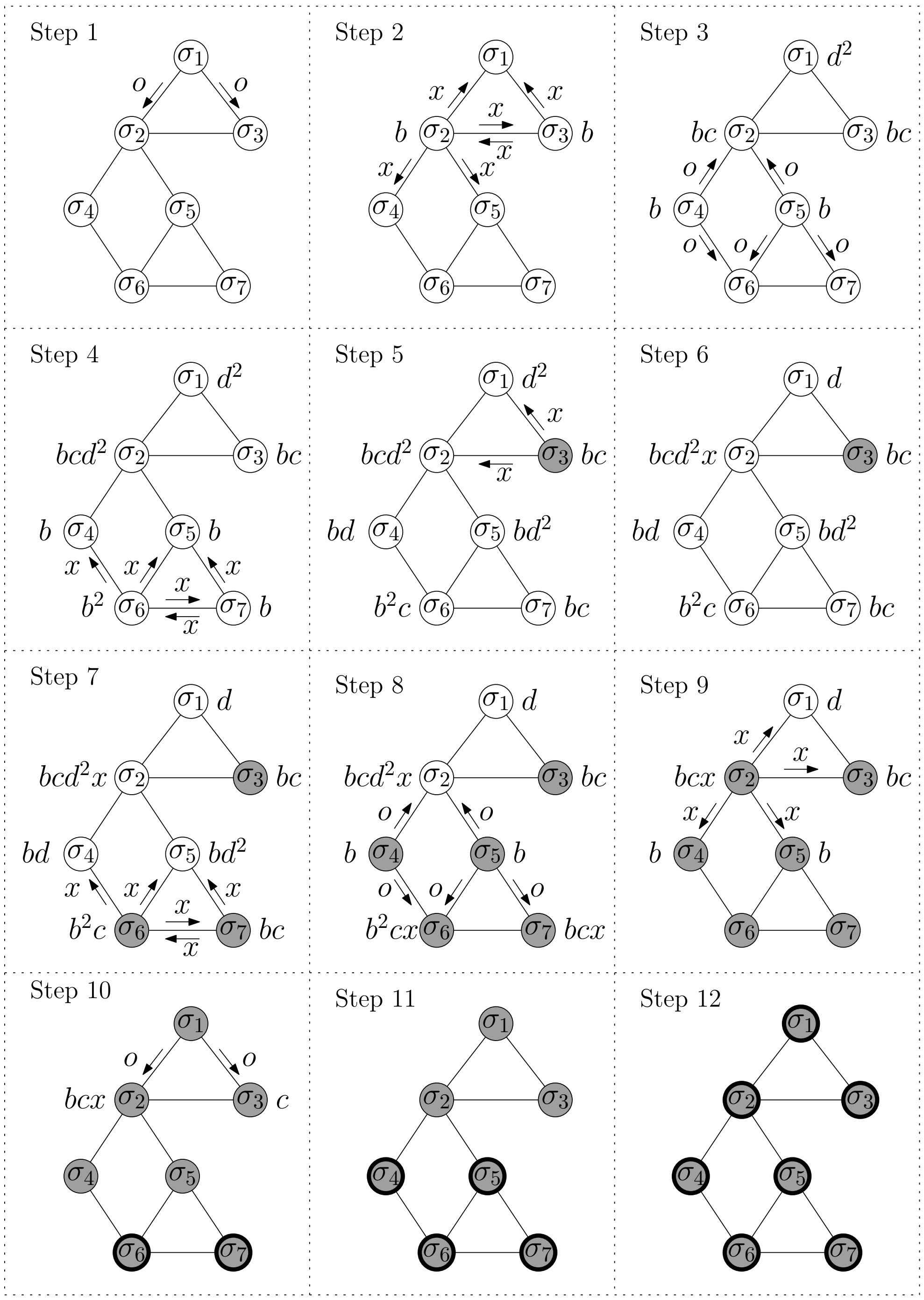


Figure 4.9: The propagation of symbols o and x in Algorithm 4.3.3, for the graph of Figure 4.6 (b), where σ_1 is the source cell.

4.3.4 Algorithm: Cell heights

This algorithm, derived from Algorithm 4.3.3, determines the height of each σ_i, $\texttt{height}_s(i)$.

A set of evolution rules that transforms the framework Ψ (of Definition 4.21) into a simple P system Π of Algorithm 4.3.4 is presented. Then, the description and analysis of system Π are provided.

Precondition of system Π

Each cell $\sigma_i \in K$ starts with the initial configuration described in Definition 4.21.

Postcondition of system Π

When Π halts, the configuration of cell $\sigma_i \in K$ is (s_6, w_i), where:

- $|w_i|_h = \texttt{height}_s(i)$, the height of σ_i.

States and symbols of system Π

The final multiplicity of symbol h in each cell represents the height of that cell. The meaning of other symbols and states are described in Algorithm 4.3.3.

Evolution rules of system Π

The set of evolution rules below determines the height of each cell σ_i, $\texttt{height}_s(i)$. Note, the evolution rules below are the evolution rules used in Algorithm 4.3.3, with the following changes: rules 3.3 and 4.3 are modified to produce $2 \cdot \texttt{height}_s(i)$ copies of symbol v, and rules 3.5 and 4.5 are added to rewrite every two copies of symbol v into one copy of symbol h.

Rules used in Phase I:

0. Rules for state s_0:
   1. $s_0\ s \rightarrow_{\texttt{min}} s_1\ a\ (o, \updownarrow)$
   2. $s_0\ x \rightarrow_{\texttt{min}} s_1\ ab\ (o, \updownarrow)$
   3. $s_0\ o \rightarrow_{\texttt{min}} s_1\ eb\ (x, \updownarrow)$
   4. $s_0\ x \rightarrow_{\texttt{max}} s_1\ b$
   5. $s_0\ o \rightarrow_{\texttt{max}} s_1\ b$

1. Rules for state s_1:
   1. $s_1\ a \rightarrow_{\texttt{min}} s_2\ a$
   2. $s_1\ e \rightarrow_{\texttt{min}} s_2\ e$
   3. $s_1\ o \rightarrow_{\texttt{max}} s_2\ c$
   4. $s_1\ x \rightarrow_{\texttt{max}} s_2\ c$

2. Rules for state s_2:
   1. $s_2\ a \rightarrow_{\texttt{min}} s_3\ a$
   2. $s_2\ e \rightarrow_{\texttt{min}} s_4\ a$
   3. $s_2\ x \rightarrow_{\texttt{max}} s_3\ d$
   4. $s_2\ o \rightarrow_{\texttt{max}} s_4\ d$

Rules used in Phase II:

3. Rules for state s_3:
   1. $s_3\ t \rightarrow_{\texttt{min}} s_5$
   2. $s_3\ xd \rightarrow_{\texttt{max}} s_3$
   3. $s_3\ ad \rightarrow_{\texttt{min}} s_3\ adv$
   4. $s_3\ a \rightarrow_{\texttt{min}} s_3\ at\ (o, \updownarrow)$
   5. $s_3\ vv \rightarrow_{\texttt{max}} s_5\ h$

4. Rules for state s_4:
   1. $s_4\ t \rightarrow_{\texttt{min}} s_5$
   2. $s_4\ od \rightarrow_{\texttt{max}} s_4$
   3. $s_4\ ad \rightarrow_{\texttt{min}} s_4\ adv$
   4. $s_4\ a \rightarrow_{\texttt{min}} s_4\ at\ (x, \updownarrow)$
   5. $s_4\ vv \rightarrow_{\texttt{max}} s_5\ h$

5. Rules for state s_5:
   1. $s_5\ ob \rightarrow_{\texttt{max}} s_5$
   2. $s_5\ xb \rightarrow_{\texttt{max}} s_5$
   3. $s_5\ oc \rightarrow_{\texttt{max}} s_5$
   4. $s_5\ xc \rightarrow_{\texttt{max}} s_5$
   5. $s_5\ ab \rightarrow_{\texttt{min}} s_5\ ab$
   6. $s_5\ ac \rightarrow_{\texttt{min}} s_5\ ac$
   7. $s_5\ a \rightarrow_{\texttt{min}} s_6$

Overview of system Π

In Algorithm 4.3.3, each cell σ_i (i) receives broadcast symbols from all its predecessors (in Phase I) at step $\texttt{depth}_s(i)$ (Proposition 4.28) and (ii) receives convergecast symbols from all its successors (in Phase II) by step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 3$ (Proposition 4.38).

For cell σ_i, consider two steps, t_i and t'_i, where:

- $t_i = \texttt{depth}_s(i) + 3$, i.e. three steps after σ_i receives the broadcast symbols in Phase I,
- $t'_i = \texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 3$, i.e. when σ_i receives convergecast symbols from all its successors in Phase II.

Note that, $t'_i = t_i + 2 \cdot \texttt{height}_s(i)$, where $\texttt{height}_s(i)$ is the height of cell σ_i.

Each cell σ_i produces one copy of symbol v between steps t_i+1 and t'_i (both inclusive) (rule 3.3 for an even-parity cell; rule 4.3 for an odd-parity cell) to accumulate $2 \cdot \texttt{height}_s(i)$ copies of symbol v. Then, at step $t'_i + 1$, each cell σ_i rewrites every two copies of symbol v into one copy of symbol h (rule 3.5 for an even-parity cell; rule 4.5 for an odd-parity cell) to obtain $\texttt{height}_s(i)$ copies of symbol h. Thus, each cell σ_i determines its height, $\texttt{height}_s(i)$, according to the final multiplicity of symbol h.

Table 4.7 contains the traces of system Π, for the graph of Figure 4.6 (b). Figure 4.10 provides a visual description of Π, for the graph of Figure 4.6 (b).

Table 4.7: The traces of the system Π of Algorithm 4.3.4, which determines the height of each cell, for the graph of Figure 4.6 (b), where σ_1 is the source cell. The final multiplicity of symbol h in each cell σ_i, $1 \leq i \leq 7$, corresponds to the height of σ_i. For example, the height of σ_1 is three. Thus, at step 12, σ_1 ends with three copies of symbol h.

Step	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7
0	$s_0\ s$	s_0	s_0	s_0	s_0	s_0	s_0
1	$s_1\ a$	$s_0\ o$	$s_0\ o$	s_0	s_0	s_0	s_0
2	$s_2\ ax^2$	$s_1\ bex$	$s_1\ bex$	$s_0\ x$	$s_0\ x$	s_0	s_0
3	$s_3\ ad^2$	$s_2\ bceo^2$	$s_2\ bce$	$s_1\ ab$	$s_1\ ab$	$s_0\ o^2$	$s_0\ o$
4	$s_3\ ad^2v$	$s_4\ abcd^2$	$s_4\ abc$	$s_2\ abx$	$s_2\ abx^2$	$s_1\ b^2ex$	$s_1\ bex$
5	$s_3\ ad^2v^2x$	$s_4\ abcd^2vx$	$s_4\ abct$	$s_3\ abd$	$s_3\ abd^2$	$s_2\ b^2ce$	$s_2\ bce$
6	$s_3\ adv^3$	$s_4\ abcd^2v^2x$	$s_5\ abc$	$s_3\ abdv$	$s_3\ abd^2v$	$s_4\ ab^2c$	$s_4\ abc$
7	$s_3\ adv^4$	$s_4\ abcd^2v^3x$	$s_5\ abc$	$s_3\ abdv^2x$	$s_3\ abd^2v^2x^2$	$s_4\ ab^2ctx$	$s_4\ abctx$
8	$s_3\ adv^5$	$s_4\ abcd^2o^2v^4x$	$s_5\ abc$	$s_3\ abtv^2$	$s_3\ abtv^2$	$s_5\ ab^2co^2x$	$s_5\ abcox$
9	$s_3\ adv^6x$	$s_4\ abctv^4x$	$s_5\ abcx$	$s_5\ abhx$	$s_5\ abhx$	$s_5\ a$	$s_5\ a$
10	$s_3\ atv^6$	$s_5\ abch^2ox$	$s_5\ aco$	$s_5\ ah$	$s_5\ ah$	s_6	s_6
11	$s_5\ ah^3$	$s_5\ ah^2$	$s_5\ a$	$s_6\ h$	$s_6\ h$	s_6	s_6
12	$\boldsymbol{s_6\ h^3}$	$s_6\ h^2$	s_6	$s_6\ h$	$s_6\ h$	s_6	s_6

Correctness and complexity of system Π

Proposition 4.44 indicates the correctness of system Π.

Proposition 4.44. When system Π halts, the configuration of each cell corresponds to the postcondition.

Proof. Even- and odd-parity cells use rules 3.3 and 4.3, respectively, to produce one copy of symbol v in each step. Each cell σ_i can start applying rule 3.3 or 4.3, when it enters state s_3 or s_4, respectively. Further, σ_i cannot apply rule 3.3 or 4.3, when it receives $|\mathtt{Succ}_s(i)|$ copies of symbol μ_i from its successors.

Each cell σ_i enters state s_3 or s_4 at step $\mathtt{depth}_s(i) + 3$. Hence, cell σ_i can start to produce one copy of symbol v in each step from step $\mathtt{depth}_s(i)+4$. Further, σ_i receives $|\mathtt{Succ}_s(i)|$ copies of symbol μ_i from its successors by step $\mathtt{depth}_s(i)+2\cdot\mathtt{height}_s(i)+3$ (Proposition 4.38). Hence, cell σ_i cannot produce symbol v after step $\mathtt{depth}_s(i) + 2 \cdot \mathtt{height}_s(i) + 3$.

Between steps $\mathtt{depth}_s(i)$ and $\mathtt{depth}_s(i) + 2 \cdot \mathtt{height}_s(i) + 3$, both inclusive, cell σ_i produces one copy of symbol v in each step. Thus, σ_i accumulates $2 \cdot \mathtt{height}_s(i)$ copies of symbol v. Then, using transition $s_3 \Rightarrow s_5$ or $s_4 \Rightarrow s_5$, σ_i rewrites every two copies of symbol v into one copy of symbol h and ends in state s_5. Cell σ_i ends in state s_6 using transition $s_5 \Rightarrow s_6$.

Therefore, σ_i ends in state s_6 with $\mathtt{height}_s(i)$ copies of symbol h. □

Time and message complexities of system Π

The time and message complexities of system Π of Algorithm 4.3.4 are indicated in Propositions 4.45 and 4.46, respectively.

Proposition 4.45. System Π halts in $2 \cdot \mathtt{ecc}(s) + 6$ steps.

Proof. Follows from Proposition 4.41. □

Proposition 4.46. The total number of symbols that are transferred between cells of Π is $4 \cdot |\Delta|$.

Proof. Follows from Proposition 4.42. □

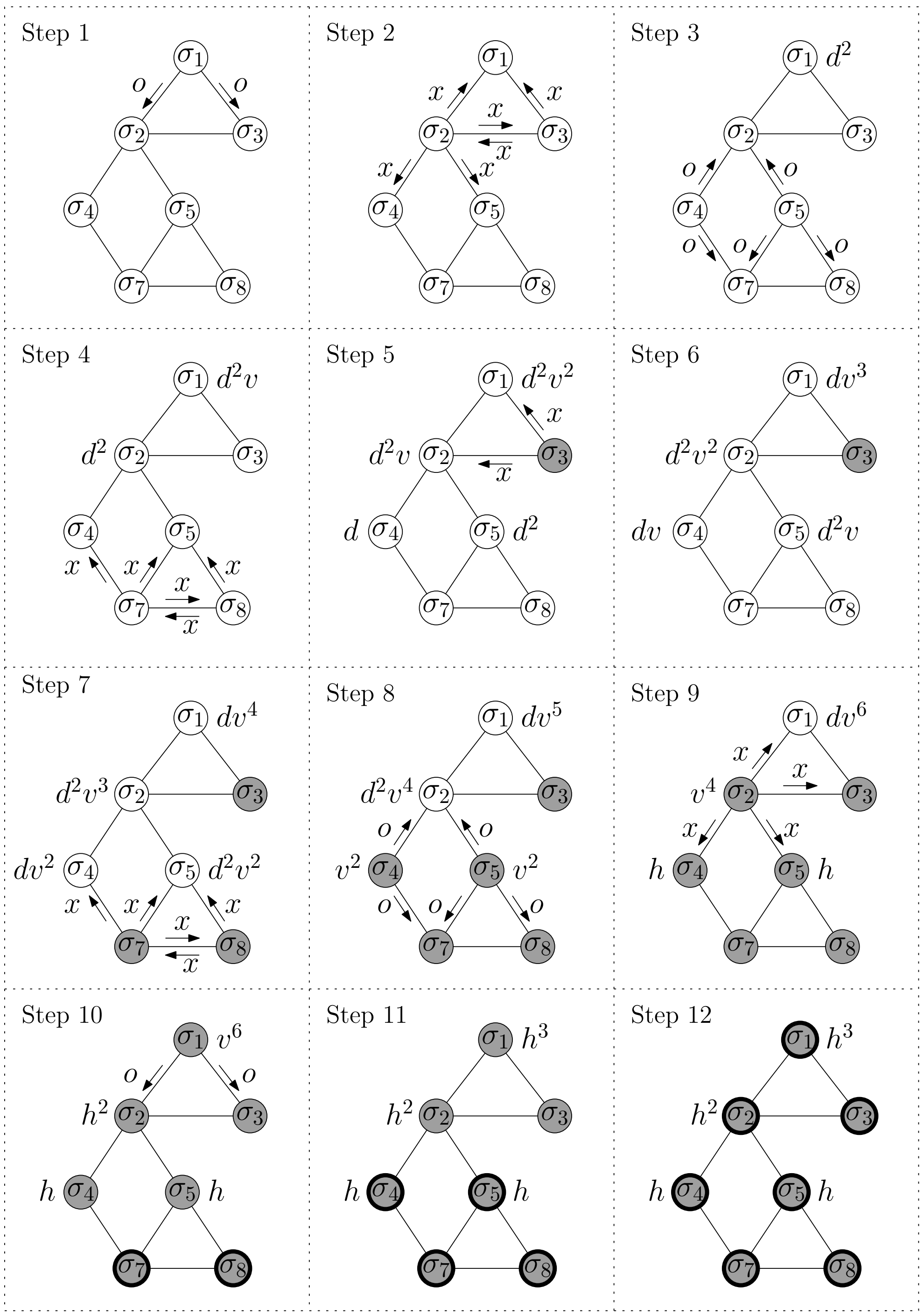


Figure 4.10: The propagation of symbols o and x in Algorithm 4.3.4 that determines the height of each cell, for the graph of Figure 4.6 (b), where σ_1 is the source cell. In each cell σ_i, $1 \leq i \leq 7$, the final multiplicity of symbol h represents the height of σ_i (i.e. $\texttt{height}_s(i)$).

4.4 Summary

Using the standard synchronous distributed broadcast algorithm (Definition 4.1) and echo algorithm (Definition 4.2), this chapter presented *broadcast-based* and *echo-based* P algorithms (i.e. broadcast and echo algorithms that perform additional computations, such as count the number of children and compute the tree height). Explicit evolution rules are presented for each of these P algorithms.

As indicated in Tables 4.8, 4.9, 4.10 and 4.11, these broadcast- and echo-based P algorithms are comparable to the standard synchronous distributed broadcast and echo algorithms, with respect to time and program-size complexities.

Table 4.8: Comparing a synchronous distributed broadcast pseudo-code against Algorithm 4.2.1 (Broadcast with acknowledgement), on trees, where h denotes the tree height.

Algorithm	Time complexity	Program-size complexity
Synchronous distributed broadcast pseudo-code (Definition 4.1)	h	8 pseudo-code lines
Algorithm 4.2.1 (Broadcast with acknowledgement)	$h+2$	2 evolution rules

Table 4.9: Comparing a synchronous distributed broadcast pseudo-code against Algorithms 4.3.1 (Number of shortest paths) and 4.3.2 (Distance parity), on digraphs, where e denotes the eccentricity of the source.

Algorithm	Time complexity	Program-size complexity
Synchronous distributed broadcast pseudo-code (Definition 4.1)	e	8 pseudo-code lines
Algorithm 4.3.1 (Number of shortest paths)	$e+3$	6 evolution rules
Algorithm 4.3.2 (Distance parity)	$e+3$	13 evolution rules

Table 4.10: Comparing a synchronous distributed echo pseudo-code against Algorithms 4.2.2 (Echo) and 4.2.3 (Tree height), on trees, where h denotes tree height.

Algorithm	Time complexity	Program-size complexity
Synchronous distributed echo pseudo-code (Definition 4.2)	$2h$	14 pseudo-code lines
Algorithm 4.2.2 (Echo)	$2h + 4$	6 evolution rules
Algorithm 4.2.3 (Tree height)	$2h + 4$	7 evolution rules

Table 4.11: Comparing a synchronous distributed echo pseudo-code against Algorithms 4.3.3 (Graph echo) and 4.3.4 (Cell heights), on digraphs, where e denotes the eccentricity of the source.

Algorithm	Time complexity	Program-size complexity
Synchronous distributed echo pseudo-code (Definition 4.2)	$2e$	14 pseudo-code lines
Algorithm 4.3.3 (Graph echo)	$2e + 6$	28 evolution rules
Algorithm 4.3.4 (Cell heights)	$2e + 6$	30 evolution rules

Chapter 5

The Firing Squad Synchronization Problem

The concept of *synchronization* in a system is that all components of the system perform a particular task simultaneously. One of the most well studied synchronization problems in cellular automata is known as the *firing squad synchronization problem* (FSSP). This chapter presents two FSSP solutions that synchronize tree- and digraph-structured simple P systems, respectively.

This chapter is organized as follows. Section 5.1 provides the formulation of the FSSP-equivalent in simple P systems. Section 5.2 describes a phase-based decomposition of the FSSP. Sections 5.3 and 5.4 present FSSP solutions that synchronize graph- and tree-structured simple P systems, respectively. Finally, Section 5.5 summarizes this chapter.

5.1 Synchronization

The *firing squad synchronization problem* (FSSP) was devised by Myhill in 1957 and introduced by Moore [54] in 1964. The original FSSP can be described as follows. Consider a finite one-dimensional array of synchronous finite state machines, i.e. all machines evolve in synchronous steps. The state of each machine at step $t+1$ depends on the states of itself and its (left and right, if any) neighbours at step t. The left-most (or right-most) machine is called the *general* and all other machines are called the *soldiers*. The problem is to specify the states and transition functions for the soldiers, such that the general can make all the soldiers to enter a particular state (called the firing state) *simultaneously* and *for the first time*.

The minimal time FSSP solutions (i.e. $2n-2$ steps, where n is the length of an array) was presented by Goto [38], Waksman [81], Balzer [4] and Mazoyer [52].

5.1.1 Classification of the FSSP

After the original problem was introduced, several generalizations of the problem have been proposed and studied. The FSSP can be classified according to the following criteria.

1. **Number of generals:** [72]

   There can be a single or multiple generals.

2. **Network of the general(s) and soldiers:** [59, 78, 39, 60, 8, 40]

   The general(s) and soldiers are arranged in a line, a tree, a ring or an arbitrary graph.

3. **Position of a general:** [55]

   A general is located at (i) an arbitrary node of a network or (ii) a distinguished node of a network (such as left-most or right-most position in a line and at the root of a tree).

4. **Communications between neighbours:** [61, 29]

   Either bidirectional or unidirectional communications between network nodes.

