

Colored Petri Nets for Systems Biology

Von der Fakultät für Mathematik, Naturwissenschaften und Informatik
der Brandenburgischen Technischen Universität Cottbus

zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
(Dr. rer. nat.)

genehmigte Dissertation

vorgelegt von

PhD Engineering

Fei Liu

geboren am 24. 11. 1976 in Pingdu, China

Gutachter: Prof. Dr.-Ing. Monika Heiner

Gutachter: Prof. Dr. rer. nat. Wolfgang Marwan

Gutachter: Prof. David Gilbert

Tag der mündlichen Prüfung: 31.01.2012

Acknowledgements

Firstly, I wish to express my deep gratitude to my supervisor Professor Monika Heiner for giving me this research opportunity and for her constant support and advice during this work.

Secondly, I would like to thank Professor Wolfgang Marwan and Professor David Gilbert for their careful reviews and valuable discussions and comments. I also thank all the other members in my defence committee.

Next, I am very grateful to all members in the Data Structure and Software Dependability Chair for their daily help, assistance and discussions, including Mrs Sigrid Schenk and Mostafa Herajy, Christian Rohr and Martin Schwarick. Special thanks go to Mary Ann Blätke for her kind help and discussions about systems biology.

Further, my warmest thanks belong to my family for their support. I am grateful to all the other people and friends who helped me during this work.

Last but not least, I acknowledge the financial support of German Federal Ministry of Education and Research (BMBF) grand 0315449H.

Cottbus
Fei Liu

Abstract

Modeling plays a crucial role in *Systems Biology* in order to provide a system-level understanding of biological systems. With the rapid development of systems biology, modeling of biological systems has shifted from single scales to multiple scales. This introduces a series of challenges that should be addressed, e.g. repetition of components such as genes and cells, variation of components, or hierarchical organization of components. Traditional modeling approaches, e.g. Petri nets, can not afford to cope with these challenges, which, however, can be tackled using colored Petri nets.

This thesis aims to present a technology based on colored Petri nets and associated techniques to address challenges introduced by multiscale modeling in systems biology and to implement them in our modeling tool, Snoopy.

To this aim, we present a colored Petri net framework for systems biology, which relates three modeling paradigms: *colored qualitative Petri net* (QPN^C), *colored stochastic Petri net* (SPN^C) and *colored continuous Petri net* (CPN^C). Using this framework, we can model and analyze a biological system from three different perspectives: qualitative, stochastic and continuous by converting them into each other.

We implement this framework in our modeling tool, Snoopy, and therefore in this thesis we explore three key problems concerning the implementation of colored Petri nets. For animating/simulating colored Petri nets, we present an efficient algorithm for the computation of enabled transition instances. In order to utilize the analysis techniques of Petri nets we present an efficient unfolding algorithm for large-scale colored Petri nets. In addition, we discuss three special cases for automatic folding (colorizing): colorizing T-invariants, master nets and twin nets in order to reduce the amount of work for folding Petri nets.

Petri nets offer a large variety of analysis techniques ranging from informal techniques, e.g. animation/simulation to formal techniques, e.g. model checking. We summarize those analysis techniques that can be used for colored Petri nets, e.g. structural analysis, numerical and simulative model checking from the application point of view.

We discuss some scenarios to illustrate the potential capability of colored Petri nets to cope with challenges in systems biology. Moreover, we apply our colored Petri net technology and techniques to three case studies, *C. elegans* vulval development, coupled Ca^{2+} channels and membrane systems. These case studies not only demonstrate how to use the colored Petri net framework and related analysis techniques for modeling and analyzing biological systems, but also show how to address the challenges of systems biology.

Keywords Systems Biology; colored Petri nets; stochastic Petri nets; continuous Petri nets; unfolding; folding; analysis techniques

Zusammenfassung

Die Modellierung ist für die Systembiologie von zentraler Bedeutung, um biologische Systeme ganzheitlich auf einer Systemebene zu verstehen. Mit der rasanten Entwicklung der Systembiologie hat sich die Modellierung biologischer Systeme von single-skalen zu multi-skalen Systemen verschoben. Dies führt zu einer Reihe von Herausforderungen, die bewältigt werden müssen, unter anderem die Modellierung der Vervielfältigung, Variation und hierarchischen Organisation von Komponenten, z. B. Gene und Zellen. Traditionelle Modellierungsansätze, wie z. B. Petrinetze, sind diesen Herausforderungen nicht gewachsen; sie können jedoch mit gefärbten Petrinetzen in Angriff genommen werden.

Das Ziel dieser Arbeit besteht in der Einführung einer Technologie, die auf gefärbten Petrinetzen und zugehörigen Techniken beruht, um den Herausforderungen in der Systembiologie durch Multiskale-Modellierung zu begegnen, und deren Implementierung in unserem Modellierungswerkzeug Snoopy.

Für diesen Zweck präsentieren wir für die Systembiologie ein Framework, das auf gefärbten Petrinetzen beruht und drei Modellierungsparadigmen umfasst: gefärbte qualitative Petrinetze (QPN^C), gefärbte stochastische Petrinetze (SPN^C) und gefärbte kontinuierliche Petrinetze (CPN^C). Mit diesem Framework können wir ein biologisches System aus drei Perspektiven modellieren und analysieren: qualitativ, stochastisch und kontinuierlich, unterstützt durch deren Umwandelbarkeit untereinander.

Wir implementieren dieses Framework in unserem Modellierungswerkzeug Snoopy und untersuchen dazu in der vorliegenden Arbeit drei wesentliche Probleme bei der Umsetzung der gefärbten Petrinetze. Zur Animation/Simulation gefärbter Petrinetze stellen wir einen effizienten Algorithmus zur Berechnung von schaltfähigen Transitionsinstanzen vor. Um die Analyse-Techniken von Petrinetzen zu nutzen, stellen wir einen effizienten Algorithmus zur Entfaltung sehr großer gefärbter Petrinetze vor. Um den Aufwand für die Färbung von Petrinetzen zu reduzieren, diskutieren wir außerdem drei Sonderfälle für die automatische Faltung (Färbung): Färbung von T-Invarianten, von Master-Netzen sowie von Twin-Netzen.

Petrinetze bieten eine große Vielfalt von Analysetechniken an, die von der informellen Analyse, z. B. Animation/Simulation, bis zu formalen Techniken, z. B. Modelchecking, reichen. Wir geben aus Anwendersicht einen Überblick über die Analysetechniken, die für gefärbte Petrinetze verwendet werden können, z. B. Strukturanalyse sowie analytisches und simulatives Modelchecking.

Wir diskutieren einige Szenarien, in denen gefärbte Petrinetze weiterhelfen, um ihr Potential bei der Bewältigung der Herausforderungen in der Systembiologie zu illustrieren. Um die Anwendbarkeit unseres Ansatzes zu demonstrieren, wenden wir unsere Methode der gefärbten Petrinetze auf drei Fallstudien an, C. Elegans, gekoppelte Ca^{2+} -Kanäle und Membran-Systeme. Diese Fallstudien zeigen nicht nur, wie das Framework

der gefärbten Petrinetze und der zugehörigen Analyse-Techniken für die Modellierung und Analyse praktischer biologischer Systeme genutzt werden kann, sondern zeigen auch, wie man den Herausforderungen der Systembiologie begegnen kann.

Freie Schlagwörter Systembiologie; gefärbte Petrinetze; stochastische Petrinetze; kontinuierliche Petrinetze; Entfaltung; Faltung; Analyse-Techniken

Contents

| | |
|---|-----------|
| Abstract | v |
| Zusammenfassung | vi |
| 1 Introduction | 1 |
| 1.1 Motivation | 1 |
| 1.2 Background | 4 |
| 1.2.1 Petri Nets | 4 |
| 1.2.2 Colored Petri Nets | 7 |
| 1.2.3 Systems Biology | 9 |
| 1.3 Contributions | 14 |
| 1.4 Organization of Thesis | 16 |
| 2 A Colored Petri Net Framework | 17 |
| 2.1 Overview | 17 |
| 2.2 Colored Petri Nets | 19 |
| 2.2.1 Multiset | 19 |
| 2.2.2 Definition | 20 |
| 2.2.3 Dynamic Behavior | 21 |
| 2.3 Colored Qualitative Petri Nets | 22 |
| 2.3.1 Extended Petri Nets | 22 |
| 2.3.2 Colored Qualitative Petri Nets | 25 |
| 2.4 Colored Stochastic Petri Nets | 28 |
| 2.4.1 Stochastic Petri Nets | 29 |
| 2.4.2 Deterministic and Stochastic Petri Nets | 30 |
| 2.4.3 Colored Stochastic Petri Nets | 31 |
| 2.5 Colored Continuous Petri Nets | 32 |
| 2.5.1 Continuous Petri Nets | 33 |
| 2.5.2 Colored Continuous Petri Nets | 34 |
| 2.6 Scenarios for Using Colored Petri Nets in Systems Biology | 35 |
| 2.7 Encoding Components of Systems as Colors | 37 |
| 2.8 Closing Remarks | 39 |
| 3 Some Implementation Aspects | 41 |

| | | |
|----------|--|-----------|
| 3.1 | Computation of Enabled Transition Instances | 42 |
| 3.1.1 | Patterns | 43 |
| 3.1.2 | Binding Process | 46 |
| 3.1.3 | Algorithms | 48 |
| 3.1.4 | Optimization Techniques | 53 |
| 3.1.5 | Related Work | 56 |
| 3.1.6 | Conclusions | 56 |
| 3.2 | Unfolding of Colored Petri Nets | 57 |
| 3.2.1 | Equivalent Standard Petri Nets | 58 |
| 3.2.2 | Unfolding Algorithm | 59 |
| 3.2.3 | Algorithms for Computing Transition Instances | 59 |
| 3.2.4 | Optimization Techniques | 66 |
| 3.2.5 | Experimental Results | 66 |
| 3.2.6 | Related Work | 68 |
| 3.2.7 | Conclusions | 68 |
| 3.3 | Folding of Petri Nets | 68 |
| 3.3.1 | Colorizing T-invariants of Petri Nets | 69 |
| 3.3.2 | Colorizing Master Petri Nets | 71 |
| 3.3.3 | Colorizing Twin Nets | 75 |
| 3.3.4 | Conclusions | 77 |
| 3.4 | Closing Remarks | 77 |
| 4 | Analysis Techniques | 79 |
| 4.1 | Structural Analysis | 79 |
| 4.2 | Model Checking | 80 |
| 4.2.1 | Linear Temporal Logic | 81 |
| 4.2.2 | Computation Temporal Logic | 81 |
| 4.2.3 | Model Checking of QPN^C | 82 |
| 4.3 | Numerical Model Checking | 82 |
| 4.3.1 | Continuous Stochastic Logic | 83 |
| 4.3.2 | CSL Model Checking of SPN^C | 83 |
| 4.4 | Simulative Model Checking | 84 |
| 4.4.1 | Probabilistic Linear Temporal Logic with Numerical Constraints | 85 |
| 4.4.2 | PLTLc Model Checking of SPN^C | 86 |
| 4.5 | Analysis of QPN^C Using CPN Tools | 88 |
| 4.5.1 | CPN Tools | 88 |
| 4.5.2 | Transformation from QPN^C Models to CP-Net Models | 88 |
| 4.5.3 | Analysis of QPN^C Models with CPN Tools | 89 |
| 4.6 | Discussions | 90 |
| 4.6.1 | Comparison of Two Approaches: Folded versus Unfolded | 90 |
| 4.6.2 | Partial Unfolding - Tackling Dynamic Color Sets | 92 |

| | | |
|----------|--|------------|
| 4.7 | Closing Remarks | 92 |
| 5 | Case Studies | 93 |
| 5.1 | Modeling C. Elegans Vulval Development | 94 |
| 5.1.1 | C. Elegans Vulval Development | 94 |
| 5.1.2 | Modeling | 95 |
| 5.1.3 | Structural Analysis | 99 |
| 5.1.4 | Determining the Fate of VPCs Using Simulative Model Checking | 101 |
| 5.1.5 | Results and Discussions | 103 |
| 5.1.6 | Conclusions | 109 |
| 5.2 | Modeling Coupled Ca^{2+} Channels | 109 |
| 5.2.1 | Ca^{2+} -Regulated Ca^{2+} Channels | 111 |
| 5.2.2 | Modeling | 113 |
| 5.2.3 | Analysis and Validation | 115 |
| 5.2.4 | Discussions | 122 |
| 5.2.5 | Conclusions | 123 |
| 5.3 | Modeling Membrane Systems | 123 |
| 5.3.1 | Membrane Systems | 124 |
| 5.3.2 | Modeling Using Petri Nets | 127 |
| 5.3.3 | Modeling Using Colored Petri Nets | 132 |
| 5.3.4 | An Example: the Viral Infection | 134 |
| 5.3.5 | Conclusions | 139 |
| 5.4 | Closing Remarks | 139 |
| 6 | Conclusions and Outlook | 141 |
| 6.1 | Conclusions | 141 |
| 6.2 | Outlook | 142 |
| | Bibliography | 145 |

Contents

List of Figures

| | | |
|------|---|----|
| 1.1 | A Petri net model for the repressilator | 6 |
| 1.2 | A colored Petri net model for the repressilator | 8 |
| 1.3 | A diagrammatic representation of some biological scales. | 10 |
| 2.1 | A colored Petri net framework | 18 |
| 2.2 | Special arcs drawn in Snoopy | 23 |
| 2.3 | An extended Petri net | 25 |
| 2.4 | A QPN^C model for Figure 2.3 | 27 |
| 2.5 | A Petri net model for circadian rhythms. | 27 |
| 2.6 | A QPN^C model for circadian rhythms | 28 |
| 2.7 | A CPN^C model for circadian rhythms | 35 |
| 2.8 | An arrangement of M components in a line. | 38 |
| 2.9 | An arrangement of $M \times N$ components in a two dimensional lattice. | 39 |
| 3.1 | Demonstrating patterns | 43 |
| 3.2 | Demonstrating the partial binding - partial test principle | 55 |
| 3.3 | Demonstrating the constraint satisfaction approach | 61 |
| 3.4 | A colored Petri net for computing transition instances. | 64 |
| 3.5 | The unfolded Petri net for Figure 3.4. | 65 |
| 3.6 | A diffusion model for testing the unfolding efficiency. | 67 |
| 3.7 | A Petri net model for colorizing T-invariant | 70 |
| 3.8 | A colored Petri net model for Figure 3.7 | 71 |
| 3.9 | A colored Petri net model for the hypoxia response network | 72 |
| 3.10 | Two Petri net models for demonstrating master nets | 73 |
| 3.11 | A colored Petri net model for Figure 3.10 | 73 |
| 3.12 | A Petri net model for the sensory control of sporulation | 74 |
| 3.13 | A colored Petri net model for the sensory control of sporulation | 74 |
| 3.14 | The signaling subnetwork for one plasmodium | 75 |
| 3.15 | A colored Petri net model for Figure 3.14 | 76 |
| 3.16 | A colored Petri net model for Figure 3.14 with a coarse node | 76 |
| 4.1 | Stochastic simulation result for the repressilator model | 86 |
| 4.2 | A colored Petri net model for cooperative ligand binding | 89 |
| 4.3 | A CP-net model for Figure 4.2 | 90 |

| | | |
|------|---|-----|
| 5.1 | Spatial patterning of VPCs of vulval development | 95 |
| 5.2 | A \mathcal{SPN}^C model for <i>C. elegans</i> vulval development | 97 |
| 5.3 | Stochastic simulation result averaged over 38,000 runs for VPC 3. | 105 |
| 5.4 | Stochastic simulation result averaged over 38,000 runs for VPC 4. | 105 |
| 5.5 | Stochastic simulation result averaged over 38,000 runs for VPC 5. | 106 |
| 5.6 | Stochastic simulation result averaged over 38,000 runs for VPC 6. | 106 |
| 5.7 | Stochastic simulation result averaged over 38,000 runs for VPC 7. | 107 |
| 5.8 | Stochastic simulation result averaged over 38,000 runs for VPC 8. | 107 |
| 5.9 | A two-state channel with Ca^{2+} activation. | 111 |
| 5.10 | A six-state channel with Ca^{2+} activation and inactivation. | 112 |
| 5.11 | Two instantaneously coupled two-state channels | 112 |
| 5.12 | \mathcal{SPN} models for two-state Ca^{2+} channels | 113 |
| 5.13 | A \mathcal{SPN} model for the six-state channel | 114 |
| 5.14 | \mathcal{SPN}^C models for two-state Ca^{2+} channels | 116 |
| 5.15 | A \mathcal{SPN}^C model for coupled six-state Ca^{2+} channels | 117 |
| 5.16 | Stochastic simulation result for the model of 19 two-state channels | 120 |
| 5.17 | Stochastic simulation result for the model of 4 two-state channels | 121 |
| 5.18 | Stochastic simulation result for the model of 19 six-state channels | 121 |
| 5.19 | A basic membrane system. | 126 |
| 5.20 | A dynamic membrane system. | 127 |
| 5.21 | Petri net models for typical evolution rules | 129 |
| 5.22 | A Petri net model for the basic membrane system | 129 |
| 5.23 | A Petri net model for the dynamic membrane system | 131 |
| 5.24 | A colored Petri net model for the basic membrane system | 133 |
| 5.25 | A colored Petri net model for the dynamic membrane system | 134 |
| 5.26 | The viral infection process | 135 |
| 5.27 | A \mathcal{SPN} model for the viral infection. | 136 |
| 5.28 | A \mathcal{SPN}^C model for the virus infection. | 137 |
| 5.29 | Stochastic simulation result with 1 initial virus molecule | 138 |
| 5.30 | Stochastic simulation result with 10 initial virus molecule | 138 |
| 5.31 | Stochastic simulation result with 100 initial virus molecule | 139 |

List of Tables

| | | |
|------|---|-----|
| 2.1 | Rate functions of the \mathcal{SPN}^c model for Circadian rhythms | 32 |
| 3.1 | Declarations for the diffusion model in Figure 3.6. | 67 |
| 3.2 | Comparison of the size of the diffusion model and unfolding runtime . . | 68 |
| 3.3 | The minimal T-invariants of the hypoxia response network model. | 71 |
| 5.1 | Descriptions of some transitions in the \mathcal{SPN}^c model | 98 |
| 5.2 | Descriptions of some places in the \mathcal{SPN}^c model | 99 |
| 5.3 | The T-invariants in the MAPK pathway of each VPC | 100 |
| 5.4 | The T-invariants in the LIN-12/Notch signaling pathway of VPC 4 . . . | 100 |
| 5.5 | Fate patterns to be validated, excerpted from [LNUM09] | 104 |
| 5.6 | Detailed statistical results of 50 simulations for the unstable patterns . . | 108 |
| 5.7 | The minimal T-invariants of the one six-state channel model | 118 |
| 5.8 | The minimal P-invariants of the two coupled six-state channels model . | 118 |
| 5.9 | The minimal T-invariants of the two coupled six-state channels model . | 119 |
| 5.10 | Comparison of the state space construction | 120 |
| 5.11 | The size of the Ca^{2+} model and its unfolding and simulation runtime . . | 123 |

1 Introduction

1.1 Motivation

Systems Biology [Ade05], [IWL06], [Kit02], [SW05] is an emerging scientific discipline in bioscience research, which aims to understand the behavior of a biological system at the system level by means of investigating the behavior and interactions of all of the components in the system. To achieve the system-level understanding, modeling plays a crucial role in systems biology [FHL⁺04]. Modeling is a widely used scientific approach to represent, explain, analyze and predict the system behavior, thus facilitating the understanding of a (biological) system from a holistic viewpoint. Usually modeling follows some standard steps, e.g. construction, simulation, validation and analysis. With the increasing magnitude and complexity of interactions in biological systems and the rapid growth of data being generated in the biological field, an overwhelmingly increasing number of challenges are introduced to modeling of biological systems.

A large variety of modeling approaches, e.g. Petri nets, Boolean networks and ordinary differential equations, have already been applied to modeling a wide field of biological systems (see [BCMS10], [HK09] for a review). Among them, Petri nets are particularly suitable for representing and modeling the concurrent, asynchronous and dynamic behavior of biological systems. Since Reddy et al. [RML93] introduced qualitative Petri nets to model metabolic pathways, a large variety of applications of Petri nets (e.g. stochastic Petri nets, timed Petri nets, continuous Petri nets and hybrid Petri nets) have been developed for modeling and simulating different types of biological systems [BCMS10], [GH06]. For example, due to inherently stochastics in biological processes, stochastic Petri nets have recently become a popular modeling paradigm for capturing their complex dynamics, which can help to understand the behavior of complex biological systems by integrating detailed biochemical data and providing quantitative analysis results, see e.g. [GP98], [PRA05].

Petri nets offer a number of attractive advantages for modeling biological systems [HGD08]:

- intuitive graphical and executable modeling formalisms,
- rich mathematically founded analysis techniques, covering structural and behavioral properties,
- integration of qualitative and quantitative analysis techniques and methods, and

- a wealth of computer tool support.

However, standard Petri nets do not easily scale. So attempts to simulate biological systems by standard Petri nets have been mainly restricted so far to relatively small models. Standard Petri nets tend to grow quickly for modeling complex systems, which makes it more difficult to manage and understand the nets, thus increasing the risk of modeling errors. Two known modeling concepts improving this situation are hierarchy and color. Hierarchical structuring has been discussed a lot, e.g. in [MWW11], while the color has gained little attention so far.

Colored Petri nets [GL79], [GL81], [Jen81] are a colored extension of standard Petri nets, where a group of similar components are represented by one component, each of which is defined as and thus distinguished by a color. Colored Petri nets provide parameterized and compact representations of complex biological systems; however they do not lose the analysis capabilities of standard Petri nets, which can still be supported by automatic unfolding. Moreover, another attractive advantage of colored Petri nets for a biological modeler is that they provide the possibility to easily increase the size of a model consisting of many similar subnets just by adding new colors.

While there is a lot of reported work on the application of different classes of standard Petri nets to a variety of biochemical networks, see [BCMS10] for a recent review, there are only a few which take advantage of the additional power and ease of modeling offered by colored Petri nets. To our knowledge, the existing applications of colored Petri nets in systems biology can be summarized as follows.

The approach taken in [GKV01], [LZLP06] to use colored Petri nets is to encode the concentration of species as colored tokens in order to implement continuous simulation in the given net annotation language ML in Design/CPN or CPN tools. While this is a nice exercise in demonstrating the power of the annotation language, the burden to implement standard simulation algorithms is left to the modeler.

Colored Petri nets have been used for qualitative modeling and analysis in [PGA02] to predict pathological phenotypes based on genetic mutations and in [TMK⁺06] to model signal transduction networks. Here, colors encode mutations of the modeled molecules or distinguish between different molecules via their identifiers (colors). Colors have also been used to discriminate metabolites which follow different T-invariants (elementary flux modes) [HKV01], [Run04], [VHK03].

Benefits of colored Petri nets in a stochastic setting were first demonstrated in [BP03] using a very simple epidemic model. The host population is divided into at-risk classes, which are modeled as places, and color is used to encode the serological state of individuals (e.g. susceptible, infected, removed).

From this evaluation of related work, we can see that existing studies usually resort to Design/CPN [CJK97] or its successor CPN tools [RWL⁺03] in order to model and

analyze biological systems. However neither tool was specifically designed with the requirements of systems biology in mind. Thus they are not suitable in many aspects, e.g. they do not directly support stochastic or continuous modeling, nor the simulative analysis of the models by stochastic or deterministic simulation.

In addition, due to the ability to produce data of the same phenomenon at different scales, modeling of biological systems shifts from single scales to multiple scales, e.g. from the molecular scale to the cell level, the tissue level and the whole organism level [Dal10], [MFV10]. Modeling biological systems beyond one spatial scale (multi-scale modeling) introduces a series of challenges which should be addressed. In the following a component can be a gene, a molecule, a cellular compartment, a cell, a multicellular complex, a tissue, an organ, an organism, a population etc. in the biological context. Systems biology is distinguished by the following challenges according to [Dal10], [GLG⁺11], [MFV10]:

- *Repetition of components* – for example in the tissue modeling there may be the need to describe multiple cells each of which has a similar definition.
- *Variation of components* – sets of similar components with defined variations, e.g. mutants. For example, variations may exist among cells in a multicellular biological system.
- *Organization of components* – for example how cells are organized into regular or irregular patterns over spatial networks in one, two or three dimensions in the tissue modeling.
- *Communication between components* – for example, quorum sensing takes place among cell populations in one, two or three dimensional space.
- *Movement of components* – some components have the ability to move in a certain region, e.g. molecules within an individual cell or cells within a tissue.
- *Hierarchical organization of components* – enabling the description of (possibly repeated) components which contain repeated sub-components. For example, in the tissue modeling, tissues contain a number of cells and cells contain several compartments.
- *Differentiation of components* – for example, differentiation of embryonic stem cells or immune cells makes a less specialized cell more specialized.
- *Replication of components* – e.g. cell division.
- *Deletion of components* – e.g. cell death.

- *Pattern formation of components* – organizing a number of cells in appropriate one, two or three dimensional structures in space and time.

Moreover, in modeling of multiscale biological systems, we usually can not just consider one of those aspects but a combination of some aspects. All these challenges potentially could be tackled by colored Petri nets. This thesis will particularly explore how to use colored Petri nets to address these challenges in systems biology.

This thesis aims to present a technology based on colored Petri nets and associated techniques to address challenges introduced by multiscale modeling in systems biology and offer a solution to perform multiscale modeling and analysis of biological systems.

In order to implement this aim, we extend our software tool Snoopy [RMH10], building upon the lessons learned so far, by specific functionalities and features to support editing, simulating and analyzing of biological models based on colored qualitative, stochastic and continuous Petri nets [LH10a].

This technology and associated techniques can be applied to all fields where colored Petri nets have a long tradition, e.g. technical systems.

Specifically they can be applied to other kinds of biological systems, e.g. synthetic biology [CBW08], [LLZ11], although all case studies in this thesis are about systems biology.

Besides, systems biology as well as synthetic biology can take advantage of biomodel engineering to systematically design, construct and analyze biological systems using computational tools and techniques in a principled way [BDGH10]. Our technology and associated techniques can be immediately integrated into biomodel engineering to offer a solution for multiscale modeling of biological systems.

1.2 Background

1.2.1 Petri Nets

Petri nets originating from the dissertation of Carl Adam Petri [Pet62], are an excellent modeling formalism for describing and studying systems that are characterized as being concurrent, asynchronous, distributed, parallel or nondeterministic. The formalism combines an intuitive graphical notation with a number of advanced analysis techniques with a firm mathematical foundation. Petri nets are applied in practice by industry, academia, and other fields to the analysis of systems arising in asynchronous circuit design, communication protocols, distributed computing, production systems, flexible manufacturing, transportation, systems biology etc. Particularly in systems biology different classes of Petri nets have been used to model a large variety of biological systems, see [BCMS10] for a recent review.

Petri nets are weighted, directed, bipartite graphs, consisting of places, transitions and arcs that connect them. Places usually represent passive system components like conditions or resources, while transitions represent active system components like events. In systems biology, places may represent species or any kind of chemical compounds, e.g. genes, proteins or proteins complexes, while transitions represent any kind of chemical reactions, e.g. association, disassociation, translation or transcription. For a chemical reaction, its precursors correspond to the preplaces of a transition while its products to the postplaces of the transition. The arcs lead from places to transitions, or from transitions to places, whose weights indicate the multiplicity of each arc, reflecting e.g. stoichiometries for chemical reactions. The arc weight 1 is usually not labeled explicitly. A place may contain an arbitrary (natural) number of tokens, represented as black dots or a natural number. A distribution of tokens over all places of a Petri net represents a state of it, which is called a marking.

A transition is called enabled if each of its preplaces contains at least the number of tokens specified by the weight of the corresponding arc. An enabled transition may fire and the firing of a transition transfers tokens from its preplaces to its postplaces according to their weights. The firing of a transition updates the current marking to a new reachable one. The repeated firing of transitions establishes the behavior of a net. The set of markings reachable from the initial marking constitutes the state space of the net. These reachable markings and transitions between them constitute the reachability graph of the net.

Petri nets can be characterized by some important structural and behavioral properties. Examining these properties is usually a key step in the analysis of Petri nets. Below we give an informal characterization of several of the most useful properties.

- *Reachability.* Reachability answers the question: is there a firing sequence for a given marking executable from the initial marking and finally reaching it. Reachability is fundamental for studying the dynamic behavior of Petri nets.
- *Boundedness.* Boundedness describes how many tokens a place may hold for a set of reachable markings. If all places of a net are bounded, the net is bounded. If it is bounded in any initial marking, it is called structurally bounded.
- *Liveness.* Liveness describes the possibility for a transition to be enabled and then to fire infinitely often. If each transition of a net is live, the net is live.
- *Reversibility.* Reversibility describes the possibility of a net being able to come back to a previous state. If a Petri net can reach its initial marking again from any reachable marking, the net is reversible.

Petri nets offer rich mathematically founded analysis techniques, which can be used to check their structural and behavioral properties. We briefly describe some important analysis techniques.

- *Graph-based structural analysis techniques.* These analysis techniques can be used to analyze graph properties of Petri nets, e.g. connectedness or strongly connectedness, which is an efficient preliminary consistency check.
- *Linear algebra-based analysis techniques.* A Petri net can be (partly) represented by an incidence matrix, which describes the change of the number of tokens in places due to the firing of transitions. Using it, *P-invariants* and *T-invariants* can be obtained through linear algebra operations.
- *State space analysis techniques.* These techniques are based on the construction of the reachability graph for a Petri net. When the reachability graph has been constructed, different behavioral properties can be decided through state space search.

We have introduced some basic background of Petri nets. For more details see, e.g. [HGD08] and [Mur89].

Now we give a simple Petri net model (illustrated in Figure 1.1) that is adapted from a stochastic π -machine model in [BCP08], modeling the well-known repressilator [EL00]. The repressilator is a regulatory cycle of three genes, e.g. denoted by G_a , G_b and G_c . Each gene generates its protein, e.g. P_a , P_b and P_c , respectively, by means of transcription and translation and proteins can degrade. Each protein represses the expression of its successor's gene, namely, P_a inhibits G_b , P_b inhibits G_c , and P_c inhibits G_a by binding to the gene to form a complex species, e.g. $G_b_P_a$, $G_c_P_b$ and $G_a_P_c$, respectively. Each complex species can be unbound.

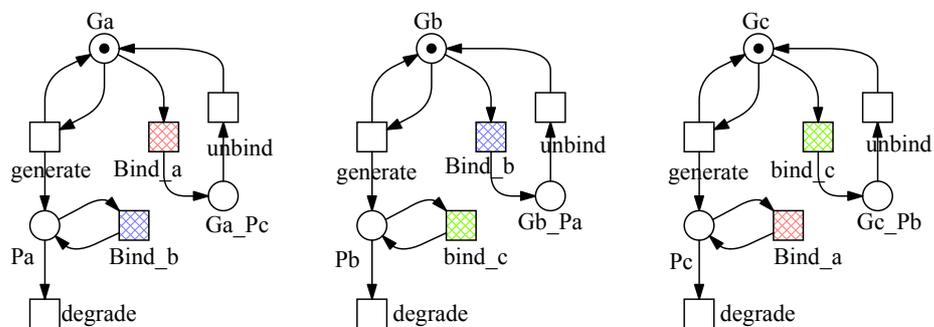


Figure 1.1: A Petri net model for the repressilator. In this model, the highlighted transitions are logic transitions to facilitate the readability of the net, e.g. the two transitions named *bind_a* are in fact the same transition with two graphic representations.

In this model, places represent such species as genes, proteins and complex species, and transitions represent such reaction rules as generation (transcription and translation),

degradation, binding and unbinding. Initially, there is a token for each gene respectively. As there are enough tokens, each of these three generation transitions can be fired. For example, when the transition for generating P_a fires, it will remove one token from and then return one token to place G_a , and add one token to place P_a .

1.2.2 Colored Petri Nets

Colored Petri nets [GL79], [GL81], [Jen81] are a colored extension of standard Petri nets. Since 1970s, both the theory and applications of Petri nets have been extensively researched. A number of analysis techniques, modeling tools and extensions of Petri nets have been proposed, one of which are colored Petri nets.

Since the introduction of colored Petri nets, several types of variants have been proposed by adding special restrictions to colored Petri nets. Regular nets [Had87] put a restriction on the color domains and on the arc functions, so that the existing algorithms for the computation of flows and reductions and the definition of the symbolic reachability graph can be applied. Unfortunately, these restrictions are so strong that they can only be used in very few scenarios. To alleviate these restrictions, the well-formed nets [CDFH93] have been proposed to implement the same purpose, the use of symbolic reachability graphs for performance evaluation.

Like standard Petri nets, colored Petri nets are also an excellent formalism for specifying, designing and analyzing concurrent systems, which also combine an intuitive graphical notation with a set of mathematically founded analysis techniques. They have been applied to protocols and networks, software, workflows and business processes, hardware, manufacturing systems and systems biology.

Colored Petri nets are a discrete event modeling formalism combining the strengths of Petri nets with the expressive power of programming languages. Petri nets provide the graphical notation and constructions for modeling systems with concurrency, communication and synchronization. The programming languages offer the constructions for the definition of data types, then used for creating compact models. This is the most important advantage of colored Petri nets.

Colored Petri nets consist, as standard Petri nets, of places, transitions and arcs. In systems biology, places also represent species or any kind of chemical compounds, while transitions represent any kind of chemical reactions or transport steps. Additionally, a colored Petri net model is characterized by a set of color sets with data types. A data type in a programming language is a set of values that obey some properties [CW85]. Examples of data types are integer, Boolean, and string. Each place gets assigned a color set and may contain distinguishable tokens colored with a color of this color set. For example, in Figure 1.2 that illustrates a colored Petri net model for the repressilator in Figure 1.1, we define a color set *Gene* of the enumeration type with three colors, a , b and c , distinguishing three genes. We use one place to represent three

similar objects, e.g. representing three protein objects as one colored place P . Each place gets assigned the color set, $Gene$. As there can be several tokens of the same color on a given place, the tokens on a place define a multiset over the place's color set. For example, in Figure 1.2, we denote the initial marking for the place P by a multiset expression $1'a++1'b++1'c$, which means one token of each color of $Gene$. A specific distribution of tokens on all places together constitutes a marking of a colored Petri net. Each transition has a guard, which is a Boolean expression over defined variables. The guard must be evaluated to true for the enabling of the transition. The trivial guard "true" is usually not explicitly given. For example, in Figure 1.2, all colored transitions have the trivial guard "true". Each arc gets assigned an expression; the result type of this expression is a multiset over the color set of the connected place. For example, in Figure 1.2, we define a variable x on $Gene$, which is used in arc expressions. The predecessor operator "-" in the arc expression $-x$ returns the predecessor of x in an ordered finite color set. If x is the first color, then it returns the last color. The result type of all expressions is a multiset over $Gene$.

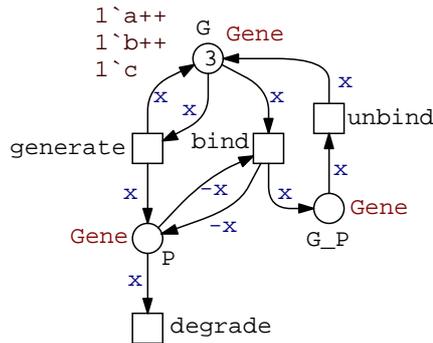


Figure 1.2: A colored Petri net model for the repressilator in Figure 1.1. The declarations: *colorset* $Gene = enum$ with a, b, c and *variable* $x : Gene$.

The variables associated with a transition consist of the variables in the guard of the transition and in the expressions of adjacent arcs. Before the expressions are evaluated, the variables must be assigned values with suitable data types, which is called binding [JKW07]. A binding of a transition corresponds to a transition instance. Enabling and firing of a transition instance are based on the evaluation of its guard and arc expressions. If the guard is evaluated to true and the preplaces have sufficient and appropriately colored tokens, the transition instance is enabled and may fire. When a transition instance fires, it removes colored tokens from its preplaces and adds colored tokens to its postplaces, i.e. it changes the current marking to a new reachable one. The colors of the tokens that are removed from preplaces and added to postplaces are

decided by the arc expressions. For example, in Figure 1.2, for the colored transition *generate*, there is only one related variable, x , which can be assigned the value, a , b or c , so it obtains three bindings, $x = a$, $x = b$ and $x = c$, i.e. it has three instances. As each binding has enough tokens for enabling, any of these three transition instances can be fired. For example, if we fire this transition under the binding $x = a$, one token with the color a is removed from and then added to place G and one token with the same color is added to place P . The set of markings reachable from the initial marking constitutes the state space of a given net. These reachable markings and instances of transitions between them constitute the reachability graph of the net.

There are also a variety of structural and behavioral properties for colored Petri nets, such as reachability, boundedness, liveness and reversibility. They have a similar meaning as in standard Petri nets. Most properties can be decided by analyzing the state space of a colored Petri net. For more details see [JKW07] and [JK09].

In addition, please note that the colored model in Figure 1.2 when unfolded yields the uncolored Petri net model in Figure 1.1. From Figure 1.2, we can see that the colored Petri net model reduces the size of the original Petri net model to one third. More importantly, when other similar subnets have to be added, the model structure does not need to be modified; what has to be done is only to add extra colors by changing the color set definition.

1.2.3 Systems Biology

Systems Biology originated from the molecular biology and genomic biology revolutions [IWL06]. With their revolutions, the study of biological systems is recognized not only to focus on isolated components, but to consider also the interactions of all these components. Systems biology has an important impact on understanding living systems and on giving rise to advances in diagnosis, treatment, prevention and relieving of human diseases.

There is no widely accepted definition of systems biology, although several definitions have been proposed. For example, Ideker et al. [IGH01] defines that systems biology "does not investigate individual genes or proteins one at a time", but "investigates the behavior and relationships of all of the elements in a particular biological system while it is functioning". Kitano [Kit02] describes systems biology as that "to understand biology at the system level, we must examine the structure and dynamics of cellular and organismal function, rather than the characteristics of isolated parts of a cell or organism". Popel et al. [PH09] define systems biology "as a field of study that takes into account complex interactions in biological systems at different scales of biological organization, from the molecular to cellular, organ, organism, and even societal and ecosystem levels".

As described by all those definitions, the study of systems biology shifts from com-

ponents of a system to the structure and dynamics of that system, so it gives the system-level understanding of the system with the use of mathematical and computational techniques. With the increasing magnitude and complexity of interactions in biological systems and the rapid growth of quantitative biological data, modeling plays a more and more important role in the study of systems biology.

Accordingly, due to the ability to produce data of the same phenomenon at different scales, modeling of biological systems shifts from single biological scales to multiple scales (*multiscale modeling*), e.g. from the molecular scale to the cell, tissue, and the whole organism level [Dal10], [MFV10], [PH09]. See Figure 1.3 for a diagrammatic representation of some different biological scales and some of their hallmark phenomena [MSFK09], [MFV10]. As shown in this figure, in a biological system, each level is composed of components of its lower ones, e.g. molecules compose cells, cells integrate themselves into tissues, and tissues organize themselves into organs. For each level, there are some particular phenomena, which are also the issues that modeling focuses on. Most importantly, those large data sets have pushed us to integrate all these facets [MFV10]. Multiscale modeling has become one of the most important issues in the study of systems biology [PH09]. Modeling biological systems beyond one spatial scale introduces a series of challenges, which will be discussed in the following.

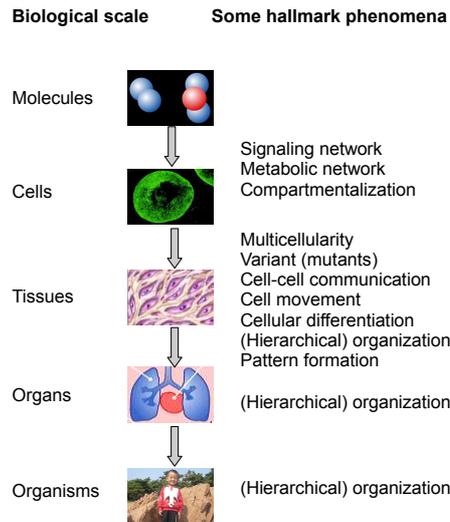


Figure 1.3: A diagrammatic representation of some biological scales.

(1) Repetition of components.

As depicted in Figure 1.3, each level is composed of components of its lower level, e.g. cells integrate themselves into tissues and tissues organize themselves into organs. To

model this, we usually have to describe multiple components, e.g. cells and tissues, each of which has a similar definition.

For example, at the intracellular level, consider a taxis signal transduction network consisting of multiple receptors and transducers [SOM10]. We have to distinguish and model each receptor or transducer if we want to clearly describe the effects among them. Consider another example, Ca^{2+} release site models that are composed of a number of individual Ca^{2+} channel models [DLKS08]. In order to clarify the effects among these channels, each of these channels has to be represented by an individual model with an identical structure.

Another challenge is the modeling of multicellular biological systems [HGBT09], [PUS11], e.g. an adult *Drosophila* wing comprising about 30,000 cells [GLG⁺11]. In this situation, we also have to face models with a high number of components of a similar definition.

For biological systems with repetitive components, we can easily use colored Petri nets to model them, where we only need to define each similar component as a color of a color set standing for the set of similar components and to develop a model for one component only. Usually, colored Petri nets can reduce the model size of repetitive components to that of one component by representing repetitive components with one component model, each of which is encoded and distinguished by a color. Thus colored Petri nets make it possible to model such kind of large-scale biological systems.

(2) Variation of components.

Variation, e.g. mutation [KB71], [PGA02] widely exists in biological systems and it is important to understand the effects that a variation brings. In multicellular biological systems, we often have to consider variations among cells [BMD07]. To do this, we have to add different mutant conditions to these models and change them conveniently for simulation. Using colored Petri nets, we can easily deal with this issue, where we can encode mutations as colors and switch a mutation by changing the parameters of a colored place or transition relating to this mutation. In this case, color is used to mark the differences between normal and mutant cells.

(3) Organization of components.

Every component in a biological system has a spatial aspect, and to consider this during modeling is sometimes essential [GAS⁺06], [NJT⁺05], [PO00]. In the example of coupled Ca^{2+} channels, the local Ca^{2+} concentration experienced by a channel is affected not only by the number of other open channels but also their distances to this channel. Here we have to arrange these channels in regular or irregular space in order to consider the distance between two channels [DLKS08]. It is also natural to organize cells in a tissue or tissues in an organ if it is not acceptable to neglect this spatial aspect.

Moreover, sometimes we have to consider a hierarchical organization of components.

For example, there are two hierarchies when we model cells which contain several compartments or three hierarchies if we build a model involving three levels, cells, tissues and organs.

To address these issues, the selected modeling tools have to support spatial and hierarchical modeling. Fortunately, colored Petri nets provide all these capabilities. Using the hierarchical modeling capability and hierarchical color sets, we can easily address all these problems. In this case, colors encode discrete positions of elements in one, two or three dimensional space.

(4) Communication between components.

The understanding of the communication mechanism between cells, e.g. quorum sensing (QS) is another fundamental issue in systems biology, which has great impact on biological processes such as development, growth and tissue repair [JKT⁺01], [MKH⁺06]. QS takes place among cell populations and depends on the cell density, which can be organized in one, two or three dimensional space. By means of colored Petri nets, we can easily construct such communication processes, in which we still use colors to represent each cell.

(5) Movement of components.

Some components in biological systems have the ability to move in a certain region, e.g. molecules within an individual cell or cells within a tissue, which results in the rearrangement and sorting of these components, possibly forming patterns. For example, coordinated movements of cells or tissues will lead to larger tissue rearrangements [PO00]. Such motion plays a crucial role throughout the life span of a number of organisms, which can occur at different levels, e.g. molecules, cells or tissues [PO00]. Reference [PSQH06] gives an example of the modeling of molecule movement in a cell. In [GMR08], six different types of cells move in a two dimensional grid in order to investigate the regeneration of the chronic chagasic cardiomyopathy after bone marrow stem cell transplantation.

In this situation, colored Petri nets provide an interesting solution, where each element of a grid is considered as a color. If a component occupies a grid element, it will have the color of this element. The movement of a component from a grid element to another means to change the color of the former grid element to that of the new grid element.

(6) Differentiation of components.

Differentiation widely occurs during the development of a biological system. For example, cellular differentiation turns a less specialized cell into a more specialized cell type, e.g. embryonic stem cells are able to differentiate into all cell types in the body. Reference [GMR08] gives an example by investigating the regeneration of the chronic chagasic cardiomyopathy after bone marrow stem cell transplantation, which involves the modeling of the differentiation of bone marrow stem cells into cardiomyocyte. In this scenario, we can use colors to encode different states of components, e.g. cells.

(7) Replication of components.

Replication of a component usually means the process of duplicating or producing a copy of this component, e.g. DNA replication or cell division [Mor06]. For example, cell division and thereby DNA replication is a key step in life, which plays important roles in the development and growth of organisms, regular renewal of cells and repair of damaged cells. The modeling of cell division has been widely studied, e.g. [Wat09]. Using colored Petri nets, we can represent a cell as a color of a color set and add a new color to the color set for each division.

(8) Death of components.

Death of components, e.g. cell death [Mor06] is also essential in life like apoptosis of cells during organism development and necrosis caused by detrimental factors. Using colored Petri nets, we can still represent a cell as a color of a color set, but when it dies, we have to remove this color from the color set.

(9) Pattern formation of components.

The processes of pattern formation [Gri08] mean to distribute and organize a number of components (e.g. cells or signal molecules) in appropriate one, two or three dimensional structures in space and time [MFV10]. Pattern formation in fact results from the combination of organization, communication and movement of components, e.g. cell populations, so it can be tackled by colored Petri nets.

Multiscale modeling poses technical challenges in order to solve the application challenges above, the most important two of which are given as follows.

Modeling methods.

