AGH

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA INFORMATYKI

PRACA DYPLOMOWA MAGISTERSKA

Tensor Networks approach to simulating Continuous-Time Stochastic Automata Networks

Symulacja sieci automatów stochastycznych za pomocą sieci tensorowych

Autor:	*Bartłomiej Grochal*
Kierunek studiów:	*Informatyka*
Opiekun pracy:	*dr inż. Katarzyna Rycerz*
Konsultacja merytoryczna:	*dr hab. inż. Piotr Gawron, Instytut Informatyki Teoretycznej i Stosowanej Polskiej Akademii Nauk w Gliwicach*

Kraków, 2018

First and foremost, I would like to express my sincere gratitude to my supervisor, dr inż. Katarzyna Rycerz and dr hab. inż. Piotr Gawron from the Institute of Theoretical and Applied Informatics, Polish Academy of Sciences in Gliwice for their invaluable support and leading my scientific research conducted within this thesis. Additionally, I would like to thank prof. dr hab. inż. Robert Schaefer, dr hab. inż. Renata Słota and dr inż. Marian Bubak for their valuable comments improving the quality of the following outcomes and document. Similarly, I would like to express my appreciation to dr hab. inż. Aleksander Byrski for assisting me in my first steps in the world of scientific research.

Finally, I cannot forget about all the people who influenced me during twenty-four years of my life. Thank you.

The research was partially financed by the National Science Centre, Poland—project number 2014/15/B/ST6/05204.

Abstract

Theoretical performance evaluation of highly parallel, distributed systems has been a subject matter of numerous research efforts in recent decades. One of the most widespread approaches dedicated to the simulation of such models are stochastic Markovian models of various types. However their greatest obstacle is an inability to handle systems built up from multiple components due to the ubiquitous state space explosion issue. Hence the Stochastic Automata Networks formalism, grounded in the Kronecker algebra foundations became a method of first choice recently, since its inherent structured representation of the model under simulation turns out to be notably effective when simulating a collection of loosely coupled components, such as distributed environment processes. Despite the thirty-years old history of Stochastic Automata Networks, which resulted in the elaboration of multiple novel simulation algorithms, there is still a serious gap due to the lack of numerical method for obtaining transient probability distributions of states belonging to the modelled system at arbitrary instant of time.

In order to tackle the aforementioned issue, the TNSAN algorithm – built on top of the Tensor Networks formalism, which has proven efficiency in overcoming the curse of dimensionality problem originating form quantum many-body states – is proposed. The main contribution of the following thesis is a formally derived concept of reducing the task of determining a transient probability distribution of states to the matter of iterative contraction between such networks. Besides all the theoretical foundations of the TNSAN algorithm, this document presents preliminary numerical evaluation results obtained by performing repeated simulations of the resource sharing benchmark model, as well as discusses strengths and limitations of the proposed approach.

Within this thesis, a novel algorithm for the simulation of Stochastic Automata Networks with Tensor Networks is introduced and comprehensively justified. The TNSAN approach allows to determine a transient probability distribution of states belonging to any model expressible in terms of stochastic automata, however the method in question is exceptionally effective for particular areas of applicability. The results obtained within this thesis constitute a contribution to the scientific domain of this research since they exhibit a previously unexplored approach based on the hybridization of Stochastic Automata Networks and Tensor Networks formalisms.

Keywords: Continuous-Time Markov Chain, Stochastic Automata Network, Tensor Network, performance evaluation

Table of Contents

List of Figures

List of Tables

List of Code Snippets

List of Symbols

General

$\mathbb{R}$	Real numbers
$\mathbb{R}^{0+}$	Non-negative real numbers
$\mathbb{R}^{+}$	Positive real numbers
$\mathbb{N}$	Natural numbers (including zero)
$\mathbb{N}^{+}$	Positive natural numbers

Matrices, vectors and scalars

A	Matrix A
$A_{m\times n} = [a_{ij}]$	Matrix A composed of elements a_{ij}, where: $1 \leq i \leq m$ is the row number and $1 \leq j \leq n$ is the column number
$A^{(1)}, A^{(2)}, \ldots, A^{(N)}$	N matrices
I_n	Identity matrix of size n
0_n	Zero (null) matrix of size n
$S_{m,n}$	Matrix of the $\sigma_{m,n}$ Perfect Shuffle permutation
$\mathbf{v}$	Row vector $\mathbf{v}$
$\mathbf{v}^{\mathsf{T}}$	Column vector being the transposition of the vector $\mathbf{v}$
$\mathbf{v}_n = \left(v^{(1)}, v^{(2)}, \ldots, v^{(n)}\right)$	Vector $\mathbf{v}$ composed of n elements $v^{(i)}$, $1 \leq i \leq n$
$\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \ldots, \mathbf{v}^{(N)}$	N vectors
a	Scalar value

Basic algebra of matrices

$A + B$	Sum of two matrices A and B
$\sum_{k=1}^{N} A^{(k)}$	Sum of N matrices $A^{(1)}, A^{(2)}, \ldots, A^{(N)}$
$AB, A \cdot B$	Ordinary product of two matrices A and B
$\prod_{k=1}^{N} A^{(k)}$	Ordinary product of N matrices $A^{(1)}, A^{(2)}, \ldots, A^{(N)}$

$\mathbf{v}A, \mathbf{v} \cdot A$	Product of the vector **v** and the matrix A
$aB, a \cdot B$	Product of the scalar a and the matrix B
A^k	k-th power of the matrix A
A^{-1}	Inverse of the matrix A
$\exp(A)$, e^A	Application of the exponential function to the matrix A

Kronecker (tensor) algebra

$A \otimes B$	Kronecker product of two matrices A and B
$\bigotimes_{k=1}^{N} A^{(k)}$	Kronecker product of N matrices $A^{(1)}, A^{(2)}, \ldots, A^{(N)}$
$A \oplus B$	Kronecker sum of two matrices A and B
$\bigoplus_{k=1}^{N} A^{(k)}$	Kronecker sum of N matrices $A^{(1)}, A^{(2)}, \ldots, A^{(N)}$

Sets

$\{1, 2, \ldots, n\}$	Set of n consecutive, increasing integers starting from one
$\mathscr{A} = \{a^{(1)}, a^{(2)}, \ldots, a^{(N)}\}$	Set $\mathscr{A}$ composed of N elements $a^{(1)}, a^{(2)}, \ldots, a^{(N)}$
$\mathscr{M}_{m \times n}$	Set of all matrices with m rows and n columns
$\mathscr{M}_n$	Set of all square matrices of size n

Basic algebra of sets

$\forall$	Universal quantifier
$\exists$	Existential quantifier
$a \in \mathscr{A}$	Object a is a member of the set $\mathscr{A}$
$\mathscr{A} \times \mathscr{B}$	Cartesian product of sets $\mathscr{A}$ and $\mathscr{B}$
$\mathscr{A}^k$	k-th Cartesian power of the set $\mathscr{A}$

Stochastic Automata

$\mathcal{A}$	Stochastic Automaton
$\mathscr{S}_{\mathcal{A}} = \{s^{(1)}, s^{(2)}, \ldots, s^{(S_{\mathcal{A}})}\}$	State space of the automaton $\mathcal{A}$ composed of $S_{\mathcal{A}}$ states denoted by $s^{(i)}$, $1 \leq i \leq S_{\mathcal{A}}$
$\mathscr{L}_{\mathcal{A}} = \{l^{(1)}, l^{(2)}, \ldots, l^{(L_{\mathcal{A}})}\}$	Set of labels associated with the automaton $\mathcal{A}$ composed of $L_{\mathcal{A}}$ labels denoted by $l^{(i)}$, $1 \leq i \leq L_{\mathcal{A}}$
$f_{\mathcal{A}}$	State-transition function of the automaton $\mathcal{A}$
$p^{e}_{s^{(i)}}$	Alternative probability function associated with departing from the state $s^{(i)}$ while processing the event e

$\mathbf{p}_{\mathcal{A}}(t)$	Transient probability distribution of states belonging to the automaton $\mathcal{A}$ at time t
$P(t)$	Transition probability matrix at time t
$Q_{\mathcal{A}}$	Transition rate matrix of the automaton $\mathcal{A}$

Stochastic Automata Networks

$\boldsymbol{\mathcal{N}}$	Stochastic Automata Network
$\mathscr{A}_{\boldsymbol{\mathcal{N}}} = \{\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(N_{\boldsymbol{\mathcal{N}}})}\}$	Set of $N_{\boldsymbol{\mathcal{N}}}$ automata composing the network $\boldsymbol{\mathcal{N}}$, denoted by $\mathcal{A}^{(i)}$, $1 \leq i \leq N_{\boldsymbol{\mathcal{N}}}$
$\mathscr{E}_{\boldsymbol{\mathcal{N}}} = \{e^{(1)}, e^{(2)}, \ldots, e^{(E_{\boldsymbol{\mathcal{N}}})}\}$	Set of $E_{\boldsymbol{\mathcal{N}}}$ events incorporated by the network $\boldsymbol{\mathcal{N}}$, denoted by $e^{(i)}$, $1 \leq i \leq E_{\boldsymbol{\mathcal{N}}}$
$\mathscr{E}_{\boldsymbol{\mathcal{N}}}^{loc}$, $\mathscr{E}_{\boldsymbol{\mathcal{N}}}^{syn}$	Sets of, respectively, local and synchronizing events associated with the network $\boldsymbol{\mathcal{N}}$
$\mathscr{S}_{\mathcal{N}}$	Product State Space of the network $\boldsymbol{\mathcal{N}}$ of size $S_{\mathcal{N}}$
$\mathscr{R}_{\boldsymbol{\mathcal{N}}}$	Reachable State Space of the network $\boldsymbol{\mathcal{N}}$ of size $R_{\mathcal{N}}$
$Q_{\mathcal{A}}^{loc}$	A part of the transition rate matrix associated with the automaton $\mathcal{A}$ with entries corresponding to its local transitions
$Q_{\mathcal{A}}^{e_{pos}}$	A part of the transition rate matrix associated with the automaton $\mathcal{A}$ with (positive) non-diagonal entries corresponding to the synchronized transition triggered by the event e
$Q_{\mathcal{A}}^{e_{neg}}$	A part of the transition rate matrix associated with the automaton $\mathcal{A}$ with (negative) diagonal entries corresponding to the synchronized transition triggered by the event e

Tensors

$T_{\alpha^{(1)},\alpha^{(2)},\ldots,\alpha^{(d)}}$	Tensor of d indices denoted by $\alpha^{(i)}$, $1 \leq i \leq d$
$T^{(1)}_{\alpha_1^{(1)},\alpha_1^{(2)},\ldots,\alpha_1^{(d_1)}}, \ldots, T^{(N)}_{\alpha_N^{(1)},\alpha_N^{(2)},\ldots,\alpha_N^{(d_N)}}$	N tensors, each of which defined over $d_1, \ldots, d_N$ indices respectively
$T_{i_1,i_2,\ldots,i_d}$	Element of the tensor $T_{\alpha^{(1)},\alpha^{(2)},\ldots,\alpha^{(d)}}$
$\lvert\alpha\rvert$	Size of the index α
$T \circ_\alpha S$	Contraction of tensors T and S by the common index α

Simulation algorithm

Δt	Simulation time step

$\mathbb{S}^{t}_{\boldsymbol{\mathcal{N}}}$	Tensor Network representing global state of the network $\boldsymbol{\mathcal{N}}$ at time t
$\mathbb{E}^{\Delta t}_{\boldsymbol{\mathcal{N}}}$	Tensor Network representing evolution of the network $\boldsymbol{\mathcal{N}}$ over time Δt

Abbreviations and Acronyms

CTMC	Continuous-Time Markov Chain
PSS	Product State Space
RSS	Reachable State Space
SAN	Stochastic Automata Network
TN	Tensor Network
TND	Tensor Network Diagram

All the abbreviations and acronyms ended with “s” denote the plural form.

1. Introduction

This Chapter briefly introduces the motivation of this research (Section 1.1) in order to delineate the context and basic concepts related to the thesis, which are extensively described in the succeeding three introductory Chapters. Then, the most important goals of this study are drafted in the Section 1.2. Finally, a structure of this document is outlined in the Section 1.3.

1.1. Motivation

Out of multiple high-level formalisms dedicated to the structured Markovian models description [11, 17], Stochastic Automata Network (SAN), introduced by Plateau [35], is one of the most commonly used approaches nowadays, which is notably effective when a system under simulation may be represented by a collection of infrequently interacting components [36]. Although initially considered impractical due to the lack of efficient numerical algorithms designed for their simulation [3, 8, 10], automata networks are actually established as a method of first choice when it comes to studying of distributed systems' properties. One of the greatest advantages of the SAN formalism is that it employs structured, Kronecker algebra-based, memory-efficient representation of the underlying Markov chain, which does not require neither generating nor storing a global transition rate matrix [36]. Therefore, this approach seems to be a remedy for the ubiquitous state space explosion issue [36] associated with the holistic perception of large Markovian models. During the thirty-year history of SANs, there have been multiple numerical methods elaborated (cf. Section 5.1), which facilitate the calculation of their probability distributions over time. However, there is still much research to be done, especially in case of determining transient probability distributions.

Numerical simulations, next to classical experiments and mathematical formulations, are basic methods of studying physical properties of complex systems [25, 37]. The huge advancement in this field has been made in recent years due to the Tensor Networks (TNs) formalism, which actually is the essence of many scientific disciplines, such as computational physics and quantum information science [37]. Structures of connected tensors turned out to be especially effective in overcoming the curse of dimensionality problem associated with quantum many-body states [7], as well as in understanding and investigating quantum algorithms through building quantum circuits, which constitute a representation of the special class of TNs [5]. The use of this formalism not only enables to accelerate the efficiency of quantum systems examination and study their properties in greater detail [5], but also establishes new boundaries for numerical methods, which are limited only by the amount and structure of the quantum entanglement associated with the system under simulation [30]. Moreover, TNs go hand-in-hand with a powerful, descriptive graphical language for representing complicated mathematical equations in terms of tensors and their interactions, called Tensor Network Diagrams (TNDs) [30].

The TNs formalism seems to perfectly fit both the limitations of Markovian models and the structured representation of SANs. Firstly, their applicability to numerical models struggling with the curse of dimensionality issue brings a lot of hope for overcoming the equivalent state space explosion problem. Moreover, the Kronecker algebra formalism underlying stochastic automata is inherently suitable for building tensor structures, which is covered in greater detail within subsequent Chapters of this document.

1.2. Goals

The main contribution of this research is the elaboration and formal description of a numerical method designed for computing transient probability distributions of states belonging to simulated SANs. This algorithm, called TNSAN, employs the Kronecker structure of a transition rate matrix describing the evolution of a Continuous-Time Markov Chain (CTMC) underlying the simulated network, and therefore fills the gap in the lack of efficient numerical methods for determining transient probability distributions of systems expressible by means of the SAN formalism. Additionally, this thesis presents an evaluation of the method proposed not only in terms of formal analysis of its properties, but also regarding some preliminary experimental results obtained by performing numerical tests on the implementation of this algorithm.
Please note that a detailed description of research objectives taken within the following thesis is provided by the Section 5.3.

1.3. Thesis outline

This document is laid out as follows. Chapter 1 outlines the scientific domain, fundamental assumptions and general goals of this research. The next three Chapters introduce fundamental facts in terms of Algebra and Probability theory (Chapter 2), SANs (Chapter 3) and TNs (Chapter 4). Then, a profound description and an analysis of the problem addressed is presented in the context of advances in the subject matter of this thesis within Chapter 5. In Chapter 6 a methodology chosen for this research work is delineated and justified. Finally, the main contribution of this thesis, the TNSAN algorithm, is described and thoroughly studied in Chapter 7, while Chapter 8 presents preliminary experimental results obtained by conducting multiple test scenarios. Chapter 9 summarizes this thesis and suggests further research related to the problem addressed.

2. Mathematical background

This Chapter collects a list of advanced mathematical theorems and definitions, which the Reader should be familiar with before moving on to the further parts of this thesis. It is already assumed that the Reader has basic comprehension of Algebra and Probability theory, at least at the level of the introductory university course in mathematics, therefore the following Chapter presents only supplementary facts considered extracurricular in terms of both Algebra (Section 2.1) and Stochastic Processes (Section 2.2). Throughout this thesis it is also assumed that all scalars, vectors, matrices and tensors are defined over real numbers (denoted by $\mathbb{R}$), while all their indices and enumerations are identified by positive natural numbers (denoted by $\mathbb{N}^+$) starting from one, unless otherwise stated. Finally, please note that the list of all mathematical symbols used across this document is provided separately for reference, therefore the semantics of a particular symbol is explained only when first used.

2.1. Linear and multilinear Algebra

Following the notation proposed by Horn and Johnson [24], the set of all rectangular matrices with m rows and n columns is denoted by $\mathscr{M}_{m\times n}$, while the set of all square matrices of size n is denoted simply by $\mathscr{M}_n$. However, when a matrix A of m rows and n columns is defined by its elements a_{ij} (where: $1 \leq i \leq m$ is the row number and $1 \leq j \leq m$ is the column number), the following notation is used: $A_{m\times n} = [a_{ij}]$. In general, matrices are denoted by uppercase italic Roman letters (especially the identity matrix of size n is given by I_n, albeit the zero (also called null) matrix of size n is referenced by 0_n exceptionally), vectors are written as lowercase bold Roman letters, while scalars are represented by lowercase italic Roman letters.

2.1.1. Kronecker algebra

Basic building block of a structured representation of any SAN is the Kronecker product of matrices, introduced by the Definition 1. Another fundamental operation of the Kronecker algebra is the Kronecker sum which, in fact, is a special form of ordinary sum of Kronecker products. This operation is formalized by the Definition 2. Please note that the Kronecker sum is defined for square matrices only, while the Kronecker product is valid for any two rectangular matrices.

Definition 1. [28] Let $A_{m\times n} = [a_{ij}]$ and $B_{p\times q} = [b_{ij}]$. The *Kronecker* (also called *tensor*) *product* is the block matrix $(A \otimes B) \in \mathscr{M}_{mp\times nq}$, such that:

$$A \otimes B \overset{\text{def}}{=} \begin{bmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & \dots & a_{11}b_{1q} & \dots & \dots & a_{1n}b_{11} & \dots & a_{1n}b_{1q} \\ \vdots & \ddots & \vdots & \dots & \dots & \vdots & \ddots & \vdots \\ a_{11}b_{p1} & \dots & a_{11}b_{pq} & \dots & \dots & a_{1n}b_{p1} & \dots & a_{1n}b_{pq} \\ \vdots & & \vdots & \ddots & & \vdots & & \vdots \\ \vdots & & \vdots & & \ddots & \vdots & & \vdots \\ a_{m1}b_{11} & \dots & a_{m1}b_{1q} & \dots & \dots & a_{mn}b_{11} & \dots & a_{mn}b_{1q} \\ \vdots & \ddots & \vdots & \dots & \dots & \vdots & \ddots & \vdots \\ a_{m1}b_{p1} & \dots & a_{m1}b_{pq} & \dots & \dots & a_{mn}b_{p1} & \dots & a_{mn}b_{pq} \end{bmatrix}.$$

Definition 2. [28] Let $A \in \mathscr{M}_n$ and $B \in \mathscr{M}_m$. The *Kronecker* (also called *tensor*) *sum* is the matrix $(A \oplus B) \in \mathscr{M}_{nm}$ such that:

$$A \oplus B \overset{\text{def}}{=} A \otimes I_m + I_n \otimes B.$$

Remark. [36] Both the Kronecker product and the Kronecker sum satisfy the associativity property, therefore the generalized operations:

$$\bigotimes_{k=1}^{N} A^{(k)} = A^{(1)} \otimes A^{(2)} \otimes \ldots \otimes A^{(N)} \quad \text{and} \quad \bigoplus_{k=1}^{N} B^{(k)} = B^{(1)} \oplus B^{(2)} \oplus \ldots \oplus B^{(N)}$$

for any rectangular matrices $A^{(1)}, A^{(2)}, \ldots, A^{(N)}$ and any square matrices $B^{(1)}, B^{(2)}, \ldots, B^{(N)}$ are well defined.

The Kronecker algebra operations satisfy multiple properties summarized and deeply studied in [28, 40]. Some selected of them, considered useful in case of this research, are summarized by the Theorem 1. Please note that proofs of the following identities are provided by the referenced literature.

Theorem 1. [14, 28, 40] The following identities are satisfied:

1. The Kronecker product of identity matrices is an identity matrix:
$$I_m \otimes I_n = I_n \otimes I_m = I_{nm}. \tag{2.1}$$

2. Let $A \in \mathscr{M}_{m\times n}$ and $B \in \mathscr{M}_{p\times q}$. The Kronecker product is compatible with the multiplication by a scalar value $a \in \mathbb{R}$:
$$(aA) \otimes B = A \otimes (aB) = a(A \otimes B). \tag{2.2}$$

3. Let $A, B \in \mathscr{M}_{m\times n}$ and $C, D \in \mathscr{M}_{p\times q}$. The Kronecker product is distributive over the ordinary matrix addition:
$$(A + B) \otimes (C + D) = A \otimes C + A \otimes D + B \otimes C + B \otimes D. \tag{2.3}$$

In particular, the Kronecker product is both left-distributive:

$$A \otimes (C + D) = A \otimes C + A \otimes D, \tag{2.4}$$

and right-distributive:

$$(A + B) \otimes C = A \otimes C + B \otimes C. \tag{2.5}$$

4. Let $A \in \mathscr{M}_{k\times l}$, $B \in \mathscr{M}_{m\times n}$ and $C \in \mathscr{M}_{p\times q}$. The Kronecker product is associative, which means that it is both left-associative:

$$A \otimes B \otimes C = (A \otimes B) \otimes C, \tag{2.6}$$

and right-associative:

$$A \otimes B \otimes C = A \otimes (B \otimes C). \tag{2.7}$$

5. Let $A \in \mathscr{M}_{m\times n}$, $B \in \mathscr{M}_{p\times q}$, $C \in \mathscr{M}_{n\times k}$ and $D \in \mathscr{M}_{q\times r}$. The Kronecker product is compatible with the ordinary matrix multiplication:

$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD). \tag{2.8}$$

6. Let $A \in \mathscr{M}_n$ and $B \in \mathscr{M}_m$. For any natural number (including zero) $n \in \mathbb{N}$, a power of the Kronecker product may be expressed in terms of powers of its factors:

$$(A \otimes B)^n = A^n \otimes B^n. \tag{2.9}$$

7. Let $A \in \mathscr{M}_{m\times n}$ and $B \in \mathscr{M}_{p\times q}$. The Kronecker product is pseudo-commutative:

$$B \otimes A = S_{m,p}(A \otimes B)S_{q,n}, \tag{2.10}$$

where: $S_{k,l}$ is the matrix representation of the $\sigma_{k,l}$ Perfect Shuffle permutation (cf. Definition 5) transforming k groups of l elements into l groups of k elements.

2.1.2. Matrix exponential

One of the fundamental contributions of this thesis is an elaboration of efficient method for computing exponential of a matrix represented in a structured form. Therefore, the exponential function acting on matrices is formally introduced by the Definition 3 and Theorem 2. Then, the commutator of two matrices is described by the Definition 4 and followed by selected properties of the matrix exponential summarized by the Theorem 3 (with proofs provided by the referenced literature).

Definition 3. [22] Let $A \in \mathscr{M}_n$. The *exponential of* A is the $\exp(A) \in \mathscr{M}_n$ (equivalently denoted by e^A) matrix given by the following infinite power series:

$$\exp(A) \equiv e^A \overset{\text{def}}{=} \sum_{k=0}^{\infty} \frac{1}{k!}A^k = I_n + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + \ldots,$$

where: $k!$ is the factorial of k.

Theorem 2. [22] The series given by the Definition 3 converges for any matrix $A \in \mathscr{M}_n$, therefore the matrix exponential function is well-defined for any square matrix.

Proof. Cf. Proposition 2.1 in [22]. □

Definition 4. [22] Let $A, B \in \mathscr{M}_n$. The *commutator* of matrices A and B is the matrix $[A, B] \in \mathscr{M}_n$ such that:

$$[A, B] \overset{\text{def}}{=} AB - BA.$$

Note that if $[A, B] = 0_n$, then one can say that matrices A and B *commute* or - equivalently - that they are *commutative*.

Theorem 3. [22] The following identities are satisfied:

1. The exponential of the zero matrix is the identity matrix:
$$\exp(0_n) = I_n. \tag{2.11}$$
2. For any two commutative matrices $A, B \in \mathscr{M}_n$, the exponential of their sum may be expressed in terms of a product of their exponentials:
$$\exp(A + B) = \exp(B + A) = \exp(A)\exp(B) = \exp(B)\exp(A). \tag{2.12}$$
3. Let $A, B \in \mathscr{M}_n$ and suppose B is invertible. Then, the following formula holds:
$$\exp(BAB^{-1}) = B\exp(A)B^{-1}, \tag{2.13}$$
where: $B^{-1} \in \mathscr{M}_n$ is the inverse of B.

As stated by the Equation 2.12, the matrix exponential function does not preserve all widely-known properties satisfied by the algebraic exponential function. Especially, the exponential of a sum of two matrices is equivalent to a product of their exponentials if these matrices are commutative. There is, however, a possibility to express the exponential of a sum as a product of exponentials for any two (even non-commutative) square matrices, which is summarized by the Theorem 4.

Theorem 4. [22] Let $A, B \in \mathscr{M}_n$. The *Lie product formula* (also called *Trotter decomposition* or *Suzuki-Trotter expansion of the first order*) states that:

$$\exp(A + B) = \lim_{k \to \infty} \left(\exp\left(\frac{1}{k}A\right) \exp\left(\frac{1}{k}B\right) \right)^k.$$

Proof. Cf. Theorem 2.10 in [22]. □

The following Theorems 5 and 6 present selected properties of the matrix exponential applied to the Kronecker product of matrices. The former one introduces the relationship between the exponential of the Kronecker sum and the Kronecker product of exponentials, while the latter one describes a powerful approach to compute the exponential of the Kronecker product containing identity factors.

Theorem 5. [21] Let $A \in \mathscr{M}_n$ and $B \in \mathscr{M}_m$. The exponential of the Kronecker sum of these matrices may be expressed in terms of the Kronecker product of their exponentials:

$$\exp(A \oplus B) = \exp(A) \otimes \exp(B).$$

Proof. Cf. [21]. □

Theorem 6. The following identities are satisfied:

1. Let $A \in \mathscr{M}_n$. For any identity matrix I_m:
$$\exp(A \otimes I_m) = \exp(A) \otimes I_m. \tag{2.14}$$

2. Let $A \in \mathscr{M}_n$. For any identity matrix I_m:
$$\exp(I_m \otimes A) = I_m \otimes \exp(A). \tag{2.15}$$

3. Let $A \in \mathscr{M}_n$ and $B \in \mathscr{M}_m$. For any identity matrix I_k:
$$\exp(A \otimes I_k \otimes B) = S_{km,n}(I_k \otimes \exp(B \otimes A))S_{n,km} = S_{m,nk}(\exp(B \otimes A) \otimes I_k)S_{nk,m}. \tag{2.16}$$

Proof. Cf. Appendix A. □

2.1.3. Supplement

Remark. Consider the canonical basis of the $\mathbb{R}^N$ space, $N \in \mathbb{N}^+$, i.e. the set of N orthonormal vectors, each of size N:

$$\left\{\mathbf{e}^{(1)}, \mathbf{e}^{(2)}, \dots, \mathbf{e}^{(N)}\right\},$$

such that: $\mathbf{e}^{(i)}$ $(1 \le i \le N)$ is composed of $N-1$ zeros and a single one placed on the i-th position of this vector:

$$\mathbf{e}_N^{(i)} = (\underbrace{0, \dots, 0}_{(i-1)}, 1, \underbrace{0, \dots, 0}_{(N-i)})^1.$$

Taking advantage from the Remark above, it is possible to introduce the Perfect Shuffle permutation by the Definition 5. Then, an inversion of the Perfect Shuffle permutation matrix is presented by the Theorem 7.

