République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA

N° d'ordre :.....
Série :.....

Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département d'Informatique

THÈSE

Présentée pour obtenir le diplôme de

DOCTORAT EN SCIENCES EN INFORMATIQUE

Par

Samir TIGANE

THÈME

Reconfiguration in Stochastic Petri Nets

Soutenue le : 30/06/2020
Devant le jury composé de :

Pr. Kazar Okba	**Président**	Professeur à l'Université de Biskra.
Pr. Kahloul Laïd	**Rapporteur**	Professeur à l'Université de Biskra.
Dr. Baarir Souheib	**Co-Rapporteur**	Maitre de Conférences Habilité à l'Université Paris Ouest Nanterre.
Pr. Bennoui Hamadi	**Examinateur**	Professeur à l'Université de Biskra.
Pr. Chaoui Allaoua	**Examinateur**	Professeur à l'Université de Constantine 2.
Pr. Saidouni Djemal Eddine	**Examinateur**	Professeur à l'Université de Constantine 2.

ملخص

في أيامنا المعاصرة، أصبحت العديد من أنظمة الأحداث المنفصلة معقدة بشكل متزايد و ذات بنية ديناميكية ومترابطة بشكل متغاير. فلقد تم تصميم هذه الأنظمة لتتمكن من تغيير بنيتها، في وقت التشغيل، عن طريق إضافة أو إزالة بعض الأجزاء، لتمكينها من التكيف مع الظروف والمتطلبات الجديدة. كأداة رسمية، استخدام شبكات بتري في دراسة مثل هذه الأنظمة يجذب العديد من الباحثين.

على الرغم من أن شبكات بتري تمثل أداة قوية، إلا أنها غير قادرة على التحديد و التحقق بطريقة سلسة من الأنظمة المتقدمة ذات البنية الديناميكية. في الواقع، الأنظمة التي تدعم البيئات المتقلبة، والتغيرات المستمرة، والهياكل القابلة لإعادة التشكيل معقدة للغاية. للتغلب على هذه المشكلة، أثرى العديد من الباحثين شبكات بتري بخاصية إعادة التشكيل. ومع ذلك، فإن زيادة قوة النمذجة يقلل من قوة القرار. في الواقع، العديد من الخصائص تصبح غير قابلة للتقصي.
لذلك، تحاول الإضافات المقترحة التي تدخل إعادة التشكيل إلى شبكات بتري إيجاد حل وسط بين مستويات النمذجة والتحقق .

في هذه الأطروحة، سنقوم بتقديم ثلاثة طرق - تم تطويرها تدريجياً - للنمذجة والتحقق من إعادة التكوين في شبكات بيتري العشوائية المعممة (**GSPNs**)، كل منها له مزاياه وحدوده واتجاهه، مع الحفاظ على إمكانية التحقق من العديد من الخصائص مع تعقيد أقل.

أولاً، نقترح شكلية، تسمى **GSPNs** مع طبولوجيا قابلة لإعادة الكتابة (**GSPNs-RT**)، والتي تسمح بنمذجة طوبولوجيا ديناميكية وتحويل **GSPNs-RT** إلى **GSPNs** مكافئة، من أجل الاستفادة الكاملة من أدوات التحقق الجاهزة.
بعد ذلك، نقترح شكلية أخرى، تسمى **GSPNs** الديناميكية (**D-GSPNs**)، والتي تسمح بالنمذجة والتحقق من الهياكل الديناميكية (حيث أن مجموعات الأماكن والانتقالات ديناميكية) وبتحويل **D-GSPNs** إلى **GSPNs** مكافئة. يمكن لهذه التحويلات أن تحدث عندما تمتلك النماذج الأصلية عددا محددا من التكوينات.
أخيرًا، تطور **GSPNs** إلى شكلية قابلة لإعادة التشكيل، تسمى **GSPNs** القابلة لإعادة التكوين (**RecGSPNs**)، والتي تدعم مجموعة أكبر من التغييرات الهيكلية بما هو مسموح به في الأساليب الحالية، وكذلك، يسمح لأي **GSPN** قابلة للتكوين بالحصول على عدد لا حصر له من التكوينات مع الحفاظ على قابلية التحقق من العديد من الخصائص الهامة.

الكلمات المفتاحية: شبكات بتري العشوائية المعممة - النماذج و الهياكل الديناميكية - أنظمة تحويل النماذج - شبكات بتري القالبة لإعادة الهيلكة - النمذجة و التحقق الشكلي - تقييم الأداء.