The selected modeling methods have to take into account the challenges addressed by multiscale modeling. As described above, colored Petri nets have the ability to address these challenges. Besides, the modeling method for each level may be different due to the complex nature of biological systems; therefore, the selected modeling methods have to offer at least continuous and stochastic modeling capabilities. Colored Petri nets provide these capabilities through colored stochastic and continuous Petri nets.

Analysis techniques.

In the situation of multiscale modeling, the constructed models become more and more intractable due to the involved multiple scales, which demand for more analysis techniques, not only continuous or stochastic simulation, but also formal analysis techniques, e.g. model checking. Colored Petri nets offer a variety of analysis methods, ranging from animation/simulation to structural analysis to qualitative and quantitative model checking.

In summary, in order to address these challenges, new modeling approaches have to be provided and fortunately colored Petri nets could have the required potential. In Chapter 5, we will illustrate how to use colored Petri nets to address some of these challenges.

1.3 Contributions

This thesis aims to present a technology based on colored Petri nets and associated techniques to support multiscale modeling and analysis of biological systems. In particular, the contributions of this thesis are as follows.

(1) A colored Petri net framework for systems biology.

Modeling in systems biology usually takes two approaches: qualitative and quantitative. Qualitative approaches often serve as initial investigations of systems or apply when the amount of quantitative information about systems is limited. Quantitative approaches offer quantitative and realistic insights into systems to be studied, which usually take two forms: stochastic and deterministic.

The first contribution of this thesis is that we present a colored Petri net framework for systems biology in Chapter 2, which relates three modeling paradigms: *colored qualitative Petri net* (\mathcal{QPN}^C), *colored stochastic Petri net* (\mathcal{SPN}^C) and *colored continuous Petri net* (\mathcal{CPN}^C). Hence, we can model and analyze a biological system from three perspectives: qualitative, stochastic and continuous by transforming them into each other. That is to say, these three formalisms can be combined together to accomplish the modeling and analysis of biological systems.

(2) An algorithm for the computation of enabled transition instances.

Animation/simulation is an important technique for obtaining an intuitive understanding of a Petri net model as it demonstrates the dynamic behavior of the model in a visual way. The computation of enabled transition instances plays a core role in the animation/simulation of colored Petri nets. The main difference between the computation of enabled transition instances for animation/simulation and the computation of transition instances for unfolding is that the former computes enabled transition instances in terms of available tokens on places while the latter computes transition instances in terms of color sets of variables.

In Chapter 3 we present an efficient algorithm for the computation of enabled transition instances for colored Petri nets. This algorithm uses a pattern matching mechanism and a new partial binding - partial test principle, and adopts some optimization techniques. The pattern matching mechanism improves the computational efficiency by binding variables to available tokens on places. The partial binding - partial test principle allows us to test expressions during the partial binding process so as to prune invalid bindings as early as possible. The optimization techniques prune invalid potential bindings before the binding begins, and also find disabled transitions at an early phase. This algorithm realizes an efficient computation of enabled transition instances for colored Petri nets.

(3) An unfolding algorithm for colored Petri nets.

Colored Petri nets provide a compact and convenient way for modeling complex systems, but many basic properties and analysis techniques for standard Petri nets are

difficult to extend to colored Petri nets. So it is a reasonable approach to unfold colored Petri nets to equivalent standard Petri nets in order to use existing analysis techniques and tools for standard Petri nets. Besides, we can also simulate colored stochastic (continuous) Petri nets using stochastic (continuous) simulation algorithms by unfolding. Therefore, unfolding is an essential problem in simulating and analyzing colored Petri nets.

For this, in Chapter 3 we present an efficient unfolding algorithm, in which we provide two approaches to efficiently compute transition instances. That is, if the color set of each variable in a guard is a finite integer domain, the constraint satisfaction approach is used to obtain all valid bindings; otherwise, a general algorithm is adopted, in which some optimization techniques, e.g. the partial binding - partial test principle, are used.

(4) Automatic folding of Petri nets, illustrated by three special cases.

Folding (colorizing) is a challenging approach to obtain a colored Petri net for a given standard Petri net. In Chapter 3, we consider three special cases for automatic folding: colorizing T-invariants, master nets and twin nets in order to reduce the amount of manual work required for folding Petri nets. Among them, colorizing T-invariants contributes to the further understanding of T-invariants of a biological network, and colorizing master nets and twin nets offers a convenient way for reconstructing biological networks from experimental data.

(5) Summarizing analysis techniques for colored Petri nets.

Petri nets offer a large variety of analysis techniques ranging from informal techniques, e.g. animation/simulation to formal techniques, e.g. model checking. In Chapter 4, we summarize those analysis techniques that can be used for colored Petri nets and pay attention to applying them for analyzing colored Petri nets.

(6) Application and validation of the colored Petri net technology and techniques using case studies.

In Chapter 5, we give three case studies, *C. elegans* vulval development, coupled Ca^{2+} channels and membrane systems, to explore the application of our colored Petri net technology and techniques. These case studies not only demonstrate how to apply the colored Petri net framework and related analysis techniques to modeling and analyzing practical biological systems, but also show how to address the challenges in systems biology given above.

Concrete outputs.

We have implemented our colored Petri net technology and associated techniques in our modeling tool, Snoopy; as a result, we provide three colored net classes, QPN^c , SPN^c and CPN^c , available in Snoopy. We have also finished a detailed manual for how to model and analyze colored Petri nets in Snoopy [LH11] and provided a library of examples, see e.g. [GLTG11], [GLG⁺11] and [GH11].

We have also published a set of related papers, see [LH10a], [LH10b], [GLTG11] and [GLG⁺11].

1.4 Organization of Thesis

This thesis is organized as follows:

Chapter 2 We propose a colored Petri net framework for modeling and analyzing biological systems, which contains three related modeling paradigms: QPN^c , SPN^c and CPN^c . Besides, we summarize the scenarios where colored Petri nets can be used to model biological systems and discuss how to encode components of biological systems as colors.

Chapter 3 We focus on three key problems in implementing colored Petri nets. We first give an algorithm for the computation of enabled transition instances. We then present an algorithm for unfolding colored Petri nets and discuss how to improve the efficiency of the unfolding process. Besides we discuss three scenarios of the automatic folding (colorizing) of Petri nets: T-invariants, master nets and twin nets.

Chapter 4 We summarize analysis techniques for colored Petri nets from two points of view. From the unfolding point of view, we describe how to employ existing techniques, structural techniques, numerical and simulative model checking, and tools for standard Petri nets to realize the analysis of colored Petri nets. From the folding point of view, we transform our qualitative Petri nets to those read by CPN tools and then analyze them using CPN tools.

Chapter 5 We give three case studies, *C. elegans* vulval development, coupled Ca^{2+} channels and membrane systems to demonstrate the applications of colored Petri nets.

Chapter 6 We summarize the achieved results and give an outlook for future research.

2 A Colored Petri Net Framework

Petri nets provide a formal and clear representation of biological systems based on their firm mathematical foundation for the analysis of biochemical properties. However, standard Petri nets do not easily scale. So attempts to simulate biological systems by standard Petri nets have been mainly restricted so far to relatively small models. They tend to grow quickly for modeling complex systems, which makes it more difficult to manage and understand the nets, thus increasing the risk of modeling errors. Two known modeling concepts improving the situation are hierarchy and color. Hierarchical structuring has been discussed a lot, see e.g. [MWW11], while the color has gained little attention so far. Thus, we investigate how to apply colored Petri nets to modeling and analyzing biological systems. To do so, we not only provide compact and readable representations of complex biological systems, but also do not lose the analysis capabilities of standard Petri nets, which can still be supported by automatic unfolding. Moreover, another attractive advantage of colored Petri nets for a biological modeler is that they provide the possibility to easily increase the size of a model consisting of many similar subnets just by adding colors.

In this chapter, we present a colored Petri net framework for modeling and analyzing biological systems, which contains three related modeling paradigms: *colored qualitative Petri net* (QPN^C), *colored stochastic Petri net* (SPN^C) and *colored continuous Petri net* (CPN^C). These three formalisms can be combined together to accomplish the modeling and analysis of biological systems.

This chapter is organized as follows. We first give an overview of this framework. After that, we briefly recall some prerequisites of colored Petri nets and then formally define these three formalisms. We finally summarize the scenarios where colored Petri nets can be used in systems biology and discuss how to encode components of biological systems as colors.

2.1 Overview

In this section, we present a colored Petri net framework for modeling and analyzing biological systems (illustrated in Figure 2.1), which extends the Petri net framework for modeling and analyzing biological systems in [GHL07], i.e. the new proposed framework is in fact the colored version of the existing framework, but the colored version provides possibilities for compact and readable representations of complex biological systems.

Both of these frameworks unify the qualitative, stochastic and continuous Petri net paradigms. More specifically, the new framework relates three modeling paradigms: QPN^c , SPN^c and CPN^c , just like the Petri net framework relating *qualitative Petri net* (QPN), *stochastic Petri net* (SPN) and *continuous Petri net* (CPN).

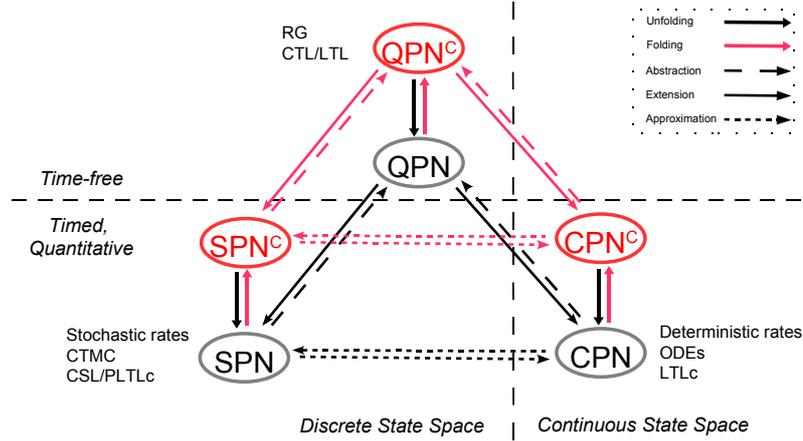


Figure 2.1: A colored Petri net framework.

QPN^c is a colored version of QPN . Like QPN , QPN^c is time-free, i.e. transitions are not assigned a time and tokens on places are not associated with a sojourn time. So the analysis on a QPN^c model in fact considers all possible behavior of it under any timing, for example, using the state space analysis based on Computational Tree Logic (CTL), one branching time temporal logic [CGP01], to analyze its reachability graph (RG).

SPN^c is a colored version of SPN . Similarly, a firing delay rate is introduced and associated with each transition, which is a random variable defined by an exponential probability distribution. Therefore, the semantics of SPN^c is equivalent to a continuous time Markov chain (CTMC), which is constructed from the reachability graph of the underlying qualitative Petri net by labeling the arcs between the states with transition rates. So we can use such quantitative analysis techniques as Continuous Stochastic Logic (CSL) [Kwi03], a probabilistic counterpart of CTL, or Probabilistic Linear-time Temporal Logic with numerical constraints (PLTLc) [DG08]. QPN^c is an abstraction of SPN^c , so all qualitative properties valid in the QPN^c are also valid in the SPN^c , and vice versa.

CPN^c is a colored version of CPN . In CPN^c , the discrete values of places are replaced with continuous values, which describe the overall behavior of species represented by places via concentrations. A deterministic rate is associated with each transition, which

makes a continuous Petri net model represent a set of ordinary differential equations (ODEs). Contrary to discrete Petri nets, the state space for \mathcal{CPN}^c is continuous and linear, so we can analyze it using a Linear Temporal Logic (LTL) [Pnu81], e.g. Linear Temporal Logic with constraints (LTLc) in the manner of [CCFS06] or PLTLc [DG08]. The stochastic and continuous models are mutually related by approximation. See [GHL07] for more details about the relationships among qualitative, stochastic and continuous Petri nets.

2.2 Colored Petri Nets

In this section, we recall some prerequisites of colored Petri nets according to [Jen81] and [JK09] before introducing our own colored Petri net classes.

Let us begin with introducing the definition and notations of multisets, which will be used in the later definitions of colored Petri nets.

2.2.1 Multiset

A *multiset* is a set in which there can be several occurrences for the same element. The number of occurrences of an element is called *coefficient* or *multiplicity*.

Definition 1 (Multiset)

Let S be a finite, non-empty set, a multiset over S is a function $m : S \rightarrow \mathbb{N}_0$ that maps each element $s \in S$ onto a non-negative integer $m(s) \in \mathbb{N}_0$. It is denoted by a formal sum: $m = \sum_{s \in S} m(s) \cdot s$.

The collection of all the multisets over S is denoted by S_{MS} . The empty multiset over S is denoted by ϕ_{MS} where the coefficient for each element is zero.

Definition 2 (Multiset operations)

Let S be a finite, non-empty set, and $\forall m_1, m_2, m \in S_{MS}$. Addition (+), scalar multiplication (*), comparison (\leq), subtraction (-) and size $|m|$ are defined as follows:

1. $(m_1 + m_2)(s) = m_1(s) + m_2(s), \forall s \in S$.
2. $(n * m)(s) = n * m(s) \forall n \in \mathbb{N}_0, \forall s \in S$.
3. $m_1 \leq m_2 \Leftrightarrow m_1(s) \leq m_2(s), \forall s \in S$.
4. $(m_2 - m_1)(s) = m_2(s) - m_1(s), \forall s \in S$ and $m_1 \leq m_2$.
5. $|m| = \sum_{s \in S} m(s)$.

We also use S_{MS} to represent the multiset type over S . For example, the marking of a place p and the expressions on the arcs connected by p are of the multiset type $C(p)_{MS}$, where $C(p)$ represents the color set of p , that is, they are required to evaluate to the multiset over the color set of their corresponding place, p .

For instance, let a color set be $S = \{a, b, c\}$, then $m = 1'a+2'b+4'c$ is a multiset over S , which contains 1 occurrence of element a , 2 occurrences of element b and 4 occurrences of element c , i.e. $m(a) = 1$, $m(b) = 2$ and $m(c) = 4$.

2.2.2 Definition

In colored Petri nets, there are different types of expressions, e.g. arc expressions, guards, or expressions for defining initial markings. An expression is built up from variables, constants, and operation symbols. It is not only associated with a particular color set, but also written in terms of a predefined syntax. In the following, we denote by EXP a set of expressions that comply with a predefined syntax. The formal definition of colored Petri nets is as follows.

Definition 3 (Colored Petri net)

A colored Petri net is a tuple $N = \langle P, T, F, \sum, C, g, f, m_0 \rangle$, where:

- P is a finite, non-empty set of places.
- T is a finite, non-empty set of transitions.
- F is a finite set of directed arcs.
- \sum is a finite, non-empty set of color sets.
- $C : P \rightarrow \sum$ is a color function that assigns to each place $p \in P$ a color set $C(p) \in \sum$.
- $g : T \rightarrow EXP$ is a guard function that assigns to each transition $t \in T$ a guard expression of the Boolean type.
- $f : F \rightarrow EXP$ is an arc function that assigns to each arc $a \in F$ an arc expression of a multiset type $C(p)_{MS}$, where p is the place connected to the arc a .
- $m_0 : P \rightarrow EXP$ is an initialization function that assigns to each place $p \in P$ an initialization expression of a multiset type $C(p)_{MS}$.

We then introduce the following notions that may be used in the following sections.

- $\bullet t$, the preplaces of a transition t ,
- $t\bullet$, the postplaces of a transition t ,

- $\bullet p$, the pretransitions of a place p ,
- p^\bullet , the posttransitions of a place p .

2.2.3 Dynamic Behavior

Before giving the behavior of colored Petri nets, we first define some concepts and denotations.

In a colored Petri net $N = \langle P, T, F, \sum, C, g, f, m_0 \rangle$, each place $p \in P$ is associated with a color set $C(p)$. Each color $c \in C(p)$ exactly corresponds to a place instance, i.e. each color will become an uncolored place after unfolding. We let $p(c)$ denote an instance of p with color c , $I_P(p)$ all the instances of p and I_P all the instances of all places $p \in P$.

Definition 4 (Place instance)

A place instance $p(c)$ is a pair (p, c) with $p \in P$ and $c \in C(p)$.

The variables associated with a transition $t \in T$ are denoted $Var(t)$, which is composed of the variables in the guard of t and in the expressions of arcs connected to t . Before the expressions are evaluated to values, the variables must get assigned values, which is called binding [JKW07].

A binding b of a transition t is a function that maps each variable $v \in Var(t)$ onto a value $b(v)$ that is of the same type as the variable. The set of all bindings for a transition t is denoted $B(t)$.

A binding $b \in B(t)$ of a transition t exactly corresponds to a transition instance, denoted by $t(b)$, i.e. it will become an uncolored transition after unfolding. The set of all bindings for a transition t constitutes the set of all the instances of transition t , denoted by $I_T(t)$. The set of all instances of all transitions $t \in T$ is denoted I_T .

Definition 5 (Transition instance)

A transition instance $t(b)$ is a pair (t, b) with $t \in T$ and $b \in B(t)$.

In the following, we use $exp\langle b \rangle$, where $exp \in EXP$, to represent the result of evaluating an expression exp in terms of a binding b of a transition t .

Definition 6 (Transition instance enabling)

A transition instance $t(b) \in I_T$ is enabled in a marking m , denoted by $m[t(b)]$, if and only if the following conditions are satisfied:

1. $g(t)\langle b \rangle = true$,
2. $m(p) \geq f(p, t)\langle b \rangle, \forall p \in \bullet t$.

Definition 7 (Transition instance firing)

A transition instance $t(b) \in I_T$ enabled in a marking m may fire, and reach a new marking m' , denoted by $m[t(b)]m'$, with

$$m'(p) = m(p) + f(t, p)\langle b \rangle - f(p, t)\langle b \rangle, \forall p \in P.$$

2.3 Colored Qualitative Petri Nets

The introduction of colored Petri nets greatly increases modeling convenience by associating data types with tokens and parameterizing transitions and arcs, which makes it possible to create compact models for complex systems.

QPN^C has a similar expressive power to colored Petri nets given by K. Jensen [Jen81], but adds some restrictions on data types and the syntax of expressions. However, QPN^C considers many features that are very helpful for the modeling of biological systems, such as flexible ways for specifying initial marking and support of special arcs. In fact, QPN^C is a colored extension of Petri nets extended by such special arcs as inhibitor, read, equal and reset arcs.

In the following, we first describe Petri nets extended by special arcs, and then give the definition of QPN^C and describe its behavior.

2.3.1 Extended Petri Nets

There have been many various extensions based on the basic Place/Transition nets (P/T nets) [HGD08] so far. One of these extensions is to consider special arcs, which either make the model representation more compact or extend the modeling power of the Petri net formalism.

In our extended Petri nets, we consider four special arcs: inhibitor arcs, read arcs (also called test arcs), equal arcs and reset arcs, which are particularly helpful for biological modeling. Figure 2.2 gives the graphical representation of these three special arcs in Snoopy.

The inhibitor arc, read arc and equal arc add constraints on the firing of a transition, but are not affected by the firing. The inhibitor arc was first introduced in [FA73] to solve a synchronization problem beyond the power of P/T nets. The inhibitor arc reverses the logic of the enabling condition of a place, i.e. it imposes a precondition that a transition may only fire if the place contains less tokens than the weight the arc indicates. The inhibitor arc is only allowed to lead from a place to a transition that is inhibited. In systems biology, the inhibitor arc is especially useful for modeling the inhibition function widely existing in biological systems [GCPL⁺98].

The read/test arc was introduced in [MR95], which is also directed from a place to a transition. It allows to model that some resource is read, but does not consume the

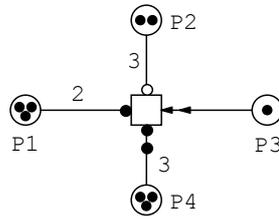


Figure 2.2: Special arcs drawn in Snoopy, where the arc terminating with a small hollow circle is the inhibitor arc, the arc with a filled circle is the read/test arc, the arc with double filled circles is the equal arc, and the arc with double arrowheads is the reset arc.

tokens of the source place. Hence the same token can be used by many transitions at the same time. In systems biology, the read arc provides a convenient representation of such chemical interactions as enzymatic reactions, since the enzyme itself is not consumed in the enzyme reaction [Cha07].

The equal arc [RMH10] imposes a precondition that a transition may only fire if the number of tokens of the place connected by the equal arc is equal to the arc weight. After fired, the tokens of this place do not change. An equal arc can be replaced by an inhibitor arc and a read arc, so it makes the model representation more compact.

The reset arc makes it possible to empty the place connected by this arc once the transition connected by it is fired. This adds expressive power and makes reachability undecidable [AK76]. The reset arc is also only allowed to lead from a place to a transition.

The following gives the formal definition of extended Petri nets by special arcs.

Definition 8 (Extended Petri net)

An *extended Petri net* is a tuple $N = \langle P, T, F, f, m_0 \rangle$ where:

- P is a finite, non-empty set of places.
- T is a finite, non-empty set of transitions.
- F is a finite set of directed arcs. F is refined as the union of five disjunctive arc sets, i.e. $F := F_S \cup F_I \cup F_T \cup F_E \cup F_R$ with:
 - $F_S \subseteq (P \times T) \cup (T \times P)$ is the set of standard arcs,
 - $F_I \subseteq P \times T$ is the set of inhibitor arcs,
 - $F_T \subseteq P \times T$ is the set of test/read arcs,
 - $F_E \subseteq P \times T$ is the set of equal arcs, and

- $F_R \subseteq P \times T$ is the set of reset arcs.
- $f : F \rightarrow \mathbb{N}_0$ is a function that assigns a non-negative integer to each arc $a \in F$.
- $m_0 : P \rightarrow \mathbb{N}_0$ gives the initial marking.

Compared with standard P/T nets, the firing rule for extended Petri nets needs to be adapted accordingly. The enabling conditions are extended in the following way.

Definition 9 (Extended enabling condition)

Let $N = \langle P, T, F, f, m_0 \rangle$ be an extended Petri net and m be a marking of N . $F := F_S \cup F_I \cup F_T \cup F_E \cup F_R$. A transition $t \in T$ is enabled in the marking m , denoted by $m[t]$, if the following conditions are satisfied:

- $\forall p \in {}^\bullet t, m(p) \geq f(p, t)$, if $(p, t) \in F_S$,
- $\forall p \in {}^\bullet t, m(p) < f(p, t)$, if $(p, t) \in F_I$,
- $\forall p \in {}^\bullet t, m(p) \geq f(p, t)$, if $(p, t) \in F_T$,
- $\forall p \in {}^\bullet t, m(p) = f(p, t)$, if $(p, t) \in F_E$.

Definition 10 (Extended firing)

Let $N = \langle P, T, F, f, m_0 \rangle$ be an extended Petri net, m be a marking of N , and a transition $t \in T$ be enabled in the marking m . $F := F_S \cup F_I \cup F_T \cup F_E \cup F_R$. The transition t can be fired and reach a new marking m' , denoted by $m[t]m'$, with

$$m'(p) = \begin{cases} m(p) - f(p, t) + f(t, p) & \text{if } (p, t) \in F_S, \\ m(p) + f(t, p) & \text{if } (p, t) \in F_I, \\ m(p) + f(t, p) & \text{if } (p, t) \in F_T, \\ m(p) + f(t, p) & \text{if } (p, t) \in F_E, \\ f(t, p) & \text{if } (p, t) \in F_R. \end{cases}$$

Consider an extended Petri net in Figure 2.3, which illustrates how different extended arcs work. For example, the transition $t0_1$ is connected with $p0_1$ by an inhibitor arc. $t0_1$ is enabled when $p0_1$ has less than 2 tokens. Firing of $t0_1$ does not change the number of tokens on $p0_1$. $t3_1$, connected with $p0_1$ by an equal arc, is enabled when $p0_1$ has exactly 2 tokens. If $t3_1$ is fired, the equal arc also does not change the number of tokens on $p0_1$. $t1_1$ is connected with $p0_1$ by a read arc and with $p1_1$ by a reset arc, whose firing will make $p1_1$ empty due to the reset arc.

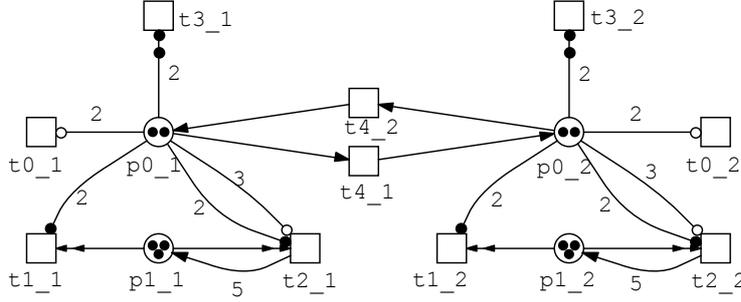


Figure 2.3: An extended Petri net that illustrates how different extended arcs work.

2.3.2 Colored Qualitative Petri Nets

Colored qualitative Petri nets are a colored extension of extended Petri nets defined in the previous section. In the following, we give the formal definition of QPN^c .

Definition 11 (Colored qualitative Petri net)

A colored qualitative Petri net QPN^c is an eight-tuple $N = \langle P, T, F, \Sigma, C, g, f, m_0 \rangle$, where:

- P is a finite, non-empty set of places.
- T is a finite, non-empty set of transitions.
- F is a finite set of directed arcs. F is refined as the union of five disjunctive arc sets, i.e. $F := F_S \cup F_I \cup F_T \cup F_E \cup F_R$ with:
 - $F_S \subseteq (P \times T) \cup (T \times P)$ is the set of standard arcs,
 - $F_I \subseteq P \times T$ is the set of inhibitor arcs,
 - $F_T \subseteq P \times T$ is the set of test/read arcs,
 - $F_E \subseteq P \times T$ is the set of equal arcs, and
 - $F_R \subseteq P \times T$ is the set of reset arcs.
- Σ is a finite, non-empty set of color sets.
- $C : P \rightarrow \Sigma$ is a color function that assigns to each place $p \in P$ a color set $C(p) \in \Sigma$.
- $g : T \rightarrow EXP$ is a guard function that assigns to each transition $t \in T$ a guard expression of the Boolean type.

- $f : F \rightarrow EXP$ is an arc function that assigns to each arc $a \in F$ an arc expression of a multiset type $C(p)_{MS}$, where p is the place connected to the arc a .
- $m_0 : P \rightarrow EXP$ is an initialization function that assigns to each place $p \in P$ an initialization expression of a multiset type $C(p)_{MS}$.

The meaning of each element in this definition is the same as that of colored Petri nets by K. Jensen [Jen81]. The only difference lies in the consideration of special arcs, which results in the change of enabling conditions.

Definition 12 (Transition instance enabling)

Let $N = \langle P, T, F, \sum, C, g, f, m_0 \rangle$ be a QPN^c and m be a marking of N . $F := F_S \cup F_I \cup F_T \cup F_E \cup F_R$. A transition instance $t(b) \in I_T$ is enabled in a marking m , denoted by $m[t(b)]$, if and only if the following conditions are satisfied:

1. $g(t) \langle b \rangle = true$,
2. $\forall p \in \bullet t, m(p) \geq f(p, t) \langle b \rangle$, if $(p, t) \in F_S$,
3. $\forall p \in \bullet t, m(p) < f(p, t) \langle b \rangle$, if $(p, t) \in F_I$,
4. $\forall p \in \bullet t, m(p) \geq f(p, t) \langle b \rangle$, if $(p, t) \in F_T$,
5. $\forall p \in \bullet t, m(p) = f(p, t) \langle b \rangle$, if $(p, t) \in F_E$.

Definition 13 (Transition instance firing)

Let $N = \langle P, T, F, \sum, C, g, f, m_0 \rangle$ be a QPN^c , m be a marking of N , and a transition instance $t(b) \in I_T$ be enabled in the marking m . $F := F_S \cup F_I \cup F_T \cup F_E \cup F_R$. The transition instance $t(b)$ can be fired, and reach a new marking m' , denoted by $m[t(b)]m'$, with

$$m'(p) = \begin{cases} m(p) - f(p, t) \langle b \rangle + f(t, p) \langle b \rangle & \text{if } (p, t) \in F_S, \\ m(p) + f(t, p) \langle b \rangle & \text{if } (p, t) \in F_I, \\ m(p) + f(t, p) \langle b \rangle & \text{if } (p, t) \in F_T, \\ m(p) + f(t, p) \langle b \rangle & \text{if } (p, t) \in F_E, \\ f(t, p) \langle b \rangle & \text{if } (p, t) \in F_R. \end{cases}$$

For example, if we fold the left part and the right part of the model in Figure 2.3 by defining a color set CS with two colors, e.g. 1 and 2, we can obtain a colored version of the extended Petri net model, illustrated in Figure 2.4.

We then use a circadian model introduced in [GHG02] to give another QPN^c example. It describes circadian rhythms, which are widely used in organisms to keep a sense of daily time and regulate their behavior accordingly. In this model, two genes, e.g. a and r , are transcribed into mRNA and then translated into proteins respectively.

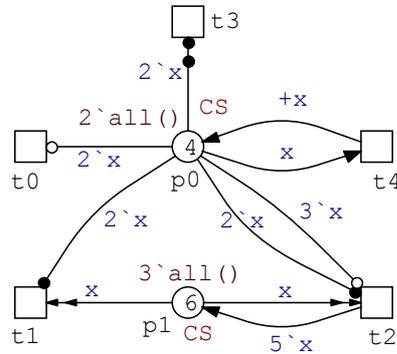


Figure 2.4: A QPN^C model for Figure 2.3. The declarations: *colorset CS = int with 1,2 and variable $x : CS$.*

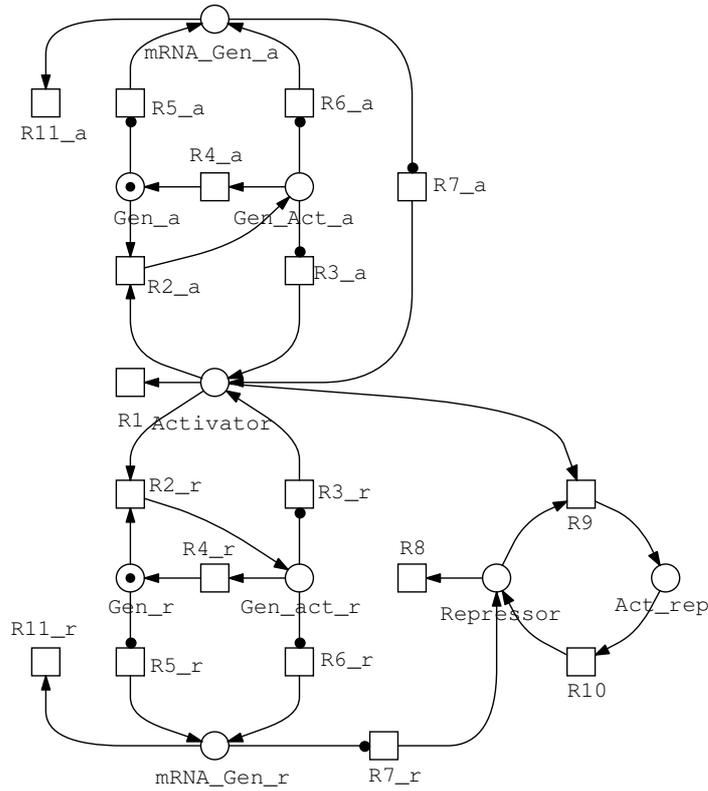


Figure 2.5: A Petri net model for circadian rhythms.

Gene a positively regulates the transcription, whereas gene r negatively regulates the transcription. More details can be found in [GHG02] and [VKBL02].

According to the kinetic equations provided in [GHG02], we build a qualitative Petri net model and further a QPN^C model by folding two genes, illustrated in Figure 2.5 and Figure 2.6, respectively. From the colored model, we can see that we define a color set CS with two colors a and r to distinguish two genes. Besides, we use read arcs to accomplish the control of $R3$, $R5$ and $R6$ for example, as tokens on places Gen and Gen_act are not consumed.

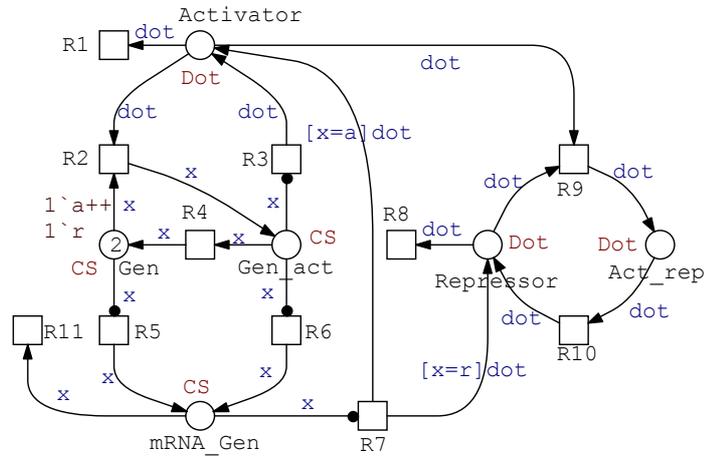


Figure 2.6: A QPN^C model for circadian rhythms in Figure 2.5. The declarations: *colorset* $Dot = with\ dot$, *colorset* $CS = enum\ with\ a, r$ and *variable* $x : CS$.

2.4 Colored Stochastic Petri Nets

Stochastic Petri nets are an extension of qualitative Petri nets by associating random delay firing rates with transitions, which provide a possibility to model stochastic systems [HGD08]. Due to inherently stochastic in biological processes, stochastic Petri nets have recently become a popular modeling paradigm for capturing their complex stochastic dynamics, which offer a suitable way to understand the behavior of complex biological systems by integrating detailed biochemical data and providing quantitative analysis results, see e.g. [GP98], [PRA05].

There are several extensions based on the fundamental stochastic Petri net class SPN . For example, the generalized stochastic Petri net ($GSPN$) is SPN extended by in-

hibitor arcs and immediate transitions. The deterministic and stochastic Petri net (\mathcal{DSPN}) is \mathcal{GSPN} extended by deterministic transitions. Different extensions serve different requirements of biological modeling, e.g \mathcal{DSPN} can be used to model biological systems with a stochastic and deterministic firing behavior. See [HLGM09] for more details.

While \mathcal{SPN} and its extensions offer enormous modeling power, managing large-scale low-level Petri net models is difficult due to the fact that tokens are indistinguishable. To alleviate this limitation, \mathcal{SPN}^c is presented to uplift biochemically interpreted extended stochastic Petri nets introduced in [HLGM09] to a colored version. As in \mathcal{QPN}^c , in \mathcal{SPN}^c , tokens are distinguished by the "color", and arc expressions and guards have the same meaning.

In the following, we start with recalling the biochemically interpreted \mathcal{SPN} and \mathcal{DSPN} introduced in [HLGM09] and then focus on \mathcal{SPN}^c , the colored version of \mathcal{DSPN} .

2.4.1 Stochastic Petri Nets

Stochastic Petri nets are an extension of qualitative Petri nets [HGD08]. Contrary to the qualitative Petri net, a firing delay rate is introduced and associated with each transition t of a Petri net, which is a random variable X_t , defined by the following exponential probability distribution:

$$F_{X_t}(\tau) = 1 - e^{-\lambda_t \bullet \tau}, \tau \geq 0.$$

The formal definition of stochastic Petri nets is as follows [HLGM09].

Definition 14 (Stochastic Petri net)

A *biochemically interpreted stochastic Petri net* \mathcal{SPN} is a six-tuple $N = \langle P, T, F, f, v, m_0 \rangle$, where:

- P is a finite, non-empty set of places.
- T is a finite, non-empty set of transitions.
- $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of directed arcs.
- $f : F \rightarrow \mathbb{N}_0$ is a function that assigns a non-negative integer to each arc $a \in F$.
- $v : T \rightarrow H$ is a function that assigns a stochastic hazard function $h(t)$ to each transition $t \in T$, whereby $H := \bigcup_{t \in T} \{h_t | h_t : \mathbb{N}_0^{|\bullet t|} \rightarrow \mathbb{R}^+\}$ is the set of all stochastic hazard functions, and $v(t) = h(t)$ for all transitions $t \in T$. \mathbb{R}^+ denotes the set of all non-negative real numbers.
- $m_0 : P \rightarrow \mathbb{N}_0$ gives the initial marking.

The stochastic hazard function h_t defines the marking-dependent transition rate $\lambda_t(m)$ for the transition $t \in T$, i.e. $h_t = \lambda_t(m)$. The domain of h_t is restricted to the set of preplaces of t to enforce a close relation between network structure and hazard functions.

The semantics of a stochastic Petri net is equivalent to a continuous time Markov chain (CTMC), which is constructed from the reachability graph of the underlying qualitative Petri net by labeling the arcs between the states with transition rates. For more details, see [HLGM09].

2.4.2 Deterministic and Stochastic Petri Nets

Generalized stochastic Petri nets are stochastic Petri nets extended by inhibitor arcs and immediate transitions. Deterministic and stochastic Petri nets are generalized stochastic Petri nets extended by deterministic transitions. In this section, we skip \mathcal{GSPN} and recall \mathcal{DSPN} in [HLGM09] based on which we will define our colored stochastic Petri nets.

Definition 15 (Deterministic and stochastic Petri net)

A biochemically interpreted deterministic and stochastic Petri net \mathcal{DSPN} is a seven-tuple $N = \langle P, T, F, f, v, l, m_0 \rangle$, where:

- P is a finite, non-empty set of places.
- T is a finite, non-empty set of transitions. T is the union of three disjunctive transition sets, i.e. $T := T_{stoch} \cup T_{im} \cup T_{timed}$ with:
 - T_{stoch} , the set of stochastic transitions with exponentially distributed waiting time,
 - T_{im} , the set of immediate transitions with waiting time zero, and
 - T_{timed} , the set of transitions with deterministic waiting time.
- F is a finite set of directed arcs. F is the union of two disjunctive arc sets, i.e., $F := F_S \cup F_I$ with:
 - $F_S \subseteq (P \times T) \cup (T \times P)$ is the set of directed standard arcs, and
 - $F_I \subseteq P \times T$ is the set of directed inhibitor arcs.
- $f : F \rightarrow \mathbb{N}_0$ is a function that assigns a non-negative integer to each arc $a \in F$.
- $v : T_{stoch} \rightarrow H$ is a function that assigns a stochastic hazard function $h(t)$ to each transition $t \in T_{stoch}$, whereby $H := \bigcup_{t \in T_{stoch}} \{h_t | h_t : \mathbb{N}_0^{|\bullet t|} \rightarrow \mathbb{R}^+\}$ is the set of all stochastic hazard functions, and $v(t) = h(t)$ for all transitions $t \in T_{stoch}$.

- $l : T_{timed} \rightarrow \mathbb{R}^+$ assigns a non-negative deterministic waiting time to each deterministic transition $t \in T_{timed}$.
- $m_0 : P \rightarrow \mathbb{N}_0$ gives the initial marking.

The stochastic transitions also have an exponentially distributed waiting time. For sake of simplicity, such features as read arcs and scheduled transitions are not explicitly mentioned in this definition. See [HLGM09] for more details.

2.4.3 Colored Stochastic Petri Nets

The biochemically interpreted colored stochastic Petri net (\mathcal{SPN}^c) is a colored version of the biochemically interpreted deterministic and stochastic Petri net \mathcal{DSPN} . In the following, based on \mathcal{DSPN} and colored Petri nets, we give the formal definition of \mathcal{SPN}^c .

Definition 16 (Colored stochastic Petri net)

A biochemically interpreted colored stochastic Petri net \mathcal{SPN}^c is a ten-tuple $N = \langle P, T, F, \sum, C, g, f, v, l, m_0 \rangle$, where:

- P is a finite, non-empty set of places.
- T is a finite, non-empty set of transitions. T is the union of three disjunctive transition sets, i.e. $T := T_{stoch} \cup T_{im} \cup T_{timed}$ with:
 - T_{stoch} , the set of stochastic transitions with exponentially distributed waiting time,
 - T_{im} , the set of immediate transitions with waiting time zero, and
 - T_{timed} , the set of transitions with deterministic waiting time.
- F is a finite set of directed arcs. F is the union of two disjunctive arc sets, i.e., $F := F_S \cup F_I$ with:
 - $F_S \subseteq (P \times T) \cup (T \times P)$ is the set of directed standard arcs, and
 - $F_I \subseteq P \times T$ is the set of directed inhibitor arcs.
- \sum is a finite, non-empty set of color sets.
- $C : P \rightarrow \sum$ is a color function that assigns to each place $p \in P$ a color set $C(p) \in \sum$.
- $g : T \rightarrow EXP$ is a guard function that assigns to each transition $t \in T$ a guard expression of the Boolean type.

- $f : F \rightarrow EXP$ is an arc function that assigns to each arc $a \in F$ an arc expression of a multiset type $C(p)_{MS}$, where p is the place connected to the arc a .
- $v : I_{T_{stoch}} \rightarrow H$ is a function that assigns a stochastic hazard function $h(t(b))$ to each transition instance $t(b) \in I_{T_{stoch}}(t)$ of each transition $t \in T_{stoch}$, whereby $H := \bigcup_{t(b) \in I_{T_{stoch}}} \{h_{t(b)} | h_{t(b)} : \mathbb{N}_0^{|\bullet t(b)|} \rightarrow \mathbb{R}^+\}$ is the set of all stochastic hazard functions, and $v(t(b)) = h(t(b))$ for all transitions $t \in T_{stoch}$.
- $l : I_{T_{timed}} \rightarrow \mathbb{R}^+$ assigns a non-negative deterministic waiting time to each transition instance $t(b) \in I_{T_{timed}}(t)$ of each deterministic transition $t \in T_{timed}$.
- $m_0 : P \rightarrow EXP$ is an initialization function that assigns to each place $p \in P$ an initialization expression of a multiset type $C(p)_{MS}$.

Please note, the stochastic hazard function in \mathcal{SPN}^C is defined for each instance of each colored transition. The domain of $h(t(b))$ is also restricted to the set of preplace instances of $t(b)$, denoted by $\bullet t(b)$ with $\bullet t(b) := \{p(c) \in I_P | f(p(c), t(b)) \neq 0\}$. For sake of simplicity, such features as read arcs and scheduled transitions are also not explicitly mentioned in the definition above.

The semantics of \mathcal{SPN}^C is equivalent to that of its unfolded \mathcal{DSPN} . For the semantics of \mathcal{DSPN} refer to [HLGM09].

We still use circadian rhythms to give a \mathcal{SPN}^C model. The \mathcal{SPN}^C model for circadian rhythms is the same in structure as the \mathcal{QPN}^C model in Figure 2.6. The only difference is that we now assign a rate function to each transition, illustrated in Table 2.1.

Table 2.1: Rate functions of the \mathcal{SPN}^C model for Circadian rhythms. *MassAction* denotes the mass action function [Lun65].

| Transition | Rate function | Transition | Rate function |
|------------|--------------------------|------------|-------------------------|
| R1 | <i>MassAction</i> (1) | R7 | <i>MassAction</i> (5) |
| R2 | <i>MassAction</i> (1) | R8 | <i>MassAction</i> (0.2) |
| R3 | <i>MassAction</i> (100) | R9 | <i>MassAction</i> (1) |
| R4 | <i>MassAction</i> (100) | R10 | <i>MassAction</i> (5) |
| R5 | <i>MassAction</i> (0.01) | R11 | <i>MassAction</i> (0.5) |
| R6 | <i>MassAction</i> (0.01) | | |

2.5 Colored Continuous Petri Nets

In a continuous Petri net, the discrete token values of places are replaced with continuous values, which describe the overall behavior of species represented by places via

concentrations. A deterministic rate is associated with each transition, which makes a continuous Petri net model represent a set of ordinary differential equations (ODEs). Contrary to discrete Petri nets, the state space for a continuous Petri net is continuous and linear [GHL07].

Like other types of uncolored Petri nets, continuous Petri nets also face the largeness problem in representing complex systems due to the fact that tokens are indistinguishable. To alleviate this limitation, the \mathcal{CPN}^c is presented to uplift continuous Petri nets introduced in [HLGM09] to a colored version. As in other types of colored Petri nets, in \mathcal{CPN}^c , tokens are distinguished by the "color"; arc expressions and guards have the same meaning.

In the following, we start with recalling \mathcal{CPN}^c introduced in [HLGM09] and then focus on \mathcal{CPN}^c , the colored version of \mathcal{CPN} .

2.5.1 Continuous Petri Nets

Definition 17 (Continuous Petri net)

A continuous Petri net \mathcal{CPN} is a six-tuple $N = \langle P, T, F, f, v, m_0 \rangle$, where:

- P is a finite, non-empty set of continuous places.
- T is a finite, non-empty set of continuous transitions.
- $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of directed arcs.
- $f : F \rightarrow \mathbb{R}^+$ is a function that assigns a non-negative real values to each arc $a \in F$.
- $v : T \rightarrow H$ is a function that assigns a firing rate function h_t to each transition $t \in T$, whereby $H := \bigcup_{t \in T} \{h_t | h_t : \mathbb{R}^{+|\bullet t|} \rightarrow \mathbb{R}\}$ is the set of all firing rate functions, and $v(t) = h_t$ for all transitions $t \in T$.
- $m_0 : P \rightarrow \mathbb{R}^+$ gives the initial marking.

The firing rate function h_t defines the marking-dependent continuous transition rate for the transition t . The domain of h_t is restricted to the set of preplaces of t to enforce a close relation between network structure and firing rate functions.

The underlying semantics of a continuous Petri net is a system of ODEs, where each equation describes the continuous token flow of a given place, i.e. continuously increasing its pretransitions' flow and decreasing its posttransitions' flow. An equation for a place p has the following form:

$$\frac{dm(p)}{dt} = \sum_{t \in \bullet p} f(t, p)v(t) - \sum_{t \in p \bullet} f(p, t)v(t).$$

See [HLGM09] for more details.