Definition 5. [14, 23] Consider two numbers $n, m \in \mathbb{N}^+$ and a set of nm consecutive natural numbers starting from one $\mathscr{X} = \{1, 2, \dots, nm\}$. Naturally it is possible to express any element $x \in \mathscr{X}$ unambiguously employing any of the following forms: $x = (k-1)n + l$ or $x = (l-1)m + k$, where: $1 \le k \le m$, $1 \le l \le n$ and $k, l \in \mathbb{N}^+$. Please note that the former approach corresponds to a partition of

¹Please note that all vectors within this document are assumed to be row ones, while their transpositions – denoted by $\mathbf{v}^{\mathrm{T}}$ for any vector $\mathbf{v}$ – are arranged as columns.

the set $\mathscr{X}$ into m blocks, each of size n, while the latter one conforms with a splitting of the set $\mathscr{X}$ into n blocks, each of size m.

The *Perfect Shuffle permutation* is a bijective mapping $\sigma_{m,n} : \mathscr{X} \to \mathscr{X}$ such that:

$$\forall_{x\in\mathscr{X}} : x = (k-1)n + l \Rightarrow \sigma_{m,n}(x) = (l-1)m + k.$$

An algebraic form of the permutation $\sigma_{m,n}$ is given by the matrix $(S_{m,n})_{mn\times mn} = [s_{ij}]$ such that:

$$s_{ij} = \begin{cases} 1, & \text{if } \sigma_{m,n}(j) = i \\ 0, & \text{otherwise} \end{cases}.$$

Equivalently, employing the aforementioned canonical basis of size either m or n, one may define respectively:

$$S_{m,n} \overset{\text{def}}{=} \sum_{k=1}^{m} \mathbf{e}^{(k)} \otimes I_n \otimes \left(\mathbf{e}^{(k)}\right)^{\mathsf{T}} = \sum_{k=1}^{n} \left(\mathbf{e}^{(k)}\right)^{\mathsf{T}} \otimes I_m \otimes \mathbf{e}^{(k)}.$$

Remark. Informally speaking, the matrix $S_{m,n}$ is composed of rearranged columns of the identity matrix I_{mn}. Let $1 \le i,j \le mn$ and $i,j \in \mathbb{N}^+$. Then, if $\sigma_{m,n}(i) = j$, the i-th column of the $S_{m,n}$ matrix is the j-th column of the I_{mn} matrix. This observation is a direct consequence derived from the explicit form of elements s_{ij} given by the Definition 5.

Theorem 7. [14, 23] Let $m,n \in \mathbb{N}^+$ and $\sigma_{m,n}, \sigma_{n,m}$ be Perfect Shuffle permutations acting on a set $\mathscr{X}$. Then, the $\sigma_{n,m}$ permutation is an inverse of the $\sigma_{m,n}$ permutation, in other words:

$$\sigma_{n,m}\left(\sigma_{m,n}\left(\mathscr{X}\right)\right) = \mathscr{X}.$$

Let also $S_{m,n}, S_{n,m}$ be matrices of the, respectively, $\sigma_{m,n}, \sigma_{n,m}$ Perfect Shuffle permutations. In consequence, the $S_{n,m}$ matrix is an inverse of the $S_{m,n}$ matrix likewise:

$$S_{m,n} = S_{n,m}^{-1}.$$

Proof. Cf. Section II.C in [14] and Section 4 in [23]. □

Finally, some essential concepts regarding the evaluation of results accuracy are presented. The $\mathscr{L}_p$ norm – used extensively throughout this thesis – acting on vectors is introduced by the Definition 6 and followed by brief remarks on the norms of matrices.

Definition 6. [24] Let $p \ge 1$ and $p \in \mathbb{R}$. The $\mathscr{L}_p$ *norm* of a vector $\mathbf{v}_n = \left(v^{(1)}, v^{(2)}, \ldots, v^{(n)}\right)$ is given by:

$$\|\mathbf{v}\|_p \overset{\text{def}}{=} \left(\sum_{k=1}^{n} \left|v^{(k)}\right|^p\right)^{\frac{1}{p}},$$

where: $\|\cdot\|_p$ denotes the $\mathscr{L}_p$-norm, while $|\cdot|$ stands for the absolute value operator acting on scalars.

Additionally, the $\mathscr{L}_\infty$ norm of the vector $\mathbf{v}$ is defined as follows:

$$\|\mathbf{v}\|_\infty \overset{\text{def}}{=} \max_{1\le k\le n} \left|v^{(k)}\right|.$$

Remark. [24] Although not all vector norms applied to elements of the vector space defined over $\mathscr{M}_n$ are matrix norms, all the $\mathscr{L}_p$ norms are. Therefore, the $\mathscr{L}_p$ norm applied to a vectorized matrix $A_{n\times n} = [a_{ij}]$ (in fact, to a vector of n^2 elements) is given within this document by:

$$\|A\|_p \stackrel{\text{def}}{=} \left(\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}|^p\right)^{\frac{1}{p}},$$

with the special case for $\mathscr{L}_\infty$:

$$\|A\|_\infty \stackrel{\text{def}}{=} \max_{1\le i,j\le n} |a_{ij}|\,.$$

2.2. Stochastic Processes

Within this document a discrete random variable indexed by a real non-negative time parameter $t \in \mathbb{R}^{0^+}$ is denoted by a lowercase Greek letter with a subscript, such as χ_t. Then, a probability that a discrete random variable χ_t takes the value k is given by $\Pr(\chi_t = k)$.

2.2.1. Continuous-Time Stochastic Processes

Among numerous types of stochastic processes, introduced by the Definition 7, CTMCs (described by the Definition 8) are special ones. These Markovian models satisfy a, so called, memorylessness property, which states that a probability of switching to any of its states is dependent only on a currently occupied one and not on a history of transitions at all. Additionally, all CTMCs considered within this research are time-homogeneous, what is clarified by the Definition 9.

Definition 7. [6] The *stochastic process* is given by a set $\left\{\chi_t : t \in \mathbb{R}^{0^+}\right\}$ of random variables, each of which taking values from a set $\mathscr{S} = \left\{s^{(1)}, s^{(2)}, \dots, s^{(N)}\right\}$ called the *state space*.

Definition 8. [6] Consider a stochastic process $\mathscr{X} = \left\{\chi_t : t \in \mathbb{R}^{0^+}\right\}$ over a state space $\mathscr{S}$. Then, for any value $k \in \mathbb{N}$, strictly increasing times indexed up to these values $t_0 < t_1 < t_2 < \ldots \in \mathbb{R}^{0^+}$ and all states indexed at these times $s_0, s_1, s_2, \ldots \in \mathscr{S}$, the stochastic process $\mathscr{X}$ constitutes the *Continuous-Time Markov Chain* if it satisfies the following *Markov property*:

$$\Pr\left(\chi_{t_{k+1}} = s_{k+1} \mid \chi_{t_k} = s_k, \chi_{t_{k-1}} = s_{k-1}, \dots, \chi_{t_0} = s_0\right) = \Pr\left(\chi_{t_{k+1}} = s_{k+1} \mid \chi_{t_k} = s_k\right),$$

where: $\Pr(\cdot \mid \cdot)$ denotes the conditional probability.

Definition 9. [6] A CTMC $\mathscr{X} = \left\{\chi_t : t \in \mathbb{R}^{0^+}\right\}$ over a state space $\mathscr{S}$ is said to be *time-homogeneous* if its conditional transition probability is invariant with respect to time, i.e.:

$$\Pr\left(\chi_t = s \mid \chi_{t'} = s'\right) = \Pr\left(\chi_{t-t'} = s \mid \chi_0 = s'\right),$$

where: $t' \le t \in \mathbb{R}^{0+}$ and $s, s' \in \mathscr{S}$.

Now it is possible to define the evolution of a CTMC over time. Firstly, the Kolmogorov equations describing how the probability distribution of states belonging to a Markovian model changes over time are introduced by the Theorem 8. Then, a note on the solution of the Kolmogorov backward equation is given by the Theorem 9 and followed by a brief remark on computing the probability distribution of states over time.

Remark. [6] Let $p_{s_i,s_j}\left(t_k,t_l\right)$ denote the probability of departing from the state s_i and arriving to the state s_j during the period of time $[t_k,t_l)$ (left-closed and right-open interval) for a CTMC $\mathscr{X}$ defined over a state space $\mathscr{S}$. Naturally, $s_i,s_j\in\mathscr{S}$, $t_k,t_l\in\mathbb{R}^{0+}$ and $t_k\leq t_l$. If $t_k=t_l$, then:

$$p_{s_i,s_j}(t_k,t_k)=\begin{cases}1, & \text{if } s_i=s_j\\ 0, & \text{otherwise}\end{cases}.$$

Please note that for time-homogeneous Markov chains it is sufficient to consider only the case $p_{s_i,s_j}(0,t)\equiv p_{s_i,s_j}(t)$.

Remark. [6] In practice, all available transitions between states of any time-homogeneous CTMC $\mathscr{X}$ defined over a state space $\mathscr{S}=\left\{s^{(1)},s^{(2)},\ldots,s^{(N)}\right\}$ are described by the infinitesimal generator matrix (cf. Definition 12) $Q_{N\times N}=[q_{ij}]$, such that each of its elements q_{ij} indicates rate of the transition from state $s^{(i)}\in\mathscr{S}$ to another state $s^{(j)}\in\mathscr{S}$. The elements q_{ij} are then given by:

$$q_{ij}=\lim_{\Delta t\to 0}\frac{p_{s^{(i)},s^{(j)}}(\Delta t)-p_{s^{(i)},s^{(j)}}(0)}{\Delta t}.$$

Theorem 8. [6] Let $t\in\mathbb{R}^{0+}$ denote arbitrary simulation time of a time-homogeneous CTMC $\mathscr{X}=\left\{\chi_t : t\in\mathbb{R}^{0+}\right\}$ over a state space $\mathscr{S}=\left\{s^{(1)},s^{(2)},\ldots,s^{(N)}\right\}$ and associated with an infinitesimal generator matrix $Q\in\mathscr{M}_N$. Consider any pair of states $\left(s^{(i)},s^{(j)}\right)\in\mathscr{S}^2$ and the probability that the model $\mathscr{X}$ occupies the state $s^{(j)}$ at time t if its initial state was $s^{(j)}$:

$$p_{s^{(i)},s^{(j)}}(t)=\Pr\left(\chi_t=s^{(j)}\mid\chi_0=s^{(i)}\right).$$

Then, regarding all possible pairs of states $\left(s^{(i)},s^{(j)}\right)$, the probabilities $p_{s^{(i)},s^{(j)}}(t)$ form a stochastic matrix $P(t)_{N\times N}=[p_{ij}]$ with entries $p_{ij}=p_{s^{(i)},s^{(j)}}(t)$.

The aforementioned transition probabilities satisfy both the *Kolmogorov forward equation*:

$$\frac{d}{dt}P(t)=P(t)Q,$$

and the *Kolmogorov backward equation*:

$$\frac{d}{dt}P(t)=QP(t).$$

Proof. Cf. Theorem 3.1 in [48]. □

Theorem 9. [48] The unique solution of both the Kolmogorov equations given by the Theorem 8 with an initial condition $P(0)=I_N$ is:

$$P(t)=\exp(tQ).$$

Proof. Cf. Theorem 3.2 in [48]. □

Remark. [6] Let $\mathbf{p}(t)$ be a probability distribution of states belonging to a time-homogeneous CTMC $\mathscr{X}$ over time. Assuming $\mathbf{p}(0)$ is the initial probability distribution, the probability distribution of states of $\mathscr{X}$ at any instant of time t is given by:

$$\mathbf{p}(t) = \mathbf{p}(0)P(t). \tag{2.17}$$

Naturally, for $\mathscr{X}$ defined over a state space $\mathscr{S} = \{s^{(1)}, s^{(2)}, \ldots, s^{(N)}\}$, an i-th element $p^{(i)}(t)$ of the distribution $\mathbf{p}_N(t) = \big(p^{(1)}(t), p^{(2)}(t), \ldots, p^{(N)}(t)\big)$ stands for the probability of being the CTMC $\mathscr{X}$ in the state $s^{(i)}$ at time t.

2.2.2. Supplement

Any (discrete) probability distribution of states considered within this thesis is represented by a probability vector introduced by the Definition 10. Additionally, the evolution of both DTMCs and CTMCs in time may be expressed in terms of matrices. In case of a DTMC, the process of switching between its states is described by the probability matrix introduced by the Definition 11, while this mechanism for a CTMC is given by the transition rate matrix presented by the Definition 12.

Definition 10. [6] The *probability vector* $\mathbf{p}_n = \big(p^{(1)}, p^{(2)}, \ldots, p^{(n)}\big)$ satisfies the following properties:

1. Each of its elements is a probability:
$$\forall_{1 \le k \le n} : 0 \le p^{(k)} \le 1.$$
2. It completely describes the probability mass function of a discrete random variable:
$$\sum_{k=1}^{n} p^{(k)} = 1.$$

Definition 11. [6] $P \in \mathscr{M}_n$ is the *probability* (also called *stochastic* or *transition*) *matrix* if its rows are probability vectors.

Definition 12. [6] Matrix $Q_{n\times n} = [q_{ij}]$ is the *transition rate* (also called *intensity* or *infinitesimal generator*) *matrix* if satisfies the following properties:

1. Its non-diagonal elements are non-negative:
$$\forall_{\substack{1 \le i,j \le k \\ i \ne j}} : q_{ij} \in \mathbb{R}^{0^+}.$$
2. Its diagonal elements make its rows sum to zero:
$$\forall_{1 \le i \le n} : \sum_{j=1}^{n} q_{ij} = 0.$$

2.3. Summary

In this Chapter a collection of mathematical definitions and theorems falling outside the syllabus of introductory university courses was presented. Initially, fundamental algebraic concepts regarding the Kronecker algebra and its properties, as well as the matrix exponential function and the laws of exponent were given. Then, the Stochastic and Markovian Processes were introduced along with accompanying Kolmogorov equations describing the time evolution of such models. This formalism is the foundation for further considerations in the matter of stochastic automata (cf. Chapter 3), tensor structures (cf. Chapter 4) and their hybridization (cf. Chapter 7).

3. Introduction to Stochastic Automata Networks

This Chapter contains a formal introduction to stochastic automata (Section 3.1) and SANs (Section 3.2) composed of both non-interacting and synchronized components. Please note that the following description is adapted to the scope of this research and should not be considered as an exhaustive study in the field of the eponymous formalism. The comprehensive introduction to SANs is presented in the suggested literature [35, 39, 44].

3.1. Stochastic Automata

The stochastic automaton – a fundamental building block of the SAN formalism, which constitutes a high-level approach to describe complex Markovian models – is introduced by the Definition 13, which is followed by several remarks regarding multiple representations of discussed automata.

Definition 13. [35, 39] The *Continuous-Time Stochastic Automaton* $\mathcal{A}$ is a triple: $(\mathscr{S}_\mathcal{A}, \mathscr{L}_\mathcal{A}, f_\mathcal{A})$, where: $\mathscr{S}_\mathcal{A} = \left\{s^{(1)}, s^{(2)}, \ldots, s^{(S_\mathcal{A})}\right\}$ is the finite state space of size $S_\mathcal{A}$, $\mathscr{L}_\mathcal{A} = \left\{l^{(1)}, l^{(2)}, \ldots, l^{(L_\mathcal{A})}\right\}$ is the finite set of labels of size $L_\mathcal{A}$ and $f_\mathcal{A} : \mathscr{S}_\mathcal{A}^2 \to \mathscr{L}_\mathcal{A}$ is the state-transition function, which defines a stochastic nature of shifting between any two distinct states $\left(s^{(i)}, s^{(j)}\right) \in \mathscr{S}^2$.

The transition of a stochastic automaton is an act of switching between its states. Within this document, the transition from a state $s^{(i)} \in \mathscr{S}_\mathcal{A}$ (called origin) to another state $s^{(j)} \in \mathscr{S}_\mathcal{A}$ (called destination) is denoted by an ordered pair $\left(s^{(i)}, s^{(j)}\right)$. The evolution of an automaton is a process of changing its states in time, according to the state-transition function $f_\mathcal{A}$, which – for any transition – returns a set of labels associated with it. Thus, each transition may be related to single label, multiple labels or no labels (in particular, the empty set means that a corresponding transition must not occur).

Each *label* is a pair $\left(e, p^e_{s^{(i)}}\right)$, where: e is the unique identifier of a particular event (cf. Definition 15) and $p^e_{s^{(i)}} : \mathscr{S} \to [0, 1]$ is the alternative probability function describing the chance of choosing a state $s^{(j)} \in \mathscr{S}_\mathcal{A}$ as a destination of the transition originating from the state $s^{(i)} \in \mathscr{S}_\mathcal{A}$ when the event e occurs. Formally, let $\mathscr{T}^e_\mathcal{A}$ be a set containing all transitions taking place within the automaton $\mathcal{A}$ as a result of the event e firing: $\mathscr{T}^e_\mathcal{A} = \left\{\left(s^{(i)}, s^{(j)}\right) \in \mathscr{S}^2_\mathcal{A} : \left(e, p^e_{s^{(i)}}\right) \in f_\mathcal{A}\left(s^{(i)}, s^{(j)}\right)\right\}$ and $\mathscr{S}^e_\mathcal{A}\left(s^{(i)}\right)$ be a set containing all destination states (also called successors) of the transitions originating from the state $s^{(i)} \in \mathscr{S}_\mathcal{A}$ and associated with the event e: $\mathscr{S}^e_{\mathcal{A}}\left(s^{(i)}\right) = \left\{s^{(j)} \in \mathscr{S}_{\mathcal{A}} : \left(s^{(i)}, s^{(j)}\right) \in \mathscr{T}^e_{\mathcal{A}}\right\}$. Then, for each event e:

$$\forall_{s^{(i)} \in \mathscr{S}_{\mathcal{A}}} : \sum_{s^{(j)} \in \mathscr{S}^e_{\mathcal{A}}\left(s^{(i)}\right)} p^e_{s^{(i)}}\left(s^{(j)}\right) = 1.$$

Remark. [39] Structurally, a stochastic automaton $\mathcal{A} = (\mathscr{S}_{\mathcal{A}}, \mathscr{L}_{\mathcal{A}}, f_{\mathcal{A}})$ may be represented by a directed graph with vertices defined by the state space $\mathscr{S}_{\mathcal{A}}$ and edges given for any pair or two distinct states $(s, s') \in \mathscr{S}^2$ as follows:

$$(s, s') = \begin{cases} 1, & \text{if } f_{\mathcal{A}}(s, s') \neq \emptyset \\ 0, & \text{otherwise} \end{cases},$$

where: $(s, s') = 1$ means that the directed edge from the vertex represented by s to the vertex reflected by s' exists, while $(s, s') = 0$ means that this edge does not exist and $\emptyset$ denotes the empty set. Moreover, each edge is equipped with one or more labels defining all possible events, which may cause a transition along this edge. Naturally, if an edge is provided with multiple events, they need to have pairwise distinct identifiers.

Remark. Equivalently, one may interpret a stochastic automaton as an event-driven finite state machine, which evolve in time by switching between its states. Each transition in this state machine is triggered in response to a stochastic occurrence of an event.

Remark. [36] Taking the stochastic processes point of view into consideration, each stochastic automaton $\mathcal{A} = (\mathscr{S}_{\mathcal{A}}, \mathscr{L}_{\mathcal{A}}, f_{\mathcal{A}})$ satisfies the memorylessness Markov property and is underlaid by a time-homogeneous CTMC over the state space $\mathscr{S}_{\mathcal{A}}$. Therefore, the state of the automaton $\mathcal{A}$ at arbitrary time t is inherently a discrete random variable with associated probability distribution of states $\mathbf{p}_{\mathcal{A}}(t)$.

3.2. Stochastic Automata Networks

The expression power of single stochastic automaton is not sufficient to model real distributed systems, therefore multiple automata are tied together in a network of cooperating components. Such structures are called SANs and are introduced formally by the Definition 14, supplied by the Definition 15 presenting the events, which govern the time evolution of discussed models. Then, some supplementary comments concerning various representations of SANs and notation remarks are provided.

Definition 14. [35, 39] The (Continuous-Time) *Stochastic Automata Network* $\boldsymbol{\mathcal{N}}$ is a tuple: $(\mathscr{A}_{\mathcal{N}}, \mathscr{E}_{\mathcal{N}})$, where: $\mathscr{A}_{\mathcal{N}} = \left\{\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(N_{\mathcal{N}})}\right\}$ is the set of $N_{\mathcal{N}}$ cooperating stochastic automata building up the network and $\mathscr{E}_{\mathcal{N}} = \left\{e^{(1)}, e^{(2)}, \ldots, e^{(E_{\mathcal{N}})}\right\}$ is the set of $E_{\mathcal{N}}$ events (cf. Definition 15) triggering the transitions of the constituent machines.

Definition 15. [39] Let $\boldsymbol{\mathcal{N}} = (\mathscr{A}_{\mathcal{N}}, \mathscr{E}_{\mathcal{N}})$ be a SAN. Define $t_{\mathcal{N}} : \mathscr{E}_{\mathcal{N}} \to \{loc, syn\}$ to be the function returning a type of an event, $m_{\mathcal{N}} : \mathscr{E}_{\mathcal{N}} \to \mathscr{A}_{\mathcal{N}}$ be the function returning a master automaton of an event and $q_{\boldsymbol{\mathcal{N}}} : \mathscr{E}_{\boldsymbol{\mathcal{N}}} \rightarrow \mathbb{R}^{0^+}$ be the function returning a rate of an event. Then, each uniquely identified *event* $e \in \mathscr{E}$ is described by a triple $(t_{\boldsymbol{\mathcal{N}}}(e), m_{\boldsymbol{\mathcal{N}}}(e), q_{\boldsymbol{\mathcal{N}}}(e))$.

Each event e considered within the SAN formalism is either local (cf. Definition 16) – then: $t_{\boldsymbol{\mathcal{N}}}(e) = loc$ – or synchronizing (cf. Definition 17) – then: $t_{\boldsymbol{\mathcal{N}}}(e) = syn$. The master of the event e is an automaton, which generates this event during the time evolution of the model. Finally, the rate of the event e may be interpreted as an expected number of occurrences of the event e within a simulation time unit.

Remark. [35, 39] Despite the fact that any SAN $\boldsymbol{\mathcal{N}}$ with $\mathscr{A}_{\boldsymbol{\mathcal{N}}} = \left\{\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \dots, \mathcal{A}^{(N_{\boldsymbol{\mathcal{N}}})}\right\}$ is defined as a collection of individual components, one can represent it as a single, global stochastic automaton $\mathcal{N} = (\mathscr{S}_{\mathcal{N}}, \mathscr{L}_{\mathcal{N}}, f_{\mathcal{N}})$.

The state space $\mathscr{S}_{\mathcal{N}}$ – further referenced as the *Product State Space* – is a Cartesian product of state spaces belonging to all constituent automata: $\mathscr{S}_{\mathcal{N}} = \mathscr{S}_{\mathcal{A}^{(1)}} \times \mathscr{S}_{\mathcal{A}^{(2)}} \times \ldots \times \mathscr{S}_{\mathcal{A}^{(N_{\boldsymbol{\mathcal{N}}})}}$, therefore the global state $\mathbf{s} \in \mathscr{S}_{\mathcal{N}}$ is a vector $\mathbf{s} = \left(s^{(1)}, s^{(2)}, \dots, s^{(N_{\boldsymbol{\mathcal{N}}})}\right)$ composed of states belonging to respective automata: $\forall_{1 \leq i \leq N_{\boldsymbol{\mathcal{N}}}} : s^{(i)} \in \mathscr{S}_{\mathcal{A}^{(i)}}$. Furthermore, labels $\mathscr{L}_{\mathcal{N}}$ are given over the set $\mathscr{E}_{\boldsymbol{\mathcal{N}}}$ of both local events belonging to particular components and synchronizing events describing the cooperation between them.

Because of the PSS definition, imposing no restrictions on the validity of global states, it is possible that $\mathscr{S}_{\mathcal{N}}$ involves some illegal combinations of local states, which never occur in the model under simulation (such states are called unreachable). Therefore, the *reachability function* $r_{\boldsymbol{\mathcal{N}}} : \mathscr{S}_{\mathcal{N}} \rightarrow \{0, 1\}$ describes whether any global state $\mathbf{s} \in \mathscr{S}_{\mathcal{N}}$ is available within the SAN $\boldsymbol{\mathcal{N}}$ (and returns 1 then) or not (and returns 0 then). Thus, the *Reachable State Space* $\mathscr{R}_{\mathcal{N}}$ is given by: $\mathscr{R}_{\mathcal{N}} = \{\mathbf{s} \in \mathscr{S}_{\mathcal{N}} : r_{\boldsymbol{\mathcal{N}}}(\mathbf{s}) = 1\}$. Naturally, the size $R_{\mathcal{N}}$ of the RSS is not greater than the size $S_{\mathcal{N}} = \prod_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} S_{\mathcal{A}^{(k)}}$ of the PSS (in fact, it is usually much more smaller).

Finally, please note a minor but significant notation difference between a SAN and its corresponding stochastic automaton – the former one is represented by the same typeface and Roman letter as the latter one, but additionally is enriched with bold.

Remark. Similarly to the conclusion drawn for single stochastic automaton, one may consider a SAN $\boldsymbol{\mathcal{N}} = (\mathscr{A}_{\boldsymbol{\mathcal{N}}}, \mathscr{E}_{\boldsymbol{\mathcal{N}}})$ as a CTMC over the PSS of the network $\boldsymbol{\mathcal{N}}$. Then, the state of the SAN $\boldsymbol{\mathcal{N}}$ at arbitrary time t is naturally a system of N random variables, each of which corresponding to exactly one constituent automaton, associated with a multivariative joint probability distribution of states $\mathbf{p}_{\mathcal{N}}(t)$. Moreover, a probability distribution of states belonging to a particular stochastic automaton $\mathcal{A} \in \mathscr{A}_{\boldsymbol{\mathcal{N}}}$ is given by the marginal distribution derived from the distribution $\mathbf{p}_{\mathcal{N}}(t)$ with respect to the i-th random variable.

Remark. [39] Consider a SAN $\boldsymbol{\mathcal{N}} = (\mathscr{A}_{\boldsymbol{\mathcal{N}}}, \mathscr{E}_{\boldsymbol{\mathcal{N}}})$ and an event $e \in \mathscr{E}_{\boldsymbol{\mathcal{N}}}$. Denote $\mathscr{O}^{e}_{\boldsymbol{\mathcal{N}}}$ to be the set of all automata affected by the event e: $\mathscr{O}^{e}_{\boldsymbol{\mathcal{N}}} = \{\mathcal{A} \in \mathscr{A}_{\boldsymbol{\mathcal{N}}} : \mathscr{T}^{e}_{\mathcal{A}} \neq \emptyset\}$.