Abstract

Nowadays, many discrete event systems (DESs) are becoming increasingly complex, structurally dynamic and variably interconnected. These systems are designed to be able to change their structure and/or topology, at run-time, to accommodate new circumstances/requirements. As a formal tool, the use of Petri nets (PNs) in the study of such systems attracts many researchers.

Although PNs (low or high) are a powerful and expressive tool, they are unable to specify/verify, in a natural way, advanced systems having dynamic structures. Indeed, systems supporting volatile environments, continuous variations, and reconfigurable structures are expected to be extremely complex. To overcome this issue, researchers enrich PNs with reconfigurability. Nevertheless, increasing the modeling power of a formalism decreases its decision power. In fact, several properties become undecidable. Therefore, extensions proposed in the literature introducing reconfigurability to PNs try to find a compromise between the modeling and the verification levels.

In this thesis, we describe three approaches – incrementally developed – for the modeling and verification of reconfiguration in generalized stochastic Petri nets (GSPNs), while maintaining verifiability of several properties with reduced complexity.

First, we propose a formalism, called GSPNs with rewritable topology (GSPNs-RT), that extends GSPNs by allowing modeling dynamic topologies and transforming GSPNs-RT to equivalent GSPNs, to take full advantages of off-the-shelf tools proposed for GSPNs verification.

Then, we propose another formalism, called dynamic GSPNs (D-GSPNs), that allows modeling and verifying dynamic structures (sets of places and transitions are dynamic) and transforms D-GSPNs to equivalent GSPNs. This transformation can take place when the original model disposes of a finite number of configurations.

Finally, GSPNs are extended to a reconfigurable formalism, called reconfigurable GSPNs (RecGSPNs), that supports a wider range of possible structural changes than allowed in existing approaches, as well as, allows to any RecGSPN to have an infinite number of configurations while preserving the decidability of several important properties.

Keywords: Generalized stochastic Petri nets; Dynamic model and structure; Graph transformation systems; Formal modeling and verification; Performance evaluation.

Résumé

De nos jours, de nombreux systèmes à événements discrets deviennent de plus en plus complexes, structurellement dynamiques et interconnectés de manière variable. Ces systèmes sont conçus pour pouvoir modifier leur structure et/ou leur topologie, au moment de l'exécution, afin de s'adapter à des nouvelles circonstances/exigences. En tant qu'outil formel, l'utilisation de réseaux de Petri dans l'étude de tels systèmes attire de nombreux chercheurs.

Bien que les réseaux de Petri (bas ou haut niveau) constituent un outil puissant et expressif, ils ne sont pas en mesure de spécifier/vérifier, de manière naturelle, des systèmes avancés dotés de structures dynamiques. En effet, les systèmes prenant en charge des environnements volatils, des variations continues et des structures reconfigurables devraient être extrêmement complexes. Pour surmonter ce problème, les chercheurs enrichissent les réseaux de Petri avec la reconfigurabilité. Néanmoins, augmenter le pouvoir de modélisation d'un formalisme diminue son pouvoir de décision. En fait, plusieurs propriétés deviennent indécidables. Par conséquent, les extensions proposées dans la littérature introduisant la reconfigurabilité dans les réseaux de Petri tentent de trouver un compromis entre les niveaux de modélisation et de vérification.

Dans cette thèse, on définit trois approches – développées incrémentalement – pour la modélisation et la vérification des réseaux de Petri stochastiques généralisés (GSPNs) reconfigurables, tout en maintenant la vérifiabilité de plusieurs propriétés avec une complexité réduite.

En premier lieu, on propose un formalisme, appelé GSPNs avec topologie modifiable (GSPNs-RT), qui étend les GSPNs en permettant la modélisation de topologies dynamiques et la transformation de GSPNs-RT en GSPNs équivalents, afin de tirer pleinement d'avantages des outils standard proposés pour la vérification des GSPNs.

Ensuite, on propose un autre formalisme, appelé GSPNs dynamiques (D-GSPNs), qui permet de modéliser des structures dynamiques (les ensembles de places et de transitions sont dynamiques) et transforme les D-GSPNs en GSPNs équivalents. Cette transformation peut avoir lieu lorsque le modèle original disposent d'un nombre fini de configurations.