5.1.2 Formulation of the FSSP for simple P systems

In simple P systems, we can consider the FSSP for a network, where: (i) the network is a tree or an arbitrary graph, (ii) there is a single general, located at the root of the tree or at an arbitrary node of a graph and (iii) bidirectional communications between the neighbours. Under this criteria, the FSSP for simple P systems is formulated as follows.

Problem 5.1. (The formulation of the FSSP for simple P systems) For a simple P system $\Pi = (O, K, \Delta)$, the problem is to specify:

- a finite non-empty alphabet of symbols O,
- a finite set of states Q that contains
  - ○ $s_q \in Q$, a quiescent state and
  - ○ $s_f \in Q$, $s_f \neq s_q$, a firing state,
- a finite linearly ordered set of evolution rules R,

such that, given

- any finite set of cells, K, and
- any weakly connected digraph (K, Δ),

the following three conditions are satisfied:

1. All cells start from state s_q.
2. Cell σ_s is the only cell with an applicable rule in state s_q (i.e. σ_s can evolve) with its initial content (each other cell has an empty content).
3. During the last step of the Π's evolution, all cells enter state s_f, *simultaneously* and *for the first time*.

Minimizing the synchronization time, i.e. the number of steps needed for all cells to enter the firing state, is the main performance criterion considered for a solution to the FSSP of Problem 5.1.

5.1.3 Motivations

There are several applications that require synchronization.

At the biological level, cell synchronization is a process by which cells at different stages of the cell cycle (division, duplication, replication) in a culture are brought to the same phase. There are several biological methods used to synchronize cells at specific cell phases [43]. Once synchronized, monitoring the progression from one phase to another allows us to calculate the timing of specific cells' phases.

A second example relates to operating systems [73], where process synchronization is the coordination of simultaneous threads or processes to complete a task without race conditions. In distributed computing, consensus (e.g. Byzantine agreement problem, see Lynch [50]) may not be possible unless participating processes are synchronized.

Finally, in telecommunication networks [30], we often want to synchronize computers to the same time, i.e. primary reference clocks should be used to avoid local clock offsets.

5.2 Phase-based decomposition of the FSSP

The general issues a "sealed" *order*, which is delivered to all soldiers in a breadth-first search (BFS) manner. The content of this order commands a soldier to enter the firing state. However, the general may not have direct communication channels to all soldiers. In this case, the order is relayed through intermediate soldiers, which results in some soldiers to receive the order before others. To ensure that all soldiers open and execute the order simultaneously, each solider (upon receiving the order) needs to wait until all other soldiers receive the order. Thus, the general needs to incorporate some *timing mechanism* (to the order) that indicates the remaining number of steps (with respect to the global clock) before the received order can be (opened and) executed.

The general incorporates a *hop-counter* to the order, initially set to its eccentricity, that decrements by one in each step. Note that, the eccentricity of the general (i.e. the maximum of the shortest distances from the general to all other soldiers) corresponds to the minimum number of steps needed to deliver the order from the general to all soldiers. Further, for each soldier, the difference between (i) the eccentricity of the general and (ii) the number of steps needed for the order to reach this solider, corresponds to the number of remaining steps before all other soldiers receive the order. This hop-counter decreases according to the distance from the general to a soldier. Hence, the current hop-counter of the received order corresponds to the number of remaining steps before all other soldiers receive the order.

Each of the two FSSP solutions (given in Sections 5.3 and 5.4, respectively) consists of two phases, Phase I and Phase II, where:

1. **Phase I:** the general σ_s determines its eccentricity, $\texttt{ecc}(s)$.

2. **Phase II:** the general issues the order with a hop-counter, initially set to $\texttt{ecc}(s)$.

The two FSSP solutions use different approaches to compute the eccentricity of the general in Phase I. However, the same approach is used in Phase II. Hence, prior to presenting the FSSP solutions, I first present the details of Phase II in Section 5.2.1.

Figure 5.1 illustrates a rooted tree and a graph that are used as an illustration in the FSSP solutions.

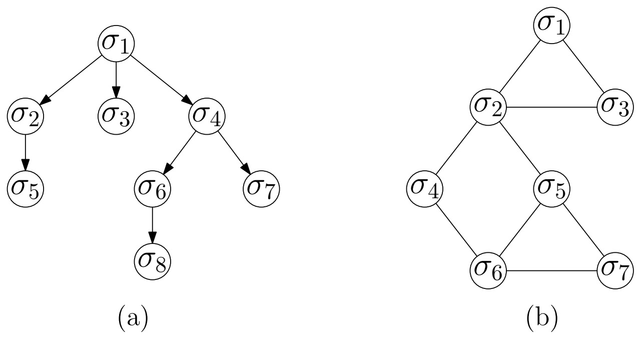


Figure 5.1: A rooted tree and a graph that are used as an illustration in the FSSP solutions.

5.2.1 Algorithm: Phase II—Decrementing hop-counter

This section presents a simple P system with a decrementing hop-counter mechanism that corresponds to Phase II of both FSSP solutions presented in this chapter.

Assume that the order is propagated from the general σ_s, in a breadth-first search (BFS) manner, at step $t \geq 1$, where t corresponds to the step in which Phase I has ended. Each cell σ_i obtains the order at step $t + \texttt{depth}_s(i)$, as described in Proposition 4.28. Further, all cells receive the order by step $t + \texttt{ecc}(s)$. Thus, when cell σ_i obtains the order at step $t + \texttt{depth}_s(i)$, the number of steps that σ_i needs to wait until all other cells receive the order is $(t + \texttt{ecc}(s)) - (t + \texttt{depth}_s(i)) = \texttt{ecc}(s) - \texttt{depth}_s(i)$.

The order is paired with a hop-counter, initially set to the eccentricity of the general, i.e. $\texttt{ecc}(s)$. Starting from the general, when a cell obtains the order, it decrements the current hop-counter by one, before forwarding the received order. Each cell σ_i (in depth $\texttt{depth}_s(i)$) obtains the order with a hop-counter of $\texttt{ecc}(s) - \texttt{depth}_s(i)$ at step $t + \texttt{depth}_s(i)$. Thus, each cell σ_i can recognize that after $\texttt{ecc}(s) - \texttt{depth}_s(i)$ steps, all cell have received the order.

A simple P system $\Pi = (O, K, \Delta)$ that performs the decrementing counter mechanism using a hop-counter is given in Definition 5.2, followed by the analysis of system Π.

Definition 5.2. (A simple P system with a decrementing counter) A simple P system that implements the decrementing counter is $\Pi = (O, K, \Delta)$ (of order n), where:

1. $O = \{h, s, u, v\}$.
2. Δ is a weakly connected digraph, without irreflexive and asymmetric arcs.
3. $K = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, where $\sigma_s \in K$ is the general.

   Each cell $\sigma_i \in K$ is of the form (Q, s_{i0}, w_{i0}, R), where:

   - $Q = \{s_6, s_7, s_8\}$, where s_6 is the initial quiescent state in Phase II and s_8 is the *firing* state.
   - $s_{i0} = s_6$.
   - $w_{i0} = \begin{cases} h^{\texttt{ecc}(s)} s & \text{if } \sigma_i = \sigma_s, \\ \emptyset & \text{if } \sigma_i \neq \sigma_s. \end{cases}$
   - R is a set of evolution rules given below.

6. Rules for state s_6:
   1. $s_6\ hs \rightarrow_{\texttt{max}} s_7\ u\ (s, \updownarrow)$
   2. $s_6\ h \rightarrow_{\texttt{max}} s_7\ v\ (h, \updownarrow)$
   3. $s_6\ s \rightarrow_{\texttt{max}} s_8$

7. Rules for state s_7:
   1. $s_7\ uv \rightarrow_{\texttt{max}} s_7\ u$
   2. $s_7\ u \rightarrow_{\texttt{max}} s_8$
   3. $s_7\ s \rightarrow_{\texttt{max}} s_8$
   4. $s_7\ h \rightarrow_{\texttt{max}} s_8$

Precondition of system Π

Each cell $\sigma_i \in K$ starts with the initial configuration described in Definition 5.2.

Postcondition of system Π

When system Π halts, the configuration of cell $\sigma_i \in K$ is $(s_8, \emptyset)$. Moreover, during the last of the Π's evolution, all cells simultaneously enter firing state s_8 for the first time.

States and symbols of system Π

The multiplicity of symbol h represents the value of the hop-counter. Symbols s and u are used by each cell to determine and record the number of shortest paths from the general, respectively. Symbol v (together with symbol u) is used to indicate the number of remaining steps before all cells receive the firing order. State s_6 is a quiescent state, where each cell remains idle until it receives symbol h. State s_7 is a "wait" state (entered after obtaining the firing order symbols), where each cell waits until all other cells receive the firing order. State s_8 is the firing state.

Tables 5.1 and 5.2 contain the traces of system Π, for the tree of Figure 5.1 (a) and the graph of Figure 5.1 (b), respectively. Figure 5.2 illustrates the visual description of system Π, for the graph of Figure 5.1 (b).

Table 5.1: The evolution traces of the system Π of Definition 5.2, for the tree of Figure 5.1 (a).

Step	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8
0	$s_6\ h^3s$	s_6	s_6	s_6	s_6	s_6	s_6	s_6
1	$s_7\ uv^2$	$s_6\ h^2s$	$s_6\ h^2s$	$s_6\ h^2s$	s_6	s_6	s_6	s_6
2	$s_7\ h^3s^3uv$	$s_7\ uv$	$s_7\ uv$	$s_7\ uv$	$s_6\ hs$	$s_6\ hs$	$s_6\ hs$	s_6
3	$s_7\ h^3s^3u$	$s_7\ su$	$s_7\ u$	$s_7\ s^2u$	$s_7\ u$	$s_7\ u$	$s_7\ u$	$s_6\ s$
4	s_8	s_8	s_8	s_8	s_8	s_8	s_8	s_8

Table 5.2: The traces of the system Π of Definition 5.2, for the graph of Figure 5.1 (b).

Step	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7
0	$s_6\ h^3s$	s_6	s_6	s_6	s_6	s_6	s_6
1	$s_7\ uv^2$	$s_6\ h^2s$	$s_6\ h^2s$	s_6	s_6	s_6	s_6
2	$s_7\ h^2s^2uv$	$s_7\ hsuv$	$s_7\ hsuv$	$s_6\ hs$	$s_6\ hs$	s_6	s_6
3	$s_7\ h^2s^2u$	$s_7\ hs^3u$	$s_7\ hsu$	$s_7\ u$	$s_7\ u$	$s_6\ s^2$	$s_6\ s$
4	s_8	s_8	s_8	s_8	s_8	s_8	s_8

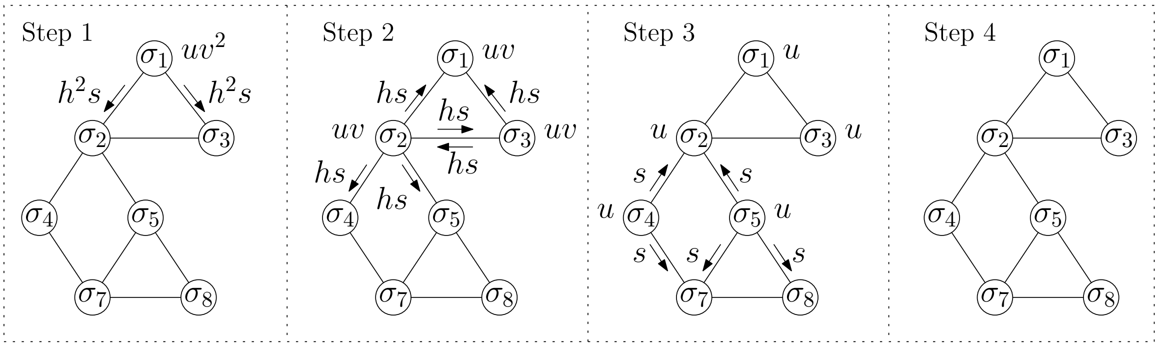


Figure 5.2: The propagation of the order from the general σ_1, with an initial hop-count of $\mathtt{ecc}(1) = 3$.

Correctness of construction of system Π

At the start of Phase II, the general σ_s sends: (i) $\mathtt{ecc}(s) - 1$ copies of symbol h (rule 6.1) and (ii) one copy of symbol s to each of its neighbours (rule 6.2). The one copy of symbol s is propagated as described in Algorithm 4.3.1, which results each cell σ_i to receive $\mathtt{paths}_s(i)$ copies of symbol s. Further, $\mathtt{ecc}(s) - 1$ copies of symbol h are also propagated as described in Algorithm 4.3.1, such that each cell receives $(\mathtt{ecc}(s) - 1)\mathtt{paths}_s(i)$ copies of symbol h. However, when σ_i receives $k \leq (\mathtt{ecc}(s) - 1)\mathtt{paths}_s(i)$ copies of symbol h, σ_i sends $k - \mathtt{paths}_s(i)$ copies of symbol h to each of its neighbours. Specifically:

- At step $\mathtt{depth}_s(i)$, cell σ_i receives:
  - ◦ $\mathtt{paths}_s(i)$ copies of symbol s (Propositions 5.5) and
  - ◦ $(\mathtt{ecc}(s) - \mathtt{depth}_s(i))\mathtt{paths}_s(i)$ copies of symbol h (Propositions 5.4).

- Then, at step $\texttt{depth}_s(i)+1$, cell σ_i sends:
    - $\texttt{paths}_s(i)$ copies of symbol s to each of its neighbours (rule 6.1) (Propositions 5.5) and
    - $(\texttt{ecc}(s)-\texttt{depth}_s(i)-1)\texttt{paths}_s(i)$ copies of symbol h to each of its neighbours (rule 6.2) (Propositions 5.4).

At step $\texttt{depth}_s(i)$, for σ_i, the value $\texttt{ecc}(s)-\texttt{depth}_s(i)$ corresponds to the number of remaining steps until all other cells receive the order. Thus, σ_i needs to extract the value $\texttt{ecc}(s)-\texttt{depth}_s(i)$ from $(\texttt{ecc}(s)-\texttt{depth}_s(i))\texttt{paths}_s(i)$, such that, from step $\texttt{depth}_s(i)$, σ_i can remain idle for the next $\texttt{ecc}(s)-\texttt{depth}_s(i)$ steps before it enters the firing state.

If $\texttt{depth}_s(i)=\texttt{ecc}(s)$, then σ_i receives $\texttt{paths}_s(i)$ copies of symbol s and zero copy of symbol h. In this case, σ_i is one of the last cells to receive the order. Thus, at step $\texttt{depth}_s(i)+1=\texttt{ecc}(s)+1$, σ_i enters the firing state s_8 (rule 6.3); at the same time, σ_i discards $\texttt{paths}_s(i)$ copies of symbol s.

If $\texttt{depth}_s(i)<\texttt{ecc}(s)$, then σ_i obtains $(\texttt{ecc}(s)-\texttt{depth}_s(i))\texttt{paths}_s(i)$ copies of symbol h and $\texttt{paths}_s(i)$ copies of symbol s. In this case, at step $\texttt{depth}_s(i)+1$, σ_i sends symbol s and h as described earlier, and at the same time, σ_i produces: (i) $\texttt{paths}_s(i)$ copies of symbol u (rule 6.1) and (ii) $(\texttt{ecc}(s)-\texttt{depth}_s(i))\texttt{paths}_s(i)$ copies of symbol v (rule 6.2). From step $\texttt{depth}_s(i)+2$, σ_i rewrites every copy of multiset uv (i.e. one copy of symbol u and one copy of symbol v) into one copy of symbol u (rule 7.1). Since σ_i has $\texttt{paths}_s(i)$ copies of symbol u, consuming all copies of symbol v take $\texttt{ecc}(s)-\texttt{depth}_s(i)$ steps. Thus, σ_i can remain idle for $\texttt{ecc}(s)-\texttt{depth}_s(i)$ steps. Then, at step $\texttt{depth}_s(i)+(\texttt{ecc}(s)-\texttt{depth}_s(i))+1=\texttt{ecc}(s)+1$, σ_i enters the firing state s_8 (rule 7.2) and discards $\texttt{paths}_s(i)$ copies of symbol u; further, σ_i discards symbols s and h that are received from its peers and successors (rules 7.3 and 7.4).

Proposition 5.3. All cells of system Π enter the firing state at step $\texttt{ecc}(s)+1$ with empty contents. Thus, the postcondition of Π is satisfied.

Proposition 5.4. Cell σ_i receives $(\texttt{ecc}(g)-\texttt{depth}_s(i))\texttt{paths}_s(i)$ copies of symbol h from its predecessors at step $\texttt{depth}_s(i)$, and sends $(\texttt{ecc}(s)-\texttt{depth}_s(i)-1)\texttt{paths}_s(i)$ copies of symbol h to each of its successors, if any, at step $\texttt{depth}_s(i)+1$.

Proof. Proof by induction, on depth $m = \texttt{depth}_s(i) \geq 1$.

At step 1, the general sends $\texttt{ecc}(s) - 1$ copies of symbol h to each of its neighbours. Thus, at step 1, each cell σ_k in depth 1 receives $\texttt{ecc}(s) - 1 = (\texttt{ecc}(s) - \texttt{depth}_s(k))\texttt{paths}_s(k)$ copies of symbol h. Then, at step 2, cell σ_k sends $\texttt{ecc}(s) - 2 = (\texttt{ecc}(s) - \texttt{depth}_s(k) - 1)\texttt{paths}_s(k)$ copies of symbol h to each of its neighbours.

Assume that the induction hypothesis holds for each cell σ_j at depth m. Consider cell σ_i in $m' = m + 1$. By induction hypothesis, at step $m + 1$, each $\sigma_j \in \texttt{Pred}_s(i)$ sends $(\texttt{ecc}(s) - \texttt{depth}_s(i) - 1)\texttt{paths}_s(i)$ copies of symbol h to each of its neighbours. Thus, at step $m + 1 = m'$, σ_i receives $\sum_{\sigma_j \in \texttt{Pred}_s(i)}(\texttt{ecc}(s) - m - 1)\texttt{paths}_s(j)$ copies of symbol h, where

$$\begin{aligned}
\textstyle\sum_{\sigma_j \in \texttt{Pred}_s(i)}(\texttt{ecc}(s) - m - 1)\texttt{paths}_s(j) &= (\texttt{ecc}(s) - m - 1) \cdot \textstyle\sum_{\sigma_j \in \texttt{Pred}_s(i)} \texttt{paths}_s(j) \\
&= (\texttt{ecc}(s) - m - 1)\texttt{paths}_s(i) \\
&= (\texttt{ecc}(s) - (\texttt{depth}_s(i) - 1) - 1)\texttt{paths}_s(i) \\
&= (\texttt{ecc}(s) - \texttt{depth}_s(i))\texttt{paths}_s(i)
\end{aligned}$$

Then, at step $\texttt{depth}_s(i) + 1$, σ_i removes $\texttt{paths}_s(i)$ copies of symbol h and sends the remaining $(\texttt{ecc}(s) - \texttt{depth}_s(i) - 1)\texttt{paths}_s(i)$ copies of symbol h to each of its neighbours (which include its successors). □

Proposition 5.5. Cell σ_i receives $\texttt{paths}_s(i)$ copies of symbol s from its predecessors and sends $\texttt{paths}_s(i)$ copies of symbol s to each of its successor.

Proof. Follows from Proposition 4.22. □

Time and message complexities of system Π

The time and message complexities of system Π are indicated in Propositions 5.6 and 5.7, respectively.

Proposition 5.6. System Π halts in $\texttt{ecc}(s) + 1$ steps.

Proof. The general σ_s sends the firing order in a BFS manner. Each cell σ_i receives the firing order at step $\texttt{depth}_s(i)$ with the counter of value $(\texttt{ecc}(s) - \texttt{depth}_s(i))\texttt{paths}_s(i)$. From step $\texttt{depth}_s(i)$, cell σ_i decrements the current counter by $\texttt{paths}_s(i)$ in each step. The value of the counter becomes 0 at step $\texttt{depth}_s(i) + (\texttt{ecc}(s) - \texttt{depth}_s(i)) = \texttt{ecc}(s)$.

Thus, at step $\texttt{ecc}(s)+1$, all cells enter the firing state. □

Proposition 5.7. The total number of symbols exchanged between cells is $\sum_{\sigma_i \in K}((\texttt{ecc}(s) - \texttt{depth}_s(i) - 1)\texttt{paths}_s(i) \cdot |\texttt{Neighbour}(i)|)$.

Proof. Each cell σ_i sends $(\texttt{ecc}(s) - \texttt{depth}_s(i) - 1)\texttt{paths}_s(i)$ copies of symbol h to each of its neighbours (Proposition 5.4). Hence, σ_i sends $(\texttt{ecc}(s) - \texttt{depth}_s(i) - 1)\texttt{paths}_s(i) \cdot |\texttt{Neighbour}(i)|$ copies of symbol h.

Further, σ_i sends $\texttt{paths}_s(i)$ copies of symbol s to each of its neighbours (Proposition 5.5). Hence, σ_i sends $\texttt{paths}_s(i) \cdot |\texttt{Neighbour}(i)|$ copies of symbol s.

Thus, the total number of symbols sent by σ_i is $(\texttt{ecc}(s) - \texttt{depth}_s(i))\texttt{paths}_s(i) \cdot |\texttt{Neighbour}(i)|$.

Therefore, the total number of symbols transferred between cells is $\sum_{\sigma_i \in K}((\texttt{ecc}(s) - \texttt{depth}_s(i))\texttt{paths}_s(i) \cdot |\texttt{Neighbour}(i)|)$. □

5.3 Static FSSP solution for graphs

This section presents an FSSP solution that synchronizes digraph-structured simple P systems. This solution is considered *static*, in that the position of the general remains the same. The simple P system of Definition 5.8 solves the FSSP for any arbitrary digraph, where the position of the general remains the same. This FSSP solution consists of two phases, where: (i) **Phase I** (derived from Algorithm 4.3.3) determines the eccentricity of the general, and (ii) **Phase II** (Algorithm 5.2.1) the general sends the order with a hop-counter, initially set to its eccentricity.

Definition 5.8. (Simple P system for solving FSSP) A simple P system that solves the FSSP for any arbitrary weakly-connected digraph is $\Pi = (O, K, \Delta)$, where:

1. $O = \{a, b, c, d, e, h, o, r, s, v, x\}$.

2. Δ is a weakly connected digraph.

3. $K = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, where $\sigma_s \in K$ is the general.

   Each cell $\sigma_i \in K$ is of the form (Q, s_{i0}, w_{i0}, R), where:

- $Q = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$, where s_0 is the initial quiescent state and s_8 is the firing state.
- $s_{i0} = s_0$.
- $w_{i0} = \begin{cases} s & \text{if } \sigma_i = \sigma_s, \\ \emptyset & \text{if } \sigma_i \neq \sigma_s. \end{cases}$
- R is defined by the following set of evolution rules.