Additionally, let $\mathscr{E}^{loc}_{\boldsymbol{\mathcal{N}}}$ be a set of local events defined within $\boldsymbol{\mathcal{N}}$: $\mathscr{E}^{loc}_{\boldsymbol{\mathcal{N}}} = \{e \in \mathscr{E}_{\boldsymbol{\mathcal{N}}} : t_{\boldsymbol{\mathcal{N}}}(e) = loc\}$ and $\mathscr{E}^{syn}_{\boldsymbol{\mathcal{N}}}$, accordingly, be a set of synchronizing events: $\mathscr{E}^{syn}_{\boldsymbol{\mathcal{N}}} = \{e \in \mathscr{E}_{\boldsymbol{\mathcal{N}}} : t_{\boldsymbol{\mathcal{N}}}(e) = syn\}$. Recalling the

Definition 15 of the function $t_{\mathcal{N}}$, it obviously follows that the union of these two disjoint sets contains all defined events: $\mathscr{E}_{\mathcal{N}} = \mathscr{E}_{\mathcal{N}}^{loc} \cup \mathscr{E}_{\mathcal{N}}^{syn}$.

3.2.1. Non-interacting automata

The most straightforward SAN models are the groups of non-interacting automata, which do not communicate each other, but semantically are concerned as a collection of components working towards achieving a common goal. Within such models, the constituents are allowed to perform only local transitions, which are introduced by the Definition 16.

Definition 16. [39] Consider a SAN $\mathcal{N} = (\mathscr{A}_{\mathcal{N}}, \mathscr{E}_{\mathcal{N}})$ with. An event $e \in \mathscr{E}_{\mathcal{N}}$ triggers the *local transition* if and only if $\mathscr{O}_{\mathcal{N}}^{e}$ is a singleton (i.e. one-element) set. Thus, the local transition caused by an occurrence of the event e affects only the state of the automaton $\mathcal{A} \in \mathscr{A}_{\mathcal{N}}$, such that: $m_{\mathcal{N}}(e) = \mathcal{A}$.

The time evolution of all CTMCs, and hence stochastic automata and their networks too, may be represented by an infinitesimal generator matrix under the assumption that a time interval between firing consecutive events is an exponentially distributed random variable (which is the case within this research). Moreover, the SAN formalism employs a handy representation of the global transition rate matrix presented below. Firstly, consider the trivial case, when a network is composed of automata without synchronization.

Remark. [36, 39] Let $\mathcal{N} = (\mathscr{A}_{\mathcal{N}}, \mathscr{E}_{\mathcal{N}})$ with $\mathscr{A}_{\mathcal{N}} = \left\{\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(N_{\mathcal{N}})}\right\}$ be a SAN composed of $N_{\mathcal{N}}$ automata without synchronization, i.e. $\forall_{e \in \mathscr{E}_{\mathcal{N}}} : t_{\mathcal{N}}(e) = loc$ (and therefore: $\mathscr{E}_{\mathcal{N}} = \mathscr{E}_{\mathcal{N}}^{loc}$). Moreover, let each automaton $\mathcal{A}^{(k)} \in \mathscr{A}_{\mathcal{N}}$ be associated with its infinitesimal generator matrix $Q_{\mathcal{A}^{(k)}}^{loc} \in \mathscr{M}_{S_{\mathcal{A}^{(k)}}}$. Then, the global infinitesimal generator matrix $Q_{\mathcal{N}} \in \mathscr{M}_{S_{\mathcal{N}}}$ of the network $\mathcal{N}$ is given by the Kronecker sum of transition rate matrices associated with each automaton:

$$Q_{\mathcal{N}} = \bigoplus_{k=1}^{N_{\mathcal{N}}} Q_{\mathcal{A}^{(k)}}^{loc}. \tag{3.1}$$

Each of the matrices $\left(Q_{\mathcal{A}^{(k)}}^{loc}\right)_{S_{\mathcal{A}^{(k)}} \times S_{\mathcal{A}^{(k)}}} = [q_{ij}]$ is given by:

$$q_{ij} = \sum_{\left(e, p_{s^{(i)}}^{e}\right) \in f\left(s^{(i)}, s^{(j)}\right)} q_{\mathcal{N}}(e) \cdot p_{s^{(i)}}^{e}\left(s^{(j)}\right),$$

for $s^{(i)}, s^{(j)} \in \mathscr{S}_{\mathcal{A}^{(k)}} = \left\{s^{(1)}, s^{(2)}, \ldots, s^{\left(S_{\mathcal{A}^{(k)}}\right)}\right\}$.

When the automata do not interact, random variables reflecting states of the constituents at arbitrary time t are statistically independent. Therefore, the global probability distribution $\mathbf{p}_{\mathcal{N}}(t)$ of states belonging to the SAN $\mathcal{N}$ is simply the Kronecker product of corresponding probability distributions belonging to the components building up the network under simulation:

$$\mathbf{p}_{\mathcal{N}}(t) = \bigotimes_{k=1}^{N} \mathbf{p}_{\mathcal{A}^{(k)}}(t), \tag{3.2}$$

where: $\mathbf{p}_{\mathcal{A}^{(k)}}(t)$ is the probability distribution of states associated with the automaton $A^{(k)}$ at arbitrary time t.

Please note that the form of a global probability distribution of states (imposed by the Equation 3.2) significantly facilitates calculating transient probability distributions for SANs without synchronization – it is sufficient to obtain desired solution for each of the components separately. Although this representations is really helpful, it is almost never used in practice, because the models of non-interacting components are too simplistic to handle real problems.

Remark. [36] Throughout this thesis it is assumed that the elements of any infinitesimal generator matrix are constant real numbers. Nevertheless it is possible for these elements to be functions of states belonging to the automata composing a network under simulation. Then, the Ordinary Tensor Algebra (i.e. the Kronecker algebra) formalism is no longer valid and the Generalized Tensor Algebra [18] has to be employed, which, however, is not considered within this research, because its properties are not sufficient enough to be applicable to the TNSAN algorithm requirements (cf. Chapter 7 for the insightful justification). Therefore, the Generalized Tensor Algebra and, so called, functional transitions are not considered within this research.

3.2.2. Interacting automata

The presence of synchronized transitions, introduced by the Definition 17, significantly enhances the modelling power of the SAN formalism. By employing the synchronization mechanism, constituent automata are able to communicate each other, and therefore it is possible to express practically every possible scenario of cooperation between them.

Definition 17. [39] Consider a SAN $\mathcal{N} = (\mathscr{A}_{\mathcal{N}}, \mathscr{E}_{\mathcal{N}})$. An event $e \in \mathscr{E}_{\mathcal{N}}$ triggers the *synchronized transition* if and only if $\mathscr{O}^{e}_{\mathcal{N}}$ contains at least two elements. Thus, the synchronized transition caused by an occurrence of the event e affects not only the state of the master automaton of e, but additionally triggers a transition in at least one other automaton constituting the network $\mathcal{N}$.

Following the considerations from the Section 3.2.1 regarding a structured representation of the infinitesimal generator matrix describing the time evolution of any SAN, examine the case when the constituent automata interacts by incorporating synchronizing events.

Remark. [36, 39] Let $\mathcal{N} = (\mathscr{A}_{\mathcal{N}}, \mathscr{E}_{\mathcal{N}})$ with $\mathscr{A}_{\mathcal{N}} = \left\{\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(N_{\mathcal{N}})}\right\}$ be a SAN composed of $N_{\mathcal{N}}$ automata. For simplicity, assume that there are no local events defined within $\mathcal{N}$, i.e. $\forall_{e \in \mathscr{E}_{\mathcal{N}}}$: $t_{\mathcal{N}}(e) = syn$ (and therefore: $\mathscr{E}_{\mathcal{N}} = \mathscr{E}^{syn}_{\mathcal{N}}$). Then, the global infinitesimal generator matrix $Q_{\mathcal{N}} \in \mathscr{M}_{S_{\mathcal{N}}}$ of the network $\mathcal{N}$ is given by the sum of $2 \cdot E_{\mathcal{N}}$ Kronecker products, each of which composed of $N_{\mathcal{N}}$ matrices (one per each automaton $\mathcal{A}^{(k)} \in \mathscr{A}_{\mathcal{N}}$):

$$Q_{\mathcal{N}} = \sum_{e \in \mathscr{E}_{\mathcal{N}}} \left[\bigotimes_{k=1}^{N_{\mathcal{N}}} Q^{e_{pos}}_{\mathcal{A}^{(k)}} + \bigotimes_{k=1}^{N_{\mathcal{N}}} Q^{e_{neg}}_{\mathcal{A}^{(k)}} \right]. \tag{3.3}$$

Consider an automaton $\mathcal{A}^{(k)} \in \mathscr{A}_{\mathcal{N}}$ and a synchronizing event $e \in \mathscr{E}_{\mathcal{N}}$. If the automaton $\mathcal{A}^{(k)}$ is affected by the event e (i.e. $\mathcal{A}^{(k)} \in \mathscr{O}^{e}_{\mathcal{N}}$), the *positive* matrix $\left(Q^{e_{pos}}_{\mathcal{A}^{(k)}}\right)_{S_{\mathcal{A}^{(k)}} \times S_{\mathcal{A}^{(k)}}} = \left[q^{pos}_{ij}\right]$ associated with the event e and corresponding to the automaton $\mathcal{A}^{(k)}$ is given by:

$$q^{pos}_{ij} = \begin{cases} q_{\mathcal{N}}(e) \cdot p^{e}_{s^{(i)}}\left(s^{(j)}\right) & \text{if } m_{\mathcal{N}}(e) = \mathcal{A}^{(k)} \\ p^{e}_{s^{(i)}}\left(s^{(j)}\right) & \text{otherwise} \end{cases},$$

for the label $\left(e, p^{e}_{s^{(i)}}\right) \in \mathscr{L}_{\mathcal{A}^{(k)}}$ associated with the synchronized transition within the automaton $\mathcal{A}^{(k)}$ triggered by the event e, and $s^{(i)}, s^{(j)} \in \mathscr{S}_{\mathcal{A}^{(k)}} = \left\{s^{(1)}, s^{(2)}, \ldots, s^{\left(S_{\mathcal{A}^{(k)}}\right)}\right\}$. Otherwise, when $\mathcal{A}^{(k)} \notin \mathscr{O}^{e}_{\mathcal{N}}$, the discussed matrix is the identity matrix:

$$Q^{e_{pos}}_{\mathcal{A}^{(k)}} = I_{S_{\mathcal{A}^{(k)}}}.$$

Taking into assumption all the aforementioned prerequisites, the corresponding *negative* (also called *diagonal corrector*) matrix $\left(Q^{e_{neg}}_{\mathcal{A}^{(k)}}\right)_{S_{\mathcal{A}^{(k)}} \times S_{\mathcal{A}^{(k)}}} = \left[q^{neg}_{ij}\right]$ associated with the event e and corresponding to the automaton $\mathcal{A}^{(k)}$ reduces the sum of elements in each row of the matrix $Q^{e_{pos}}_{\mathcal{A}^{(k)}}$ to zero, thus it is defined as:

$$q^{neg}_{ij} = \begin{cases} -\sum_{l=1}^{S_{\mathcal{A}^{(k)}}} q^{pos}_{il} & \text{if } i = j \text{ and } m_{\mathcal{N}}(e) = \mathcal{A}^{(k)} \\ \sum_{l=1}^{S_{\mathcal{A}^{(k)}}} q^{pos}_{il} & \text{if } i = j \text{ and } m_{\mathcal{N}}(e) \neq \mathcal{A}^{(k)} \\ 0 & \text{otherwise} \end{cases}.$$

Finally, please note that the solution form pointed out by the Equation 3.2 is no longer valid when synchronizing events are incorporated into the simulated model. Moreover, there is no general formula expressing the transient probability distribution of states $\mathbf{p}_{\mathcal{N}}(t)$ structurally in such case.

To conclude, all the considerations regarding the Kronecker algebra-based representation of an infinitesimal generator matrix for both local transitions (cf. Equation 3.1) and synchronized transitions (cf. Equation 3.3) are summarized by the Theorem 10, which presents the general formula describing the structural representation of any transition rate matrix.

Theorem 10. [34] Let $\mathcal{N} = (\mathscr{A}_{\mathcal{N}}, \mathscr{E}_{\mathcal{N}})$ be a SAN with $\mathscr{A}_{\mathcal{N}} = \left\{\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(N_{\mathcal{N}})}\right\}$. For each automaton $A^{(k)} \in \mathscr{A}_{\mathcal{N}}$ and each event $e \in \mathscr{E}_{\mathcal{N}}$ define the matrices: $Q^{loc}_{\mathcal{A}^{(k)}}$, $Q^{e_{neg}}_{\mathcal{A}^{(k)}}$ and $Q^{e_{pos}}_{\mathcal{A}^{(k)}} \in \mathscr{M}_{S_{\mathcal{A}^{(k)}}}$ according to the aforementioned rules. Then, by separating local transitions from synchronized transitions, the global infinitesimal generator matrix $Q_{\mathcal{N}} \in \mathscr{M}_{S_{\mathcal{N}}}$ describing the time evolution of the SAN $\mathcal{N}$ is given by the sum of $N_{\mathcal{N}} + 2 \cdot E_{\mathcal{N}}$ tensor products as follows:

$$Q_{\mathcal{N}} = \bigoplus_{k=1}^{N_{\mathcal{N}}} Q^{loc}_{\mathcal{A}^{(k)}} + \sum_{e \in \mathscr{E}^{syn}_{\mathcal{N}}} \left[\bigotimes_{k=1}^{N_{\mathcal{N}}} Q^{e_{pos}}_{\mathcal{A}^{(k)}} + \bigotimes_{k=1}^{N_{\mathcal{N}}} Q^{e_{neg}}_{\mathcal{A}^{(k)}}\right],$$

where:

$$\bigoplus_{k=1}^{N_{\mathcal{N}}} Q^{loc}_{\mathcal{A}^{(k)}} = \sum_{k=1}^{N_{\mathcal{N}}} I_{S_{\mathcal{A}^{(1)}}} \otimes \ldots \otimes I_{S_{\mathcal{A}^{(k-1)}}} \otimes Q^{loc}_{\mathcal{A}^{(k)}} \otimes I_{S_{\mathcal{A}^{(k+1)}}} \otimes \ldots \otimes I_{S_{\mathcal{A}^{(N_{\mathcal{N}})}}}.$$

This structured representation of a transition rate matrix $Q_{\mathcal{N}}$ is also called the *descriptor* of the network $\mathcal{N}$.

Proof. Cf. [34]. □

Example 1. To sum up all the considerations regarding stochastic automata and their networks, consider the following example describing a SAN model of the resource sharing mechanism, which is a modification of one of benchmark problems proposed by Benoit et al. [3]. The original model incorporates a set of homogeneous automata representing competing processes, each of which equipped with two states corresponding to either exploitation or no use of the shared resource by this particular process. The following extension includes three types of heterogeneous processes, each of which enriched with additional states and diversified local transitions corresponding to local computations carried out independently by particular machines. All the competing processes do not cooperate each other, but synchronize their access to the shared resource by referring to the additional counting automaton, which constantly tracks the number of entities occupying the shared resource and controls the access to the protected member of the simulated system.

An automaton of the first type – denoted by $\mathcal{A}^{(1)}$ – incorporates three local states – denoted by $S1$, $S2$ and $S4$ – and one state $S3$ corresponding to the exploitation of the shared resource. A graphical representation of the automaton $\mathcal{A}^{(1)}$ as a directed, labelled graph is given by the Figure 3.1.

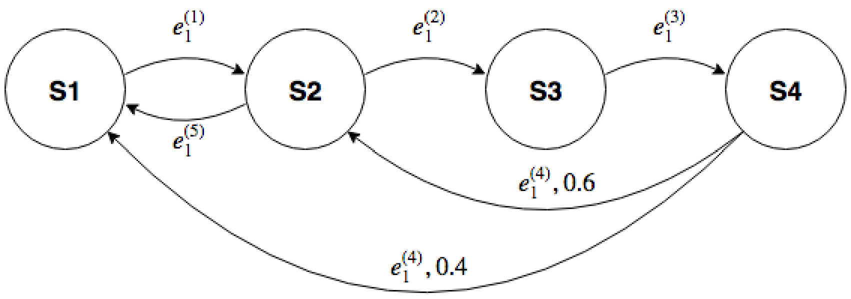


Figure 3.1. Graph representation of the automaton of the first kind.

Definitions of all events mastered by the automaton $\mathcal{A}^{(1)}$ are summarized by the Table 3.1.

event	**type**	**rate**
$e_1^{(1)}$	local	λ_1
$e_1^{(2)}$	synchronizing	λ_2
$e_1^{(3)}$	synchronizing	λ_3
$e_1^{(4)}$	local	λ_4
$e_1^{(5)}$	local	λ_5

Table 3.1. Specification of events triggered by the automaton of the first kind.

Finally, recalling the definitions of descriptor-constituent matrices provided by the Sections 3.2.1 and 3.2.2, all the non-identity matrices corresponding to both local and synchronized transitions involving the automaton $\mathcal{A}^{(1)}$ are given below.

$$Q^{loc}_{\mathcal{A}^{(1)}} = \begin{bmatrix} -\lambda_1 & \lambda_1 & 0 & 0 \\ \lambda_5 & -\lambda_5 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.4 \cdot \lambda_4 & 0.6 \cdot \lambda_4 & 0 & -\lambda_4 \end{bmatrix}$$

$$Q^{\left(e_1^{(2)}\right)_{pos}}_{\mathcal{A}^{(1)}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda_2 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q^{\left(e_1^{(2)}\right)_{neg}}_{\mathcal{A}^{(1)}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -\lambda_2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$Q^{\left(e_1^{(3)}\right)_{pos}}_{\mathcal{A}^{(1)}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q^{\left(e_1^{(3)}\right)_{neg}}_{\mathcal{A}^{(1)}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\lambda_3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

An automaton of the second type – denoted by $\mathcal{A}^{(2)}$ – is composed of two local states – denoted by $S1$ and $S2$ – and one state $S3$ corresponding to the utilization of the shared resource. A graph structure describing the automaton $\mathcal{A}^{(2)}$ is presented in the Figure 3.2.

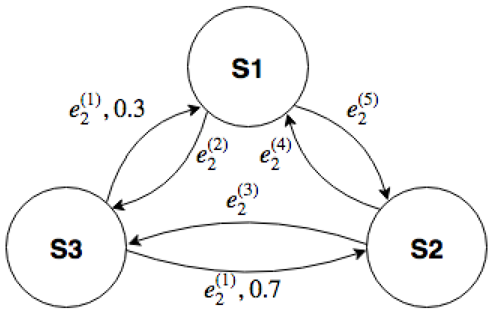


Figure 3.2. Graph representation of the automaton of the second kind.

Definitions of all events triggered by the automaton $\mathcal{A}^{(2)}$ are gathered in the Table 3.2.

event	**type**	**rate**
$e_2^{(1)}$	synchronizing	μ_1
$e_2^{(2)}$	synchronizing	μ_2
$e_2^{(3)}$	synchronizing	μ_3
$e_2^{(4)}$	local	μ_4
$e_2^{(5)}$	local	μ_5

Table 3.2. Specification of events triggered by the automaton of the second kind.

All the non-identity matrices corresponding to both local and synchronized transitions affecting the automaton $\mathcal{A}^{(2)}$ are presented below.

$$Q^{loc}_{\mathcal{A}^{(2)}} = \begin{bmatrix} -\mu_4 & \mu_4 & 0 \\ \mu_5 & -\mu_5 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$Q^{\left(e_2^{(1)}\right)_{pos}}_{\mathcal{A}^{(2)}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.3\cdot\mu_1 & 0.7\cdot\mu_1 & 0 \end{bmatrix} \qquad Q^{\left(e_2^{(1)}\right)_{neg}}_{\mathcal{A}^{(2)}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -\mu_1 \end{bmatrix}$$

$$Q^{\left(e_2^{(2)}\right)_{pos}}_{\mathcal{A}^{(2)}} = \begin{bmatrix} 0 & 0 & \mu_2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad Q^{\left(e_2^{(2)}\right)_{neg}}_{\mathcal{A}^{(2)}} = \begin{bmatrix} -\mu_2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$Q^{\left(e_2^{(3)}\right)_{pos}}_{\mathcal{A}^{(2)}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & \mu_3 \\ 0 & 0 & 0 \end{bmatrix} \qquad Q^{\left(e_2^{(3)}\right)_{neg}}_{\mathcal{A}^{(2)}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\mu_3 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

An automaton of the third type – denoted by $\mathcal{A}^{(3)}$ – incorporates three local states – denoted by $S1$, $S2$ and $S4$ – and one state $S3$ corresponding to the exploitation of the shared resource. A graphical representation of this automaton is provided by the Figure 3.3.

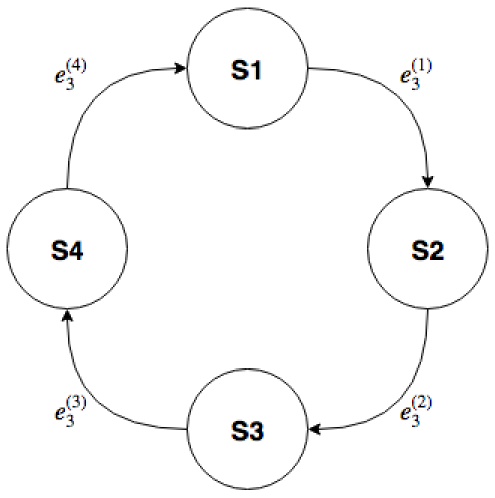


Figure 3.3. Graph representation of the automaton of the third kind.

Definitions of all events mastered by the automaton $\mathcal{A}^{(3)}$ are summarized by the Table 3.3.

event	**type**	**rate**
$e_3^{(1)}$	local	δ_1
$e_3^{(2)}$	synchronizing	δ_2
$e_3^{(3)}$	synchronizing	δ_3
$e_3^{(4)}$	local	δ_4

Table 3.3. Specification of events triggered by the automaton of the third kind.

All the non-identity matrices corresponding to both local and synchronized transitions involving the automaton $\mathcal{A}^{(3)}$ are given below.

$$Q^{loc}_{\mathcal{A}^{(3)}} = \begin{bmatrix} -\delta_1 & \delta_1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \delta_4 & 0 & 0 & -\delta_4 \end{bmatrix}$$

$$Q^{\left(e_3^{(2)}\right)_{pos}}_{\mathcal{A}^{(3)}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & \delta_2 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q^{\left(e_3^{(2)}\right)_{neg}}_{\mathcal{A}^{(3)}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -\delta_2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$Q^{\left(e_3^{(3)}\right)_{pos}}_{\mathcal{A}^{(3)}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \delta_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q^{\left(e_3^{(3)}\right)_{neg}}_{\mathcal{A}^{(3)}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\delta_3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The counting automaton, controlling the access to the shared resource – denoted by $\mathcal{A}^{(4)}$ – does not master any events, because it does not evolve independently. The component in question acts as a slave of other automata (representing competing processes), thus all its transitions are triggered in consequence of acquiring and unleashing permissions to utilize the protected entity by the cooperating machines. Any state Si, $1 \leq i \leq P+1$, represents a situation that exactly $i-1$ processes currently utilize the shared resource. Therefore, an automaton of $P+1$ states allows at most P processes to use this entity at the same time. If $P=1$, the model reflects a solution to the well-known mutual exclusion problem. On the other hand, when P is equal to the number of competing processes, they are completely independent. Summarizing, the graphical representation of the automaton $\mathcal{A}^{(4)}$ is given by the Figure 3.4.

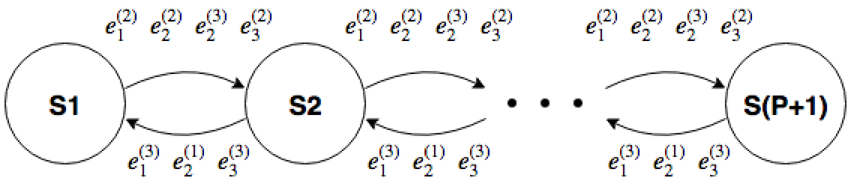

S1
S2
S(P+1)
$e_1^{(2)}$ $e_2^{(2)}$ $e_2^{(3)}$ $e_3^{(2)}$
$e_1^{(3)}$ $e_2^{(1)}$ $e_3^{(3)}$

Figure 3.4. Graph representation of the counting automaton.

When the events corresponding to acquiring the access to the shared resource are triggered, the counting automaton contributes to the descriptor of the simulated SAN in the following way:

$$Q_{\mathcal{A}^{(4)}}^{\left(e_1^{(2)}\right)_{pos}} = Q_{\mathcal{A}^{(4)}}^{\left(e_2^{(2)}\right)_{pos}} = Q_{\mathcal{A}^{(4)}}^{\left(e_2^{(3)}\right)_{pos}} = Q_{\mathcal{A}^{(4)}}^{\left(e_3^{(2)}\right)_{pos}} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

$$Q_{\mathcal{A}^{(4)}}^{\left(e_1^{(2)}\right)_{neg}} = Q_{\mathcal{A}^{(4)}}^{\left(e_2^{(2)}\right)_{neg}} = Q_{\mathcal{A}^{(4)}}^{\left(e_2^{(3)}\right)_{neg}} = Q_{\mathcal{A}^{(4)}}^{\left(e_3^{(2)}\right)_{neg}} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix}.$$

Similarly, when the access right is released, the following matrices are included in the descriptor form:

$$Q_{\mathcal{A}^{(4)}}^{\left(e_1^{(3)}\right)_{pos}} = Q_{\mathcal{A}^{(4)}}^{\left(e_2^{(1)}\right)_{pos}} = Q_{\mathcal{A}^{(4)}}^{\left(e_3^{(3)}\right)_{pos}} = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 & 0 \\ 1 & 0 & \dots & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & 0 & \dots & 0 & 1 & 0 \end{bmatrix}$$

$$Q_{\mathcal{A}^{(4)}}^{\left(e_1^{(3)}\right)_{neg}} = Q_{\mathcal{A}^{(4)}}^{\left(e_2^{(1)}\right)_{neg}} = Q_{\mathcal{A}^{(4)}}^{\left(e_3^{(3)}\right)_{neg}} = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & \dots & 0 & 0 & 1 \end{bmatrix}.$$

3.3. Summary

This Chapter presented a comprehensive introduction to SANs without functional dependencies between components and methods of their representation, both formally and graphically. It began with a definition of single stochastic automaton and was followed by an exhaustive description of automata networks composed of a collection of both independent and cooperating components. Additionally, the Kronecker representation of a SAN descriptor – which is a fundamental structure employed by the TNSAN algorithm (cf. Chapter 7) – was provided and supplemented by a description of the benchmark problem, extensively investigated in the Chapter 8.

4. Introduction to Tensor Networks

This Chapter begins with a formal description of both algebraic and diagrammatic representations of tensors (Section 4.1), which is followed by an introduction to selected operations performed on them (Section 4.2). Finally, TNs and their contractions are described (Section 4.3). Please note that the content of the following Chapter is adapted to the scope of this research, therefore it should not be considered as a comprehensive study in the field of the TNs formalism, which is provided by the suggested literature.

4.1. Tensors

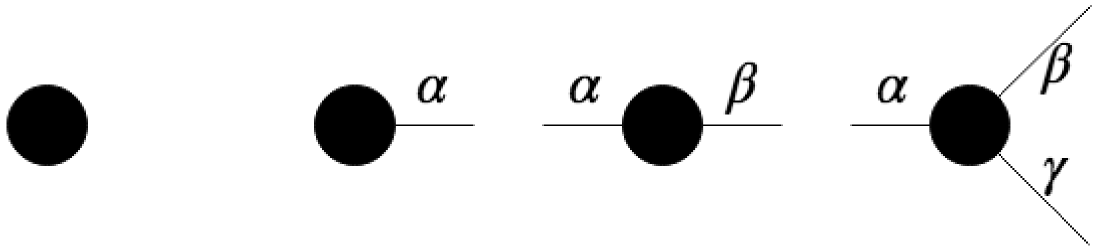
Tensors, introduced together with their attributes by the Definition 18 and supplemented by the Example 2, are fundamental building blocks of TN structures exploited within this thesis to perform efficient computations on complex algebraic objects. Then, a remark regarding the graphic notation of tensors is provided and followed by the Example 3. Please note that within this document any tensor is denoted by uppercase italic Roman letter with a list of its indices (represented by distinguishable Greek letters) placed in a subscript.