2.5.2 Colored Continuous Petri Nets

\mathcal{CPN}^c is a colored version of \mathcal{CPN} . In the following, based on \mathcal{CPN} and colored Petri nets, we give the formal definition of \mathcal{CPN}^c .

Definition 18 (Colored continuous Petri net)

A colored continuous Petri net \mathcal{CPN}^c is a nine-tuple $N = \langle P, T, F, \Sigma, C, g, f, v, m_0 \rangle$, where:

- P is a finite, non-empty set of continuous places.
- T is a finite, non-empty set of continuous transitions.
- $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of directed arcs.
- Σ is a finite, non-empty set of color sets.
- $C : P \rightarrow \Sigma$ is a color function that assigns to each place $p \in P$ a color set $C(p) \in \Sigma$.
- $g : T \rightarrow EXP$ is a guard function that assigns to each transition $t \in T$ a guard expression of the Boolean type.
- $f : F \rightarrow EXP$ is an arc function that assigns to each arc $a \in F$ an arc expression of a multiset type $C(p)_{MS}$, where p is the place connected to the arc a .
- $v : I_T \rightarrow H$ is a function that assigns a firing rate function $h_{t(b)}$ to each transition instance $t(b) \in I_T(t)$ of each transition $t \in T$, whereby $H := \bigcup_{t(b) \in I_T} \{h_{t(b)} \mid h_{t(b)} : \mathbb{R}^{+|\bullet t(b)|} \rightarrow \mathbb{R}\}$ is the set of all firing rate functions, and $v(t(b)) = h_{t(b)}$ for all transitions $t \in T$.
- $m_0 : P \rightarrow EXP$ is an initialization function that assigns to each place $p \in P$ an initialization expression of a multiset type $C(p)_{MS}$.

Please note, the firing rate function in \mathcal{CPN}^c is defined for each instance of each colored transition. The domain of $h_{t(b)}$ is restricted to the set of preplace instances of $t(b)$, denoted by $\bullet t(b)$ with $\bullet t(b) := \{p(c) \in I_P \mid f(p(c), t(b)) \neq 0\}$.

The semantics of \mathcal{CPN}^c is equivalent to that of its unfolded \mathcal{CPN} . For the detailed semantics of \mathcal{CPN} , please refer to [HLGM09].

We still use circadian rhythms to give a \mathcal{CPN}^c model. The \mathcal{CPN}^c model for circadian rhythms also looks like the same in structure as the \mathcal{QPN}^c model in Figure 2.6. We can also use the same rate functions for transitions as those illustrated in Table 2.1.

But they are now not stochastic but deterministic. As there is a difference in graphic representations between discrete and continuous Petri nets, we give the CPN^C model for circadian rhythms in Figure 2.7.

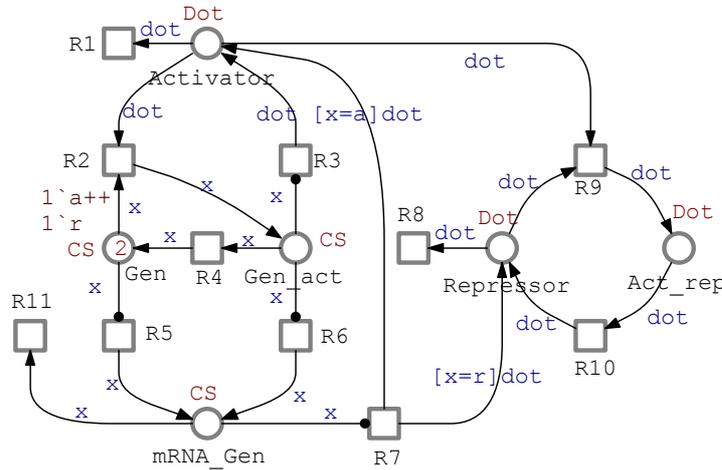


Figure 2.7: A CPN^C model for circadian rhythms. The declarations: *colorset Dot = with dot*, *colorset CS = enum with a, r* and *variable x : CS*.

2.6 Scenarios for Using Colored Petri Nets in Systems Biology

Colored Petri nets provide a means to construct compact models for large-scale systems by folding (overlying) several similar components or objects with similar content and structure. In theory we can use colored Petri nets to model any system, but only in some scenarios we can see big advantages of colored Petri nets. In this section, we are going to find out those scenarios that are suitable for using colored Petri nets for modeling biological systems. The scenarios we have explored so far are summarized as follows.

(1) Modeling a system with repetition of components.

This is the most commonly seen scenario to use colored Petri nets for modeling systems. In this scenario, similar components (cells, receptors, transducers etc.) are represented by one component, which are distinguished by colors, e.g. modeling a tissue with repetition of cells, modeling an organ with repetition of tissues or modeling a transducer cluster with repetition of transducers. Consequently, the size of the whole system usually decreases to that of one component. In contrast, using such approaches as standard

Petri nets, we have to draw every component to form a connected model, which usually becomes very large if the number of components is very big. See systems such as the repressilator [EL00], circadian rhythms [GHG02] and *C. elegans* vulval development [LNUM09] for some examples in this scenario. Specifically in Chapter 5, we will take *C. elegans* vulval development as an example to explore this scenario.

(2) Modeling a system with variation of components.

A system may have a set of similar components with defined variations, e.g. mutants. In this case, we can define each of similar components as a color. In [GLG⁺11], we demonstrate this issue during modeling planar cell polarity in *Drosophila* Wing using colored Petri nets, which shows that colored Petri nets provide a very flexible way to study the variation of components for biological systems.

(3) Modeling a system with organization of components.

In some systems such as the cellular automaton [Ila01], coupled Ca^{2+} channels [NMS05], tissue models or organ models [MFV10], species are organized into regular or irregular patterns over a spatial network in one, two or three dimensions. For such a system, we can model species as colored places and elements of a grid as colors and thus obtain a compact but scalable spatial model. Compared with other approaches, colored Petri nets can not only tackle large-scale models but also explicitly represent the spatial properties of models, which facilitates the description of communication of components and the study of pattern formation [MFV10]. In Chapter 5, we will use coupled Ca^{2+} channels to demonstrate this scenario.

(4) Modeling a system with hierarchical organization of components.

For example, in a system, cells are organized in a grid and further each cell contains several well-organized compartments. For this, we have to model the hierarchical organization of components. Using hierarchical product color sets of colored Petri nets, we can easily cope with this issue. In Chapter 5, we will give a brief discussion about the hierarchical organization of coupled Ca^{2+} channels, in which there are more than one clusters and each cluster contains a number of coupled Ca^{2+} channels. In addition, in [GLTG11] and [GLG⁺11] we have deeply discussed this issue by modeling planar cell polarity in *Drosophila* Wing using colored Petri nets.

(5) Modeling membrane systems.

Membrane systems (also known as P systems) [Pau99] are a very powerful and efficient computational model inspired by the internal organization of living cells with different membranes hierarchically arranged. The membranes enclose compartments where objects evolve by means of specific biochemical reactions. Colored Petri nets provide a suitable way for representing membrane systems or even dynamic membrane systems with creation, merging or dissolving rules, where each object at different compartments is represented as a colored place and compartments are differentiated by colors of a color set. As a result, we not only distinguish and show compartment information in

a colored Petri net model of a membrane system, but also make it more compact. In fact, this addresses an advanced application of colored Petri nets, which we will give in Chapter 5.

(6) Colorizing twin nets.

In the biological context, when we have a net model, we sometimes need create another net model with the identical structure for some experimental purpose, which we call twin nets. For example, Marwan et al. [MSS05] explore the network structure by mixing two cytoplasms of mutants with the identical structure. For this, it is easy to use colored Petri nets for modeling two nets with the same structure, i.e. we only need assign two colors to the original net. This scenario is in fact a special case of the first scenario.

(7) Colorizing T-invariants.

T-invariants play an important role in the preliminary analysis of a Petri net model. In order to clearly display T-invariants, we can employ colored Petri nets to colorize T-invariants; as a result, we not only easily differentiate overlapped T-invariants by using colors, but also can dynamically demonstrate them by animation. To do this, we just need define a color set which contains as many colors as the number of T-invariants, i.e. we define each T-invariant as a color. We then assign the color set to each place of a net, write the same expression to all arcs and write a guard for each transition to indicate which T-invariants cover this transition.

Moreover, we can also use this approach to model and display interesting subnets. For this purpose, we also need define a color set in which we define a subnet as a color. In Chapter 3, we will deeply discuss the last two scenarios.

2.7 Encoding Components of Systems as Colors

In order to use colored Petri nets to model a system, one of the key problems is to encode components of the system as colors. Usually, we need this encoding not only for locating the components represented by the colors but also for easily finding the neighborhood of each component. In the following, we will in detail discuss this issue.

One dimensional space. In one dimensional space, components are arranged in a straight line (only X axis). For example, Figure 2.8 illustrates an arrangement of M components in a line, where we use a square to denote a component. For this, we can define a simple color set $\{x|x = 1, 2, \dots, M\}$, i.e. using one dimensional coordinate x to differentiate each component. For a non-boundary component x , it has two immediate neighbors, $x - 1$ and $x + 1$. If we want to evaluate if a component a is a neighbor of a given component x , we can define the following function that returns a Boolean value "true" or "false" (where "|" denotes "logical or", "&" "logical and" and "!" "logical not".):

$$\begin{aligned}
 & \text{Boolean } IsNeighbor1D(x, a) \\
 & \{ \\
 & \quad (a = x + 1 | a = x - 1) \& \\
 & \quad a \geq 1 \& a \leq M \\
 & \}
 \end{aligned}$$



Figure 2.8: An arrangement of M components in a line.

Two dimensional space. Components can also be arranged in a two dimensional lattice or grid. For example, Figure 2.9 gives an arrangement of $M \times N$ components. For this, we can define a product color set $\{(x, y) | x = 1, 2, \dots, M \text{ and } y = 1, 2, \dots, N\}$, i.e. using two dimensional coordinates (x, y) to denote each component. For a non-boundary component, it has 8 neighbors in terms of the Moore neighborhood and 4 neighbors according to the von Neumann neighborhood [Ter06]. Likewise, we can use the following function to evaluate if a component (a, b) is a neighbor of a given component (x, y) in terms of the Moore neighborhood:

$$\begin{aligned}
 & \text{Boolean } IsNeighbor2D((x, y), (a, b)) \\
 & \{ \\
 & \quad (a = x | a = x + 1 | a = x - 1) \& \\
 & \quad (b = y | b = y + 1 | b = y - 1) \& \\
 & \quad !(a = x \& b = y) \& \\
 & \quad a \geq 1 \& a \leq M \& \\
 & \quad b \geq 1 \& b \leq N \\
 & \}
 \end{aligned}$$

Higher dimensional space. The encoding way above can be easily extended to higher dimensions. The key problem is still the definition of the neighborhood, which would become difficult with the increasing dimensions.

Hierarchically organized systems. For hierarchically organized systems, e.g. a system consisting of clusters each of which is further composed of transducers or a system with a lattice occupied by cells in each of which there is a further regular organization, we

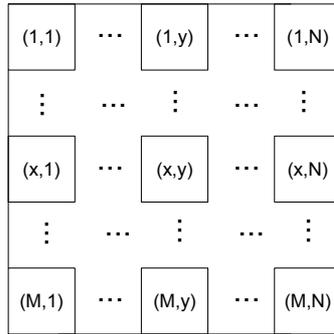


Figure 2.9: An arrangement of $M \times N$ components in a two dimensional lattice.

can use hierarchical color sets. For example, we can use a product color (a tuple) (x_1, y_1) to represent the location of a cluster, in which we can use a second product color (x_2, y_2) to denote the location of a transducer. Now each transducer obtains a hierarchical color $((x_1, y_1), (x_2, y_2))$. We can easily extend this way to a system with more hierarchies.

2.8 Closing Remarks

In this chapter, we have presented a colored Petri net framework for modeling and analyzing biological systems, which relates three modeling paradigms: QPN^c , SPN^c and CPN^c and then described their definitions and behavior. All these three modeling paradigms can be combined together to accomplish the modeling and analysis of biological systems. We have also summarized the scenarios we have explored so far to apply colored Petri nets to modeling biological systems. In addition, we have discussed how to encode components of systems as colors especially for those scenarios. In the future, we will continue to investigate new scenarios using colored Petri nets in the biological context.

3 Some Implementation Aspects

We have implemented all colored Petri net classes defined in Chapter 2 in our Petri net tool Snoopy [RMH10], [LH11]. Now we can use Snoopy to model, animate or simulate colored qualitative, stochastic and continuous Petri nets.

We provide specific support for modeling colored Petri nets. For example, we offer rich data types for color set definition: dot, integer, string, Boolean, enumeration, index, product and union, and support user-defined functions. We allow several extended arc types, inhibitor, read, equal, reset or modifier arcs, and several special transitions, stochastic, immediate, deterministic or scheduled transitions. In addition, we take into account some features suitable for biological modeling, e.g. concise specification of initial marking for larger color sets and automatically colorizing some special subnets. All these features facilitate the modeling of biological systems using colored Petri nets.

We also offer a variety of analysis techniques for colored Petri nets. Using Snoopy, we can not only run automatic animation but also run single-step animation by manually choosing a binding. We allow stochastic simulation, e.g. using the Gillespie stochastic simulation algorithm [Gil77] for colored stochastic Petri nets and continuous simulation for colored continuous Petri nets. Moreover, within Snoopy different net classes can be exported to one another or to some specific formats, e.g. APNN [BKK95]. This allows for applying other tools, e.g. structural analysis or model checking, to further analyzing colored Petri nets.

In this chapter, we do not address all implementation details; instead, we concentrate on three key implementation problems. The first problem is the computation of enabled transition instances, which plays a crucial role in the animation/simulation of colored Petri nets. For this, we will give an efficient algorithm by using a pattern matching mechanism and considering some optimization techniques.

The second problem concerns the unfolding of colored Petri nets. When we want to use the existing analysis techniques and tools of standard Petri nets for colored Petri nets or equip them with stochastic or continuous simulation capabilities, we have to unfold colored Petri nets into their corresponding uncolored ones. For this we will present an unfolding algorithm and discuss how to improve the efficiency of the unfolding process.

The third problem deals with the automatic folding (colorizing) of Petri nets in order to ease the modeling work of colored Petri nets. We will consider to automatically colorize three special cases: T-invariants, master nets and twin nets, which would bring

benefits to biologists for a better understanding of biological networks or reconstructing networks from experimental data.

This chapter is organized as follows. Section 3.1 discusses the computation of enabled transition instances for colored Petri nets. Section 3.2 describes the unfolding algorithm for colored Petri nets. Section 3.3 addresses the folding problem of Petri nets. Section 3.4 concludes this chapter.

3.1 Computation of Enabled Transition Instances

Animation is an important technique for obtaining an intuitive understanding of a Petri net model as it demonstrates the dynamic behavior of the model in a visual way. Nearly all visual tools for modeling Petri nets provide the animation functionality [Pet11]. For standard Petri nets, the core of the animation is the scheduling algorithm for transitions. However, for colored Petri nets, we have to consider another key problem, the computation of enabled transition instances.

When checking whether a transition is fireable or not at a given marking, we have to assign values to the variables that occur in the arc expressions and the guard of the transition. This is called binding. A binding of a transition corresponds to an instance of it. Then we evaluate if the transition respects the firing rule. The introduction of colors to Petri nets makes it difficult to compute their firing rules [JKW07].

The efficiency of the animation for large-scale colored Petri nets is mainly determined by the efficiency of the computation of enabled transition instances, which, however, is a NP-hard search problem because of the expressiveness of colored Petri nets. One theoretically possible way is to make an exhaustive search to check all bindings and then prune invalid ones, which is inefficient at all especially if the transitions have many variables, but only a few bindings can fire the transitions.

In this section, we focus on the problem of the computation of enabled transition instances for colored Petri nets. We adopt the idea given in [KC04], that is, extracting patterns from input arc expressions and guards, then binding these patterns to tokens on input places, and thus obtaining an enabled binding set. The main contribution of this section is that we give a more efficient algorithm for our colored Petri net tool, Snoopy [LH10b], in which we use a new partial binding - partial test principle and several heuristics techniques to compute enabled transition instances.

This section is organized as follows. Section 3.1.1 describes the patterns that are used for the computation of enabled transition instances, and discusses how to classify and find patterns. Section 3.1.2 discusses the binding process and recalls related concepts. Section 3.1.3 gives the computation algorithm. Section 3.1.4 discusses some heuristics that are used to optimize the computation process. Section 3.1.5 summarizes and compares related work. Section 3.1.6 gives the conclusion of this section.

3.1.1 Patterns

We use the same pattern matching mechanism as CPN tools [KC04]. A pattern is defined as an expression with variables which can be matched with other expressions to assign values to variables [KC04]. CPN tools are based on a standard meta language (SML) and thus employ the patterns defined in SML [Ull98]. In contrast, our tool is not based on SML but we consider a subset of SML patterns, as we use less data types than CPN tools. The patterns that we use have the following syntactical structure:

$$\begin{aligned}
 \textit{Pattern} & ::= && \textit{Variable} \\
 & | && \textit{Constant} \\
 & | && \textit{TuplePattern} \\
 \textit{TuplePattern} & ::= && (\textit{Pattern}(\textit{Pattern})^*)
 \end{aligned}$$

Consider the example illustrated in Figure 3.1. According to the syntax of the patterns, we can see that (x, y) is a tuple pattern. If we assign token $(1, a)$ on place $P2$ to (x, y) , we obtain an assignment $x = 1$ and $y = a$. This process is called pattern matching [KC04].

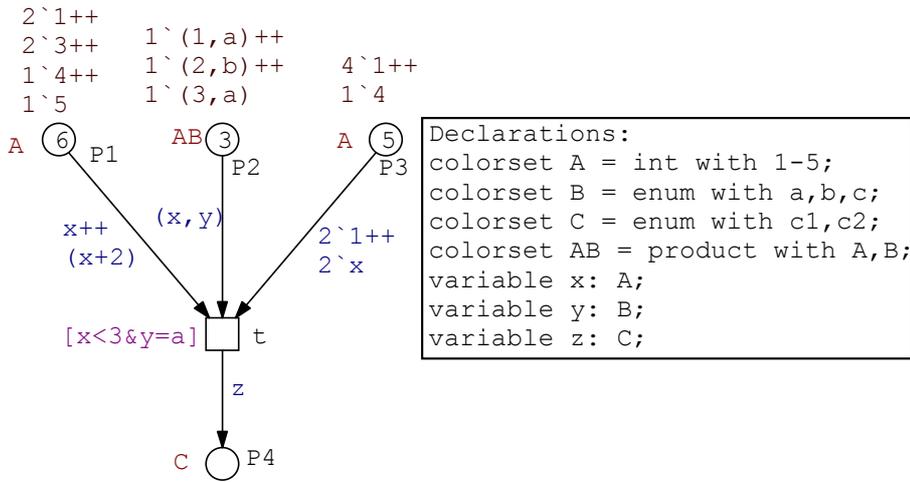


Figure 3.1: A colored Petri net for demonstrating patterns.

Pattern matching provides an easy and efficient way to compute enabled transition instances; therefore, in order to improve the efficiency of the computation, we have to find and use patterns for assigning tokens on places to variables as much as possible. To do this, we can search all input arc expressions of a transition to find available

patterns, which we call a pattern set concerning arc expressions, denoted by $AS(t)$ for a transition t . Besides, we can search the guard of a transition t to find patterns in the guard, denoted by $GS(t)$. These two sets constitute the overall pattern set for a transition t , $PS(t) = AS(t) \cup GS(t)$, which are used to assign tokens to variables. In the following, we in detail discuss how to find these two types of patterns.

Patterns in arc expressions

The patterns in arc expressions are the basic ones that are used for the computation of enabled transition instances. To obtain them, we can search through all input arc expressions of a transition t .

We can write an input arc expression as $c_1 \text{'exp}_1 ++ \dots ++ c_n \text{'exp}_n$, where c_i is the multiplicity of exp_i ($i = 1, 2, \dots, n$) that has the type of its corresponding input place. If exp_i is a pattern, then we add exp_i to $AS(t)$. For example, for the expression $2'1++2'x$ in Figure 3.1, we obtain two patterns: the constant pattern, 1, and the variable pattern, x , while for the expression $x++(x+2)$, we only obtain one variable pattern, x , as $x+2$ is not a pattern. At the same time, we record the multiplicity corresponding to each pattern so as to use it to test bindings once its corresponding pattern is used. For example, for the variable pattern x , if we assign a token to it, for instance '4' on place $P3$, we immediately test if there are enough tokens with color '4' on place $P3$. To do so, the invalid bindings can be discarded earlier.

Patterns in guards

As the guard of a transition often imposes a rather strong constraint on efficient binding, it is better to consider it early when computing bindings. For this, we adopt the similar approach as that in [KC04]. Like [KC04], we consider a guard in the conjunctive form, $g(t) \equiv \bigwedge_{i=1}^n g_i(t)$. For each conjunct, $g_i(t)$, we only consider the following form: $g_{il}(t) = g_{ir}(t)$, where $g_{il}(t)$ and $g_{ir}(t)$ are expressions of patterns, but one of them must be a constant. We add these special expressions to the pattern set $GS(t)$ for a transition t . The advantage of using patterns in guards for binding is obvious. For example, consider the pattern, $y = a$, it directly makes the bindings relating to tokens without the color a , invalid.

Binding variables to color sets

For a transition t , if there are variables that are not covered by $PS(t)$, we have to bind them to their corresponding color sets. For example, for the variables that only appear in output arcs, we have to assign their color sets to them. In Figure 3.1, we can see that the variable z is of this case, which has to be bound to its corresponding color set C .

Formal representation of patterns

We herein give a formal representation of each pattern $S \in PS(t)$ for a transition t , which is a five-tuple $S = \langle P, E, X, M, m \rangle$, where

1. P , the type of the pattern: constant, variable, tuple, or guard (a special pattern),
2. E , the expression of the pattern,
3. X , the set of variables in the pattern,
4. M , the initial/current tokens on the place that connects the arc whose expression contains the pattern, and
5. m , the multiplicity of the pattern.

For the patterns in $AS(t) \subseteq PS(t)$, all components above would be used, but for the patterns in $GS(t) \subseteq PS(t)$, only the first three components P , E and X would be used. For a constant pattern, X will always be empty, denoted by $\{\phi\}$.

For example, the pattern set $PS(t)$ in Figure 3.1 can be formally written as follows:

1. $S_1 = \langle \text{Variable}, x, \{x\}, \{2^1, 2^3, 1^4, 1^5\}, 1 \rangle$
2. $S_2 = \langle \text{Tuple}, (x, y), \{x, y\}, \{1^1(1, a), 1^1(2, b), 1^1(3, a)\}, 1 \rangle$
3. $S_3 = \langle \text{Constant}, 1, \{\phi\}, \{4^1, 1^4\}, 2 \rangle$
4. $S_4 = \langle \text{Variable}, x, \{x\}, \{4^1, 1^4\}, 2 \rangle$
5. $S_5 = \langle \text{Guard}, y = a, \{y\} \rangle$

Optimized pattern set

In order to further improve the efficiency of computation, we define an optimized pattern set like [KC04]. Let $PS(t) = AS(t) \cup GS(t)$ be the pattern set of a transition t . An optimized pattern set $OPS(t)$ for transition t is a set at least satisfying the following conditions:

1. $OPS(t) \subseteq PS(t)$,
2. $GS(t) \subseteq OPS(t)$, and
3. $V(OPS(t)) = V(PS(t))$.

The first item ensures that all members of $OPS(t)$ come from $PS(t)$. The second item states that all guard patterns must be included in $OPS(t)$. The third item ensures that the optimized pattern set should cover all variables that appear in $PS(t)$. Please note that there may be some variables of transition t that are not covered by $PS(t)$, and these variables will be bound to their color sets.

In the preprocessing section below, we will give the steps to obtain an optimized pattern set, $OPS(t)$ for a transition t from its pattern set, $PS(t)$, where we will see more conditions that an optimized pattern set should satisfy.

Besides, we collect other expressions that are not in the optimized pattern set to a test set $TS(t)$ for a transition t , which will be used to test if bindings are valid during the binding process. Each expression $S \in TS(t)$ is denoted by a tuple $S = \langle E, X, M \rangle$, where

1. E , the expression,
2. X , the set of variables in the expression, and
3. M , the initial/current tokens on the place that connects the arc to which the expression belongs.

For the expressions in $TS(t)$, we do not leave them until finishing all bindings and then test them. Instead, we will use the partial binding - partial test principle to test an expression in $TS(t)$ once we find that all variables of it have been bound during the partial binding process. This could exclude invalid bindings as early as possible.

For example, in Figure 3.1, if the variable x is bound by values 1, 3, 4, 5 on place $P1$, we can immediately evaluate and test the expression $x++(x+2)$. As a result, at this moment we can exclude the partial bindings $x = 3$, $x = 4$ and $x = 5$, as place $P1$ has no enough tokens for these bindings.

3.1.2 Binding Process

In this section, we recall the binding process and some related definitions according to [KC04].

In order to evaluate the arc expressions and the guard of a transition t , the variables relating to the transition (denoted by $V(t)$) must be bound to values (tokens). A binding of a transition t is written as: $\langle v_1 = c_1, v_2 = c_2, \dots, v_n = c_n \rangle$, where $v_i \in V(t)$ and c_i is a color of the color set of v_i , $i = 1, 2, \dots, n$.

Matching a token on an input place with a pattern would usually only bind a subset of $V(t)$ for a transition t . For example, consider transition t in Figure 3.1, matching token (1,a) with pattern (x, y) will bind variable x to 1, and y to a , but will not bind variable z to any value. So the concept of partial binding is presented. A partial binding of a

transition is a binding in which not all variables of this transition are bound to values. In the following, we use $PartialBinding(p, c)$ to denote a partial binding by matching a pattern p with a value c . If they are not matched, $PartialBinding(p, c) = \perp$.

In order to obtain a complete binding, we have to gradually merge the partial bindings. For example, in Figure 3.1, matching pattern x and the tokens on $P1$ yields the following four partial bindings:

1. $\langle x = 1, y = \perp, z = \perp \rangle$
2. $\langle x = 3, y = \perp, z = \perp \rangle$
3. $\langle x = 4, y = \perp, z = \perp \rangle$
4. $\langle x = 5, y = \perp, z = \perp \rangle$

Matching pattern (x, y) with the tokens on $P2$ yields the following three partial bindings:

1. $\langle x = 1, y = a, z = \perp \rangle$
2. $\langle x = 2, y = b, z = \perp \rangle$
3. $\langle x = 3, y = a, z = \perp \rangle$

If we merge them, we obtain the following two partial bindings:

1. $\langle x = 1, y = a, z = \perp \rangle$
2. $\langle x = 3, y = a, z = \perp \rangle$

We can continue to match patterns with values and merge them until all variables are bound.

The merging of two partial bindings relates to the concept of compatible bindings. Two partial bindings b_1 and b_2 of a transition t are compatible (written as $Compatible(b_1, b_2)$), if and only if

$$\forall v \in V(t) : b_1(v) \neq \perp \wedge b_2(v) \neq \perp \Rightarrow b_1(v) = b_2(v).$$

For two compatible partial bindings b_1 and b_2 , the combined partial binding, b , (written as $Combine(b_1, b_2)$) can be obtained by:

$$b(v) = \begin{cases} b_1(v) & \text{if } b_1(v) \neq \perp, \\ b_2(v) & \text{if } b_2(v) \neq \perp, \\ \perp & \text{otherwise.} \end{cases}$$

Based on these definitions above, the merging of two partial binding sets B_1 and B_2 is defined as:

$$Merge(B_1, B_2) = \{Combine(b_1, b_2) | \exists (b_1, b_2) \in B_1 \times B_2 : Compatible(b_1, b_2)\}.$$

3.1.3 Algorithms

In this section, we first give a top-level algorithm for computing enabled transition instances, which is illustrated in Algorithm 1. The algorithm inputs the pattern set $PS(t)$ and the test set $TS(t)$ of a transition t , and outputs a complete binding set C .

The algorithm works as follows. First it conducts a preprocessing (Line 1) on $PS(t)$, and obtains an optimized pattern set, $OPS(t)$ by considering some optimization techniques. Afterwards, it executes the *BindbyPatterns* process (Line 2) to assign tokens on places to patterns. After that, it executes the *BindbyColorSets* process (Line 3) to assign color sets to the variables that are not contained in the pattern set, $V(TS(t)) \setminus V(OPS(t))$. During these two processes, the algorithm checks whether the guard is satisfied and whether the input places have sufficient tokens. So, finally we will obtain all valid complete bindings. In the next subsections, we will continue to discuss these three processes in this algorithm in detail.

Algorithm 1: Computing enabled transition instances.

Input: $PS(t), TS(t)$

Output: C

- 1 $OPS(t) = Preprocess(PS(t));$
 - 2 $C = BindbyPatterns(OPS(t), TS(t));$
 - 3 $C = BindbyColorSets(C, TS(t), V(TS(t)) \setminus V(OPS(t)));$
-

Preprocessing of a pattern set

The preprocessing of the pattern set of a transition is very important as it may prune a lot of invalid partial bindings earlier and find whether the transition can be enabled as early as possible, thus improving the efficiency of the computation of enabled transition instances. In this section, we give the steps to preprocess a pattern set, which results in an optimized pattern set.

(1) Testing multiplicity.

We begin the preprocessing of a pattern set with multiplicity testing. During this step, we can discard the tokens in the current marking that do not contribute to valid bindings. This is done by checking whether the number of tokens of the same color is greater than or equal to the multiplicity of a pattern. For a constant pattern, if this is

evaluated to false, we immediately stop the preprocessing process, and directly disable this transition. If true, we now can remove the constant pattern from the pattern set, as we will not use it any longer for the following process. For a variable or tuple pattern, if this is evaluated to false, we will remove these tokens from the current tokens. If the current tokens becomes zero, we stop the preprocessing process, and directly disable this transition. This idea partly comes from [Mäk01] and [Gae96].

The main algorithm is illustrated in Algorithm 2, which works as follows. It executes a loop for each pattern in the pattern set $PS(t)$. If a pattern is a constant pattern, its multiplicity is checked against the number of the current tokens of the constant color. Here $S.M\langle c \rangle$ denotes the number of tokens of color c . If this is evaluated to false, the transition is determined not to be enabled (Lines 2-7). If a pattern is a variable or tuple pattern, for each color in the current tokens, the multiplicity is tested (Lines 10-11). The tokens will be removed if the testing is false. If the current tokens relating to the pattern become empty, the transition is determined to be disabled (Lines 14-18).

After the multiplicity testing for the example in Figure 3.1, we obtain the following pattern set, where pattern S_3 is removed.

1. $S_1 = \langle Variable, x, \{x\}, \{2^1, 2^3, 1^4, 1^5\}, 1 \rangle$
2. $S_2 = \langle Tuple, (x, y), \{x, y\}, \{1^1(1, a), 1^1(2, b), 1^1(3, a)\}, 1 \rangle$
3. $S_4 = \langle Variable, x, \{x\}, \{4^1\}, 2 \rangle$
4. $S_5 = \langle Guard, y = a, \{y\} \rangle$

(2) Merging identical patterns.

Often, there exist several identical patterns (identical expressions) for a transition. Merging them can remove invalid partial bindings as many as possible before the binding begins. The main algorithm is illustrated in Algorithm 3. To merge two identical patterns, for example, S_i and S_j in $OPS(t)$, $i \neq j$, we need to obtain their colors from their current tokens, denoted by C_i and C_j (Lines 1-2), respectively. We calculate the merged colors by $C_k = C_i \cap C_j$ (Line 3). If C_k is not empty, we create a new pattern S_k , where $S_k.M$ saves the merged colors with the multiplicity being 1 and $S_k.m$ is set to 1 (Lines 4-9). At the same time, we remove the old patterns S_i and S_j and add a new pattern S_k to the pattern set. If the set C_k is empty, we can directly set the transition disabled.

For example, Figure 3.1 has two identical patterns: S_1 and S_4 . The colors of the current tokens on their corresponding places are $\{1, 3, 4, 5\}$ and $\{1\}$, respectively, and the merged color is $\{1\}$. So we remove the patterns, S_1 and S_4 and add a new pattern, S_{14} . Now the patterns for Figure 3.1 become:

1. $S_2 = \langle Tuple, (x, y), \{x, y\}, \{1^1(1, a), 1^1(2, b), 1^1(3, a)\}, 1 \rangle$

Algorithm 2: Testing multiplicity.

Input: $PS(t)$
Output: $OPS(t)$

```

1 for each pattern  $S \in PS(t)$  do
2   if  $S$  is a constant pattern then
3      $c \leftarrow S.E$ ;
4     if  $S.M\langle c \rangle < S.m$  then
5       transition  $t$  is disabled;
6     endif
7   endif
8   if  $S$  is a variable or tuple pattern then
9     for each color  $c \in S.M$  do
10      if  $S.M\langle c \rangle < S.m$  then
11         $S.M \leftarrow S.M \setminus \{S.M\langle c \rangle\}$ ;
12      endif
13    endfor
14    if  $S.M$  is not empty then
15       $OPS(t) \leftarrow S$ ;
16    else
17      transition  $t$  is disabled;
18    endif
19  endif
20 endfor

```

Algorithm 3: Merging identical patterns.

Input: S_i, S_j
Output: S_k

```

1  $C_i \leftarrow S_i.M$ ;
2  $C_j \leftarrow S_j.M$ ;
3  $C_k \leftarrow C_i \cap C_j$ ;
4 if  $C_k$  is not empty then
5    $S_k.P \leftarrow S_i.P$ ;
6    $S_k.E \leftarrow S_i.E$ ;
7    $S_k.X \leftarrow S_i.X$ ;
8    $S_k.M \leftarrow C_k$ ;
9    $S_k.m \leftarrow 1$ ;
10 else
11   transition  $t$  is disabled;
12 endif

```

2. $S_{14} = \langle \text{Variable}, x, \{x\}, \{1'1\}, 1 \rangle$
3. $S_5 = \langle \text{Guard}, y = a, \{y\} \rangle$

(3) Sorting patterns in terms of the less different tokens first policy [Gae96].

After that, we can sort the patterns in terms of the less different tokens first policy that will be discussed in detail later. For example, after sorting, the patterns in Figure 3.1 become:

1. $S_5 = \langle \text{Guard}, y = a, \{y\} \rangle$
2. $S_{14} = \langle \text{Variable}, x, \{x\}, \{1'1\}, 1 \rangle$
3. $S_2 = \langle \text{Tuple}, (x, y), \{x, y\}, \{1'(1, a), 1'(2, b), 1'(3, a)\}, 1 \rangle$

(4) Removing redundant patterns.

For a pattern set, after some patterns are matched, all variables in the left patterns maybe have been bound, so these left patterns are not necessary to contribute to the binding process. Instead, they can be used to test if the current bindings are valid or not. Hence, we can remove these patterns from the pattern set and add them to the test set. For example, in Figure 3.1, after we bind S_5 and S_{14} , we will find that all variables in S_2 have been bound. So we can move it to the test set. Now the pattern set for Figure 3.1 becomes:

1. $S_5 = \langle \text{Guard}, y = a, \{y\} \rangle$
2. $S_{14} = \langle \text{Variable}, x, \{x\}, \{1'1\}, 1 \rangle$

After finishing the preprocessing, we finally obtain an optimized pattern set $OPS(t)$ for a transition t , which will be used as the input of the following algorithm.

Binding by matching tokens with patterns

Now we describe a key component of the algorithm for the computation of enabled transition instances, binding by matching tokens on the preplaces of a transtion t and patterns in the pattern set, $OPS(t)$, illustrated in Algorithm 4, which is based on the algorithm in [KC04].

The algorithm executes a loop to handle each member in the pattern set $OPS(t)$. Lines 4-6 consider the guard patterns, where the right hand side of each guard pattern is matched against the left hand side of it. Lines 8-12 consider the matching of the current tokens on a place and an arc expression pattern, where the pattern is bound to each

Algorithm 4: Binding by matching tokens with patterns.

Input: $OPS(t), TS(t)$
Output: C

```
1  $C \leftarrow \phi$ ;  
2 for each pattern  $S \in OPS(t)$  do  
3    $C' \leftarrow \phi$ ;  
   // binding  
4   if  $S$  is a guard pattern and  $S \equiv g_l = g_r$  then  
5      $b \leftarrow PartialBinding(g_l, g_r)$ ;  
6      $C \leftarrow Merge(C, \{b\})$ ;  
7   else  
8     for each color  $c \in S.M$  do  
9        $b \leftarrow PartialBinding(S.E, c)$ ;  
10       $C' \leftarrow C' \cup \{b\}$ ;  
11    endfor  
12     $C \leftarrow Merge(C, C')$ ;  
13  endif  
   // testing  
14  for each expression  $S_t \in TS(t)$  do  
15    if  $V(S_t) \subseteq V(C)$  then  
16      for each binding  $b \in C$  do  
17        if  $S_t.E$  is a guard expression and  $S_t.E\langle b \rangle$  is false then  
18           $C \leftarrow C \setminus \{b\}$ ;  
19        endif  
20        if  $S_t.E$  is an arc expression and  $S_t.E\langle b \rangle > S_t.M\langle c \rangle$  then  
21           $C \leftarrow C \setminus \{b\}$ ;  
22        endif  
23      endfor  
24       $TS(t) \leftarrow TS(t) \setminus \{S_t\}$ ;  
25    endif  
26  endfor  
27 endfor
```

colored token. Lines 14-26 test if each partial binding is valid using the test set $TS(t)$. For an expression in $TS(t)$ whose variables are fully bound, if it is a guard expression and is evaluated to false for a partial binding, then the partial binding is invalid. If the expression is an arc expression and can not obtain enough tokens by evaluating it with a partial binding, then the partial binding is also invalid.

Compared to the algorithm in [KC04], the biggest difference is that our algorithm employs the partial binding - partial test principle, that is, during a partial binding process, if the variables in a test expression have been detected to be fully bound, then we evaluate and test it immediately. As a result, this would not produce any invalid complete binding when the binding process ends.

We still use the example in Figure 3.1 to demonstrate how this algorithm works. For the first loop, the guard pattern $y = a$ is processed, and let y bind to a . Then the pattern S_{14} is processed, and let x be bound to 1. After that, the test expression, e.g. $x < 3$ begins to work as it finds that the variable x has been bound. After these steps, we obtain the following partial binding.

1. $\langle x = 1, y = a, z = \perp \rangle$

Binding variables to color sets

When there are variables that are not covered by a pattern set, they have to be bound to colors of their color sets. The algorithm is illustrated in Algorithm 5. It works as follows. It executes a loop for each variable v in $V(TS(t)) \setminus V(OPS(t))$ that represents all the variables that have to be bound to color sets. Lines 3-7 bind variables to colors. Here $c(v)$ represents the color set of variable v . Lines 8-20 test if the expressions in $TS(t)$ satisfy the guard or have sufficient tokens on the corresponding places.

We continue to apply this algorithm to the example in Figure 3.1. Here, we bind the variable z to the color set C with colors, $c1$ and $c2$. Then we obtain the following complete bindings.

1. $\langle x = 1, y = a, z = c1 \rangle$
2. $\langle x = 1, y = a, z = c2 \rangle$

3.1.4 Optimization Techniques

In this section, we briefly summarize some of the optimization techniques that we use to improve the efficiency of the computation of enabled transition instances.

(1) Partial binding - partial test principle.

As described above, we collect all the arc and guard expressions that do not appear in the pattern set of a transition. We do not leave them until finishing all complete

Algorithm 5: Binding variables to color sets.

Input: $C, V(TS(t)) \setminus V(OPS(t)), TS(t)$
Output: C

// binding

```

1 for each variable  $v \in V(TS(t)) \setminus V(OPS(t))$  do
2    $C' \leftarrow \phi$ ;
3   for each color  $c \in c(v)$  do
4      $b \leftarrow PartialBinding(v, c)$ ;
5      $C' \leftarrow C' \cup \{b\}$ ;
6   endfor
7    $C \leftarrow Merge(C, C')$ ;
   // testing
8   for each expression  $S \in TS(t)$  do
9     if  $V(S) \subseteq V(C)$  then
10      for each binding  $b \in C$  do
11        if  $S.E$  is a guard expression and  $S.E\langle b \rangle$  is false then
12           $C \leftarrow C \setminus \{b\}$ ;
13        endif
14        if  $S.E$  is an arc expression and  $S.E\langle b \rangle > S.M\langle c \rangle$  then
15           $C \leftarrow C \setminus \{b\}$ ;
16        endif
17      endfor
18       $TS(t) \leftarrow TS(t) \setminus \{S\}$ ;
19    endif
20  endfor
21 endfor

```

bindings and then test them. Instead, we test them once we find that all the variables of them have been bound during the partial binding process. For example, in Figure 3.2, the optimized pattern set is x , y and z . If we do not use this policy, we would first obtain $20 \times 30 \times 40$ complete bindings, then test the other expressions $x + 1$ and $y + 1$ using these bindings and finally obtain 480 valid bindings. However, if we use this policy, x is first bound and 20 partial bindings are gotten. After that the expression $x + 1$ is tested, and the valid bindings for x are now 3. Then y is bound, and the partial bindings for x and y become 90. When the expression $y + 1$ is tested, the partial bindings become 12. Finally, the variable z is bound, and the final complete bindings are gotten, whose number is also 480. Obviously, using this principle usually excludes invalid partial bindings earlier.

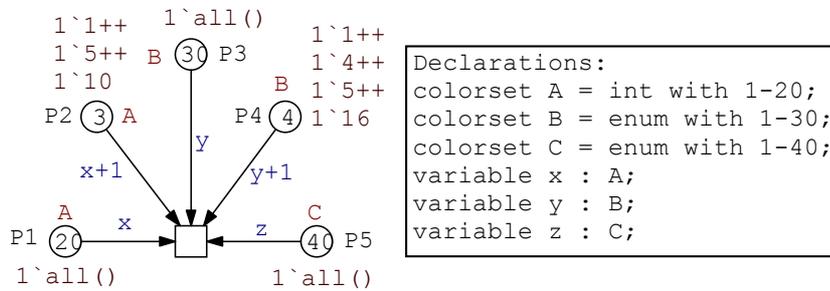


Figure 3.2: A colored Petri net for demonstrating the partial binding - partial test principle.

(2) Less different tokens first policy [Gae96].

As can be easily noticed and analyzed, the information of tokens on different places can affect the efficiency of the computation of enabled transition instances. For example, in Figure 3.1, for transition t , if we first bind x to the tokens on $P1$, we have 4 bindings, but if we bind x to the tokens on $P2$ first, we obtain only 2 bindings. That is to say, the binding order of variables is quite different in efficiency; therefore, we can optimize the binding process by taking account into the binding order. For this we use the less different tokens first policy, which has been given in [Gae96].

(3) Testing multiplicity.

When searching patterns, we also collect the multiplicity for each pattern. We use them to test if places contain enough tokens for enabling before the binding begins, which is already reflected in Algorithm 2. This idea partly comes from [Mäk01] and [Gae96].

(4) Merging identical patterns.

Merging identical patterns before the binding process starts is more efficient than

binding one pattern and then testing the other identical patterns during the binding, which has been discussed in Algorithm 3. This heuristic is very useful when there are many identical patterns for a transition and the tokens available for each pattern are notably different.

All the heuristics have been used in our algorithm, which can be seen in different parts in Algorithm 1-5.

3.1.5 Related Work

In this section, we describe and compare related work concerning the computation of enabled transition instances.

Mäkelä [Mäk01] used a unification technique to calculate enabled transition instances for the algebraic system nets that are in fact another kind of high-level Petri nets, which gave a different idea on finding enabled bindings.

Sanders [San00] considered the calculation of enabled binding as a constraint satisfaction problem. He imposed strong constraints on the form of arc expressions, only considering the constant multiplicity of expressions of such form, $n \text{ 'exp}$, where n is the multiplicity of a color expression exp .

Gaeta [Gae96] studied the enabled test problem of Well-Formed Nets, and gave some heuristics for determining the binding elements, e.g. the less different tokens first policy, which are very useful for improving the efficiency of calculation of enabled transition instances.

Mortensen [Mor01] described data structure and algorithms used in CPN tools. He used the locality principle to discover enabled transitions rather than calculating all the transition each time. He also discussed how to optimize the binding sequences.

Kristensen et al. [KC04] gave a pattern reference algorithm for the enabled binding calculation in CPN tools. We also adopt that idea to design our binding algorithm, but compared their algorithm, our algorithm considers more optimization techniques.

In our work, we take into account the main idea of [KC04] and also some ideas of [Mäk01] and [Gae96], which have been explicitly stated. However, compared with all the previous work, we adopt a new principle, partial binding - partial test, and consider more optimization techniques to improve the efficiency of computing enabled transition instances for colored Petri nets.

3.1.6 Conclusions

In this section, we have presented an algorithm for the computation of enabled transition instances for colored Petri nets. This algorithm uses a pattern matching mechanism and a partial binding - partial test principle and adopts some optimization techniques. The pattern matching mechanism improves the computational efficiency by binding

variables to available tokens on places. The partial binding - partial test principle allows us to test expressions during the partial binding process so as to prune invalid bindings as early as possible. The use of optimization techniques prunes invalid partial bindings before the binding process begins, and also finds the disabled transitions at an early phase. Among them, the less different tokens first policy allows variables to have less bindings, the multiplicity test excludes partial bindings due to insufficient tokens and the merging of identical patterns avoids repeated bindings for identical patterns. All these techniques contribute to the improvements of efficiency. This algorithm can realize an efficient computation of enabled transition instances for large-scale colored Petri nets. In the future, we will investigate more optimization techniques to further improve the computational efficiency.

3.2 Unfolding of Colored Petri Nets

Colored Petri nets provide a compact and convenient way for modeling complex systems, but many basic properties and analysis techniques of standard Petri nets are difficult to extend to colored Petri nets. Therefore it is a reasonable approach to unfold colored Petri nets to equivalent standard Petri nets in order to use existing analysis techniques and tools for standard Petri nets. Fortunately, each of our colored Petri nets corresponds to a standard Petri net as we only support finite color sets. Besides, we can also simulate colored stochastic (continuous) Petri nets using stochastic (continuous) simulation algorithms by automatic unfolding. Thus, unfolding plays an important role in simulating and analyzing colored Petri nets.