Enfin, on étend les GSPNs à un formalisme reconfigurable, appelé GSPNs reconfigurable (RecGSPNs), qui prend en charge une plus large gamme de changements structurels possibles que ce qui est autorisé dans les approches existantes, ainsi que, permet à tout GSPN reconfigurable d'avoir un nombre infini de configurations tout en préservant la décidabilité de plusieurs propriétés importantes.

Mot-clés : Réseaux de Petri stochastiques généralisés ; Modèle et structure dynamiques; Système de transformation de graphes ; Modélisation et vérification formelles ; Évaluation de performance.

Dedication

This thesis is dedicated to....
My beloved mother for her great support, love, pray, and care.
My valuable treasures in my life: family and friends.

Acknowledgment

I would like to express my sincere gratitude to my supervisors Laid Kahloul and Souheib Baarir for their constant guidance, support, and encouragement.

I have to thank also all the team in **LIP6** (Paris 6) in which I had the chance to pass one year during the finalization of this thesis.

Special thanks to Prof. Kazar Okba who always offers his unlimited support to the members of **LINFI** laboratory.

Finally, I would like also to present my thanks to the members of the jury who give me the honor by accepting to evaluate and review this work.

Contents

List of Figures

List of Tables

Introduction

In this introduction, we start by presenting the background of this thesis, namely the formal modeling and verification based on dynamic-structure stochastic Petri nets. Then, we focus on the motivations of this work, specify the problem and the objectives, and highlight our contributions. Finally, we end with the description of the manuscript organization.

Background

The major advancements in computing power, connectivity, sensors, storage capacity, and software development have motivated companies as well as individuals to adopt and integrate IT solutions into their daily tasks (from a small device at the house to gigantic infrastructure).

As the list of advantages and benefits resulting from this orientation continues to grow, so do the risks of failure and malfunctioning that may threaten companies as well as individuals.

Therefore, it is absolutely mandatory to ensure the proper functioning of computer systems according to customers' and designers' expectations. This concern had been identified since the late 60s that marked the birth of software engineering. One of the main goals of the latter is enabling developers to implement complex systems that work properly. In this regard, several approaches have emerged to meet this crucial requirement in the various stages of the software life cycle [Woo+09].

Approaches in which the syntax, semantics, and manipulation rules of specification language are explicitly defined by mathematics, are called *formal approaches*. These approaches include: Petri nets[Mur89], sequential process communication (SPC) [BHR84], LOTOS [BB87], B-method [Abr05], etc.

Actually, formal approaches allow a complete verification of the whole system behavior and proving the presence of certain desired properties for all possible inputs. By formal methods, one can write a formal specification of a system on which different properties can be proved, and thereafter one can mathematically prove that a system implementation meets this specification [CGR93, Hax10, GG13, RK15]. However, the use of formal methods does not guarantee a priori the accuracy of developed systems. Indeed, their use enhances our understanding of a system under construction while revealing its shortcomings, inconsistencies and ambiguities that might otherwise go unnoticed [CW96].

Among the most widespread formalisms, we find Petri nets (PNs) [Pet77, Mur89]. They are characterized by three major advantages [Pet81, KV86]:

1. **Modeling level**: They have a powerful mathematical foundation, as well as, an intuitive graphical representation. The graphical representation gives a flat view to PN models, making it possible to have simple and very explicit models. As well, their graphic modeling enables easy visualization of complex systems,

2. **Verification level**: Their mathematical foundation is at the origin of all the analysis techniques that were proposed in order to verify the modeled systems. Indeed, they dispose of a well-developed qualitative/quantitative analysis panoply,

3. **Coupling modeling and verification**: They offer a careful balance of modeling and decision power. In fact, Petri nets have been used in the modeling of a wide variety of systems. As for their decision power, the reachability problem is decidable in Petri nets (note that most problems can be converted into reachability problems).

The model, in its origin proposed by "Adam Petri" in [Pet62] was initially concerned with describing the causal relationships between events that can be occurred in a computer system, has known significant evolution and adaptation to meet several requirements imposed by the appearance of new complicated systems. The most notable extensions can be found in four main categories:

- Colored PNs [Jen13]: Each token becomes rather a distinguishable value from other tokens. The weights on the arcs are no longer constants, but rather mathematical functions that can be complicated. This model makes it possible to have models of reasonable size for complicated systems,

- Temporal PNs [Ram73]: The time introduced in PNs allows to put explicit constraints on the dynamics of the model which reflect the real temporal constraints imposed on the system,

- Stochastic PNs [Mar+94]: Stochastic PNs are a response to another missing realistic aspect in previous models, which is the aspect of hazard, randomness, and non-absolute events. The random events in their arrivals will be explicitly considered and modeled, allowing the model to get closer to the real system and thus to have a good representation of the studied system,

- Reconfigurable PNs [LO04b, EP04, PK18]: This last category groups formalisms allowing the modeling of structure flexibility.