Rules used in Phase I: (derived from Algorithm 4.3.3)

0. Rules for state s_0:
   1. $s_0\ s \rightarrow_{\texttt{min}} s_1\ as\ (o, \updownarrow)$
   2. $s_0\ x \rightarrow_{\texttt{min}} s_1\ ab\ (o, \updownarrow)$
   3. $s_0\ o \rightarrow_{\texttt{min}} s_1\ eb\ (x, \updownarrow)$
   4. $s_0\ x \rightarrow_{\texttt{max}} s_1\ b$
   5. $s_0\ o \rightarrow_{\texttt{max}} s_1\ b$

1. Rules for state s_1:
   1. $s_1\ a \rightarrow_{\texttt{min}} s_2\ a$
   2. $s_1\ e \rightarrow_{\texttt{min}} s_2\ e$
   3. $s_1\ o \rightarrow_{\texttt{max}} s_2\ c$
   4. $s_1\ x \rightarrow_{\texttt{max}} s_2\ c$

2. Rules for state s_2:
   1. $s_2\ a \rightarrow_{\texttt{min}} s_3\ a$
   2. $s_2\ e \rightarrow_{\texttt{min}} s_4\ a$
   3. $s_2\ x \rightarrow_{\texttt{max}} s_3\ d$
   4. $s_2\ o \rightarrow_{\texttt{max}} s_4\ d$

3. Rules for state s_3:
   1. $s_3\ t \rightarrow_{\texttt{min}} s_5$
   2. $s_3\ xd \rightarrow_{\texttt{max}} s_3$
   3. $s_3\ ads \rightarrow_{\texttt{min}} s_3\ adsv$
   4. $s_3\ ad \rightarrow_{\texttt{min}} s_3\ ad$
   5. $s_3\ a \rightarrow_{\texttt{min}} s_3\ at\ (o, \updownarrow)$
   6. $s_3\ vv \rightarrow_{\texttt{max}} s_5\ h$

4. Rules for state s_4:
   1. $s_4\ t \rightarrow_{\texttt{min}} s_5$
   2. $s_4\ od \rightarrow_{\texttt{max}} s_4$
   3. $s_4\ ad \rightarrow_{\texttt{min}} s_4\ ad$
   4. $s_4\ a \rightarrow_{\texttt{min}} s_4\ at\ (x, \updownarrow)$

5. Rules for state s_5:
   1. $s_5\ ob \rightarrow_{\texttt{max}} s_5$
   2. $s_5\ xb \rightarrow_{\texttt{max}} s_5$
   3. $s_5\ oc \rightarrow_{\texttt{max}} s_5$
   4. $s_5\ xc \rightarrow_{\texttt{max}} s_5$
   5. $s_5\ ab \rightarrow_{\texttt{min}} s_5\ ab$
   6. $s_5\ ac \rightarrow_{\texttt{min}} s_5\ ac$
   7. $s_5\ a \rightarrow_{\texttt{min}} s_6$

Rules used in Phase II: (Algorithm 5.2.1)

6. Rules for state s_6:
   1. $s_6\ hs \rightarrow_{\texttt{max}} s_7\ u\ (s, \updownarrow)$
   2. $s_6\ h \rightarrow_{\texttt{max}} s_7\ v\ (h, \updownarrow)$
   3. $s_6\ s \rightarrow_{\texttt{max}} s_8$

7. Rules for state s_7:
   1. $s_7\ uv \rightarrow_{\texttt{max}} s_7\ u$
   2. $s_7\ u \rightarrow_{\texttt{max}} s_8$
   3. $s_7\ s \rightarrow_{\texttt{max}} s_8$
   4. $s_7\ h \rightarrow_{\texttt{max}} s_8$

5.3.1 Algorithm: Phase I—Compute general's eccentricity

Phase I (derived from Algorithm 4.3.3) of system Π finds the eccentricity of the general σ_s, $\texttt{ecc}(s)$. The details of Phase I of system Π are as follow.

Precondition of Phase I of system Π

Each cell $\sigma_i \in K$ starts with the initial configuration described in Definition 5.8.

Postcondition of Phase I of system Π

The configuration of each cell $\sigma_i \in K$ at the end of Phase I is (s_6, w_i), where

- $|w_i|_s = 1$, if $\sigma_i = \sigma_s$.
- $|w_i|_h = \texttt{height}_s(i) = \texttt{ecc}(s)$, if $\sigma_i = \sigma_s$.

States and symbols of Phase I of system Π

The resulting multiplicity of symbol h in the general σ_s corresponds to $\texttt{ecc}(s)$. The meaning of other symbols and states are described in Algorithm 4.3.3.

Evolution rules of Phase I of system Π

The set of evolution rules given in Definition 5.8 is the set of evolution rules of Algorithm 4.3.3, with the following changes: (i) rule 0.1 is modified to keep symbol s in σ_s, (ii) rule 3.3 is added to accumulate $2 \cdot \texttt{height}_s(s)$ copies of symbol v in σ_s and (iii) rule 3.6 is added to rewrite every two copies of symbol v into one copy of symbol h in σ_s.

Overview of Phase I of system Π

In Algorithm 4.3.3, each cell σ_i (i) obtains broadcast symbols at step $\texttt{depth}_s(i)$ (Proposition 4.28) and (ii) receives convergecast symbols from all its successors by step $\texttt{depth}_s(i) + 2 \cdot \texttt{height}_s(i) + 3$ (Proposition 4.38).

The general σ_s can accumulate $2 \cdot \texttt{height}_s(s)$ copies of symbol v, if σ_s produces one copy of symbol v in each step, from step t_s until step t'_s (both inclusive), where:

- $t_s = \texttt{depth}_s(s) + 3 = 3$, i.e. two steps after σ_s sends a broadcast symbol to its successors,
- $t'_s = \texttt{depth}_s(s) + 2 \cdot \texttt{height}_s(s) + 3 = 2 \cdot \texttt{height}_s(s) + 3$, i.e. when σ_s receives convergecast symbols from all its successors.

Then, σ_s can obtain $\texttt{height}_s(s)$ copies of symbol h by rewriting every two copies of symbol v into one copy of symbol h. Note that, $\texttt{height}_s(s) = \texttt{ecc}(s)$. Thus, σ_s can determine its eccentricity, $\texttt{ecc}(s)$, according to the resulting multiplicity of symbol h.

Table 5.3 contains the traces of Algorithm 5.3.1 (i.e. system Π in Phase I), for the graph of Figure 5.1 (b). Figure 5.3 provides a visual description of Algorithm 5.3.1.

Rules description of Phase I of system Π

For the general σ_s produces one copy of symbol v from step t_s until step t'_s (rule 3.3), both inclusive, where steps t_s and t'_s are described in the overview. After receiving convergecast symbols from the successors, σ_s rewrites every two copies of symbol v into one copy of symbol h (rule 3.6). The other rules are described in Algorithm 4.3.3.

Correctness of construction of Phase I of system Π

From Algorithm 4.3.3, all cells end in state s_6 with empty contents. The changes to the evolution rules of Algorithm 4.3.3 enable the general σ_s to (i) keep one copy of symbol s and (ii) obtain $\texttt{ecc}(s)$ copies of symbol h. Thus, the postcondition of Phase I of system Π is satisfied.

Time and message complexities of Phase I of system Π

The time and message complexities of Phase I of system Π are indicated in Propositions 5.9 and 5.10, respectively.

Proposition 5.9. Phase I of system Π takes $2 \cdot \texttt{ecc}(s) + 6$ steps.

Table 5.3: The traces of Phase I of the system Π, that determines the eccentricity of the general, for the graph of Figure 5.1 (b), where σ_1 is the general. The resulting multiplicity of symbol h in cell σ_1 corresponds to the eccentricity of σ_1. The eccentricity of σ_1 is three. Thus, at step 12, σ_1 contains three copies of symbol h.

Step	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7
0	$s_0\ s$	s_0	s_0	s_0	s_0	s_0	s_0
1	$s_1\ as$	$s_0\ o$	$s_0\ o$	s_0	s_0	s_0	s_0
2	$s_2\ asx^2$	$s_1\ bex$	$s_1\ bex$	$s_0\ x$	$s_0\ x$	s_0	s_0
3	$s_3\ ad^2s$	$s_2\ bceo^2$	$s_2\ bce$	$s_1\ ab$	$s_1\ ab$	$s_0\ o^2$	$s_0\ o$
4	$s_3\ ad^2sv$	$s_4\ abcd^2$	$s_4\ abc$	$s_2\ abx$	$s_2\ abx^2$	$s_1\ b^2ex$	$s_1\ bex$
5	$s_3\ ad^2sv^2x$	$s_4\ abcd^2x$	$s_4\ abct$	$s_3\ abd$	$s_3\ abd^2$	$s_2\ b^2ce$	$s_2\ bce$
6	$s_3\ adsv^3$	$s_4\ abcd^2x$	$s_5\ abc$	$s_3\ abd$	$s_3\ abd^2$	$s_4\ ab^2c$	$s_4\ abc$
7	$s_3\ adsv^4$	$s_4\ abcd^2x$	$s_5\ abc$	$s_3\ abdx$	$s_3\ abd^2x^2$	$s_4\ ab^2ctx$	$s_4\ abctx$
8	$s_3\ adsv^5$	$s_4\ abcd^2o^2x$	$s_5\ abc$	$s_3\ abt$	$s_3\ abt$	$s_5\ ab^2co^2x$	$s_5\ abcox$
9	$s_3\ adsv^6x$	$s_4\ abctx$	$s_5\ abcx$	$s_5\ abx$	$s_5\ abx$	$s_5\ a$	$s_5\ a$
10	$s_3\ astv^6$	$s_5\ abcox$	$s_5\ aco$	$s_5\ a$	$s_5\ a$	s_6	s_6
11	$s_5\ ah^3s$	$s_5\ a$	$s_5\ a$	s_6	s_6	s_6	s_6
12	$\boldsymbol{s_6\ h^3s}$	s_6	s_6	s_6	s_6	s_6	s_6

Proof. Follows from Proposition 4.41. □

Proposition 5.10. The total number of symbols that are transferred between cells in Phase I of Π is $4 \cdot |\Delta|$.

Proof. Follows from Proposition 4.42. □

5.3.2 Algorithm: Phase II—Propagation of the order

Phase II (Algorithm 5.2.1) of system Π starts at the end of Phase I, where the general σ_s issues the order, paired with a hop-count with an initial value set to its eccentricity $\texttt{ecc}(s)$, to inform all soldiers the correct step to enter the firing state. Refer Section 5.2.1 for full details of Phase II.

5.3.3 Summary of Phase I and Phase II

The time and message complexities of system Π are indicated in Propositions 5.11 and 5.12, respectively. Table 5.4 contains the traces of system Π in Phase I and Phase II, for the graph of Figure 5.1 (b).

Theorem 5.11. System Π halts (i.e. synchronizes) in $3 \cdot \texttt{ecc}(s) + 7$ steps.

Proof. Phase I takes $2 \cdot \texttt{ecc}(s) + 6$ steps (Proposition 5.9). Phase II takes $\texttt{ecc}(s) + 1$ steps (Proposition 5.6). Thus, in total, Π synchronizes in $3 \cdot \texttt{ecc}(s) + 7$ steps. $\square$

Theorem 5.12. The total number of symbols transferred between cells of Π is $4 \cdot |\Delta| + \sum_{\sigma_i \in K}((\texttt{ecc}(s) - \texttt{depth}_s(i))\texttt{paths}_s(i) \cdot |\texttt{Neighbour}(i)|)$.

Proof. The number of symbols transferred between cells in Phase I is $4 \cdot |\Delta|$ (Proposition 5.10). The number of symbols transferred between cell in Phase II is $\sum_{\sigma_i \in K}((\texttt{ecc}(s) - \texttt{depth}_s(i))\texttt{paths}_s(i) \cdot |\texttt{Neighbour}(i)|)$ (Proposition 5.7).

Therefore, the total number symbols transferred between cells is $4 \cdot |\Delta| + \sum_{\sigma_i \in K}((\texttt{ecc}(s) - \texttt{depth}_s(i))\texttt{paths}_s(i) \cdot |\texttt{Neighbour}(i)|)$. $\square$

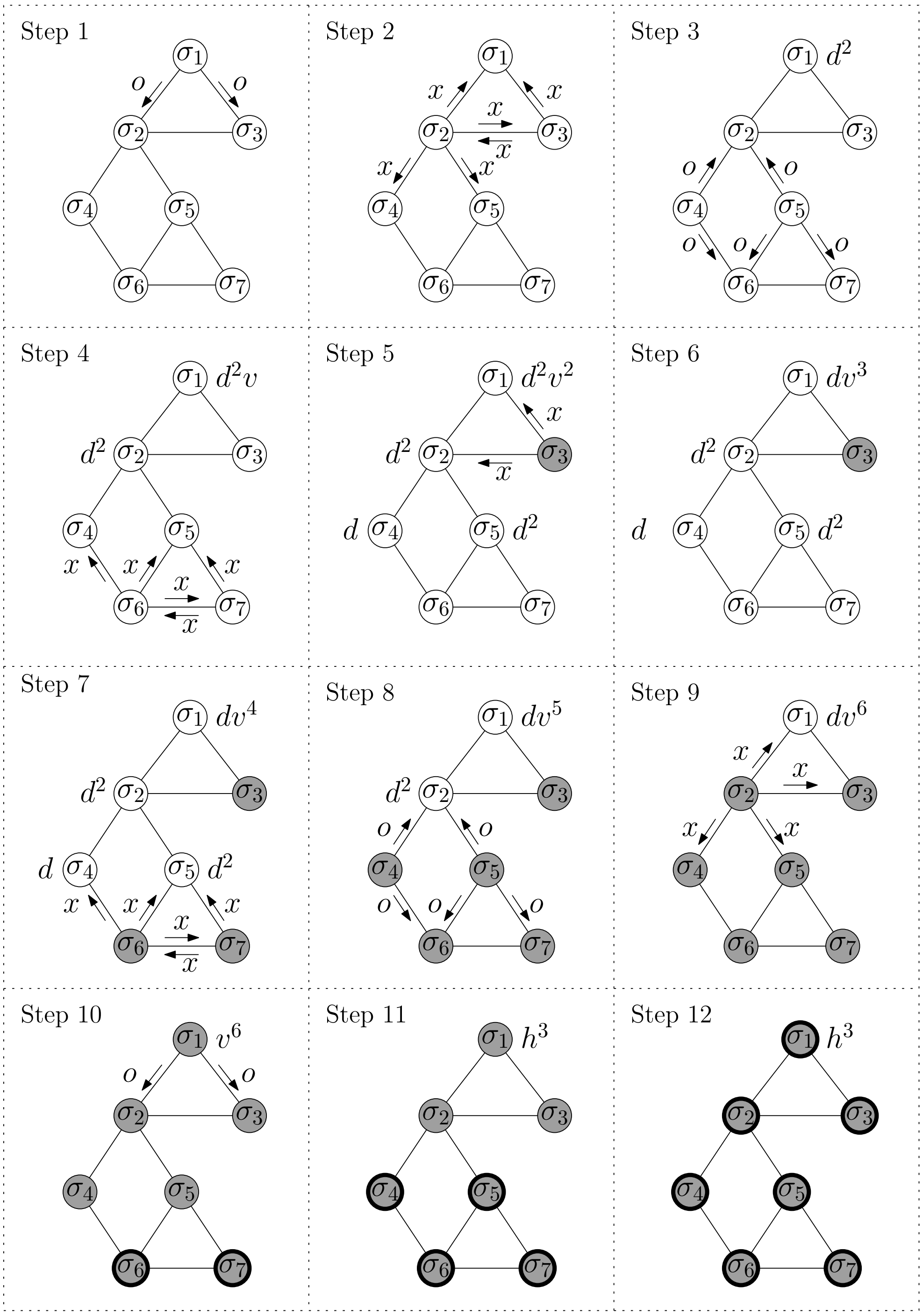


Figure 5.3: The general σ_1 produces one copy of symbol v in each step, from step t_1 until step t'_1, where t_1 is the two steps after it sends its broadcast symbol o and t'_1 is the step in which it receives convergecast symbol x from all its successors. Then, at step $t'_1 + 1$, σ_1 rewrites every two copies of symbol v into one copy of symbol h. The resulting multiplicity of symbol h corresponds to the eccentricity of σ_1. The multiplicity of symbol d in each cell σ_i, $1 \leq i \leq 7$, indicates the remaining number of successors that have not sent convergecast symbol to σ_i.

Table 5.4: The traces of the static FSSP solution (i.e. system Π of Definition 5.8), where cell σ_1 is the general. The shaded tables cells indicate the start of Phase II (equivalently, the end of Phase I). At the last step, all cells simultaneously enter the firing state s_8 for the first time. Note, the table rows for Phase I are same as Table 5.3 and the table rows for Phase II are same as Table 5.2.

Step	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7
0	$s_0\ s$	s_0	s_0	s_0	s_0	s_0	s_0
1	$s_1\ as$	$s_0\ o$	$s_0\ o$	s_0	s_0	s_0	s_0
2	$s_2\ asx^2$	$s_1\ bex$	$s_1\ bex$	$s_0\ x$	$s_0\ x$	s_0	s_0
3	$s_3\ ad^2s$	$s_2\ bceo^2$	$s_2\ bce$	$s_1\ ab$	$s_1\ ab$	$s_0\ o^2$	$s_0\ o$
4	$s_3\ ad^2sv$	$s_4\ abcd^2$	$s_4\ abc$	$s_2\ abx$	$s_2\ abx^2$	$s_1\ b^2ex$	$s_1\ bex$
5	$s_3\ ad^2sv^2x$	$s_4\ abcd^2x$	$s_4\ abct$	$s_3\ abd$	$s_3\ abd^2$	$s_2\ b^2ce$	$s_2\ bce$
6	$s_3\ adsv^3$	$s_4\ abcd^2x$	$s_5\ abc$	$s_3\ abd$	$s_3\ abd^2$	$s_4\ ab^2c$	$s_4\ abc$
7	$s_3\ adsv^4$	$s_4\ abcd^2x$	$s_5\ abc$	$s_3\ abdx$	$s_3\ abd^2x^2$	$s_4\ ab^2ctx$	$s_4\ abctx$
8	$s_3\ adsv^5$	$s_4\ abcd^2o^2x$	$s_5\ abc$	$s_3\ abt$	$s_3\ abt$	$s_5\ ab^2co^2x$	$s_5\ abcox$
9	$s_3\ adsv^6x$	$s_4\ abctx$	$s_5\ abcx$	$s_5\ abx$	$s_5\ abx$	$s_5\ a$	$s_5\ a$
10	$s_3\ astv^6$	$s_5\ abcox$	$s_5\ aco$	$s_5\ a$	$s_5\ a$	s_6	s_6
11	$s_5\ ah^3s$	$s_5\ a$	$s_5\ a$	s_6	s_6	s_6	s_6
12	$s_6\ h^3s$	s_6	s_6	s_6	s_6	s_6	s_6
13	$s_7\ uv^2$	$s_6\ h^2s$	$s_6\ h^2s$	s_6	s_6	s_6	s_6
14	$s_7\ h^2s^2uv$	$s_7\ hsuv$	$s_7\ hsuv$	$s_6\ hs$	$s_6\ hs$	s_6	s_6
15	$s_7\ h^2s^2u$	$s_7\ hs^3u$	$s_7\ hsu$	$s_7\ u$	$s_7\ u$	$s_6\ s^2$	$s_6\ s$
16	s_8	s_8	s_8	s_8	s_8	s_8	s_8

5.4 Adaptive FSSP solution for trees

This section presents an FSSP solution that synchronizes tree-structured simple P systems, where the initial general is the tree root and the final general is a tree centre, i.e. a cell with the eccentricity that equals the tree radius.

A well-known strategy used in one-dimensional cellular automata for solving the FSSP is to locate the *midpoint* (i.e. a centre) of a one-dimensional array [80], which was used in the minimal time FSSP solutions of cellular automata [38, 81, 4, 52]. By locating one of this array's centres, the number of steps needed to propagate the order in Phase II can be reduced, since the distance from a centre to all other cells is minimal. This strategy suggests that finding a centre of a given network should be one of the main considerations in an FSSP solution design.

In other network structures, such as trees and digraphs, sending the order from a centre can reduce the propagation time. Thus, for a given network, first find a soldier located at a network centre, if any, then "promote" this soldier as the *new general* and "demote" the original general to a soldier, such that the eccentricity of the new general equals the network radius. If the initial general is already a centre, then it continues to function as the general. The problem of finding a centre of a network corresponds to the *centre problem* of the *facility location problems* [41].

The simple P system of Definition 5.13 solves the FSSP for any arbitrary tree. This FSSP solution consists of two phases, where: (i) **Phase I** locates a cell with eccentricity that equals the tree radius, sets this cell as the new general and computes its eccentricity, and (ii) **Phase II** (Algorithm 5.2.1) the new general (which could be the initial general) found in Phase I sends the order with a hop-counter, initially set to its eccentricity. Note, the centre cell found in Phase I is referred to as the *middle* cell.

Definition 5.13. (Simple P system for solving the FSSP) A simple P system that solves the FSSP for any arbitrary tree is $\Pi = (O, K, \Delta)$, where:

1. $O = \{a, b, c, e, h, o, s, t, u, v\}$.

2. Δ is a rooted tree, where the general $\sigma_s \in K$ is the tree root.

3. $K = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, where $\sigma_s \in K$ is the general.

Each cell $\sigma_i \in K$ is of the form (Q, s_{i0}, w_{i0}, R), where:

- $Q = \{s_0, s_1, s_2, s_3, s_6, s_7, s_8\}$, where s_0 is the initial quiescent state and s_8 is the *firing* state.
- $s_{i0} = s_0$.
- $w_{i0} = \begin{cases} s & \text{if } \sigma_i = \sigma_s, \\ \emptyset & \text{if } \sigma_i \neq \sigma_s. \end{cases}$
- R is a set of evolution rules given below.

Rules used in Phase I: (find a tree centre)

0. Rules for state s_0:
   1. $s_0\ s \rightarrow_{\texttt{min}} s_1\ ao\ (b, \downarrow)$
   2. $s_0\ b \rightarrow_{\texttt{min}} s_1\ a\ (c, \uparrow)\ (b, \downarrow)$

1. Rules for state s_1:
   1. $s_1\ a \rightarrow_{\texttt{min}} s_2\ a$

3. Rules for state s_3:
   1. $s_3\ ao \rightarrow_{\texttt{min}} s_6$
   2. $s_3\ ae \rightarrow_{\texttt{min}} s_6$

2. Rules for state s_2:
   1. $s_2\ aot \rightarrow_{\texttt{min}} s_6\ a$
   2. $s_2\ at \rightarrow_{\texttt{min}} s_3\ a$
   3. $s_2\ ce \rightarrow_{\texttt{max}} s_2$
   4. $s_2\ acc \rightarrow_{\texttt{min}} s_2\ acch$
   5. $s_2\ acooo \rightarrow_{\texttt{min}} s_2\ aet\ (o, \downarrow)$
   6. $s_2\ aco \rightarrow_{\texttt{min}} s_2\ achoo$
   7. $s_2\ aooo \rightarrow_{\texttt{min}} s_2\ aot\ (e, \downarrow)$
   8. $s_2\ ao \rightarrow_{\texttt{min}} s_2\ aot\ (e, \downarrow)$
   9. $s_2\ ac \rightarrow_{\texttt{min}} s_2\ ach$
   10. $s_2\ a \rightarrow_{\texttt{min}} s_2\ at\ (e, \updownarrow)$
   11. $s_2\ hh \rightarrow_{\texttt{max}} s_6\ h$
   12. $s_2\ h \rightarrow_{\texttt{max}} s_3$

Rules used in Phase II: (Algorithm 5.2.1)

6. Rules for state s_6:
   1. $s_6\ hs \rightarrow_{\texttt{max}} s_7\ u\ (s, \updownarrow)$
   2. $s_6\ h \rightarrow_{\texttt{max}} s_7\ v\ (h, \updownarrow)$
   3. $s_6\ s \rightarrow_{\texttt{max}} s_8$

7. Rules for state s_7:
   1. $s_7\ uv \rightarrow_{\texttt{max}} s_7\ u$
   2. $s_7\ u \rightarrow_{\texttt{max}} s_8$
   3. $s_7\ s \rightarrow_{\texttt{max}} s_8$
   4. $s_7\ h \rightarrow_{\texttt{max}} s_8$

5.4.1 Algorithm: Phase I—Find a tree centre

This phase performs a breadth-first search (BFS) from the root, which propagates symbol b from the root to all other cells. When the symbol b from the BFS reaches a leaf cell, symbol e is *reflected* back up the tree, in the same manner as described Algorithm 4.2.2.