In order to unfold a colored Petri net we usually have to make an exhaustive search of colors for places and bindings (transition instances) for transition and then unfold each color to an unfolded place and each transition instance to an unfolded transition. However if we consider some features of color Petri nets and exploit some optimization techniques we do improve the unfolding efficiency.

In this section, we will propose an efficient unfolding algorithm, in which we present two approaches to efficiently compute transition instances. That is, if the color set of each variable in a guard is a finite integer domain, the constrain satisfaction approach is used to obtain all valid bindings; otherwise, a general algorithm is adopted, in which some optimization techniques, e.g. the partial binding - partial test principle, are used.

This section is organized as follows. Section 3.2.1 describes the mapping from colored Petri nets to their unfolded Petri nets. Section 3.2.2 presents an unfolding algorithm. Section 3.2.3 discusses how to compute transition instances for unfolding. Section 3.2.4 summarizes the used optimization techniques. Section 3.2.6 analyzes related work. Section 3.2.5 gives some experimental results. Section 3.2.7 concludes the whole section.

3.2.1 Equivalent Standard Petri Nets

If the color sets of a colored Petri net are finite, it exactly corresponds to an equivalent standard Petri net [Jen92]. In the following, we recall how to obtain an unfolded Petri net for a given colored Petri net by unfolding according to [Jen92].

Definition 19 (Unfolded Petri net)

Let $N = \langle P, T, F, \sum, C, g, f, m_0 \rangle$ be a colored Petri net, its unfolded Petri net $N^* = \langle P^*, T^*, F^*, f^*, m_0^* \rangle$ is defined by:

1. $P^* = I_P$.
2. $T^* = I_T$.
3. $F^* = \{(p(c), t(b)) \in P^* \times T^* \mid (f(p, t)\langle b \rangle)\langle c \rangle > 0\} \cup \{(t(b), p(c)) \in T^* \times P^* \mid (f(t, p)\langle b \rangle)\langle c \rangle > 0\}$.
4. $\forall (p(c), t(b)) \in F^* : f^*(p(c), t(b)) = (f(p, t)\langle b \rangle)\langle c \rangle,$
 $\forall (t(b), p(c)) \in F^* : f^*(t(b), p(c)) = (f(t, p)\langle b \rangle)\langle c \rangle.$
5. $\forall p(c) \in P^* : m_0^*(p(c)) = m_0(p)\langle c \rangle.$

The explanations about the definition are as follows.

1. Each place instance (each color) in the place instance set I_P of the colored Petri net N corresponds to a place of the Petri net N^* . That is, the colored tokens in the colored Petri net are now distinguished by different places in its corresponding Petri net.
2. Each transition instance (each binding) in the transition instance set I_T of the colored Petri net N corresponds to a transition of the Petri net N^* . This means that each binding of the colored Petri net is instantiated as a transition in its corresponding Petri net.
3. If the occurrence of t with binding b removes at least one token of color c from p , denoted by $((f(p, t)\langle b \rangle)\langle c \rangle > 0)$, then an arc that connects $p(c)$ and $t(b)$ exists for the Petri net, whose weight is the number of tokens of color c , denoted by $((f(p, t)\langle b \rangle)\langle c \rangle)$. Analogously, If the occurrence of t with the binding b adds at least one token of color c to p , denoted by $((f(t, p)\langle b \rangle)\langle c \rangle > 0)$, then an arc that connects $t(b)$ and $p(c)$ exists for the Petri net, whose weight is the number of tokens with color c , denoted by $((f(t, p)\langle b \rangle)\langle c \rangle)$.
4. If the initial marking of the colored Petri net N contains tokens of color c on a place p , then the place $p(c)$ of the Petri net N^* has initial tokens, whose coefficient is the number of tokens of color c , denoted by $m_0(p)\langle c \rangle$.

For colored Petri nets with special arcs and time information, they have similar unfolded Petri nets with only minor differences. Specifically, for colored Petri nets with special arcs, we only need to set colored arcs and their corresponding unfolded arcs to the same arc types, while for colored Petri nets with time information, we only need to add such time information to the unfolded transitions.

3.2.2 Unfolding Algorithm

In this section, we present an unfolding algorithm for colored Petri nets without special arcs and time information for sake of simplicity. For colored Petri nets with special arcs and time information, it is easy to consider them in this algorithm. The algorithm is illustrated in Algorithm 6, which works as follows.

The algorithm executes a loop for each transition t of the net N . In each loop, it first computes valid bindings for transition t , which is realized by a sub-algorithm, *ComputeBindings*(t). It then evaluates whether the binding set B is empty if the variable set $V(t)$ of transition t is not empty. If so, the transition either is unfolded to an isolated transtion or excluded from the unfolded net right now (Lines 3-5). Afterwards, for each binding $b \in B$ (corresponding to a transition instance $t(b)$) of transition t , we assign it to a Petri net transition $t^*(b)$ (Line 7). For each pre-arc $F(p, t)$ of t , we first evaluate its expression in terms of binding b , denoted by $(f(p, t)\langle b \rangle)$. For each color c in the evaluated expression $(f(p, t)\langle b \rangle)$, we assign a place instance $p(c)$ to a Petri net place $p^*(c)$ (Line 10). At the same time, the tokens corresponding to color c on place p is assigned to $p^*(c)$ (Line 11). Besides the number of tokens of color c in $(f(p, t)\langle b \rangle)$, denoted $(f(p, t)\langle b \rangle)\langle c \rangle$, is assigned to a Petri net arc expression $(f^*(p(c), t(b)))$ (Line 12) and an arc $(p(c), t(b))$ is added to the unfolded net (Line 13). We deal with the post-arcs in the same way as the pre-arcs.

This algorithm implicitly omits that the guards are evaluated to false during computing bindings, which is in fact an optimal way for unfolding colored Petri nets. Besides, we consider other ways to optimize unfolding, e.g. removal of isolated places or transitions during the unfolding process, not waiting until finishing unfolding. For colored Petri nets with special arcs and time information, we only need to add arc types and time information in this algorithm.

3.2.3 Algorithms for Computing Transition Instances

When we unfold a transition, we have to compute all its instances (or bindings). In fact, the gain in the unfolding efficiency mostly depends on the computation of transition instances. For a transition, the number of its instances is only decided by its guard, which is in fact a logical expression. We can consider the computation of transition instances as a combinatorial problem. If we test a guard for each combination of values, then we would have the combinatorial explosion problem, which we do not want to

Algorithm 6: Unfolding a colored Petri net.

Input: a colored Petri net $N = \langle P, T, F, \Sigma, C, g, f, m_0 \rangle$

Output: an unfolded Petri net $N^* = \langle P^*, T^*, F^*, f^*, m_0^* \rangle$

```

1 for each transition  $t \in T$  do
2    $B = \text{ComputeBindings}(t)$ ;
3   if  $V(t)$  is not empty and  $B$  is empty then
4      $t$  is isolated;
5   endif
6   for each binding  $b \in B$  do
7      $t^*(b) \leftarrow t(b)$ ;
8     for each pre-arc  $(p, t)$  of  $t$  do
9       for each color  $c$  in  $(f(p, t)\langle b \rangle)$  do
10         $p^*(c) \leftarrow p(c)$ ;
11         $m_0^*(p^*(c)) \leftarrow m_0(p)\langle c \rangle$ ;
12         $f^*(p^*(c), t^*(b)) \leftarrow (f(p, t)\langle b \rangle)\langle c \rangle$ ;
13         $(p^*(c), t^*(b)) \leftarrow (p(c), t(b))$ ;
14      endfor
15    endfor
16    for each post-arc  $(t, p)$  of  $t$  do
17      for each color  $c$  in  $(f(t, p)\langle b \rangle)$  do
18         $p^*(c) \leftarrow p(c)$ ;
19         $m_0^*(p^*(c)) \leftarrow m_0(p)\langle c \rangle$ ;
20         $f^*(t^*(b), p^*(c)) \leftarrow (f(t, p)\langle b \rangle)\langle c \rangle$ ;
21         $(t^*(b), p^*(c)) \leftarrow (t(b), p(c))$ ;
22      endfor
23    endfor
24  endfor
25 endfor

```

expect. In this situation, we can think of using the constraint satisfaction approach to solve the problem of the computation of transition instances for it has been used in many combinatorial problems, e.g. scheduling and timetabling in operational research. In the following, we will first address this issue and then consider a more general algorithm for computing transition instances.

Computing transition instances using the constraint satisfaction approach

A constraint satisfaction problem (CSP) [BPS99], [Tsa93] is to pick a value from a given finite domain and then assign it to each variable in a problem, so that all constraints concerning the variables are satisfied.

Formally, a constraint satisfaction problem is a triple $CSP = \langle V, D, C \rangle$, where:

- $V = \{v_1, v_2, \dots, v_n\}$ is a set of variables.
- $D = \{d_1, d_2, \dots, d_n\}$ defines a finite domain d_i of possible values for each variable v_i , $i = 1, 2, \dots, n$.
- C is a set of constraints on variables in V , which restricts the values taken by the variables.

A state of a CSP is defined by an assignment of values to variables. A complete assignment means that each variable gets assigned a value. A solution to a CSP is a complete assignment satisfying all the constraints. For our purpose, we will restrict the domains of variables to finite integer domains and we want to find all solutions of a CSP. We implement the constraint satisfaction approach in Snoopy using Gecode, an open constraint solving library [Gec11].

Take Figure 3.3 as an example, we will see how to formulate the computation of transition instances as a CSP, which can take the following steps.

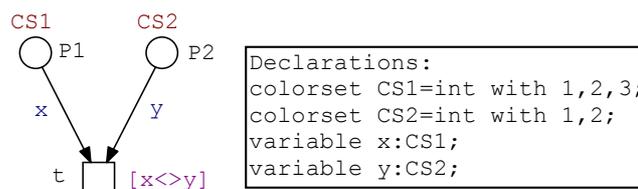


Figure 3.3: A colored Petri net for demonstrating the constraint satisfaction approach.

1. Obtain the guard of a transition, e.g. $x \neq y$ (x is not equal to y) for transition t , which is a logic expression.

2. Obtain all the variables in the guard, e.g. x and y in $x \langle \rangle y$, which will be the variables of the CSP.
3. Define the color set of each variable as its domain in the CSP. For example, the color sets $CS1$ and $CS2$ will become the domains of x and y in the CSP, respectively.
4. Define the guard as a constraint in the CSP. For example, $x \langle \rangle y$ will become a constraint of the CSP. If the guard is a conjunctive form, we can define each conjunct as a constraint.

Using this procedure, we can formulate Figure 3.3 as a CSP: $V = \{x, y\}$, $D = \{CS1, CS2\}$ and $C = \{x \langle \rangle y\}$. Then we can obtain all the solutions:

1. $\langle x = 1, y = 2 \rangle$
2. $\langle x = 2, y = 1 \rangle$
3. $\langle x = 3, y = 1 \rangle$
4. $\langle x = 3, y = 2 \rangle$

That is, transition t in Figure 3.3 has four instances.

A general algorithm for computing transition instances

However, if the guard of a transition is always evaluated to true or not all the variables in the guard are integer types, we have to take another way to compute transition instances. Here we will adopt a similar approach as that in Section 3.1.

Unlike the computation of enabled transition instances in Section 3.1 where variables are bound to tokens on places, during the computation of transition instances for unfolding, all variables have to be bound to the colors of their color sets. As the algorithm here is similar as that in Section 3.1, we do not give the formal algorithm any more and only discuss how to adapt the algorithm in Section 3.1 to obtain an algorithm for the unfolding purpose by illustrating their differences in the following.

We can use the same top-level algorithm as in Algorithm 1, but there are some differences in each sub-algorithms. We use a slightly different way to preprocess the pattern set, which will be discussed in detail. In the *BindbyPatterns* algorithm (see Algorithm 4), Line 8 is now read as binding the variables to the colors of their color sets rather than tokens on places. Lines 20-22 are removed because we do not need to test if places have enough tokens. In the *BindbyColorSets* algorithm (see Algorithm 5), Lines 14-16 are removed because we do not need to test if places have enough tokens.

Before we discuss the preprocessing process, we have to slightly modify the data structure of the pattern $S \in PS(t)$ for a transition t as $S = \langle P, E, X, M \rangle$

1. P , the type of the pattern: constant, variable, tuple or guard (a special pattern),
2. E , the expression of the pattern,
3. X , the set of variables in the pattern, and
4. M , the color set relating to the pattern.

In this data structure we now have to collect the color set for a pattern rather than the initial tokens on its corresponding place as we only need to bind patterns to color sets and we do not need the multiplicity testing for a pattern any more. In the following, we discuss how to modify the preprocessing algorithms in Section 3.1 to deal with the current problem.

For the unfolding purpose, we do not need to test the multiplicity for each pattern, so we do not use Algorithm 2, but keep the left three steps: merging identical patterns, sorting patterns and removing redundant patterns. Among them, in the merging of identical patterns (see Algorithm 3), Lines 1-2 are now read as merging two color sets for two patterns and Line 4 and Lines 9-12 are removed. In the sorting of patterns we need to change to sort patterns in terms of the number of colors in color sets, which we call the less colors first policy.

The reasons for merging identical patterns are as follows. We support to define subsets of a color set, which then may be used by different places relating to a transition and maybe have identical arc expressions for this transition, which can be seen as identical patterns. Merging these identical patterns means to obtain the intersection of the subsets of a color set, which can reduce the number of the values to which variables are bound before the binding process begins.

For example, Figure 3.4 has the following patterns before preprocessing.

1. $S_1 = \langle \text{Variable}, x, \{x\}, A \rangle$
2. $S_2 = \langle \text{Variable}, y, \{y\}, B \rangle$
3. $S_3 = \langle \text{Variable}, x, \{x\}, A1 \rangle$
4. $S_4 = \langle \text{Tuple}, (x, y), \{x, y\}, AB \rangle$
5. $S_5 = \langle \text{Guard}, y = a, \{y\} \rangle$

After merging identical patterns, we obtain:

1. $S_2 = \langle \text{Variable}, y, \{y\}, B \rangle$
2. $S_3 = \langle \text{Variable}, x, \{x\}, A1 \rangle$
3. $S_4 = \langle \text{Tuple}, (x, y), \{x, y\}, AB \rangle$

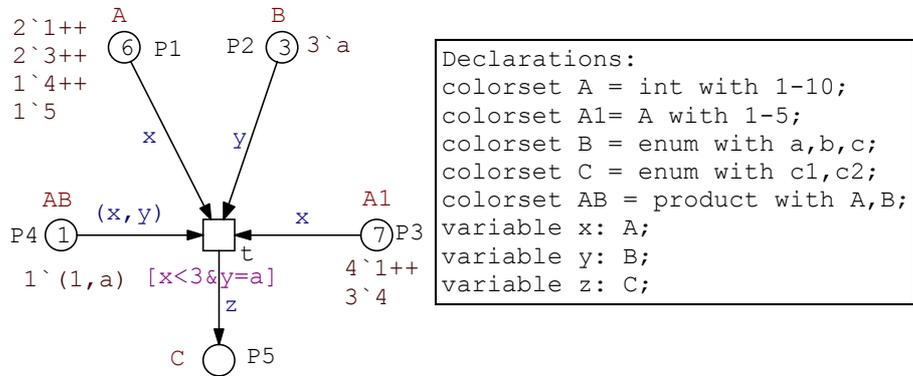


Figure 3.4: A colored Petri net for computing transition instances.

$$4. S_5 = \langle \text{Guard}, y = a, \{y\} \rangle$$

After sorting the patterns, we further obtain:

1. $S_5 = \langle \text{Guard}, y = a, \{y\} \rangle$
2. $S_2 = \langle \text{Variable}, y, \{y\}, B \rangle$
3. $S_3 = \langle \text{Variable}, x, \{x\}, A1 \rangle$
4. $S_4 = \langle \text{Tuple}, (x, y), \{x, y\}, AB \rangle$

After removing redundant patterns, we finally obtain an optimized pattern set $OPS(t)$ for transition t :

1. $S_5 = \langle \text{Guard}, y = a, \{y\} \rangle$
2. $S_3 = \langle \text{Variable}, x, \{x\}, A1 \rangle$

After the preprocessing, we begin to bind patterns to color sets and then obtain the following partial bindings:

1. $\langle x = 1, y = a, z = \perp \rangle$
2. $\langle x = 2, y = a, z = \perp \rangle$

For the left variables, we bind them to their color sets. For this example, we finally obtain the complete bindings:

1. $\langle x = 1, y = a, z = c1 \rangle$
2. $\langle x = 2, y = a, z = c1 \rangle$
3. $\langle x = 1, y = a, z = c2 \rangle$
4. $\langle x = 2, y = a, z = c2 \rangle$

Using our unfolding tools, for this example we obtain an unfolded Petri net, illustrated in Figure 3.5.

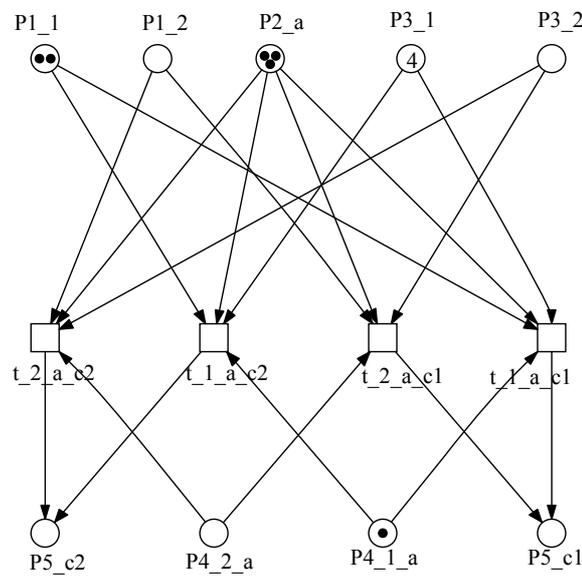


Figure 3.5: The unfolded Petri net for Figure 3.4.

In summary, the features of our algorithm are as follows:

- The algorithm also employs the partial binding - partial test principle, that is, during a partial binding process, if the variables in a test expression have been detected to be fully bound, then we evaluate and test it immediately. As a result, this would not produce any invalid complete binding when the binding process ends.
- If we use subsets of color sets, the pattern matching approach can make us bind variables to subsets of color sets. This prevents a lot of useless bindings from the difference between color subsets and their father color sets.

- The algorithm also makes full use of the guard patterns to keep false transition instances away as early as possible.

3.2.4 Optimization Techniques

In this section, we briefly summarize some of the optimization techniques that we use to improve the unfolding efficiency.

(1) Optimized computation of transition instances.

In our unfolding algorithm, we adopt an optimized algorithm to compute transition instances and thus improve the efficiency of unfolding. The optimization techniques include:

- constraint satisfaction approach,
- partial binding - partial test principle,
- merging identical patterns, and
- less colors first policy.

(2) Removal of false guarded transitions.

We remove all transition instances whose guards are evaluated to false. Unlike [KLPA06], we remove them in the binding process, not waiting until finishing the unfolding and then removing them.

(3) Removal of isolated places or transitions.

As the isolated places or transitions do not contribute to the behavior of the whole net, so they can be removed from the unfolded Petri nets. Unlike [KLPA06], we can remove them in the binding process, not waiting until finishing the unfolding and then removing them.

3.2.5 Experimental Results

We now compare the computational efficiency of unfolding before and after using the optimization techniques, especially the constraint satisfaction approach given in Section 3.2.4. The model we use is illustrated in Figure 3.6 (Table 3.1 gives its declarations.), which simulates the diffusion of chemical signals in a grid to form a chemical gradient. This model can be parameterized by the row M and the column N of the grid.

The experimental results are shown in Table 3.2. From it, we can clearly see that the unfolding efficiency improves greatly after we use the optimization techniques, especially the constraint satisfaction approach. For example, for a 100×100 grid, the unfolding time before optimization is about 933 times as long as that after optimization. The unfolding algorithm in this section basically can tackle large-scale models within a reasonable time.

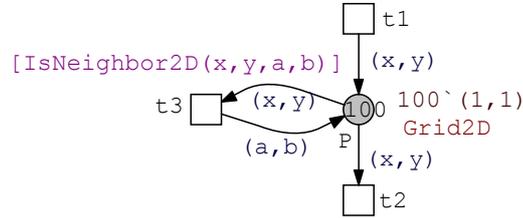


Figure 3.6: A diffusion model for testing the unfolding efficiency.

Table 3.1: Declarations for the diffusion model in Figure 3.6.

| Type | Declaration |
|----------|---|
| constant | $int : M = 100;$ |
| constant | $int : N = 100;$ |
| colorset | $CD1 = int \text{ with } 1 - M;$ |
| colorset | $CD2 = int \text{ with } 1 - N;$ |
| colorset | $Grid2D = product \text{ with } CD1 \times CD2;$ |
| variable | $x : CD1,$ |
| variable | $y : CD2;$ |
| variable | $a : CD1;$ |
| variable | $b : CD2;$ |
| function | $bool \ IsNeighbor2D$ $(CD1 \ x, CD2 \ y, CD1 \ a, CD2 \ b)$ $\{(a=x \mid a = x+1 \mid a = x-1) \ \& \ (b=y \mid b = y+1 \mid b = y-1) \ \&$ $(!(a=x \ \& \ b=y)) \ \& \ (a \leq D1 \ \& \ b \leq D2) \ \& \ (a \geq 1 \ \& \ b \geq 1); \}$ |
| function | $bool \ IsLateral$ $(CD1 \ x, CD2 \ y, CD1 \ a, CD2 \ b)$ $\{(a=x \ \& \ b=y+1) \mid (a=x \ \& \ b=y-1) \mid (a=x-1 \ \& \ b=y) \mid (a=x+1 \ \& \ b=y); \}$ |
| function | $bool \ IsDiagonal$ $(CD1 \ x, CD2 \ y, CD1 \ a, CD2 \ b)$ $\{(a=x+1 \ \& \ b=y+1) \mid (a=x-1 \ \& \ b=y-1) \mid (a=x-1 \ \& \ b=y+1) \mid (a=x+1 \ \& \ b=y-1); \}$ |

Table 3.2: Comparison of the size of the diffusion model in Figure 3.6 and unfolding runtime*.

| $M \times N$ | Size | | Unfolding time (seconds) | |
|------------------|--------|-------------|--------------------------|------------------|
| | Places | Transitions | before optimizing | after optimizing |
| 10×10 | 100 | 884 | 4.024 | 1.024 |
| 50×50 | 2500 | 24,404 | 2,057.318 | 8.532 |
| 100×100 | 10,000 | 98,804 | 40,301.145 | 43.199 |
| 200×200 | 40,000 | 397,604 | ◇ | 238.429 |

* done on PC, Intel(R) Xeon(R) CPU 2.83GHz, RAM 4.00GB.

◇ we did not obtain the result within 24 hours.

3.2.6 Related Work

In this section, we give the analysis of related work.

In [Mäk01], unfolding is obtained by an enabling test algorithm, so the places of an unfolded net are all ever marked under the initial marking, that is, all other places that can not get marked under the initial marking are excluded. Maria uses an explicit data structure to represent unfolded Petri nets.

Kordon et al. [KLPA06] use symbolic representation, data decision diagrams, to represent unfolded nets, but the optimization, such as removal of false guarded transitions, is performed on unfolded nets, which is not efficient.

Compared to those work, our unfolding algorithm adopts an efficient algorithm to compute transition instances, in which some optimization techniques are used to improve the efficiency of unfolding. The disadvantage is that we still adopt an explicit representation of unfolded Petri nets. In the next step, we will concentrate on how to combine the efficient unfolding algorithm with a compact data structure.

3.2.7 Conclusions

In this section, we have presented an efficient unfolding algorithm for colored Petri nets, which contains an optimized algorithm for computing transition instances. Compared with existing unfolding algorithms, our algorithm has some advantages, e.g. most optimization techniques are used before the real unfolding begins. In the future, we will concentrate on how to equip the unfolding algorithm with a compact data structure.

3.3 Folding of Petri Nets

Folding is a challenging approach to obtain a colored Petri net for a given standard Petri net. Just like folding a paper, two halves of this paper become overlaid, thus the

size decreasing to $1/2$. Generally speaking, folding a Petri net means grouping several similar subnets and then overlay them, which we also call coloring. Folding can be realized manually or automatically. Although automatic folding is usually attractive, to find similar subnets from a net for a given subnet (pattern) involves a subgraph isomorphism problem, which is NP-complete [Coo71].

In this section, we do not address the automatic folding based on subgraph isomorphism. Rather we consider three special scenarios concerning automatic folding: coloring T-invariants, master nets and twin nets. In the first two scenarios, we can find subnets of a Petri net through T-invariants or T-invariant-like subnets, while in the third scenario we only need to colorize two copies for a given half of twin nets. In the following, we will in detail describe them.

3.3.1 Colorizing T-invariants of Petri Nets

T-invariants play an important role in the preliminary analysis of a Petri net model. For a Petri net, we usually use some analysis tools, e.g. Charlie [Cha11], to obtain its T-invariants and then analyze if it is covered by these T-invariants. Although this process can be aided by graphical display, e.g. to mark different T-invariants with different colors, sometimes some T-invariants are overlapped, so it is not easy to differentiate them.

In order to more clearly show T-invariants, we employ colored Petri nets to colorize T-invariants. To do so, we not only easily differentiate overlapped T-invariants by using colors, but also can dynamically exploit them via animation.

For this, we can define a color set that contains as many colors as the number of T-invariants of a net, i.e. we define each T-invariant as a color. We then assign the color set to each place of this net and write the same expression for all arcs. Finally we use guards to indicate whether a transition is covered by a T-invariant or not.

The core algorithm for colorizing T-invariants is illustrated in Algorithm 7. The algorithm begins with the processing of each transition t . It first executes a loop to write the information of all T-invariants (or colors) that contain t to its guard $G(t)$ (Lines 1-3). Each item $x = c$ is considered as a logical disjunct of $G(t)$. After that, it executes another loop (Lines 4-10) to cope with the marking of the preplaces of t . In this loop, it first gets each preplace ($SourceNode(e)$) from the target edges of t ($TargetEdges(t)$) and evaluates if it is not empty. If so, this preplace is assigned as many tokens as the number of its black tokens ($\#(p)$) for each color in $T_{inv}(t)$.

Figure 3.7 [HGD08] and Figure 3.8 give an example to demonstrate how to colorize T-invariants using colored Petri nets. This net is covered by two minimal T-invariants: $y_1 = (1, 0, 2) = (r1, 2 \bullet r3)$ and $y_2 = (0, 1, 1) = (r2, r3)$. Therefore we define a color set CS containing two colors, e.g. 1 and 2, and assign it to all places. We define a variable x of the color set CS and assign it to each arc. As $r1$ is covered by y_1 , we define its guard

Algorithm 7: A core algorithm for coloring T-invariants.

Input: a transition: t , the T-invariants/colors of t : $T_{inv}(t)$, a variable: x
Output: the guard of t : $G(t)$, the marking of the net: M

```

1 for each color or T-invariant  $c \in T_{inv}(t)$  do
2    $G(t) \leftarrow "x = c";$ 
3 endfor
4 for each edge  $e \in TargetEdges(t)$  do
5    $p = SourceNode(e);$ 
6   if  $\#(p)$  is not empty then
7     for each color or T-invariant  $c \in T_{inv}(t)$  do
8        $M(p) \leftarrow \#(p) \cdot c;$ 
9     endfor
10  endif
11 endfor

```

as $x = 1$. Similarly, the guards of $r2$ and $r3$ are $x = 2$ and $x = 1|x = 2$, respectively. We can observe the behavior of these T-invariants by animation. For example, when we put a token with color "2" to place A, then if we animate this net, we can see the flow of T-invariant y_2 .

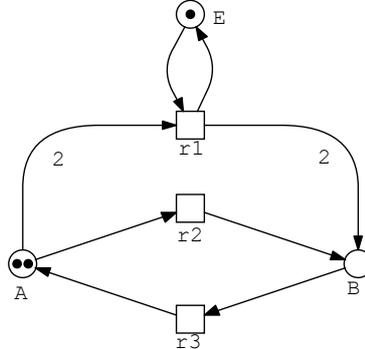


Figure 3.7: A Petri net model for coloring T-invariant. This net has two minimal T-invariants: $y_1 = (1, 0, 2) = (r1, 2 \bullet r3)$ and $y_2 = (0, 1, 1) = (r2, r3)$.

Moreover, we can easily extend this approach to colorize other kinds of interesting subnets, such as master nets that will be described in the next section. For this purpose, we also can define a color set in which we define each subnet as a color.

In the following, we apply this approach to a biological example, the hypoxia response

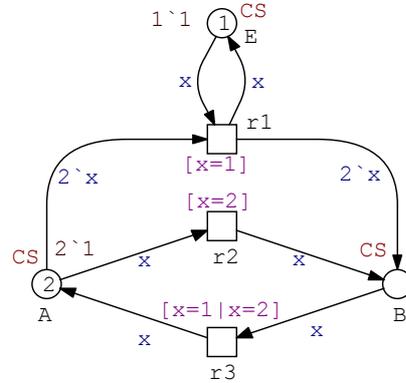


Figure 3.8: A colored Petri net model for Figure 3.7. The declarations: *colorset CS = int with 1,2, and variable x : CS.*

network [HS10], to demonstrate its usefulness. The hypoxia response network consists of three pathways: one oxygen-independent pathway and two oxygen-dependent pathway which are responsible for degrading HIF (the Hypoxia Induced Factor) transcription factor. The Petri net model for the hypoxia response network (see [HS10]) enjoys ten minimal T-invariants, illustrated in Table 3.3, which contribute much to a better understanding of the structure of the net. See [HS10] for more details.

Using the automatic colorizing functionality for T-invariants in Snoopy, when we input the T-invariant file, we obtain a colored Petri net model, illustrated in Figure 3.9. Using the colored Petri net model, we can run animation to observe the flow of a T-invariant when we choose a corresponding initial marking set.

Table 3.3: The minimal T-invariants of the hypoxia response network model.

| No. | Transitions | No. | Transitions |
|-----|-------------|-----|---------------------------------|
| 1 | k5, k6 | 6 | k1, k2 |
| 2 | k12, k13 | 7 | k21, k22 |
| 3 | k18, k19 | 8 | k1, k12, k14, k18, k20 |
| 4 | k15, k16 | 9 | k1, k3, k15, k17, k18, k20, k22 |
| 5 | k3, k4 | 10 | k29, k30 |

3.3.2 Colorizing Master Petri Nets

The network reconstruction problem [MWW08] aims to find a fitting model, e.g. a Petri net model, from a group of given experimental data by considering all possible

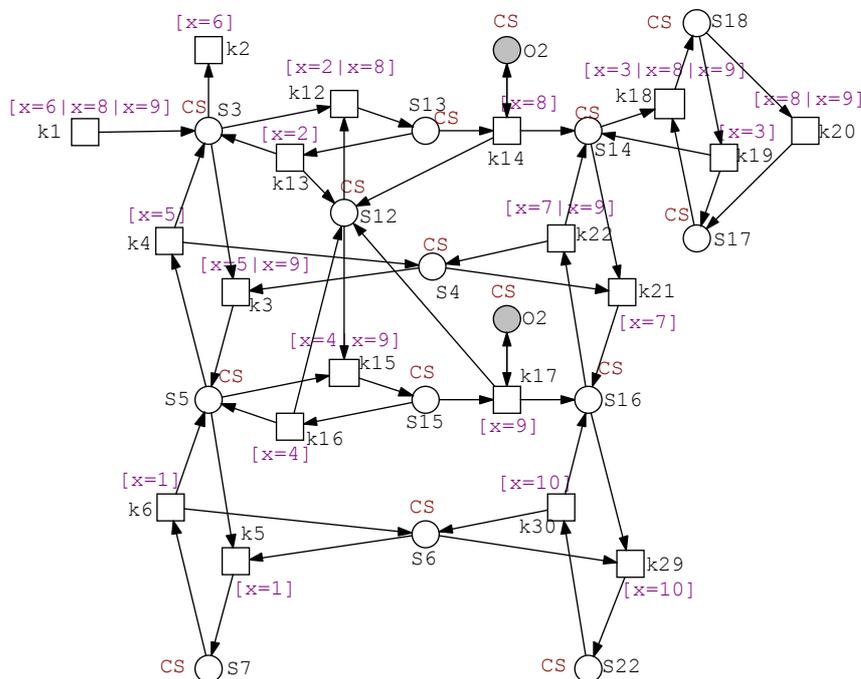


Figure 3.9: A colored Petri net model for the hypoxia response network. The declarations: *colorset* $CS = int$ with $1 - 10$, and *variable* $x : CS$. Please note that for clarity we hide all arc expressions which are all set to x .

models (or networks). For this, we have to construct each of these possible networks one by one and then compare its behavior with experimental data, which is usually time-consuming especially for a large number of possibilities.

To alleviate this process, we propose to use colored Petri nets to model all possible networks (master Petri nets) for a group of experimental data, i.e. to consider each possibility as a color. After that, we can run simulation for this model in different settings to decide the fitting network.

Figure 3.10 gives a demonstration example, where for three substrates, $P1$, $P2$ and $P3$, there are two reaction possibilities: only one reaction $t1$ or two reactions $t1$ and $t2$. For these two candidate networks (possibilities), we can model them as a colored Petri net, illustrated in Figure 3.11. For this colored net, if we set the initial marking to color 1 (2), then the first (second) network will work. Thus, in order to find the fitting network, we only need to change the marking setting before each simulation.

To implement automatic colorizing of master Petri nets, we now have to let the user write them in the same file format as T-invariants. After that, we will deal with them

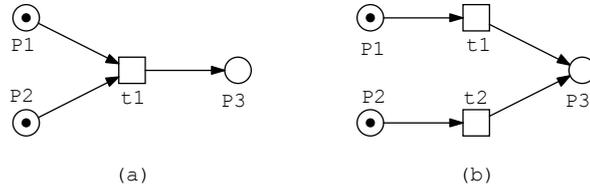


Figure 3.10: Two Petri net models (reaction possibilities) for demonstrating master nets.

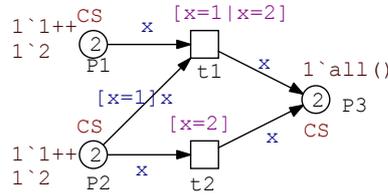


Figure 3.11: A colored Petri net model for Figure 3.10. The declarations: *colorset* $CS = \text{int with } 1, 2$, and *variable* $x : CS$.

in the same way as the colorizing of T-invariants. As this format can not save the information about arcs, we have to rewrite some arc expressions manually at present. But we are going to develop a suitable format for saving master Petri nets and until then we will realize fully automatic colorizing of master Petri nets.

In the following, we give an application of the colorizing of master Petri nets by taking the sensory control of sporulation in *P. polycephalum* [MWW08] as an example. During the cell differentiation process, the photoreceptor protein phytochrome Pfr that detects the far-red light Fr can turn into another stage Pr , which can go back to Pfr by the red light Re . Figure 3.12 illustrates a common network for this process. From this figure, we can see that we need to find the suitable reactions among three places, Pfr , Pr and Re (Figure 3.12 (a)) by considering three possibilities (Figure 3.12 (b)-(d)). See [DWW10] and [MWW08] for details.

Obviously, we can use colored Petri nets to model the whole system, illustrated in Figure 3.13. For readability, we use logic places to separate the main network and the possible networks.

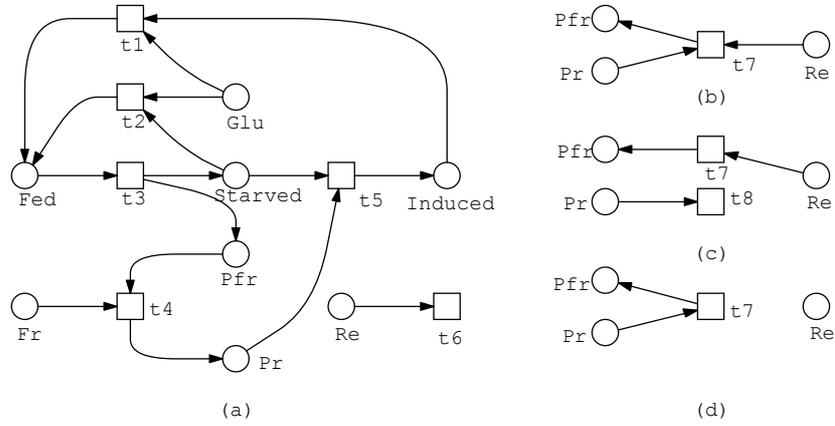


Figure 3.12: A Petri net model for the sensory control of sporulation in *P. polycephalum* (a) and three possible subnetworks (b)-(d), which are adapted from [DWW10].

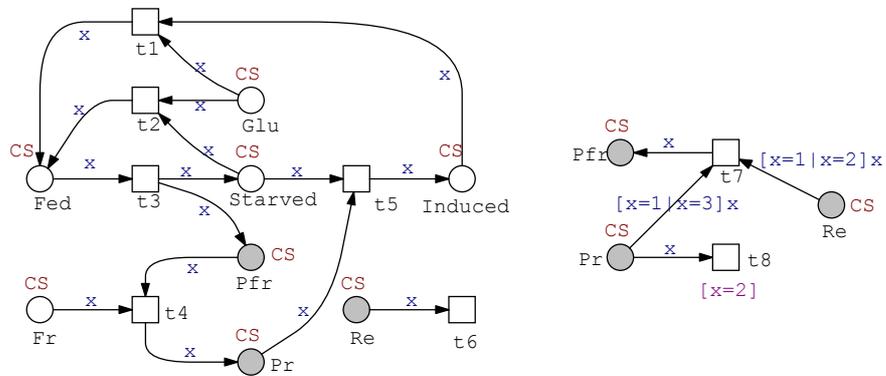


Figure 3.13: A colored Petri net model for the sensory control of sporulation in *P. polycephalum* in Figure 3.12. The declarations: *colorset* $CS = int$ with 1, 2, 3, and *variable* $x : CS$.

3.3.3 Colorizing Twin Nets

During the reconstruction of the regulatory network for sporulation of *Physarum polycephalum* in [MSS05], Marwan et al. model the fusion of two mutant plasmodia. As these two plasmodia have identical signaling subnetworks (Figure 3.14 illustrates the signaling subnetwork for one plasmodium), we call them twin nets.

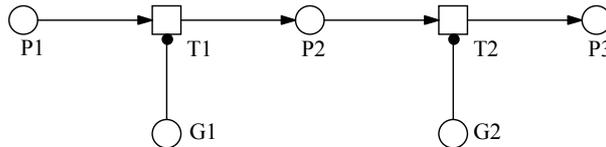


Figure 3.14: The signaling subnetwork for one plasmodium, which is adapted from [MSS05].

It is natural that we can fold twin nets to construct a colored Petri net model by defining two colors, e.g. a and b , differentiating each half of twin nets. We can also extend to any kind of twin nets.

In order to implement automatic folding of twin nets, we informally summarize our algorithm as follows (for a given half of twin nets, e.g. Figure 3.14).

1. define a color set Dot and a color set CS with two colors a and b , each representing a half of twin nets,
2. declare a variable x of the color set CS ,
3. assign the color set CS to each place of the net,
4. create a new transition, e.g. t , for each place p in the original net,
5. create a new edge from p to t and assign it an expression x ,
6. create a new edge from t to p and assign it an expression $+x$,
7. create another new place P_f with the color set Dot and connect it to each newly created transition using a read arc, and
8. create a scheduled transition connecting P_f .

For example, for Figure 3.14, we can obtain two equivalent colored Petri net models, illustrated in Figure 3.15 and Figure 3.16. The latter gives a more compact representation than the former due to the use of a coarse node. If we unfold them, they produce the same uncolored Petri net model as in [MSS05].

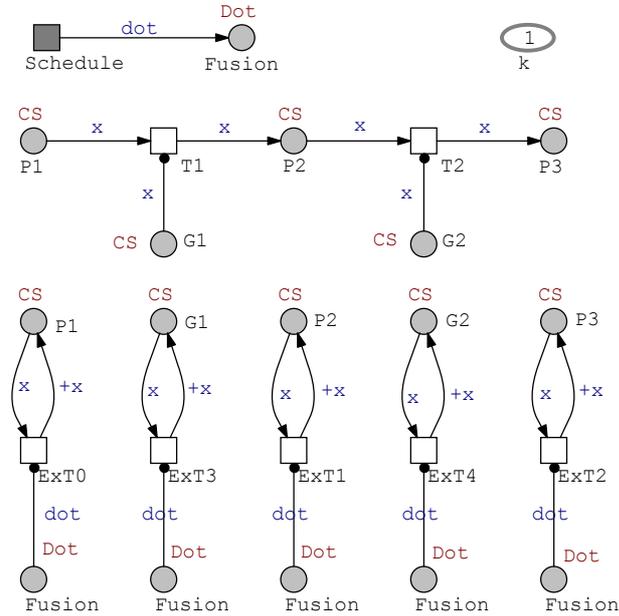


Figure 3.15: A colored Petri net model for Figure 3.14. The declarations: *colorset Dot = with dot*, *colorset CS = enum with a, b*, and *variable x : CS*.

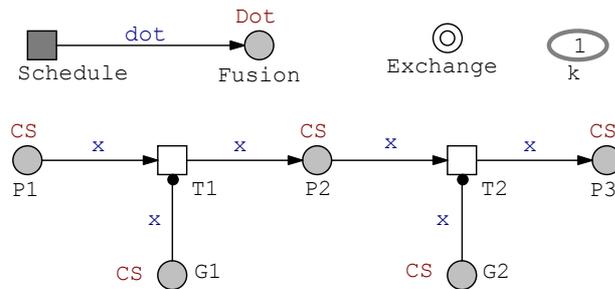


Figure 3.16: A colored Petri net model for Figure 3.14 with a coarse node "Exchange". The coarse node contains the bottom subnet of Figure 3.15. The declarations: *colorset Dot = with dot*, *colorset CS = enum with a, b* and *variable x : CS*.

3.3.4 Conclusions

In this section, we have discussed how to automatically colorize three special scenarios: T-invariants, master nets and twin nets, which may be interesting to biologists. In summary, colorizing T-invariants contributes to the further understanding of T-invariants for a biological network, while colorizing master nets or twin nets offers a convenient way for reconstructing biological networks from experimental data. In the future, we will exploit automatic folding of Petri nets based on subgraph isomorphism by considering the characteristics of Petri nets.

3.4 Closing Remarks

In this chapter, we have addressed three key implementation aspects. The first aspect is the computation of enabled transition instances. For this, we have given an efficient algorithm by using a pattern matching mechanism and considering some optimization techniques. The second problem concerns the unfolding of colored Petri nets, and we have presented an unfolding algorithm and discussed how to improve the efficiency of the unfolding process. In the third problem, the automatic folding (colorizing) of Petri nets, we have discussed how to automatically colorize three special cases: T-invariants, master nets and twin nets, which would bring benefits to biologists for a better understanding of biological networks or reconstructing networks from experimental data.

In the future, for the first two aspects, we will continue to explore more efficient techniques to improve the computation efficiency in order to satisfy new requirements. For the last problem, we not only need to consider more special folding problems, but we also will focus on the automatic folding of Petri nets based on subgraph isomorphism by considering the potential characteristics of Petri nets.

4 Analysis Techniques

The animation of colored Petri nets enables us to experience the model behavior by following the token flow, which establishes the initial confidence in the model. The simulation of colored Petri nets strengthens this confidence by allowing us to investigate specific simulation traces. However, neither animation nor simulation is enough to obtain a credible model. In order to gain deeper insights into the constructed models, formal analysis techniques have to be investigated.

As discussed in Chapter 3, colored Petri nets can be unfolded into equivalent standard Petri nets, so we can use existing analysis techniques and tools for standard Petri nets to analyze colored Petri nets. In this chapter, we will first discuss how to analyze colored Petri nets from this point of view. We will describe four analysis techniques: structural analysis [HGD08] to investigate the structural properties, model checking [CE81] to qualitatively analyze the behavior, numerical model checking [ASSB00] to accurately analyze the behavior and simulative model checking [DG08] to approximately analyze the behavior of a net. Please note that we will focus on how to use existing tools based on these analysis techniques to accomplish the analysis of colored Petri nets.

In addition, we can directly analyze colored Petri nets without unfolding. For this purpose, we do not develop our own analysis tools; instead we resort to external tools. In this chapter, we will discuss how to transform our colored qualitative Petri nets to those read by CPN tools [RWL⁺03] and then use the analysis capabilities offered by CPN tools to accomplish the analysis of our colored Petri nets.

This chapter is organized as follows. Section 4.1 describes structural analysis techniques of Petri nets and how to conduct structural analysis on colored Petri nets. Section 4.2 illustrates model checking of qualitative colored Petri nets. Section 4.3 describes numerical model checking of colored stochastic Petri nets. Section 4.4 discusses simulative model checking of colored stochastic Petri nets. Section 4.5 describes how to use CPN tools to analyze our colored qualitative Petri nets. Finally, the conclusions for this chapter are given.

4.1 Structural Analysis

Structural analysis [MBC⁺95], [HGD08] means to investigate the properties of Petri nets from their structure without constructing the reachability graph. If a property is

proved structurally for a Petri net, it holds for this Petri net in any initial marking. The important structural properties can be classified as follows:

1. *Elementary graph properties*, e.g. connectedness and strongly connectedness. All of them are decided only by the graph structure using graph algorithms. They can be used for preliminary checks of the design of a net.
2. *P- and T-invariants*. A P-invariant represents a set of places over which the weighted token count keeps constant, and a T-invariant describes how often the transitions contained by it have to fire to return to the original marking. Both of them can be obtained by solving a linear equation system that describes a net, which are also independent of the initial marking.
3. *Deadlock and trap*. A deadlock is a set of places where the set of its pretransitions is contained in the set of its posttransitions, while a trap is a set of places where the set of its posttransitions is contained in the set of its pretransitions. Both of them contribute to the study the liveness of Petri nets.