Definition 18. [7, 30] The *rank-d tensor* $T_{\alpha^{(1)},\alpha^{(2)},\dots,\alpha^{(d)}}$ of *indices* $\alpha^{(1)},\alpha^{(2)},\dots,\alpha^{(d)}$, each of which of *size* (i.e. the number of possible values of given index) $\left|\alpha^{(i)}\right|$, $1 \leq i \leq d$, is an element of the $\mathbb{R}^{\left|\alpha^{(1)}\right|\cdot\left|\alpha^{(2)}\right|\cdot\ldots\cdot\left|\alpha^{(d)}\right|}$ space (generally tensors are defined over complex numbers, however the scope of this research requires reals). Equivalently, one may define the tensor $T_{\alpha^{(1)},\alpha^{(2)},\dots,\alpha^{(d)}}$ as a d-dimensional array of real numbers, with i-th dimension described by the index $\alpha^{(i)}$ of size $\left|\alpha^{(i)}\right|$. An element of the tensor $T_{\alpha^{(1)},\alpha^{(2)},\dots,\alpha^{(d)}}$ is given by $T_{i_1,i_2,\dots,i_d}$ for $1 \leq i_1 \leq \left|\alpha^{(1)}\right|, 1 \leq i_2 \leq \left|\alpha^{(2)}\right|, \dots, 1 \leq i_d \leq \left|\alpha^{(d)}\right|$.

Example 2. One may immediately observe than a rank-0 tensor is a scalar, a rank-1 tensor T_α is a vector of $|\alpha|$ real numbers, while a rank-2 tensor $T_{\alpha,\beta}$ is a matrix of $|\alpha|$ rows and $|\beta|$ columns. Further, a rank-3 tensor $T_{\alpha,\beta,\gamma}$ may be viewed as a cuboid of $|\alpha|\cdot|\beta|\cdot|\gamma|$ numbers, or – equivalently – as a vector of equal-size matrices.

Remark. [7, 30] Probably the most powerful feature of tensors is a diagrammatic notation of their structures and operations performed on them, which significantly facilitates the comprehension of calculations applied to complex numerical objects. Within this graphical representation tensors are drawn as circles, while their indices are given by lines emerging from these circles (often called *legs* and labeled by the index sign).

Example 3. Following the Example 2 presenting tensors of various ranks, their diagrammatic equivalents are summarized by the Figure 4.1.



(a) A rank-0 tensor. (b) A rank-1 tensor. (c) A rank-2 tensor. (d) A rank-3 tensor.

Figure 4.1. Examples of diagrammatic representations of variable-rank tensors.

4.2. Tensor operations

To begin with, the trace operation is introduced by the Definition 19 and followed by the Example 4 describing the analogy between the well-known definition of trace for square matrices and the trace of a tensor.

Definition 19. [7] Let $T_{\alpha^{(1)},\alpha^{(2)},\ldots,\alpha^{(d)}}$ be a tensor and $\alpha^{(m)}$, $\alpha^{(n)}$ be two distinct indices of T, such that $m < n$ and $\left|\alpha^{(m)}\right| = \left|\alpha^{(n)}\right| = a$. The (partial) *trace* of the tensor T over indices $\alpha^{(m)}$ and $\alpha^{(n)}$ is a tensor $\left(\mathrm{Tr}_{\alpha^{(m)},\alpha^{(n)}}T\right)_{\alpha^{(1)},\ldots,\alpha^{(m-1)},\alpha^{(m+1)},\ldots,\alpha^{(n-1)},\alpha^{(n+1)},\ldots,\alpha^{(d)}}$, such that:

$$\left(\mathrm{Tr}_{\alpha^{(m)},\alpha^{(n)}}T\right)_{\alpha^{(1)},\ldots,\alpha^{(m-1)},\alpha^{(m+1)},\ldots,\alpha^{(n-1)},\alpha^{(n+1)},\ldots,\alpha^{(d)}} \overset{\text{def}}{=} \sum_{i=1}^{a} T_{\alpha^{(1)},\ldots,\alpha^{(m-1)},i,\alpha^{(m+1)},\ldots,\alpha^{(n-1)},i,\alpha^{(n+1)},\ldots,\alpha^{(d)}}.$$

Example 4. Consider a rank-2 tensor $T_{\alpha,\alpha}$. Its trace is a scalar number (i.e. a rank-0 tensor) $\mathrm{Tr}_\alpha T$, such that:

$$\mathrm{Tr}_\alpha T = \sum_{i=1}^{|\alpha|} T_{i,i}.$$

Please note that the tensor $T_{\alpha,\alpha}$ is equivalent to a matrix $T_{|\alpha|\times|\alpha|} = [t_{ij}]$, thus the rand-hand side of the equation above may be expressed as a sum of diagonal elements of the matrix T: $\sum_{i=1}^{|\alpha|} t_{ii}$, which complies with the definition of trace for any square matrix.

Diagrammatically, the trace operation performed on a tensor is represented by joined lines representing indices the tensor is traced over. As an example, consider the Figure 4.2 of a rank-3 tensor with two corresponding indices and single non-traced index. In fact, performing the trace operation on this tensor results in obtaining a rank-1 tensor with one and only index reflecting the non-contracted leg belonging to the original tensor.

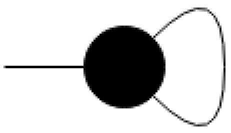


Figure 4.2. Tracing of a rank-3 tensor over two corresponding indices.

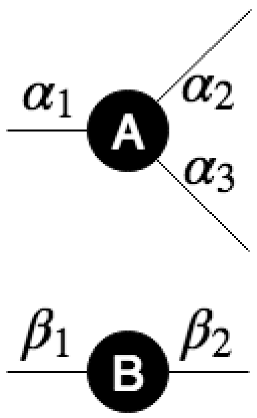
Although defined for rectangular matrices, the Kronecker product operation is generalized to multidimensional tensor structures by the tensor product introduced by the Definition 20 and supplemented by the Example 5 presenting its various diagrammatic notations.

Definition 20. [7] Let $T^{(1)}_{\alpha_1^{(1)},\alpha_1^{(2)},\ldots,\alpha_1^{(d_1)}}$ and $T^{(2)}_{\alpha_2^{(1)},\alpha_2^{(2)},\ldots,\alpha_2^{(d_2)}}$ be tensors. The *tensor product* of $T^{(1)}$ and $T^{(2)}$ is a tensor $\left(T^{(1)} \otimes T^{(2)}\right)_{\alpha_1^{(1)},\alpha_1^{(2)},\ldots,\alpha_1^{(d_1)},\alpha_2^{(1)},\alpha_2^{(2)},\ldots,\alpha_2^{(d_2)}}$, being a result of the element-wise product of the values belonging to each constituent tensor:

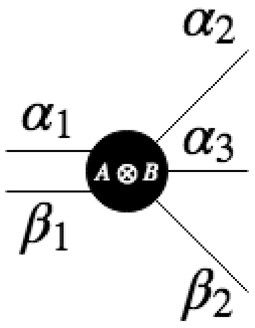
$$\left(T^{(1)} \otimes T^{(2)}\right)_{i_1,i_2,\ldots,i_{d_1},j_1,j_2,\ldots,j_{d_2}} = T^{(1)}_{i_1,i_2,\ldots,i_{d_1}} \cdot T^{(2)}_{j_1,j_2,\ldots,j_{d_2}},$$

where: $\forall_{1\leq k\leq d_1} : 1 \leq i_k \leq \left|\alpha_1^{(k)}\right|$ and $\forall_{1\leq k\leq d_2} : 1 \leq j_k \leq \left|\alpha_2^{(k)}\right|$.

Example 5. Diagrammatically, the tensor product of two tensors is represented either by two tensors separately placed next to each other (cf. Figure 4.3a) or single tensor equipped with indices derived from constituents (cf. Figure 4.3b). Please note the equivalence between the notations of a tensor product of tensors $A_{\alpha_1,\alpha_2,\alpha_3}$ and B_{β_1,β_2}, presented in the Figure 4.3.



(a) Tensor product of tensors $A_{\alpha_1,\alpha_2,\alpha_3}$ and B_{β_1,β_2}.



(b) Tensor $(A \otimes B)_{\alpha_1,\alpha_2,\alpha_3,\beta_1,\beta_2}$.

Figure 4.3. Diagrammatic representations of a tensor product operation.

The most fundamental operation performed on tensors is the contraction of their indices, formally described by the Definition 21. Afterwards, this concept is clarified by the Example 6 presenting the similarity between the ordinary matrix multiplication and the contraction of rank-2 tensors by a common index, as well as a remark concerning efficiency of such computations.

Definition 21. [7, 30] Let $T^{(1)}_{\alpha_1^{(1)},\ldots,\alpha_1^{(m-1)},\alpha,\alpha_1^{(m+1)},\ldots,\alpha_1^{(d_1)}}$ and $T^{(2)}_{\alpha_2^{(1)},\ldots,\alpha_2^{(n-1)},\alpha,\alpha_2^{(n+1)},\ldots,\alpha_2^{(d_2)}}$ be tensors with corresponding indices $\alpha_1^{(m)}$ and $\alpha_2^{(n)}$, $1 \leq m \leq d_1$ and $1 \leq n \leq d_2$, denoted by α. The *contraction* of $T^{(1)}$ and $T^{(2)}$ by α is a tensor $\left(T^{(1)} \circ_\alpha T^{(2)}\right)_{\alpha_1^{(1)},\ldots,\alpha_1^{(m-1)},\alpha_1^{(m+1)},\ldots,\alpha_1^{(d_1)},\alpha_2^{(1)},\ldots,\alpha_2^{(n-1)},\alpha_2^{(n+1)},\ldots,\alpha_2^{(d_2)}}$, such that:

$$\left(T^{(1)} \circ_\alpha T^{(2)}\right)_{\alpha_1^{(1)},\ldots,\alpha_1^{(m-1)},\alpha_1^{(m+1)},\ldots,\alpha_1^{(d_1)},\alpha_2^{(1)},\ldots,\alpha_2^{(n-1)},\alpha_2^{(n+1)},\ldots,\alpha_2^{(d_2)}} \overset{\text{def}}{=} \mathrm{Tr}_\alpha \left(T^{(1)} \otimes T^{(2)}\right).$$

Additionally, please note that the index α is called the *contracted index*, while all other indices – which are not subjected to contraction – are referred as *open indices*.

Example 6. As an example, consider two rank-2 tensors $A_{\alpha,\beta}$ and $B_{\beta,\gamma}$ with a corresponding index β. The contraction of tensors A and B by the index β results in obtaining a new tensor $C_{\alpha,\gamma} = A \circ_\beta B$, such that:

$$C_{i,j} = \left(A_{\alpha,\beta} \circ_\beta B_{\beta,\gamma}\right)_{i,j} = \left(\mathrm{Tr}_\beta \left(A \otimes B\right)_{\alpha,\beta,\beta,\gamma}\right)_{i,j} = \sum_{k=1}^{|\beta|} \left(A \otimes B\right)_{i,k,k,j} = \sum_{k=1}^{|\beta|} A_{i,k} \cdot B_{k,j}.$$

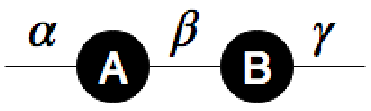
for any $1 \leq i \leq |\alpha|$ and $1 \leq j \leq |\gamma|$.

Now consider two matrices $A_{|\alpha|\times|\beta|} = [a_{ij}]$ and $B_{|\beta|\times|\gamma|} = [b_{ij}]$, and their ordinary product being a matrix $C_{|\alpha|\times|\gamma|} = [c_{ij}]$. Naturally, each element c_{ij} is given by:

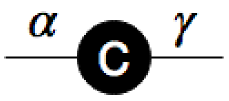
$$c_{ij} = \sum_{k=1}^{|\beta|} a_{ik} \cdot b_{kj},$$

for any $1 \leq i \leq |\alpha|$ and $1 \leq j \leq |\gamma|$, which is straightforwardly equivalent to the form of any element $C_{i,j}$ obtained for its tensor counterpart. Therefore, the contraction of tensors $A_{\alpha,\beta}$ and $B_{\beta,\gamma}$ by the corresponding index β is equivalent to the ordinary matrix-by-matrix multiplication.

Diagrammatically, a contracted index is represented by a line connecting two circles denoting the tensors which this index belongs to, while an open index is given by a leg connected to single shape. The preceding example is visualized by the Figure 4.4. Please note that performing the contraction results graphically in disappearing the contracted index and "collapsing" the two connected circles into a single one, with all other indices (previously belonging to the constituent tensors) attached.



(a) Contraction of two tensors $A_{\alpha,\beta}$ and $B_{\beta,\gamma}$ by the index β.



(b) Contraction result.

Figure 4.4. Visual representations of a tensor contraction operation and its result.

Remark. One may immediately observe that there may exist multiple ways to contract a set of many interconnected tensors. In general, the order of tensors contraction execution does not affect the final result of the whole operation, although it may have significant footprint on time and memory requirements of such computations.

Following the example given by Bridgeman and Chubb [7], consider a "ladder-shaped" set of $2n$ ($n \in \mathbb{N}^+$) tensors presented in the Figure 4.5. Then, examine two contraction orders, presented respectively in the Figures 4.6 and 4.7, where consecutive contraction steps are separated by horizontal right arrows and an index contracted within particular step is marked in red.

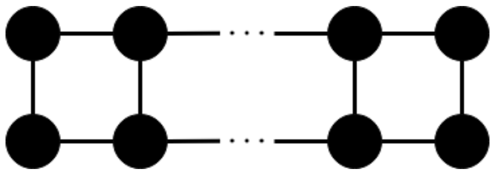


Figure 4.5. A "ladder-shaped" set of interconnected tensors.

One of the most straightforward ways to perform a contraction of the aforementioned model is to iterate along the top of the ladder, and then along its bottom, as presented in the Figure 4.6. Note that during the contraction process, the tensor marked in blue in the Figure 4.6 is equipped with n indices, therefore its contraction time, as well as memory requirements scale exponentially with the growth of n. This phenomenon, in turn, makes it impossible to obtain desired result for reasonable values of n.

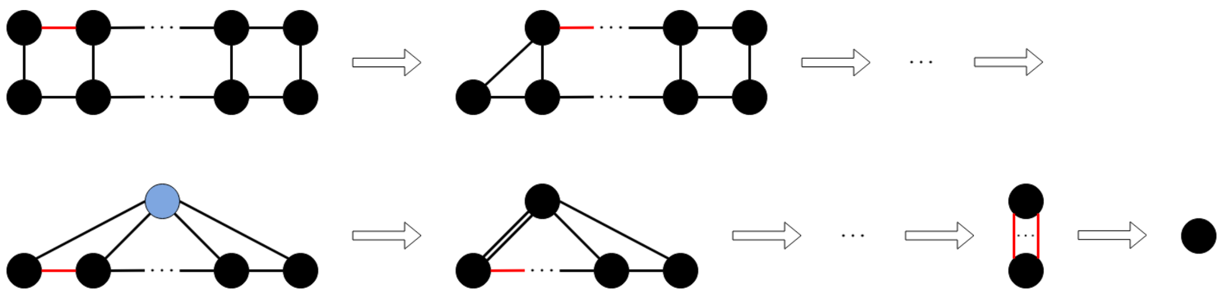


Figure 4.6. Inefficient order of contractions performed on the "ladder-shaped" tensors.

Alternatively, one may perform contractions of consecutive ladder rungs, one after another, as presented in the Figure 4.7. Then, each tensor is always equipped with at most three indices, which imposes polynomial contraction time and constant memory requirements regardless the value of n.

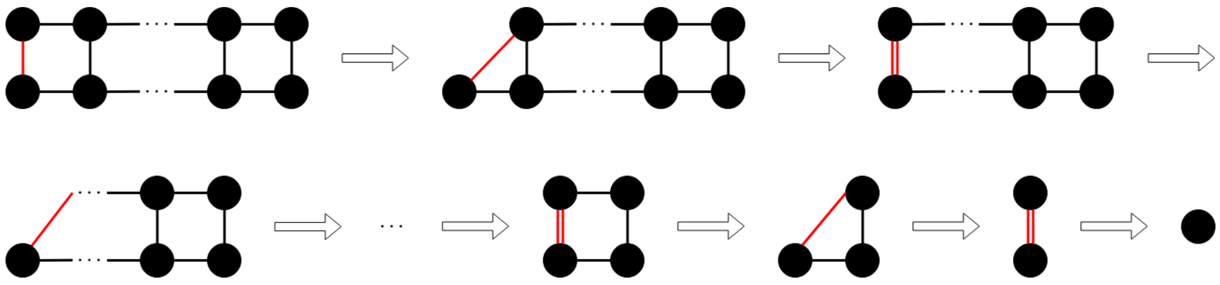


Figure 4.7. Efficient order of contractions performed on the "ladder-shaped" tensors.

Following the considerations above, tensor-based structures usually allow both efficient and inefficient contraction schemes, although there exist some that do not admit any efficient contraction ordering – as an example, one may consider a two-dimensional lattice of tensors [7]. Furthermore, irrespective of multiple ways of performing contractions on given set of tensors, the problem of determining the optimal contraction sequence is proven to be NP-hard [7, 32].

Another powerful feature of the formalism discussed is an ability to break down multidimensional tensors into various structural representations composed of many low-rank tensors. These decompositions are natural extensions of matrix decomposition strategies like Singular Value Decomposition (SVD) [24]. Out of multiple tensor splitting approaches (cf. Figure 5.1), one of them – Tensor Train (also called Matrix Product States (MPS) with open boundary conditions) – used within this research is introduced by the Definition 22 and followed by the Example 7.

Definition 22. [31] Let $T_{\alpha^{(1)},\alpha^{(2)},\ldots,\alpha^{(d)}}$ be a rank-d tensor. The *Tensor Train* decomposition of the tensor T is given by the sequence of d tensors $G^{(1)}_{\beta^{(0)},\alpha^{(1)},\beta^{(1)}}, G^{(2)}_{\beta^{(1)},\alpha^{(2)},\beta^{(2)}}, \ldots, G^{(d)}_{\beta^{(d-1)},\alpha^{(d)},\beta^{(d)}}$, such that:

$$T_{\alpha^{(1)},\alpha^{(2)},\ldots,\alpha^{(d)}} = G^{(1)}_{\beta^{(0)},\alpha^{(1)},\beta^{(1)}} \circ_{\beta^{(1)}} G^{(2)}_{\beta^{(1)},\alpha^{(2)},\beta^{(2)}} \circ_{\beta^{(2)}} \ldots \circ_{\beta^{(d-1)}} G^{(d)}_{\beta^{(d-1)},\alpha^{(d)},\beta^{(d)}}.$$

Hence, it has to be noted that the boundary indices $\beta^{(0)}$ and $\beta^{(d)}$ are trivial ones, i.e. $\left|\beta^{(0)}\right| = \left|\beta^{(d)}\right| = 1$.

Example 7. An example of the diagrammatic representation of the Tensor Train decomposition for a rank-3 tensor $T_{\alpha^{(1)},\alpha^{(2)},\alpha^{(3)}}$ is presented in the Figure 4.8.

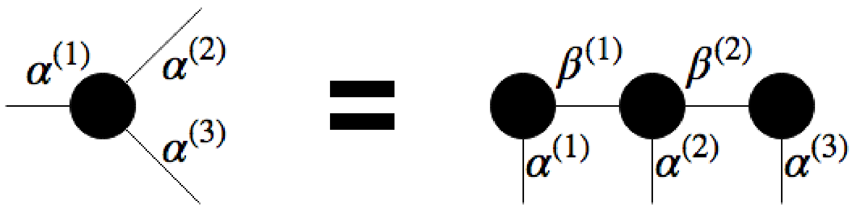


Figure 4.8. Tensor Train decomposition of a rank-3 tensor.

4.3. Tensor Networks

Finally, TNs – the second eponymous component of this thesis – are introduced by the Definition 23, which is followed by an explanatory remark. Please note that within this document all TNs are denoted by uppercase Roman letters with the "blackboard bold" typeface, such as $\mathbb{T}$ (excluding the symbols $\mathbb{N}$ and $\mathbb{R}$, intended for, respectively, natural and real numbers).

Definition 23. [30] A set $\mathscr{T} = \left\{ T^{(1)}_{\alpha_1^{(1)},\alpha_1^{(2)},\ldots,\alpha_1^{(d_1)}}, T^{(2)}_{\alpha_2^{(1)},\alpha_2^{(2)},\ldots,\alpha_2^{(d_2)}}, \ldots, T^{(N)}_{\alpha_N^{(1)},\alpha_N^{(2)},\ldots,\alpha_N^{(d_N)}} \right\}$ of N tensors, where some – or all – of their indices are subjected to contraction, is called the *Tensor Network*.

Remark. For the sake of simplicity, one should notice that the process of performing contractions on the tensors belonging to a TN is called the contraction of this network. Please also note that all tensor structures presented in Examples 6 and 7 are, in fact, TNs. It should be also pointed out that TNDs are notably useful for extensive, complex network structures, when the number of different indices ceases to be trackable using only algebraic methods, which is the case of this research.

4.4. Summary

In this Chapter essential information regarding tensors and their structures were collected. To begin with, tensors, their properties and diagrammatic notation were defined. Afterwards, multiple operations to be performed on tensors were collected, with emphasis on the contraction – which is a fundamental operation employed by the TNSAN algorithm (cf. Chapter 7) – supplemented by comprehensive performance notes. Then, the idea of tensor decompositions was presented by the example of the Tensor Train format. Finally, the TN was introduced formally.

This Chapter finishes the introductory part of this document, hence the Reader should be familiar with the formalism used across the thesis from now on. Based on the gathered knowledge, the research problem is going to be fully defined in the subsequent Chapter 5.

5. Formulation of the research problem

This Chapter begins with an extensive study of recent advances in the areas of SANs and TNs (Section 5.1). Then, it is followed by a detailed description of the problem addressed (Section 5.2) and supplemented by a list of research questions, hypotheses and goals (Section 5.3).

5.1. Related work

Elaboration and development of simulation methods for Markovian models is the subject of extensive research carried out for decades, therefore the first part of the following state of the art analysis is limited only to methods designed with a view to the SAN modelling formalism and its characteristic properties. The second part of this Section presents a brief survey of TNs capabilities utilized in solving multidimensional numerical problems.
Please note that this Section is intended only to outline multiple approaches taken by researchers to solve problems related to the subject matter of this research, and detailed information about particular solutions are provided by the suggested literature. However the Readers not familiar with the concepts of SANs and TNs are strongly advised to get familiar with Chapters 3 and 4 at first in order to comprehensively understand contents of the following Section.

5.1.1. Survey of Stochastic Automata Networks simulation techniques

There are two general groups of numerical methods for Markovian models simulation, used to determine either stationary or transient probability distributions of states over time [6]. The stationary distribution of a Markov chain describes long-term behavior of the system under simulation, which is independent of its initial state. Although steady states of Markov chains usually reflect distributed systems characteristics (such as deadlock or starvation), they may also represent many measurable quantities (including throughput and average response time). On the contrary, the transient distribution of a Markov chain describes state of the system under simulation (e.g. the number of processes using a shared resource and temporary occupancy of each constituent queue) at any time.

All the numerical methods designed especially for the SAN simulation exploit the Kronecker structure of the infinitesimal generator matrix describing evolution of the underlying CTMC. This representation is appreciated for its natural memory efficiency (because it does not require neither generating nor storing single, usually sparse, global matrix [36]), however it implies larger CPU time overhead [3, 36]. In general, a vast majority of methods dedicated to SAN simulation struggle with the memory versus CPU time trade-off [36], which means that memory-efficient algorithms are less time-effective, while the reduction in computational time may be obtained at the expense of greater memory consumption.

It should be also noted that nearly all SAN simulation methods elaborated so far are designed to compute stationary distributions of underlying CTMCs, while transient distributions are usually determined using algorithms universal to all Markovian models.

Before getting into specifics, some general literature overviews should be suggested. The elementary monograph by Stewart [44] comprehensively discusses the vast majority of numerical methods ever used for determining both transient and stationary probability distributions of states belonging to CTMCs. Another reliable knowledge resources in the subject matter of this research are selected chapters of a book edited by Grassmann [42, 45], as well as the monograph on Markovian models by Bolch [6]. There are also multiple survey papers extensively summarizing numerous approaches taken by researchers to adapt the CTMC simulation methods so as to exploit specific properties of SANs, to cite a few: [1, 3, 46].

5.1.1.1. General approaches to reduce computational cost

Making use of sparse representation methods is probably the most natural approach to overcome sparsity of the global infinitesimal generator matrix. There are multiple techniques which allow to store such structures in a memory-saving way [1, 13], as well as to perform time efficient computations on them (cf. Sparse algorithm [13, 44]). Another popular data structures used for storing the global infinitesimal generator matrix are acyclic graph structures such as Binary Decision Diagrams (BDDs), Multi-Terminal BDDs [3] and Probabilistic Decision Graphs [2]. Nevertheless, all of the methods presented are not suitable to matrices with significant percentage of non-zero elements, but also keeping one single matrix may still exceed available memory capabilities [1].

The aforementioned sparsity issue is a direct consequence of the state space explosion problem associated with large Markovian models. Regarding the SAN formalism, this exponential growth in the number of states belonging to the underlying CTMC is directly related to the number of constituent automata and the amount of interactions between them.

In order to reduce the number of automata composing a SAN, the Generalized Tensor Algebra [18] concept is employed. This formalism enables to handle functional entries in matrices forming the Kronecker structure of the infinitesimal generator matrix, so that some synchronization between two automata involving a "middleman" automaton may be expressed in terms of direct interaction between the two communicating components. Please note that every model using Generalized Tensor Algebra may always be expressed in terms of Ordinary Tensor Algebra only [3, 11], however the exclusion of functional cooperation between model components usually requires to introduce additional automata (which enhance effects of the state space explosion phenomenon and increase sparsity of the global infinitesimal generator matrix) [11].

The reduction in the number of automata may be also achieved by using automata grouping methods [3, 36], which main purpose is to join multiple constituent automata into groups and to provide interaction between these structures in order to build an equivalent but simplified SAN. This manipulation naturally reduces the number of components building up a SAN model, and consequently it may lead to disappearing of some cooperation between them [3, 36].

Iterative methods constitute the most popular approach to obtaining both stationary and transient distributions of SANs [36]. The fundamental building block employed by this family of algorithms is the multiplication of a solution vector by the infinitesimal generator matrix represented in the Kronecker structure [36], therefore much attention has been paid to increasing the efficiency of this operation [10]. The most popular approach to perform the vector-descriptor multiplication [36] is the Shuffle algorithm [18], which employs the Kronecker structure of the infinitesimal generator matrix and makes use of the Generalized Tensor Algebra properties in order to speed up execution of a single step of iterative methods. This algorithm usually provides less time complexity of the vector-matrix multiplication and reduces to the standard quadratic time in the worst case [18].
One of the biggest drawbacks associated with the Kronecker-based representation is the fact that it involves introducing (possibly many) unreachable states to the SAN model [2]. There is obviously no need to compute probability distributions for them, because they are constantly equal to zero. It is therefore justified to elaborate algorithms, which operate on the RSS [38] composed of CTMC states with non-zero visit probability instead of the PSS being a Cartesian product of state spaces belonging to all automata composing a SAN model. Thus, there are two modifications of the Shuffle algorithm used: the Partially Reduced (PR) algorithm [1] employs both probability iteration vectors of size equal to the size of RSS and intermediate data structures of size corresponding to the size of PSS, while the Fully Reduced (FR) algorithm [1] utilizes only RSS-size structures.
Depending on types of matrices composing the Kronecker representation of the infinitesimal generator matrix, the vector-descriptor multiplication should be handled by methods applicable to either sparse or dense matrices. This observation led to the elaboration of hybrid algorithms such as Slice [13] and Split [13]. The Slice algorithm exploits the Additive Decomposition property [13] of tensor product in order to handle all but one matrices building up the Kronecker structure using the Sparse algorithm, and the remaining matrix using the Shuffle algorithm. The Split algorithm generalizes the approach adopted by the Slice algorithm by separating all the constituent matrices into two groups - the first one treated by the Sparse algorithm and the other one subjected to the Shuffle algorithm. Please note that in corner cases (when one of these two groups is empty) the Split algorithm reduces to standard Sparse or Shuffle algorithm.