Finding a tree centre (i.e. the middle cell) is based on the following idea. Assume that, at step t, the root receives symbol e from all its children, except one child (σ_c). If the root does not receive symbol e from cell σ_c by step $t+2$, then the height of σ_c must be at least two greater than all other children of the root. In this case, a tree centre is must be contained inside the subtree rooted at cell σ_c. At step $t+3$, the root sends another symbol, o, to σ_c, as the "next general" notification. Thus, σ_c becomes the new general at step $t+3$ and σ_c searches for a subtree that contains a tree centre in the same manner. Note, the propagation speed of symbol o is 1/3 of the propagation speed of symbols b and c.

A visual description of the propagations of symbols b, e and o is provided in Figure 5.4 (for a tree with one centre) and Figure 5.5 (for a tree with two centres).

Precondition of system Π

Each cell starts with the initial configuration described in Definition 5.13.

Postcondition of system Π

When system Π halts the final configuration of cell $\sigma_i \in K$ is (s_6, w_i), where

- $|w_i|_s = 1$ and $|w_i|_h = \texttt{height}_s(i)$, if $\sigma_i = \sigma_m$.
- $w_i = \emptyset$, if $\sigma_i \neq \sigma_m$.

States and symbols of system Π

Symbol h represents one unit of tree height. Symbol o represents the "next general" notification sent from the current general to the next general, i.e. a cell that contains

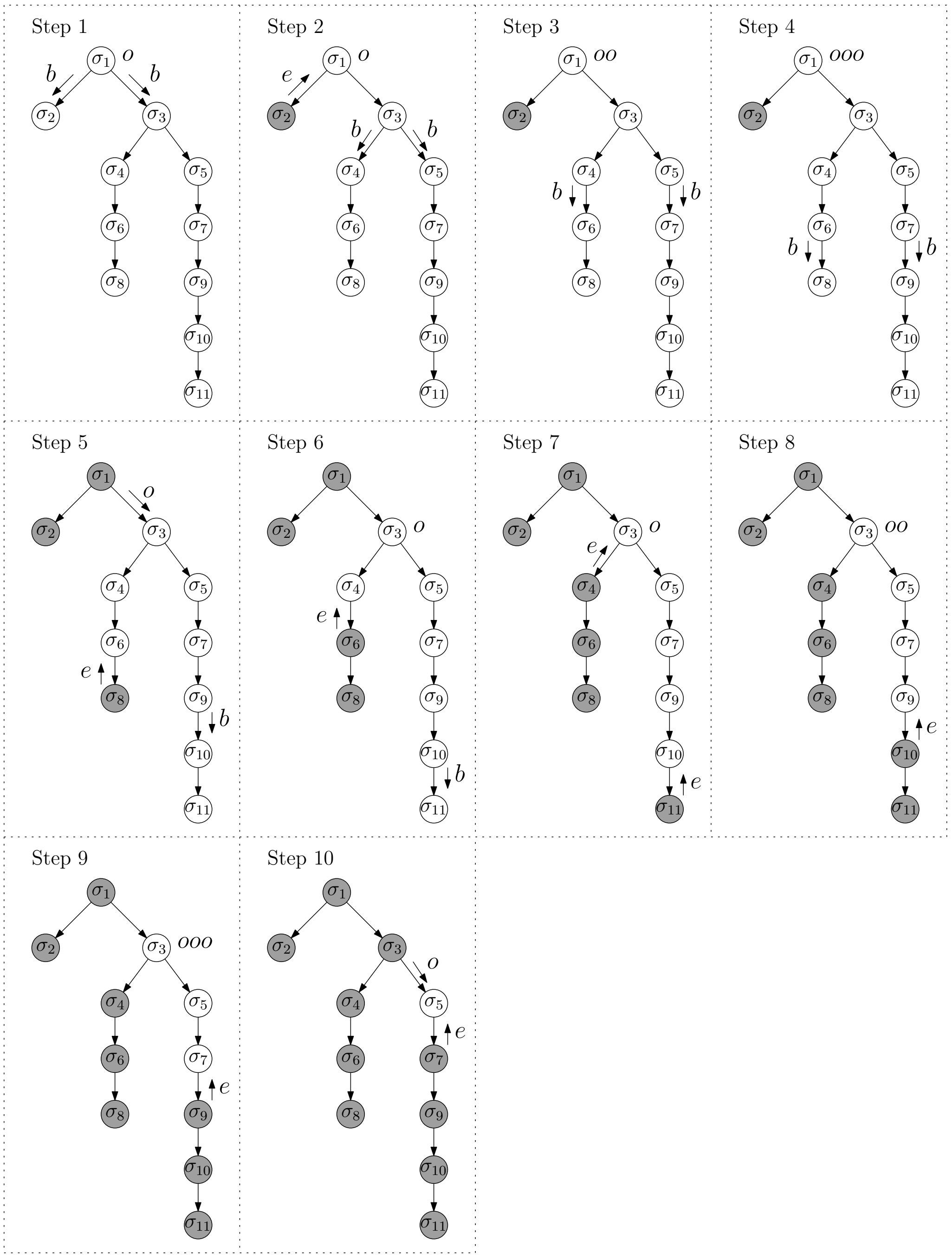


Figure 5.4: Propagations of symbols b, e and o, in a tree with one centre. The symbols e and o meet at the middle cell σ_5. Cells that have sent symbol e or o are shaded. The propagation of symbol o to a shaded cell is omitted. In cell σ_j, $j \in \{1, 3\}$, $|w_j|_o - 1$ represents the number of steps since σ_j received symbol e from all of its children but one.

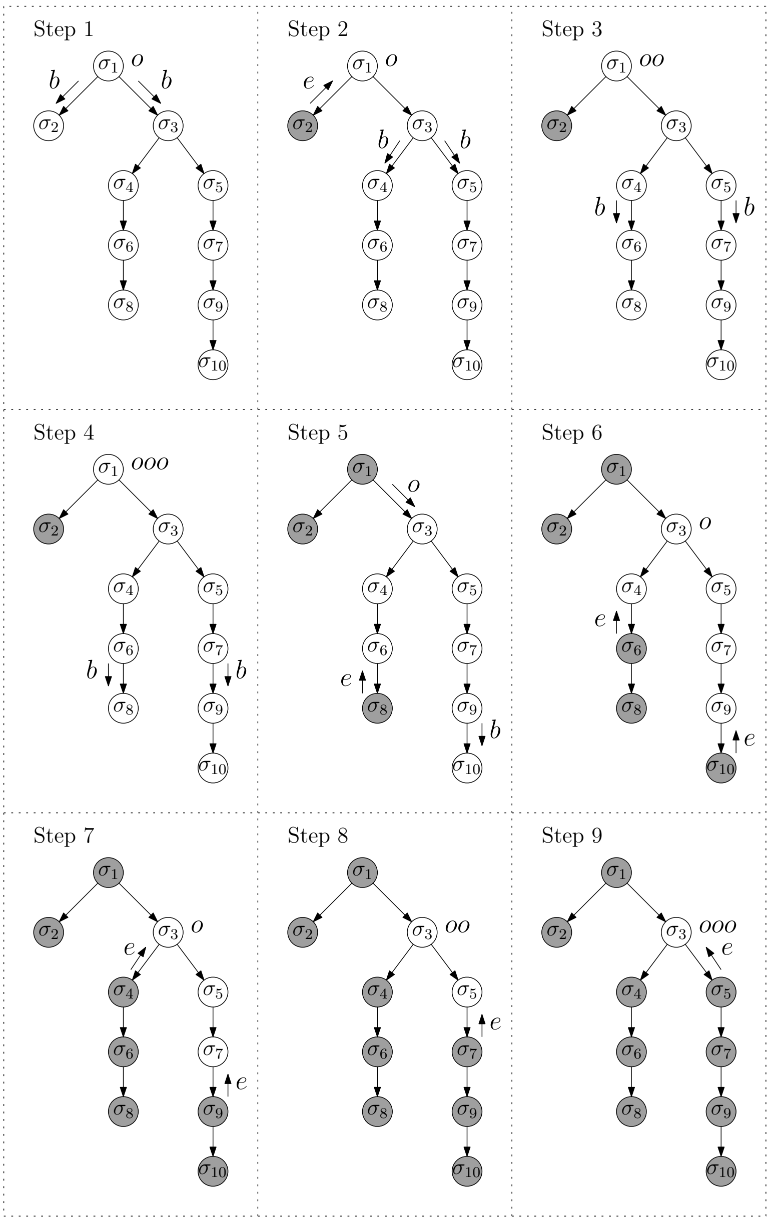


Figure 5.5: Propagations of symbols b, e and o, in a tree with two centres. The symbols e and o meet at the middle cell σ_3. Cells that have sent symbol e or o are shaded. The propagation of symbol o to a shaded cell is omitted. In cell σ_j, $j \in \{1, 3\}$, $|w_j|_o - 1$ represents the number of steps since σ_j received symbol e from all of its children but one.

a tree centre in its subtree. Symbols b, c and e represent the broadcast, acknowledgement and convergecast symbols, respectively, as described in Algorithm 4.2.2. The description of states s_0, s_1 and s_2 are as described in Algorithm 4.2.2. State s_3 is an idle state, where a cell remains until it receives symbol e or o from its parent. State s_6 is the end state.

Evolution rules of system Π

The evolution rules are given in Definition 5.13.

Rules description of system Π

Symbols b, e and o are propagated in the following manner.

- **Propagation of symbol b:** The root cell sends symbol b to all its children (rule 0.1). A non-root cell forwards the received symbol b to all its children (rule 0.2), if any. After applying rule 0.1 or 0.2, each cell produces a copy of symbol h in each step, until it receives symbol e from all its children (rules 1.1, 2.5 and 2.10).

- **Propagation of symbol e:** A leaf cell (Proposition 4.7) sends one copy of symbol e to its parent (rule 2.11). If a non-leaf cell receives symbol e from all its children, then it sends symbol e to its parent (rule 2.11), consumes all copies of symbol h and enters state s_6 (rule 3.2).

- **Propagation of symbol o:** The root cell initially contains the symbol o. Let σ_j denote the current cell that contains symbol o and has not entered state s_6.

  Assume, at step t, σ_j received symbol e from all but one subtree rooted at σ_v. Starting from step $t+1$, σ_j produces a copy of symbol o in each step, until it receives symbol e from σ_v (rule 2.8), That is, $|w_j|_o - 1$ indicates the number of steps since σ_j received symbol e from all of its children except σ_v.

  If σ_j receives symbol e from σ_v by step $t+2$, i.e. $|w_j|_o \leq 3$, then σ_j is the middle cell; σ_j keeps all copies of symbol h and enters state s_6 (rule 2.1). Otherwise, σ_j sends a copy of symbol o to σ_v at step $t+3$ (rule 2.6 or 2.7); in the subsequent steps, σ_j consumes all copies of symbol h and enters state s_6 (rules 2.2 and 3.2). Note, using current setup, σ_j cannot send a symbol to a specific child; σ_j has to send a copy of symbol o to all its children. However, all σ_j's children, except σ_v, would have entered state s_6.

Correctness of construction of system Π

Proposition 5.14 indicates the step in which σ_m receives symbol c from all its children and Proposition 5.15 indicates the number of steps needed to propagate symbol o from σ_s to σ_m.

Proposition 5.14. Cell σ_m receives the symbol c from all its children by step $\texttt{height}_s(s) + \texttt{height}_s(m) + 2$.

Proof. Follows from Proposition 4.10. □

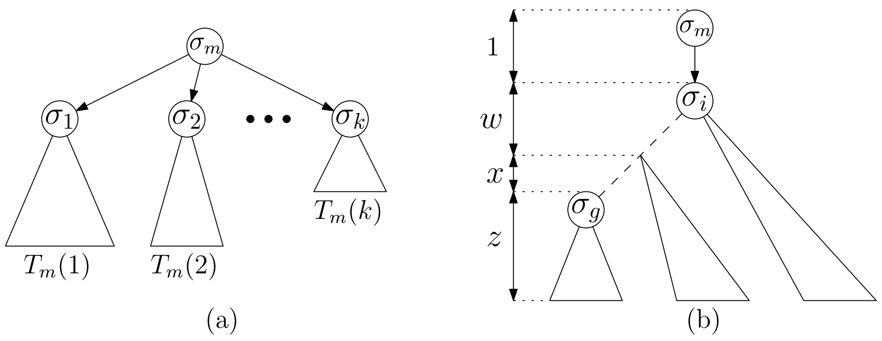


Figure 5.6: (a) k subtrees of σ_m, $T_m(1), T_m(2), \ldots, T_m(k)$. (b) The structure of subtree $T_m(j)$, which contains σ_s.

Proposition 5.15. The propagation of the symbol o from σ_s to σ_m takes at most $\texttt{height}_s(s) + \texttt{height}_s(m) + 2$ steps.

Proof. For a given tree T_s, rooted at σ_s, we construct a tree T_m, which re-roots T_s at σ_m. Recall, $T_m(i)$ denotes a subtree rooted at σ_i in T_m. Assume that σ_m has $k \geq 2$ subtrees, $T_m(1), T_m(2), \ldots, T_m(k)$, such that $\texttt{height}_m(1) \geq \texttt{height}_m(2) \geq \cdots \geq \texttt{height}_m(k)$ and $\texttt{height}_m(1) - \texttt{height}_m(2) \leq 1$. Figure 5.6 (a) illustrates the subtrees of σ_m.

Assume $T_m(i)$ is a subtree of σ_m, which contains σ_s. In $T_m(i)$, let z be the height of σ_s and $x + w \geq 0$ be the distance between σ_s and σ_i. Figure 5.6 (b) illustrates the z, x and w in $T_m(i)$.

To prove Proposition 5.15, we determine the number of steps needed to propagate symbol o from σ_s to σ_m. In $T_m(i)$, let p be a path from σ_i to its farthest leaf and t be the number of steps needed to propagate symbol o from σ_s to σ_m. Note, $\texttt{height}_m(m) = \texttt{height}_s(m)$ and $x + w + 1 = \texttt{height}_s(s) - \texttt{height}_s(m)$.

If σ_s is a part of path p, then $z + x + w + 1 = \texttt{height}_m(i) + 1 = \texttt{height}_m(m) - j$, $j \geq 0$, and $t = 2z + 3(x + w + 1)$. Hence,

$$\begin{aligned} t &= 2(z + x + w + 1) + (x + w + 1) \\ &= 2(\texttt{height}_m(m) - j) + (\texttt{height}_s(s) - \texttt{height}_s(m)) \\ &= \texttt{height}_s(s) + \texttt{height}_s(m) - 2j \end{aligned}$$

If σ_s is not a part of p, then $z+x+w+1 < v+w+1 = \texttt{height}_m(i)+1 = \texttt{height}_m(m)-j$, $j \geq 0$, and $t = x + 2v + 3(w + 1)$. Hence,

$$\begin{aligned} t &= 2(v + w + 1) + (x + w + 1) \\ &= 2(\texttt{height}_m(m) - j) + (\texttt{height}_s(s) - \texttt{height}_s(m)) \\ &= \texttt{height}_s(s) + \texttt{height}_s(m) - 2j \end{aligned}$$

Note, when a leaf receives a copy of broadcast symbol, it takes two additional steps before it sends convergecast symbol. Thus, $t = \texttt{height}_s(s)+\texttt{height}_s(m)-2j+2$. $\square$

Time and message complexities of system Π

The time and message complexities of system Π are indicated in Propositions 5.16 and 5.17, respectively.

Proposition 5.16. System Π halts in $\texttt{height}_s(s) + \texttt{height}_s(m) + 4$ steps.

Proof. From Propositions 5.14 and 5.15, symbols o and e meet in the middle cell σ_m at step $\texttt{height}_s(s)+\texttt{height}_s(m)+2$. Cell σ_m enters state s_6 by applying rules 2.9 and 2.1, which take two steps. Thus, finding cell σ_m takes $\texttt{height}_s(s) + \texttt{height}_s(m) + 4$

steps. □

Proposition 5.17. The total number of symbols exchanged in Π is at most $4 \cdot |\Delta|$.

Proof. In each tree arc $(\sigma_j, \sigma_k) \in \Delta$, (i) cell σ_j sends down one copy of symbol b to σ_k, (ii) cell σ_k sends up one copy of symbol c. Further, cell σ_k could up one copy of symbol e and cell σ_j could send down one copy of symbol o. Thus, over each tree arc, at least two symbols are sent and two additional symbols could be sent. Therefore, the total number of symbols exchanged between cells of Π is at most $4 \cdot |\Delta|$. □

5.4.2 Algorithm: Phase II—Propagation of the order

Phase II (Algorithm 5.2.1) of system Π starts at the end of Phase I, where the new general σ_m (i.e. the middle cell found in Phase I) issues the order paired with a hop-counter, initially set to its eccentricity $\texttt{ecc}(m)$, to inform all soldiers the correct step to enter the firing state. Refer Section 5.2.1 for full details of Phase II.

5.4.3 Summary of Phase I and Phase II

The time and message complexities of system Π are indicated in Propositions 5.18 and 5.19, respectively. Table 5.5 contains the traces of this adaptive FSSP solution (i.e. system Π of Definition 5.13), for the tree of Figure 5.1 (c), where σ_1 is the original general and σ_5 is the new general (i.e. a tree centre).

Theorem 5.18. System Π halts (i.e. synchronizes) in $\texttt{ecc}(s) + 2 \cdot \texttt{ecc}(m) + 5$ steps, where σ_s is the original general and σ_m is the middle cell.

Proof. Phase I takes $\texttt{ecc}(s) + \texttt{ecc}(m) + 4$ steps (Proposition 5.16). Phase II, which starts at the end of Phase I, takes $\texttt{ecc}(m) + 1$ steps (Proposition 5.6). Thus, in total, system Π takes $\texttt{ecc}(s) + 2 \cdot \texttt{ecc}(m) + 5$ steps. □

Theorem 5.19. The total number of symbols transferred between cells of Π is bounded by $4 \cdot |\Delta| + \sum_{\sigma_i \in K}((\texttt{ecc}(s) - \texttt{depth}_s(i))\texttt{paths}_s(i) \cdot |\texttt{Neighbour}(i)|)$.

Proof. The number of symbols transferred between cells in Phase I is bounded by $4 \cdot |\Delta|$ (Proposition 5.17). The number of symbols transferred between cell in Phase II is $\sum_{\sigma_i \in K}((\texttt{ecc}(s) - \texttt{depth}_s(i))\texttt{paths}_s(i) \cdot |\texttt{Neighbour}(i)|)$ (Proposition 5.7).

Therefore, the total number symbols transferred between cells is bounded by $4 \cdot |\Delta| + \sum_{\sigma_i \in K}((\texttt{ecc}(s) - \texttt{depth}_s(i))\texttt{paths}_s(i) \cdot |\texttt{Neighbour}(i)|)$. □

Table 5.5: The traces of the system Π of Definition 5.13, for the tree of Figure 5.4, where σ_1 is the original general and σ_5 is the new general (i.e. a tree centre). The shaded table cells indicate the steps in which Phase II begins (equivalently, Phase I ends). During the last step of Π's evolution, all cells simultaneously enter the firing state s_8 for the first time.

Step	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}
0	$s_0\ s$	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	$s_1\ ao$	$s_0\ b$	$s_0\ b$	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0
2	$s_2\ ac^2o$	$s_1\ a$	$s_1\ a$	$s_0\ b$	$s_0\ b$	s_0	s_0	s_0	s_0	s_0	s_0
3	$s_2\ ac^2ho$	$s_2\ a$	$s_2\ ac^2$	$s_1\ a$	$s_1\ a$	$s_0\ b$	$s_0\ b$	s_0	s_0	s_0	s_0
4	$s_2\ ac^2eh^2o$	$s_2\ at$	$s_2\ ac^2h$	$s_2\ ac$	$s_2\ ac$	$s_1\ a$	$s_1\ a$	$s_0\ b$	$s_0\ b$	s_0	s_0
5	$s_2\ ach^3o^2$	$s_3\ a$	$s_2\ ac^2h^2$	$s_2\ ach$	$s_2\ ach$	$s_2\ ac$	$s_2\ ac$	$s_1\ a$	$s_1\ a$	$s_0\ b$	s_0
6	$s_2\ ach^4o^3$	$s_3\ a$	$s_2\ ac^2h^3$	$s_2\ ach^2$	$s_2\ ach^2$	$s_2\ ach$	$s_2\ ach$	$s_2\ a$	$s_2\ ac$	$s_1\ a$	$s_0\ b$
7	$s_2\ aeh^4t$	$s_3\ ao$	$s_2\ ac^2h^4o$	$s_2\ ach^3$	$s_2\ ach^3$	$s_2\ aceh^2$	$s_2\ ach^2$	$s_2\ at$	$s_2\ ach$	$s_2\ ac$	$s_1\ a$
8	$s_3\ ae$	s_6	$s_2\ ac^2h^5o$	$s_2\ aceh^4$	$s_2\ ach^4$	$s_2\ ah^2t$	$s_2\ ach^3$	$s_3\ ae$	$s_2\ ach^2$	$s_2\ ach$	$s_2\ a$
9	s_6	s_6	$s_2\ ac^2eh^6o$	$s_2\ ah^4t$	$s_2\ ach^5$	$s_3\ ae$	$s_2\ ach^4$	s_6	$s_2\ ach^3$	$s_2\ aceh^2$	$s_2\ at$
10	s_6	s_6	$s_2\ ach^7o^2$	$s_3\ a$	$s_2\ ach^6$	s_6	$s_2\ ach^5$	s_6	$s_2\ aceh^4$	$s_2\ ah^2t$	$s_3\ ae$
11	s_6	s_6	$s_2\ ach^8o^3$	$s_3\ a$	$s_2\ ach^7$	s_6	$s_2\ aceh^6$	s_6	$s_2\ ah^4t$	$s_3\ ae$	s_6
12	s_6	s_6	$s_2\ aeh^8t$	$s_3\ ao$	$s_2\ aceh^8o$	s_6	$s_2\ ah^6t$	s_6	$s_3\ ae$	s_6	s_6
13	s_6	s_6	$s_3\ ae$	s_6	$s_2\ ah^8ot$	s_6	$s_3\ ae$	s_6	s_6	s_6	s_6
14	s_6	s_6	s_6	s_6	$s_6\ h^4s$	s_6	s_6	s_6	s_6	s_6	s_6
15	s_6	s_6	$s_6\ h^3s$	s_6	$s_7\ uv^3$	s_6	$s_6\ h^3s$	s_6	s_6	s_6	s_6
16	$s_6\ h^2s$	s_6	$s_7\ uv^2$	$s_6\ h^2s$	$s_7\ h^4s^2uv^2$	s_6	$s_7\ uv^2$	s_6	$s_6\ h^2s$	s_6	s_6
17	$s_7\ uv$	$s_6\ hs$	$s_7\ h^2s^2uv$	$s_7\ uv$	$s_7\ h^4s^2uv$	$s_6\ hs$	$s_7\ hsuv$	s_6	$s_7\ uv$	$s_6\ hs$	s_6
18	$s_7\ su$	$s_7\ u$	$s_7\ h^2s^2u$	$s_7\ su$	$s_7\ h^4s^2u$	$s_7\ u$	$s_7\ hsu$	$s_6\ s$	$s_7\ su$	$s_7\ u$	$s_6\ s$
19	s_8	s_8	s_8	s_8	s_8	s_8	s_8	s_8	s_8	s_8	s_8

5.4.4 Experimental work

This section compares the synchronization time of the adaptive FSSP solution (Algorithm 5.4) against the synchronization time of the static FSSP solution (Algorithm 5.3), for synchronizing tree-structured simple P systems.