Among them, P- and T-invariants play crucial roles in analyzing biological systems due to their biological interpretations. For example, in a metabolic network, P-invariants correspond to the conservation law in chemistry, reflecting substrate conservations, and T-invariants reflect the steady state behavior [BCMS10], [HGD08].

Charlie is a software tool to analyze standard Petri nets, e.g. analyzing the structural properties given above. We also use Charlie to analyze colored Petri nets. In order to do that, we have to unfold them and then export them to a file in the APNN format, which will be inputed to Charlie to prove those properties. See [Fra09] for more details about how to use Charlie. We will give some case studies to demonstrate how to analyze structural properties of colored Petri nets in Chapter 5.

4.2 Model Checking

Model checking is a technique developed for the formal verification of hardware and software systems. It answers whether a finite state model of a system meets a given specification, expressed by temporal logics, the most popular two of which are Linear Temporal Logic (LTL) [Pnu81] and branching Computation Tree Logic (CTL) [CE81]. One of the big advantages of model checking is that it can be performed fully automatically.

As a discrete event modeling formalism, qualitative discrete Petri nets can be verified by model checking. In this section, we will discuss how to use existing model checking tools to analyze colored qualitative Petri nets. In the following, we first briefly recall the two popular temporal logics, LTL and CTL, and then discuss the way to check colored qualitative Petri nets.

4.2.1 Linear Temporal Logic

LTL [Pnu81] is a linear temporal logic, in which the future is seen as a sequence of states (a path) and each moment has a unique possible future (successor). The syntax of LTL is defined as follows:

$$\begin{aligned} \phi ::= & X\phi \mid G\phi \mid F\phi \mid \phi U \phi \\ & \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \rightarrow \phi \\ & \mid \text{true} \mid \text{false} \mid \text{ap}. \end{aligned}$$

The formulas of LTL are composed of atomic propositions $ap \in AP$, usual Boolean operators \neg (complement), \vee (disjunction), \wedge (conjunction), \rightarrow (implication), and temporal operators X (Next), G (Globally), F (Finally), and U (Until).

Suppose a Kripke model M . $\sigma = s_0, s_1, \dots$ denotes any infinite execution trace of M , $\sigma^i = s_i, s_{i+1}, \dots$ and \models denotes the satisfaction relation. The semantics of LTL is as follows:

$$\begin{aligned} M, \sigma \models \text{ap} & \Leftrightarrow \text{ap is true at } s_0 \text{ (written } s_0(\text{ap})) \\ M, \sigma \models \neg\phi & \Leftrightarrow \neg(M, \sigma \models \phi) \\ M, \sigma \models \phi_1 \vee \phi_2 & \Leftrightarrow M, \sigma \models \phi_1 \vee M, \sigma \models \phi_2 \\ M, \sigma \models \phi_1 \wedge \phi_2 & \Leftrightarrow M, \sigma \models \phi_1 \wedge M, \sigma \models \phi_2 \\ M, \sigma \models \phi_1 \rightarrow \phi_2 & \Leftrightarrow M, \sigma \models \neg\phi_1 \vee M, \sigma \models \phi_2 \\ M, \sigma \models X\phi & \Leftrightarrow M, \sigma^1 \models \phi \\ M, \sigma \models G\phi & \Leftrightarrow \forall i \geq 0 M, \sigma^i \models \phi \\ M, \sigma \models F\phi & \Leftrightarrow \exists i \geq 0 M, \sigma^i \models \phi \\ M, \sigma \models \phi_1 U \phi_2 & \Leftrightarrow \exists i \geq 0 M, \sigma^i \models \phi_2 \wedge \forall 0 \leq j < i M, \sigma^j \models \phi_1 \end{aligned}$$

The formulas of LTL are interpreted over a set of paths; a formula is true if and only if it is evaluated to true for all paths. See e.g. [CGP01] for detailed semantics.

4.2.2 Computation Temporal Logic

CTL [CE81] is a branching time logic, in which each moment has various possible futures in time. CTL also contains atomic propositions $ap \in AP$, usual Boolean operators \neg , \vee , \wedge and \rightarrow , and temporal operators, X , G , F , and U , but it has two more path quantifiers E (there exists a path) and A (for all paths). Each of the temporal operators must be immediately preceded by one of the two path quantifiers. The syntax of CTL is defined as follows:

$$\begin{aligned} \phi ::= & AX\phi \mid EX\phi \mid AG\phi \mid EG\phi \mid AF\phi \mid EF\phi \mid A[\phi U \phi] \mid E[\phi U \phi] \\ & \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \rightarrow \phi \\ & \mid \text{true} \mid \text{false} \mid \text{ap}. \end{aligned}$$

The semantics of CTL can be interpreted over Kripke models. Suppose a Kripke model M . s_i denotes a state of M , and $\pi = (s_i, s_{i+1}, \dots)$ denotes any path outgoing from s_i in the model M . The semantics of CTL is as follows:

$$\begin{aligned}
 M, s_i \models ap & \Leftrightarrow ap \text{ is true at } s_i \text{ (written } s_i(ap)) \\
 M, s_i \models \neg\phi & \Leftrightarrow \neg(M, s_i \models \phi) \\
 M, s_i \models \phi_1 \vee \phi_2 & \Leftrightarrow M, s_i \models \phi_1 \vee M, s_i \models \phi_2 \\
 M, s_i \models \phi_1 \wedge \phi_2 & \Leftrightarrow M, s_i \models \phi_1 \wedge M, s_i \models \phi_2 \\
 M, s_i \models \phi_1 \rightarrow \phi_2 & \Leftrightarrow M, s_i \models \neg\phi_1 \vee M, s_i \models \phi_2 \\
 M, s_i \models AX\phi & \Leftrightarrow \forall \pi = (s_i, s_{i+1}, \dots) M, s_{i+1} \models \phi \\
 M, s_i \models EX\phi & \Leftrightarrow \exists \pi = (s_i, s_{i+1}, \dots) M, s_{i+1} \models \phi \\
 M, s_i \models AF\phi & \Leftrightarrow \forall \pi = (s_i, s_{i+1}, \dots) \exists j \geq i M, s_j \models \phi \\
 M, s_i \models EF\phi & \Leftrightarrow \exists \pi = (s_i, s_{i+1}, \dots) \exists j \geq i M, s_j \models \phi \\
 M, s_i \models AG\phi & \Leftrightarrow \forall \pi = (s_i, s_{i+1}, \dots) \forall j \geq i M, s_j \models \phi \\
 M, s_i \models EG\phi & \Leftrightarrow \exists \pi = (s_i, s_{i+1}, \dots) \forall j \geq i M, s_j \models \phi \\
 M, s_i \models A[\phi_1 U \phi_2] & \Leftrightarrow \forall \pi = (s_i, s_{i+1}, \dots) \exists j \geq i M, s_j \models \phi_2 \wedge \forall i \leq k < j M, s_k \models \phi_1 \\
 M, s_i \models E[\phi_1 U \phi_2] & \Leftrightarrow \exists \pi = (s_i, s_{i+1}, \dots) \exists j \geq i M, s_j \models \phi_2 \wedge \forall i \leq k < j M, s_k \models \phi_1
 \end{aligned}$$

Compared with LTL, the formulas of CTL are interpreted over trees but not individual paths. See e.g. [CGP01] for more details.

4.2.3 Model Checking of QPN^c

We can use Charlie [Cha11] or Marcie [SRH11] to perform model checking for QPN^c . Charlie can conduct not only structural analysis but also explicit CTL/LTL model checking of Petri nets. Moreover, Marcie provides more efficient CTL model checking based on Interval Decision Diagrams (IDD). Either tool accepts an input in the APNN format.

The procedure to use either tool is the same. We first unfold and export a QPN^c model to a file in the APNN format. Then we can deal with it like dealing with a standard Petri net model.

Take Figure 1.2 as an example, we can write the following properties e.g. using CTL:

- $AG(G_a)$. It checks if it always holds that G_a (gene a) has tokens.
- $AG AF(P_a)$. It checks if P_a (protein a) has tokens infinitely often.

4.3 Numerical Model Checking

Stochastic Petri nets have gained widespread use in modeling biological systems with stochastic characteristics. The underlying semantics of a stochastic Petri net is described by a continuous time Markov chain, so it can be quantitatively analyzed by

numerical model checking based on the Continuous Stochastic Logic (CSL) [ASSB00], [BHHK03], which is very useful for the validation of biological systems.

Some tools for CSL model checking have been developed. For example, PRISM [KNP09] has been used to analyze probabilistic systems in a variety of application areas. Marcie [SRH11] particularly aims to quantitative analysis of generalized stochastic Petri nets.

In this section, we will investigate how to utilize those existing model checking tools to realize the verification and validation of colored stochastic Petri nets. For a \mathcal{SPN}^C model, we can exactly obtain its equivalent stochastic Petri net model by unfolding; hence we can realize the analysis of a \mathcal{SPN}^C model via its equivalent uncolored model using those tools. In the following, we first briefly recall CSL and then give the procedure for CSL model checking of \mathcal{SPN}^C .

4.3.1 Continuous Stochastic Logic

CSL is based on the temporal logics CTL [CES86] and PCTL (Probabilistic Computation Tree Logic) [HJ94]. CSL provides a powerful means to specify both path-based and state-based properties on a CTMC. CSL is composed of not only the standard propositional logic operators, but also two probabilistic operators, P and S , where the P operator checks the probability of a path formula while the S operator checks the steady-state behavior of a CTMC. The syntax of CSL [ASSB00], [BHHK03] is defined as follows:

$$\begin{aligned} \Phi & ::= P_{\trianglelefteq p}[\phi] \mid S_{\trianglelefteq p}[\Phi] \\ & ::= \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \Phi \rightarrow \Phi \\ & ::= \text{true} \mid \text{false} \mid \text{ap}. \\ \phi & ::= X^I\Phi \mid F^I\Phi \mid G^I\Phi \mid \Phi U^I\Phi. \end{aligned}$$

where $\text{ap} \in AP$ is an atomic proposition, $\trianglelefteq \in \{<, \leq, \geq, >\}$, $p \in [0, 1]$ and I is an interval of \mathbb{R}^+ .

CSL contains state formulas, Φ , interpreted over states of a CTMC, and path formulas, ϕ , interpreted over paths of a CTMC. It also has two probabilistic operators. $P_{\trianglelefteq p}[\phi]$ asserts that the probability of the path formula ϕ being satisfied from a given state meets the bound given by $\trianglelefteq p$, and $S_{\trianglelefteq p}[\Phi]$ indicates that the steady-probability of being in a state satisfying Φ meets the bound given by $\trianglelefteq p$.

We do not give the detailed semantics of CSL here because it occupies large space. See [ASSB00], [BHHK03] for more details.

4.3.2 CSL Model Checking of \mathcal{SPN}^C

We can use Marcie [SRH11] to realize CSL model checking of the uncolored stochastic Petri net of a colored stochastic Petri net. Marcie is a tool for qualitative and quantita-

tive analysis of generalized stochastic Petri nets with extended arcs. It realizes symbolic CSL model checking of stochastic Petri nets written in the APNN format.

The procedure to use Marcie for CSL model checking of \mathcal{SPN}^C is very simple. We first unfold and export a \mathcal{SPN}^C model to a file in the APNN format. Then we can deal with it just like dealing with a stochastic Petri net model. We will demonstrate this in Chapter 5.

We can also use PRISM [KNP09] to conduct CSL model checking on colored stochastic Petri nets. PRISM is a probabilistic model checker, which supports three types of probabilistic models, i.e. discrete time Markov chains, continuous time Markov chains and Markov decision processes and incorporates several temporal logics, e.g. CSL and PCTL. PRISM uses a simple state-based language to describe its models.

Please note that Marcie is more efficient than PRISM due to the following reasons [SH09]. Marcie is based on IDD (Interval decision diagrams), which is more efficient than MTBDD (Multi-terminal binary decision diagrams) that is used by PRISM. Moreover, Marcie supports multi-threaded analysis, while PRISM can not. From the test result given in [SH09], we can see that Marcie clearly outperforms PRISM, e.g. consuming much less time for constructing the state space of the same size, being able to construct much larger state space.

In order to use PRISM for colored stochastic Petri nets, we first have to export a \mathcal{SPN}^C model to a stochastic Petri net that can be transformed into a PRISM file in Snoopy. When we obtain the PRISM file for a \mathcal{SPN}^C model, we can use PRISM to open it and then conduct CSL model checking on it.

For example, for the model of circadian rhythms in Figure 2.5, we can write the following property using CSL:

$$P_{>p}[F_{[t,t]}Activator = 1].$$

It checks if the probability of the species *Activator* being 1 at some time instance t is greater than a specified value p .

4.4 Simulative Model Checking

Numerical model checking adopts numerical techniques for transient analysis of CTMC models [ASSB00], which is usually highly accurate. However, it suffers from the problem of state space explosion due to its intensive computation and is often limited to models with the Markovian behavior [YKNP06]. In contrast, simulative model checking follows the idea of Monte Carlo sampling and analyzes only a subset of the state space to obtain an approximate result. So it is not limited to specified formal models and not subject to the state space explosion; however it is less accurate [DG08].

In this section, we investigate how to use simulative model checking to analyze colored stochastic Petri nets. We first recall basic ideas of PLTLc, a probabilistic linear temporal logic with numerical constraints [DG08], and then discuss how to use it to analyze colored stochastic Petri nets.

4.4.1 Probabilistic Linear Temporal Logic with Numerical Constraints

PLTLc [DG08] is an extension of standard LTL [Pnu81] to a stochastic setting with a probability operator and a filter construct that defines the initial state of a property. The syntax of PLTLc is defined as follows:

$$\begin{array}{ll}
\Phi & ::= P_{\leq p}[\phi] \\
& \quad | P_{\leq p}[\phi\{ap\}]. \\
\phi & ::= X\phi \mid G\phi \mid F\phi \mid \phi U \phi \\
& \quad | \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \rightarrow \phi \\
& \quad | true \mid false \mid ap. \\
ap & ::= \neg ap \mid ap \vee ap \mid ap \wedge ap \mid ap \rightarrow ap \\
& \quad | value \textit{Lop} value. \\
\textit{Lop} & ::= = \mid \neq \mid \geq \mid > \mid < \mid \leq . \\
value & ::= value \textit{op} value \\
& \quad | variable \mid function \\
& \quad | Integer \mid Real. \\
function & ::= max(variable) \mid d(variable). \\
Aop & ::= + \mid - \mid * \mid /.
\end{array}$$

The atomic propositions in PLTLc are extended to include e.g. variables and functions, so we here also describe the syntax of atomic propositions, which we have omitted for LTL, CTL and CSL. The probabilistic operator $P_{\leq p}[\phi]$ is used to compare the probability of the property ϕ being true with a given value p . $P_{=?}$ is allowed to replace $P_{\leq p}$ to return the probability of a property ϕ being true. In $P_{\leq p}[\phi\{ap\}]$, ap is a filter construct that defines the initial state of a property ϕ , i.e. ϕ will be checked from the first state that ap is satisfied.

Variable is defined as the concentration of a species, which is dependent on the time, e.g. any place in a Petri net can be considered as a variable. *time* is a predefined variable to represent the time of states. The function *max* returns the maximum concentration value of a species in a simulation run and *d* returns the derivative of the concentration of a species at each time instance.

The semantics of PLTLc is defined over a finite set of finite linear traces of temporal behavior, coming from e.g. stochastic simulation runs. The temporal operators, X , G , F , and U follow the standard LTL semantics. For example, for $P_{\leq p}[\phi]$, each trace is

evaluated to a Boolean value in terms of a property ϕ being true in this trace and the probability of the property ϕ holding true in a set of traces is computed by the fraction of the set that is evaluated to true over the whole set.

No doubt, the number of the traces and the length of each trace affect the correctness of the result. So one way to improve the accuracy is to increase both of them.

4.4.2 PLTLc Model Checking of \mathcal{SPN}^c

We can use MC2 [DG08], a model checker by Monte Carlo sampling, for simulative PLTLc model checking. MC2 can read sets of simulation traces as generated by Snoopy and expects additionally a file with the temporal-logical formulas. So in order to use MC2 for model checking of colored stochastic Petri nets, we only need to run simulation to obtain simulation traces and feed them to MC2.

For a colored stochastic Petri net, its simulation is done on its automatically unfolded stochastic Petri net, so we can obtain traces of unfolded places directly through simulation. In addition, we can also obtain traces for colored places by summing up the traces of their corresponding unfolded places. For example, if we run the repressilator model in Figure 1.2, we can obtain a stochastic simulation result, which shows the traces of three unfolded places P_a , P_b and P_c (illustrated in Figure 4.1(a)) of the colored place P and also the trace of P by summing up the traces of its three unfolded places (illustrated in Figure 4.1(b)).

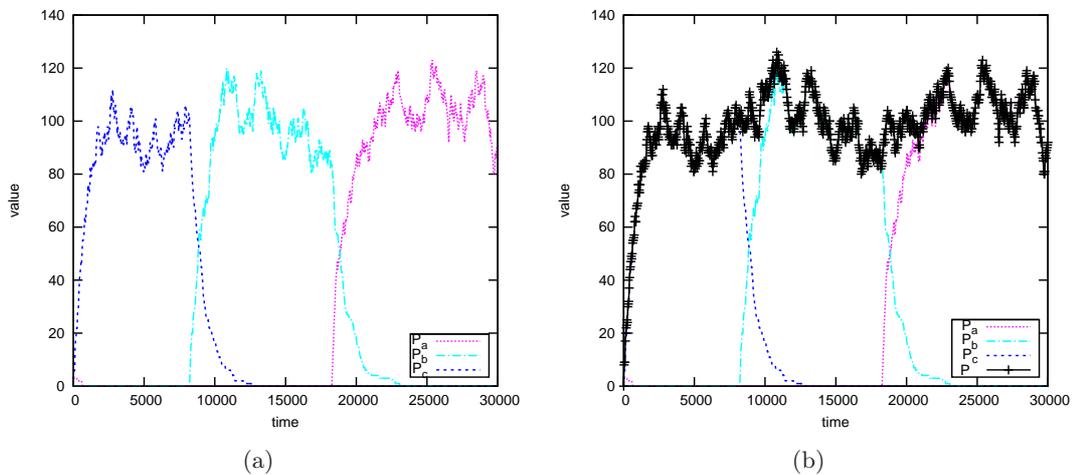


Figure 4.1: Stochastic simulation result of one simulation run for the repressilator model in Figure 1.2: (a) three unfolded places of colored place P , (b) colored place P and its three unfolded places.

Therefore, for a colored stochastic Petri net, we can specify its properties to be checked

not only for its unfolded places but also for its colored places. We use the repressilator example in Figure 1.2 to demonstrate how to specify properties using PLTLc and then obtain their results. For this purpose, we generate a set of 200 stochastic (single) simulation traces.

(1) Properties of colored places.

As described above, when we simulate a \mathcal{SPN}^c model, we can obtain the traces not only for unfolded places but also for colored places. Therefore, we can directly specify properties for colored places.

For example, we can use the following query to check the maximum on the trace of the colored place, P , which sums up the traces of its three unfolded places, P_a , P_b and P_c . The probability for this query is evaluated to 1.0.

$$P_{=?}[G([P] < 150)]$$

(2) Properties of unfolded places.

As described above, we can directly obtain unfolded places (colored place instances) from simulation, so we can also specify properties for unfolded places.

$$P_{=?}[F(\neg(G(P_a < 5) \vee (G(P_a > 110))))]$$

This query (the probability being evaluated to 0.9) states that the value of the P_a oscillates between two extremes: 5 and 110.

From these results, we can see that simulative model checking provides a quantification of the probabilities at which qualitative properties hold. Although the results are approximate, they do provide insights into how a system behaves, especially when the numerical model checking is not available.

In addition, Snoopy also provides simulative model checking of linear-time properties for \mathcal{SPN} and \mathcal{SPN}^c based on a subset of PLTLc [HRSS10]. For the time being, the formulas Snoopy supports take the following form:

$$P_{=?}[F_{[t_1, t_2]}[ap]].$$

The procedure to use simulative model checking of Snoopy is as follows. In Snoopy, before each simulation, we specify the properties to be checked, then run simulation and obtain the model checking result. Please note that we can also specify properties not only for unfolded places but also for colored places.

4.5 Analysis of QPN^c Using CPN Tools

A QPN^c model without special arcs exactly corresponds to a colored Petri net model drawn by CPN tools [RWL⁺03] (which we will call a CP-net in the following); therefore it is natural to transform a QPN^c model to a CP-net model and then utilize analysis capabilities offered by CPN tools.

In this section, we will address this problem. We first recall a brief introduction of CPN tools, then describe how to transform a QPN^c model to a CP-net model and finally discuss how to analyze QPN^c models using CPN tools.

4.5.1 CPN Tools

CPN tools [RWL⁺03] are a suite of tools for editing, simulating and analyzing CP-nets. They use the SML language for editing declarations and net inscriptions and provide several ways for simulation/animation, e.g. single-step simulation by firing one enabled transition or fast simulation by skipping a number of steps. CPN tools also facilitate generating and analyzing full or partial state spaces for CP-nets based on an explicit state enumeration and thus make it possible to verify different behavioral properties of CP-nets. Specifically, CPN tools not only provide a number of built-in standard queries to investigate standard properties of CP-nets but also allow to define your own queries using the SML language to investigate other interesting properties.

4.5.2 Transformation from QPN^c Models to CP-Net Models

In order to transform a QPN^c model to a CP-net model, we need first to build a map from QPN^c components to CP-net components, and then transform components of a QPN^c model to the counterparts of a CP-net model. The components to be transformed include:

- Declarations about color sets, variables, constants and user-defining functions,
- Place: name, identity, graphic information and initial marking,
- Transition: name, identity, graphic information and guard,
- Arc: source node, target node and expression, and
- General information: the size of the canvas for a net.

Among them, a key problem is to convert expressions of a QPN^c model to those of a CP-net model, as the syntax for these two kinds of expressions is different. For this, we have made automatic conversions as much as possible. For those parts that are not

automatically converted, we have to manually modify them so as to make them comply with the syntax of CPN tools.

CPN tools save all information of a model in an XML file and use a document type definition (DTD) file to validate the XML file. After the mapping above is built, we can convert a QPN^c model to a CP-net model by exporting to CPN tools in Snoopy, which in fact serializes a QPN^c file to a CP-net file readable by CPN tools. We can also transform a SPN^c/CPN^c model to a CP-net model by omitting all rate functions.

4.5.3 Analysis of QPN^c Models with CPN Tools

After obtaining a transformed CP-net model from a QPN^c model, we can analyze it using CPN tools. In the following, we use the example for cooperative ligand binding in Figure 4.2 (constructed according to [MWW11]) to illustrate how to use CPN tools to analyze a QPN^c model. When we export the example to a CP-net file in Snoopy, we obtain the model in CPN tools, illustrated in Figure 4.3, which exactly corresponds to the QPN^c model.

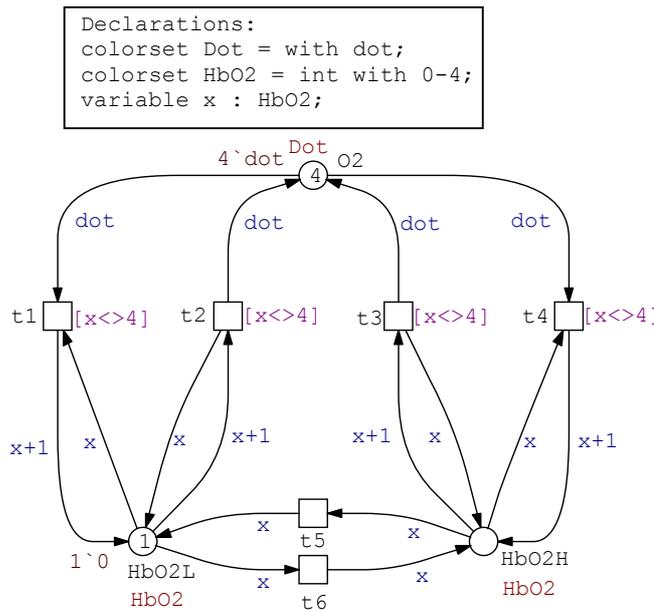


Figure 4.2: A colored Petri net model for cooperative ligand binding [MWW11].

When we generate the state space for this example in CPN tools, we can obtain a standard report that provides general information about the net. From this report, we will see that this net is bounded, live and reversible.

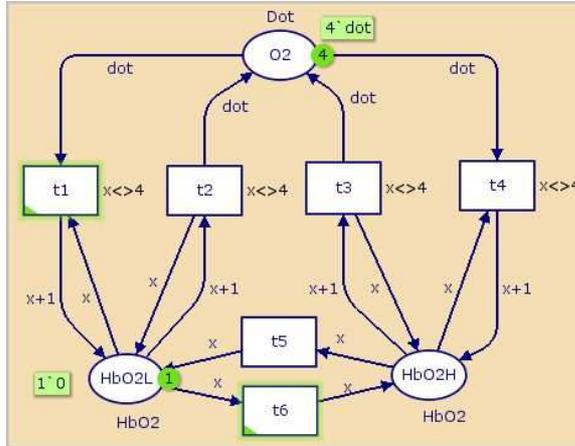


Figure 4.3: A CP-net model for Figure 4.2. Its declarations are the same as those in Figure 4.2.

We then demonstrate how to use standard queries to analyze the net as follows.

- *InitialHomeMarking()*: returning whether the initial marking is a home marking. It is evaluated to true for this net.
- *ListDeadMarkings()*: returning a list of all dead nodes. It is evaluated to empty for this net.

Although CPN tools offer an alternative way to directly analyze colored Petri nets, they have disadvantages in several aspects, e.g. unable to cope with special arcs and limited by a small state space.

4.6 Discussions

4.6.1 Comparison of Two Approaches: Folded versus Unfolded

Colored Petri nets can be analyzed at two levels: folded (colored) or unfolded. At the folded level, we can utilize the characteristics of colored Petri nets and do not generate their corresponding unfolded Petri nets while at the unfolded level we have to explicitly generate unfolded Petri nets for them. However, there are advantages and disadvantages for both approaches. In the following, we will in detail discuss and compare these two approaches from the point of view of simulation and analysis.

At the folded level, the computation of enabled transition instances for a colored Petri net has to be done for each firing (or in each simulation step) so as to move simulation on or to construct its reachability graph. Therefore, the repetitive computation of

enabled transition instances becomes a quite expensive task. So far, two tools, CPN tools [JK09] and GreatSPN [Gre11], have extensively explored simulation and analysis techniques at the folded level, both of which utilize the symmetries inherent to colored Petri nets.

In CPN tools [JK09], they can freely use the syntax of the SML language for constructing colored Petri nets, so they allow very complex but friendly expressions on guards or arcs. CPN tools provide simulation at the folded level, in which a pattern matching mechanism is used to compute enabled transition instances, but they do not make use of the symmetries of colored Petri nets. Moreover, CPN tools offer state space analysis at the folded level. To do this, they try to find the symmetries in colored Petri nets and construct reduced state space, occurrence graphs with equivalence classes (OE graphs) [Jen95], [Jen96]. However, so far CPN tools are still limited to a small state space.

In GreatSPN [CFG92], [CDFH97], [Gre11] a number of restrictions have been added on the syntax of colored Petri nets (which are called well-formed nets (SWN)). This restrained syntax brings several benefits, e.g. easily finding symmetries of a colored net only from its color sets and improving the efficiency of the computation of enabled transition instances. However the disadvantages are also obvious, e.g. not easy to write complex expressions, particularly guards. GreatSPN provides symbolic simulation of SWN, where the efficiency of the computation of enabled transition instances is gained due to the restrained syntax of SWN and another great efficiency improvement is gotten by substantially reducing the length of the event list using symbolic transition instances that result from the symmetries of SWN. GreatSPN allows to construct reduced state space, symbolic reachability graphs, where it also uses the restrained syntax of SWN to improve the efficiency of the computation of enabled transition instances and uses symmetries to greatly reduce the size of state space. The steady state and transient Markovian analysis can be done by converting symbolic reachability graphs into lumped Markov chains in GreatSPN.

In summary, at the folded level, we can utilize the symmetries inherent to colored Petri nets; however we usually have to develop specific simulation algorithms and analysis techniques for colored Petri nets with a specific kind of syntax.

In contrast, at the unfolded level, we have to pay the price to obtain an unfolded Petri net for a colored Petri net and sacrifice its intrinsic symmetry. However, the benefits we obtain are also great. When we obtain unfolded Petri nets, we can reuse all the simulation and analysis techniques applicable for standard Petri nets. So there are usually much richer simulation algorithms and analysis techniques available at the unfolded level than at the folded level.

Therefore in the future there are at least two ways for us to further strengthen analysis capabilities of our colored Petri nets, i.e. to export our colored Petri nets to GreatSPN or CPN tools that is already done in Section 4.5 or to equip our modeling tool with simulation and analysis techniques at the folded level. In addition, at the unfolded

level, we will continue to improve the unfolding efficiency of our colored Petri nets.

4.6.2 Partial Unfolding - Tackling Dynamic Color Sets

In biological systems there are usually such biological phenomena as compartment creation, division, merging and dissolving, or cell division and death. When we use colored Petri nets to model these phenomena, one way is to use dynamic color sets, i.e. to simulate these phenomena by adding or removing colors. There are basically two approaches to implement the simulation and analysis under dynamic color sets: at the folded or unfolded level.

At the folded level, it seems easier to introduce dynamic color sets. We can implement simulation by for each firing checking if dynamic color sets change or not and then computing enabled transition instances of colored Petri nets. However, there are still many challenges, e.g. when some color set changes, we have to recompute its permutation or rotation and then change everything concerning the permutation or rotation in the symbolic simulation [CFG92], [CDFH97].

We can also introduce dynamic color sets and then simulate or analyze colored Petri nets by partial unfolding. For this, we have to track all the transitions that result in the change of dynamic color sets. We can start with a full unfolding of a color Petri net. When a transition resulting in the change of a color set fires, we will pause the simulation and immediately do a partial unfolding for the newly added or removed colors, which will be fed to the simulation, and then resume the simulation. In the next step, we will focus on the partial unfolding way to tackle dynamic color sets.

4.7 Closing Remarks

Formal analysis techniques play crucial roles in verifying and validating constructed models. One of the advantages of Petri nets is that they have rich mathematically founded analysis techniques, covering both structural and behavioral properties.

In this chapter, we have discussed analysis techniques for colored Petri nets from two points of view. From the unfolding point of view, we have described how to employ existing techniques, structural techniques, numerical and simulative model checking, and tools for standard Petri nets to realize the analysis of colored Petri nets. From the folding point of view, we have transformed our qualitative Petri nets to those read by CPN tools and then analyze them using CPN tools. All these analysis techniques together provide an effective means to verify and validate colored Petri nets. In the future, we will investigate more analysis techniques suitable for colored Petri nets. In the next chapter, we will give some case studies in which we will show how these analysis techniques work for colored Petri nets.

5 Case Studies

So far we have introduced modeling and analysis techniques of colored Petri nets. In this chapter, we will present three case studies to demonstrate how to apply these techniques to modeling and analyzing practical biological systems.

The first two case studies demonstrate how colored Petri nets can be used to solve some of multiscale challenges introduced in Chapter 1, e.g. repetition of components, organization of components, communication of components, and further hierarchical organization of components. In the third case study, we want to further explore advanced applications of colored Petri nets, i.e. converting existing modeling paradigms discussed in the literature to colored Petri nets.

The first case study illustrates how to model *C. elegans* vulval development, in which there are six cells of identical structure. This system has such features as *repetition of cells* that are *organized in one dimensional space*, and *communicate between immediate neighbors*. Thus this case study will address all these three features (challenges). Other purposes of this case study are mainly twofold. (1) In Chapter 2, we have presented a colored Petri net framework for modeling and analyzing biological systems, which relates three modeling paradigms: QPN^C , SPN^C and CPN^C that can be converted into each other in our modeling tool Snoopy, thus we can explore a system from three perspectives: qualitative, stochastic and continuous, respectively. This case study will demonstrate how these three formalisms are combined to accomplish the analysis of a biological system. (2) Colored Petri nets enjoy a large variety of analysis techniques, e.g. animation, stochastic and continuous simulation, structural analysis and model checking. This case study will also show how these techniques are used to analyze a biological system.

The second case study models coupled Ca^{2+} channels, which addresses the problems of *repetition of Ca^{2+} channels*, *the organization of Ca^{2+} channels in two dimensional space* and *the communication of Ca^{2+} channels*. We also focus on how to model coupled Ca^{2+} channels with different levels of details: two-state and six-state models. In addition, we briefly discuss *the hierarchical organization* of coupled Ca^{2+} channels with more than one clusters in two dimensional space.

The third case study explores a more advanced application of colored Petri nets, i.e. modeling membrane systems with colored Petri nets. The purpose of this case study is trying to discover more special scenarios to apply colored Petri nets. A membrane system composed of compartments can be modeled as a colored Petri net model by

encoding each compartment as a color. As a result, we not only distinguish and show compartment information using colored Petri nets but also make the model very compact and offer a large range of analysis techniques.

This chapter is organized as follows. Section 5.1 describes how to model and analyze *C. elegans* vulval development. Section 5.2 discusses the modeling of Ca^{2+} channels. Section 5.3 addresses an advanced application, modeling membrane systems with colored Petri nets. Finally, the conclusions of this chapter are given.

5.1 Modeling *C. Elegans* Vulval Development

In this section, we illustrate our approach by presenting a case study about *C. elegans* vulval development. Based on the model given in [LNUM09], we develop a colored Petri net model for *C. elegans* vulval development. This model is then validated by comparing its behavior with that reported in [LNUM09].

This section is further organized as follows. Section 5.1.1 briefly recalls the background of *C. elegans* vulval development. Section 5.1.2 discusses how to model *C. elegans* vulval development using colored Petri nets. Section 5.1.3 gives the structural analysis of the *C. elegans* vulval development model. Section 5.1.4 describes how to use simulative model checking to determine the fate of vulval precursor cells. Section 5.1.5 gives results and discussions. Section 5.1.6 concludes this section.

5.1.1 *C. Elegans* Vulval Development

Caenorhabditis elegans (*C. elegans*) vulval development is an important paradigm to study how multiple pathways in multiple cells interact to produce developmental patterns. It involves six vulval precursor cells (VPCs) that are consecutively numbered P3.p to P8.p (see Figure 5.1). Each VPC is potentially capable of adopting one of three cell fates (termed 1°, 2° or 3°) depending upon the interaction of two opposing signals that it receives. Vulval development is initiated by an EGF-like soluble factor, LIN-3 that is produced by a centrally positioned anchor cell (AC). LIN-3 activates the EGFR homolog LET-23 in the VPCs, and specifies the 1° cell fate by means of a canonical Ras-MAPK cascade. In response to the inductive signal, the VPCs produce LIN-12-mediated lateral signals, which counteract the inductive signals in the neighboring VPCs. The lateral signal activates the receptor LIN-12/Notch in the neighboring VPCs, causing them to adopt the 2° fate by means of the LIN-12/Notch lateral signaling pathway. The fate of each VPC is decided by its relative distance to AC and the lateral signals it obtains from its neighboring VPCs. As a result, in the wild-type *C. elegans* the cell closest to the AC (P6.p) adopts the 1° cell fate by receiving most of the inductive signals. The two neighboring cells (P5.p and P7p) of it adopt the 2° cell fate by receiving the stronger lateral signals from P6.p. The other cells (P3.p, P4.p and

P8.p) adopt the 3° fate without enough inductive signals or lateral signals [YBG04], [FPHH07].

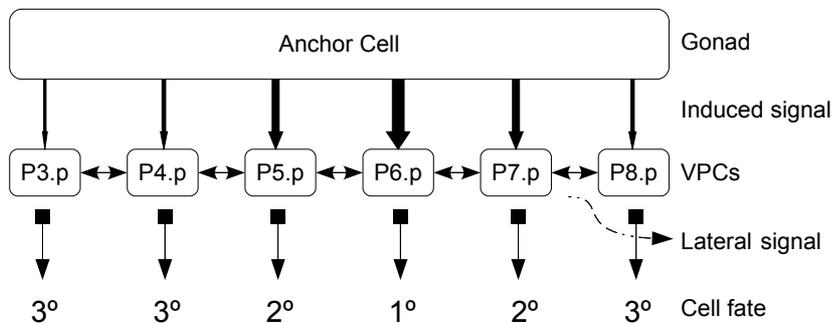


Figure 5.1: Spatial patterning of VPCs of vulval development in the wild-type *C. elegans*. The AC in the gonad releases the inductive signal, in response to which the VPCs produce the lateral signal, causing P3.p – P8.p cells to adopt three distinct cell fates (1°, 2° and 3°).

In order to understand the processes involved in multi-cellular pattern formation, a lot of computational models have been developed for *C. elegans* vulval development. For example, Sternberg and Horvitz [SH89] proposed the first diagrammatic model, describing the regulatory network underlying VPC determination. Fisher et al. used state charts to create a formal dynamic model of vulval fate specification [FPHH07]. Giurumescu et al. [GSA06] proposed a partial model based on ordinary differential equations. Li et al. [LNUM09] recently modeled *C. elegans* vulval development using hybrid functional Petri nets with extensions. Bonzanni et al. [BKF⁺09] used the basic Petri net formalism to build a coarse-grained model for simulating the behavior of *C. elegans* vulval development. Among them, the latter two seem to be more detailed and quantitative, but both of them become less readable and manageable.

5.1.2 Modeling

In this section, we use colored Petri nets to model *C. elegans* vulval development based on the hybrid functional Petri net model developed in [LNUM09]. As *C. elegans* vulval development involves six similar vulval precursor cells, it is natural to use colored Petri nets to model it, i.e. defining six colors to differentiate six cells. As a result, we can use only one-cell model to represent six cells. This shows one attractive advantage of colored Petri nets, representing the whole system with only one component, thus greatly decreasing the size of the whole model.

Before giving a colored model, we first briefly recall the hybrid functional Petri net

model for *C. elegans* vulval development reported in [LNUM09]. This model is composed of six vulval precursor cells, each of which contains a detailed Ras-MAPK pathway, a LIN-12/Notch lateral signaling pathway and a signaling pathway induced from hyp7 (the hypodermal syncytium). In this model, there are 427 entities (places), 554 processes (transitions) and 780 connectors (arcs). They use two biological fate determination rules based on temporal order and temporal interval, respectively, to determine the fate of cells. See [LNUM09] for more details about this model.

In order to construct a colored stochastic Petri net model, we first construct a stochastic Petri net model for one of the six VPCs and the anchor cell. Then we define a color set with six colors, e.g. an integer color set, CS , with values 3 to 8 encoding six VPCs and a second color set Dot with only one color "dot" encoding the anchor cell. After that, we assign CS to the constructed one-cell model, and thus we obtain one copy for each of the six VPCs. Besides, we assign the color set Dot to the anchor cell. Finally, we obtain a colored stochastic Petri net model for *C. elegans* vulval development, illustrated in Figure 5.2.

In order to easily understand our model and compare it with the original model in [LNUM09], we give our model a similar structure as one cell of the model in [LNUM09], and all rate functions and parameters are set in terms of [LNUM09]. Instead of using implicit user-defined functions, we use scheduled/deterministic transitions ($Tsim1$ – $Tsim3$) to add an initial signal to AC and switch on the production of signals of AC and hyp7. Table 5.1 and 5.2 give explanations about those transitions and places that are different from those in [LNUM09], respectively. Other transitions and places have the same meaning as those in [LNUM09]. All arc multiplicities have been marked in the model. Please note that we use modifier arcs for the arc from place $Pe1$ to transition $Te1$ and the arc from place $Pe2$ to transition $Te4$, as $Pe1$ and $Pe2$ are only used to control the rate of $Te1$ and $Te4$, respectively.

We use the combination of inhibitor arcs, read arcs and logic places to accomplish the switch of different genotypes. At the top of the model, we can see a genotype configuration-like panel, which includes an AC and four mutant variables. AC can toggle between true and false; $lin12$ can toggle between "wt", "ko" and "gf" (by selecting $lin12_wt$, $lin12_ko$ and $lin12_gf$, respectively), indicating three genetic conditions of wild, knockout and overexpression of the gene $lin-12$; $lin15$, vul and lst can toggle between "wt" and "ko".

The colored Petri net model comprises 41 places, 68 transitions and 131 arcs. Compared with the model in [LNUM09], we can see that the size of the colored Petri net model substantially decreases by folding similar cells to one cell. Therefore, for such models as *C. elegans* vulval development with some similar components, it is a very convenient and compact way to model them with colored Petri nets in a visual way.

Table 5.1: Descriptions of some transitions in the \mathcal{SPN}^C model of *C. elegans* vulval development. The values of *high*, *medium* and *low* are 100, 1 and 0.01, respectively. All transitions not given here correspond exactly to those given in [LNUM09].

| Transition | Rate function | Description |
|------------|--|---|
| Tsim1 | (400,0,400) | Add an initial value, 100, to LIN3_AC at simulation time point 400 and switch on ts1 |
| Tsim2 | (450,0,450)/(500,0,500) | Switch on transition t1 for cell 5 and 7 at time point 450, and for cell 3,4 and 8 at 500 |
| Tsim3 | rand(200,400) | Switch on transition ts7 at a random time between 200 and 400 for each cell |
| t1 | MassAction(0.1*low)/ MassAction(0.1*medium) | for cells 3,4,8 and 5,7, respectively |
| t2 | MassAction(0.1*low)/ MassAction(0.1*medium) | for cells 3,4,8 and 5,7, respectively |
| t3 | MassAction(0.1*high) | only for cell 6 |
| ts8 | 1 | If lin12 equals "wt", it can be fired |
| ts9 | 0.2 | If lin12 equals "gf", it can be fired |
| t18 | MassAction(0.1) | If lin12 equals "gf" and lst equals "wt", it can be fired |
| t19 | MassAction(0.1) | If lin12 equals "gf" and lst equals "ko", it can be fired |
| Te1 | MassAction(0.04*Pe1) | Production of lst_protein |
| Te2 | MassAction(0.06) | Degradation of Pe1 |
| Te3 | MassAction(0.011) | Degradation of Pe2 |
| Te4 | MassAction(0.6*Pe2) | Production of lst_protein |

Table 5.2: Descriptions of some places in the \mathcal{SPN}^C model of *C. elegans* vulval development. All places not given here correspond exactly to those given in [LNUM09].

| Place | Initial value | Description |
|-------------------------|---------------|--------------------------------|
| AC | 1 | Switch on/off AC |
| lin12_wt | 1 | Switch on/off "wt" |
| lin12_ko | 0 | Switch on/off "ko" |
| lin12_gf | 0 | Switch on/off "gf" |
| vul | 1 | Switch on/off vul |
| lin15 | 1 | Switch on/off lin15 |
| lst | 1 | Switch on/off lst |
| Steady_state_time_gonad | 0 | Switch on/off transition ts1 |
| Steady_state_time_VPC | 0 | Switch on/off transition t1 |
| Steady_state_time_hyp7 | 0 | Switch on/off transition ts7 |
| Psim3 | 1 | Switch on/off transition Tsim3 |
| Pe1 | 0 | Production of lst_protein |
| Pe2 | 0 | Production of lst_protein |

5.1.3 Structural Analysis

In order to increase our confidence in the constructed model, we will validate it by structural analysis. As this model has inhibitor arcs, we have to ignore them during structural analysis; as a result, the whole net breaks down into disconnected parts. The two main parts are the MAPK pathways of all six VPCs connected by the logic place *vul*, and the LIN-12/Notch lateral signaling pathways of all six VPCs connected by the logic places *lin12_wt*, *lin12_gf* and *lst* and the transitions *t14* and *t15*. For these two parts we perform structural analysis, respectively, after we unfold them to standard Petri nets, which are inputs of the analysis tool Charlie [Fra09]. Here we exploit two of the fundamental behavioral properties, P- and T-invariants.

The MAPK pathways of six VPCs as a whole are not covered by P-invariants, but are covered by 132 minimal semipositive T-invariants, i.e. each VPC has 22 T-invariants (see Table 5.3). The LIN-12/Notch lateral signaling pathways of six VPCs as a whole are not covered by P-invariants, but are covered by 130 minimal semipositive T-invariants, specifically 22 T-invariants for VPC 4 – VPC 7 and 21 T-invariants for VPC 3 and 8, respectively. For example, Table 5.4 gives the T-Invariants of VPC 4 and other VPCs have similar T-invariants. All these T-Invariants have their biological meaning.