5.1.1.2. Steady state simulation

The problem of finding a stationary distribution for given CTMC naturally reduces to the task of finding a solution of a linear equations system [33, 46]. Therefore, when the system in question is relatively small, one may adopt exact solvers of such equations. Probably the most popular direct method for solving

a system of linear equations is Gaussian elimination [6, 44], which reduces the matrix of coefficients to a triangular form and computes values for unknown variables employing the backward substitution steps. Another common approach for solving such systems is finding the LU decomposition [33, 44] of the coefficient matrix, i.e. the two matrices: lower-triangular L and upper-triangular U, such that their product equals the original matrix. Again, multiple algorithms may be used here, however the most popular two are proposed by Doolittle and Crout [33].
The Gauss-Jordan elimination, a variant of the Gaussian elimination, is also a fundamental approach to finding symbolic solutions [16] of Markovian models. A symbolic solution is in fact a function of input parameters which returns a numeric solution of a particular problem. The algorithm for finding a symbolic solution for a stationary distribution of a SAN, presented in details in [16], starts with slight modification of the Kronecker structure of the infinitesimal generator matrix. Then, the Gauss-Jordan elimination reduces the global infinitesimal generator matrix (without generating it, i.e. using only its tensor-based structure) to the row echelon format which acts as a symbolic solution. Finally, in order to obtain a stationary distribution of a proper CTMC, one has to substitute variables with their numerical values and perform backward substitution steps.
Direct methods for solving systems of linear equations are simple and efficient approaches for systems composed of at most thousands of equations [8]. They also need to be implemented carefully, especially when the numerical stability is a critical issue (cf. Grassmann algorithm [6]).

The most popular algorithms for computing stationary distributions of CTMCs nowadays belong to the family of iterative methods [33, 46], which calculate solution vectors on the basis of a solution obtained in previous iteration and a matrix which remains unchanged during the execution. The simplest iterative method applicable to the SAN formalism is the power method [44, 46] which computes the solution vector in k-th iteration as a product of the initial vector and k-th power of a stochastic matrix derived from the infinitesimal generator matrix of the underlying Markov chain.
Another group of iterative methods designed for computing stationary distributions is constituted by methods based on splittings [47], i.e. representing a matrix as a sum of other matrices. The most popular variants of iterative methods based on splittings represent the global infinitesimal generator matrix as a sum of three matrices: lower-triangular, upper-triangular and diagonal in a Kronecker structure and perform various iteration steps. The exact form of this step is specified by a particular method, including: Jacobi, Gauss-Seidel and Successive Over-Relaxation (SOR) ones and their block variants [33, 44, 47].

Natural decomposition of SANs as a set of cooperating components suggests that it may be possible to express a stationary distribution of a SAN as a tensor product of stationary distributions obtained for constituent automata. The probability vectors satisfying this property are called product form solutions [20] and can be computed for certain classes of SAN models with either no or limited synchronization between constituent automata [20, 36]. Moreover, the approximate product form solution - computed by applying the Alternating Least Squares (ALS) iterative method in order to minimize the norm of residual vector - is obtainable for any network with any desired accuracy (which increases along with the order of product form representation) [9].

Presented iterative approaches to computing stationary distributions of Markov chains may be easily extended by exploiting the fact that a solution vector containing a joint probability distribution may be easily rearranged into a multidimensional tensor, with each dimension corresponding to single component belonging to the SAN under simulation. Therefore, by employing low-rank tensor approximation techniques (such as Canonical Polyadic (CP) decomposition or Matrix Product States (MPS)/Tensor Train (TT) - cf. Section 5.1.3) [26], one may obtain approximate stationary distributions using methods such as Truncated Power Method, ALS-TT - the ALS algorithm with TT representation of probability vectors and Alternating Minimal Energy (AMEn) - the ALS algorithm with adaptive choice of TT ranks for the ALS-TT method [26].

To conclude, despite numerous unquestionable advantages, including preservation of the unaltered Kronecker structure of the infinitesimal generator matrix [33, 46], ease of implementation and ability to control results accuracy [8, 33], iterative algorithms are also well known for their disadvantages.

Due to the memory versus CPU time trade-off, all the aforementioned approaches to increase efficiency of iterative methods attempt to optimize one of these two factors at the expense of the other one. In case of increasing the memory performance, the methods in question need to be modified to utilize auxiliary data structures of size proportional to the number of reachable states. When it comes to time complexity improvements, it should be noted that iterative methods require to minimize both the time devoted to performing calculations during single iteration and the overall number of iterations required to achieve convergence [36, 46]. Furthermore, another shortcoming of this type of algorithms is the fact that their execution time can not be determined in advance [33] (unless a stop condition based on the number of iterations is used).

Technically speaking, a vast majority of iterative methods is infamous for their numerical sensitivity and vulnerability to initial conditions [33], therefore it is usually required to carefully investigate proper values of their parameters and determine appropriate initial conditions, which are essential factors for achieving convergence [36, 46]. Most of the iterative algorithms, however, converge under certain conditions only [44, 46], and, additionally, theoretically convergent cases turn out to be unacceptably slow in reaching a final solution [46]. Thus multiple strategies, at the forefront with preconditioning (including methods such as Neumann series [46], diagonal preconditioning [3] and incomplete LU factorization [3, 33]) and aggregation/disaggregation methods [8] (based on a widespread divide and conquer approach), are employed in order to accelerate the convergence of iterative methods.

5.1.1.3. Transient state simulation

The family of iterative algorithms is applicable not only to finding stationary probability distributions of states belonging to simulated CTMCs, but also to computing their transient probability distributions [42]. In the latter case, the most popular approaches employed nowadays are projection methods [33, 42, 46], especially those known as simultaneous (or subspace) iteration ones [46]. The fundamental assumption taken by this class of algorithms is finding an approximate solution belonging to the lower-dimensional space than the original solution [33, 42] by continuous iteration with a fixed number of vectors [46]. The projection process may be considered as a general framework for multiple algorithms, which differ in the way of choosing adequate subspace and setting additional constraints [33]. In practice, the approximation space is usually a Krylov subspace (which is spanned by consecutive vectors of the power method, i.e. products of an initial vector and subsequent powers of an iteration matrix) [33, 42, 46], while the additional constraints require the residual vector to be orthogonal to the selected Krylov subspace [33]. Probably the greatest disadvantage related to the orthogonal projection on the Krylov subspace is the fact that vectors resulting from successive iterations of the power method are, in principle, almost linearly dependent. Therefore, it is usually desirable to find an orthonormal basis of the chosen Krylov space [42], in order to express any computed approximate solution unambiguously. The most popular numerical algorithm designed for computing approximations of the largest eigenvalues and their corresponding eigenvectors for any Krylov subspace is the Arnoldi Process [33, 42, 46] (and the related Lanczos algorithm [33, 44]), which employs the slightly modified Gram-Schmidt orthogonalization scheme [33, 46]. This concept has been immediately employed by solvers of linear equations systems, primarily by the Generalized Minimum RESidual (GMRES) method [33, 46]. It begins with the construction of an orthonormal basis of a Krylov subspace, acting as an approximation space for a solution of the system under calculations. Then, an approximate solution is computed iteratively by subsequent minimization of the (usually Euclidean) norm of the residual vector [33, 46]. It should be noted that both Arnoldi and GMRES algorithm exploit the vector-descriptor multiplication scheme extensively [33], which may be efficiently calculated making use of the aforementioned approaches. Additionally, these algorithms usually offers much faster convergence than the ordinary power method [46], especially when accompanied with dedicated preconditioning [33].

Each of all possible CTMCs, in particular those underlying any SAN model, satisfy both Kolmogorov backward and Kolmogorov forward equations [6], which are used to compute a transient probability distribution of states belonging to the CTMC under simulation [6]. These equations are, in fact, examples of Ordinary Differential Equations (ODEs) of the First Order, therefore it is fairly justified to employ numerous ODE solvers in order to find a desired transient probability distribution [6, 42, 44]. One of the most popular methods for finding solutions of such equations is the Euler method [42], based on the discretization of both domain and time spaces. This algorithm performs the approximation of the derivative term by a difference quotient, and then builds a system of linear equations to compute a final solution. Although Euler method is one of the most straightforward approaches, its properties are thoroughly investigated and multiple variants have been proposed [42, 44]. The algorithm in question may be naturally generalized by employing higher-order approximations of the first order derivative - this approach is taken by the family of Runge-Kutta methods [42]. Another successful alternative is to apply the Laplace Transform to the Kolmogorov backward equation [6, 42], however this approach is relevant only to small state spaces [6].

In many practical cases, the Uniformization (also called Randomization or Jensen) method [6, 42, 44] turned out to be especially effective in computing transient probability distributions of states belonging to various CTMCs [42]. This algorithm approximates the continuous process under simulation by a

Discrete-Time Markov Chain (DTMC) [6] with a transition probability distribution modelled according to the Poisson process [6]. Then, the problem of finding a desired probability distribution may be reduced to a recursive evaluation of the infinite series [6, 42, 44], which is usually terminated after finitely many terms (when the convergence has been reached) [6, 44].

By employing basic laws of ODE calculus one may derive that a solution of the Kolmogorov backward equation is given simply by an exponential of a transition rate matrix [6] describing the evolution of a CTMC underlying the SAN model under simulation. Therefore, a transient probability distribution of states belonging to any CTMC is nothing more than just a product of the initial probability distribution and the aforementioned matrix exponential [42]. This formula, although trivial to express, is a subject matter of extensive research till nowadays. Numerous methods have been elaborated over the decades in order to determine the matrix exponential, however still there is no panacea for all issues related to the numerical stability and efficiency of these algorithms [29]. Among numerous approaches utilized so far (and quite comprehensively described by Moler and Van Loan in [29]), in the context of CTMCs one may distinguish methods based on: truncation of the explicit power series (given by definition), discretization of the time domain and similarity transformation (primarily eigendecomposition [24]) of the transition rate matrix [42].

5.1.2. Overview of Stochastic Automata Networks simulation software

Discovering new numerical methods for the simulation of CTMCs and SANs has been implying simultaneous development of software packages designed for evaluation of Markovian models. Most of the software available nowadays is dedicated to performing computations on Markov Chains in general, irrespective of any high-level formalism. There are, besides, some powerful applications created with a view to specific modelling methods, especially SANs.

One of the oldest software packages for the simulation of Markov chains is MARCA [43], which provides a possibility to compute stationary probability distribution of states (using both direct and iterative algorithms), as well as transient probability distributions (employing, among others, the Uniformization and Runge-Kutta methods). Apart from software packages, another broad group of applications are model checkers, which main purpose is to examine whether any input model meets given specification and requirements. One of the most widely known model checkers is PRISM [27], while one of the most recent (and also rich-functionality) ones is STORM [15]. Both these applications support numerical analysis of stochastic processes expressed in terms of DTMCs and CTMCs.

Despite the aforementioned software, operating on general Markovian models, there are some applications performing numerical simulations of Markov chains with the exploitation of exceptional properties associated with particular modelling formalisms. Regarding Kronecker algebra-based models, two main approaches are employed - Generalized Stochastic Petri Nets (GSPNs) and SANs [11]. In case of the former one, the software package SMART [12] may be used to perform analysis of systems expressed in terms of Petri Nets formalism. The undisputed and unquestionable leader in the field of SAN simulation is PEPS [19], the tool developed since the very beginning of the latter approach in late ’80s. In comparison to the advances in SAN modelling presented in previous Sections, this software implements the following features: efficient vector-descriptor multiplication (employing the Shuffle and Split algorithms), computation of both stationary and transient probability distributions of states (using the Power, Arnoldi and GMRES methods in the former case, as well as the Uniformization method in the latter one), derivation of a symbolic solution, RSS generation and automata grouping.

5.1.3. Applications of Tensor Networks

Recent advances in TNs have proven their usefulness in a broad area of scientific disciplines, ranging from physics (including, among others, condensed matter physics, atomic physics and statistical physics) [37], through quantum chemistry [37, 30], up to quantum information science (to mention only quantum teleportation and purification of quantum states) [7]. The flexibility of this formalism allows to simulate efficiently systems of both finite and infinite size, with different boundary conditions and underlying structures [30]. It turns out that multiple issues originated from the aforementioned domains of science may be expressed according to the TNs formalism and solved by reduction of the initial problem to the matter of tensor contractions [37].

The main advantage of the TNs formalism is an ability to break down multidimensional structures into a set of interconnected tensors of smaller size, each of which representing a part of the original object. This concept is often informally referred as "quantum LEGOs" [5, 30], where tensors are interpreted as bricks which build overall structure of the system under simulation, while quantum entanglement acts as a cohesive force between them. Naturally there are many ways to decompose a multidimensional tensor into a TN, just to name a few: Matrix Product States (MPS, which builds a one-dimensional sequence of tensors), Tensor Train (TT, which is a variant of the MPS with open boundary conditions), Projected Entangled Pair States (PEPS, which builds a two-dimensional grid of tensors), Tree Tensor Networks (TTN) and Multi-scale Entanglement Renormalization Ansatz (MERA, which both build a hierarchical tree structure of tensors) [7, 30, 37]. An example of numerous ways to represent a multidimensional tensor is presented in the Figure 5.1.

The fundamental operation which may be performed on TNs is the contraction of corresponding indices belonging to two ore more tensors [30]. There are multiple algorithms to be employed in order to perform an effective contraction of a set of tensors, however a description of them exceeds the scope of this Section (a quite comprehensive overview on this topic is presented in [32, 37]). Nevertheless, it should be noted here that choosing the right algorithm, as well as determining the optimal order of contraction for multiple tensors are usually non-trivial tasks [30, 32].

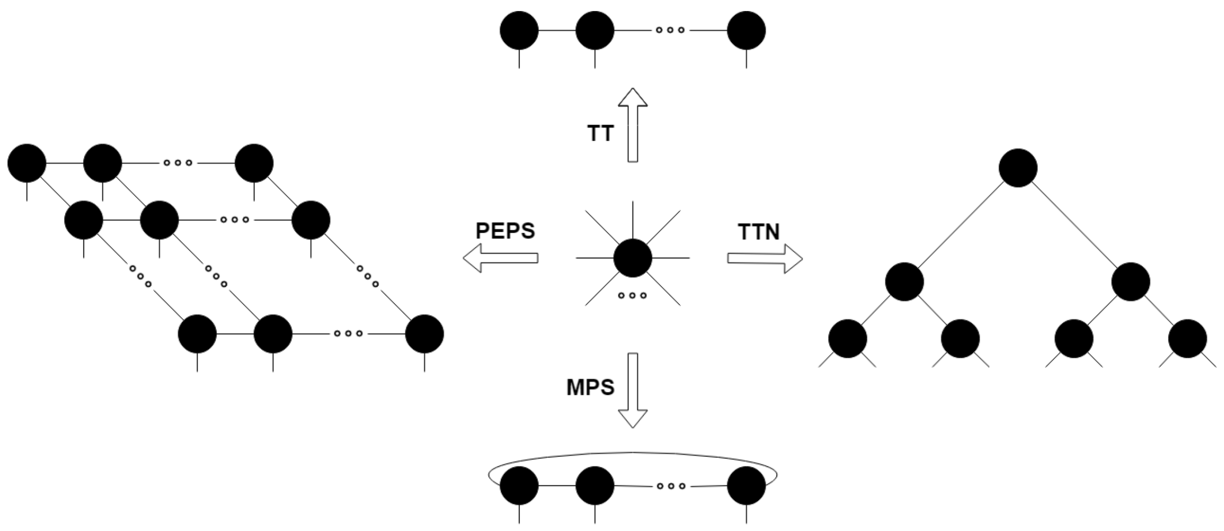


Figure 5.1. Multiple possibilities of the tensor decomposition.

5.2. Problem statement

As already presented in the state of the art analysis (cf. Section 5.1), there is basically no numerical method designed for computing transient probability distributions of states belonging to CTMCs expressible in terms of the SAN formalism, which employ the Kronecker algebra-based structure of the transition rate matrix describing the evolution of such systems. Therefore, this research is emphasized on filling this gap by the elaboration and examination of the new algorithm, called TNSAN, which takes advantage of both the Markovian models formalism and the advanced matrix Algebra in order to establish its theoretical background, as well as utilizes the concept of TNs in order to simplify numerical computations and enhance its expression power by providing a possibility to build circuit-like structures facilitating its comprehension.

In order to determine a transient probability distribution of states, the method of computing an exponential of a transition rate matrix is employed. Therefore, the formal derivation of the proposed method firstly explains how to efficiently compute the exponential of a structured representation of a matrix, as a sum of tensor products. Simultaneously, numerous mathematical theorems are applied in order to significantly facilitate numerical computations to be carried out. Then, the algorithm of building a TN reflecting the modified structure of the matrix exponential is presented, and finally it is shown how to perform contractions on such a network in order to obtain desired transient probability distribution. Further parts of this thesis are focused on the "twin-track" evaluation of the TNSAN algorithm – on the one hand a formal analysis of its properties is carried out, and on the other its implementation for particular benchmark problems is evaluated in terms of classical quantitative measures (such as computation time and results accuracy).

The TNSAN algorithm is expected primarily to provide a new way of constructing numerical algorithms dedicated to the SAN formalism. This method connects two, separated so far, worlds – well-established and thoroughly investigated one belonging to Markovian models and rapidly evolving, increasingly used one of TNs. Undeniable benefits flowing from this method to SAN models are: a completely fresh, previously unexplored approach defining new boundaries for methods of their simulation and an additional possibility to overcome the curse of dimensionality issue, so much associated with the state space explosion problem. Furthermore, thanks to this research the TNs formalism increases its prosperity by proven applicability in the field of stochastic processes simulation.

5.3. Research questions and hypotheses

In order to precisely describe objectives of this research, as well as to delineate its expected results, three research questions have been formulated. Each of these questions is also supplemented by an answer given a priori, acting as a research hypothesis. The hypotheses are going to be verified throughout this thesis by achieving specified research goals.

Question 1. Is it possible to build a TN structure describing the evolution of any SAN in time?

Hypothesis 1. Yes, it is. The TNSAN algorithm exploits the modular representation of a transition rate matrix (describing the evolution of any SAN in time) in order to build a TN, which contraction results in obtaining the exponential of the aforementioned matrix.

The following research goals are specified to be achieved in order to verify the truthfulness of the hypothesis above:

1. elaborate a comprehensive collection of mathematical facts concerning foundations of TNs and SANs to be used when performing a formal derivation and justification of possible research outcomes;
2. propose an efficient way to compute an exponential of a transition rate matrix making use of its structured representation based on the Kronecker algebra;
3. build a TN structure representing the evolution of the model under simulation, based on the structure obtained above.

Question 2. Is it possible to study transient probability distributions of any SAN taking advantage of the aforementioned TN describing the evolution of the simulated model?

Hypothesis 2. Yes, it is. The TNSAN algorithm builds another TN representing transient probability distributions of states belonging to all constituent automata building up the SAN under simulation. The contraction between the network of state and evolution results in obtaining the desired global distribution in any requested moment of time.

Within this hypothesis, the following research goals are planned to be accomplished:

1. build a TN structure representing the initial probability distributions of states belonging to all constituent automata composing the model under simulation (both when this distribution is given by a collection of components' distributions, as well as when it is given by a global joint probability distribution over all the automata);
2. define the contraction pattern for the TNs of initial state and evolution, which leads to obtaining a transient probability distribution of states of the model under simulation;

Question 3. What are the numerical properties of the proposed method, particularly in terms of results accuracy and computational efficiency?

Hypothesis 3. The TNSAN algorithm performs numerical simulations in an efficient manner, with polynomial time complexity and memory requirements proportional to the sum of squares of the number of states belonging to all individual automata composing the network under simulation. Moreover, the proposed method allows to perform the computations with desired accuracy, affecting the computation time (the more precise results, the longer simulation time). Additionally, the TNSAN algorithm handles SAN models with multiple unreachable states and numerous synchronizing events easily.

The verification of this hypothesis is going to be performed by meeting the following research goals:

1. conduct a qualitative analysis of the TNSAN algorithm, especially in terms of memory efficiency and time performance;
2. choose some existing scalable and diversified benchmark problems, and then implement the TNSAN algorithm for them;
3. perform a quantitative analysis of the algorithm's properties – especially in case of computation time and results accuracy – compared to an already known method of computing transient probability distributions of SAN states.

5.4. Summary

Within this Chapter the research problem was defined formally. Firstly, current problems and issues in the matter of SANs and TNs were described, with emphasis on: numerical methods for determining both stationary and transient distributions of any SAN states, existing software and libraries implementing the aforementioned algorithms and fundamental advances in the area of TNs. Within the context of this state of the art analysis, the research problem was exhaustively described and followed by the specification of research questions, hypotheses (being expected answers to the research questions) and goals to be achieved in order to confirm or deny them. The following Chapter 6 describes the methodology chosen to reach all the aforementioned objectives, while the results of successive goals accomplishment are collected in the Chapters 7 and 8, and then summarized and reviewed in the Chapter 9.

6. Methodology

This Chapter starts with a presentation of the research method adopted for the development of this thesis (Section 6.1). Then, the evaluation strategy applied to assess the outcomes resulting from this research is outlined (Section 6.2). Finally, a collection of tools used to achieve the aforementioned research goals is described (Section 6.3).

6.1. Research method

A methodology selected for the development of this thesis is grounded in the classic scientific research method, which mainly put emphasis on building mathematical foundations of observable phenomena throughout its empirical investigation and verification of hypotheses. This methodology is inherently designed to work with formal models and numerical data, as it happens here. The main tools used to carry out the quantitative research nowadays are mathematics (with special regard to statistics) and computational techniques.
In order to comply with the requirements of the scientific research method, numerous actions should be taken. Firstly, the subject matter of research has to be strictly defined in terms of research questions and hypotheses (cf. Sections 5.2 and 5.3) arising from the observation of current state of affairs (cf. Sections 1.1 and 5.1). Then, appropriate instruments need to be developed (cf. Section 6.3 and Chapters 2 to 4 and 7) and impartial methods for the evaluation of results are required to be elaborated (cf. Section 6.2) in order to verify stated hypotheses. Afterwards, the experiments must be performed for selected diversified benchmark problems (cf. Chapter 8), and finally their results has to be collected, evaluated and objectively discussed (cf. Chapter 9).

As already mentioned, one of the very basic purposes of computing transient probability distributions of SAN models is studying the properties of distributed systems, which is notably useful when there is no possibility to perform real-world experiments on such systems. Then – according to Kleiber [25], who pointed out three pillars of contemporary science: theory, experiment and computer simulation, as well as their interplay – one may employ formal mathematical models together with numerical simulations in order to get to know the characteristics of the system under examination. Thus, this research is emphasized on elaborating a numerical algorithm for the simulation of SANs, which is intended to be used in determining transient properties of highly distributed systems. The TNSAN algorithm is built on top of reliable mathematical theories, which are the foundations of the proposed method correctness and also make its results interpretable.

6.2. Evaluation strategy

Most of all, the proposed method is going to be analyzed theoretically by proving its correctness, as well as studying its memory requirements and time efficiency. It should be noted, however, that determining exact time and memory complexities of the proposed algorithm is strictly dependent on the particular problem being solved (cf. Section 7.4), therefore only general remarks in this topic are provided within this thesis.

Moreover, the TNSAN algorithm is going to be implemented and preliminary evaluated in terms of a modified resource sharing benchmark problem [3]. This model simulates a typical situation encountered in distributed systems, when N distinguishable processes compete for access to the shared resource which allows only P processes to use it concurrently. The relation between numbers N and P affects the participation of the RSS in the PSS (the smaller the difference between N and P, the bigger size of the RSS).
Main aspects of the numerical evaluation which are going to be examined are: computation time and results accuracy. The latter quantity is going to be determined in terms of related approach used to calculate transient probability distributions of states belonging to CTMCs, based on computing an exponential of a transition rate matrix taking advantage of methods designed for large sparse matrices (cf. Section 5.1). Due to the expectation that the TNSAN algorithm returns more accurate results as the number of its iterations increases, the evaluation should be primarily focused on a trade-off between method's performance and results quality. Additionally, the proposed approach is naturally more memory-saving than storing the explicit form of a transition rate matrix, but this advantage usually implies less time efficiency, therefore the memory versus CPU time trade-off should be investigated while performing the comparison between the two algorithms discussed.

6.3. Tools overview

First and foremost, this thesis employs the mathematical apparatus behind the concepts of CTMCs, SANs and TNs, with emphasis on Algebra and Probability theory, in order to describe formal assumptions of the TNSAN algorithm, as well as to perform a theoretical analysis of its properties. Considering implementation details, the TNSAN algorithm does not put any restrictions on the choice of neither data structures and numerical methods, nor programming languages, frameworks and environments used to turn the mathematical formulation into code. Regarding the implementation carried out for the needs of this thesis, the description of adopted data structures and numerical methods is presented in Chapter 7, while the list of employed software tools is given later in this Section.

For the needs of the implementation within this thesis, the `Julia` [4] language has been chosen. This tool incorporates simplicity of scripting languages like Python together with powerful underneath known from C and Fortran languages. Moreover, Julia is equipped with an extensive standard mathematical library backed by high-performance implementations, as well as enormous open-source support for numerous algorithms and data structures, including TNs and their manipulations. Finally, this language natively supports parallel and distributed computing with high-level mechanisms for processes and threads management. That's way Julia is considered as one of the most powerful modern programming languages designed with a view to performing numerical simulations in highly distributed, heterogeneous environments like supercomputers, grids and clouds.
Apart from rich-functionality standard library of Julia, the additional open-source software library `TensorOperations.jl`[1] supporting the manipulation of TNs is used. Moreover, the Julia wrapper for `Plotly`[2] plotting library is employed. An implementation of numerical algorithms for computing exponentials of large sparse matrices is provided by the Julia wrapper for `EXPOKIT`[3] [41].

6.4. Summary

This Chapter discussed the methodology chosen to develop the thesis and evaluate its outcomes. It began with a description of the scientific research method and the theory-experiment-simulation triad, which constitute the skeleton for activities undertaken within this research. Then, the process of results evaluation (both formally and numerically) was presented, including a benchmark problem description and measurable parameters specification. Finally, a set of tools – regarding both mathematical apparatus and software implementation – to be employed in order to accomplish the specified research goals was provided. The following two Chapters 7 and 8 describe the adoption of selected methodology to the area of this research.

[1]`https://github.com/Jutho/TensorOperations.jl` (access: 3 May 2018)
[2]`https://github.com/sglyon/PlotlyJS.jl` (access: 3 May 2018)
[3]`https://github.com/acroy/Expokit.jl` (access: 3 May 2018)

7. Algorithm for the simulation of Stochastic Automata Networks with Tensor Networks

This Chapter presents the main contribution of this thesis – theoretical foundations of the TNSAN algorithm. It begins with a formal derivation of the proposed approach (Section 7.1), which is supplemented by a high-level description and a pseudocode of the simulation algorithm (Section 7.2). Then, multiple explanatory remarks are provided in the Section 7.3 and some notes on the method's performance are given in the Section 7.4. Please note that the symbols used within this Chapter follow the convention adopted in previous Chapters, therefore they are not introduced formally again.