The purpose of this last variant is to provide a formal model for dynamic-structure systems, e.g., flexible manufacturing systems (FMSs) [BS80], reconfigurable manufacturing systems (RMSs) [Kor+99], production in cloud [Xu12], Industry 4.0 [Las+14], etc.

In fact, many discrete event systems (DESs) are becoming increasingly complex, structurally dynamic and variably interconnected. These systems are designed to be able to change their structure and/or topology, at run-time, by adding/removing interconnections, objects, or even subsystems, to accommodate new circumstances/requirements.

Ongoing studies on this class of systems focus on their key feature, namely, the reconfigurability [Bre+14] that must occur at run-time (i.e., dynamic reconfigurability) [JEB16]. Dynamic reconfigurability is a critical activity that influences the performance, security and cost of such systems. To overcome the previous challenges, the designer must dispose of a rigorous approach and a set of appropriate tools.

The use of PNs in the study of such systems attracts many researchers [SV90, Mar+94, Rec+04, Che+17, LZB17, Lat+18, Liu+18, You+18]. In the literature, many classes of PNs have been proposed and applied to specify/verify reconfigurable systems. The chosen PN class is often motivated by the aspects to be specified and the properties to be verified. In fact, we can distinguish three classes of work: that uses basic PNs, that uses temporal or stochastic PNs, and finally, that applies reconfigurable PNs.

Stochastic PNs (SPNs) and generalized SPNs (GSPNs) represent an extension of PNs [Mar+94] used to model and evaluate stochastic systems. These formalisms allow the analysis of performance metrics such as productivity, energy consumption, machine utilization, etc.

Marsan *et al.* [Mar+94] strongly emphasize the importance of GSPNs and SPNs as versatile design tools that fit well with the behavior of DESs at different stages of development [LZB15, Čap17, Sim+18, Lat+18].

Although PNs (low or high) are a powerful and expressive tool, they are unable to specify/verify, in a natural way, advanced dynamic-structure systems [CC18]. Systems supporting volatile environments, continuous variations, and reconfigurable structures are expected to be extremely complex [Chr+13]. The design of such systems is an increasingly complex and omnipresent challenge. Therefore, designers must dispose of the necessary approaches, models, and tools to handle this complexity [Möl16]. To overcome this issue, researchers introduce dynamic structures into PNs, thus expanding the standard formalism [PK18].

On the other hand, rule-based graph transformations [EP04] offer a mathematically-based graphical framework for modeling the reconfigurations in PN structures. Nevertheless, increasing the modeling power of a formalism decreases its decision power. Therefore, extensions proposed in the literature introducing reconfigurability to PNs try to find a compromise between the modeling and the verification levels. From this perspective, we can distinguish three main directions.

On one hand, researchers develop pre-processing techniques that encode, unfold or compile graphs and transformation rules into existing formalisms in order to exploit the panoply of their tools [LO04b, CC18, CBC18]. Although they can naturally model the reconfigurations, these approaches did not increase the modeling power comparing to existing ones, since they depend on target formalisms expressiveness and in particular do not allow modeling infinite graphs [RSV04]. For instance, classical model-checkers [BK08] use a fixed number of propositions, which prevents the modeling of infinite-structure systems [Ren08].

On the other hand, some techniques execute graph transformation systems and compute the reachability graph, nevertheless an upper artificial threshold is still needed [KR06]. To mitigate this issue, some approaches compute either under-approximations of a system's behavior, so that any property that holds in an under-approximated model is satisfied in its original system, or over-approximations including all system behaviors, and possibly more [CR12]. Nevertheless, a property that does not hold in an under-estimated model may holds in its original system and a property that holds in an over-estimated model may not be satisfied in its original system [BCK08].

A promising approach described in [Li+09], called improved net rewriting system (INRS), preserves particular properties, namely, liveness, boundedness, and reversibility of PNs after each reconfiguration. These properties are therefore decidable regardless of the number of obtained configurations. However, INRSs are limited to (i) ordinary, live, bounded and reversible PNs, and (ii) particular forms of reconfiguration.