Table 5.3: The minimal T-invariants in the MAPK pathway of each VPC.

| No. | Transitions | No. | Transitions |
|-----|----------------------|-----|-------------------|
| 1 | t11,td18 | 12 | td20,t12,t13 |
| 2 | ts2,td3 | 13 | td11,ts5 |
| 3 | td6,t6,ts2,ts7,t4,t5 | 14 | ts5,t8,td12 |
| 4 | td10,ts4,t7 | 15 | ts5,t8,t26,td13 |
| 5 | ts6,td14 | 16 | ts3,td7 |
| 6 | td15,ts6,td16,t9 | 17 | td25,ts3,td8 |
| 7 | t12,td19 | 18 | t10,td17 |
| 8 | td9,ts4 | 19 | td4,ts2,t1,t2 |
| 9 | t1,td2 | 20 | td4,t6,ts2,ts7 |
| 10 | td22,ts7 | 21 | td5,ts2,t4,t2,t1 |
| 11 | td6,ts2,t1,t2,t4,t5 | 22 | td5,ts2,ts7,t6,t4 |

Table 5.4: The minimal T-invariants in the LIN-12/Notch lateral signaling pathway of VPC 4. Please note that the transitions with the suffix "_3" (or "_5") belong to VPC 3 (or 5), and other transitions belong to VPC 4. This is because there are connections between VPC 4 and VPC 3 (or VPC 5).

| No. | Transitions | No. | Transitions |
|-----|------------------|-----|---------------------------------------|
| 1 | td30,ts10 | 12 | td25,t17,t23,t20,td27,ts8,t14,t13_3 |
| 2 | t13,td20 | 13 | td25,t17,t23,t20,td27,ts8,t16,t13 |
| 3 | td24,t23,t13 | 14 | td25,td26,t17,t23,ts8,t14,t13_3 |
| 4 | t22,Te3 | 15 | td25,t17,t23,t20,td27,ts8,t15,t13_5 |
| 5 | td31,t21,td29 | 16 | td25,td26,t17,t23,ts8,t16,t13 |
| 6 | td23,ts8 | 17 | td25,td26,t17,t23,ts9,t14,t13_3 |
| 7 | td23,ts9 | 18 | td25,td26,t17,t23,t16,ts9,t13 |
| 8 | td29,ts8,t19 | 19 | td25,td26,t17,t23,t15,ts9,t13_5 |
| 9 | t18,td29,ts8,Te2 | 20 | td25,t17,t23,t20,td27,t16,ts9,t13 |
| 10 | td29,ts9,t19 | 21 | td25,td26,t17,t23,ts8,t15,t13_5 |
| 11 | t18,td29,ts9,Te2 | 22 | td25,t17,t20,td27,ts9,t14,t13_3,t23_3 |

5.1.4 Determining the Fate of VPCs Using Simulative Model Checking

As our model is a stochastic one, whose average behavior of multiple runs approximates the behavior of its corresponding continuous model, we can adapt Rule I in [LNUM09] to determine the fate of VPCs. Here we name the adapted rule as the fate determination rule in average setting (shortly Rule A). On the other hand, we can also transform the fate determination rule in average setting to its counterpart of the stochastic setting, which we name the fate determination rule in stochastic setting (shortly Rule S). By analyzing the probabilities of simulation runs satisfying the rule, we can obtain the fate of each VPC. In the following, we will describe how to implement these two rules using simulative model checking.

(1) Implementation of fate determination rule in average setting.

Rule A works like Rule I in [LNUM09]. In the case of *lin12* being "wt" or "ko", the fate is determined in terms of the following criteria. If the concentrations of both *Fate_1* and *Fate_2* (two places) are not less than the respective threshold values (*threshold1* and *threshold2*), and keep these states for given time periods, then 1° (or 2°) fate will be adopted if *Fate_1* (or *Fate_2*) first exceeds its threshold. If only the concentration of *Fate_1* (or *Fate_2*) is kept over its threshold for the given time period, then 1° (or 2°) fate will be adopted. Otherwise, 3° will be adopted.

In the case of *lin12* being "gf", the fate is determined in terms of the following criteria. If the concentration of *Fate_1* is not less than its threshold value (*threshold1*), and keeps this state for the given time period, then 1° fate will be adopted. If the concentration of *Fate_2* is not less than its threshold value (*threshold2*), and keeps this state for the given time period, but the concentration of *Fate_1* decreases below its threshold value (*threshold1*) during the given time period, then 2° fate will be adopted. Otherwise, 3° will be adopted.

We use PLTLc [DG08] to implement these rules. To do that, we have to formalize these rules that lead to the corresponding VPC pattern as queries. After that, we can use these queries to formally check whether the model reproduces the expected fate patterns.

In the case of *lin12* being "wt" or "ko", the queries for the fate determination are as follows (*threshold1* adops 0.69, and *threshold2* adops 0.06):

$$P=?[F(Fate_1 \geq threshold1 \ \& \ !G(Fate_2 \geq threshold2) \ \& \ G(Fate_1 \geq threshold1))] \quad (5.1)$$

which reads: what is the probability of the following assertion: eventually *Fate_1* is greater than or equal to *threshold1*, and from now on *Fate_2* is not always greater than or equal to *threshold2*, but *Fate_1* remains always greater than or equal to

threshold1.

$$P_{=?}[F(\textit{Fate_2} \geq \textit{threshold2} \ \& \ !G(\textit{Fate_1} \geq \textit{threshold1}) \ \& \ G(\textit{Fate_2} \geq \textit{threshold2}))] \quad (5.2)$$

which reads: what is the probability of the following assertion: eventually *Fate_2* is greater than or equal to *threshold2*, and from now on *Fate_1* is not always greater than or equal to *threshold1*, but *Fate_2* remains always greater than or equal to *threshold2*.

If query 5.1 (or 5.2) is evaluated to true, then 1° (or 2°) will be adopted. Otherwise, 3° will be adopted.

In the case of *lin12* being "gf", the queries for the fate determination are as follows (*threshold1* adops 0.65, and *threshold2* adops 0.35):

$$P_{=?}[F(\textit{Fate_1} \geq \textit{threshold1} \ \& \ G(\textit{Fate_1} \geq \textit{threshold1}))] \quad (5.3)$$

which reads: what is the probability of the following assertion: eventually *Fate_1* is always greater than or equal to *threshold1*.

$$P_{=?}[F(\textit{Fate_2} \geq \textit{threshold2} \ \& \ G(\textit{Fate_2} \geq \textit{threshold2})) \ \& \ G(\textit{Fate_1} \geq \textit{threshold1} \ \rightarrow \ F(\textit{Fate_1} < \textit{threshold1}))] \quad (5.4)$$

which reads: what is the probability of the following assertion: eventually *Fate_2* is greater than or equal to *threshold2*, and from now on *Fate_2* remains always greater than or equal to *threshold2*, but always *Fate_1* being greater than or equal to *threshold1* implies *Fate_1* being less than *threshold1*.

If query 5.3 (or 5.4) is evaluated to true, then 1° (or 2°) will be adopted. Otherwise, 3° will be adopted.

For Rule A, we have to use the average behavior of multiple simulation runs to determine the fate of cells. These queries above describe the fate determination according to the average behavior. We recast these queries to PLTLc and perform model checking using MC2 fed with simulation traces produced by Snoopy.

(2) Implementation of fate determination rule in stochastic setting.

As seen above, Rule S works on each simulation run of a stochastic model. By analyzing the probability of simulation runs satisfying the rule, we can obtain the fate of each VPC.

Rule S works as follows. For a given time period when the average behavior of multiple simulation runs is in a steady state, for each simulation run, if its average value exceeds

its threshold (the same as those in Rule A), then the evaluation of this run is true. Therefore for multiple simulation runs, we will obtain a probability of simulation runs being evaluated to true. According to the predefined probabilistic thresholds, we can determine the fate of each cell.

Using PLTLc, the queries for Rule S can be written as follows. In the case of *lin12* being "wt"/"ko" ("gf"), *threshold1* adopts 0.69 (0.65), and *threshold2* adopts 0.06 (0.35).

$$P_{=?}[time > 1000 \rightarrow average([Fate_1]) \geq threshold1] \quad (5.5)$$

which reads: what is the probability of the following assertion: when the simulation time is greater than 1000, the average value of *Fate_1* is greater than or equal to *threshold1*.

$$P_{=?}[time > 1000 \rightarrow average([Fate_2]) \geq threshold2] \quad (5.6)$$

which reads: what is the probability of the following assertion: when the simulation time is greater than 1000, the average value of *Fate_2* is greater than or equal to *threshold2*.

If the probability for query 5.5 exceeds 0.55, then 1° is adopted. If the probability for query 5.5 does not exceed 0.55, but the probability for query 5.6 exceeds 0.55, then 2° is adopted. Otherwise 3° is adopted.

As our model is a stochastic one, one has to determine the required amount of simulation runs to achieve an appropriate accuracy of the results. We use the idea of the confidence interval as described in [SM08]. The confidence interval contains the property of interest with some predefined probability, called confidence level. This confidence level has usually values of 90%, 95%, or 99%. Choosing the confidence of 95% and the accuracy of 10^{-2} , we perform 50 simulation experiment for each genotype; for each simulation experiment we need to perform 38,000 stochastic simulation runs. We then apply the rule above to the average behavior of these runs. Besides, we choose the same threshold values for the rules above as those in [LNUM09].

5.1.5 Results and Discussions

To determine the capability of our colored Petri net model to reproduce and predict the biological behavior, we simulate 48 different experimental conditions (genotypes), which have been used in [LNUM09]. In order to keep this thesis self-contained, we repeat these 48 genotypes, illustrated in Table 5.5.

In terms of the experimental setting above, we first perform simulation runs for 44 stable genotypes, which turns out that our model reliably reproduces all these 44 stable genotypes by using either Rule A or Rule S. We here only give simulation plots for the genotype where all genes are wild, each of which is an average behavior of 380,000

Table 5.5: Fate patterns to be validated, excerpted from [LNUM09]. In the AC column, +/- stands for anchor cell formed/ablated. In the Genotypes column, "wt", "ko" and "gf" represent wild-type, knockout and overexpression, respectively. In the Fate Patterns column, 1, 2 and 3 indicate 1°, 2°, and 3°, respectively. For the unstable patterns, each cell adopts either 1° or 2°, which we will discuss in depth.

| No. | AC | Genotypes | | | Patterns | No. | AC | Genotypes | | | Patterns | | |
|-----|----|-----------|-------|-----|----------|----------|----|-------------|-------|-------|----------|-----|-------------|
| | | lin12 | lin15 | vul | lst | | | 3,4,5,6,7,8 | lin12 | lin15 | vul | lst | 3,4,5,6,7,8 |
| 1 | + | wt | wt | wt | wt | [332123] | 25 | - | wt | wt | wt | wt | [333333] |
| 2 | + | wt | wt | wt | ko | [331113] | 26 | - | wt | wt | wt | ko | [333333] |
| 3 | + | wt | wt | ko | wt | [333333] | 27 | - | wt | wt | ko | wt | [333333] |
| 4 | + | wt | wt | ko | ko | [333333] | 28 | - | wt | wt | ko | ko | [333333] |
| 5 | + | wt | ko | wt | wt | Unstable | 29 | - | wt | ko | wt | wt | Unstable |
| 6 | + | wt | ko | wt | ko | [111111] | 30 | - | wt | ko | wt | ko | [111111] |
| 7 | + | wt | ko | ko | wt | [333333] | 31 | - | wt | ko | ko | wt | [333333] |
| 8 | + | wt | ko | ko | ko | [333333] | 32 | - | wt | ko | ko | ko | [333333] |
| 9 | + | ko | wt | wt | wt | [331113] | 33 | - | ko | wt | wt | wt | [333333] |
| 10 | + | ko | wt | wt | ko | [331113] | 34 | - | ko | wt | wt | ko | [333333] |
| 11 | + | ko | wt | ko | wt | [333333] | 35 | - | ko | wt | ko | wt | [333333] |
| 12 | + | ko | wt | ko | ko | [333333] | 36 | - | ko | wt | ko | ko | [333333] |
| 13 | + | ko | ko | wt | wt | [111111] | 37 | - | ko | ko | wt | wt | [111111] |
| 14 | + | ko | ko | wt | ko | [111111] | 38 | - | ko | ko | wt | ko | [111111] |
| 15 | + | ko | ko | ko | wt | [333333] | 39 | - | ko | ko | ko | wt | [333333] |
| 16 | + | ko | ko | ko | ko | [333333] | 40 | - | ko | ko | ko | ko | [333333] |
| 17 | + | gf | wt | wt | wt | [222122] | 41 | - | gf | wt | wt | wt | [222222] |
| 18 | + | gf | wt | wt | ko | [221112] | 42 | - | gf | wt | wt | ko | [222222] |
| 19 | + | gf | wt | ko | wt | [222222] | 43 | - | gf | wt | ko | wt | [222222] |
| 20 | + | gf | wt | ko | ko | [222222] | 44 | - | gf | wt | ko | ko | [222222] |
| 21 | + | gf | ko | wt | wt | Unstable | 45 | - | gf | ko | wt | wt | Unstable |
| 22 | + | gf | ko | wt | ko | [111111] | 46 | - | gf | ko | wt | ko | [111111] |
| 23 | + | gf | ko | ko | wt | [222222] | 47 | - | gf | ko | ko | wt | [222222] |
| 24 | + | gf | ko | ko | ko | [222222] | 48 | - | gf | ko | ko | ko | [222222] |

simulation runs, illustrated in Figure 5.3 – 5.8. To support their comparison, we use the same scaling in all plots.

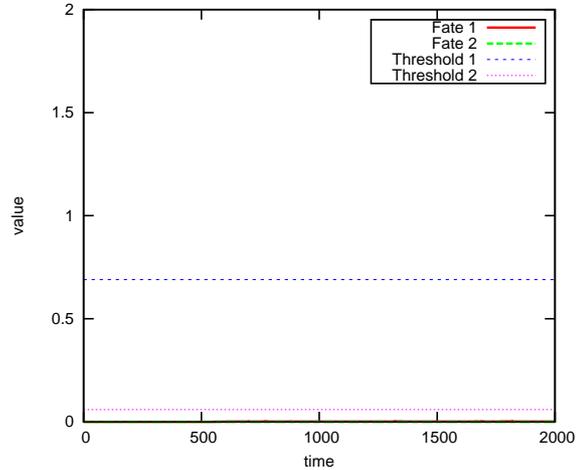


Figure 5.3: Stochastic simulation result averaged over 38,000 runs for VPC 3.

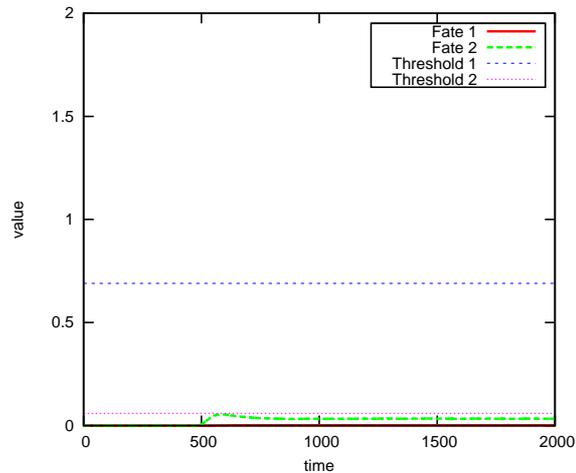


Figure 5.4: Stochastic simulation result averaged over 38,000 runs for VPC 4.

We notice for 4 unstable patterns the time when we switch on the production of hyp7 directly affects the fate each cell will adopt. Therefore, we use a random function to generate different switch times to produce hyp7 so as to obtain different fates. Table 5.6 gives detailed statistical results of these 4 unstable patterns by using Rule A and Rule S.

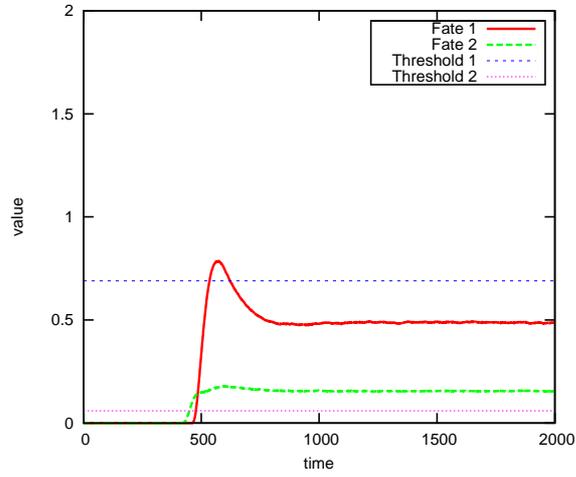


Figure 5.5: Stochastic simulation result averaged over 38,000 runs for VPC 5.

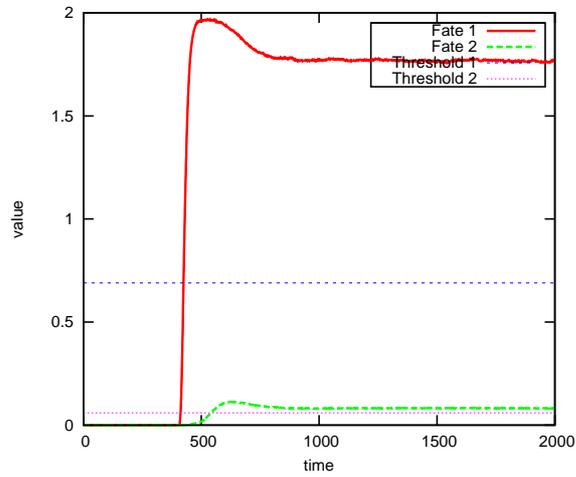


Figure 5.6: Stochastic simulation result averaged over 38,000 runs for VPC 6.

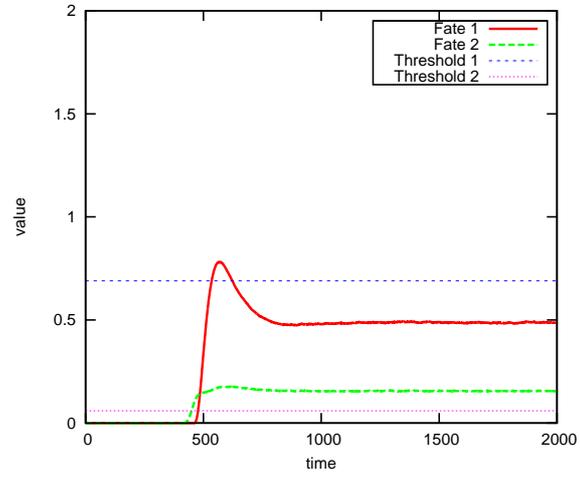


Figure 5.7: Stochastic simulation result averaged over 38,000 runs for VPC 7.

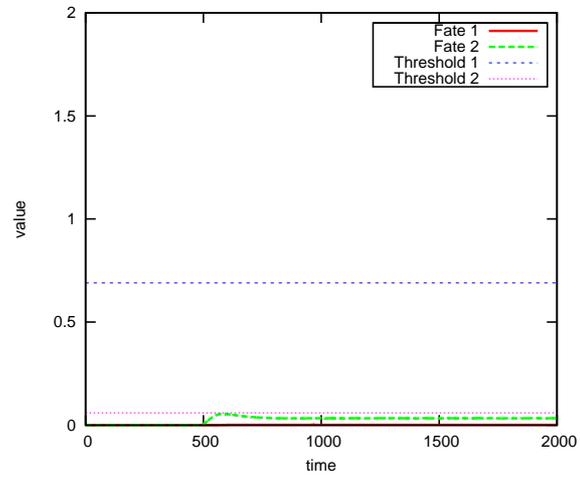


Figure 5.8: Stochastic simulation result averaged over 38,000 runs for VPC 8.

Table 5.6: Detailed statistical results of 50 simulations for the unstable patterns in Table 5.5. We perform 50 simulation experiments for each unstable pattern in Table 5.5, respectively. In each simulation experiment, we perform 38,000 stochastic simulation runs choosing the confidence of 95% and the accuracy of 10^{-2} .

| Rule A | | | Rule S | | |
|----------|-------------|------------|----------|-------------|------------|
| Patterns | Occurrences | Percentage | Patterns | Occurrences | Percentage |
| Row 5 | | | Row 5 | | |
| [122121] | 20/50 | 40% | [122121] | 50/50 | 100% |
| [121121] | 10/50 | 20% | | | |
| [121221] | 5/50 | 10% | | | |
| Row 21 | | | Row 21 | | |
| [121121] | 50/50 | 100% | [121121] | 50/50 | 100% |
| Row 29 | | | Row 29 | | |
| [122121] | 11/50 | 22% | [122121] | 6/50 | 12% |
| [121121] | 24/50 | 48% | [121121] | 32/50 | 64% |
| [121221] | 5/50 | 10% | [121221] | 12/50 | 24% |
| Row 45 | | | Row 45 | | |
| [122121] | 2/50 | 4 % | [122121] | 12/50 | 24% |
| [121121] | 42/50 | 84% | [121221] | 12/50 | 24% |
| [121221] | 6/50 | 12% | [121121] | 20/50 | 40% |
| | | | [122221] | 6/50 | 12% |

From these results we derive the following conclusions:

1. For 44 stable patterns, either Rule A or Rule S detects the expected patterns. But Rule S makes more sense, as it deals with each simulation run of the stochastic model.
2. For 4 unstable patterns, our model produces less unstable patterns than the model in [LNUM09]. This is because we only add noise at the switch time to produce hyp7, but not anywhere else. If we consider more noise, we would obtain more unstable patterns.
3. For the unstable pattern of Row 5 in Table 5.5, we obtain less patterns using Rule S than Rule A. This is because in Rule S we do not consider the time order of *Fate_1* and *Fate_2*, i.e. we do not consider which one first enters a steady state.

Moreover, we can validate our \mathcal{SPN}^c model by transforming it to its corresponding \mathcal{CPN}^c model. This is done by exporting \mathcal{SPN}^c to \mathcal{CPN}^c in Snoopy. By running

continuous simulation and performing model checking using Rule A, we obtain similar results to those above. To be precise, we obtain the same results for the stable patterns and similar results for the 4 unstable patterns. Again, we can also unfold the \mathcal{SPN}^c model into an uncolored model so as to validate it using more analysis techniques.

To summarize up, we can clearly see some advantages of our method:

1. Using colored Petri nets, we obtain a very compact representation of *C. elegans* vulval development. Compared to the model in [LNUM09], our model becomes more readable. Thus it can be deduced that with increasing size of models, colored Petri nets will become more helpful.
2. We can use formal analysis techniques, e.g. simulative model checking of PLTLc, to determine the fate of VPCs.
3. We can analyze our colored model by exporting it to other Petri net paradigms (e.g. \mathcal{QPN}^c , \mathcal{CPN}^c) so as to employ more analysis techniques and perform cross check.
4. A side effect of our method is that we obtain a stochastic model of *C. elegans* vulval development, which is in our opinion more meaningful because of the inherently stochastic nature of biological processes.

5.1.6 Conclusions

We have illustrated how to model *C. elegans* vulval development using colored Petri nets, which addresses such challenges in systems biology as *repetition of cells* that are *organized in one dimensional space*, and *communicate between immediate neighbors*. We have also demonstrated how these three formalisms in the colored Petri net framework and different analysis techniques are combined together to achieve the analysis of a biological system.

For the current colored Petri net model of *C. elegans* vulval development, we could further refine it according to new experimental results and employ more formal analysis techniques to analyze it.

5.2 Modeling Coupled Ca^{2+} Channels

Calcium is a ubiquitous second messenger used to regulate a wide range of cellular processes [Ber97], [SF05]. The most important internal store for free calcium ions ($[Ca^{2+}]$) is the endoplasmic or sarcoplasmic reticulum (ER or SR), and release of Ca^{2+} from this store is mainly mediated by the inositol 1,4,5-trisphosphate receptor (IPR) and the ryanodine receptor (RyR), which are also Ca^{2+} channels. These channels can be activated as well as inactivated by cytosolic Ca^{2+} , and the effect of Ca^{2+} released by

open Ca^{2+} channels on subsequent channel gating is presumably dependent on the details of single-channel kinetics [MTS05].

In order to clarify the possible effect of residual Ca^{2+} on the stochastic gating of Ca^{2+} -regulated Ca^{2+} channels, many mathematical models, e.g. [NMS05], [LSK09], have been built, which are composed of a number of individual channel models whose dynamic behavior depends on the local Ca^{2+} concentration influenced by all open channels. However, most of these models that are based on continuous time Markov chains (CTMCs) suffer from the largeness challenges, either in the model representation or state space [NMS05].

On one hand, there have been some modeling formalisms for addressing model representations of CTMCs, e.g. stochastic Petri nets, whose underlying semantics are CTMCs or Kronecker representations [NMS05], [LSK09]; however the former can not alleviate the largeness in representation of CTMCs although it is graphical and intuitive, while the latter allows a compact representation of CTMCs, but it is not intuitive and can not represent the spatial arrangement of systems to be modeled. Fortunately, colored stochastic Petri nets offer the possibility for combining both compact and intuitive representations of large CTMCs (thus coupled Ca^{2+} channels). More importantly, colored Petri nets provide the possibility to represent spatial arrangements of coupled Ca^{2+} channels, which is very important in modeling coupled Ca^{2+} channels.

In this section we will investigate to use colored stochastic Petri nets to construct scalable spatial models of coupled Ca^{2+} channels that are described by CTMCs. We consider clusters of Ca^{2+} -regulated channels whose stochastic gating depends on Ca^{2+} . The local Ca^{2+} concentration experienced by a particular channel depends on its own state (open or closed) and the state of other channels, that is, open channels increase the Ca^{2+} concentration experienced by neighboring channels. As colored Petri nets can represent a group of similar objects (here Ca^{2+} channels) as a place and use colors to differentiate them, they allow a very compact representation for a large system [LH10a]. Hence, colored Petri nets are a potentially suitable formalism for representing coupled Ca^{2+} channels where each Ca^{2+} channel is encoded as a color. As a result, we may use a very simple model to represent a model with a large quantity of Ca^{2+} channels. More importantly, we do not need to change the structure of this model, and usually only need to change the color set if the number of channels change. It will be proved that this provides a convenient and powerful way to express and analyze scalable clusters of Ca^{2+} -regulated channels.

On the other hand, in order to address the challenge of the large state space, colored stochastic Petri nets offer a large variety of qualitative and quantitative analysis techniques for verifying and analyzing constructed CTMCs models. For example, structural analysis can be used to exploit the structure of models from a graph point of view. Gillespie simulation [Gil77] offers a possibility for approximately analyzing models with huge state space. Numerical analysis techniques can accurately conduct transient and

steady-state analysis on CTMCs models, where both symbolic data structure and specific optimization techniques for Petri nets can contribute to the improvements of the computation efficiency [ST11]. In summary, colored stochastic Petri nets are capable of analyzing CTMCs models with both large representations and large state space.

This section is organized as follows. Section 5.2.1 describes the background of Ca^{2+} -Regulated Ca^{2+} Channels. Section 5.2.2 discusses how to model Ca^{2+} channels using stochastic Petri nets and colored stochastic Petri nets. Section 5.2.3 describes the analysis and validation of the constructed colored models of coupled Ca^{2+} channels. Section 5.2.4 discusses the construction of models with an array of clusters and analysis capabilities offered by colored Petri nets. Section 5.2.5 concludes this section.

5.2.1 Ca^{2+} -Regulated Ca^{2+} Channels

In this section, we will first describe two-state and six-state Ca^{2+} channel models and then introduce coupled Ca^{2+} channels.

(1) A two-state channel model with Ca^{2+} activation.

We first recall the simplest model of a Ca^{2+} channel, i.e. the two-state channel model [Smi02]. The transition diagram for the two-state (closed and open) channel activated by Ca^{2+} is illustrated in Figure 5.9.

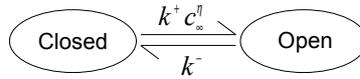


Figure 5.9: A two-state channel with Ca^{2+} activation.

In Figure 5.9, $k^+ c_\infty^\eta$ and k^- are transition rates with units of reciprocal time, k^+ is an association rate constant with units of $conc^{-\eta} time^{-1}$, η is the cooperativity of Ca^{2+} binding, and c_∞ is the fixed background [Ca^{2+}]. This transition diagram is immediately read as a CTMC model. Later we will see how to obtain its Petri net model.

(2) A six-state channel model with Ca^{2+} activation and inactivation.

A six-state channel model with Ca^{2+} activation and inactivation [DLKS08] is illustrated in Figure 5.10. This model is derived from a two-subunit Ca^{2+} channel including both fast Ca^{2+} activation and slower Ca^{2+} inactivation. It has six states: one open state, two closed states and three refractory states.

In Figure 5.10, $k_i^+ c_\infty$ (or $k_d^+(c_\infty + c_d)$) and k_i^- with $i \in \{a, \dots, f\}$ are transition rates with units of reciprocal time, k_i^+ or k_d^+ is an association rate constant with units of $conc^{-1} time^{-1}$, and c_∞ is the fixed background [Ca^{2+}]. The new parameter, c_d , denotes the domain [Ca^{2+}].

(3) Instantaneously coupled channels.

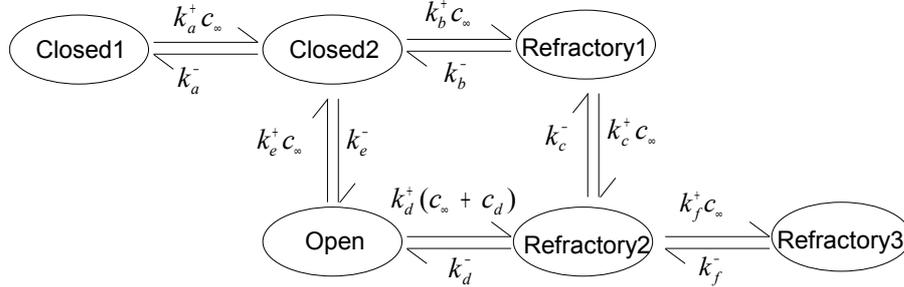


Figure 5.10: A six-state channel with Ca^{2+} activation and inactivation.

Using the assumption of "instantaneous coupling" [NMS05], we can write the transition diagram for two coupled, for example and also for simplicity, two-state channels where both are activated by Ca^{2+} [DS05], illustrated in Figure 5.11.

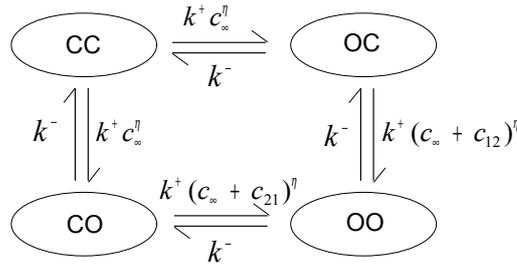


Figure 5.11: Two instantaneously coupled two-state channels. "O" denotes "Open" and "C" "Closed".

This diagram contains four states: both channels closed (CC), both channels open (OO), or one channel open and the other closed (CO and OC). c_{12} (c_{21}) represents the effect experienced by channel 2 (1) when channel 1 (2) is open.

This state transition diagram indicates how to model two instantaneously coupled channels, and we can easily to extend it to a cluster of N coupled two-state channels activated by Ca^{2+} . For example, if we use the assumption of mean-field coupling [NMS05], which assumes that the local $[Ca^{2+}]$ experienced by a channel depends only on the number of open channels at the Ca^{2+} release site, we can write the local $[Ca^{2+}]$ experienced by each channel as $c_\infty + N_O * c_*$, where c_∞ is the background $[Ca^{2+}]$, c_* is the $[Ca^{2+}]$ above background contributed by any open channel, and N_O is the number of open channels.

5.2.2 Modeling

In this section, we will describe how to use stochastic Petri nets and colored stochastic Petri nets to model Ca^{2+} channels, respectively.

(1) Modeling using stochastic Petri nets.

The stochastic Petri net model for the two-state Ca^{2+} channel model in Figure 5.9 is shown in Figure 5.12(a), where each state in Figure 5.9 is modeled as a place in Figure 5.12(a) and each arc in Figure 5.9 is modeled as a transition in Figure 5.12(a) with the rate of the arc being assigned to this transition.

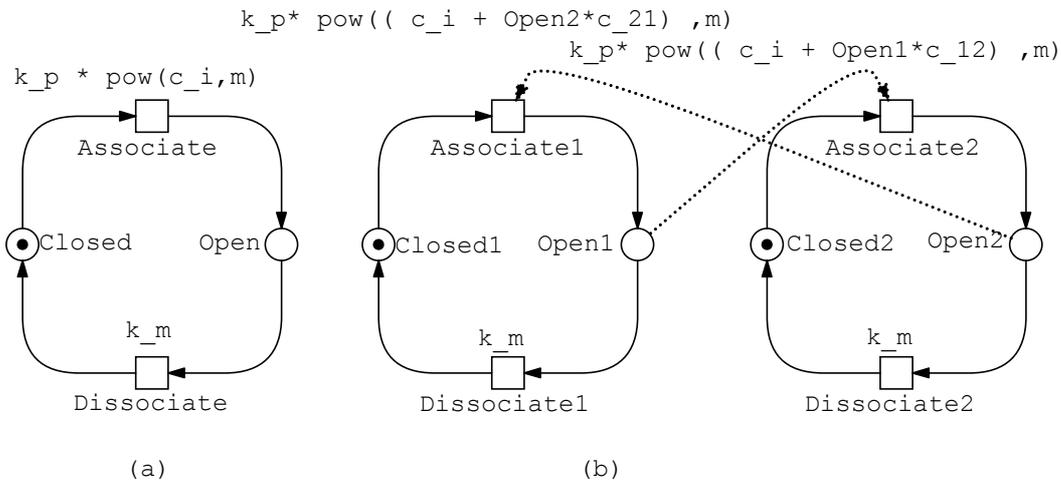


Figure 5.12: (a) A SPN model for the two-state Ca^{2+} channel model in Figure 5.9, and (b) a SPN model for two instantaneously coupled two-state Ca^{2+} channels in Figure 5.11. The place connected by a modifier arc can be used in the rate function of the transition connected by this modifier arc. The mappings of parameters are as follows: $k_p = k^+$, $c_i = c_\infty$, $m = \eta$, $k_m = k^-$, $c_{12} = c_{12}$, $c_{21} = c_{21}$ and $pow()$ is the power function defined in Snoopy.

Further, we build a stochastic Petri net model for two identical coupled two-state channels in Figure 5.11, illustrated in Figure 5.12(b). To model the coupling effects, we use modifier arcs to connect open places with association transitions, e.g. the modifier arc between *Open1* and *Associate2* in Figure 5.12(b). All the rates are labeled in the figure. Please note if both channels are closed, the rate functions for both *Associate1* and *Associate2* become $k_p * pow(c_i, m)$, which corresponds to the rate from *CC* to

CO or OC in Figure 5.11.

We can easily extend this idea to build Ca^{2+} channel models with more states using stochastic Petri nets. For example, Figure 5.13 gives a stochastic Petri net model for the six-state Ca^{2+} channel model in Figure 5.10, in which all rate functions are set according to Figure 5.10.

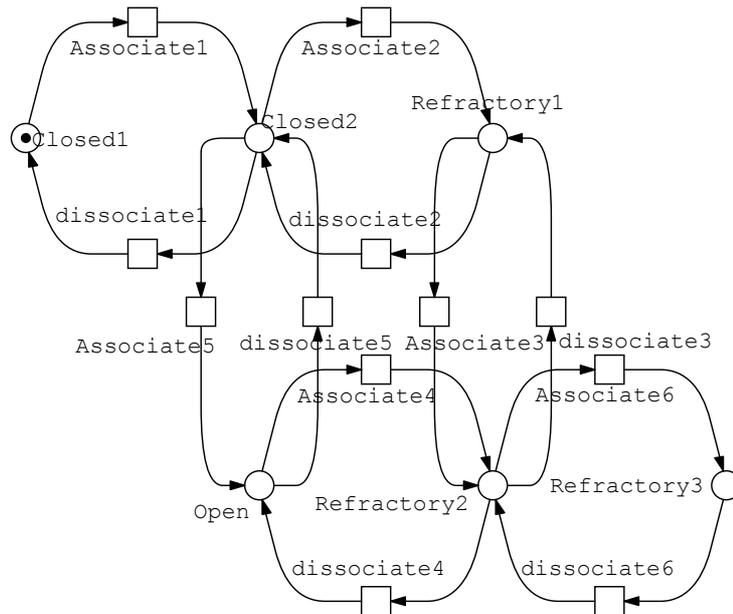


Figure 5.13: A stochastic Petri net model for the six-state channel in Figure 5.10.

(2) Modeling using colored stochastic Petri nets.

One of the key problems during modeling coupled Ca^{2+} channels using colored Petri nets is to encode channels as colors. Using the assumption of mean-field coupling, we do not need to consider spatial arrangements of coupled Ca^{2+} channels, so we can use an integer color to represent each channel.

However, in practice coupled Ca^{2+} channels usually have spatial arrangements, see e.g. [DLKS08], [LSK09]. To explicitly represent this geometry is necessary, especially if the local $[Ca^{2+}]$ experienced by a channel is not only affected by the number of other open channels but also the distances between this channel and the other open channels. When considering this effect during modeling, we often suppose that channels are positioned in a hypothetical spatial arrangement, e.g. in a square [Fal03]. Here, we can address this issue using colored Petri nets. For simplicity, we only consider a regular grid arrangement of coupled Ca^{2+} channels (see Figure 2.9 for an arrangement

of $M \times N$ components), in which each channel is represented by a small rectangle and denoted by two dimensional coordinates (x, y) . No doubt, using colored Petri nets, we can also deal with other regular or irregular arrangements.

We begin with the colorizing of Figure 5.12(b) so as to clearly describe our modeling idea. It has two channels and we can define each channel as a color, so we need to define a color set CS with two colors, e.g. 1 and 2, which will be assigned to each colored place. We then define a variable x on CS , which will be assigned to each arc. Now by folding the two channels in Figure 5.12(b), we obtain its colored Petri net model, illustrated in Figure 5.14(a).

Now we are going to construct a more general colored Petri net model for coupled two-state Ca^{2+} channels, illustrated in Figure 5.14(b). In terms of the encoding way in Figure 2.9, we first define two simple color sets $CRow$ with M colors and $CCol$ with N colors, representing the row and column of a rectangular grid. Based on these two simple color sets, we define a product color set CS , representing a rectangular $M \times N$ grid, which is used to differentiate $M \times N$ channels. To simplify the function of the transition *Associate*, we introduce a new place *NumOpen* to count all the open channels. We can easily increase the number of channels simply by increasing the color set CS . Please note that this model applies under the assumption of mean-field coupling or considering spatial arrangements, and the only difference between these two cases lies in the definition of the rate function of the transition *Associate*.

Besides, using the same modeling idea, we build a colored stochastic Petri net model for coupled six-state Ca^{2+} channels, illustrated in Figure 5.15. In this model we use the same color sets as in Figure 5.14(b) and all the rate functions are set according to Figure 5.10.

5.2.3 Analysis and Validation

Under the formalism of colored Petri nets, a variety of methods and tools are offered to analyze models of coupled Ca^{2+} channels. Among them, structural analysis can be used to validate the structure of models of Ca^{2+} channels; simulation offers the analysis of models with a large number of Ca^{2+} channels or with large state space; model checking (numerical analysis) provides accurate analysis results for models but is subject to the largeness of the state space.

In this section, we will analyze and validate models of coupled Ca^{2+} channels from these three aspects: structural analysis, simulation analysis and model checking (numerical analysis).

(1) Structural analysis.

In order to obtain an initial confidence in colored Petri net models of Ca^{2+} channels, we take the six-state model as an example and conduct structural analysis on it using the analysis tool, Charlie [Cha11].

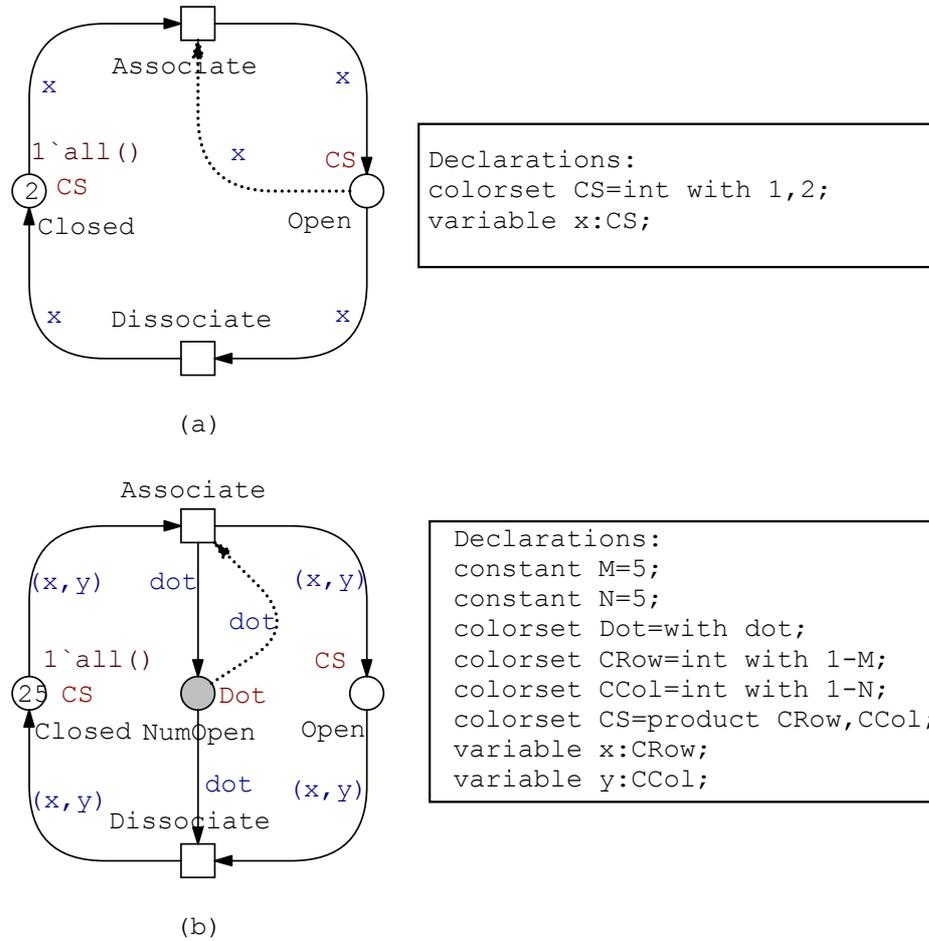


Figure 5.14: (a) A \mathcal{SPN}^C model for two coupled two-state Ca^{2+} channels in Figure 5.12(b), and (b) a general \mathcal{SPN}^C model for coupled two-state Ca^{2+} channels of any number.

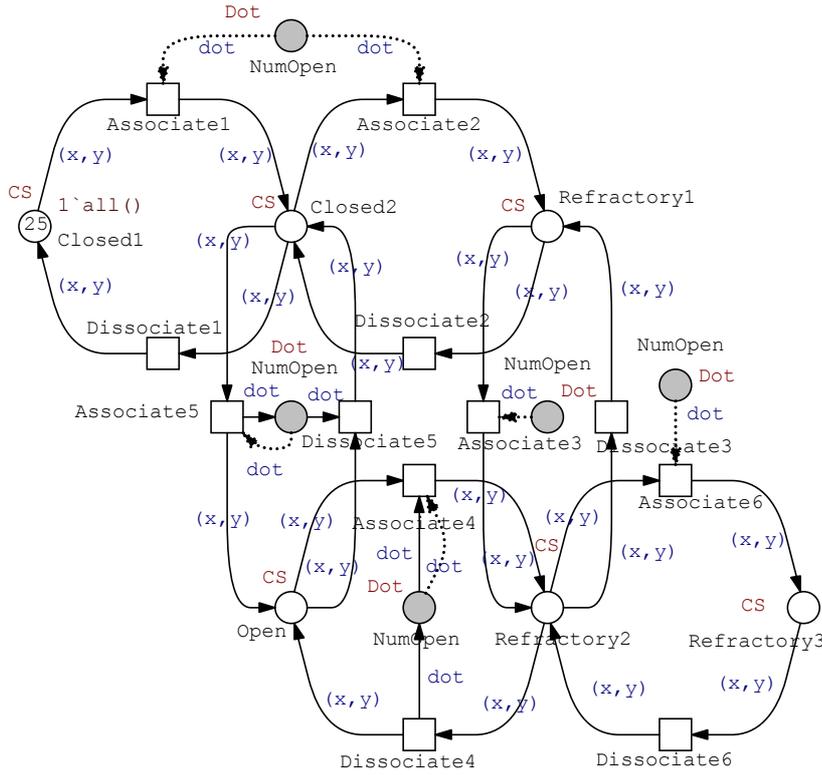


Figure 5.15: A SPN^C model for coupled six-state Ca^{2+} channels. The gray nodes with the name *NumOpen* are logic nodes, i.e. they are the same node with different graphic representations. This model uses the same color sets as in Figure 5.14(b) and all the rate functions are set according to Figure 5.10.

We first conduct structural analysis on the net of one channel model (see Figure 5.13). From the analysis results, we see this net enjoys some interesting properties, e.g. it is pure, ordinary, 1-bounded, live, reversible, and a state machine [GHL07]. Besides, this net is covered by one P-invariant (composed of all places) and eight T-invariants (illustrated in Table 5.7) that reflect the association and dissociation process of the channel.

Table 5.7: The minimal T-invariants for the one channel model in Figure 5.13, where "D" denotes "Dissociate" and "A" "Associate".

| No. | Transitions | No. | Transitions |
|-----|-------------|-----|----------------|
| 1 | D1, A1 | 5 | D5, A5 |
| 2 | D2, A2 | 6 | D6, A6 |
| 3 | D3, A3 | 7 | D2, D3, A4, A5 |
| 4 | D4, A4 | 8 | D4, D5, A2, A3 |

We then conduct structural analysis on the net of two coupled six-state channels (see Figure 5.15, where we set the color set CS to two colors.). This net is also pure, ordinary, live and reversible. Although it is still structurally bounded, but is not 1-bounded because of the place *NumOpen*. This net is not a state machine any more due to the branching transitions. Moreover, this net is covered by three P-invariants (illustrated in Table 5.8) and sixteen T-invariants (illustrated in Table 5.9).

With the increase of states of a channel, it will become more important to conduct structural analysis on Ca^{2+} channel models.

Table 5.8: The minimal P-invariants of the two coupled channels model in Figure 5.15, where "C" denotes "Closed", "A" "Refractory", "O" "Open" and "N" "NumOpen". The number following the underscore denotes a channel.

| No. | Transitions |
|-----|---|
| 1 | C1_1,C2_1,R1_1,R2_1,R3_1,O_1 |
| 2 | C1_2,C2_2,R2_2,R2_2,R3_2,O_1 |
| 3 | C1_1,C1_2,C2_1,C2_2,R1_1,R2_1, R3_1,R2_2,R2_2,R3_2,N_1 |

(2) Simulation analysis.