7.1. Method derivation

The main goal of the TNSAN algorithm is to determine a transient probability distribution $\mathbf{p}_{\mathcal{N}}(t)$ of states belonging to any SAN model $\boldsymbol{\mathcal{N}} = (\mathscr{A}_{\boldsymbol{\mathcal{N}}}, \mathscr{E}_{\boldsymbol{\mathcal{N}}})$ with $\mathscr{A}_{\boldsymbol{\mathcal{N}}} = \{\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(N_{\boldsymbol{\mathcal{N}}})}\}$ and $\mathscr{E}_{\boldsymbol{\mathcal{N}}} = \{e^{(1)}, e^{(2)}, \ldots, e^{(E_{\boldsymbol{\mathcal{N}}})}\}$ at arbitrarily given instant of time t. To achieve this goal, the method proposed below employs exceptional features of the SAN formalism and then facilitates complex algebraic calculations by exploiting the capabilities of TNs. The following steps describe comprehensively formal foundations of the novel approach.

1. According to the Equation 2.17, desired probability distribution $\mathbf{p}_{\mathcal{N}}(t)$ at arbitrary time t is a product of the initial probability distribution $\mathbf{p}_{\mathcal{N}}(0)$ and the probability matrix $P(t)$ resulting from Kolmogorov equations (cf. Theorem 8 (under the assumption that the simulated model is Markovian, which is naturally the case here):

   $$\mathbf{p}_{\mathcal{N}}(t) = \mathbf{p}_{\mathcal{N}}(0) \cdot P(t).$$

   Furthermore, following the Theorem 9, one may immediately observe that the probability matrix in question is given by an exponential of the infinitesimal generator matrix $Q_{\mathcal{N}}$ associated with the SAN $\boldsymbol{\mathcal{N}}$ (cf. Section 2.2.1), multiplied by the scalar value t:

   $$P(t) = \exp\left(t \cdot Q_{\mathcal{N}}\right).$$

2. Considering the Theorem 10, the matrix $Q_{\mathcal{N}}$ of any SAN model may be written as a sum of $N_{\boldsymbol{\mathcal{N}}} + 2 \cdot E_{\boldsymbol{\mathcal{N}}}$ Kronecker products, each of which consisting of $N_{\boldsymbol{\mathcal{N}}}$ factors, in the following way:

$$Q_{\mathcal{N}} = \bigoplus_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} Q^{loc}_{\mathcal{A}^{(k)}} + \sum_{e \in \mathscr{E}^{syn}_{\boldsymbol{\mathcal{N}}}} \left[\bigotimes_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} Q^{e_{pos}}_{\mathcal{A}^{(k)}} + \bigotimes_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} Q^{e_{neg}}_{\mathcal{A}^{(k)}} \right].$$

Therefore, the matrix $P(t)$ for the SAN model $\boldsymbol{\mathcal{N}}$ is given explicitly by:

$$P(t) = \exp\left(t \cdot \bigoplus_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} Q^{loc}_{\mathcal{A}^{(k)}} + \sum_{e \in \mathscr{E}^{syn}_{\boldsymbol{\mathcal{N}}}} \left[t \cdot \bigotimes_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} Q^{e_{pos}}_{\mathcal{A}^{(k)}} + t \cdot \bigotimes_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} Q^{e_{neg}}_{\mathcal{A}^{(k)}} \right] \right).$$

3. The summands constituting the descriptor form of $Q_{\mathcal{N}}$ usually do not commute, therefore the Lie product formula (alternatively called the Suzuki-Trotter expansion of the first order) given by the Theorem 4 needs to be applied to transform the exponential of the sum of matrices to the product of their exponentials:

$$P(t) = \lim_{n \to \infty} \left(\exp\left(\Delta t \cdot \bigoplus_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} Q^{loc}_{\mathcal{A}^{(k)}} \right) \cdot \prod_{e \in \mathscr{E}^{syn}_{\boldsymbol{\mathcal{N}}}} \left[\exp\left(\Delta t \cdot \bigotimes_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} Q^{e_{pos}}_{\mathcal{A}^{(k)}} \right) \cdot \exp\left(\Delta t \cdot \bigotimes_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} Q^{e_{neg}}_{\mathcal{A}^{(k)}} \right) \right] \right)^n,$$

where: $\Delta t = \frac{t}{n}$.

This representation, however, is not applicable to the numerical computations because it requires to calculate a limit of a function at infinity. Therefore, according to the definition of limit, it may be assumed that for sufficiently large, finite value of $n \in \mathbb{N}$, the following formula "nearly" (or "approximately", what is indicated by the symbol $\cong$) holds:

$$P(t) \cong \left(\exp\left(\Delta t \cdot \bigoplus_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} Q^{loc}_{\mathcal{A}^{(k)}} \right) \cdot \prod_{e \in \mathscr{E}^{syn}_{\boldsymbol{\mathcal{N}}}} \left[\exp\left(\Delta t \cdot \bigotimes_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} Q^{e_{pos}}_{\mathcal{A}^{(k)}} \right) \cdot \exp\left(\Delta t \cdot \bigotimes_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} Q^{e_{neg}}_{\mathcal{A}^{(k)}} \right) \right] \right)^n.$$

Please note that an approximation error is deliberately introduced to the algorithm right now. Furthermore, still utilizing the definition of limit, one may expect that the bigger value of n, the less this error is. Naturally, the more iterations the algorithm has, the longer time its execution takes, therefore the results quality versus method's performance trade-off mentioned in the Section 6.2 has to be considered.

4. The attentive Reader should notice that arguments of the matrix exponentials still contains $S^2_{\mathcal{N}}$ elements (where $S_{\mathcal{N}}$ is the PSS size of the SAN $\boldsymbol{\mathcal{N}}$), thus there are still no benefits from the representation obtained so far, in comparison to the straightforward calculation of the explicitly given matrix $Q_{\mathcal{N}}$ exponential, employing algorithms designed for spare structures. Therefore, the following two observations should be applied, considering local and synchronized transitions separately.

Firstly, according to the Theorem 5, the exponential of the Kronecker sum of matrices may be effectively written in terms of a Kronecker product of their exponentials. Then, the probability matrix discussed is given by:

$$P(t) \cong \left(\bigotimes_{k=1}^{N_{\mathcal{N}}} \exp\left(\Delta t \cdot Q_{\mathcal{A}^{(k)}}^{loc}\right) \cdot \prod_{e \in \mathscr{E}_{\mathcal{N}}^{syn}} \left[\exp\left(\Delta t \cdot \bigotimes_{k=1}^{N_{\mathcal{N}}} Q_{\mathcal{A}^{(k)}}^{e_{pos}}\right) \cdot \exp\left(\Delta t \cdot \bigotimes_{k=1}^{N_{\mathcal{N}}} Q_{\mathcal{A}^{(k)}}^{e_{neg}}\right) \right] \right)^n .$$

Now, the exponential of descriptor matrices corresponding to local transitions occurring within the network $\mathcal{N}$ is represented in a compact form composed of dense, low-size constituent matrices. This approach naturally reduces the amount of resources (both time and memory) required to compute the matrix exponential.

On the other hand, the exponential of a Kronecker product is not as such easily breakable as is the case for the Kronecker sum. In general, an exponential of a Kronecker product of matrices may be represented in terms of a Kronecker product of their exponentials when the constituent factors are mostly identities. Fortunately, this is the case here.

Now consider a form of matrices $Q_{\mathcal{A}^{(k)}}^{e_{pos}}$ and $Q_{\mathcal{A}^{(k)}}^{e_{neg}}$ for any automaton $\mathcal{A}^{(k)}$ and an arbitrarily given event $e \in \mathscr{E}_{\mathcal{N}}^{syn}$, presented comprehensively in the Section 3.2.2. Recalling the remarks provided there, one may observe that the aforementioned matrices are non-identity ones if the automaton $\mathcal{A}^{(k)}$ is affected by the synchronizing event e. In most practical cases, any synchronizing event affects usually a few automata, which constitute insignificant part of the whole system. This, in turn, implies that a vast majority of matrices $Q_{\mathcal{A}^{(k)}}^{e_{pos}}$ and $Q_{\mathcal{A}^{(k)}}^{e_{neg}}$ are identities, therefore it is possible to significantly facilitate the calculation of a matrix exponential of a Kronecker product of matrices using the following ideas (presented below). Nevertheless, one should notice the reduced scope of applicability of the TNSAN algorithm to models, which synchronized transitions do not affect a majority of components being a part of the system under simulation.

Consider two automata (the generalization for more of them is straightforward) $\mathcal{A}^{(i)}$ and $\mathcal{A}^{(j)}$, such that $1 \leq i < j \leq N_{\mathcal{N}}$, exclusively affected by a synchronizing event e. Then, the product form of the matrix $P(t)$ presented above contains the following factor:

$$\exp\left(\Delta t \cdot \left(\bigotimes_{k=1}^{i-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes Q_{\mathcal{A}^{(i)}}^{e_{pos}} \otimes \left(\bigotimes_{k=i+1}^{j-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes Q_{\mathcal{A}^{(j)}}^{e_{pos}} \otimes \left(\bigotimes_{k=j+1}^{N_{\mathcal{N}}} I_{S_{\mathcal{A}^{(k)}}}\right)\right) \cdot$$
$$\exp\left(\Delta t \cdot \left(\bigotimes_{k=1}^{i-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes Q_{\mathcal{A}^{(i)}}^{e_{neg}} \otimes \left(\bigotimes_{k=i+1}^{j-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes Q_{\mathcal{A}^{(j)}}^{e_{neg}} \otimes \left(\bigotimes_{k=j+1}^{N_{\mathcal{N}}} I_{S_{\mathcal{A}^{(k)}}}\right)\right) ,$$

for non-identity matrices $Q^{e_{pos}}_{\mathcal{A}^{(i)}}$, $Q^{e_{pos}}_{\mathcal{A}^{(j)}}$, $Q^{e_{neg}}_{\mathcal{A}^{(i)}}$ and $Q^{e_{neg}}_{\mathcal{A}^{(j)}}$. Furthermore, by applying Equations 2.14 and 2.15, in conjunction with the Kronecker product properties given within the Theorem 1, one may immediately obtain:

$$\left(\bigotimes_{k=1}^{i-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes \exp\left(\Delta t \cdot Q^{e_{pos}}_{\mathcal{A}^{(i)}} \otimes \left(\bigotimes_{k=i+1}^{j-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes Q^{e_{pos}}_{\mathcal{A}^{(j)}}\right) \otimes \left(\bigotimes_{k=j+1}^{N_{\mathcal{N}}} I_{S_{\mathcal{A}^{(k)}}}\right) \cdot$$
$$\left(\bigotimes_{k=1}^{i-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes \exp\left(\Delta t \cdot Q^{e_{neg}}_{\mathcal{A}^{(i)}} \otimes \left(\bigotimes_{k=i+1}^{j-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes Q^{e_{neg}}_{\mathcal{A}^{(j)}}\right) \otimes \left(\bigotimes_{k=j+1}^{N_{\mathcal{N}}} I_{S_{\mathcal{A}^{(k)}}}\right).$$

Finally, denote the perfect shuffle permutation matrix $S_{S_{\mathcal{A}^{(j)}} \cdot (\prod_{k=i+1}^{j-1} S_{\mathcal{A}^{(k)}}), S_{\mathcal{A}^{(i)}}}$ by R, while its inverse matrix by R^{-1}. Then, by employing the Equation 2.16, the following formula holds (please note the permutation of matrices being the arguments of the exponential operation):

$$\left(\bigotimes_{k=1}^{i-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes R \cdot \left[\left(\bigotimes_{k=i+1}^{j-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes \exp\left(\Delta t \cdot Q^{e_{pos}}_{\mathcal{A}^{(j)}} \otimes Q^{e_{pos}}_{\mathcal{A}^{(i)}}\right)\right] \cdot R^{-1} \otimes \left(\bigotimes_{k=j+1}^{N_{\mathcal{N}}} I_{S_{\mathcal{A}^{(k)}}}\right) \cdot$$
$$\left(\bigotimes_{k=1}^{i-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes R \cdot \left[\left(\bigotimes_{k=i+1}^{j-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes \exp\left(\Delta t \cdot Q^{e_{neg}}_{\mathcal{A}^{(j)}} \otimes Q^{e_{neg}}_{\mathcal{A}^{(i)}}\right)\right] \cdot R^{-1} \otimes \left(\bigotimes_{k=j+1}^{N_{\mathcal{N}}} I_{S_{\mathcal{A}^{(k)}}}\right).$$

Moreover, one may easily observe that:

$$\left(\bigotimes_{k=1}^{i-1} I_{S_{\mathcal{A}^{(k)}}}\right)^2 = \left(\bigotimes_{k=1}^{i-1} I_{S_{\mathcal{A}^{(k)}}}\right) \quad \text{and} \quad \left(\bigotimes_{k=j+1}^{N_{\mathcal{N}}} I_{S_{\mathcal{A}^{(k)}}}\right)^2 = \left(\bigotimes_{k=j+1}^{N_{\mathcal{N}}} I_{S_{\mathcal{A}^{(k)}}}\right),$$

as well as:

$$R \cdot \left[\left(\bigotimes_{k=i+1}^{j-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes \exp\left(\Delta t \cdot Q^{e_{pos}}_{\mathcal{A}^{(j)}} \otimes Q^{e_{pos}}_{\mathcal{A}^{(i)}}\right)\right] \cdot R^{-1} \cdot$$
$$R \cdot \left[\left(\bigotimes_{k=i+1}^{j-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes \exp\left(\Delta t \cdot Q^{e_{neg}}_{\mathcal{A}^{(j)}} \otimes Q^{e_{neg}}_{\mathcal{A}^{(i)}}\right)\right] \cdot R^{-1} =$$
$$R \cdot \left[\left(\bigotimes_{k=i+1}^{j-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes \exp\left(\Delta t \cdot Q^{e_{pos}}_{\mathcal{A}^{(j)}} \otimes Q^{e_{pos}}_{\mathcal{A}^{(i)}}\right)\right] \cdot$$
$$\left[\left(\bigotimes_{k=i+1}^{j-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes \exp\left(\Delta t \cdot Q^{e_{neg}}_{\mathcal{A}^{(j)}} \otimes Q^{e_{neg}}_{\mathcal{A}^{(i)}}\right)\right] \cdot R^{-1},$$

since $R^{-1} \cdot R$ results in the identity matrix. Therefore, by employing associativity and compatibility with ordinary matrix multiplication properties of the Kronecker product, the definitive form of the exponential calculated for matrices related to synchronized transitions is given by:

$$\left(\bigotimes_{k=1}^{i-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes R \cdot \left[\left(\bigotimes_{k=i+1}^{j-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes \exp\left(\Delta t \cdot Q^{e_{pos}}_{\mathcal{A}^{(j)}} \otimes Q^{e_{pos}}_{\mathcal{A}^{(i)}}\right)\right] \cdot$$
$$\left[\left(\bigotimes_{k=i+1}^{j-1} I_{S_{\mathcal{A}^{(k)}}}\right) \otimes \exp\left(\Delta t \cdot Q^{e_{neg}}_{\mathcal{A}^{(j)}} \otimes Q^{e_{neg}}_{\mathcal{A}^{(i)}}\right)\right] \cdot R^{-1} \otimes \left(\bigotimes_{k=j+1}^{N_{\mathcal{N}}} I_{S_{\mathcal{A}^{(k)}}}\right).$$

It should be also pointed out, referring again to the Section 3.2.2, that negative matrices are diagonal, therefore the exponential of their Kronecker product is trivially given by calculating the algebraic exponents of its diagonal elements.

5. When the final form of the matrix $P(t)$ is obtained, a TN facilitating its computation should be created according to the rules presented throughout the Section 4.2. The Reader familiar with quantum computing should notice that the approach undertaken to build tensor structures for the needs of the TNSAN algorithm is similar to the rules governing the construction of quantum circuits.

   The matrices corresponding to local transitions contribute to the product form of the matrix $P(t)$ with the Kronecker product of their exponentials. According to the Figure 4.3a, these exponentials are presented in a TND as separate, rank-2 tensors. Furthermore, the ordinary matrix multiplication is given visually by two tensors connected with common index, as shown in the Figure 4.4a. Finally, any matrix corresponding to an interaction with multiple automata is presented by a multirank tensor, as presented in the Figure 4.3b. A graphical representation of identity matrices may be omitted because they constitute the neutral element of ordinary matrix multiplication, therefore it is not necessary to carry out computations on them.

   As presented in the Step 1, the desired probability distribution $\mathbf{p}_{\mathcal{N}}(t)$ is given by a product of the initial probability distribution of states $\mathbf{p}_{\mathcal{N}}(0)$ and the matrix $P(t)$. Therefore, apart from the tensor representation of the probability matrix $P(t)$, it is necessary to express the initial probability distribution vector in terms of TNs. There are generally two cases here. Firstly, the initial probability distribution may be given separately for each component being a part of the system under simulation. Then, the global initial probability distribution is given by a Kronecker product of such vectors, which may be inherently represented by a set of independent rank-1 tensors. On the other hand, the joint initial probability distribution may be given holistically for the whole system, which is equivalent to a high-rank tensor. Then, in order to extract a probability distribution for each constituent separately, one may employ a tensor decomposition algorithm such as Tensor Train (cf. Definition 22).

   Considering the TND diagram representation, from the TN of initial state naturally emerge $N_{\mathcal{N}}$ lines, each one corresponding to a constituent automaton composing the SAN $\mathcal{N}$ under simulation. Then, each matrix exponential belonging to the TN of evolution and affecting the i-th component, $1 \leq i \leq N_{\mathcal{N}}$, should be effectively connected to the i-th line emerging from the aforementioned TN of state. This concept is further clarified by the Example 8 presenting sample tensor structure utilized by the TNSAN algorithm.

6. To sum up, the TNSAN algorithm employs two TNs to compute a transient probability of states belonging to the simulated SAN at arbitrary time t. Firstly, the TN of initial state is built using either initial probability distribution of states belonging to constituent automata or a Tensor Train decomposition of the joint initial probability distribution given for the whole system. Then, the TN of evolution is built making use of the efficient representation of the probability matrix $P(t)$

summarized by the Step 4 within this Section. The TN of state of the simulated system at time Δt is obtained by performing a contraction between the two aforementioned networks, maintaining the form of the TN of state (in particular, the auxiliary indices should not be contracted when the TN of state is given by the Tensor Train format). Finally, this contraction process is repeated iteratively n times (because the matrix $P(t)$ is given by the n-th power of the algebraic structure corresponding to the utilized TN) and results in obtaining a transient probability distribution of states at desired instant of time t.

Example 8. Summarizing the considerations above, an example of the TN structures exploited by the TNSAN algorithm is given in the Figure 7.1. Presented network is composed of the TN of initial state (marked in orange) in the Tensor Train format and single TN of evolution (marked in black), employing one synchronized transition involving components $\mathcal{A}^{(2)}$ and $\mathcal{A}^{(4)}$. Please note that the parameter Δt is omitted for clarity reasons and it is assumed that the matrices presented in the TND below have been already multiplied by this constant.

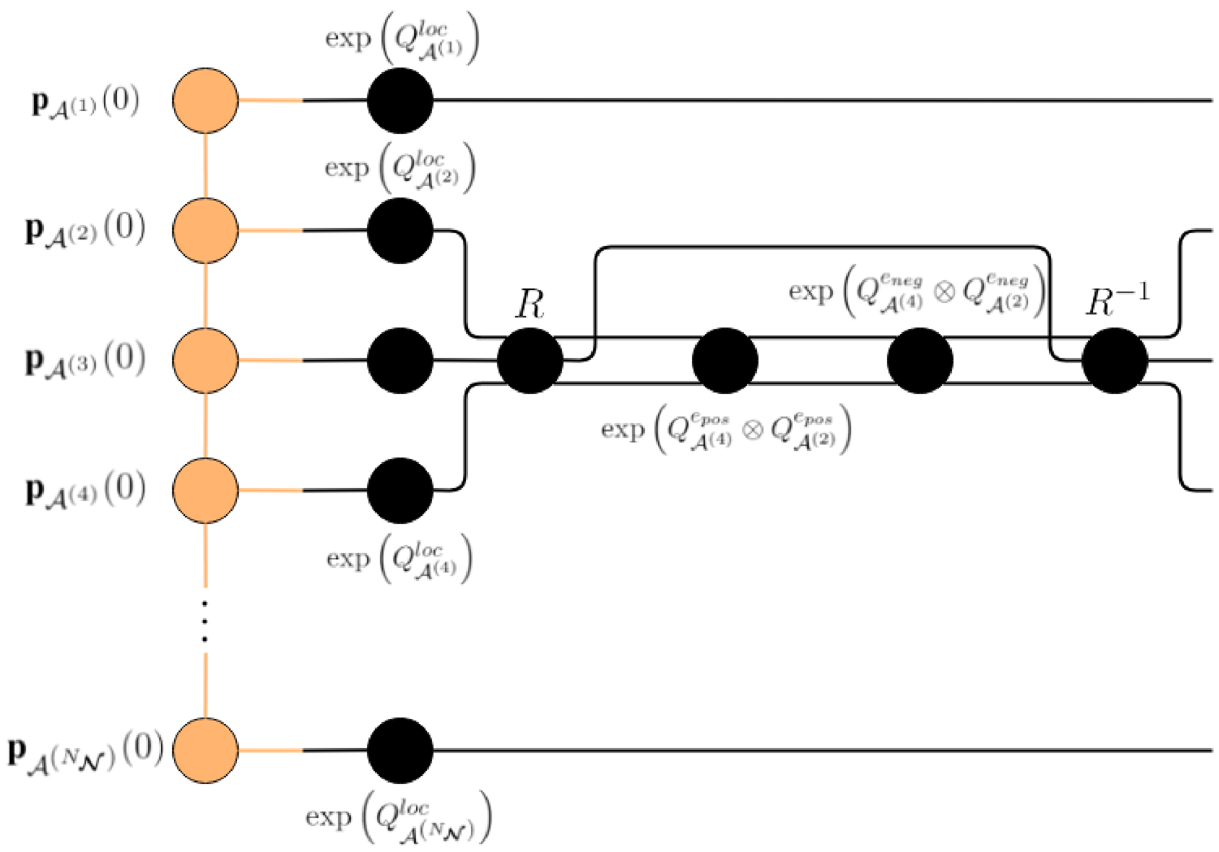

$\mathbf{p}_{\mathcal{A}^{(1)}}(0)$
$\mathbf{p}_{\mathcal{A}^{(2)}}(0)$
$\mathbf{p}_{\mathcal{A}^{(3)}}(0)$
$\mathbf{p}_{\mathcal{A}^{(4)}}(0)$
$\mathbf{p}_{\mathcal{A}^{(N_\mathcal{N})}}(0)$
$\exp\left(Q^{loc}_{\mathcal{A}^{(1)}}\right)$
$\exp\left(Q^{loc}_{\mathcal{A}^{(2)}}\right)$
$\exp\left(Q^{loc}_{\mathcal{A}^{(4)}}\right)$
$\exp\left(Q^{loc}_{\mathcal{A}^{(N_\mathcal{N})}}\right)$
R
R^{-1}
$\exp\left(Q^{e_{neg}}_{\mathcal{A}^{(4)}} \otimes Q^{e_{neg}}_{\mathcal{A}^{(2)}}\right)$
$\exp\left(Q^{e_{pos}}_{\mathcal{A}^{(4)}} \otimes Q^{e_{pos}}_{\mathcal{A}^{(2)}}\right)$

Figure 7.1. Sample tensor structure employed by the TNSAN algorithm.

7.2. Simulation algorithm

Please note that all numerical data structures used to implement the following pseudocode are suggested to be floating-point ones. Within the implementation carried out for the needs of this thesis, all numbers are stored and managed as `Float64`-type variables built-in the `Julia` language.

The TNSAN simulation algorithm takes the following input:

- specification of the SAN model $\boldsymbol{\mathcal{N}}$ to be simulated in the form of its Ordinary Tensor Algebra (i.e. plain Kronecker algebra) descriptor, incorporating only local and synchronized transitions and denoted by $Q_{\mathcal{N}}$;
- initial probability distribution $\mathbf{p}_{\mathcal{N}}(0)$ of states belonging to the SAN under simulation, given either by a multidimensional tensor of probabilities or a set of probability vectors, one per each constituent of the network $\boldsymbol{\mathcal{N}}$;
- single floating-point number t representing the time instant, when the transient probability of states should be computed;
- single integer n representing the number of iterations of the algorithm, given a priori, acting as a method's parameter.

Then, the TNs of initial state $\mathbb{S}^0_{\boldsymbol{\mathcal{N}}}$ and evolution $\mathbb{E}^{\Delta t}_{\boldsymbol{\mathcal{N}}}$ over time Δt are built according to the rules presented in the Section 7.1. Finally, for each iteration $k \in \mathbb{N}$, $1 \leq k \leq n$, the TN of state at time $k \cdot \Delta t$, denoted by $\mathbb{S}^{k \cdot \Delta t}_{\boldsymbol{\mathcal{N}}}$, is obtained by the contraction of the TN of state at time $(k-1) \cdot \Delta t$ (obtained in previous iteration, or the initial one for $k = 1$) with the TN of evolution (remaining constant during the execution of the algorithm), as described in the Section 7.1.
The output of the TNSAN algorithm is, in fact, the TN of state $\mathbb{S}^t_{\boldsymbol{\mathcal{N}}}$ at desired time t. In order to obtain a joint probability distribution of the PSS of the simulated SAN $\boldsymbol{\mathcal{N}}$ one has to either perform a contraction of the output network (if it is given in a Tensor Train format) or compute the Kronecker product of final vectors (if the initial state was given separately for each component of the model under simulation).

The high-level pseudocode of the TNSAN algorithm is presented in the Listing 7.1. Please note that declarations of data types are omitted for all variables, because they are comprehensively discussed in the description above.

```
procedure TNSAN($Q_{\mathcal{N}}$, $\mathbf{p}_{\mathcal{N}}(0)$, $t$, $n$):
    $\Delta t := \frac{t}{n}$

    if $\mathbf{p}_{\mathcal{N}}(0)$ is joint probability distribution:
        $\mathbb{S}^0_{\boldsymbol{\mathcal{N}}}$ := TensorTrain($\mathbf{p}_{\mathcal{N}}(0)$)
    else:
        $\mathbb{S}^0_{\boldsymbol{\mathcal{N}}} := \mathbf{p}_{\mathcal{N}}(0)$
    $\mathbb{E}_{\boldsymbol{\mathcal{N}}}^{\Delta t} := \bigotimes_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} \exp\left(\Delta t \cdot Q_{\mathcal{A}^{(k)}}^{loc}\right) \cdot \prod_{e \in \mathscr{E}_{\boldsymbol{\mathcal{N}}}^{syn}} \left[\exp\left(\Delta t \cdot \bigotimes_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} Q_{\mathcal{A}^{(k)}}^{e_{pos}}\right) \cdot \exp\left(\Delta t \cdot \bigotimes_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} Q_{\mathcal{A}^{(k)}}^{e_{neg}}\right)\right]$

    for $k$ in 1 .. $n$:
        $\mathbb{S}_{\boldsymbol{\mathcal{N}}}^{k \cdot \Delta t} := \mathbb{S}_{\boldsymbol{\mathcal{N}}}^{(k-1) \cdot \Delta t} \circ \mathbb{E}_{\boldsymbol{\mathcal{N}}}^{\Delta t}$

    return $\mathbb{S}_{\boldsymbol{\mathcal{N}}}^{t}$
end
```

Listing 7.1. Pseudocode of the TNSAN simulation algorithm.