Motivation and Objectives

Exception for few work [MD97] and [Cap17], reconfigurability in either SPNs or GSPNs has not received much attention. Existing approaches in the literature often focus on performance evaluation of reconfigurable systems using SPNs (which are not reconfigurable formalisms) or on reconfigurability simulation and verification using reconfigurable PNs (which do not consider stochastic aspects).

Our goal in this thesis is to introduce reconfigurability into the well-known GSPNs formalism using graph transformation systems. The objective of this thesis is doubled:

- Integration of the dynamic aspect in GSPNs: this requires a formal definition of a new formalism combining the stochastic and the dynamic aspects in a single formalism,
- Study of this hybridization consequences on GSPNs analysis: introducing reconfigurability into GSPNs requires adapting the classical algorithms towards the new formalisms and/or proposing new analysis techniques.

Contributions

The present thesis falls within the field of reconfigurable system modeling and verification based on GSPNs, hence the need for a GSPNs-based formalism supporting dynamic structures. However, increasing the modeling power of a formalism involves straightforward increasing the verification complexity. Faced with these challenges, we have defined five approaches – incrementally developed – for the modeling and the verification of dynamic-structure GSPNs, each of which has its advantages, limits and orientation.

Orientation 1: Transforming dynamic GSPNs into GSPNs

In the first place, we were interested in the modeling and the verification of reconfigurable systems using (non-stochastic) reconfigurable Petri nets. With this in mind, our first intention was to propose an extension for one of these reconfiguration approaches to the stochastic ones.

Our primary contributions in this direction were reconfigurable SPNs [TKB17b] and GSPNs with rewritable topology [TKB17a], extending formalism presented in [LO04b], to cope with reconfigurability in SPNs and GSPNs, respectively.

However, both proposed formalisms allow only dynamic topology, i.e., sets of places and transitions cannot be changed. By fixing both sets of places and transitions, we can transform SPNs and GSPNs having dynamic topologies into basic SPNs and GSPNs, respectively, by encompassing all topologies in one model. In the latter, the switching between configurations and appearing/disappearing/reappearing of topologies are modeled via the token game.

The transformation into basic SPNs or GSPNs straightforwardly allows exploiting the methods and techniques proposed in the literature for SPNs and/or GSPNs to verify the properties of dynamic ones. However, allowing only dynamic topologies limits severely the modeling power of both formalisms.

To overcome this shortcoming, we propose a new formalism, called dynamic generalized stochastic Petri nets (D-GSPNs), that allows to model dynamic sets of places and transitions, as well as, keeping the possibility to transform D-GSPNs into GSPNs for verification purposes. The obtained GSPNs preserve the stochastic behaviors of dynamic GSPNs, allowing the use of the panoply of verification methods and tools proposed for GSPNs in D-GSPNs analysis.

Orientation 2: Preserving properties in reconfigurable generalized stochastic Petri nets (RecGSPNs)

Although D-GSPNs allow natural modeling and verification of dynamic structure, they do not increase the modeling power of GSPNs. In fact, the dynamic structure of any D-GSPN must be finite, otherwise transforming D-GSPNs towards GSPNs may become infinite.

To consider infinite structures, we extended INRSs [Li+09] to model reconfiguration in GSPNs and could publish two papers [TKB17c, TKB16]. In these two extensions, each reconfiguration is expressed by a rule having left- and right-hand sides. The application of a rule implies the substitution of its left-hand side image, in a given GSPN to be reconfigured, by its right-hand side. These two sides must belong to particular sets in order to allow developers to reconfigure a live, bounded and reversible GSPN while preserving these three essential properties in the resulting model.

However, these formalisms suffer from three major drawbacks: (i) system states are not considered (reconfigurations are done in an off-line mode), (ii) only three qualitative properties, namely liveness, boundedness and reversibility are decidable, and finally, (iii) the quantitative aspect of GSPNs is not studied.

To remedy these problems, we have proposed a new formalism, called INRSs-GSPNs [Tig+18], which takes into account, inter alia, system states in the reconfiguration application and provides an algorithm for both qualitative and quantitative verifications.