Aim

To estimate the expected reduction in the number of steps needed to synchronize tree-structured P systems using the adaptive FSSP solution of Algorithm 5.4 ($h + 2r + 5$) over the FSSP solution of Algorithm 5.3 ($3h + 7$), where h and r are the tree height and tree radius, respectively. The reduction in synchronization time is denoted by `reduce` (i.e. $(3h + 7) - (h + 2r + 5)$), and the percentage of `reduce` with respect to the synchronization time of $3h + 7$ is denoted by `reduce`% (i.e. `reduce`$/(h + 2r + 5)$).

Hypothesis

Theoretically, for a given tree with the radius r, height h and diameter d, the `reduce` and `reduce`% are:

- In the worst-case scenario, where $h = r$, the synchronization time of the adaptive FSSP solution is $3h+5$, such that `reduce` $= (3h+7)-(3h+5) = 2$. In this case, `reduce`% $= 200/(3h + 7)$, such that `reduce`% converges to zero as h increases, i.e. there is no reduction in synchronization time.

- In the best-case scenario, where $h = d = 2r$, the synchronization time of the adaptive FSSP solution is $2h+5$, such that `reduce` $= (3h+7)-(2h+5) = h+2$. In this case, `reduce`% $= 100(h + 2)/(3h + 7)$, such that `reduce`% converges to value 33 as h increases, i.e. about 33% reduction in synchronization time.

Thus, for trees, the adaptive FSSP solution (Algorithm 5.4) can yield up to about 33% reduction in synchronization time over the static FSSP solution (Algorithm 5.3), i.e. $0 \leq$ `reduce`% ≤ 33.

Method

For uniformly random trees of order $n \in \{1000, 2000, \dots, 20000\}$, which are generated using the well-known Prüfer correspondence [82] (using the implementation given in Sage [76]), I compute `reduce` and `reduce`% as follows.

1. For each n, generate ten uniformly random labelled trees using the Prüfer correspondence. Each of the ten generated trees of order n is denoted by T_i^n, $1 \leq i \leq 10$.
2. For each tree T_i^n, its radius by $r(T_i^n)$ and consider its height, are denoted by $h(T_i^n)$, as the average eccentricity of all of its nodes.
3. For the ten trees of order n, i.e. T_i^n, $1 \leq i \leq 10$,

$$\texttt{reduce} = \frac{1}{10} \sum_{i \in \{1,2,\dots,10\}} (3h(T_i^n) + 7) - (h(T_i^n) + 2r(T_i^n) + 5)$$

$$\texttt{reduce}\% = \frac{100}{10} \sum_{i \in \{1,2,\dots,10\}} \frac{(3h(T_i^n) + 7) - (h(T_i^n) + 2r(T_i^n) + 5)}{3h(T_i^n) + 7}$$

Results

The values of `reduce` and `reduce`% for ten uniformly random trees of order n, $n \in \{1000, 2000, \dots, 20000\}$, are presented in the last two columns of Table 5.6. Moreover, Table 5.6 includes the auxiliary columns `radT` and `hT`, which correspond to the average radius and average height of the ten trees of order n, respectively.

Discussions

As shown in column `reduce`% of Table 5.6, the adaptive FSSP solution (Algorithm 5.4) yields on randomly generated trees, at least 20% reduction in the synchronization time over the static FSSP solution (Algorithm 5.3). Furthermore, the observed 20% reduction in synchronization time is closer to the best-case scenario (about 33% reduction in synchronization time) than the worst-case scenario (no reduction in synchronization time).

Table 5.6: The observed reduction in synchronization time for uniformly random trees.

n	radT	hT	reduce	reduce%	n	radT	hT	reduce	reduce%
1000	47.1	71.3	48.4	22.3	11000	158.3	236.3	155.9	21.9
2000	77.6	117.0	78.8	22.2	12000	177.6	268.7	182.2	22.5
3000	90.9	137.4	93.0	22.3	13000	191.3	283.9	185.2	21.6
4000	105.3	158.2	105.7	22.1	14000	193.9	292.4	196.9	22.3
5000	118.1	177.1	118.0	22.1	15000	190.6	284.0	186.9	21.8
6000	116.1	175.8	119.5	22.5	16000	208.6	311.3	205.5	21.9
7000	139.9	210.8	141.8	22.4	17000	228.8	342.3	227.0	22.1
8000	139.3	211.5	144.3	22.6	18000	210.0	314.1	208.2	22.0
9000	155.6	234.4	157.5	22.2	19000	224.0	333.6	219.1	21.9
10000	156.1	232.4	152.7	21.8	20000	236.3	356.5	240.4	22.4

5.5 Summary

This chapter presented two FSSP solutions, static FSSP solution (Section 5.3) and adaptive FSSP solution (Section 5.4). Both FSSP solutions consist of two independent phases, where:

1. **Phase I** computes the eccentricity of the general and determines the maximum number of steps needed to send the order to all soldiers.
2. **Phase II** propagates the order with a hop-counter (from the general to all other soldiers), which is used to indicate the precise step to synchronize, i.e. enter the firing state.

The features used in the design of the FSSP solutions are listed below.

- **Feature 1:** Locate the best general position to issue the firing order, i.e. find a centre of a given structure.
    - **Used:** Phase I of the adaptive FSSP solution.

- ◦ **Contribution:** Minimizes the the number of steps needed to deliver the firing order to all soldiers in Phase II.

- **Feature 2:** Compute the minimum number of steps it takes to deliver the firing order from the current general position.
    - ◦ **Used:** Phase I of the static and adaptive FSSP solutions.
    - ◦ **Method:** Half of the round-trip time from the current general to its farthest soldier.
    - ◦ **Contribution:** Minimizes the number of steps needed to deliver the firing order from the current general position in Phase II.

- **Feature 3:** Incorporate a timing-mechanism into the order.
    - ◦ **Used:** Phase II of the static and adaptive FSSP solutions.
    - ◦ **Method:** The firing order is a decrementing counter, initially set to the eccentricity of the current general, and the current counter decreases by one in every step.
    - ◦ **Contribution:** Enables each soldier to determine the remaining number of steps before all other soldiers receive the firing order.

The two FSSP solutions use the same approach in Phase II. However, in Phase I, different approaches are used to compute the eccentricity of the general. The summary of the FSSP solutions is as follows.

1. The static FSSP solution [27] (Section 5.3):
    - **Structure:** a weakly connected digraph.
    - **The general:** one general located at an arbitrary digraph node.
    - **Synchronization time:** $3e + 7$ steps, where e is the eccentricity of the general.
    - **Phase I:** the general computes its eccentricity (using Feature 2).
    - **Phase II:** the general sends the firing order with the decrementing counter, initially set to its eccentricity, in a BFS manner (using Feature 3).

2. The adaptive FSSP solution [26] (Section 5.4):

    - **Structure:** a rooted tree.
    - **The general:** one general located at the tree root.
    - **Synchronization time:** $h + 2r + 5$ steps, where h is the height of the tree and r is the radius of the tree.
    - **Phase I:** locates a tree centre and computes the tree radius (using Features 1 and 2).
    - **Phase II:** the centre (found in Phase I) sends the firing order with the decrementing counter, initially set to the tree radius, in a BFS manner (using Feature 3).

The static and adaptive FSSP solutions, presented in this chapter, are compared against the FSSP solutions of cellular automata (CA), for tree and digraph structures, with respect to time and program-size complexities. In CA, for trees, the FSSP solution described by Romani [71] synchronizes trees in $h + 2r$ steps, where h and r are the tree height and radius, respectively. This CA FSSP solution uses the same idea as the adaptive FSSP solution of Definition 5.13, discovered independently. This CA FSSP solution is not formally described and is not implemented. The adaptive FSSP solution of Definition 5.13 is presented with explicit evolution rules and extended to synchronize digraphs [26]. In CA, for digraphs, the FSSP presented by Nishitani and Honda [59] synchronizes digraphs in $3e + 1$ steps, where e is the eccentricity of the general. As indicated in Tables 5.7 and 5.8:

- Time complexity: the static and adaptive FSSP solutions require 7 and 5 additional steps than CA FSSP solutions, respectively. Note, CA models assume several features that are typically not present in standard membrane system models, such as bounded input and output connections—this is not assumed in the FSSP formulation for membrane systems. This could explain the better additive constants obtained by CA.

- Program-size complexity: the static and adaptive FSSP solutions require less instructions than CA FSSP solutions.

Hence, for synchronizing trees and digraphs, the FSSP solutions presented in this chapter are comparable with the FSSP solutions in CA, with respect to time and program-size complexities.

Table 5.7: Comparing the FSSP solution [59] of cellular automata (CA) against the static FSSP solution of Definition 5.8, where e denotes the eccentricity of the general.

Algorithm	Time complexity	Program-size complexity
CA FSSP solutions Nishitani and Honda [59]	$3e + 1$	135 transition rules
Static FSSP solution of Definition 5.8	$3e + 7$	37 evolution rules

Table 5.8: Comparing the FSSP solution [71] of cellular automata (CA) against the adaptive FSSP solution of Definition 5.13, where h and r denote the tree height and radius, respectively.

Algorithm	Time complexity	Program-size complexity
CA FSSP solutions Romani [71]	$h + 2r$	Not available
Adaptive FSSP solution of Definition 5.13	$h + 2r + 5$	24 evolution rules

Additionally, the FSSP solutions of this chapter contain a broadcast-based P algorithm, Algorithm 5.2.1 (Decrementing hop-counter), and an echo-based P algorithm, Algorithm 5.3.1 (Compute general's eccentricity). Tables 5.9 and 5.10 contain comparison results of these P algorithms against the synchronous broadcast pseudo-code (Definition 4.1) and echo pseudo-code (Definition 4.2).

Hence, with respect to the goals of this thesis, I have obtained in this chapter: (i) two comparable P algorithms that solve the FSSP, (ii) one comparable broadcast-based P algorithm and (iii) one comparable echo-based P algorithm.

Table 5.9: Comparing a synchronous distributed broadcast pseudo-code (Definition 4.1) against Algorithm 5.2.1 (Decrementing hop-counter), on digraphs, where e denotes the eccentricity of the source.

Algorithm	Time complexity	Program-size complexity
Synchronous distributed broadcast pseudo-code (Definition 4.1)	e	8 pseudo-code lines
Algorithm 5.2.1 (Decrementing hop-counter)	$e + 1$	7 evolution rules

Table 5.10: Comparing a synchronous distributed echo pseudo-code (Definition 4.2) against Algorithm 5.3.1 (Compute general's eccentricity), on digraphs, where e denotes the eccentricity of the source.

Algorithm	Time complexity	Program-size complexity
Synchronous distributed echo pseudo-code (Definition 4.2)	$2e$	14 pseudo-code lines
Algorithm 5.3.1 (Compute general's eccentricity)	$2e + 6$	30 evolution rules

Chapter 6

The Disjoint Paths Problem

This chapter presents native membrane system versions of two fundamental problems in graph theory:

1. *The edge-disjoint paths problem*: find the maximum number of paths from a source node to a target node, that have no edge in common.
2. *The node-disjoint paths problem*: find the maximum number of paths from a source node to a target node that have no other node in common, except the source and target nodes.

Starting from the standard Ford-Fulkerson's depth-first search maximum flow algorithms [48], totally distributed approach is used, where (i) no structural information is available initially and (ii) each membrane system cell has to learn its immediate neighbours. In the case of node-disjoint paths, the presented membrane system evolution rules are designed to enforce (i) node weight capacities of one and (ii) edge capacities of one.

This chapter is organized as follows. Section 6.1: (i) summarizes the standard network flow approaches for finding edge- and node-disjoint paths in digraphs, (ii) discusses alternative strategies that are more appropriate for membrane systems and (iii) describes three possible relations between the structural digraph of a membrane system and the search digraph used for determining paths. Section 6.2 presents: (i) breadth-first search based evolution rules for discovering the local cell topologies, i.e. neighbouring cells, (this is a common preliminary phase for both the edge- and node-disjoint paths implementations) and (ii) depth-first search based evolution rules for finding a maximum set of edge-disjoint paths. Section 6.3 presents depth-first search based evolution rules for finding a maximum set of node-disjoint paths. Finally, a summary of this chapter is given in Section 6.4.

6.1 Disjoint paths in digraphs

This section describes the basic algorithms for finding sets of edge- or node-disjoint paths, based on network flow, particularized for unweighted edges (i.e. all edge capacities are one), see Ford-Fulkerson [48]. The presentation will largely follow the standard approach (i.e. finding augmenting paths in residual graphs [37]), specifically targeted for running on highly distributed, parallel and synchronous computing models, such as membrane systems.

Consider a digraph $G = (V, E)$ and two nodes, a source node, $s \in V$, and a target node, $t \in V$. A set of *edge-disjoint paths*, from s to t, satisfies:

1. each of these paths starts from s and ends at t,
2. any two paths in the set have no edge in common.

A set of *node-disjoint paths*, from s to t, satisfies:

1. each of these paths starts from s and ends at t,
2. any two paths in the set have no other node in common, except s and t.

We consider the following two optimization problems: (i) find a maximum set of edge-disjoint paths from s to t; and (ii) find a maximum set of node-disjoint paths from s to t. Any set of node-disjoint paths is also a set of edge-disjoint paths, but the converse is not true. For example:

- Figure 6.1 (a) shows a maximum set of edge-disjoint paths of size two, which is also a maximum set of node-disjoint paths.

- Figure 6.1 (b) shows another maximum set of edge-disjoint paths of size two, which is not a set of node-disjoint paths.
- Figure 6.2 shows a digraph, where the maximum size of any edge-disjoint paths set is greater than the maximum size of any node-disjoint paths set.

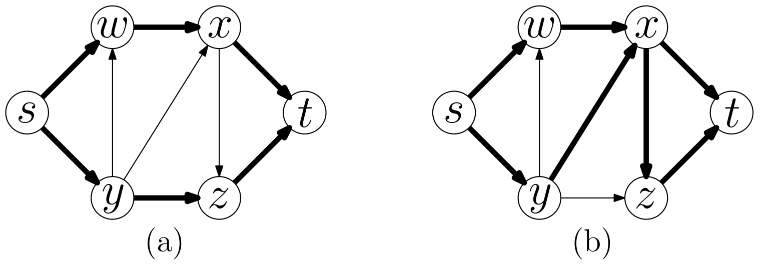


Figure 6.1: Let $p_1 = s.w.x.t$, $p_2 = s.y.z.t$, $p_3 = s.y.x.z.t$, $p_4 = s.w.x.z.t$ and $p_5 = s.y.x.t$. The set $\{p_1, p_2\}$ of (a) is a maximum set of edge-disjoint paths and the maximum set of node-disjoint paths. The sets $\{p_1, p_3\}$ and $\{p_4, p_5\}$ of (b) are maximum sets of edge-disjoint paths, but neither of these sets is a maximum set of node-disjoint paths.

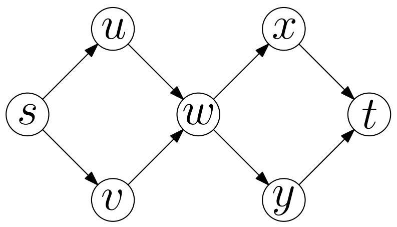


Figure 6.2: This digraph admits two maximum sets of edge-disjoint paths of size two: $\{s.u.w.x.t, s.v.w.y.t\}$ and $\{s.u.w.y.t, s.v.w.x.t\}$. This digraph admits four maximum sets of node-disjoint paths of size one: $\{s.u.w.x.t\}$, $\{s.u.w.y.t\}$, $\{s.v.w.x.t\}$ and $\{s.v.w.y.t\}$.

6.1.1 Preliminaries

For a given digraph $G = (V, E)$, consider two nodes $s \in V$ and $t \in V$, and a set of edge-disjoint paths from s to t, denoted by P. Nodes in P are called *flow-nodes* and arcs in P are called *flow-arcs*.

Given a path $\pi \in P$, each flow-arc $(u, v) \in \pi$ has a natural *incoming* and *outgoing* direction–the flow is from the source to the target; with respect to π, u is the *flow-predecessor* of v and v is the *flow-successor* of u.

The *residual digraph* is the digraph $R = (V, E')$, where the arcs in P are reversed, i.e. $E' = (E \setminus \{(u, v) \mid (u, v) \in P\}) \cup \{(v, u) \mid (u, v) \in P\}$. Any path from s to t in R is called an *augmenting path*.

Given augmenting path α, each flow-arc $(u, v) \in \alpha$ has also a natural *incoming* and *outgoing* direction–the flow is from the source to the target; with respect to α, u is the *search-predecessor* of v and v is the *search-successor* of u.

6.1.2 Edge-disjoint paths in digraphs

In both edge- and node-disjoint cases, the basic algorithms repeatedly search paths (called augmenting paths) in an auxiliary structure (called residual network or residual digraph). First, an edge-disjoint paths set is considered, since a node-disjoint paths set can be considered as an edge-disjoint paths problem, with additional constraints.

For the following "network flow" definition for digraphs with non-weighted arcs, an arc (u, v) is in a set of paths P, denoted by the slightly abused notation $(u, v) \in P$, if there exists a path $\pi \in P$ that uses arc (u, v).

Fact 6.1. Augmenting paths can be used to construct a larger set of edge-disjoint paths. More precisely, consider a digraph G and two nodes s and t. A set of edge-disjoint paths P_k of size k from s to t and an augmenting path α from s to t can be used together to construct a set of edge-disjoint paths P_{k+1} of size $k + 1$. First, paths in $\{\alpha\} \cup P_k$ are fragmented, by removing "conflicting" arcs, i.e. arcs that appear in $Q \cup \tilde{Q}$, where $Q = P \cap \tilde{\alpha}$ (where $\tilde{}$ indicates arc reversal). Then, new paths are created by concatenating resulting fragments. For the formal definition of this construction, refer to Ford and Fulkerson [48]. Note that including a reversed arc in an augmenting path is known as *flow pushback operation*.

This construction, described in Fact 6.1, is illustrated in Figure 6.3. Figure 6.3 (a) illustrates a digraph G and a set of edge-disjoint paths P_1 from s to t, currently the singleton $\{\pi_0\}$, where $\pi_0 = s.y.x.t$. Figure 6.3 (b) shows its associated residual digraph R (note the arcs reversal). Figure 6.3 (c) shows an augmenting path α in R, $\alpha = s.w.x.y.z.t$. Figure 6.3 (d) shows the extended set P_2 (after removing arcs (x, y) and (y, x)), consisting of a set of edge-disjoint paths of size two from s to t, $\{\pi_1 = s.w.x.t, \pi_2 = s.y.z.t\}$. Figure 6.4 shows a similar scenario, where another augmenting path is found. Note that the two paths illustrated in Figure 6.3 (d) form both a maximum edge-disjoint set and a maximum node-disjoint set; however, the two paths sets shown in Figure 6.4 (d) form two other maximum edge-disjoint paths sets, but none of them is node-disjoint.

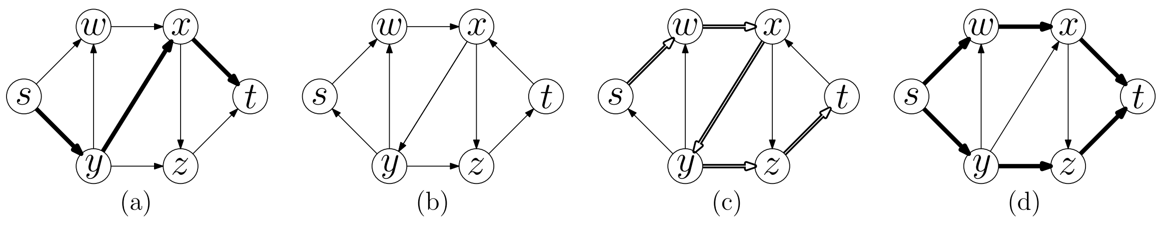


Figure 6.3: (a) A digraph G and one (edge-disjoint) path π_0 from s to t (indicated by bold arrows). (b) The residual digraph R_0 associated to digraph G and path π_0. (c) An augmenting path α in R_0 (indicated by hollow arrows). (d) Two new edge-disjoint paths π_1 and π_2, reconstructed from π_0 and α (both indicated by bold arrows).

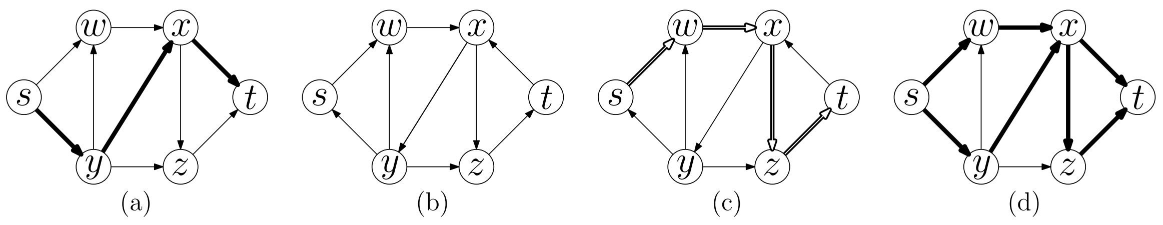


Figure 6.4: The residual digraph of Figure 6.3 with another augmenting path and two new paths sets, $\{s.w.x.t, s.y.x.z.t\}$, $\{s.w.x.z.t, s.y.x.t\}$, which are edge-disjoint but not node-disjoint.

The pseudo-code of Algorithm 6.2 effectively finds the maximum number (and a representative set) of edge-disjoint paths from s to t. A potential speed-up approach for Algorithm 6.2 is given in [22].