Using the Gillespie's simulation algorithm [Gil77] we can estimate some response measures, such as the distribution of the number of open channels, which gives a preliminary analysis of models. In the following, we first give some simulation results according to the simulation settings in [DS05] and [LSK09] to validate our models.

Table 5.9: The minimal T-invariants of the two coupled channels model in Figure 5.15, where "D" denotes "Dissociate" and "A" "Associate". The number following an underscore denotes a channel.

| No. | Transitions | No. | Transitions |
|-----|-------------|-----|------------------------|
| 1 | D1_2, A1_1 | 9 | D3_1, A3_1 |
| 2 | D1_2, A1_2 | 10 | D3_2, A3_2 |
| 3 | D6_1, A6_1 | 11 | D4_1, A4_1 |
| 4 | D6_2, A6_2 | 14 | D4_2, A4_2 |
| 5 | D2_1, A2_1 | 13 | D4_1, D5_1, A2_1, A3_1 |
| 6 | D5_1, A5_1 | 12 | D2_1, D3_1, A4_1, A5_1 |
| 7 | D2_2, A2_2 | 15 | D2_2, D3_2, A4_2, A5_2 |
| 8 | D5_2, A5_2 | 16 | D4_2, D5_2, A2_2, A3_2 |

Figure 5.16(a) gives a simulation plot of a single simulation run for a Ca^{2+} release site with 19 two-state channels under the assumption of mean-field coupling. From this plot, we can see that this model exhibits stochastic Ca^{2+} excitability, which is consistent with the result given in [DS05]. Figure 5.16(b) gives a simulation plot of the average behavior of 1000 simulation runs in the same setting, which shows that the mean of the number of open channels is around 2.47 that is given in [LSK09]. For other settings, we obtain similar simulation results as those in [DS05] and [LSK09].

Furthermore, Figure 5.17 gives a simulation plot of the average behavior of 1000 simulation runs for a Ca^{2+} release site with 4 two-state channels by considering the position effects of individual channels, which is also consistent with the result given in [LSK09].

Besides, we give a simulation plot of the average behavior of 1000 simulation runs for a Ca^{2+} release site with 19 six-state channels under the assumption of mean-field coupling. The parameters in this setting partly come from [DLKS08].

(3) State space construction.

For models of coupled Ca^{2+} channels, the state space explosion will occur when the number of channels or number of states per channel becomes larger. Here we use Marcie to explore the construction of the state space for coupled six-state Ca^{2+} channels.

Marcie provides functionalities for the analysis of standard Petri net properties as well as model checking of CTL and CSL. It uses the Interval decision diagrams (IDD) to alleviate the problem of state space explosion, so it usually shows good performance [SRH11]. Using Marcie, we explore the size of the state space and its construction time for coupled six-state Ca^{2+} channels in Figure 5.15 by changing the number of channels, illustrated in Table 5.10. As we can see, with the increasing of the number of Ca^{2+} channels, both the state space and its construction time increase rapidly. However, the number of Ca^{2+} channels that Marcie can deal with is limited within 25.

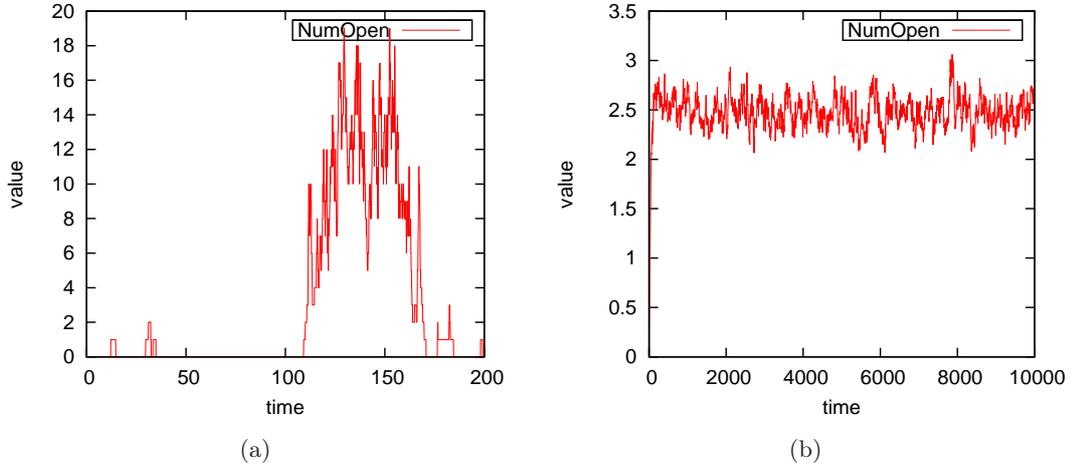


Figure 5.16: Stochastic simulation result for the model of 19 two-state Ca^{2+} channels under the assumption of mean-field coupling (a) for one simulation run and (b) averaged over 1000 runs. Parameters used: $N = 19$, $\eta = 2$, $c_\infty = 0.05$, $k^+ = 1.5$, $k^- = 0.5$, $c^* = 0.0637$.

Table 5.10: Comparison of the state space construction in terms of the number of channels (NC), the number of states and the construction time for the model of coupled Ca^{2+} channels in Figure 5.15.

| NC | States | Time (seconds) |
|------|-------------|----------------|
| 1 | 7 | 0.00 |
| 5 | 14,256 | 0.10 |
| 10 | 161,243,136 | 0.37 |
| 15 | 1.645e+12 | 102.25 |
| 20 | 1.584e+16 | 4,036.57 |
| 22 | 6.142e+17 | 22,337.28 |
| 25 | ◇ | ◇ |

* done on MAC Pro, 8×2.2GHz, 16GB RAM.

◇ Marcie failed to give the result.

(4) CSL model checking.

We now explore how to use the temporal logic CSL to express properties of our models and then conduct CSL model checking (numerical analysis) [ST11] to analyze them. For example, for the model of six coupled six-state channels, we give the following properties from the steady-state analysis, transient analysis and analysis of arbitrarily specified properties, respectively:

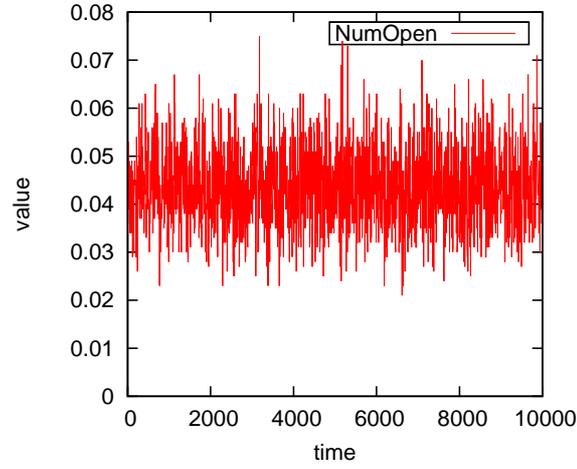


Figure 5.17: Stochastic simulation result averaged over 1000 runs for the model of 4 two-state Ca^{2+} channels by considering the position of individual channels. Parameters used: $N = 19$, $\eta = 2$, $c_\infty = 0.05$, $k^+ = 1.5$, $k^- = 0.5$, $c^* = 0.1452$.

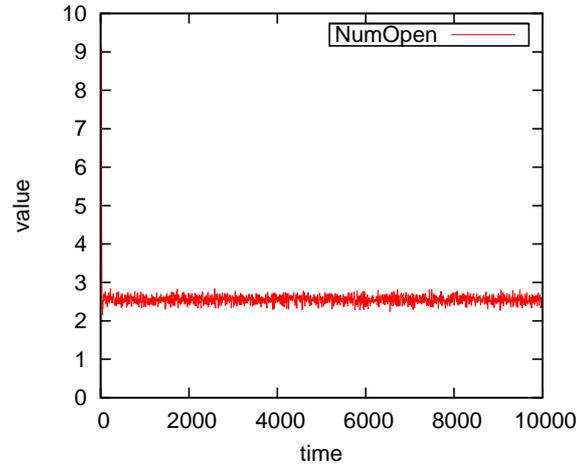


Figure 5.18: Stochastic simulation result averaged over 1000 runs for the model of 19 six-state Ca^{2+} channels under the assumption of mean-field coupling. Parameters used: $k_a^+ = 1.5$, $k_b^+ = k_d^+ = 0.015$, $k_c^+ = k_e^+ = 300$, $k_a^- = 3.0$, $k_f^- = 49.5$, $k_b^- = k_d^- = 0.2475$, $k_c^- = k_e^- = 6.0$, $k_f^- = 0.03$, $c_\infty = 0.05$, $c^* = 0.4$, $c^d = 0.5$.

- Query 1: $R\{\text{"NumOpen"}\}_{=?}[S]$: the mean value of the open channels in the steady state ("NumOpen" is a reward, which is used to measure the number of open channels).
- Query 2: $P_{=?}[F[100, 100][\text{Open}_1 = 1]]$: the probability of one of the channels (e.g. channel 1) being in the "Open" state at some time instant (e.g. 100).
- Query 3: $P_{=?}[G[\text{Closed1}_1 = 1 \rightarrow P_{>=1}[F[\text{Open}_1 = 1]]]]$: if one of the channels (e.g. channel 1) is in the "Closed1" state, then what is the probability of it going to the "Open" state in the future.

Using Marcie [SRH11], we obtain results for these three queries: 0.32085 for Query 1 that is consistent with the simulation result, 0.055 for Query 2 and 1 for Query 3. From this analysis, we can see that model checking offers another way to yield a better insight into the dynamics of the model and identify its interesting behavior.

5.2.4 Discussions

In this section, we will discuss what else could be offered for modeling coupled Ca^{2+} channels by colored Petri nets from the following two points.

Construction of models with an array of clusters.

We have encoded coupled channels in two dimensional space, i.e. using two dimensional coordinates, e.g. (x, y) to give the position of each channel. However, so far all the models given in this paper only consider a single cluster (a group of strongly coupled channels).

As stated in [Fal03], [SF05], in order to obtain oscillations and waves, it is necessary to consider more than one clusters that are weakly coupled. It is both the strong coupling within clusters and the weak coupling between clusters that contribute to the oscillations and waves. Using colored Petri nets, we can easily build models with an array of clusters. For this purpose, we can use hierarchical color sets, i.e. we use a pair of coordinates (a, b) to encode the locality of each cluster, in which we use another pair of coordinates (x, y) to encode each channel in a cluster. Now we can use a hierarchical color set $((a, b), (x, y))$ to locate each channel in a cluster. This offers an nice way to build hierarchically scalable models with an array of clusters.

Analysis capabilities offered by colored Petri nets.

With the increasing number of channels, numerical methods become unsuitable. In this situation, we can still resort to stochastic simulation to do analysis. Here we will explore the computational capabilities of colored Petri nets in simulation analysis of channel models. For this, we consider two key technical problems: unfolding [LH10b] and simulation. For colored Petri nets, we have to unfold them into flat nets and then

Table 5.11: The size of the Ca^{2+} channel model and its unfolding and simulation run time*. The simulation runtime means the total time of 100 simulation runs.

| Size | | | Time (seconds) | | | |
|----------|--------|-------------|----------------|------------------------|------------|-------------------------|
| Channels | Places | Transitions | Unfolding | Unfolding/ channels | Simulation | Simulation/ channels |
| 19 | 115 | 228 | 0.085 | 0.0045 | 2.032 | 0.1070 |
| 100 | 601 | 1,200 | 0.146 | 0.0015 | 14.152 | 0.1415 |
| 500 | 3,100 | 6,000 | 0.464 | 0.0009 | 100.648 | 0.2013 |
| 1,000 | 6,001 | 12,000 | 0.903 | 0.0009 | 558.442 | 0.5584 |

* done on PC, Intel(R) Xeon(R) CPU 2.83GHz, RAM 4.00GB.

simulate them using continuous or stochastic simulation algorithms. So both unfolding and simulation decide how far we can go using colored Petri nets.

Using the colored Petri net model in Figure 5.15, we perform a group of testing by increasing the number of channels and obtain the unfolding/simulation run time for different size of the model, illustrated in Table 5.11. From the ratio of the unfolding/simulation run time to the number of channels, we can see that either unfolding or simulation run time approximately linearly increases with the increasing size of the model, so no doubt we can simulate much larger systems if enough unfolding/simulation run time is allowed.

5.2.5 Conclusions

We have demonstrated how to use colored Petri nets to construct spatial models of coupled Ca^{2+} channels, which shows that colored Petri nets are an effective visual modeling formalism to represent scalable coupled Ca^{2+} channels with specific spatial arrangements. More importantly, colored Petri nets provide rich analysis techniques for analyzing and validating models of Ca^{2+} channels. In the future, we are going to extend to model coupled Ca^{2+} channels with more states and to exploit a more scalable model that models more than one clusters (scalable not only in the number of clusters but also in the number of channels in one cluster).

5.3 Modeling Membrane Systems

Membrane systems (also known as P systems) [Pau99], [Pau02] are a very powerful and efficient computational model inspired by the internal organization of living cells with different membranes hierarchically arranged. The membranes enclose compartments where specific biochemical reactions take place. These reactions can transport packages of objects (molecules) from one part of a cell to other parts of the cell. The

objects evolve by means of evolution rules in a nondeterministic and maximally parallel manner. A number of variants of membrane systems have been proposed to deal with different problems. For example, dynamic membrane systems are presented to consider membrane creation, dissolution, or division [Pau02]. Stochastic membrane systems are proposed to model biological systems exhibiting stochastics [GMRC10].

In order to complement the description and analysis of the dynamic behavior of membrane systems, Petri nets have been used to remodel membrane systems and translations of some classes of membrane systems into Petri nets have been proposed, e.g. in [QJY04], [KKR06] and [KK09]. But nearly all these researches aim to employ the maximal concurrency semantics of Petri nets to simulate the nondeterministic and maximally parallel evolution manner.

In this section, we will first investigate how to use Petri nets to represent basic, dynamic and stochastic membrane systems. Consequently once a membrane system is transformed into a Petri net model, all techniques and tools for Petri nets can be used to investigate what is going on during an evolution of a membrane system. However, in a membrane system objects perform their function only at the right membrane, so each object at different locations (compartments) has to be represented as a couple of places in standard Petri nets; as a result, the Petri net model for a large-scale membrane system may become quite large and hard to manage.

To address this issue, we will explore to use colored Petri nets to model membrane systems. As colored Petri nets can represent a group of similar objects as a place and use colors to differentiate them, they allow a very compact representation for a large system. Hence, colored Petri nets are a potentially suitable formalism for representing membrane systems where each object at different compartments is represented as a colored place and compartments are differentiated by colors. As a result, we not only distinguish and show compartment information in a colored Petri net model of a membrane system, but also make it more compact. Please note that Qi et al. [QJY04] proposed a high-level framework called membrane Petri nets based on colored Petri nets to model membrane systems, but it is just a general idea and hard to operate. In contrast, we will explore how to systematically construct and analyze membrane systems using colored Petri nets.

This section is organized as follows. Section 5.3.1 describes the background of membrane systems. Section 5.3.2 discusses how to use Petri nets to model membrane systems. Section 5.3.3 discusses how to use colored Petri nets to model membrane systems. Section 5.3.4 gives an example, the viral infection. Section 5.3.5 concludes this section.

5.3.1 Membrane Systems

In this section, we will briefly introduce basic, dynamic and stochastic membrane systems, respectively.

Throughout this section, we use $\{ \}$ or $\{ \}_l$, a curly bracket or a curly bracket with a label, to specify a multiset or a multiset in a compartment l . We can also omit the curly bracket and only represent a multiset over S as a string $m(s_1)s_1, m(s_2)s_2, \dots, m(s_n)s_n$, where $S = \{s_1, s_2, \dots, s_n\}$.

(1) Basic membrane systems.

We now recall the basic definition of membrane systems according to [Pau99], [Pau02].

Definition 20 (Membrane system)

A *membrane system* is a construct of the form:

$\amalg = (O, \mu, w_1, w_2, \dots, w_m, R_1, R_2, \dots, R_m)$, where

- O is a finite and non-empty alphabet of objects (e.g. molecules, proteins or complexes of proteins).
- μ is a membrane structure, consisting of m membranes, labeled with $1, 2, \dots, m$.
- w_i is a multiset of objects (which gives the number of occurrences of objects in an unordered way) associated with membrane i , $i = 1, 2, \dots, m$.
- R_i is a finite set of evolution rules associated with membrane i , $i = 1, 2, \dots, m$.

An evolution rule takes the form of $r : u_r \rightarrow v_r$, where u_r is a multiset over O , and v_r is a multiset over $O \times (\{here, out\} \cup \{in_j | 1 \leq j \leq m-1\})$, where *here*, *out* or *in_j* represents a location having the following meanings (assume the compartment containing u_r is compartment k):

- *here*: the product remains in the same compartment k , and we usually omit it;
- *out*: the product is transported out of the current compartment k and sent to its parent compartment;
- *in_j*: the product is transported to a compartment j , immediately enclosed by k .

In this definition, the membrane structure defines a hierarchy of compartments enclosed by membranes. Each membrane encloses a compartment and each compartment may contain basic objects (molecules) or other compartments. Membrane systems evolve in the maximally parallel manner, non-deterministically choosing rules and objects. The membrane structure and the multisets of objects in its compartments consist of a configuration of a membrane system. The initial configuration is given by the membrane structure and the multisets of objects available in their compartments at the beginning of a computation. During the evolution of the system, by means of applying the rules, both the multisets of objects and the membrane structure may change.

In the following, we give some examples of evolution rules in basic membrane systems.

- $a \rightarrow \lambda$: a degrades, where λ denotes empty.
- $a, b \rightarrow c$: a and b are combined to form a complex c .
- $c \rightarrow a, b$: c is broken into a and b .
- $a \rightarrow (a, out)$: a is sent to its parent compartment.
- $a \rightarrow (a, in_j)$: a is sent to a child compartment j .

Figure 5.19 gives a membrane system. It consists of three membranes (compartments). The membrane structure is $\{\{\{ \}_3\}_2\}_1$. $w_1 = \{a\}$, $w_2 = \{ \}$ and $w_3 = \{b\}$. $R_1 = \{r_{11} : b \rightarrow a; r_{12} : a \rightarrow (a, in_2), b\}$, $R_2 = \{r_{21} : a \rightarrow (a, in_3), b; r_{22} : b \rightarrow \lambda\}$ and $R_3 = \{r_{31} : a, b \rightarrow b, (b, out)\}$.

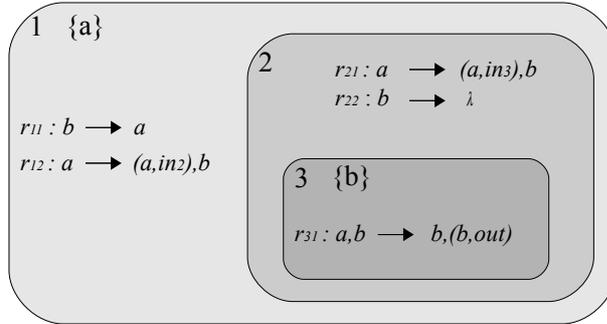


Figure 5.19: A basic membrane system.

(2) Dynamic membrane systems.

The basic membrane systems can be extended to have dynamic structure by considering membrane creation, dissolution, merging or division [Pau02]. All these dynamic changes in structure are reflected by evolution rules. Here we just give some typical evolution rules:

- $\{a, b\}_{l_1} \rightarrow \{\{a\}_{l_2}, b\}_{l_1}$: a is included in a newly created compartment l_2 .
- $\{a, b\}_l \rightarrow \{a\}_{l_1} + \{b\}_{l_2}$: compartment l is divided into two compartment l_1 and l_2 .
- $\{a\}_{l_1} + \{b\}_{l_2} \rightarrow \{a, b\}_l$: compartment l_1 and l_2 are merged into one compartment l .

- $\{\{a\}_{l_2}, b\}_{l_1} \rightarrow \{a, b\}_{l_1}$: compartment l_2 containing a is dissolved and a is now released into its parent compartment l_1 . For simplicity, we can also use another way to represent the dissolution rule, $a \rightarrow a, \delta$, where δ denotes dissolving.

The reader can refer to [Pau02] for detailed descriptions of evolution rules in dynamic membrane systems.

For example, Figure 5.20 gives a membrane system with dynamic structure, where the evolution rule r_{31} dissolves membrane m_3 when it is executed.

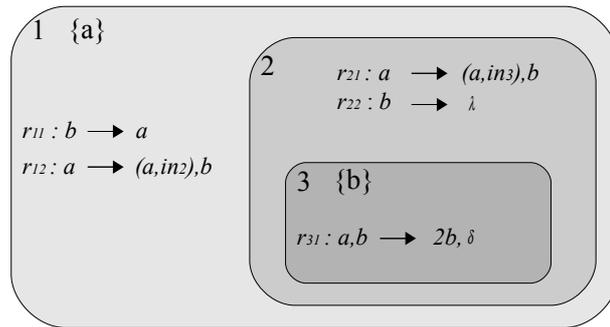
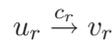


Figure 5.20: A dynamic membrane system.

(3) Stochastic membrane systems.

The membrane systems above usually evolve in the maximally parallel manner, non-deterministically choosing rules and objects. However, if we want to employ membrane systems to model biological systems with stochastics, a stochastic strategy for evolution rules has to be introduced [GMRC10], [SMC⁺08]. To do this, we have to change the form of a evolution rule as



where c_r is introduced to denote a stochastic constant that is used to compute the probability of the rule according to the Gillespie's theory of stochastic kinetics [Gil77]. For example, in the rule $a \xrightarrow{0.1} b$, we compute the probability of the rule as $0.1 * |a|$, where $|a|$ is the quantity of object a . After that, we can use the Gillespie's exact stochastic simulation algorithm [Gil77] to simulate stochastic membrane systems.

5.3.2 Modeling Using Petri Nets

Petri nets are a suitable formalism to describe membrane systems [KKR06], [KK09]. However, in [KKR06] and [KK09], they aim to employ the maximal concurrency semantics of Petri nets to simulate the nondeterministic and maximally parallel evolution

manner of membrane systems. In contrast, we want to model membrane systems using standard Petri nets, so that we can use all analysis techniques applicable to Petri nets to exploit qualitative properties and dynamic behavior of membrane systems.

(1) Mapping basic and stochastic membrane systems to Petri nets.

In order to model membrane systems using Petri nets, places of Petri nets are used to represent objects at different locations, whereas transitions of Petri nets are used to represent evolution rules associated with specific compartments. When a transition fires, it removes objects from its input places and add objects to its output places, thus simulating the behavior of an evolution rule. The initial resources in all compartments consist of the initial marking of a Petri net. The following mapping formalizes the informal descriptions based on [KKR06].

Definition 21

Let $\Pi = (O, \mu, w_1, w_2, \dots, w_m, R_1, R_2, \dots, R_m)$ be a basic and stochastic membrane system. Its corresponding stochastic Petri net $\mathcal{SPN} = (P, T, F, f, v, m_0)$ can be obtained by

- $P = O \times \{1, 2, \dots, m\}$.
- $T = R_1 \cup R_2 \cup \dots \cup R_m$.
- For every place $p = (o, j) \in P$, where $o \in O$ and $j \in \{1, 2, \dots, m\}$, and every transition $t \in T$, where t takes the form of $r : u_r \xrightarrow{c_r} v_r$,
 - $f(p, t) = u_r(o)_j$, which means the occurrence of object o belonging to membrane j in u_r of rule r . If $f(p, t) > 0$, there exists an arc $F(p, t)$.
 - $f(t, p) = v_r(o)_j$, which means the occurrence of object o belonging to membrane j in v_r of rule r . If $f(t, p) > 0$, there exists an arc $F(t, p)$.
- For every transition $t \in T$, whose corresponding evolution rule r takes the form of $u_r \xrightarrow{c_r} v_r$, the rate function for t is: $v(t) = MA(c_r)$, where MA denotes the mass action function.
- For every place $p = (o, j) \in P$, $m_0(p) = w_j(o)$.

To model membrane systems using Petri nets, the key is to model different kinds of evolution rules, and then we can build the whole model based on these basic Petri net components. In Figure 5.21 we give Petri net models for some typical evolution rules in basic membrane systems according to the mapping rule.

For example, based on the basic Petri net components, Figure 5.22 gives a Petri net model for the membrane system in Figure 5.19.

(2) Mapping dynamic and stochastic membrane systems to Petri nets.

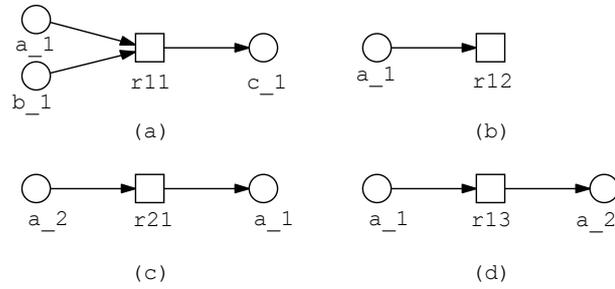


Figure 5.21: Petri net models for typical evolution rules (assume the membrane structure is $\{\{ \}_2\}_1$, $R_1 = \{r_{11}, r_{12}, r_{13}\}$ and $R_2 = \{r_{21}\}$): (a) $r_{11} : a, b \rightarrow c$, (b) $r_{12} : a \rightarrow \lambda$, (c) $r_{21} : a \rightarrow (a, out)$ and (d) $r_{13} : a \rightarrow (a, in_2)$. Please note that each place is named by a compartment label given as a suffix in its corresponding object name.

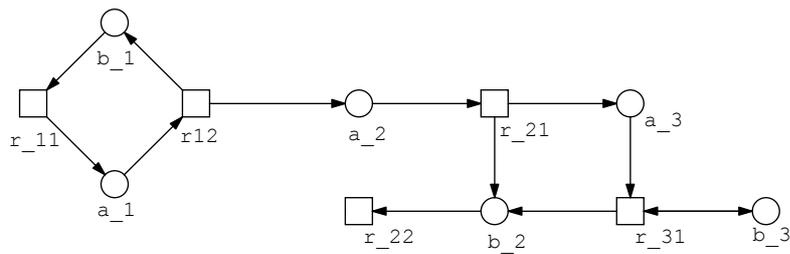


Figure 5.22: A Petri net model for the basic membrane system in Figure 5.19.

In the following, we first take dynamic membrane systems only containing the dissolution rule as an example to describe how to transform them to stochastic Petri nets in a precise manner based on [QJY04] and [KK09]. We then briefly describe how to construct Petri net models for dynamic membrane systems with other dynamic rules.

Definition 22

Let $\amalg = (O, \mu, w_1, w_2, \dots, w_m, R_1, R_2, \dots, R_m)$ be a dynamic and stochastic membrane system only with a dissolution rule. Its corresponding stochastic Petri net $\mathcal{SPN} = (P \cup P_\delta, T \cup T_\delta, F \cup F_\delta \cup F_I \cup F_R, f \cup f_\delta \cup f_I \cup f_R, v, m_0 \cup m_\delta)$ can be obtained by

- (P, T, F, f, v, m_0) can be obtained using Definition 21.
- P_δ is the set of disabling places.
- T_δ is the set of immediate transitions, responsible for transferring all objects of the dissolved membrane to its parent membrane.
- F_δ, F_I and F_R are the sets of standard, inhibitor and read arcs, respectively, from P_δ and T_δ .
- f_δ, f_I and f_R are the weights of F_δ, F_I and F_R , respectively, and are always set to 1.
- m_δ is the initial marking of P_δ and is set to empty.

Definition 23

$P_\delta, T_\delta, F_\delta, F_I$ and F_R are constructed for membrane i with a dissolution rule, $r_\delta \in R_i$ as follows:

1. A disabling place d_i is added to P_δ :

$$P_\delta = P_\delta \cup \{d_i\}.$$
2. Let O_i denote all objects in membrane i . For each object $o_{ij} \in O_i$, an immediate transition it_{ij} is added to T_δ :

$$T_\delta = T_\delta \cup \{it_{ij} | j = 1, 2, \dots, |O_i|\}.$$
3. A standard arc (r_δ, d_i) is added to F_δ .

$$F_\delta = F_\delta \cup \{(r_\delta, d_i)\}.$$
4. For each object $o_{ij} \in O_i$ and its corresponding object o_{kj} in its parent membrane k , a pair of standard arcs are added to F_δ .

$$F_\delta = F_\delta \cup \{(o_{ij}, it_{ij}) | j = 1, 2, \dots, |O_i|\} \cup \{(it_{ij}, o_{kj}) | j = 1, 2, \dots, |O_i|\}.$$
5. For each rule r_{ik} in the rule set R_i , an inhibitor arc (d_i, r_{ik}) is added.

$$F_I = F_I \cup \{(d_i, r_{ik}) | k = 1, 2, \dots, |R_i|\}.$$

6. Let R_{in} denote all rules which can send objects to membrane i from its parent membrane. For each rule $r_k \in R_{in}$, an inhibitor arc (d_i, r_k) is added.
 $F_I = F_I \cup \{(d_i, r_k) | k = 1, 2, \dots, |R_{in}|\}$.
7. For each transition $t_k \in T_\delta$, a read arc (d_i, t_k) is added.
 $F_R = F_R \cup \{(d_i, t_k) | k = 1, 2, \dots, |T_\delta|\}$.

For dynamic membrane systems containing creation rules, we can deal with them using the same transformation method as basic membrane systems in Definition 21. For others with division or merging rules, we can handle them using a similar way as membrane systems with dissolution rules. We will not discuss them deeply because of space constraint.

For dynamic membrane systems with dissolution rules, Qi et al. [QJY04] use sending transitions to send all objects in a membrane to its parent membrane when a dissolution rule is executed. However, for standard qualitative Petri nets, it is difficult to implement sending transitions and thus few tools support this special situation. Alternatively, in [KK09] Kleijn et al. make copies of all transitions in a dissolved membrane, but the price is that a lot of extra transitions have to be added, which makes Petri net models more complex and difficult to manage.

In contrast, we consider a kind of stochastic dynamic membrane systems, so we can adopt a different way to deal with dissolution rules, i.e. we employ immediate transitions to transfer the resources of dissolved membranes to their parent membranes. For example, if we consider the membrane system in Figure 5.20 as a stochastic membrane system, we can build its stochastic Petri net model as in Figure 5.23.

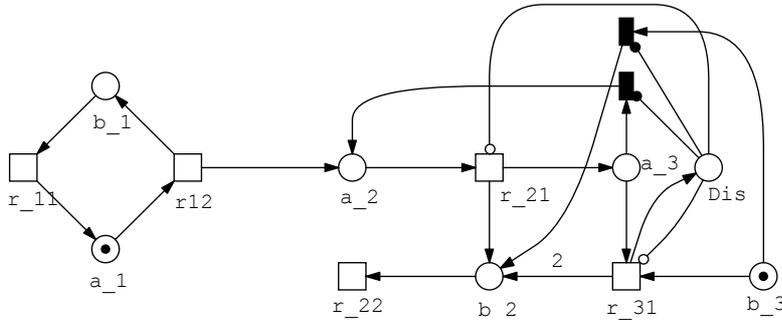


Figure 5.23: A Petri net model for the dynamic membrane system in Figure 5.20. A read arc is represented as an arc with a solid circle and an inhibitor arc with a hollow circle. An immediate transition is represented as a black rectangle.

5.3.3 Modeling Using Colored Petri Nets

Using standard Petri nets, each object of a membrane system at different compartments has to be represented as a coupled of places, thus the Petri net model for the membrane system may become quite large and hard to manage. In contrast, colored Petri nets represent a group of similar objects as a place and use colors to differentiate them, so they offer a very compact representation for a large membrane system.

(1) Mapping basic and stochastic membrane systems to colored Petri nets.

In order to model membrane systems using colored Petri nets, we define each membrane as a color consisting of a color set, which is used to differentiate the location of each object. We then fold the same object at different compartments as one colored place and assign the color set to this place. The following definition gives a formal translation from basic and stochastic membrane systems into colored stochastic Petri nets.

Definition 24

Let $\Pi = (O, \mu, w_1, w_2, \dots, w_m, R_1, R_2, \dots, R_m)$ be a basic and stochastic membrane system. Its corresponding colored stochastic Petri net $\mathcal{SPN}^C = (P, T, F, \Sigma, C, g, f, v, m_0)$ can be obtained by

- $\Sigma = \{\{1, 2, \dots, m\}\}$, which contains a color set $\{1, 2, \dots, m\}$ that encodes each compartment as a color.
- $P = O$.
- $T = R_1 \cup R_2 \cup \dots \cup R_m$.
- For every place $p = o \in O$, every color $j \in \{1, 2, \dots, m\}$, and every transition $t \in T$, where t takes the form of $r : u_r \xrightarrow{c_r} v_r$,
 - $f(p, t) = \sum_{j=1}^m u_r(o)_j j$, where $u_r(o)_j$ means the occurrence of object o belonging to membrane j in u_r of rule r . If $f(p, t) > 0$, there exists an arc $F(p, t)$.
 - $f(t, p) = \sum_{j=1}^m v_r(o)_j j$, where $v_r(o)_j$ means the occurrence of object o belonging to membrane j in v_r of rule r . If $f(t, p) > 0$, there exists an arc $F(t, p)$.
- For every place $p \in P$, the color set $\{1, 2, \dots, m\}$ is assigned.
- For every transition $t \in T$, $g(t)$ is always set to true.
- For every transition $t \in T$, whose corresponding evolution rule r takes the form of $u_r \xrightarrow{c_r} v_r$, the rate function for t is defined as $v(t) = MA(c_r)$.
- For every place $p = o \in O$, $m_0(p) = \sum_{j=1}^m w_j(o)$.

For example, Figure 5.24 gives a colored Petri net model for the membrane system in Figure 5.19 according to the transformation rules. We define a color set CS with 3 colors, 1, 2 and 3, representing three membranes, respectively.

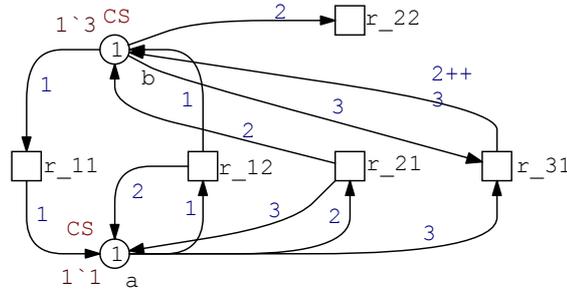


Figure 5.24: A colored Petri net model for the basic membrane system in Figure 5.19.

(2) Mapping dynamic and stochastic membrane systems to colored Petri nets.

In this section, we also take dynamic membrane systems only containing dissolution rules as an example to give the translation from dynamic membrane systems to colored Petri nets. We can deal with membrane systems with other types of dynamic rules in a similar way.

Definition 25

Let $\Pi = (O, \mu, w_1, w_2, \dots, w_m, R_1, R_2, \dots, R_m)$ be a dynamic and stochastic membrane system only containing a dissolution rule. Its corresponding colored Petri net $\mathcal{SPN}^C = (P \cup P_\delta, T \cup T_\delta, F \cup F_\delta \cup F_I \cup F_R, \Sigma, g, f \cup f_\delta \cup f_I \cup f_R, v, m_0 \cup m_\delta)$ can be obtained by

- $(P, T, F, \Sigma, g, f, v, m_0)$ can be obtained using Definition 24.
- Other components have the same meaning as in Definition 22 and 23.

For example, Figure 5.25 gives a colored Petri net model for the membrane system in Figure 5.20. We also define a color set CS with 3 colors, 1, 2 and 3, representing three membranes, respectively.

From these examples we can see that using colored Petri nets we can obtain a compact model for a membrane system by folding the same kind of object at different compartments into a place and defining each membrane as a color. Then we can use the color to differentiate the object at a specific compartment. The colored Petri net model would become very compact if a membrane system consists of a lot of objects but few evolution rules.

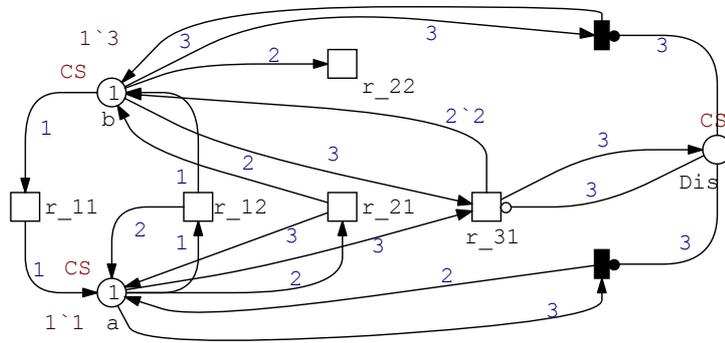


Figure 5.25: A colored Petri net model for the dynamic membrane system in Figure 5.20.

5.3.4 An Example: the Viral Infection

In this section, we use the process of the viral infection [SMC⁺08] to demonstrate how to use colored Petri nets to model a membrane system. This example shows a dynamic membrane system, involving not only the transportation of molecules and entire compartments but also the creation and dissolution of membranes. In the following, we describe the modeling and analysis of this membrane system using colored Petri nets.

Modeling

The viral infection process is illustrated in Figure 5.26, together with the changes of the membrane structure. At the beginning, the virus consists of a viral RNA wrapped by the capsid, which is further enclosed in the envelope. If a virus encounters a healthy cell, it will enter the healthy cell and then will be wrapped by a vesicle membrane. After that, the vesicle and envelope membranes dissolve and release the capsid, which disassembles itself into the viral RNA and C proteins. Now the viral RNA begins to act through three distinct paths. First, it replicates itself and produces more copies of the viral RNA. Second, it is translated into proteins, some of which contribute to the generation of the capsid, enclosing the viral RNA. Finally, the newly assembled capsid buds out to regenerate a new virus, which continues to infect other healthy cells. See [SMC⁺08] for more details.

In the following, we briefly recall how to model the viral infection process using membrane systems according to [SMC⁺08].

(1) Objects.

In this process, three kinds of objects are involved: healthy cells, viral RNA and Protein C.

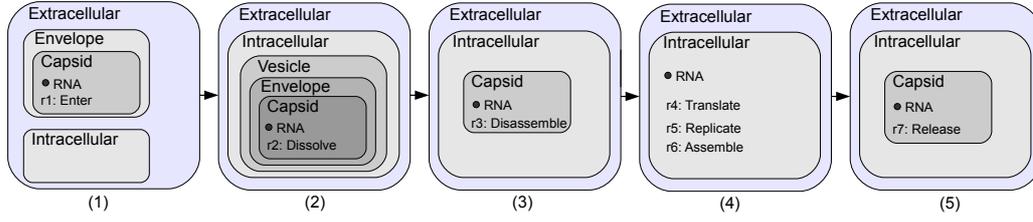


Figure 5.26: The viral infection process together with the changes of the membrane structure (taking one virus and one healthy cell as an example).

(2) The membrane structure.

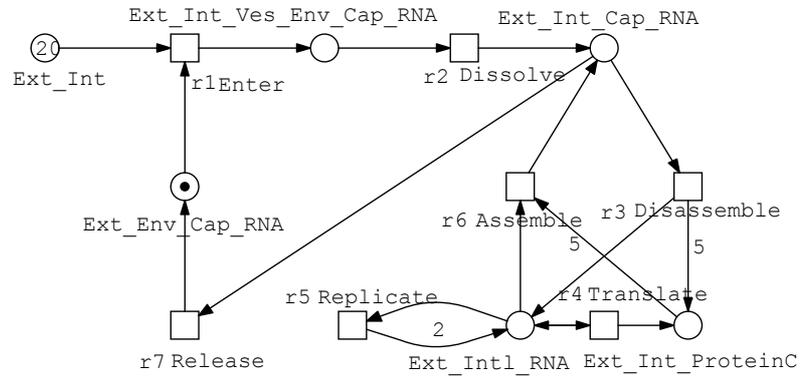
The membranes (compartments) that are involved are as follows. The extracellular (shortly "Ext") environment provides a space that cells and viruses live in. A virus in the extracellular is wrapped by the capsid (shortly "Cap"), which is enclosed by the envelope (shortly "Env"). In the intracellular (shortly "Int") of an infected cell, the virus is further wrapped by the vesicle (shortly "Ves"). During the infection process, the membrane structure will change according to Figure 5.26.

(3) Evolution rules.

The infection process uses the following rules (for simplicity, we omit the extracellular membrane for each rule):

- $r1: \{\{RNA\}_{Cap}\}_{Env}, Cell \xrightarrow{c^1} \{\{\{\{RNA\}_{Cap}\}_{Env}\}_{Ves}\}_{Int},$
- $r2: \{\{\{\{\{RNA\}_{Cap}\}_{Env}\}_{Ves}\}_{Int} \xrightarrow{c^2} \{\{RNA\}_{Cap}\}_{Int},$
- $r3: \{\{RNA\}_{Cap}\}_{Int} \xrightarrow{c^3} \{RNA, 5'ProteinC\}_{Int},$
- $r4: \{RNA\}_{Int} \xrightarrow{c^4} \{RNA, ProteinC\}_{Int},$
- $r5: \{RNA\}_{Int} \xrightarrow{c^5} \{2'RNA\}_{Int},$
- $r6: \{RNA, 5'ProteinC\}_{Int} \xrightarrow{c^6} \{\{RNA\}_{Cap}\}_{Int},$
- $r7: \{\{RNA\}_{Cap}\}_{Int} \xrightarrow{c^7} \{\{RNA\}_{Cap}\}_{Env}.$

According to these evolution rules, we first build a stochastic Petri net model, illustrated in Figure 5.27. From this model we can see that we have to represent each object, e.g. viral RNA, at each compartment as a place of the Petri net.

Figure 5.27: A *SPN* model for the viral infection.

Further, we build a colored stochastic Petri net model for the viral infection, illustrated in Figure 5.28. This membrane system involves the transportation of entire compartments, so we can not differentiate them only using their labels. Rather we have to combine their labels with the labels of their father compartments. Thus, we define a color set CS with four colors: Ext , Ext_Int , Ext_Env_Cap , $Ext_Int_Ves_Env_Cap$ and Ext_Int_Cap . We use a place to represent a kind of objects at different locations. For example, we use place RNA to represent the viral RNA at four compartments. We use arc expressions to indicate the location change of an object. For the two dissolution rules, $r2$ and $r3$, the involved membranes when dissolved become empty (only one object for each membrane) and do not affect other rules, so we just deal with them as ordinary rules.

Analysis

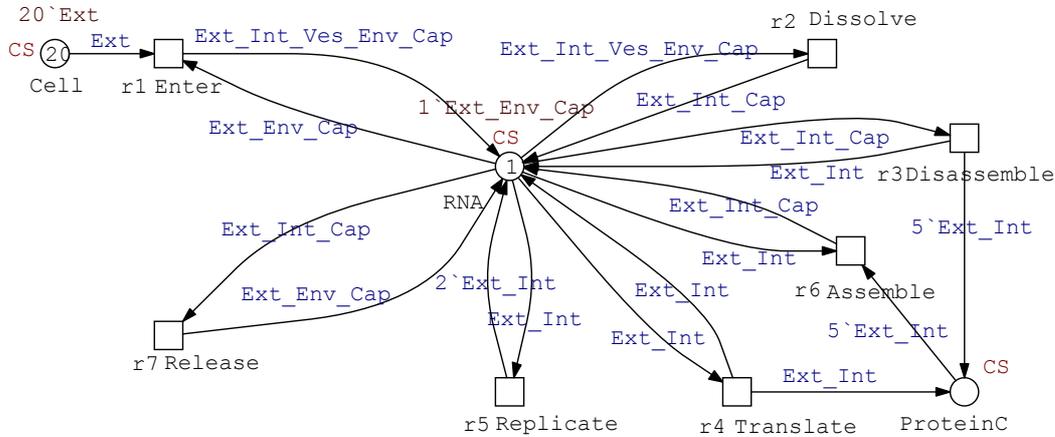
In the following, we conduct analysis on the model in 5.28 in three ways: structural analysis, simulation and model checking.

(1) Structural analysis.

In order to exploit basic properties this net enjoys, We first use our structural analysis tool, Charlie [Cha11], to analyze its P- and T- invariants. Unfortunately, this net is covered by neither P-invariants nor T-invariants. The only T-invariant of this net is composed by $r3$ and $r6$, which is as we expect.

(2) Simulation analysis.

We use the Gillespie's stochastic simulation algorithm [Gil77] to simulate our colored stochastic Petri net model. As we can not obtain quantitative data, we set all the stochastic reaction constants $c1$ to $c7$ to the same value of 1.0. For comparisons with

Figure 5.28: A SPN^C model for the virus infection.

the results in [SMC⁺08], we also set the initial marking to 20 healthy cells and 1, 10, 100 virus molecules and we obtain the simulation results shown in Figure 5.29, Figure 5.30 and Figure 5.31, respectively. As expected, the simulation results show that the number of viruses in environment increases exponentially with time, which are consistent with the results in [SMC⁺08].

(3) Simulative model checking.

We can also support formal analysis of membrane systems using model checking. Here we will demonstrate how to use a simulative model checking tool, the MC2 tool [DG08], to check some properties of the viral infection model. For the demonstration purpose, we give the following queries to be checked and generate a set of 100 stochastic (single) simulation traces.

$$(1) P_{=?}[time > 9 \rightarrow G([Ext_Int_RNA] > [Ext_Env_Cap_RNA])]$$

After some time, the viruses in the environment become more than those in the infected cells. The probability for this query as computed by the model checking tool is 0.87.

$$(2) P_{=?}[G([Ext_Int_ProteinC] < 15)]$$

The number of Protein C in the cells remains steady (always less than a constant). The probability for this query is 0.95.

From the colored Petri net model above, we can see that it represents each object at different locations as one place. Therefore, with the increase of objects and compartments in membrane systems, colored Petri nets provide very compact models. Besides, we can make full use of analysis techniques of Petri nets to analyze membrane systems.