At the modeling level, designers will be able to model a reconfigurable system and its dynamic structure using GSPNs and rewriting rules that are controlled by the system state. Unlike our extensions described in [TKB17c, TKB16] which limit rule application to an initial marking, we associate each reconfiguration rule with a marking controlling its activation, that is, if the net has not yet reached a marking then the rule is not yet applicable. As for the verification level, an algorithm computing from the dynamic model the Markov chain describing the stochastic behavior of the system is proposed. As a result, the designers can evaluate the system performance.

Nevertheless, the reconfiguration remains too limited in INRSs-GSPNs formalism. On the one hand, only live, bounded and reversible GSPNs are concerned which limits its application field. On the other hand, left- and right-hand sides of rules must belong imperatively to a particular set which can only further limit the formalism applicability.

The need to (i) relax the constraints imposed by INRSs-GSPNSs formalism, (ii) address all types of GSPNs (not only live, bounded and reversible GSPNs), and (iii) enrich the set of nets used in reconfiguration, led us to propose reconfigurable generalized stochastic Petri nets (RecGSPNs) [Tig+19].

Actually, RecGSPNs formalism allows designers to model a wider range of structural changes where both sides of any rule are no longer defined by their structure. Instead, they are defined by their behaviors. The use of RecGSPNs based reconfiguration allows preserving five important properties, namely, liveness, boundedness, reversibility, deadlock-freedom and home state. Moreover, many properties expressed by linear time logic [BK08] can be preserved after a system reconfiguration. Thus, these properties are decidable whatever the number of obtained configurations that can be infinite.

This practice enjoys double advantages, such that:

1. Temporal and spatial complexity are reduced since these properties are verified only at the first configuration, hence no need to compute and explore the whole set of reachable states of all reachable configurations,

2. Often, applying rules to graphs leads to structurally infinite models, and hence properties are not decidable based on classical verification techniques. In RecGSPNs approach, several properties are still decidable since applying any rule preserve them.

Thesis Organization

This chapter has introduced the thesis topic, stated research motivation and objectives, and given an overview of our contributions. The remainder of this thesis is organized as follows.

Chapter 2 presents the background theory of Petri nets, model checking, Markov chains and generalized stochastic Petri nets. In order to introduce generalized stochastic Petri net, we describe Petri nets in their basic form, as well as, both model checking and Markov chains basic definitions are provided. Finally, the extension of Petri nets toward GSPNs is presented.

Chapter 3 introduces the graph transformation field and its use in PNs context. Initially, graph transformation systems are outlined. Then, we focus on graph transformation applications to PNs in the literature. As for verification aspects, we discuss some proposed verification algorithms obtained by either developing new ones or updating and transferring existing ones to graph transformation systems. Finally, we conclude by showing the advantages/disadvantages of today's formalisms and their verification techniques proposed for dynamic structure PNs.

Chapter 4 describes one of our primary contributions, namely GSPNs-RT [TKB17a], in which we present a trivial approach that introduces dynamic topologies into GSPNs, as well as, transforms GSPNs with dynamic topologies into equivalent basic GSPNs. These equivalent GSPNs are exploited in the verification of dynamic nets using classical analysis approaches.

Chapter 5 develop an extension of GSPNs-RT, namely D-GSPNs, in which the sets of places and transitions are dynamic and can be transformed using transformation rules formalized by DPO-approach. As well, this formalism is equipped by an algorithm that transforms D-GSPNs into equivalent GSPNs.

Chapter 6 consists of our major contribution in this thesis. It describes a new approach, called reconfigurable GSPNs [Tig+19]. Similarly to graph transformation systems, it formalizes system configuration as GSPNs, as well as, it describes structure evolution as transformation rules. The chapter concludes with a discussion about the decidability of some

properties of dynamic systems even if they can be structurally infinite, i.e., the reachable configuration set is infinite.

In **Chapter 7**, we compare the proposed approaches with the current state-of-the-art. We start showing new qualitative modeling aspects provided by RecGSPNs and D-GSPNs formalisms. Then, a quantitative comparison that shows how the proposed approaches optimize time and memory consumption in the verification phase is provided.

Finally, in **Conclusion** we summarize this thesis and discuss possible directions in future work.

Part I

State-of-the-Art

Chapter 2

Stochastic Petri Nets: A General Overview

2.1 Introduction

Petri nets (PNs) [Mur89] present a well-known and versatile paradigm for the modeling and verification of various discrete-event systems [ST96]. Mainly, PNs have been used to model systems in which some events can occur concurrently with some constraints on the concurrence, precedence, or frequency of their occurrences [Pet77]. In fact, their graphical nature allows intuitive modeling of parallelism, synchronization, conflicts, concurrency, etc. in complex systems, while their formal semantics and mathematics foundation allow unambiguous descriptions, as well as, formal verification.