Definition 6.2. (Basic edge-disjoint paths algorithm)

Input: A digraph $G = (V, E)$ and two nodes $s \in V$, $t \in V$.
Output: P_k and k, where P_k is a maximum set of edge-disjoint paths and k is the size of P_k.

```
1.  $k = 0$ (the stage counter)
2.  $P_0 = \emptyset$ (the current set of edge-disjoint paths)
3.  $R_0 = G$ (the current residual digraph)
4.  loop
5.     $\alpha =$ an augmenting path in $R_k$, from $s$ to $t$, if any (this is a search operation)
6.     if $\alpha =$ null then break
7.     $k = k + 1$ (next stage)
8.     $P_k =$ the larger paths set constructed using $P_{k-1}$ and $\alpha$ (Fact 6.1)
9.     $R_k =$ the residual digraph of $G$ and $P_k$
10. endloop
```

Typically, the internal implementation of search at step 6 alternates between a *forward* mode in the residual digraph, which tries to extend a partial augmenting path, and a backwards *backtrack* mode in the residual digraph, which retreats from an unsuccessful attempt, looking for other ways to move forward. The internal implementation of step 9 (i.e. Fact 6.1) walks backwards in the residual digraph, as a *consolidation* phase, which recombines the newly found augmenting path with the existing edge-disjoint paths.

This algorithm runs in $k + 1$ stages, if we count the number of times it looks for an augmenting path, and terminates when a new augmenting path is not found. The actual procedure used (in step 6) to find the augmenting path separates two families of algorithms: (i) algorithms from the Ford-Fulkerson family [48] use a depth-first search (DFS) and (ii) algorithms from the Edmonds-Karp family [28] use a breadth-first search (BFS). As usual, both DFS and BFS use “bread crumb” symbols, as markers, to avoid cycles; at the end of each stage, these markers are cleaned, to start again with a fresh context. A DFS based membrane system algorithms from the Ford-Fulkerson family is developed in this chapter.

6.1.3 Node-disjoint paths in digraphs

The edge-disjoint version can be also used to find a maximum set of node-disjoint paths. The textbook solution for the node-disjoint paths problem is usually achieved by a simple procedure, called the *node-splitting technique* [77], which transforms the original digraph in such a way that, on the transformed digraph, the edge-disjoint paths problem is identical to the node-disjoint paths problem of the original digraph. Essentially, this procedure globally replaces every node v, other than the source s and the target t, with two nodes, an *entry* node v_1 and an *exit* node v_2, connected by a single arc (v_1, v_2). That is, the new digraph $G' = (V', E')$ has $V' = \{s, t\} \cup \{v_1, v_2 \mid v \in V \setminus \{s, t\}\}$, $E' = \{(v_1, v_2) \mid v \in V \setminus \{s, t\}\} \cup \{(u_2, v_1) \mid (u, v) \in E\}$, where, for convenience, assume that $s_1 = s_2 = s$ and $t_1 = t_2 = t$ are aliases. Figure 6.5 illustrates this standard node-splitting technique. It is straightforward to see that the newly introduced arcs (w_1, w_2), (x_1, x_2), (y_1, y_2) and (z_1, z_2), constrain any edge-disjoint paths set to be also a node-disjoint paths set.

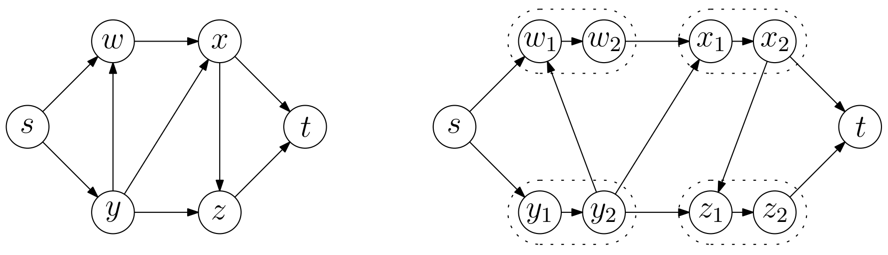


Figure 6.5: The node-splitting technique [77].

However, in this case, each node is identified with a membrane systems cell, hence, the node-disjoint paths problem cannot be solved using the standard node-splitting technique. Two **non-standard search rules** are proposed, which together limit the out-flow capacity of each $v \in V \setminus \{s, t\}$ to one, by simulating the node-splitting technique, without actually splitting the nodes. These rules could be used in other distributed network models, where the standard node-splitting technique is not applicable. These rules are illustrated by the scenario presented in Figure 6.6, where we assume that we have already determined a first flow-path, $s.x.y.z.t$, and we are now trying to build a new augmenting path.

1. Consider the case when the augmenting path, consisting of s, tries flow-node y via the non-flow arc (s, y). We cannot continue with the existing non-flow arc

(y, t) (as the edge-disjoint version would do), because this will exceed node y's capacity, which is one already. Therefore, we continue the search with just the *reversed* flow-arc (y, x). Note that, in the underlying node-splitting scenario, we are only visiting the entry node y_1, but not its exit pair y_2.

2. Consider now the case when the augmenting path, extended now to $s.y.x.z$, tries again the flow-node y, via the reversed flow-arc (z, y). It may appear that we are breaking the traditional search rules, by re-visiting the already visited node y. However, there is no infringement in the underlying node-splitting scenario, where we are now trying the not-yet-visited exit node y_2 (to extend the underlying augmenting path $s.y_1.x_2.z_1$). From y, we continue with any available *non-flow* arc, if any, otherwise, we backtrack. In this example, we continue with arc (y, t). We obtain a new augmenting "path", $s.y.x.z.y.t$ (corresponding to the underlying augmenting path $s.y_1.x_2.z_1.y_2.t$). We further recombine it with the already existing flow-path $s.x.y.z.t$, and we finally obtain a set of node-disjoint paths of size two, i.e. $\{s.x.z.t, s.y.t\}$.

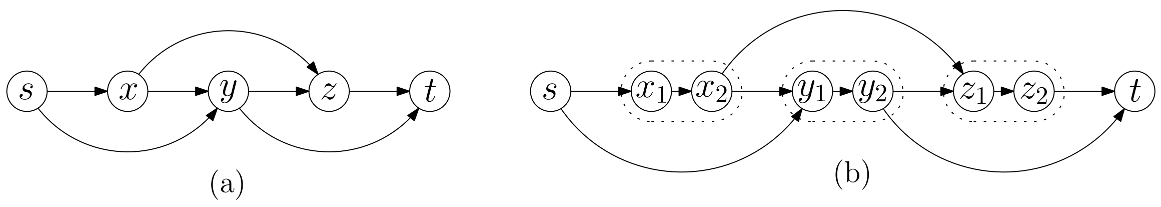


Figure 6.6: Node-disjoint paths. (a) Non-standard search: flow path $s.x.y.z.t$ and augmenting "path" $s.y.x.z.y.t$. (b) Node-splitting: flow path $s.x_1.x_2.y_1.y_2.z_1.z_2.t$ and augmenting path $s.y_1.x_2.z_1.y_2.t$.

Theorem 6.3. If the augmented path search in step 6 of Algorithm 6.2 is modified as indicated above, the algorithm will terminate with a set of edge-disjoint paths, forming a maximum set of node-disjoint paths.

6.1.4 Pointer management

With respect to the implementation, the edge-disjoint version provides its own additional challenge, not present in the node-disjoint version. In the node-disjoint version, each flow-node needs only one pointer to its flow-predecessor and another to its flow-successor. However, in the edge-disjoint version, a flow-node can have $k \geq 1$ flow-predecessors and k flow-successors, where each combination is possible, giving rise to $k!$ different paths sets, each of size k, passing through this node. A naive approach would require recording full details of all $k!$ possible size-k paths sets, or, at least, full details for one of them.

In this simplified approach, full path details are not kept; instead, a node needs only two size-k lists: (i) its flow-predecessors list and (ii) its flow-successors list. Using this information, any of the actual $k!$ paths sets can be formed, by properly matching flow-predecessors with flow-successors. As an example, consider node x of Figure 6.4 (d), which has two flow-predecessors, w and y, and two flow-successors, t and z; thus w is part of four distinct paths. Node w needs only two size-k lists: its flow-predecessors list, $\{w, y\}$, and its flow-successors list $\{z, t\}$.

6.1.5 Structural and search digraphs in membrane systems

Various ways are considered to reformulate the digraph edge- and node-disjoint paths problems as a native membrane system problem. The considered membrane system is "physically" based on a digraph, but this digraph is not necessarily the *virtual* search digraph $G = (V, E)$, on which edge- and node-disjoint paths need to be found. Given a simple P system $\Pi = (O, K, \Delta)$, where Δ is its structural digraph, first identify cells as nodes of interest, $V \simeq K$. However, after that, there are three fundamentally distinct scenarios, which differ in the way how the *forward* and *backward* modes (i.e. *backtrack* and *consolidation*) of Algorithm 6.2 map to the residual arcs and finally to the structural arcs.

1. Set $E \simeq \Delta$. In this case, the forward mode follows the direction of parent-child arcs of Δ, while the backward modes follow the reverse direction, from child to parent.

2. Set $E \simeq \{(v, u) \mid (u, v) \in \Delta\}$. In this case, the the backward modes follow the direction of parent-child arcs of Δ, while the forward mode of the search follows the reverse direction, from child to parent.

3. Set $E \simeq \{(u, v), (v, u) \mid (u, v) \in \Delta\}$. In this case, the resulting search digraph

is symmetric, and each of the arcs followed by the forward or backward modes of the search can be either a parent-child arc in the original Δ or its reverse.

Cases (1) and (2) are simpler to develop. However, solutions to case (3) are considered, where all messages must be sent to all neighbours, i.e. parents and children together. Therefore, the presented evolution rules use the target indicator $\tau = \updownarrow$. Figure 6.7 illustrates a simple P system and these three scenarios.

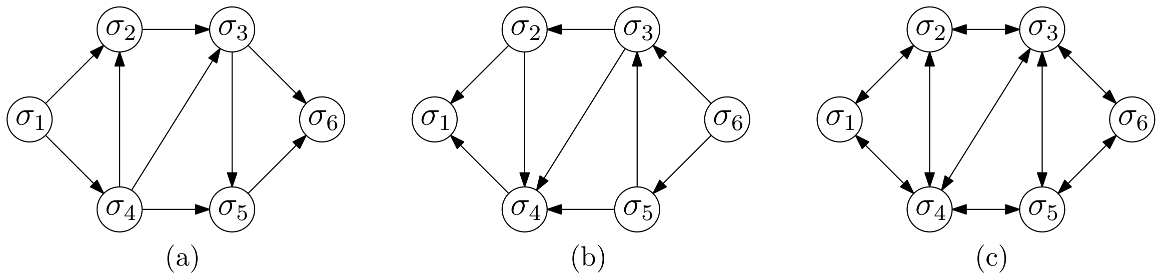


Figure 6.7: Three virtual search digraphs for the same simple P system. (a) Same "physical" and search structure. (b) The search structure reverses the "physical" structure. (c) The search structure covers both the "physical" structure and its reverse.

Note that, in any of the three cases, Algorithm 6.2 needs to be able to follow both the parent-child and the child-parent directions of membrane system structure. Therefore, the structural arcs must support bidirectional communication channels.

After fixing the directions used by the virtual graph G, the next problem is to let the nodes discover their neighbours, i.e. discover the local network topology.

6.2 Edge-disjoint paths solution

This section provides a simple P system specification of the edge-disjoint paths algorithm presented in Section 6.1. A maximum set of edge-disjoint paths from the source cell to the target cell is computed. In Problem 6.4, the considered edge-disjoint paths problem in terms of expected input and output is explicitly stated. Definition 6.5 provides the formal description of a simple P system that solves Problem 6.4.

Problem 6.4. (Edge-disjoint paths problem)

Input: A simple P system $\Pi = (O, K, \Delta)$, where the source cell $\sigma_s \in K$ contains a token t_t identifying the ID of the target cell $\sigma_t \in K$.
Output: If $\sigma_s \neq \sigma_t$, then each cell $\sigma_i \in K$ contains a set of predecessor pointer symbols $P_i = \{p_j \mid (j, i) \text{ is a flow-arc}\}$ and a set of successor point symbols $C_i = \{c_j \mid (i, j) \text{ is a flow-arc}\}$ that represent a maximum set of edge-disjoint paths from σ_s to σ_t, where the following constraints hold:

1. **flow-arcs:** $c_i \notin C_i$, $p_i \notin P_i$, $c_j \in C_i \Leftrightarrow p_i \in P_j$ and
   $c_j \in C_i \Rightarrow \sigma_j \in \Delta(i) \cup \Delta^{-1}(i)$.
2. **source and target:** $P_s = \emptyset$ and $C_t = \emptyset$.
3. **in flow = out flow:** If $i \notin \{s, t\}$ then $|C_i| = |P_i|$.
4. **only paths:** With $S(I) = \bigcup_{i \in I} \{c_j \mid c_j \in C_i\}$,
   $S^{n-1}(I) = S(S(\cdots S(I) \cdots)) = \emptyset$.

Definition 6.5. (Simple P system for finding a maximum set of edge-disjoint paths) A simple P system (of order n) that solves the edge-disjoint paths problem, described in Problem 6.4, is $\Pi = (O, K, \Delta)$, where:

1. $O = \{o, u, v, w, z\} \cup \{a_j, \bar{a}_j, c_j, d_j, e_j, g_j, h_j, l_j, m_j, n_j, p_j, q_j, r_j \mid j \in \{1, 2, \ldots, n\}\}$
   $\cup\ \{b_{jk}, f_{jk}, x_{jk}, y_{jk} \mid j, k \in \{1, 2, \ldots, n\}\}$.
2. $K = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, where $\sigma_s \in K$ is the source cell and $\sigma_t \in K$ is the target cell. Each cell $\sigma_i \in K$ has an initial form $\sigma_i = (Q, s_{i0}, w_{i0}, R)$, where:
   - $Q = \{s_0, s_1, \ldots, s_{11}\}$, is a set of state.
   - $s_{i0} = s_0$, is the initial quiescent state.
   - $w_{i0} = \begin{cases} a_i l_t & \text{if } \sigma_i = \sigma_s, \\ a_i & \text{if } \sigma_i \neq \sigma_s, \end{cases}$ is the initial content.
   - R is a set of evolution rules, which is given below.
3. Δ forms a connected graph.

Evolution rules used in Phase I:

0. **Rules for state s_0:**

   1 $s_0\ a_i l_i \rightarrow_{\texttt{min}} s_0$

   2 $s_0\ a_i l_j \rightarrow_{\texttt{min}} s_1\ a_i o\ (n_i, \updownarrow)\ (g_j, \updownarrow)$

   3 $s_0\ a_i g_i \rightarrow_{\texttt{min}} s_1\ a_i z\ (n_i, \updownarrow)\ (g_i, \updownarrow)$

   4 $s_0\ a_i g_j \rightarrow_{\texttt{min}} s_1\ a_i\ (n_i, \updownarrow)\ (g_j, \updownarrow)$

1. **Rules for state s_1:**

   1 $s_1\ a_i \rightarrow_{\texttt{min}} s_2\ a_i$

2. **Rules for state s_2:**

   1 $s_2\ o \rightarrow_{\texttt{min}} s_3\ o$

   2 $s_2\ z \rightarrow_{\texttt{min}} s_4\ z$

   3 $s_2\ a_i \rightarrow_{\texttt{min}} s_5\ a_i$

   4 $s_2\ g_j \rightarrow_{\texttt{max}} s_3$

   5 $s_2\ g_j \rightarrow_{\texttt{max}} s_4$

   6 $s_2\ g_j \rightarrow_{\texttt{max}} s_5$

Evolution rules used in Phase II:

3. **Rules for state s_3:**

   1 $s_3\ a_i n_j \rightarrow_{\texttt{min}} s_3\ \bar{a}_i d_j\ (f_{ij}, \updownarrow)$

   2 $s_3\ \bar{a}_i d_j y_{ij} n_k \rightarrow_{\texttt{min}} s_{10}\ a_i c_j n_k ww\ (v, \updownarrow)$

   3 $s_3\ \bar{a}_i d_j y_{ij} o \rightarrow_{\texttt{min}} s_{11}\ a_i c_j ww\ (u, \updownarrow)$

   4 $s_3\ \bar{a}_i d_j x_{ij} n_k \rightarrow_{\texttt{min}} s_3\ a_i m_j n_k$

   5 $s_3\ \bar{a}_i d_j x_{ij} o \rightarrow_{\texttt{min}} s_{11}\ a_i m_j ww\ (u, \updownarrow)$

   6 $s_3\ \bar{a}_i b_{ji} \rightarrow_{\texttt{min}} s_3\ \bar{a}_i\ (x_{ji}, \updownarrow)$

   7 $s_3\ \bar{a}_i f_{ji} \rightarrow_{\texttt{min}} s_3\ \bar{a}_i\ (x_{ji}, \updownarrow)$

   8 $s_3\ f_{jk} \rightarrow_{\texttt{min}} s_3$

   9 $s_3\ b_{jk} \rightarrow_{\texttt{min}} s_3$

   10 $s_3\ y_{jk} \rightarrow_{\texttt{min}} s_3$

   11 $s_3\ x_{jk} \rightarrow_{\texttt{min}} s_3$

4. **Rules for state s_4:**

1 $s_4\ a_i n_j f_{ji} \rightarrow_{\texttt{min}} s_4\ a_i p_j\ (y_{ji}, \updownarrow)$

2 $s_4\ v \rightarrow_{\texttt{min}} s_{10}\ ww\ (v, \updownarrow)$

3 $s_4\ uz \rightarrow_{\texttt{min}} s_{11}\ ww\ (u, \updownarrow)$

4 $s_4\ f_{jk} \rightarrow_{\texttt{min}} s_4$

5 $s_4\ b_{jk} \rightarrow_{\texttt{min}} s_4$

6 $s_4\ y_{jk} \rightarrow_{\texttt{min}} s_4$

7 $s_4\ x_{jk} \rightarrow_{\texttt{min}} s_4$

5. **Rules for state s_5:**

1 $s_5\ v \rightarrow_{\texttt{min}} s_{10}\ ww\ (v, \updownarrow)$

2 $s_5\ u \rightarrow_{\texttt{min}} s_{11}\ ww\ (u, \updownarrow)$

3 $s_5\ a_i n_j f_{ji} \rightarrow_{\texttt{min}} s_6\ a_i q_j$

4 $s_5\ a_i c_j b_{ji} \rightarrow_{\texttt{min}} s_6\ a_i e_j$

5 $s_5\ a_i h_j \rightarrow_{\texttt{min}} s_9\ a_i c_j\ (x_{ji}, \updownarrow)$

6 $s_5\ p_j q_k \rightarrow_{\texttt{min}} s_8\ p_j q_k$

7 $s_5\ a_i q_j \rightarrow_{\texttt{min}} s_5\ a_i m_j\ (x_{ji}, \updownarrow)$

8 $s_5\ a_i f_{ji} \rightarrow_{\texttt{min}} s_5\ a_i\ (x_{ji}, \updownarrow)$

9 $s_5\ f_{jk} \rightarrow_{\texttt{min}} s_5$

10 $s_5\ b_{jk} \rightarrow_{\texttt{min}} s_5$

11 $s_5\ y_{jk} \rightarrow_{\texttt{min}} s_5$

12 $s_5\ x_{jk} \rightarrow_{\texttt{min}} s_5$

6. **Rules for state s_6:**

1 $s_6\ a_i n_j \rightarrow_{\texttt{min}} s_7\ a_i d_j\ (f_{ij}, \updownarrow)$

2 $s_6\ a_i \rightarrow_{\texttt{min}} s_8\ a_i$

7. **Rules for state s_7:**

1 $s_7\ a_i d_j y_{ij} e_k \rightarrow_{\texttt{min}} s_5\ a_i c_j m_k\ (y_{ki}, \updownarrow)$

2 $s_7\ a_i d_j y_{ij} q_k \rightarrow_{\texttt{min}} s_5\ a_i c_j p_k\ (y_{ki}, \updownarrow)$

3 $s_7\ a_i d_j x_{ij} \rightarrow_{\texttt{min}} s_6\ a_i m_j$

4 $s_7\ a_i c_j b_{ji} \rightarrow_{\texttt{min}} s_7\ a_i c_j\ (x_{ji}, \updownarrow)$

5 $s_7\ a_i n_j f_{ji} \rightarrow_{\texttt{min}} s_7\ a_i m_j\ (x_{ji}, \updownarrow)$

6 $s_7\ f_{jk} \rightarrow_{\texttt{min}} s_7$

7 $s_7\ b_{jk} \rightarrow_{\texttt{min}} s_7$

8 $s_7\ y_{jk} \rightarrow_{\texttt{min}} s_7$

9 $s_7\ x_{jk} \rightarrow_{\texttt{min}} s_7$

8. **Rules for state s_8:**

1 $s_8\ a_i p_j \rightarrow_{\texttt{min}} s_9\ a_i r_j\ (b_{ij}, \updownarrow)$

2 $s_8\ a_i e_j \rightarrow_{\texttt{min}} s_5\ a_i c_j\ (x_{ji}, \updownarrow)$

3 $s_8\ a_i q_j \rightarrow_{\texttt{min}} s_5\ a_i m_j\ (x_{ji}, \updownarrow)$

9. **Rules for state s_9:**
   1 $s_9\ a_i r_j y_{ij} e_k \rightarrow_{\texttt{min}} s_5\ a_i m_j m_k\ (y_{ki}, \updownarrow)$
   2 $s_9\ a_i r_j y_{ij} q_k \rightarrow_{\texttt{min}} s_5\ a_i m_j p_k\ (y_{ki}, \updownarrow)$
   3 $s_9\ a_i r_j x_{ij} \rightarrow_{\texttt{min}} s_8\ a_i t_j$
   4 $s_9\ a_i c_j b_{ji} \rightarrow_{\texttt{min}} s_5\ a_i h_j$
   5 $s_9\ a_i n_j f_{ji} \rightarrow_{\texttt{min}} s_9\ a_i m_j\ (x_{ji}, \updownarrow)$
   6 $s_9\ f_{jk} \rightarrow_{\texttt{min}} s_9$
   7 $s_9\ b_{jk} \rightarrow_{\texttt{min}} s_9$
   8 $s_9\ y_{jk} \rightarrow_{\texttt{min}} s_9$
   9 $s_9\ x_{jk} \rightarrow_{\texttt{min}} s_9$

10. **Rules for state s_{10}:**
   1 $s_{10}\ w \rightarrow_{\texttt{min}} s_{10}$
   2 $s_{10}\ v \rightarrow_{\texttt{max}} s_{10}$
   3 $s_{10}\ m_j \rightarrow_{\texttt{min}} s_{10}\ n_j$
   4 $s_{10}\ t_j \rightarrow_{\texttt{min}} s_{10}\ p_j$
   5 $s_{10}\ o \rightarrow_{\texttt{min}} s_3\ o$
   6 $s_{10}\ z \rightarrow_{\texttt{min}} s_4\ z$
   7 $s_{10}\ a_i \rightarrow_{\texttt{min}} s_5\ a_i$

11. **Rules for state s_{11}:**
   1 $s_{11}\ w \rightarrow_{\texttt{min}} s_{11}$
   2 $s_{11}\ u \rightarrow_{\texttt{max}} s_{11}$
   3 $s_{11}\ a_i \rightarrow_{\texttt{min}} s_0\ a_i$
   4 $s_{11}\ t_j \rightarrow_{\texttt{min}} s_0\ p_j$
   5 $s_{11}\ n_j \rightarrow_{\texttt{min}} s_0$
   6 $s_{11}\ m_j \rightarrow_{\texttt{min}} s_0$

6.2.1 Phase I: Algorithm—Neighbourhood discovery

Phase I is a preliminary phase that discovers the neighbours of every cell, prior to Phase II (the edge-disjoint paths set discovery phase). In Phase I, each cell $\sigma_i \in K$ *notifies* its own cell ID, i, by sending one copy of its neighbour pointer symbol n_i to each of its neighbours.