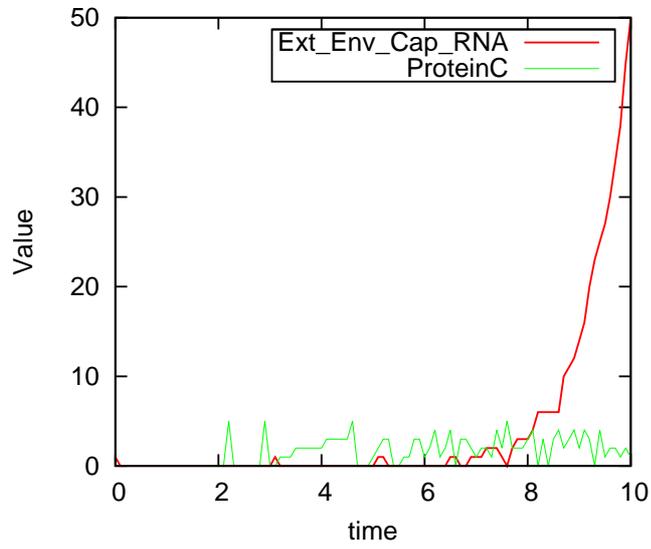


Figure 5.29: Stochastic simulation result of one simulation run for the virus infection model with 1 initial virus molecule.

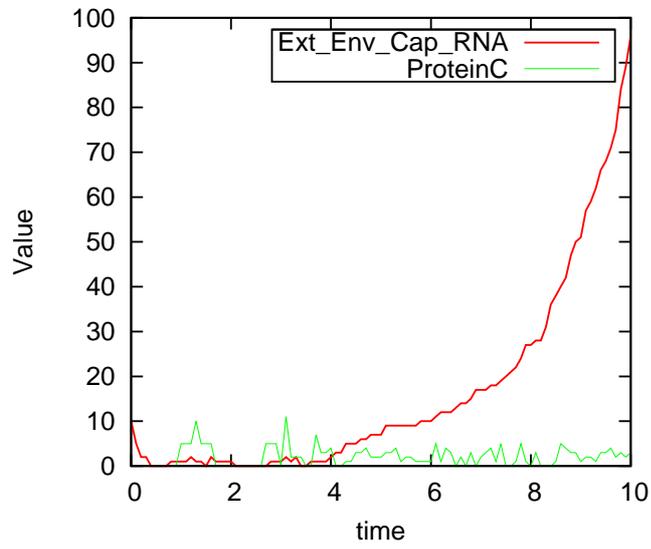


Figure 5.30: Stochastic simulation result of one simulation run for the virus infection model with 10 initial virus molecules.

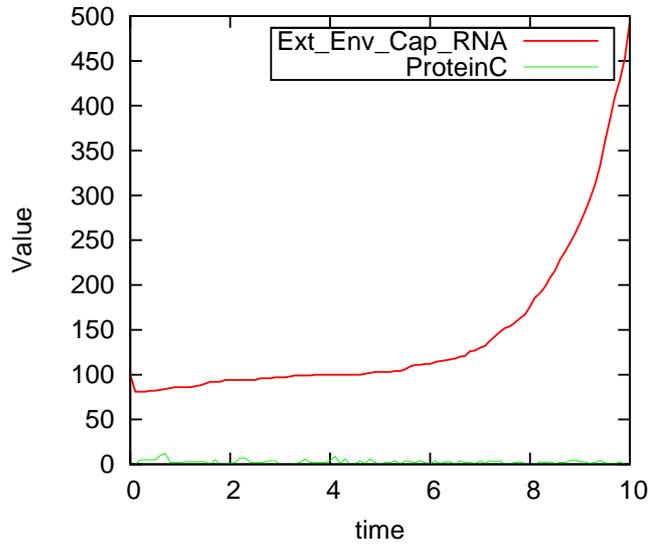


Figure 5.31: Stochastic simulation result of one simulation run for the virus infection model with 100 initial virus molecules.

5.3.5 Conclusions

In this section, we have described how to model and analyze membrane systems using Petri nets and especially colored Petri nets. By encoding each membrane as a color, colored Petri nets provide a compact representation for a membrane system in a graphical way and more importantly they allow a lot of analysis techniques. In summary, colored Petri nets are a powerful tool for modeling and analyzing membrane systems. In the future, we will use hierarchical color sets to provide a more clear representation of the structure of membrane systems. We also want to automatically create Petri net and colored Petri net models for membrane systems of a specific format.

5.4 Closing Remarks

In this chapter, we have given three case studies, each of which has its specific purpose. The first two mainly demonstrate how to use colored Petri nets to address some of the challenges faced with by systems biology given in Chapter 1. But the first case study has other important purposes, i.e. to demonstrate how to model and analyze a system from three perspectives: qualitative, stochastic and continuous, and to combine a variety of analysis techniques to analyze a system. The third case study, however, explores an advanced application of colored Petri nets, modeling membrane systems using colored Petri nets. The purpose of this case study is trying to find more scenarios to apply

colored Petri nets.

With the rapid development of systems biology, no doubt colored Petri nets will gain more and more attention and find more application scenarios.

6 Conclusions and Outlook

6.1 Conclusions

This thesis presents a technology based on colored Petri nets and associated techniques to address challenges introduced by multiscale modeling in systems biology and offer a solution to perform multiscale modeling and analysis of biological systems.

To accomplish this aim, in Chapter 2 we have first presented a colored Petri net framework for systems biology, which relates three modeling paradigms: QPN^C , SPN^C and CPN^C . Using this framework, we can model and analyze a biological system from three perspectives: qualitative, stochastic and continuous by transforming them into each other. This allows these three formalisms to work together to achieve the modeling and analysis of biological systems.

We have implemented this framework in our Petri net tool Snoopy, and in Chapter 3 we have discussed three aspects concerning the implementation of colored Petri nets. We have first presented an efficient algorithm for the computation of enabled transition instances in order to animate/simulate colored Petri nets, in which we have adopted a pattern matching mechanism and a new partial binding - partial test principle and considered some optimization techniques to improve the computation efficiency. We have then provided an efficient unfolding algorithm for colored Petri nets, in which we have offered two approaches to efficiently compute transition instances. That is, if the color set of each variable in a guard is a finite integer domain, the constrain satisfaction approach has been used to obtain all legal bindings; otherwise, a general algorithm has been adopted, in which some optimization techniques, e.g. the partial binding - partial test principle, have been used. Please note that the main difference between the computation of enabled transition instances for animation/simulation and the computation of transition instances for unfolding is that the former computes enabled transition instances in terms of available tokens (current marking) on places while the latter computes transition instances in terms of color sets of places. In addition, we have considered three special scenarios for automatic folding: colorizing T-invariants, master nets and twin nets in order to reduce the amount of manual work used for folding Petri nets. Among them, colorizing T-invariants contributes to the further understanding of T-invariants for a biological network, and colorizing master nets or twin nets offers a convenient way for reconstructing biological networks.

Petri nets offer a large variety of analysis techniques ranging from informal techniques,

e.g. animation/simulation to formal techniques, e.g. model checking. In Chapter 4, we have summarized those analysis techniques that can be used for colored Petri nets and payed attention to applying them for analyzing colored Petri nets.

Finally, in Chapter 5, we have given three case studies, *C. elegans* vulval development, coupled Ca^{2+} channels and membrane systems, to explore the application of our colored Petri net technology and techniques. These case studies not only demonstrate how to apply the colored Petri net framework and related analysis techniques to modeling and analyzing practical biological systems, but also show how to address the challenges in systems biology given in Chapter 1.

In summary, this thesis provides a solution based on colored Petri nets to model and analyze biological systems.

6.2 Outlook

This thesis explores theories and application of colored Petri nets for systems biology. There are a number of potential areas for future research:

1. We will continue to improve our colored Petri net modeling tool, Snoopy, by including more features beneficial to computational modeling of biological systems.
2. In Chapter 3, we have included an unfolding algorithm for colored Petri nets in order to cope with large-scale biological models. But with the increasing size of biological systems, it is necessary to improve this algorithm to tackle more complicated and larger biological systems in the future.
3. We will explore automatic folding based on subgraph isomorphism so as to automatically colorize arbitrarily given Petri nets. For this purpose, we will consider the particular characteristics of Petri nets to offer an efficient folding algorithms based on subgraph isomorphism.
4. As discussed in Chapter 4, colored Petri nets can be analyzed at the colored (folded) level, thus we can utilize the characteristics of colored Petri nets without generating their corresponding unfolded Petri nets. In a next step, we will investigate possible approaches to achieve this.
5. We will look into partial unfolding to tackle dynamic color sets so as to address such biological phenomena as compartment creation, division, merging or dissolving, or cell differentiation.
6. We have applied colored Petri nets to some scenarios in the context of systems biology, whereas its application is far beyond these. Therefore, we will continue to discover other challenges from biological modeling and more various ranges of scenarios in which colored Petri nets can be applied.

7. So far, we have discussed the application of colored Petri nets to systems biology. In the future we will explore more application areas, e.g. synthetic biology, where there are a number of similar challenges like repetition of components and (hierarchical) organization of components.
8. In this thesis, we focus on the colored Petri net technology and associated techniques for the modeling of biological systems; in the next step we will generalize our experiences to a methodology so as to provide guidelines for multiscale modeling of biological systems.

Bibliography

- [Ade05] ADEREM, A.: Systems Biology: Its Practice and Challenges. In: *Cell* 121 (2005), Nr. 4, pp. 511–513 {1}
- [AK76] ARAKI, T.; KASAMI, T.: Some Decision Problems Related to the Reachability Problem for Petri Nets. In: *Theoretical Computer Science* 3 (1976), Nr. 1, pp. 85–104 {23}
- [ASSB00] AZIZ, A.; SANWAL, K.; SINGHAL, V. ; BRAYTON, R.: Model Checking Continuous Time Markov Chains. In: *ACM Transactions on Computational Logic* 1 (2000), Nr. 1, pp. 162–170 {79, 83, 84}
- [BCMS10] BALDAN, P.; COCCO, N.; MARIN, A. ; SIMEONI, M.: Petri Nets for Modelling Metabolic Pathways: a Survey. In: *Natural Computing* 9 (2010), Nr. 4, pp. 955–989 {1, 2, 4, 80}
- [BCP08] BLOSSEY, R.; CARDELLI, L. ; PHILLIPS, A.: Compositionality, Stochasticity and Cooperativity in Dynamic Models of Gene Regulation. In: *HFSP Journal* 2 (2008), Nr. 1, pp. 17–28 {6}
- [BDGH10] BREITLING, R.; DONALDSON, R.; GILBERT, D. ; HEINER, M.: Biomodel Engineering - From Structure to Behavior. In: *Transactions on Computational Systems Biology XII, Special Issue on Modeling Methodologies*, Springer, 2010 (LNCS 5945), pp. 1–12 {4}
- [Ber97] BERRIDGE, M. J.: Elementary and Global Aspects of Calcium Signalling. In: *Journal of Experimental Biology* 200 (1997), Nr. 2, pp. 315–319 {109}
- [BHHK03] BAIER, C.; HAVERKORT, B.; HERMANN, H. ; KATOEN, J. P.: Model-checking Algorithms for Continuous-Time Markov Chains. In: *IEEE Transactions on Software Engineering* 29 (2003), Nr. 6, pp. 524–541 {83}
- [BKF⁺09] BONZANNI, N.; KREPSKA, E.; FEENSTRA1, K. A.; FOKKINK, W.; KIELMANN, T.; BAL, H. ; HERINGA1, J.: Executing Multicellular Differentiation: Quantitative Predictive Modelling of C. Elegans Vulval Development. In: *Bioinformatics* 25 (2009), Nr. 16, pp. 2049–2056 {95}
- [BKK95] BAUSE, F.; KEMPER, P. ; KRITZINGER, P.: Abstract Petri Net Notation. In: *Petri Net Newsletter* (1995), Nr. 49, pp. 9–27 {41}

- [BMD07] BAER, C. F.; MIYAMOTO, M. M. ; DENVER, D. R.: Mutation Rate Variation in Multicellular Eukaryotes: Causes and Consequences. In: *Nature Reviews Genetics* 8 (2007), pp. 619–631 {11}
- [BP03] BAHJ-JABER, N.; PONTIER, D.: Modeling Transmission of Directly Transmitted Infectious Diseases Using Colored Stochastic Petri Nets. In: *Mathematical Biosciences* 185 (2003), pp. 1–13 {2}
- [BPS99] BRAILSFORD, S. C.; POTTS, C. N. ; SMITH, B. M.: Constraint Satisfaction Problems: Algorithms and Applications. In: *European Journal of Operational Research* 119 (1999), Nr. 3, pp. 557–581 {61}
- [CBW08] CHANNON, K.; BROMLEY, E. H. ; WOOLFSON, D. N.: Synthetic Biology through Biomolecular Design and Engineering. In: *Current Opinion in Structural Biology* 18 (2008), Nr. 4, pp. 1–8 {4}
- [CCFS06] CALZONE, L.; CHABRIER-RIVIER, N.; FAGES, F. ; SOLIMAN, S.: Machine Learning Biochemical Networks from Temporal Logic Properties. In: *Transactions on Computational Systems Biology*, Springer, 2006 (LNCS 4220), pp. 68–94 {19}
- [CDFH93] CHIOLA, G.; DUTHEILLET, C.; FRANCESCHINIS, G. ; HADDAD, S.: Stochastic Well-Formed Coloured Nets for Symmetric Modelling Applications. In: *IEEE Transactions on Computers* 42 (1993), Nr. 11, pp. 1343–1360 {7}
- [CDFH97] CHIOLA, G.; DUTHEILLET, C.; FRANCESCHINIS, G. ; HADDAD, S.: A Symbolic Reachability Graph for Coloured Petri Nets. In: *Theoretical Computer Science* 176 (1997), Nr. 1-2, pp. 39–65 {91, 92}
- [CE81] CLARKE, E. M.; EMERSON, E. A.: Design and Synthesis of Synchronization Skeletons using branching Time Temporal Logic. In: *Proc. of the Workshop on Logic of Programs*, Springer, 1981 (LNCS 131), pp. 52–71 {79, 80, 81}
- [CES86] CLARKE, E.; EMERSON, E.; ; SISTLA, A.: Automatic Verification of Finite-State Concurrent Systems Using Temporal Logics. In: *ACM Transactions on Programming Languages and Systems* 8 (1986), Nr. 2, pp. 244–263 {83}
- [CFG92] CHIOLA, G.; FRANCESCHINIS, G. ; GAETA, R.: A Symbolic Simulation Mechanism for Well-Formed Coloured Petri Nets. In: *Proc. of the 25th annual symposium on Simulation*, 1992, pp. 192–201 {91, 92}

-
- [CGP01] CLARKE, E. M.; GRUMBERG, O. ; PELED, D. A.: *Model Checking*. Cambridge: MIT Press, 2001 {18, 81, 82}
- [Cha07] CHAOUIYA, C.: Petri Net Modelling of Biological Networks. In: *Briefings in Bioinformatics* 8 (2007), Nr. 4, pp. 210–219 {23}
- [Cha11] CHARLIE. *Charlie - A tool for the Analysis of Place/Transition Petri Nets*. <http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Charlie>. 2011 {69, 82, 115, 136}
- [CJK97] CHRISTENSEN, S.; JØRGENSEN, J. B. ; KRISTENSEN, L. M.: Design/CPN - A Computer Tool for Coloured Petri Nets. In: *Proc. of the Third International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, Springer, 1997 (LNCS 1217), pp. 209–223 {2}
- [Coo71] COOK, S. A.: The Complexity of Theorem-Proving Procedures. In: *Proc. of the 3rd ACM Symposium on Theory of Computing*, ACM, 1971, pp. 151–158 {69}
- [CW85] CARDELLI, L.; WEGNER, P.: On Understanding Types, Data Abstraction, and Polymorphism. In: *Computing Survey* 17 (1985), Nr. 4, pp. 471–522 {7}
- [Dal10] DALLON, J. C.: Multiscale Modeling of Cellular Systems in Biology. In: *Current Opinion in Colloid and Interface Science* 15 (2010), Nr. 1-2, pp. 24–31 {3, 10}
- [DG08] DONALDSON, R.; GILBERT, D.: A Model Checking Approach to the Parameter Estimation of Biochemical Pathways. In: *Proc. of the 6th International Conference on Computational Methods in Systems Biology*, Springer, 2008 (LNCS 5307), pp. 269–287 {18, 19, 79, 84, 85, 86, 101, 137}
- [DLKS08] DEREMIGIO, H.; LAMAR, M. D.; KEMPER, P. ; SMITH, G. D.: Markov Chain Models of Coupled Calcium Channels: Kronecker Representations and Iterative Solution Methods. In: *Physical Biology* 5 (2008), Nr. 3, pp. 1–14 {11, 111, 114, 119}
- [DS05] DEREMIGIO, H.; SMITH, G. D.: The Dynamics of Stochastic Attrition Viewed as an Absorption Time on a Terminating Markov Chain. In: *Cell Calcium* 38 (2005), Nr. 2, pp. 73–86 {112, 118, 119}
- [DWW10] DURZINSKY, M.; WAGLER, A. ; WEISMANTEL, R.: An Algorithmic Framework for Network Reconstruction. In: *Theoretical Computer Science* 8 (2010), Nr. 16, pp. 1–16 {73, 74}

- [EL00] ELOWITZ, M. B.; LEIBLER, S.: A Synthetic Oscillatory Network of Transcriptional Regulators. In: *Nature* 403 (2000), pp. 335–338 {6, 36}
- [FA73] FLYNN, M. J.; AGERWALA, T.: Comments on Capabilities, Limitations and Correctness of Petri nets. In: *Proc. of the 1st Annual Symposium on Computer Architecture*, ACM, 1973, pp. 81–86 {22}
- [Fal03] FALCKE, M.: On the Role of Stochastic Channel Behavior in Intracellular Ca^{2+} Dynamics. In: *Biophysical Journal* 84 (2003), Nr. 1, pp. 42–56 {114, 122}
- [FHL⁺04] FINKELSTEIN, A.; HETHERINGTON, J.; LI, L.; MARGONINSKI, O.; SAFREY, P.; SEYMOUR, R. ; WARNER, A.: Computational Challenges of Systems Biology. In: *Computer* 37 (2004), Nr. 5, pp. 26–33 {1}
- [FPHH07] FISHER, J.; PITERMAN, N.; HAJNAL, A. ; HENZINGER, T. A.: Predictive Modeling of Signaling Crosstalk during *C. Elegans* Vulval Development. In: *PLoS Computational Biology* 3 (2007), Nr. 5, pp. e92 {95}
- [Fra09] FRANZKE, A.: *Charlie 2.0 – a Multi-Threaded Petri Net Analyzer*, Computer Science Department, Brandenburg University of Technology Cottbus, Diplomarbeit, December 2009 {80, 99}
- [Gae96] GAETA, R.: Efficient Discrete-Event Simulation of Colored Petri Nets. In: *IEEE Transactions on Software Engineering* 22 (1996), Nr. 9, pp. 629–639 {49, 51, 55, 56}
- [GAS⁺06] GALLE, J.; AUST, G.; SCHALLER, G.; BEYER, T. ; DRASDO, D.: Individual Cell-Based Models of the Spatial-Temporal Organization of Multicellular Systems - Achievements and Limitations. In: *Cytometry* 69A (2006), Nr. 7, pp. 704–710 {11}
- [GCPL⁺98] GARCIA-CALVO, M.; PETERSON, E. P.; LEITING, B.; RUEL, R.; NICHOLSON, D. W. ; THORNBERRY, N. A.: Inhibition of Human Caspases by Peptide-Based and Macromolecular Inhibitors. In: *Journal Biological Chemistry* 273 (1998), Nr. 49, pp. 32606–32613 {22}
- [Gec11] GECODE. *Gecode: An Open Constraint Solving Library*. <http://www.gecode.org>. 2011 {61}
- [GH06] GILBERT, D.; HEINER, M.: From Petri Nets to Differential Equations - an Integrative Approach for Biochemical Network Analysis. In: *Proc. of the 27th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency*, Springer, 2006 (LNCS 4024), pp. 181–200 {1}

-
- [GH11] GILBERT, D.; HEINER, M. *Petri nets for multiscale Systems Biology*. <http://multiscalepn.brunel.ac.uk>. 2011 {15}
- [GHG02] GONZE, D.; HALLOY, J. ; GOLDBETER, A.: Deterministic versus Stochastic Models of Circadian Rhythms. In: *Journal of Biological Physics* 28 (2002), Nr. 4, pp. 637–653 {26, 28, 36}
- [GHL07] GILBERT, D.; HEINER, M. ; LEHRACK, S.: A Unifying Framework for Modelling and Analysing Biochemical Pathways Using Petri Nets. In: *Proc. of the 5th International Conference on Computational Methods in Systems Biology*, Springer, 2007 (LNCS 4695), pp. 200–216 {17, 19, 33, 118}
- [Gil77] GILLESPIE, D. T.: Exact Stochastic Simulation of Coupled Chemical Reactions. In: *Journal of Physical Chemistry* 81 (1977), Nr. 25, pp. 2340–2361 {41, 110, 118, 127, 136}
- [GKV01] GENRICH, H.; KÜFFNER, R. ; VOSS, K.: Executable Petri Net Models for the Analysis of Metabolic Pathways. In: *International Journal on Software Tools for Technology Transfer* 3 (2001), Nr. 4, pp. 394–404 {2}
- [GL79] GENRICH, H. J.; LAUTENBACH, K.: The Analysis of Distributed Systems by Means of Predicate/Transition-Nets. In: *Proc. of the International Symposium on Semantics of Concurrent Computation*, Springer, 1979 (LNCS 70), pp. 123–146 {2, 7}
- [GL81] GENRICH, H. J.; LAUTENBACH, K.: System Modelling with High-Level Petri Nets. In: *Theoretical Computer Science* 13 (1981), Nr. 1, pp. 109–135 {2, 7}
- [GLG⁺11] GAO, Q.; LIU, F.; GILBERT, D.; HEINER, M. ; TREE, D.: A Multiscale Approach to Modelling Planar Cell Polarity in *Drosophila* Wing using Hierarchically Coloured Petri Nets. In: *Proc. of the 9th International Conference on Computational Methods in Systems Biology (CMSB 2011)*, ACM digital library, September 2011 {3, 11, 15, 16, 36}
- [GLTG11] GAO, Q.; LIU, F.; TREE, D. ; GILBERT, D.: Multi-cell Modeling Using Coloured Petri Nets Applied to Planar Cell Polarity. In: *Proc. of the 2th International Workshop on Biological Processes & Petri nets* Volume 724, 2011, pp. 135–150 {15, 16, 36}
- [GMR08] GALVÃO, V.; MIRANDA, J. G. V. ; RIBEIRO-DOS-SANTOS, R.: Development of a Two-Dimensional Agent-Based Model for Chronic Chagasic Cardiomyopathy after Stem Cell Transplantation. In: *Bioinformatics* 24 (2008), Nr. 18, pp. 2051–2056 {12}

- [GMRC10] GHEORGHE, M.; MANCA, V. ; ROMERO-CAMPERO, F. J.: Deterministic and Stochastic P Systems for Modelling Cellular Processes. In: *Natural Computing* 9 (2010), Nr. 2, pp. 457–473 {124, 127}
- [GP98] GOSS, P. J. E.; PECCOUD, J.: Quantitative Modeling of Stochastic Systems in Molecular Biology by Using Stochastic Petri Nets. In: *The Proceedings of the National Academy of Sciences USA* 95 (1998), Nr. 12, pp. 6750–6755 {1, 28}
- [Gre11] GREATSPN. *GreatSPN, a Software Package for the Modeling, Validation, and Performance Evaluation of Distributed Systems Using Generalized Stochastic Petri Nets and Their Colored Extension: Stochastic Well-Formed Nets*. <http://www.di.unito.it/greatspn/index.html>. 2011 {91}
- [Gri08] GRIMA, R.: Multiscale Modeling of Biological Pattern Formation. In: *Current Topics in Developmental Biology* 81 (2008), pp. 435–460 {13}
- [GSA06] GIURUMESCU, C. A.; STERNBERG, P. W. ; ASTHAGIRI, A. R.: Intercellular Coupling Amplifies Fate Segregation during Caenorhabditis Elegans Vulval Development. In: *The Proceedings of the National Academy of Sciences USA* 103 (2006), Nr. 5, pp. 1331–1336 {95}
- [Had87] HADDAD, S.: *Une Catégorie Régulière de Réseau de Petri de Haut Niveau: Définition, Propriétés et réductions*, Paris: Université P. et M. Curie (Paris 6), PhD thesis, 1987 {7}
- [HGBT09] HWANG, M.; GARBAY, M.; BERCELI, S. A. ; TRAN-SON-TAY, R.: Rule-Based Simulation of Multi-Cellular Biological Systems - A Review of Modeling Techniques. In: *Cellular and Molecular Bioengineering* 2 (2009), Nr. 3, pp. 285–294 {11}
- [HGD08] HEINER, M.; GILBERT, D. ; DONALDSON, R.: Petri Nets for Systems and Synthetic Biology. In: *Proc. of the 8th international conference on Formal methods for computational systems biology*, Springer, 2008 (LNCS 5016), pp. 215–264 {1, 6, 22, 28, 29, 69, 79, 80}
- [HJ94] HANSSON, H.; JONSSON, B.: A Logic for Reasoning about Time and Reliability. In: *Formal Aspects of Computing* 6 (1994), Nr. 5, pp. 512–535 {83}
- [HK09] HEATH, A. P.; KAVRAKI, L. E.: Computational Challenges in Systems Biology. In: *Computer Science Review* 3 (2009), Nr. 1, pp. 1–17 {1}

-
- [HKV01] HEINER, M.; KOCH, I. ; VOSS, K.: Analysis and Simulation of Steady States in Metabolic Pathways with Petri Nets. In: *Proc. of the 3rd Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, University of Aarhus, 2001, pp. 15–34 {2}
- [HLGM09] HEINER, M.; LEHRACK, S.; GILBERT, D. ; MARWAN, W.: Extended Stochastic Petri Nets for Model-Based Design of Wetlab Experiments. In: *Transaction on Computational Systems Biology XI LNBI 5750* (2009), pp. 138–163 {29, 30, 31, 32, 33, 34}
- [HRSS10] HEINER, M.; ROHR, C.; SCHWARICK, M. ; STREIF, S.: A Comparative Study of Stochastic Analysis Techniques. In: *Proc. of the 8th International Conference on Computational Methods in Systems Biology*, ACM, 2010, pp. 96–106 {87}
- [HS10] HEINER, M.; SRIRAM, K.: Structural Analysis to Determine the Core of Hypoxia Response Network. In: *PLoS ONE* 5 (2010), Nr. 1, pp. e8600 {71}
- [IGH01] IDEKER, T.; GALITSKI, T. ; HOOD, L.: A New Approach to Decoding Life: Systems Biology. In: *Annual Review of Genomics and Human Genetics* 2 (2001), pp. 343–372 {9}
- [Ila01] ILACHINSKY, A.: *Cellular Automata*. World Scientific Publishing, 2001 {36}
- [IWL06] IDEKER, T.; WINSLOW, L. R. ; LAUFFENBURGER, A. D.: Bioengineering and Systems Biology. In: *Annals of Biomedical Engineering* 34 (2006), Nr. 2, pp. 257–264 {1, 9}
- [Jen81] JENSEN, K.: Coloured Petri Nets and the Invariant-Method. In: *Theoretical Computer Science* 14 (1981), Nr. 3, pp. 317–336 {2, 7, 19, 22, 26}
- [Jen92] JENSEN, K.: *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Vol 1, Basic Concepts*. Berlin Heidelberg: Springer, 1992 {58}
- [Jen95] JENSEN, K.: *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Vol 2, Analysis Methods*. Berlin Heidelberg: Springer, 1995 {91}
- [Jen96] JENSEN, K.: Condensed State Spaces for Symmetrical Coloured Petri Nets. In: *Formal Methods in System Design* 9 (1996), Nr. 1-2, pp. 7–40 {91}

- [JK09] JENSEN, K.; KRISTENSEN, L.M.: *Coloured Petri Nets*. Springer, 2009 {9, 19, 91}
- [JKT⁺01] JANOWSKI, S.; KORMEIER, B.; TÖPEL, T.; HIPPE, K.; HOFESTÄDT, R.; WILLASSEN, N.; FRIESEN, R.; RUBERT, S.; BORCK, D.; HAUGEN, P. ; CHEN, M.: Modeling of Cell-to-Cell Communication Processes with Petri Nets Using the Example of Quorum Sensing. In: *In Silico Biology* 10 (2001), pp. 0003 {12}
- [JKW07] JENSEN, K.; KRISTENSEN, L. M. ; WELLS, L. M.: Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. In: *International Journal on Software Tools for Technology Transfer* 9 (2007), Nr. 3/4, pp. 213–254 {8, 9, 21, 42}
- [KB71] KONOPKA, R. J.; BENZER, S.: Clock Mutants of *Drosophila melanogaster*. In: *The Proceedings of the National Academy of Sciences USA* 68 (1971), Nr. 9, pp. 2112–2116 {11}
- [KC04] KRISTENSEN, L. M.; CHRISTENSEN, S.: Implementing Coloured Petri Nets Using a Functional Programming Language. In: *Higher-Order and Symbolic Computation* 17 (2004), Nr. 3, pp. 207–243 {42, 43, 44, 45, 46, 51, 53, 56}
- [Kit02] KITANO, H.: Systems Biology: A Brief Overview. In: *Science* 295 (2002), Nr. 5560, pp. 1662–1664 {1, 9}
- [KK09] KLEIJN, J.; KOUTNY, M.: A Petri Net Model for Membrane Systems with Dynamic Structure. In: *Natural Computing* 8 (2009), Nr. 4, pp. 781–796 {124, 127, 130, 131}
- [KKR06] KLEIJN, J.; KOUTNY, M. ; ROZENBERG, G.: Towards a Petri Net Semantics for Membrane Systems. In: *Proc. of the 6th International Workshop on Membrane Computing*, Springer, 2006 (LNCS 3850), pp. 292–309 {124, 127, 128}
- [KLPA06] KORDON, F.; LINARD, A. ; PAVIOT-ADET, E.: Optimized Colored Nets Unfolding. In: *Proc. of the 26th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems*, Springer, 2006 (LNCS 4229), pp. 339–355 {66, 68}
- [KNP09] KWIATKOWSKA, M.; NORMAN, G. ; PARKER, D.: PRISM: Probabilistic Model Checking for Performance and Reliability Analysis. In: *ACM SIGMETRICS Performance Evaluation Review* 36 (2009), Nr. 4, pp. 40–45 {83, 84}

-
- [Kwi03] KWIATKOWSKA, M.: Model Checking for Probability and Time: from Theory to Practice. In: *Proc. of the 18th IEEE Symposium on Logic in Computer Science*, IEEE, 2003, pp. 351–360 {18}
- [LH10a] LIU, F.; HEINER, M.: Colored Petri nets to model and simulate biological systems. In: *Proc. of International Workshop on Biological Processes and Petri Nets, satellite event of Petri Nets 2010*, 2010 {4, 16, 110}
- [LH10b] LIU, F.; HEINER, M.: Computation of Enabled Transition Instances for Colored Petri Nets. In: *Proc. of the 17th German Workshop on Algorithms and Tools for Petri Nets* Volume 643, CEUR-WS.org, 2010, pp. 51–65 {16, 42, 122}
- [LH11] LIU, F.; HEINER, M.: Manual for Colored Petri Nets in Snoopy / Brandenburg University of Technology Cottbus, Department of Computer Science. 2011. – Technical Report. http://www-dssz.informatik.tu-cottbus.de/software/snoopy/Manual_for_colored_Petri_nets_2011_07.pdf {15, 41}
- [LLZ11] LIANG, J.; LUO, Y. ; ZHAO, H.: Synthetic Biology: Putting Synthesis into Biology. In: *Systems Biology and Medicine* 3 (2011), Nr. 1, pp. 7–20 {4}
- [LNUM09] LI, C.; NAGASAKI, M.; UENO, K. ; MIYANO, S.: Simulation-Based Model Checking Approach to Cell Fate Specification During *Caenorhabditis Elegans* Vulval Development by Hybrid Functional Petri Net with Extension. In: *BMC Systems Biology* 3 (2009), pp. 42 {xv, 36, 94, 95, 96, 98, 99, 101, 103, 104, 108, 109}
- [LSK09] LAMPRECHT, R.; SMITH, G. D. ; KEMPER, P.: Stochastic Petri Net Models of Ca^{2+} Signaling Complexes and Their Analysis. In: *Natural Computing* (2009), 6 {110, 114, 118, 119}
- [Lun65] LUND, E. W.: Guldberg and Waage and the Law of Mass Action. In: *Journal of Chemical Education* 42 (1965), Nr. 10, pp. 548 {32}
- [LZLP06] LEE, D.; ZIMMER, R.; LEE, S. ; PARK, S.: Colored Petri Net Modeling and Simulation of Signal Transduction Pathways. In: *Metabolic Engineering* 8 (2006), Nr. 2, pp. 112–122 {2}
- [MBC⁺95] MARSAN, M. A.; BALBO, G.; CONTE, G.; DONATELLI, S. ; FRANCESCHINIS, G.: *Modelling with Generalized Stochastic Petri Nets*. John Wiley and Sons, 1995 (Wiley Series in Parallel Computing) {79}

- [MFV10] MARTINS, M. L.; FERREIRA JR., S. C. ; VILELA, M. J.: Multiscale Models for Biological Systems. In: *Current Opinion in Colloid & Interface Science* 15 (2010), Nr. 1-2, pp. 18–23 {3, 10, 13, 36}
- [Mäk01] MÄKELÄ, M.: Optimising Enabling Tests and Unfoldings of Algebraic System Nets. In: *Proc. of the 22nd International Conference on Application and Theory of Petri Nets*, 2001 (LNCS 2075), pp. 283–302 {49, 55, 56, 68}
- [MKH⁺06] MÜLLER, J.; KUTTLER, C.; HENSE, B. A.; ROTHBALLER, M. ; HARTMANN, A.: Cell-Cell Communication by Quorum Sensing and Dimension-Reduction. In: *Journal of Mathematical Biology* 53 (2006), Nr. 4, pp. 672–702 {12}
- [Mor01] MORTENSEN, K. H.: Efficient Data-Structures and Algorithms for a Coloured Petri Nets Simulator. In: *Proc. of the 3rd Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, University of Aarhus, 2001, pp. 57–74 {56}
- [Mor06] MORGAN, D. O.: *The Cell Cycle: Principles of Control*. New Science Press, 2006 {13}
- [MR95] MONTANARI, U.; ROSSI, F.: Contextual Nets. In: *Acta Informatica* 32 (1995), Nr. 6, pp. 545–596 {22}
- [MSFK09] MEIER-SCHELLERSHEIM, M.; FRASER, I. D. C. ; KLAUSCHEN, F.: Multi-scale Modeling for Biologists. In: *Systems Biology and Medicine* 1 (2009), pp. 4–14 {10}
- [MSS05] MARWAN, W.; SUJATHA, A. ; STAROSTZIK, C.: Reconstructing the Regulatory Network Controlling Commitment and Sporulation in *Physarum Polycephalum* Based on Hierarchical Petri Net Modeling and Simulation. In: *Journal of Theoretical Biology* 236 (2005), Nr. 4, pp. 349–365 {37, 75}
- [MTS05] MAZZAG, B.; TIGNANELLI, C. ; SMITH, G. D.: The Effect of Residual Ca^{2+} on the Stochastic Gating on Ca^{2+} -Regulated Ca^{2+} Channel Models. In: *Journal of Theoretical Biology* 235 (2005), Nr. 1, pp. 121–150 {110}
- [Mur89] MURATA, T.: Petri Nets: Properties, Analysis and Applications. In: *Proceedings of the IEEE* 77 (1989), Nr. 4, pp. 541–578 {6}
- [MWW08] MARWAN, W.; WAGLER, A. ; WEISMANTEL, R.: A Mathematical Approach to Solve the Network Reconstruction Problem. In: *Mathematical Methods of Operations Research* 67 (2008), Nr. 1, pp. 117–132 {71, 73}

-
- [MWW11] MARWAN, W.; WAGLER, A. ; WEISMANTEL, R.: Petri Nets as a Framework for the Reconstruction and Analysis of Signal Transduction Pathways and Regulatory Networks. In: *Natural Computing* 10 (2011), Nr. 2, pp. 639–654 {2, 17, 89}
- [NJT⁺05] NELSON, C. M.; JEAN, R. P.; TAN, J. L.; LIU, W. F.; SNIADOCKI, N. J.; SPECTOR, A. A. ; CHEN, C. S.: Emergent Patterns of Growth Controlled by Multicellular Form and Mechanics. In: *Proceedings of National Academy of Sciences of the United States of America* 102 (2005), Nr. 33, pp. 11594–11599 {11}
- [NMS05] NGUYEN, V.; MATHIAS, R. ; SMITH, G. D.: A Stochastic Automata Network Descriptor for Markov Chain Models of Instantaneously Coupled Intracellular Ca^{2+} Channels. In: *Bulletin of Mathematical Biology* 67 (2005), Nr. 3, pp. 393–432 {36, 110, 112}
- [Pau99] PAUN, G.: Computing with Membranes. An Introduction. In: *Bulletin of the EATCS* 67 (1999), pp. 139–152 {36, 123, 125}
- [Pau02] PAUN, G.: *Membrane Computing, An Introduction*. Springer, 2002 {123, 124, 125, 126, 127}
- [Pet62] PETRI, C. A.: *Kommunikation mit Automaten*, Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, PhD thesis, 1962 {4}
- [Pet11] PETRI NETS. *Petri Nets World*. <http://www.informatik.uni-hamburg.de/TGI/PetriNets>. 2011 {42}
- [PGA02] PELEG, M.; GABASHVILI, I. S. ; ALTMAN, R. B.: Qualitative Models of Molecular Function: Linking Genetic Polymorphisms of tRNA to Their Functional Sequelae. In: *Proceedings of the IEEE* 90 (2002), Nr. 12, pp. 1875–1886 {2, 11}
- [PH09] POPEL, A. S.; HUNTER, P. J.: Systems Biology and Physiome Projects. In: *Wiley Interdisciplinary Reviews: Systems Biology and Medicine* 1 (2009), Nr. 2, pp. 153–158 {9, 10}
- [Pnu81] PNUELI, A.: The Temporal Semantics of Concurrent Programs. In: *Theoretical Computer Science* 13 (1981), Nr. 1, pp. 45–60 {19, 80, 81, 85}
- [PO00] PALSSON, E.; OTHMER, H.: A Model for Individual and Collective Cell Movement in Dictyostelium D iscoideum. In: *Proceedings of National Academy of Sciences of the United States of America* 97 (2000), Nr. 19, pp. 10448–10453 {11, 12}

- [PRA05] PELEG, M.; RUBIN, D. ; ALTMAN, R. B.: Using Petri Net Tools to Study Properties and Dynamics of Biological Systems. In: *Journal of the American Medical Informatics Association* 12 (2005), Nr. 2, pp. 181–199 {1, 28}
- [PSQH06] POGSONA, M.; SMALLWOODA, R.; QWARNSTROMB, E. ; HOLCOMBE, M.: Formal Agent-Based Modelling of Intracellular Chemical Interactions. In: *Biosystems* 85 (2006), Nr. 1, pp. 37–45 {12}
- [PUS11] PARSA, H.; UPADHYAY, R. ; SIA, S. K.: Uncovering the Behaviors of Individual Cells within a Multicellular Microvascular Community. In: *Proceedings of National Academy of Sciences of the United States of America* 108 (2011), Nr. 12, pp. 5133–5138 {11}
- [QJY04] QI, Z.; J. YOU, H. M.: P systems and Petri nets. In: *Proc. of the 5th International Workshop of Membrane Computing*, Springer, 2004 (LNCS 2933), pp. 286–303 {124, 130, 131}
- [RMH10] ROHR, C.; MARWAN, W. ; HEINER, M.: Snoopy - a Unifying Petri Net Framework to Investigate Biomolecular Networks. In: *Bioinformatics* 26 (2010), Nr. 7, pp. 974–975 {4, 23, 41}
- [RML93] REDDY, V. N.; MAVROVOUNIOTIS, M. L. ; LIEBMAN, M. N.: Petri Net Representations in Metabolic Pathways. In: *Proc. of the 1st International Conference on Intelligent Systems for Molecular Biology*, AAAI Press, 1993, pp. 328–336 {1}
- [Run04] RUNGE, T.: Application of Coloured Petri Nets in Systems Biology. In: *Proc. of the 5th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, University of Aarhus, 2004, pp. 77–95 {2}
- [RWL⁺03] RATZER, A. V.; WELLS, L.; LASSEN, H. M.; LAURSEN, M.; QVORTRUP, J. F.; STISSING, M. S.; WESTERGAARD, M.; CHRISTENSEN, S. ; JENSEN, K.: CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets. In: *Proc. of the 24th International Conference on Applications and Theory of Petri Nets*, Springer, 2003, pp. 450–462 {2, 79, 88}
- [San00] SANDERS, M. J.: Efficient Computation of Enabled Transition Bindings in High-Level Petri Nets. In: *Proc. of IEEE International Conference on Systems, Man and Cybernetics*, IEEE, 2000, pp. 3153–3158 {56}
- [SF05] SNEYD, J.; FALCKE, M.: Models of the Inositol Trisphosphate Receptor. In: *Progress in BioPhysics and Molecular Biology* 89 (2005), Nr. 3, pp. 207–245 {109, 122}

-
- [SH89] STERNBERG, P. W.; HORVITZ, H. R.: The Combined Action of Two Intercellular Signaling Pathways Specifies Three Cell Fates during Vulval Induction in *C. Elegans*. In: *Cell* 58 (1989), Nr. 4, pp. 679–693 {95}
- [SH09] SCHWARICK, M.; HEINER, M.: CSL Model Checking of Biochemical Networks with Interval Decision Diagrams. In: *Proc. of the 7th International Conference on Computational Methods in Systems Biology*, Springer, 2009 (LNCS 5688), pp. 296–312 {84}
- [SM08] SANDMANN, W.; MAIER, C.: On the Statistical Accuracy of Stochastic Simulation Algorithms Implemented in Dizzy. In: *Proc. of the 5th International Workshop on Computational Systems Biology*, 2008, pp. 153–157 {103}
- [SMC⁺08] SPICHER, A.; MICHEL, O.; CIESLAK, M.; GIAVITTO, J. ; PRUSINKIEWICZ, P.: Stochastic P Systems and the Simulation of Biochemical Processes with Dynamic Compartments. In: *BioSystems* 91 (2008), Nr. 3, pp. 458–472 {127, 134, 137}
- [Smi02] SMITH, G. D.: Modeling the Stochastic Gating of Ion Channels. In: *Computational Cell Biology* 20 (2002), pp. 285–319 {111}
- [SOM10] STREIF, S.; OESTERHELT, D. ; MARWAN, W.: A Predictive Computational Model of the Kinetic Mechanism of Stimulus-Induced Transducer Methylation and Feedback Regulation through CheY in Archaeal Phototaxis and Chemotaxis. In: *BMC Systems Biology* 4 (2010), Nr. 27 {11}
- [SRH11] SCHWARICK, M.; ROHR, C. ; HEINER, M.: MARCIE - Model Checking and Reachability Analysis Done efficiently. In: *Proc. of the 8th International Conference on Quantitative Evaluation of Systems*, IEEE, 2011, pp. 91–100 {82, 83, 119, 122}
- [ST11] SCHWARICK, M.; TOVCHIGRECHKO, A.: IDD-based model validation of biochemical networks. In: *Theoretical Computer Science* 412 (2011), Nr. 26, pp. 2884–2908 {111, 120}
- [SW05] SNOEP, J. L.; WESTERHOFF, H. V.: From Isolation to Integration, a Systems Biology Approach for Building the Silicon Cell. In: *Topics in Current Genetics* 13 (2005), pp. 13–30 {1}
- [Ter06] TERRIER, V.: Closure Properties of Cellular Automata. In: *Theoretical Computer Science* 352 (2006), Nr. 1-3, pp. 97–107 {38}

- [TMK⁺06] TÄUBNER, C.; MATHIAK, B.; KUPFER, A.; FLEISCHER, N. ; ECKSTEIN, S.: Modelling and Simulation of the TLR4 Pathway with Coloured Petri Nets. In: *Proc. of the 28th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, IEEE, 2006, pp. 2009–2012 {2}
- [Tsa93] TSANG, E. P. K.: *Foundations of Constraint Satisfaction*. London and San Diego: Academic Press, 1993 {61}
- [Ull98] ULLMAN, J. D.: *Elements of ML Programming*. Prentice-Hall, 1998 {43}
- [VHK03] VOSS, K.; HEINER, M. ; KOCH, I.: Steady State Analysis of Metabolic Pathways Using Petri Nets. In: *In Silico Biology 3* (2003), pp. 0031 {2}
- [VKBL02] VILAR, J.; KUEH, H. Y.; BARKAI, N. ; LEIBLER, S.: Mechanisms of Nois-eresistance in Genetic Oscillators. In: *Proceedings of National Academy of Sciences of the United States of America* 99 (2002), Nr. 9, pp. 5988–5992 {28}
- [Wat09] VAN DER WATH, R. C.: *Computational Modelling of Hematopoietic Stem Cell Division and Regulation Dynamics*, University of Cambridge, PhD thesis, 2009 {13}
- [YBG04] YOO, A.; BAIS, C. ; GREENWALD, I.: Crosstalk Between the EGFR and LIN-12/Notch pathways in *C. Elegans* Vulval Development. In: *Science* 303 (2004), Nr. 5658, pp. 663–66 {95}
- [YKNP06] YOUNES, H. L. S.; KWIATKOWSKA, M.; NORMAN, G. ; PARKER, D.: Numerical vs. Statistical Probabilistic Model Checking. In: *International Journal on Software Tools for Technology Transfer* 8 (2006), Nr. 3, pp. 216–228 {84}