7.3. Explanatory supplement

To begin with, the matter of Generalized Tensor Algebra should be raised. As presented in the Section 5.1, the extended Kronecker algebra formalism is widely used to model interactions between components building any SAN model. However within this research only the Ordinary Tensor Algebra is employed, because the generalized tensor product does not preserve the compatibility with ordinary matrix multiplication (strictly speaking, this property holds in three degenerate cases comprehensively presented in [18]), and thus the fundamental – in case of this thesis – properties of the Kronecker product (such as the ones presented by the Equation 2.9, as well as Theorems 5 and 6) are not still valid for its generalized equivalent. Nevertheless, the lack of functional transitions and Generalized Tensor Algebra does not neither undermine nor restrict the expression power of the SAN formalism – still any SAN may be expressed in terms of the Ordinary Tensor Algebra only, albeit such models usually require insertion of additional automata and synchronizing events to the system under simulation [3, 11]. As already presented in the Section 7.1, the TNSAN algorithm handles synchronization of components neatly and smoothly, therefore such model extensions are acceptable unless additional synchronizing events relate to insignificant number of constituent automata.

Furthermore, one should note that a descriptor of any SAN is composed of matrices of the same size, therefore their sum is commutative. However it is highly recommended to preserve the descriptor structure provided by the Theorem 10, because handling the positive and negative matrices corresponding to any synchronizing event pairwise allows to reduce the number of permutations required to compute the final result (cf. Step 4 in the Section 7.1).

7.4. Performance analysis

Following the remark provided by the Section 6.2, this Section presents some notes regarding both memory (Section 7.4.1) and time (Section 7.4.2) complexities of the TNSAN algorithm. These measures are given according to Family of Bachmann-Landau notations, which constitutes the conventional way of describing algorithm performance. The following two asymptotic growth rates are going to be used: the "Big O" notation and the "Big Theta" notation, both introduced by the Definitions 24 and 25 respectively.

Definition 24. Let $f, g : \mathbb{R}^+ \to \mathbb{R}$ and g be strictly positive for sufficiently large arguments. Then, the function f is *asymptotically bounded above* by the function g, denoted by $f(n) = O(g(n))$ for $n \in \mathbb{R}^+$, if:

$$\exists_{c \in \mathbb{R}^+} \exists_{n_0 \in \mathbb{R}^+} \forall_{n \geq n_0} : |f(n)| \leq c \cdot g(n) .$$

Definition 25. Let $f, g : \mathbb{R}^+ \to \mathbb{R}$ and g be strictly positive for sufficiently large arguments. Then, the function f is *asymptotically bounded both above and below* by the function g, denoted by $f(n) = \Theta(g(n))$ for $n \in \mathbb{R}^+$, if:

$$\exists_{c_1 \in \mathbb{R}^+} \exists_{c_2 \in \mathbb{R}^+} \exists_{n_0 \in \mathbb{R}^+} \forall_{n \geq n_0} : c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) .$$

7.4.1. Memory complexity

There are two major data structures employed by the TNSAN algorithm: the TN of state and the TN of evolution. To begin with the former one, consider its two possible representations. Firstly, the TN of evolution may be given by a set of probability distributions, one per each constituent component. For a SAN $\boldsymbol{\mathcal{N}} = (\mathscr{A}_{\boldsymbol{\mathcal{N}}}, \mathscr{E}_{\boldsymbol{\mathcal{N}}})$ with $\mathscr{A}_{\boldsymbol{\mathcal{N}}} = \left(\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \dots, \mathcal{A}^{(N_{\boldsymbol{\mathcal{N}}})}\right)$, such probability distribution for arbitrary component $\mathcal{A}^{(k)}$ is represented by a probability vector $\mathbf{p}_{\mathcal{A}^{(k)}}$ of size $S_{\mathcal{A}^{(k)}}$. Therefore, the overall memory usage for the TN of state given in the format discussed is:

$$\Theta\left(\sum_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} S_{\mathcal{A}^{(k)}}\right) .$$

Alternatively, the joint probability distribution tensor $T_{\alpha^{(1)}, \alpha^{(2)}, \dots, \alpha^{(N_{\boldsymbol{\mathcal{N}}})}}$ – with each index corresponding to single automaton belonging to the SAN $\boldsymbol{\mathcal{N}}$ – may be used to build the TN of state, employing the Tensor Train decomposition. According to the Definition 22, such tensor T may be represented by a TN composed of $N_{\boldsymbol{\mathcal{N}}}$ rank-3 arrays: $G^{(k)}_{\beta^{(k-1)}, \alpha^{(k)}, \beta^{(k)}}$, $1 \leq k \leq N_{\boldsymbol{\mathcal{N}}}$. Assuming $\alpha = \max_{1 \leq k \leq N_{\boldsymbol{\mathcal{N}}}} \left|\alpha^{(k)}\right|$ and $\beta = \max_{0 \leq k \leq N_{\boldsymbol{\mathcal{N}}}} \left|\beta^{(k)}\right|$, one can say that:

1. the tensor T has $O\left(\alpha^{N_{\boldsymbol{\mathcal{N}}}}\right)$ elements, since:

$$\prod_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} \left|\alpha^{(k)}\right| \leq \prod_{k=1}^{N_{\boldsymbol{\mathcal{N}}}} \alpha = \alpha^{N_{\boldsymbol{\mathcal{N}}}};$$

2. the tensor $G^{(k)}_{\beta^{(k-1)}, \alpha^{(k)}, \beta^{(k)}}$ has $O\left(\alpha \cdot \beta^2\right)$ elements, since:

$$\left|\alpha^{(k)}\right| \cdot \left|\beta^{(k-1)}\right| \cdot \left|\beta^{(k)}\right| \leq \alpha \cdot \beta^2;$$

3. hence, the Tensor Train format of the tensor T utilizes:

$$O\left(N_{\boldsymbol{\mathcal{N}}} \cdot \alpha \cdot \beta^2\right)$$

memory to store $O\big(\alpha^{N_{\mathcal{N}}}\big)$-elements tensor, where: β is a parameter dependent on the particular decomposition case.

Furthermore, the memory requirements of the TN of evolution are considered separately for tensors corresponding to either local transitions or synchronized transitions. According to the former case, the "local" part of the aforementioned network is composed of $N_{\mathcal{N}}$ matrices, each corresponding to single automaton $\mathcal{A}^{(k)}$, and therefore of size $S^2_{\mathcal{A}^{(k)}}$. Hence, the overall memory usage for this part of the TN of evolution is:

$$\Theta\left(\sum_{k=1}^{N_{\mathcal{N}}} S^2_{\mathcal{A}^{(k)}}\right).$$

Now consider the "synchronized" part of the TN of evolution for arbitrary synchronizing event $e \in \mathscr{E}_{\mathcal{N}}$. Firstly, two perfect shuffle permutation matrices R and R^{-1} are stored for each pair of automata $\mathcal{A}^{(i)}$ and $\mathcal{A}^{(j)}$, $i+1 < j$, such that the automata $\mathcal{A}^{(i+1)}, \mathcal{A}^{(i+2)}, \ldots, \mathcal{A}^{(j-1)}$ are not involved by the event e. Please note that in worst case (i.e. when the event e affects only two automata with $i = 1$ and $j = N_{\mathcal{N}}$) the size of such matrices degenerates to the squared size of the PSS, $S^2_{\mathcal{N}}$. In general, the number and the size of permutation matrices strictly depends on the simulated model, therefore the generic complexity formula is not provided here.
Finally, assume that the synchronizing event e affects m automata: $\mathcal{A}^{(i_1)}, \mathcal{A}^{(i_2)}, \ldots, \mathcal{A}^{(i_m)}$, $1 \leq i_1 < i_2 < \ldots < i_m \leq N_{\mathcal{N}}$. Then, exponentials of Kronecker products of the matrices $Q^{e_{pos}}_{\mathcal{A}^{(i_1)}}, Q^{e_{pos}}_{\mathcal{A}^{(i_2)}}, \ldots, Q^{e_{pos}}_{\mathcal{A}^{(i_m)}}$ and $Q^{e_{neg}}_{\mathcal{A}^{(i_1)}}, Q^{e_{neg}}_{\mathcal{A}^{(i_2)}}, \ldots, Q^{e_{neg}}_{\mathcal{A}^{(i_m)}}$ are both of size $\big(S_{\mathcal{A}^{(i_1)}} \cdot S_{\mathcal{A}^{(i_2)}} \cdot \ldots \cdot S_{\mathcal{A}^{(i_m)}}\big)^2$. Please note that in worst case (i.e. when all automata are affected by the event e), the size of such matrices degenerates to $S^2_{\mathcal{N}}$. In general, the overall memory complexity of this part of the TN of evolution is dependent on the number of synchronizing events $E_{\mathcal{N}}$ (linearly) and the aforementioned memory footprint resulting from exponential of Kronecker products of all the matrices $Q^{e_{pos}}$ and $Q^{e_{neg}}$.

Please note that the overall memory complexity is not dependent on the number of iterations – it is sufficient to store single TN of state and single TN of evolution. During the execution of the TNSAN algorithm, the iterative contraction of the TN of evolution (remaining permanently constant) with the TN of state may be performed "in place", i.e. without the necessity of utilizing additional memory. The result of such contraction may be consecutively kept in the data structure storing the TN of evolution, because the solution obtained in the k-th iteration of the TNSAN algorithm can be computed based on the outcomes from the $k-1$-th iteration only.

7.4.2. Time complexity

To begin with, the time complexity of building the initial TN of state should be investigated. As previously mentioned, such network should be given in two ways. Firstly, when the initial probability distribution of states is provided separately for each automaton, the time overhead for building the TN is negligible. On the other hand, if the distribution discussed is given by the joint probability distribution in the high-rank tensor form $T_{\alpha^{(1)},\alpha^{(2)},\ldots,\alpha^{(N_{\mathcal{N}})}}$, the Tensor Train representation needs to be computed.

Then, the TT-SVD algorithm [31] may be employed, which successively extracts the k-th Tensor Train core $G^{(k)}_{\beta^{(k-1)},\alpha^{(k)},\beta^{(k)}}$ at k-th iteration, by obtaining the truncated Singular Value Decomposition (SVD) of the matrix $T \in \mathscr{M}_{|\beta^{(k-1)}|\cdot|\alpha^{(k)}|\times\prod_{j=k+1}^{N_{\mathcal{N}}}|\alpha^{(j)}|}$. Therefore, such preprocessing demands:

$$O\big(N_{\mathcal{N}} \cdot mn^2\big)$$

operations to be performed, since the SVD algorithm requires $O\big(mn^2\big)$ complexity for any rectangular matrix belonging to $\mathscr{M}_{m\times n}$ with $m \geq n$.

The process of building the TN of evolution involves calculating exponentials of matrices corresponding to both local and synchronized transitions. The complexity of such operations strictly depends on a numerical method chosen from a bunch of possibilities (cf. [29]). The Julia implementation[1] used within this research employs either the eigendecomposition-based method or the scaling and squaring approach, both characterized by the $O\big(n^3\big)$ complexity [29] for any matrix $A \in \mathscr{M}_n$ subjected to exponentiation. Therefore, calculating exponentials of the matrices corresponding to local transitions requires:

$$O\left(\sum_{k=1}^{N_{\mathcal{N}}} S^3_{\mathcal{A}^{(k)}}\right)$$

operations to be performed in general. Assuming again that the synchronizing event e affects m automata: $\mathcal{A}^{(i_1)}, \mathcal{A}^{(i_2)}, \ldots, \mathcal{A}^{(i_m)}, 1 \leq i_1 < i_2 < \ldots < i_m \leq N_{\mathcal{N}}$, the time complexity of calculating exponentials of Kronecker products of matrices $Q^{e_{pos}}_{\mathcal{A}^{(i_1)}}$, $Q^{e_{pos}}_{\mathcal{A}^{(i_2)}}$, . . ., $Q^{e_{pos}}_{\mathcal{A}^{(i_m)}}$ and $Q^{e_{neg}}_{\mathcal{A}^{(i_1)}}$, $Q^{e_{neg}}_{\mathcal{A}^{(i_2)}}$, . . ., $Q^{e_{neg}}_{\mathcal{A}^{(i_m)}}$ is:

$$O\Big(\big(S_{\mathcal{A}^{(i_1)}} \cdot S_{\mathcal{A}^{(i_2)}} \cdot \ldots \cdot S_{\mathcal{A}^{(i_m)}}\big)^3\Big).$$

Naturally, the overall complexity of calculating such exponentials depends linearly on the number of synchronizing events $E_{\mathcal{N}}$, since the SAN formalism requires to define two matrices $Q^{e_{pos}}_{\mathcal{A}^{(k)}}$ and $Q^{e_{neg}}_{\mathcal{A}^{(k)}}$ for each synchronizing event e and for all components $\mathcal{A}^{(k)}$. Please note that in worst case (i.e. when all automata are affected by the event e), the complexity of such computations degenerates to $O\big(S^3_{\mathcal{N}}\big)$.

After the preprocessing phase, necessary to build required TN structures, the iterative contraction process follows. During each iteration the TN of evolution is “merged” to the TN of state by employing the contraction procedure. According to the remark provided to the Definition 21 and the Example 6, there are usually multiple ways to perform the contraction of any TN, which differ in both time and memory efficiency. Therefore the complexity of computing the desired probability distribution of states at arbitrary time t depends not only on the particular SAN under simulation, but also on the adopted sequence of contractions execution. Although the problem of determining the optimal contraction order is generally known to be NP-hard, note that TNs defined within the TNSAN algorithm result from the ordinary matrix operations summarized by the Step 4 in the Section 7.1, therefore the simple contraction scheme “from left to right” (cf. Figure 7.1) is bounded by the polynomial complexity of the PSS size $S_{\mathcal{N}}$. The

[1] `https://docs.julialang.org/en/v0.6.0/stdlib/linalg/#Base.LinAlg.expm` (access: 7 June 2018)

same contraction scheme is performed n times during the execution of the TNSAN algorithm, where n is the arbitrarily provided number of iterations, hence one may summarize that the overall contraction process for any TNSAN input is certainly:

$$O(n \cdot poly\,(S_{\mathcal{N}}))\,.$$

7.5. Summary

Within this Chapter the TNSAN algorithm – novel contribution of this thesis – was presented in details. To begin with, the proposed algorithm for determining transient probability distribution of states belonging to any SAN model at arbitrary instant of time was formally derived and summarized at a high level with additional implementation remarks provided. Then, a comprehensive theoretical analysis of both time and memory complexities of the method in question was given in terms of the Bachmann-Landau notation, taking into account various types of the TNSAN input data. Besides the formal evaluation, numerical assessment of the proposed algorithm is presented in the following Chapter 8.

8. Experimental results

This Chapter presents the evaluation of results obtained by conducting multiple numerical tests on the implementation of the TNSAN algorithm. Firstly, the adopted test scenarios are presented in the Section 8.1. Furthermore, the numerical results obtained within these guidelines are comprehensively described and analyzed in the Section 8.2.

8.1. Numerical evaluation scenarios

For the needs of this research, all numerical tests have been conducted on a SAN model described in the Example 1. This exhaustive benchmark problem incorporates all the features of stochastic automata discussed within this thesis, i.a. local and synchronized transitions, unreachable states and alternative probabilities of choosing destination states. Furthermore, the model in question is easily scalable regarding its capacity and density of a global transition rate matrix (depending on the number of automata), as well as participation of the RSS in the PSS (manipulated by the number of processes permitted to use the shared resource simultaneously).

Multiple test scenarios have been elaborated. The main one, which results are exhaustively described in the Section 8.2, incorporates four automata: $\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(4)}$ as presented in the Example 1, one per each kind defined there. This model simulates the mutual exclusion system, therefore the counting automaton is equipped with two states. The following configuration has been set (please note that the symbols used below follow the convention introduced by the Example 1):

1. Automaton $\mathcal{A}^{(1)}$:

   - $\lambda_1 = 5$, $\lambda_2 = 3$, $\lambda_3 = 3$, $\lambda_4 = 1$, $\lambda_5 = 3$;
   - $\mathbf{p}_{\mathcal{A}^{(1)}}(0) = (1, 0, 0, 0)$;

2. Automaton $\mathcal{A}^{(2)}$:

   - $\mu_1 = 10$, $\mu_2 = 4$, $\mu_3 = 6$, $\mu_4 = 3$, $\mu_5 = 4$;
   - $\mathbf{p}_{\mathcal{A}^{(2)}}(0) = (1, 0, 0)$;

3. Automaton $\mathcal{A}^{(3)}$:

- $\delta_1 = 3$, $\delta_2 = 3$, $\delta_3 = 2$, $\delta_4 = 2$;
- $\mathbf{p}_{\mathcal{A}^{(3)}}(0) = (1, 0, 0, 0)$;

4. Automaton $\mathcal{A}^{(4)}$:

   - this model does not master neither local nor synchronized transitions;
   - $\mathbf{p}_{\mathcal{A}^{(4)}}(0) = (1, 0)$.

The goal of the TNSAN algorithm is to determine a transient probability distribution of states belonging to the network defined above at time $t = 10$ units.

Furthermore, the models with four automata (one per each kind, as presented above) and increased size of the shared resource (allowing two and three processes to use the protected entity concurrently) have been tested in order to examine the impact of the RSS/PSS size ratio on the TNSAN performance. Also, a mutual exclusion scenario for seven machines (two per each kind and a counting one) has been investigated in order to study the effect of increasing the model capacity on the TNSAN efficiency.

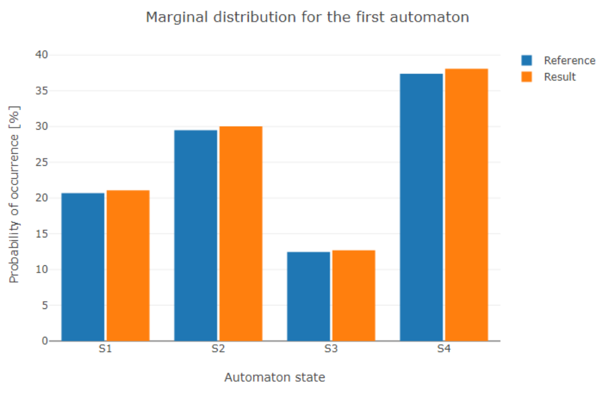
Obtained results accuracy is determined in comparison to the reference result determined by the straightforward calculation of the global transition rate matrix exponential. To compute the reference result, the Padé approximation approach [29, 41] dedicated for sparse matrices and implemented in EXPOKIT is employed.

Please note that the configuration of both physical and software properties of a machine on which the numerical tests have been carried out is provided by the Appendix B.

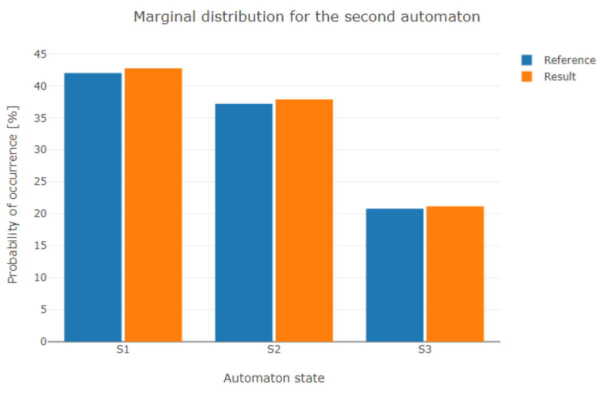
8.2. Results and conclusions

To begin with, consider the aforementioned test scenario with four automata and the mutual exclusion scheme. This model has been chosen to be a primary benchmark problem used for the evaluation of numerical properties of the TNSAN algorithm.

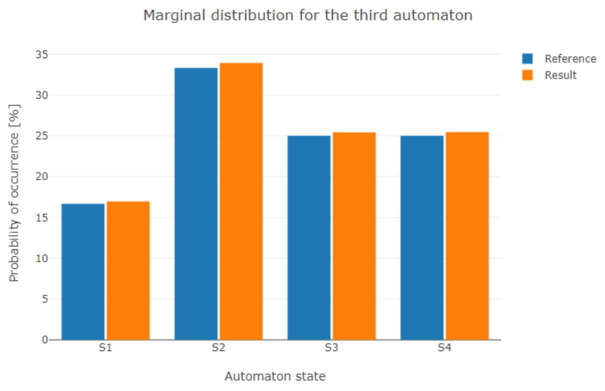
The most straightforward way to verify the accuracy of results computed by the TNSAN algorithm is to compare marginal probability distributions derived for each component of the SAN under simulation with their equivalents determined on the basis of the reference result. Such a summary has been established for two cases, when the number of iterations reached 10^5 and 10^6. A diagrammatic comparison for the former case is presented in the Figure 8.1, while the latter – in the Figure 8.2. Please note that orange bars denote the results obtained by the execution of the TNSAN algorithm, while blue bars present the reference (ground truth) values.



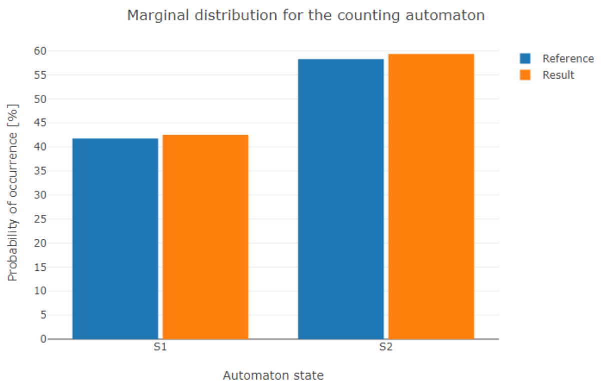
(a) Marginal distribution for the automaton $\mathcal{A}^{(1)}$.



(b) Marginal distribution for the automaton $\mathcal{A}^{(2)}$.



(c) Marginal distribution for the automaton $\mathcal{A}^{(3)}$.

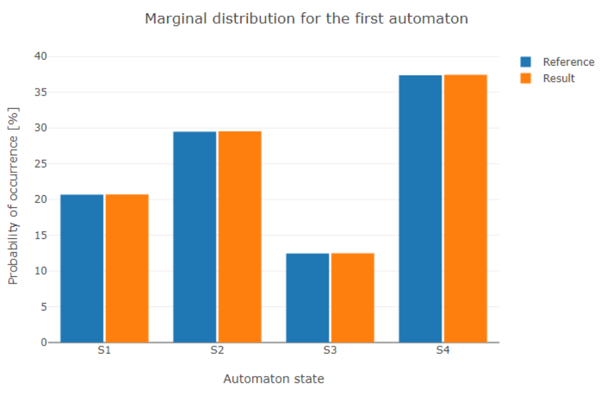


(d) Marginal distribution for the automaton $\mathcal{A}^{(4)}$.

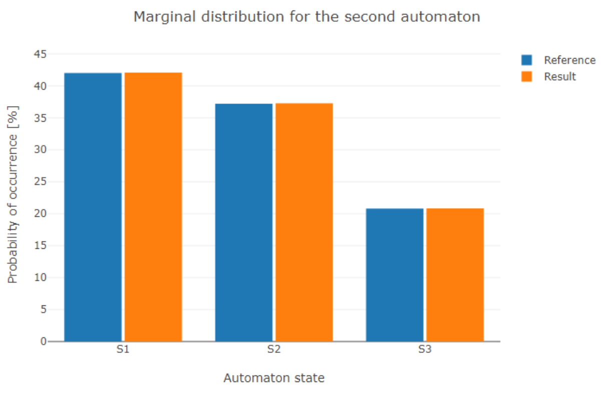
Figure 8.1. Comparison of marginal probability distributions resulting from the TNSAN algorithm with 10^5 iterations and the reference method.

According to the Figure 8.1, one may observe that the results of the TNSAN algorithm and the reference ones are roughly similar, however there are some noticeable divergences between them. Indeed, mentioned differences arise as a consequence of the approximation error introduced intentionally to the TNSAN method by employing the Suzuki-Trotter expansion of the first order (cf. Step 3 in the Section 7.1). Hence, a value of the $\mathscr{L}_1$ norm of the TNSAN result vector equals 1.018 (ideally it should be equal to 1, since it is a probability vector). Additionally, a similarity of the two aforementioned vectors is estimated by a value of the $\mathscr{L}_\infty$ norm of the difference vector between the TNSAN result and the reference one. For the currently discussed case, this measure is equal to $9 \cdot 10^{-4}$ (naturally, the value of 0 is the optimum for this metrics).

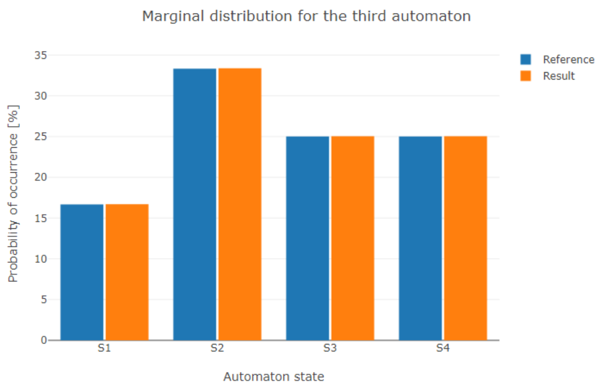
Referring to the Figure 8.2, all the differences between the results of the TNSAN method and the reference ones, previously visible with the naked eye, are just compensated. From the numerical point of view, a value of the aforementioned $\mathscr{L}_1$ norm of the TNSAN result vector is now equal to 1.002, while a value of the $\mathscr{L}_\infty$ equals $9 \cdot 10^{-5}$. Therefore, a tenfold increase in the number of iterations resulted in a results improvement by an order of magnitude.



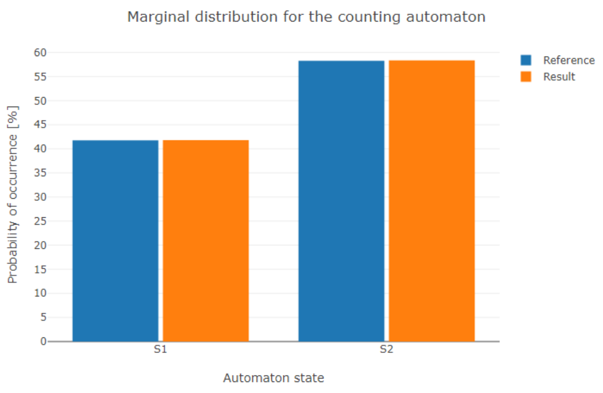
(a) Marginal distribution for the automaton $\mathcal{A}^{(1)}$.



(b) Marginal distribution for the automaton $\mathcal{A}^{(2)}$.



(c) Marginal distribution for the automaton $\mathcal{A}^{(3)}$.

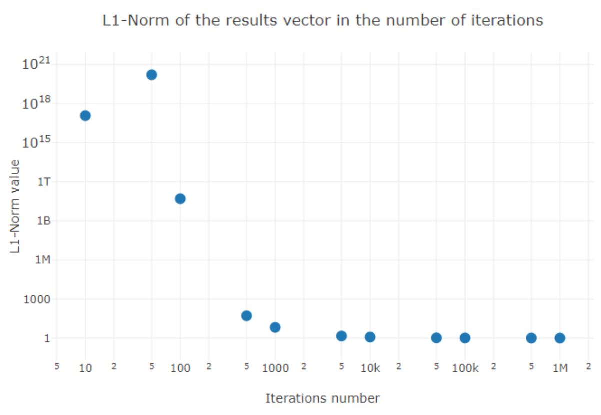


(d) Marginal distribution for the automaton $\mathcal{A}^{(4)}$.