At the end of Phase I, each cell $\sigma_i \in K$ *obtains* one copy of neighbour pointer symbol n_j from each $\sigma_j \in \texttt{Neighbour}(i)$. For each cell $\sigma_i \in K$, the set of neighbour pointer symbols is denoted by $N_i = \{n_j \mid \sigma_j \in \texttt{Neighbour}(i)\}$. Table 6.1 illustrates the expected algorithm output of Phase I (i.e. a set N_i, for each $\sigma_i \in K$), for system Π of Figure 6.7 (c).

Table 6.1: A set of neighbour pointer symbols for each cell of system Π of Figure 6.7 (c). Symbol n_j in cell σ_i indicates that cell σ_j is a neighbour of σ_i.

Cell	Neighbours	Neighbour pointer symbols, N_i
σ_1	$\{\sigma_2, \sigma_4\}$	$\{n_2, n_4\}$
σ_2	$\{\sigma_1, \sigma_3, \sigma_4\}$	$\{n_1, n_3, n_4\}$
σ_3	$\{\sigma_2, \sigma_4, \sigma_5, \sigma_6\}$	$\{n_2, n_4, n_5, n_6\}$
σ_4	$\{\sigma_1, \sigma_2, \sigma_3, \sigma_5\}$	$\{n_1, n_2, n_3, n_5\}$
σ_5	$\{\sigma_3, \sigma_4, \sigma_6\}$	$\{n_3, n_4, n_6\}$
σ_6	$\{\sigma_3, \sigma_5\}$	$\{n_3, n_5\}$

Precondition of Phase I of system Π

Each cell $\sigma_i \in K$ starts with the initial configuration described in Definition 6.5.

Postcondition of Phase I of system Π

At the end of Phase I, the configuration of cell $\sigma_i \in K$ is (s_i, w_i), where:

- $s_i = s_3$, if $\sigma_i = \sigma_s$, where s_3 is the designated state for the source cell.
- $s_i = s_4$, if $\sigma_i = \sigma_t$, where s_4 is the designated state for the target cell.
- $s_i = s_5$, if $i \notin \{s, t\}$, where s_5 is the designated state for an intermediate cell.
- $|w_i|_{a_i} = 1$, where a_i is cell ID symbol that σ_i uses to operate with its own cell ID, i.
- For each $\sigma_j \in \texttt{Neighbour}(i)$, $|w_i|_{n_j} = 1$, where n_j is a neighbour pointer symbol.
- $|w_i|_o = 1$, if $\sigma_i = \sigma_s$, where symbol o is the marker for the source cell.
- $|w_i|_z = 1$, if $\sigma_i = \sigma_t$, where symbol z is the marker for the target cell.

States and symbols of Phase I of system Π

Symbol a_i is cell ID symbol that cell σ_i uses to operate with its own cell ID, i. Symbol o is the marker for the source cell σ_s. Symbol z is the marker for the target cell σ_z. Symbol n_i is a neighbour pointer symbol sent by cell σ_i; symbol n_i in cell σ_j indicates that σ_i is one of σ_j's neighbours. Symbols l_t and g_t are used to identify the target cell σ_t, where the subscript t indicates the cell ID of the target cell σ_t.

State s_0 is the state, where each cell expects to receive one copy of symbol g_t and one copy of symbol n_j from $\sigma_j \in \texttt{Pred}_s(i)$. State s_1 is the state, where each cell expects to receive one copy of symbol g_t and one copy of symbol n_j from $\sigma_j \in \texttt{Peer}_s(i)$. State s_2 is the state, where each cell expects to receive one copy of symbol g_t and one copy of symbol n_j from $\sigma_j \in \texttt{Succ}_s(i)$.

Overview of Phase I of system Π

Initially, the source cell σ_s contains one copy of symbol l_t, where the subscript t is the cell ID of the target cell. If the subscript t matches the source cell's cell ID, s, then algorithm terminates (rule 0.1). Otherwise, σ_s broadcasts the σ_t's cell ID, t, to all cells.

Each cell $\sigma_i \in K$ compares its own cell ID, i (obtained from the initial symbol a_i), against σ_t's cell ID, t (obtained from the received symbol g_t). If $i = t$, then cell σ_i is the target cell.

The σ_t's cell ID, t, is propagated from σ_s to all other cells in the following manner.

- The source cell σ_s:
    - ◦ sends one copy of symbol g_t and one copy of symbol n_s to each of its neighbours,
    - ◦ marks itself as the source cell by producing one copy of symbol o (rule 0.2).
- When an intermediate cell σ_i receives symbol g_t for the first time, σ_i:
    - ◦ sends one copy of symbol g_t and one copy of symbol n_i to each of its neighbours (rule 0.3).

- When the target cell σ_t receives symbol g_t for the first time, σ_t:
  - sends one copy of symbol g_t and one copy of symbol n_t to each of its neighbours,
  - marks itself as the target cell by producing one copy of symbol z (rule 0.4).

Cell $\sigma_i \in K$ receives:

- one copy of symbol g_t and one copy of symbol n_j from each $\sigma_j \in \texttt{Pred}_s(i)$ at step $\texttt{depth}_s(i)$ (Proposition 6.6).
- one copy of symbol g_t and one copy of symbol n_k from each $\sigma_k \in \texttt{Peer}_s(i)$ at step $\texttt{depth}_s(i) + 1$ (Proposition 6.7).
- one copy of symbol g_t and one copy of symbol n_h from each $\sigma_h \in \texttt{Succ}_s(i)$ at step $\texttt{depth}_s(i) + 2$ (Proposition 6.8).

Proposition 6.6. At step $\texttt{depth}_s(i)$, cell σ_i receives one copy of symbol g_t and one copy of symbol n_j from each $\sigma_j \in \texttt{Pred}_s(i)$. At step $\texttt{depth}_s(i) + 1$, cell σ_i sends one copy of symbol g_t and one copy of symbol n_i to each of its neighbours.

Proposition 6.7. At step $\texttt{depth}_s(i)+1$, cell σ_i reaches configuration (s_1, w_i), where

- $|w_i|_{a_i} = 1$,
- for each $\sigma_j \in \texttt{Pred}_s(i) \cup \texttt{Peer}_s(i)$, $|w_i|_{n_j} = 1$,
- $|w_i|_{g_t} = |\texttt{Pred}_s(i)| + |\texttt{Peer}_s(i)| - 1$,
- $|w_i|_o = 1$, if $\sigma_i = \sigma_s$,
- $|w_i|_z = 1$, if $\sigma_i = \sigma_z$.

Proof. At step $\texttt{depth}_s(i)+1$, cell σ_i makes transition $s_0 \Rightarrow s_1$, where σ_i: (i) sends one copy of symbol g_t and one copy of symbol n_i to each of its neighbours, (ii) receives one copy of symbol g_t and one copy of symbol n_k from each $\sigma_k \in \texttt{Peer}_s(i)$, (iii) produces one copy of symbol o, if $\sigma_i = \sigma_s$, (iv) produces one copy of symbol z, if $\sigma_i = \sigma_t$, (v) enters state s_1. □

Proposition 6.8. At step $\texttt{depth}_s(i) + 2$, cell σ_i makes transition $s_1 \Rightarrow s_2$, where σ_i: (i) receives one copy of symbol g_t and one copy of symbol n_h from each $\sigma_h \in \texttt{Succ}_s(i)$, (ii) enters state s_2.

Proposition 6.9. At step $\texttt{depth}_s(i){+}3$, cell σ_i makes transition $s_2 \Rightarrow s_3$, where σ_i: (i) discards all copies of symbol g_t that are accumulated at steps $\texttt{depth}_s(i)$, $\texttt{depth}_s(i)+1$ and $\texttt{depth}_s(i) + 2$, (ii) enters state s_3.

Table 6.2 contains the traces of Phase I of system Π, for the graph of Figure 6.7.

Table 6.2: The traces of Phase I (Algorithm 6.2.1) of system Π, of Figure 6.7 (c), where σ_1 is the source cell and σ_5 is the target cell. As indicated at step 5, cell σ_3 ends with neighbour pointer symbols, n_1, n_2, n_4 and n_5, which correspond to σ_3's neighbours, σ_1, σ_2, σ_4 and σ_5, respectively.

Step	σ_1	σ_2	σ_3	σ_4	σ_5
0	$s_0\ a_1$ l_5	$s_0\ a_2$	$s_0\ a_3$	$s_0\ a_4$	$s_0\ a_5$
1	$s_1\ a_1 o$	$s_0\ a_2$ $g_5 n_1$	$s_0\ a_3$ $g_5 n_1$	$s_0\ a_4$	$s_0\ a_5$
2	$s_2\ a_1 o$ $g_5^2 n_2 n_3$	$s_1\ a_2$ $g_5 n_1 n_3$	$s_1\ a_3$ $g_5 n_1 n_2$	$s_0\ a_4$ $g_5^2 n_2 n_3$	$s_0\ a_5$ $g_5 n_3$
3	$s_3\ a_1 o$ $n_2 n_3$	$s_2\ a_2$ $g_5^2 n_1 n_3 n_4$	$s_2\ a_3$ $g_5^3 n_1 n_2 n_4 n_5$	$s_1\ a_4$ $g_5^2 n_2 n_3 n_5$	$s_1\ a_5 z$ $g_5 n_3 n_4$
4	$s_3\ a_1 o$ $n_2 n_3$	$s_5\ a_2$ $n_1 n_3 n_4$	$s_5\ a_3$ $n_1 n_2 n_4 n_5$	$s_2\ a_4$ $g_5^2 n_2 n_3 n_5$	$s_2\ a_5 z$ $g_5 n_3 n_4$
5	$s_3\ a_1 o$ $n_2 n_3$	$s_5\ a_2$ $n_1 n_3 n_4$	$\boldsymbol{s_5\ a_3}$ $\boldsymbol{n_1 n_2 n_4 n_5}$	$s_5\ a_4$ $n_2 n_3 n_5$	$s_4\ a_5 z$ $n_3 n_4$

Correctness and complexity of Phase I of system Π

Proposition 6.10 indicates the correctness and time complexity of Phase I of system Π. Proposition 6.11 indicates the message complexity of Phase I of system Π.

Proposition 6.10. Phase I of system Π halts at step $\texttt{ecc}(s)+3$ and the configuration of each cell corresponds to the postcondition.

Proof. Thus, at step $\texttt{depth}_s(i) + 3$, cell σ_i ends in state s_3 with

- one copy of symbol a_i,
- a set of neighbour pointer symbols, $N_i = \{n_j \mid \sigma_j \in \texttt{Neighbour}(i)\}$,
- one copy of symbol o, if $\sigma_i = \sigma_s$,
- one copy of symbol z, if $\sigma_i = \sigma_t$.

There are no rules in state s_3, hence, cell σ_i cannot evolve once it enters state s_3. For a farthest cell σ_f (with respect to the source cell σ_s), $\texttt{depth}_s(f) \geq \texttt{depth}_s(g)$, for all $\sigma_g \in K$, such that $\texttt{depth}_s(f) = \texttt{ecc}(s)$. Cell σ_f enters state s_3 at step $\texttt{depth}_s(f) + 3 = \texttt{ecc}(s) + 3$.

Therefore, system Π in Phase I halts at step $\texttt{ecc}(s) + 3$, and the final configuration of each cell in Phase I corresponds to the postcondition. □

Proposition 6.11. The total number of symbols that are transferred between cell of Π in Phase I is $4 \cdot |\Delta|$.

Proof. Each cell σ_i sends one copy of symbol n_i and one copy of symbol g_t to each of its neighbours. For each arc $(\sigma_j, \sigma_k) \in \Delta$: (i) one copy of symbol n_j, (ii) one copy of symbol n_k and (iii) two copies of symbol g_t are transferred. Therefore, the total number of transferred symbols is $4 \cdot |\Delta|$. □

6.2.2 Phase II: Algorithm—Edge-disjoint paths discovery

Phase II is the edge-disjoint paths discovery phase, that finds a maximum set of edge-disjoint paths from the source cell to the target cell. For each cell $\sigma_i \in K$: (i) a set of flow-predecessor pointer symbols is denoted by $P_i = \{p_j \mid \sigma_j \text{ is a flow-predecessor of } \sigma_i\}$ and (ii) a set of flow-successor pointer symbols is denoted by $C_i = \{c_j \mid \sigma_j \text{ is a flow-successor of } \sigma_i\}$.

Table 6.3 illustrates the expected algorithm output of Phase II, i.e. a set of flow-predecessor and flow-successor symbols of every cell, for the graph of Figure 6.7 (c). For each cell $\sigma_i \in K$, all the neighbour pointer symbols from Phase I are listed, $N_i = \{n_j \mid \sigma_j \in \texttt{Neighbour}(i)\}$, which are discarded at the end of Phase II.

Table 6.3: A representation of a maximum set of edge-disjoint paths, for simple P system of Figure 6.7 (c). Cell σ_2 has one flow-predecessor σ_1 and one flow-successor σ_3.

Cell	Neighbour pointer symbols, N_i	Flow-predecessor pointer symbols, P_i	Flow-successor pointer symbols, C_i
σ_1	$\{n_2, n_3\}$	$\emptyset$	$\{c_2, c_4\}$
σ_2	$\{n_1, n_3, n_4\}$	$\{p_1\}$	$\{c_3\}$
σ_3	$\{n_2, n_4, n_5, n_6\}$	$\{p_2, p_4\}$	$\{c_5, c_6\}$
σ_4	$\{n_1, n_2, n_3, n_5\}$	$\{p_1\}$	$\{c_3\}$
σ_5	$\{n_3, n_4, n_6\}$	$\{p_3\}$	$\{c_6\}$
σ_6	$\{n_3, n_5\}$	$\{p_3, p_5\}$	$\emptyset$

Precondition of Phase II of system Π

Each cell $\sigma_i \in K$ starts with the configuration described in the postcondition of Algorithm 6.2.1.

Postcondition of Phase II of system Π

At the end of Phase II, the configuration of cell $\sigma_i \in K$ is (s_0, w_i), where:

- $|w_i|_{a_i} = 1$, where a_i is cell ID symbol that σ_i uses to determine its own cell ID, i.
- For each $p_j \in P_i$, $|w_i|_{p_j} = 1$, where cell σ_j is a flow-predecessor of σ_i.
- For each $c_j \in C_i$, $|w_i|_{c_j} = 1$, where cell σ_j is a flow-successor of σ_i.

States and symbols of Phase II of system Π

Phase II uses the symbols used in Phase I, plus the following additional symbols: $\{u, v, w\} \cup \{c_j, d_j, e_j, h_j, m_j, p_j, q_j, r_j, t_j \mid j \in \{1, 2, \dots, n\}\} \cup \{b_{ij}, f_{ij}, x_{ij}, y_{ij} \mid i, j \in \{1, 2, \dots, n\}\}$. In each cell σ_i, $i \in \{1, 2, \dots, n\}$, these symbols have the following meanings:

- Symbol $\bar{a}_i$ is specifically used by the source cell σ_s, to indicate that σ_i is waiting for a response from its search successor.
- Symbol c_j indicates that σ_j is a flow-successor of σ_i.
- Symbol p_j indicates that σ_j is a flow-predecessor of σ_i.
- Symbol d_j indicates that σ_j is a search-successor of σ_i.
- Symbol q_j indicates that σ_j is a search-predecessor of σ_i.
- Symbol f_{ij} indicates a *search-extension request* sent from cell σ_i to its neighbour σ_j.
- Symbol b_{ij} indicates a *flow-pushback request* sent from cell σ_i to its flow-predecessor σ_j.
- Symbol x_{ij} indicates cell σ_j's *rejection response* to σ_i's search-extension or flow-pushback request.
- Symbol y_{ij} indicates cell σ_j's *acceptance response* to σ_i's search-extension or flow-pushback request.
- Symbol r_j indicates that cell σ_i sent a flow-pushback request to its flow-predecessor σ_j.
- Symbol t_j indicates that cell σ_i received a rejection response from its flow-predecessor σ_j, regarding σ_i's flow-pushback request sent to σ_j.
- Symbol e_j indicates that not-yet-visited cell σ_i has received a flow-pushback request from its flow-successor σ_j.

- Symbol h_j indicates that already-visited cell σ_i has received a flow-pushback request from its flow-successor σ_j.
- Symbol m_j indicates that cell σ_j is: (i) one of cell σ_i's former search-predecessor, where σ_i was not able to find a flow successor or (ii) one of cell σ_i's former flow-successor, where σ_i was able to find an another flow successor.
- Symbol v indicates a *reset request* that sets the status of all visited neighbours to unvisited.
- Symbol u indicates that all edge-disjoint paths are found, which prompts each cell σ_i to discard all symbols except its cell ID symbol a_i, $p_j \in P_i$ and $c_k \in C_i$.
- Each copy of symbol w prompts σ_i to remain idle in its current state for one step.

The following states are used in Phase II.

- State s_3 is the designated state for the source cell σ_s, where σ_s: (i) finds search-successors and flow-successors and (ii) rejects push-back requests and flow-extension requests.
- State s_4 is the designated state for the target cell σ_t, where σ_t waits for search-extension requests.
- States s_5, s_6, s_7, s_8, s_9 are the designated states for an intermediate cell σ_i.
    - In state s_5, cell σ_i receives: (i) a search-extension request from its search-predecessor or (ii) a flow-pushback request from a flow-successor.
    - In state s_6, cell σ_i searches for a search-successor.
    - In state s_7, cell σ_i waits for a response from its search successor.
    - In state s_8, cell σ_i sends a flow-pushback request to one of its flow-predecessors.
    - In state s_9, cell σ_i waits for a response from its flow-predecessor, regarding the flow-pushback request sent in state s_8.
- States s_{10} and s_{11} are the states for all cells.
    - In state s_{10}, each cell resets the status of all visited neighbours to unvisited.

- In state s_{11}, each cell discards all symbols, except its cell ID symbol, flow-predecessor pointer symbols and flow-successor pointer symbols.

This P algorithm is a direct implementation of Ford-Fulkerson's network flow algorithm [48]. Thus, this edge-disjoint paths P algorithm finds a maximum set of edge-disjoint paths from the source cell to the target cell in polynomial number of steps, as indicated in Theorem 6.12.

Theorem 6.12. For a simple P system with n cells and $m = |\Delta|$ edges, this P algorithm finds a maximum set of edge-disjoint paths from the source cell to the target cell in $O(mn)$ steps.

6.3 Node-disjoint paths solution

A simple P system specification of the node-disjoint paths algorithm, presented in Section 6.1, is provided. The problem is explicitly stated in terms of expected input and output. A maximum set of node-disjoint paths from a source cell to a target cells is computed.

Problem 6.13. (Node-disjoint paths problem)
Input: A simple P system $\Pi = (O, K, \Delta)$, where the source cell $\sigma_s \in K$ contains a token t_t identifying the ID of the target cell $\sigma_t \in K$.
Output: If $s \neq t$, each cell $\sigma_i \in K$ contains a set of predecessor pointer symbols $P_i = \{p_j \mid (j,i) \text{ is a flow-arc}\}$ and a set of successor pointer symbols $C_i = \{c_j \mid (i,j) \text{ is a flow-arc}\}$ that represent a maximum set of node-disjoint paths from σ_s to σ_t, where the following constraints hold:

1. **flow-arcs:** $c_i \notin C_i$, $p_i \notin P_i$, $c_j \in C_i \Leftrightarrow p_i \in P_j$ and $c_j \in C_i \Rightarrow j \in \Delta(i) \cup \Delta^{-1}(i)$.
2. **source and target:** $P_s = \emptyset$ and $C_t = \emptyset$.
3. **node-disjoint:** If $i \notin \{s,t\}$ then $|C_i| = |P_i| \leq 1$.
4. **only paths:** With $S(i) = \left\{ \begin{array}{ll} t & \text{if } i \in \{s,t\} \text{ or } |C_i| = 0 \\ j & \text{when } C_i = \{c_j\} \end{array} \right\}$, $S^{n-1}(i) = S(S(\cdots S(i) \cdots)) = t$.

Due to the network flow properties, $|C_s| = |P_t|$, which also represents a maximum set of node-disjoint paths. Notice the constraints to require only paths has been simplified in that the successor $S(i)$ of non-source cell σ_i is a single cell instead of a set of cells that was needed for the general edge-disjoint problem.

Table 6.4 illustrates the expected algorithm output, for a simple P system with the cell structure corresponding to Figure 6.7 (a). For convenience, although these are deleted near the algorithm's end, all neighbour pointer symbols are listed, $N_i = \{n_j \mid j \in \Delta(i) \cup \Delta^{-1}(i)\}$, for $i \in \{1, 2, \ldots, n\}$, which are determined in Phase I.

Table 6.4: A representation of a maximum set of node-disjoint paths, for simple P system of Figure 6.7 (a).

Cell	Neighbour pointer symbols, N_i	Flow-predecessor pointer symbols, P_i	Flow-successor pointer symbols, C_i
σ_1	$\{n_2, n_3\}$	$\emptyset$	$\{c_2, c_4\}$
σ_2	$\{n_1, n_3, n_4\}$	$\{p_1\}$	$\{c_3\}$
σ_3	$\{n_2, n_4, n_5, n_6\}$	$\{p_2\}$	$\{c_6\}$
σ_4	$\{n_1, n_2, n_3, n_5\}$	$\{p_1\}$	$\{c_5\}$
σ_5	$\{n_3, n_4, n_6\}$	$\{p_4\}$	$\{c_6\}$
σ_6	$\{n_3, n_5\}$	$\{p_3, p_5\}$	$\emptyset$

The rules of this node-disjoint paths P algorithm are exactly the rules of the edge-disjoint paths P algorithm described in Definition 6.5, where the rules for state s_5 are replaced by the following group of rules. The rules of state s_5 implement the proposed non-standard technique described in Section 6.1.3, for enforcing node capacities to one, without node-splitting.

This P algorithm is a direct implementation of Ford-Fulkerson's network flow algorithm [48]. Thus, this node-disjoint paths P algorithm finds a maximum set of node-disjoint paths from the source cell to the target cell in polynomial number of steps, as indicated in Theorem 6.14.

Theorem 6.14. For a simple P system with n cells and $m = |\Delta|$ edges, this P algorithm finds a maximum set of node-disjoint paths from the source cell to the target cell in $O(mn)$ steps.

5. Rules for a cell σ_i in state s_5:

   1 $s_5\ v \rightarrow_{\texttt{min}} s_{10}\ ww\ (v, \updownarrow)$

   2 $s_5\ u \rightarrow_{\texttt{min}} s_{11}\ ww\ (u, \updownarrow)$

   3 $s_5\ a_i n_j f_{ji} p_k \rightarrow_{\texttt{min}} s_8\ a_i q_j p_k$

   4 $s_5\ a_i n_j f_{ji} \rightarrow_{\texttt{min}} s_6\ a_i q_j$

   5 $s_5\ a_i c_j b_{ji} \rightarrow_{\texttt{min}} s_6\ a_i e_j$

   6 $s_5\ h_j \rightarrow_{\texttt{min}} s_6\ e_j$

   7 $s_5\ r_j \rightarrow_{\texttt{min}} s_9\ r_j$

   8 $s_5\ a_i q_j \rightarrow_{\texttt{min}} s_5\ a_i m_j\ (x_{ji}, \updownarrow)$

   9 $s_5\ a_i f_{ji} \rightarrow_{\texttt{min}} s_5\ a_i\ (x_{ji}, \updownarrow)$

   10 $s_5\ f_{jk} \rightarrow_{\texttt{min}} s_5$

   11 $s_5\ b_{jk} \rightarrow_{\texttt{min}} s_5$

   12 $s_5\ y_{jk} \rightarrow_{\texttt{min}} s_5$

   13 $s_5\ x_{jk} \rightarrow_{\texttt{min}} s_5$

6.4 Summary

Using the simple P system framework, this chapter presented native membrane system versions of the edge- and node-disjoint paths problems, which are based on standard network flow ideas, with additional constraints, such as cells that start without any knowledge about the local and global structure.

The P algorithms presented here use a depth-first search technique and iteratively build routing tables, until they find a maximum set of edge- or node-disjoint paths. For finding a maximum set of node-disjoint paths, an alternate set of search rules is proposed, which can be used for other synchronous network models, where the standard node-splitting technique is not applicable. In the *Byzantine Agreement problem* [24, 23], in the case of non-complete graphs, the standard solution allows for k faulty nodes (within a set of nodes of order at least $3k + 1$), if and only if there are at least $2k + 1$ node-disjoint paths between each pair of nodes, to ensure that a distributed consensus can occur [50]. Hence, these P algorithms can be used to determine the number of node-disjoint paths in the Byzantine Agreement problem.

These P algorithms run in polynomial time, comparable to the standard versions of the Ford-Fulkerson algorithms [48]. The P algorithms contain $O(n^3)$ number of evolution rules, while the disjoint paths pseudo-code of Definition 6.2 contains 10 pseudo-code lines. Note, the pseudo-code of Definition 6.2 does not explain how to:

1. Perform a search operation of line 5,
2. Find augmenting paths of line 8 and
3. Find residual graphs of line 9.

Properly explaining lines $5, 8, 9$, even at this pseudo-code level, will substantially increase the number of pseudo-code lines. Note, fixed size rule sets can be obtained using the newly proposed generic membrane system framework [56].

With respect to the goals of this thesis, this chapter presented two distributed P algorithms, that solve the edge- and node-disjoint paths problems. These P algorithms can form a component of a library of fundamental distributed algorithms in membrane systems.

Chapter 7

Conclusions

This thesis provided a coherent approach to specify and analyse a certain class of problems related to distributed algorithms, and provided evidence that membrane systems are adequate for modelling fundamental distributed algorithms. This thesis has taken a first step towards creating a library of fundamental distributed algorithms in membrane systems, which could be useful for building more complex distributed algorithms.

This thesis presented a set of P algorithms (i.e. algorithms that are implemented using membrane systems) that are *comparable* (i.e. not necessarily better, but not much worse) to:

- the best-known algorithms, with respect to *time complexity*,
- their corresponding pseudo-code, with respect to *program-size complexity* (i.e. the number of instructions or evolution rules).

Consider three levels of algorithm description [74]: (i) high-level description, (ii) implementation description and (ii) formal description. A pseudo-code is a high-level description, while a P algorithm is a formal description. Chapters 4, 5 and 6 presented a set of distributed P algorithms and showed that, even at a detailed executable level, these P algorithms compare favourably against high-level pseudo-codes on the considered criteria.

Chapter 4 presented a set of broadcast-based P algorithms (i.e. broadcast algorithms that perform additional computation) and echo-based P algorithms (i.e. echo algorithms that perform additional computation). As indicated in Tables 7.1, 7.2, 7.3 and 7.4, these P algorithms are comparable to the corresponding distributed synchronous broadcast (Definition 4.1) and echo (Definition 4.2) algorithms, with respect to time and program-size complexities.

Table 7.1: Comparing a synchronous distributed broadcast pseudo-code against Algorithm 4.2.1 (Broadcast with acknowledgement), on trees, where h denotes the tree height.

Algorithm	Time complexity	Program-size complexity
Synchronous distributed broadcast pseudo-code (Definition 4.1)	h	8 pseudo-code lines
Algorithm 4.2.1 (Broadcast with acknowledgement)	$h + 2$	2 evolution rules

Table 7.2: Comparing a synchronous distributed broadcast pseudo-code against Algorithms 4.3.1 (Number of shortest paths), 4.3.2 (Distance parity) and 5.2.1 (Decrementing hop-counter), on digraphs, where e denotes the eccentricity of the source.

Algorithm	Time complexity	Program-size complexity
Synchronous distributed broadcast pseudo-code (Definition 4.1)	e	8 pseudo-code lines
Algorithm 4.3.1 (Number of shortest paths)	$e + 3$	6 evolution rules
Algorithm 4.3.2 (Distance parity)	$e + 3$	13 evolution rules
Algorithm 5.2.1 (Decrementing hop-counter)	$e + 1$	7 evolution rules

Table 7.3: Comparing a synchronous distributed echo pseudo-code against Algorithms 4.2.2 (Echo) and 4.2.3 (Tree height), on trees, where h denotes tree height.

Algorithm	Time complexity	Program-size complexity
Synchronous distributed echo pseudo-code (Definition 4.2)	$2h$	14 pseudo-code lines
Algorithm 4.2.2 (Echo)	$2h + 4$	6 evolution rules
Algorithm 4.2.3 (Tree height)	$2h + 4$	7 evolution rules

Table 7.4: Comparing a synchronous distributed echo pseudo-code against Algorithms 4.3.3 (Graph echo), 4.3.4 (Cell heights) and 5.3.1 (Compute general's eccentricity), on digraphs, where e denotes the eccentricity of the source.

Algorithm	Time complexity	Program-size complexity
Synchronous distributed echo pseudo-code (Definition 4.2)	$2e$	14 pseudo-code lines
Algorithm 4.3.3 (Graph echo)	$2e + 6$	28 evolution rules
Algorithm 4.3.4 (Cell heights)	$2e + 6$	30 evolution rules
Algorithm 5.3.1 (Compute general's eccentricity)	$2e + 6$	30 evolution rules

Chapter 5 presented two P algorithms that solve the FSSP for tree- and graph-structured simple P systems. These FSSP solutions are comparable to the best-known FSSP solutions in cellular automata (CA), as indicated in Tables 7.5 and 7.6, respectively. Additionally, this chapter presented a broadcast-based P algorithm, Algorithm 5.2.1 (Decrementing hop-counter), and an echo-based P algorithm, Algorithm 5.3.1 (Compute general's eccentricity). As indicated in Tables 7.2 and 7.4, these P algorithms are comparable to the distributed synchronous broadcast (Definition 4.1) and echo (Definition 4.2) algorithms.

Table 7.5: Comparing the FSSP solution [59] of cellular automata (CA) against the static FSSP solution of Definition 5.8, where e denotes the eccentricity of the general.

Algorithm	Time complexity	Program-size complexity
CA FSSP solutions Nishitani and Honda [59]	$3e + 1$	135 transition rules
Static FSSP solution of Definition 5.8	$3e + 7$	37 evolution rules

Table 7.6: The FSSP solution [71] of cellular automata (CA) against the adaptive FSSP solution of Definition 5.13, where h and r denote the tree height and radius, respectively.

Algorithm	Time complexity	Program-size complexity
CA FSSP solutions Romani [71]	$h + 2r$	Not available
Adaptive FSSP solution of Definition 5.13	$h + 2r + 5$	24 evolution rules

Chapter 6 presented two P algorithms that solve the edge- and node-disjoint paths problems. These P algorithms run in polynomial time, comparable to the standard versions of the Ford-Fulkerson algorithms [48], however, these P algorithms contain $O(n^3)$ number of evolution rules, where n is the number of cells, while the disjoint paths pseudo-codes of Definition 6.2 contain 10 pseudo-code lines. Thus, while the edge- and node-disjoint paths P algorithms are comparable with respect to time complexity, they are not comparable with respect to program-size complexity. (i.e. 10 pseudo-code lines against $O(n^3)$ number of evolution rules). Even though these two P algorithms are not comparable with respect to size, they still form a component of the library of distributed algorithms in membrane systems.

As future work, we can consider using the newly proposed generic membrane system framework [56] for developing P algorithms, such that we can avoid generating P algorithms with polynomial number of evolution rules. Additionally, we can expand the library of fundamental distributed P algorithms, by presenting more distributed P algorithms, which could improve the usability of membrane systems for developing and designing distributed algorithms.

Bibliography

[1] Artiom Alhazov, Maurice Margenstern, and Sergey Verlan. Fast synchronization in P systems. In David W. Corne, Pierluigi Frisco, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Workshop on Membrane Computing*, volume 5391 of *Lecture Notes in Computer Science*, pages 118–128. Springer, 2008.

[2] Joshua J. Arulanandham. Implementing bead-sort with P systems. In Cristian Calude, Michael J. Dinneen, and Ferdinand Peper, editors, *UMC*, volume 2509 of *Lecture Notes in Computer Science*, pages 115–125. Springer, 2002.

[3] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics.* John Wiley & Sons, 2004.

[4] Robert Balzer. An 8-state minimal time solution to the firing squad synchronization problem. *Information and Control*, 10(1):22–42, 1967.

[5] Francesco Bernardini and Marian Gheorghe. Population P systems. *J. UCS*, 10(5):509–539, 2004.

[6] Francesco Bernardini and Marian Gheorghe. Cell communication in tissue P systems: universality results. *Soft Comput.*, 9(9):640–649, 2005.

[7] Francesco Bernardini, Marian Gheorghe, Maurice Margenstern, and Sergey Verlan. How to synchronize the activity of all components of a P system? *Int. J. Found. Comput. Sci.*, 19(5):1183–1198, 2008.

[8] André Berthiaume, Todd Bittner, Ljubomir Perkovic, Amber Settle, and Janos Simon. Bounding the firing synchronization problem on a ring. *Theor. Comput. Sci.*, 320(2-3):213–228, 2004.

[9] Cristian S. Calude and Michael J. Dinneen. Exact approximations of Omega numbers. *I. J. Bifurcation and Chaos*, 17(6):1937–1954, 2007.

[10] Constantin Carathéodory. *Theory of Functions of a Complex Variable.* Chelsea, 1954.

[11] Mónica Cardona, M. Angels Colomer, Antoni Margalida, Antoni Palau, Ignacio Pérez-Hurtado, Mario J. Pérez-Jiménez, and Delfí Sanuy. A computational modeling for real ecosystems based on P systems. *Natural Computing*, 10(1):39–53, 2011.

[12] Ernest J. H. Chang. Echo algorithms: Depth parallel operations on general graphs. *IEEE Trans. Software Eng.*, 8(4):391–401, 1982.

[13] Gabriel Ciobanu. Distributed algorithms over communicating membrane systems. *Biosystems*, 70(2):123–133, 2003.

[14] Gabriel Ciobanu, Rahul Desai, and Akash Kumar. Membrane systems and distributed computing. In Păun et al. [67], pages 187–202.

[15] Gabriel Ciobanu, Linqiang Pan, Gheorghe Păun, and Mario J. Pérez-Jiménez. P systems with minimal parallelism. *Theor. Comput. Sci.*, 378(1):117–130, 2007.

[16] Gabriel Ciobanu, Gheorghe Păun, and Gheorghe Ştefănescu. Sevilla carpets associated with P systems. In Matteo Cavaliere, Carlos Martín Vide, and Gheorghe Păun, editors, *BWMC*, pages 135–140, 2003.

[17] Erzsébet Csuhaj-Varjú, Maurice Margenstern, György Vaszil, and Sergey Verlan. On small universal antiport P systems. *Theor. Comput. Sci.*, 372(2-3):152–164, 2007.

[18] Daniel Díaz-Pernil, Pilar Gallego-Ortiz, Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez, and Agustin Riscos-Núñez. Descriptional complexity of tissue-like P systems with cell division. In Cristian S. Calude, José Félix Costa, Nachum Dershowitz, Elisabete Freire, and Grzegorz Rozenberg, editors, *UC*, volume 5715 of *Lecture Notes in Computer Science*, pages 168–178. Springer, 2009.

[19] Michael J. Dinneen. A program-size complexity measure for mathematical problems and conjectures. In Michael J. Dinneen, Bakhadyr Khoussainov, and André

Nies, editors, *Computation, Physics and Beyond*, volume 7160 of *Lecture Notes in Computer Science*, pages 81–93. Springer, 2012.

[20] Michael J. Dinneen and Yun-Bum Kim. A new universality result on P systems. Report CDMTCS-423, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, Auckland, New Zealand, July 2012.

[21] Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu. New solutions to the firing squad synchronization problems for neural and hyperdag P systems. *Electronic Proceedings in Theoretical Computer Science*, 11:107–122, 2009.

[22] Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu. Edge- and node-disjoint paths in P systems. *Electronic Proceedings in Theoretical Computer Science*, 40:121–141, 2010.

[23] Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu. A faster P solution for the Byzantine agreement problem. In Marian Gheorghe, Thomas Hinze, and Gheorghe Păun, editors, *Conference on Membrane Computing*, volume 6501 of *Lecture Notes in Computer Science*, pages 175–197. Springer-Verlag, Berlin Heidelberg, 2010.

[24] Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu. P systems and the Byzantine agreement. *Journal of Logic and Algebraic Programming*, 79(6):334–349, 2010.

[25] Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu. Synchronization in P modules. In Cristian S. Calude, Masami Hagiya, Kenichi Morita, Grzegorz Rozenberg, and Jon Timmis, editors, *Unconventional Computation*, volume 6079 of *Lecture Notes in Computer Science*, pages 32–44. Springer-Verlag, Berlin Heidelberg, 2010.

[26] Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu. An adaptive algorithm for P system synchronization. In Gheorghe et al. [36], pages 139–164.

[27] Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu. Faster synchronization in P systems. *Natural Computing*, 11:107–115, 2012.

[28] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, 1972.

[29] Shimon Even, Ami Litman, and Peter Winkler. Computing with snakes in directed networks of automata. *J. Algorithms*, 24(1):158–170, 1997.

[30] Roger L. Freeman. *Fundamentals of Telecommunications, 2nd Edition.* Wiley-IEEE Press, 2005.

[31] Rudolf Freund, Lila Kari, Marion Oswald, and Petr Sosík. Computationally universal P systems without priorities: two catalysts are sufficient. *Theor. Comput. Sci.*, 330(2):251–266, 2005.

[32] Rudolf Freund and Andrei Păun. Membrane systems with symport/antiport rules: Universality results. In Păun et al. [67], pages 270–287.

[33] Manuel García-Quismondo, Rosa Gutiérrez-Escudero, Ignacio Pérez-Hurtado, Mario J. Pérez-Jiménez, and Agustin Riscos-Núñez. An overview of P-lingua 2.0. In Păun et al. [65], pages 264–288.

[34] Marian Gheorghe, Florentin Ipate, and Ciprian Dragomir. Formal verification and testing based on P systems. In Păun et al. [65], pages 54–65.

[35] Marian Gheorghe, Florentin Ipate, Raluca Lefticaru, and Ciprian Dragomir. An integrated approach to P systems formal verification. In *Int. Conf. on Membrane Computing*, pages 226–239, 2010.

[36] Marian Gheorghe, Gheorghe Păun, Grzegorz Rozenberg, Arto Salomaa, and Sergey Verlan, editors. *Membrane Computing - 12th International Conference, CMC 2011, Fontainebleau, France, August 23-26, 2011, Revised Selected Papers*, volume 7184 of *Lecture Notes in Computer Science.* Springer, 2012.

[37] Andrew V. Goldberg, Éva Tardos, and Robert E. Tarjan. Network flow algorithms. In *Algorithms and Combinatorics*, volume 9, pages 101–164. Springer-Verlag, 1990.

[38] Eiichi Goto. A minimal time solution of the firing squad problem. Course notes for Applied Mathematics 298, Harvard University, 1962.

[39] John J. Grefenstette. Network structure and the firing squad synchronization problem. *J. Comput. Syst. Sci.*, 26(1):139–152, 1983.

[40] Jozef Gruska, Salvatore La Torre, and Mimmo Parente. Optimal time and communication solutions of firing squad synchronization problems on square arrays, toruses and rings. In Cristian Calude, Elena Calude, and Michael J. Dinneen, editors, *Developments in Language Theory*, volume 3340 of *Lecture Notes in Computer Science*, pages 200–211. Springer, 2004.

[41] Gabriel Y. Handler and Pitu B. Mirchandani. *Location on Networks: Theory and Algorithms.* MIT Press, 1979.

[42] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[43] Tim Carter Humphrey. *Cell Cycle Control: Mechanisms and Protocols.* Humana Press, 2005.

[44] Mihai Ionescu, Gheorghe Păun, and Takashi Yokomori. Spiking neural P systems. *Fundam. Inform.*, 71(2-3):279–308, 2006.

[45] Florentin Ipate, Raluca Lefticaru, Ignacio Pérez-Hurtado, Mario J. Pérez-Jiménez, and Cristina Tudose. Formal verification of P systems with active membranes through model checking. In Gheorghe et al. [36], pages 215–225.

[46] Tseren-Onolt Ishdorj and Mihai Ionescu. Replicative - distribution rules in P systems with active membranes. In Zhiming Liu and Keijiro Araki, editors, *ICTAC*, volume 3407 of *Lecture Notes in Computer Science*, pages 68–83. Springer-Verlag, 2004.

[47] Tseren-Onolt Ishdorj, Alberto Leporati, Linqiang Pan, and Jun Wang. Solving NP-complete problems by spiking neural P systems with budding rules. In Păun et al. [65], pages 335–353.

[48] Lester R. Ford Jr. and D. Ray Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[49] Ivan Korec. Small universal register machines. *Theor. Comput. Sci.*, 168(2):267–301, 1996.

[50] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[51] Carlos Martín-Vide, Gheorghe Păun, Juan Pazos, and Alfonso Rodríguez-Patón. Tissue P systems. *Theor. Comput. Sci.*, 296(2):295–326, 2003.

[52] Jacques Mazoyer. A six-state minimal time solution to the firing squad synchronization problem. *Theor. Comput. Sci.*, 50:183–238, 1987.

[53] Marvin L. Minsky. *Computation: finite and infinite machines.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[54] Edward F. Moore. The firing squad synchronization problem. In E.F. Moore, editor, *Sequential Machines, Selected Papers*, pages 213–214. Addison-Wesley, Reading MA., 1964.

[55] F. R. Moore and G. G. Langdon. A generalized firing squad problem. *Information and Control*, 12(3):212–220, 1968.

[56] Radu Nicolescu. Parallel and distributed algorithms in P systems. In Gheorghe et al. [36], pages 35–50.

[57] Radu Nicolescu, Michael J. Dinneen, and Yun-Bum Kim. Discovering the membrane topology of hyperdag P systems. In Păun et al. [65], pages 410–435.

[58] Radu Nicolescu, Michael J. Dinneen, and Yun-Bum Kim. Towards structured modelling with hyperdag P systems. *International Journal of Computers, Communications and Control*, 2:209–222, 2010.

[59] Yasuaki Nishitani and Namio Honda. The firing squad synchronization problem for graphs. *Theor. Comput. Sci.*, 14:39–61, 1981.

[60] Rafail Ostrovsky and Daniel Shawcross Wilkerson. Faster computation on directed networks of automata. In *In Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 38–46. ACM, 1995.

[61] Rafail Ostrovsky and Daniel Shawcross Wilkerson. Faster computation on directed networks of automata (extended abstract). In James H. Anderson, editor, *PODC*, pages 38–46. ACM, 1995.

[62] Andrei Păun and Gheorghe Păun. The power of communication: P systems with symport/antiport. *New Generation Comput.*, 20(3):295–306, 2002.

[63] Gheorghe Păun. *Membrane Computing: An Introduction.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[64] Gheorghe Păun. Introduction to membrane computing. In *Applications of Membrane Computing*, pages 1–42. Springer-Verlag, 2006.

[65] Gheorghe Păun, Mario J. Pérez-Jiménez, Agustín Riscos-Núñez, Grzegorz Rozenberg, and Arto Salomaa, editors. *Membrane Computing, 10th International Workshop, WMC 2009, Curtea de Argeş, Romania, August 24-27, 2009. Revised Selected and Invited Papers*, volume 5957 of *Lecture Notes in Computer Science*. Springer-Verlag, 2010.

[66] Gheorghe Păun and Radu A. Păun. Membrane computing as a framework for modeling economic processes. In *SYNASC*, pages 11–18. IEEE Computer Society, 2005.

[67] Gheorghe Păun, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron, editors. *Membrane Computing, International Workshop, WMC-CdeA 2002, Curtea de Argeş, Romania, August 19-23, 2002, Revised Papers*, volume 2597 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[68] Andrei Păun and Gheorghe Păun. Small universal spiking neural P systems. *Biosystems*, 90(1):48–60, 2007.

[69] Gheorghe Păun. Computing with membranes. *J. Comput. Syst. Sci.*, 61(1):108–143, August 2000.

[70] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. *The Oxford Handbook of Membrane Computing.* Oxford University Press, Inc., New York, NY, USA, 2010.

[71] Francesco Romani. Cellular automata synchronization. *Inf. Sci.*, 10(4):299–318, 1976.

[72] Hubert Schmid and Thomas Worsch. The firing squad synchronization problem with many generals for one-dimensional CA. In Jean-Jacques Lévy, Ernst W. Mayr, and John C. Mitchell, editors, *IFIP TCS*, pages 111–124. Kluwer, 2004.

[73] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating System Concepts, Seventh Edition.* Wiley, 2004.

[74] Michael Sipser. *Introduction to the theory of computation.* PWS Publishing Company, 1997.

[75] Steven Skiena. *Implementing discrete mathematics - combinatorics and graph theory with Mathematica.* Addison-Wesley, 1990.

[76] William A. Stein et al. *Sage Mathematics Software (Version 4.6).* The Sage Development Team.

[77] J. W. Suurballe and Robert Endre Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14(2):325–336, 1984.

[78] Helge Szwerinski. Time-optimal solution of the firing-squad-synchronization-problem for n-dimensional rectangles with the general at an arbitrary position. *Theor. Comput. Sci.*, 19(3):305–320, 1982.

[79] Gerard Tel. *Introduction to distributed algorithms.* Cambridge University Press, New York, NY, USA, 1994.

[80] Hiroshi Umeo, Naoki Kamikawa, Kouji Nishioka, and Shunsuke Akiguchi. Generalized firing squad synchronization protocols for one-dimensional cellular automata—a survey. *Acta Physica Polonica B Proceedings Supplement*, 3(2):267–289, 2010.

[81] Abraham Waksman. An optimum solution to the firing squad synchronization problem. *Information and Control*, 9(1):66–78, 1966.

[82] Eric W. Weisstein. Prüfer code, from MathWorld—a Wolfram web resource.

[83] Claudio Zandron, Claudio Ferretti, and Giancarlo Mauri. Solving NP-complete problems using P systems with active membranes. In Ioannis Antoniou, Cristian S. Calude, and Michael J. Dinneen, editors, *UMC*, pages 289–301. Springer-Verlag, 2000.