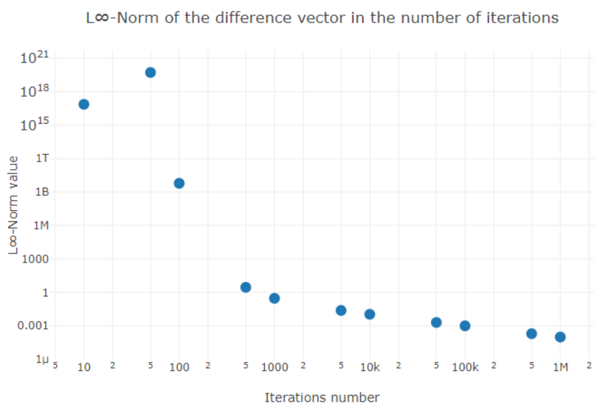
Figure 8.2. Comparison of marginal probability distributions resulting from the TNSAN algorithm with 10^6 iterations and the reference method.

Furthermore, numerical properties of the TNSAN method should be evaluated in terms of increasing number of iterations. For this purpose, the algorithm has been run eleven times with identical configuration and different number of iterations, ranging from 10 to 10^6. The following measures have been calculated: a value of the $\mathscr{L}_1$ norm of the TNSAN result vector (cf. Figure 8.3a), a value of the $\mathscr{L}_\infty$ of the difference vector between the TNSAN and reference results (cf. Figure 8.3b), overall contraction time required to compute a desired probability matrix $P(t)$ (cf. Step 1 in the Section 7.1) for $t = 10$ (cf. Figure 8.3c). Additionally, a relationship between the results accuracy (measured in accordance with the $\mathscr{L}_1$ norm) and the overall contraction time is presented in the Figure 8.3d.

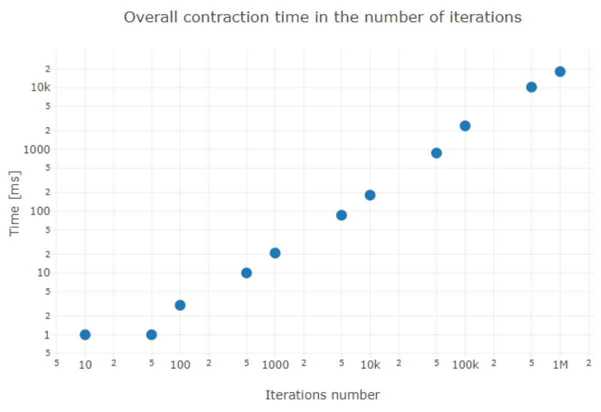
According to the Figures 8.3a and 8.3b, one may immediately confirm the assumed trade-off between the method's performance and the results quality. Values of the $\mathscr{L}_1$ norm converges to 1, while values of the $\mathscr{L}_\infty$ strongly decrease along with the growth of the iterations number. However it should be noted that the TNSAN results become reasonable for at least 10^5 iterations, which is quite a lot. Moreover, the Figure 8.3d confirms that the longer algorithm runs, the higher quality results are obtained. Finally, the overall contraction time trend presented in the Figure 8.3c is about to be linear in the double logarithmic scale, which means that it is a polynomial function of the number of iterations.



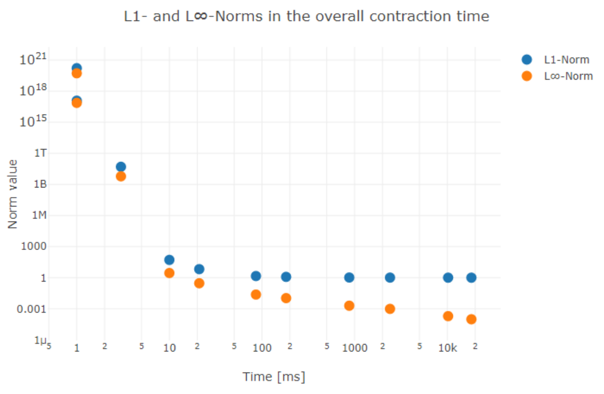
(a) $\mathscr{L}_1$ norm of the results vector in the number of iterations.



(b) $\mathscr{L}_\infty$ norm of the difference vector in the number of iterations.



(c) Overall contraction time in the number of iterations.



(d) $\mathscr{L}_1$ and $\mathscr{L}_\infty$ norms in the overall contraction time.

Figure 8.3. Numerical evaluation of the TNSAN algorithm with variable number of iterations.

According to the Section 8.1, the numerical experiments conducted within this thesis did not concern the case presented above only. The tests carried out for four automata with identical configuration as above and increased size of the counting automaton (to three and four states) confirmed all the observations and conclusions derived before. Therefore one may reason that the TNSAN algorithm handles models with variable proportion of the RSS to PSS size. Furthermore, results of the examination of the mutual exclusion scenario with seven automata are consistent with the observations made so far, which means that the TNSAN algorithm copes with high capacity models represented globally by sparse matrices with $O(10^7)$ elements and 0.17% of density.

8.3. Summary

In this Chapter the numerical results of the TNSAN approach evaluation were presented and discussed. It began with a detailed description of testing scenarios based on the resource sharing benchmark problem, which comprehensively examine the algorithm's numerical properties for variable-capacity sparse models. Then, the results concerning both results accuracy and method's efficiency were provided in terms of the aforementioned scenarios. Gathered outcomes confirm basically all expected preassumptions.

9. Conclusions and future work

This Chapter summarizes the research work done for the needs of this thesis. It begins with a verification of preassumed hypotheses and goals specified to confirm or deny them (Section 9.1). Then, the obtained results are summarized in the Section 9.2. Finally, some remarks regarding ideas for future development of this research are presented in the Section 9.3.

9.1. Hypotheses verification and review of achieved goals

At the beginning of this thesis development, a list of research questions has been elaborated. Answers to these questions act as research hypotheses, which should be confirmed or denied within this document by achieving consecutive research goals. This schedule – gathered in the Section 5.3 – is a reference point for the following review of achieved goals and verification of hypotheses.

Hypothesis 1. The verification of this hypothesis began with conducting a comprehensive state of the art research regarding applicability areas, distinctive features and impassable limitations of SAN and TN formalisms, as well as previously developed SAN simulation methods, their advantages and drawbacks (cf. Section 5.1). This analysis resulted with a conclusion that there is still a gap, which could be fulfilled by the TNSAN algorithm. Then, following theoretical foundations of Algebra and Stochastic Processes (cf. Chapter 2), SANs (cf. Chapter 3) and TNs (cf. Chapter 4), a numerical method for computing the exponential of any Kronecker-structured transition rate matrix has been proposed in the Section 7.1 and followed by a description of the algorithm for building a TN structure employing such representation. Therefore, based on the achievement of mentioned research goals, the Hypothesis 1 is considered as confirmed.

Hypothesis 2. Within the verification process conducted for this hypothesis, two methods for obtaining a TN of initial state (accepting either a collection of probability distributions belonging to model constituents or a global joint probability distribution over all automata) have been described in the Section 7.1. Then, a contraction pattern resulting in the computation of a desired transient probability distribution has been defined and presented in the Section 7.2. Hence, the Hypothesis 2 is considered as confirmed too.

Hypothesis 3. The TNSAN algorithm performance has been examined in multiple ways within this research. Firstly, a qualitative analysis regarding the elaboration of both time and memory complexities of

the proposed approach in accordance with the Bachmann-Landau has been performed in the Section 7.4. Unfortunately, this analysis pointed out that there is no possibility to simply determine overall complexities, because they are tightly coupled with the TN structure created by the TNSAN algorithm, which – in turn – strictly depends on the particular problem being solved. Furthermore, this study revealed that in the worst case scenario, both measures are polynomial in the size of the PSS, not in the function of the number of states belonging to all constituent automata (as previously expected).
Moreover, the TNSAN algorithm has been evaluated numerically on the scalable and slightly modified benchmark problem of resource sharing (cf. Example 1 and Section 8.1). The choice of this testing scenario allowed to conduct a comprehensive study of the TNSAN algorithm properties, however single model did not provide adequate diversity of tests. Therefore, the second goal specified for the verification of this hypothesis is partially achieved. Finally, a quantitative analysis of the TNSAN algorithm has been conducted and presented in the Section 8.2, which showed the correctness of computed results in comparison to the reference ones, confirmed the trade-off between method's performance and results accuracy, as well as demonstrated the TNSAN flexibility to variable model capacity and the RSS/PSS size ratio.
To conclude, this hypothesis is considered as partially confirmed.

9.2. Results summary

In general, this thesis defines a novel, previously unexplored approach connecting the SAN and TN formalisms in order to cope with the ubiquitous state space issue using methods with proven effectiveness in overcoming the curse of dimensionality problem. For this purpose, it reduces a task of computing a transient probability distribution of states belonging to any SAN to the matter of TNs contraction. Results obtained within this research may fall into two broad groups: theoretical and simulation ones.

In case of the former type, the following artifacts are distinguished:

1. Firstly, an exhaustive study in the fields of Algebra and Stochastic Processes (cf. Chapter 2), basics of SANs (cf. Chapter 3) and TNs (cf. Chapter 4) resulted in the novel elaboration of formal foundations of the aforementioned concepts. Bringing them together in a consistent and logical description may be definitely considered as a significant contribution to the scientific domain of the problem.

2. The main and unquestionable scientific contribution is the elaboration of the algorithm for determining transient probability distributions of states belonging to any SAN using TNs, called TNSAN. This artifact involves:

    - a formal elaboration of a numerical method for computing the exponential of a matrix expressed in terms of a sum of Kronecker products;
    - a formal elaboration of an algorithm for determining transient probability distribution of states employing a tensor representation of computations;

- an analysis of both time and memory complexities of the proposed approach, as well as an indication of relationships between its parameters and numerical properties;
- outlining the area of applicability of the TNSAN algorithm.

In case of the latter group, one may figure out:

1. An implementation of the TNSAN algorithm in the `Julia` programming language for the scalable benchmark model of resource sharing system.
2. Numerical evaluation results obtained by executing the aforementioned implementation on a benchmark problem in multiple test scenarios.

All the results obtained within this research are self-contained – the TNSAN algorithm currently allows to simulate any SAN without functional transitions (actually this is not a limitation since such SANs may always be expressed in terms of local and synchronized transitions only), defines all possible restrictions and prerequisites and does not need to be extended in order to handle any missing special case.

9.3. Current status and development directions

The outcomes of this thesis should be taken as a basis for further research regarding not only a later development of the TNSAN algorithm, but also the elaboration of subsequent numerical methods employing the TNs formalism to solving SAN-related numerical problems. In case of the TNSAN algorithm, its biggest shortcoming is an inability to simulate SAN models with functional transitions, which incorporation may significantly decrease model capacity. Thus, it is necessary to deeply investigate the properties of Generalized Tensor Algebra and elaborate enriched foundations of the TNSAN algorithm, which enable the method in question to simulate SAN models more efficiently. Furthermore, one may attempt to decrease both time and memory complexities by suggesting improvements to the presented algorithm. When it comes to the numerical evaluation, an efficient implementation of the TNSAN algorithm, possible to be executed on multiple computing nodes should be carried out. Then, a collection of diversified benchmark problems should be selected and a series of tests should be performed on them in order to comprehensively investigate all numerical properties of the TNSAN algorithm and extract its real areas of applicability.

Appendix A. Proof of Theorem 6

Proof. Begin with the Equation 2.14.

1. According to the Definition 3, it holds:

$$\exp(A \otimes I_m) = \sum_{k=0}^{\infty} \frac{1}{k!}(A \otimes I_m)^k = I_{nm} + \sum_{k=1}^{\infty} \frac{1}{k!}(A \otimes I_m)^k.$$

2. By applying the Equation 2.9, one may obtain further:

$$\exp(A \otimes I_m) = I_{nm} + \sum_{k=1}^{\infty} \frac{1}{k!}(A^k \otimes I_m^k).$$

3. Due to the idempotency of the identity matrix, it holds: $I_m^k = I_m$ for any integer $k \geq 1$. Additionally, from the Equation 2.1: $I_{nm} = I_n \otimes I_m$. Then:

$$\exp(A \otimes I_m) = I_n \otimes I_m + \sum_{k=1}^{\infty} \frac{1}{k!}(A^k \otimes I_m).$$

4. By virtue of the right-distributivity of the Kronecker product (cf. Equation 2.5), one can factor out the common term I_m:

$$\exp(A \otimes I_m) = \left(I_n + \sum_{k=1}^{\infty} \frac{1}{k!}A^k\right) \otimes I_m = \left(\sum_{k=0}^{\infty} \frac{1}{k!}A^k\right) \otimes I_m.$$

5. Again, according to the Definition 3:

$$\exp(A \otimes I_m) = \exp(A) \otimes I_m.$$

Please note that the proof of the Equation 2.15 is analogous to the proof given above, therefore it is not presented here. For the sake of completeness, one should notice that the right-distributivity property used in the Step 4 should be replaced with the left-distributivity property (cf. Equation 2.4) then.

Consider the Equation 2.16.

1. By employing the right-associativity of the Kronecker product (cf. Equation 2.7), one can write:

$$\exp(A \otimes I_k \otimes B) = \exp(A \otimes (I_k \otimes B)).$$

2. According to the pseudo-commutativity of the Kronecker product (cf. Equation 2.10) of matrices $A \in \mathscr{M}_n$ and $(B \otimes I_k) \in \mathscr{M}_{mk}$, it holds:

$$\exp(A \otimes I_k \otimes B) = \exp(S_{km,n}((I_k \otimes B) \otimes A)S_{n,km}).$$

3. Using the right-associativity property once again leads to:

$$\exp(A \otimes I_k \otimes B) = \exp(S_{km,n}(I_k \otimes B \otimes A)S_{n,km}).$$

4. Following the Theorem 7, it is straightforward to observe that $S_{n,km} = S_{km,n}^{-1}$. Thus:

$$\exp(A \otimes I_k \otimes B) = \exp(S_{km,n}(I_k \otimes B \otimes A)S_{km,n}^{-1}),$$

   and, by exploiting the Equation 2.13, it holds:

$$\exp(A \otimes I_k \otimes B) = S_{km,n} \exp(I_k \otimes B \otimes A)S_{km,n}^{-1} = S_{km,n} \exp(I_k \otimes B \otimes A)S_{n,km}.$$

5. Finally, from the right-associativity property:

$$\exp(A \otimes I_k \otimes B) = S_{km,n} \exp(I_k \otimes (B \otimes A))S_{n,km},$$

   and by applying the Equation 2.15 to the matrices I_k and $(A \otimes B)$, one can obtain:

$$\exp(A \otimes I_k \otimes B) = S_{km,n} \left(I_k \otimes \exp(B \otimes A)\right) S_{n,km}.$$

Please note that the proof of the second part of the Equation 2.16 is analogous to the proof given above, therefore it is not presented here. For the sake of completeness, one should notice that the right-associativity property used throughout this proof should be replaced with the left-associativity property (cf. Equation 2.6): $(A \otimes I_k) \otimes B$, as well as the Equation 2.14 should be used in place of the Equation 2.15. □

Appendix B. Testing infrastructure

The following Listing Appendix B.1 summarizes selected information regarding the host operating system parameters, obtained by executing the `systeminfo` command.

```
OS Name:                   Microsoft Windows 10 Home
OS Version:                10.0.17134 N/A Build 17134
OS Manufacturer:           Microsoft Corporation
OS Configuration:          Standalone Workstation
OS Build Type:             Multiprocessor Free
System Manufacturer:       Dell Inc.
System Model:              Inspiron 3537
System Type:               x64-based PC
Processor(s):              1 Processor(s) Installed.
                           [01]: Intel64 Family 6 Model 69 Stepping 1 GenuineIntel
    ~2401 Mhz
BIOS Version:              Dell Inc. A08, 30.04.2014
Total Physical Memory:     8 073 MB
Available Physical Memory: 2 746 MB
Virtual Memory: Max Size:  20 873 MB
Virtual Memory: Available: 12 693 MB
Virtual Memory: In Use:    8 180 MB
```

Listing Appendix B.1. Selected operating system information.

The following Listing Appendix B.2 presents selected information regarding single CPU core, stored in the `/proc/cpuinfo` system file.

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 69
model name      : Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz
stepping        : 1
cpu MHz         : 2394.000
cache size      : 256 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
    pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe pni dtes64 monitor ds_cpl
    vmx est tm2 ssse3 fma cx16 xtpr pdcm sse4_1 sse4_2 movbe popcnt aes xsave
    osxsave avx f16c rdrand lahf_lm ida arat epb xsaveopt pln pts dtherm fsgsbase
    tsc_adjust bmi1 avx2 smep bmi2 erms invpcid
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:
```

Listing Appendix B.2. Selected processor information.

The following Listing Appendix B.3 gathers selected information regarding the `Julia` programming language version and parameters, obtained by executing the `versioninfo(true)` command in the REPL. Please note that default `Julia` configuration has been used, with no exceptional tuning of the underlying LLVM compiler.

```
Julia Version 0.6.0
Commit 903644385b* (2017-06-19 13:05 UTC)
  BLAS: libopenblas (USE64BITINT DYNAMIC_ARCH NO_AFFINITY Haswell)
  LAPACK: libopenblas64_
  LIBM: libopenlibm
  LLVM: libLLVM-3.9.1 (ORCJIT, haswell)
```

Listing Appendix B.3. Selected `Julia` configuration parameters.

Appendix C. Publications and presentations

Theoretical foundations of the TNSAN algorithm have been presented by the author, in person, during the talk at the *4th Workshop for Doctoral Students and Young Researchers in Information Technology (WDSIT)*, held at 3-5 November 2017 in Kazimierz Dolny, Poland.

It is also planned to submit a scientific paper presenting the results of this thesis to a journal concerning operational research and performance evaluation topics.

Bibliography

[1] A. Benoit, B. Plateau, and W.J. Stewart. *Memory Efficient Iterative Methods for Stochastic Automata Networks*. Tech. rep. 4259. Institut National de Recherche en Informatique et en Automatique, 2001.

[2] A. Benoit, B. Plateau, and W.J. Stewart. “Memory-efficient Kronecker algorithms with applications to the modelling of parallel systems”. In: *Future Generation Computer Systems* 22.7 (2006), pp. 838–847.

[3] A. Benoit et al. “On the benefits of using functional transitions and Kronecker algebra”. In: *Performance Evaluation* 58.4 (2004), pp. 367–390.

[4] J. Bezanson et al. “Julia: A Fresh Approach to Numerical Computing”. In: *Society for Industrial and Applied Mathematics (SIAM) Review* 59.1 (2017), pp. 65–98.

[5] J. Biamonte and V. Bergholm. “Quantum Tensor Networks in a Nutshell”. In: *ArXiv e-prints* (2017). arXiv: *1708.00006* `[quant-ph]`.

[6] G. Bolch et al. *Queuing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. 2nd ed. John Wiley & Sons, Inc., 2006. ISBN: 9780471565253.

[7] J.C. Bridgeman and C.T. Chubb. “Hand-waving and Interpretive Dance: An Introductory Course on Tensor Networks”. In: *Journal of Physics A: Mathematical and Theoretical* 50.22 (2017), p. 223001.

[8] P. Buchholz. “Structured analysis approaches for large Markov chains”. In: *Applied Numerical Mathematics* 31.4 (1999), pp. 375–404.

[9] P. Buchholz. “Product form approximations for communicating Markov processes”. In: *Performance Evaluation* 67.9 (2010), pp. 797–815.

[10] P. Buchholz et al. “Complexity of Memory-Efficient Kronecker Operations with Applications to the Solution of Markov Models”. In: *INFORMS Journal on Computing* 12.3 (2000), pp. 203–222.

[11] M.-Y. Chung et al. “A comparison of structural formalisms for modeling large Markov models”. In: *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS’04)*. Ed. by B. Werner. IEEE Press, 2004, pp. 196b.

[12] G. Ciardo et al. "Logic and stochastic modeling with SMART". In: *Performance Evaluation* 63.6 (2006), pp. 578–608.

[13] R.M. Czekster et al. "Split: a flexible and efficient algorithm to vector-descriptor product". In: *Proceedings of the 2nd International ICST Conference on Performance Evaluation Methodologies and Tools (ValueTools'07)*. Ed. by P.W. Glynn. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, 2007, 83:1–83:8. ISBN: 9789639799004.

[14] M. Davio. "Kronecker Products and Shuffle Algebra". In: *IEEE Transactions on Computers* 30.2 (1981), pp. 116–125.

[15] C. Dehnert et al. "A STORM is Coming: A Modern Probabilistic Model Checker". In: *Proceedings of the 29th International Conference on Computer Aided Verification (CAV'17)*. Ed. by R. Majumdar and V. Kunčak. Springer, 2017, pp. 592–600.

[16] P. Fernandes, L. Lopes, and S. Yeralan. "Symbolic Solution of Kronecker-based Structured Markovian Models". In: *Proceedings of the 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'13)*. IEEE Press, 2013, pp. 409–413. ISBN: 9780769551029.

[17] P. Fernandes and B. Plateau. *Modeling Finite Capacity Queueing Networks with Stochastic Automata Networks*. Tech. rep. 004. Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul, 2000.

[18] P. Fernandes, B. Plateau, and W.J. Stewart. *Efficient Descriptor-Vector Multiplications in Stochastic Automata Networks*. Tech. rep. 2935. Institut National de Recherche en Informatique et en Automatique, 1996.

[19] P. Fernandes, A. Sales, and L. Zani. *PEPS2015 - Stochastic Automata Networks Software Tool*. 2015.

[20] J.-M. Fourneau, B. Plateau, and W.J. Stewart. "Product form for Stochastic Automata Networks". In: *Proceedings of the 2nd International ICST Conference on Performance Evaluation Methodologies and Tools (ValueTools'07)*. Ed. by P.W. Glynn. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, 2007, 32:1–32:10. ISBN: 9789639799004.

[21] I. Gavrilyuk, V. Makarov, and V. Vasylyk. "Exponentially Convergent Algorithms for Abstract Differential Equations". In: 1st ed. Birkhäuser, 2010. Chap. Appendix: Tensor-product approximations of the operator exponential, pp. 167–172. ISBN: 9783034801188.

[22] B.C. Hall. "Lie groups, Lie algebras, and representations: an elementary introduction". In: 1st ed. Springer-Verlag, 2003. Chap. 2, pp. 27–58. ISBN: 9781441923134.

[23] H.V. Henderson and S.R. Searle. "The Vec-Permutation Matrix, The Vec Operator and Kronecker Products: A Review". In: *Linear and Multilinear Algebra* 9.4 (1981), pp. 271–288.

[24] R.A. Horn and C.R. Johnson. *Matrix Analysis*. 2nd ed. Cambridge University Press, 2012. ISBN: 9780521548236.

[25] M. Kleiber. “Modelowanie i symulacja komputerowa – moda czy naturalny trend rozwojowy nauki”. In: *Nauka* 4 (1999), pp. 29–41.

[26] D. Kressner and F. Macedo. “Low-rank tensor methods for communicating Markov processes”. In: *Proceedings of the 11th International Conference on Quantitative Evaluation of Systems (QEST’14)*. Ed. by G. Norman and W. Sanders. Springer, 2014, pp. 25–40. ISBN: 9783319106953.

[27] M. Kwiatkowska, G. Norman, and D. Parker. “PRISM 4.0: Verification of Probabilistic Real-time Systems”. In: *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV’11)*. Ed. by G. Gopalakrishnan and S. Qadeer. Springer, 2011, pp. 585–591.

[28] A.N. Langville and W.J. Stewart. “The Kronecker product and stochastic automata networks”. In: *Journal of Computational and Applied Mathematics* 167.2 (2004), pp. 429–447.

[29] C. Moler and C. Van Loan. “Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later”. In: *Society for Industrial and Applied Mathematics (SIAM) Review* 45.1 (2003), pp. 3–49.

[30] R. Orús. “A Practical Introduction to Tensor Networks: Matrix Product States and Projected Entangled Pair States”. In: *Annals of Physics* 349 (2004), pp. 117–158.

[31] I.V. Oseledets. “Tensor-Train Decomposition”. In: *Society for Industrial and Applied Mathematics (SIAM) Journal on Scientific Computing* 33.5 (2011), pp. 2295–2317.

[32] R.N.C. Pfeifer, J. Haegeman, and F. Verstraete. “Faster identification of optimal contraction sequences for tensor networks”. In: *Physical Review E* 90 (3 2014), p. 033315.

[33] B. Philippe, Y. Saad, and W.J. Stewart. “Numerical Methods in Markov Chain Modeling”. In: *Operations Research* 40.6 (1992), pp. 1156–1179.

[34] B. Plateau. “On the Stochastic Structure of Parallelism and Synchronization Models for Distributed Algorithms”. In: *Proceedings of the 1985 ACM SIGMETRICS Conference on Measurement and Modeling of Computer System*. Ed. by H.D. Schwetman et al. ACM, 1985, pp. 147–154. ISBN: 9780897911696.

[35] B. Plateau and K. Atif. “Stochastic Automata Network For Modeling Parallel Systems”. In: *IEEE Transactions on Software Engineering* 17.10 (1991), pp. 1093–1108.

[36] B. Plateau and W.J. Stewart. “Stochastic Automata Networks”. In: *Computational Probability*. Ed. by W.K. Grassmann. 1st ed. Springer, 2000. Chap. 5, pp. 113–151. ISBN: 9781441951007.

[37] S.-J. Ran et al. “Review of Tensor Network Contraction Approaches”. In: *ArXiv e-prints* (2017). arXiv: *1708.09213* `[physics.comp-ph]`.

[38] A. Sales and B. Plateau. “Reachable state space generation for structured models which use functional transitions”. In: *Proceedings of the 6th International Conference on the Quantitative Evaluation of Systems (QEST’09)*. Ed. by S. Ceballos. IEEE Press, 2009, pp. 269–278. ISBN: 97807695338082.

[39] I. Sbeity et al. “Phase-type distributions in stochastic automata networks”. In: *European Journal of Operational Research* 186.3 (2008), pp. 1008 –1028.

[40] K. Schäcke. *On the Kronecker Product*. 2013.

[41] R.B. Sidje. “EXPOKIT: A Software Package for Computing Matrix Exponentials”. In: *Association for Computing Machinery Transactions on Mathematical Software (ACM TOMS)* 24.1 (1998), pp. 130–156.

[42] E. de Souza e Silva and H.R. Gail. “Transient Solutions for Markov Chains”. In: *Computational Probability*. Ed. by W.K. Grassmann. 1st ed. Springer, 2000. Chap. 3, pp. 43–79. ISBN: 9781441951007.

[43] W.J. Stewart. “MARCA: Markov Chain Analyzer, A Software Package for Markov Modeling”. In: *Numerical Solution of Markov Chains*. 1st ed. Marcel Dekker, Inc., 1991. Chap. 3, pp. 37–62. ISBN: 9780824784058.

[44] W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. 1st ed. Princeton University Press, 1994. ISBN: 9780691036991.

[45] W.J. Stewart. “Numerical Methods for Computing Stationary Distributions of Finite Irreducible Markov Chains”. In: *Computational Probability*. Ed. by W.K. Grassmann. 1st ed. Springer, 2000. Chap. 4, pp. 81–111. ISBN: 9781441951007.

[46] W.J. Stewart, K. Atif, and B. Plateau. “The Numerical Solution of Stochastic Automata Networks”. In: *European Journal of Operational Research* 86.3 (1996), pp. 503–525.

[47] E. Uysal and T. Dayar. “Iterative methods based on splittings for stochastic automata networks”. In: *European Journal of Operational Research* 110.1 (1998), pp. 166–186.

[48] W. Whitt. *Continuous-Time Markov Chains*. 2013.