PNs can be analyzed through either computing all reachable states or using methods in discrete mathematics such as matrix equations. PNs properties are used to detect deadlocks, overflow, irreversible situations, etc. Using stochastic and timed PNs, performance evaluation is also possible.

PNs enjoy numerous advantages that can be summarized in the following [Pet81, KV86, GV01]:

1. They have a powerful mathematical foundation, as well as, an intuitive graphical representation. The graphical representation gives a flat view to PN models, making it possible to have simple and very explicit models. As well, their graphic modeling enables easy visualization of complex systems. Usually, in many similar techniques, only either graphical or mathematical side is well developed,

2. They provide well-integrated abstraction and refinement mechanisms that enable an effective design of large scale and complex systems,

3. They have been used in a wide range of application areas. Hence, there is a high degree of expertise in the modeling field,

4. There are many extensions of Petri nets such as colored, timed, stochastic, high-level, object-oriented Petri nets, etc. that fit well with specific requirements of a wide range of applications areas,

5. Their mathematical foundation is at the origin of all the analysis techniques proposed in order to verify the modeled systems. Indeed, they dispose of a well-developed qualitative/quantitative analysis panoply,

6. The state space can be given in a compact representation of the state. It is not required to explicitly enumerate all possible reachable states, instead, only the state evolution is provided,

7. They offer a careful balance of modeling power and decision power. In fact, Petri nets have been used in the modeling of a wide variety of systems. As for their decision

power, the reachability problem is decidable in Petri nets (note that most problems can be converted into reachability problems).

However, PNs have some disadvantages [Pet81]:

- **State space explosion**: As systems become more and more complex, their state spaces increase more and more, which can lead to state space explosion problem,
- **A delicate modeling and verification coupling**: Subclasses of PNs increase the decision power, however, a large number of systems can no longer be modeled. On the other hand, extensions of PNs may increase the modeling power, but at the expense of property decidability.

The remainder of this chapter starts by introducing some intuitive aspects of Petri nets in Section 2.2. Then, their formal aspects such that structure, behavior, qualitative properties and analysis methods are presented in Sections 2.3–2.5. After providing Petri nets basics, a major class of Petri nets that considers time and quantitative properties, called stochastic Petri nets, is described in Sections 2.6 and 2.7. Finally, this chapter ends with a conclusion.

2.2 Modeling with Petri nets

Discrete event systems are defined as systems of which their states are described by discrete variables and their state evolution depends on the occurrence of discrete events [CL09]. In discrete event systems' modeling process, two important concepts are considered, namely, *events* and *conditions*, and the relationships among them.

Conditions are descriptions of system states in terms of values of some variables, while events are actions taking place in the system. The occurrence of these events is controlled by system states, i.e., conditions. As such, at a given time, the condition may hold which implies the occurrence of certain events. The conditions cause such occurrence are called *pre-conditions*. As well, the occurrence of these events may give new conditions. Such new conditions are called *post-conditions*.

This view can be intuitively modeled by Petri nets [Pet81]. In fact, conditions are modeled by **places**, depicted as circles, considered as variables. The values of these variables are given in terms of **tokens**, depicted as black dots, inside their corresponding places, which models the truth values of conditions. Events are modeled by **transitions**, depicted as bars. The pre-conditions of an event are the input places, connected by **direct arcs**, of the corresponding transition. Analogously, the post-conditions are the output places. The occurrence of an event is modeled by **firing** the corresponding transition. The state evolution after an event occurrence (i.e., firing a transition) is modeled by (i) **consuming** tokens from pre-conditions

(i.e., places) of the corresponding transition, and (ii) **producing** new tokens in its post-conditions.

Consider PN H shown in Fig. 2.1. It models two processes $\mathtt{p}_1$ and $\mathtt{p}_2$ trying access a critical resource. Initial marking (i.e., state) M_0 is modeled by one token in place $\mathtt{idle}_1$, one token in place $\mathtt{idle}_2$, one token in place res, and zero token in the other places. This initial state, called initial marking, is denoted by $M_0(\mathtt{CS}_1, \mathtt{CS}_2, \mathtt{idle}_1, \mathtt{idle}_2, \mathtt{res}, \mathtt{wait}_1, \mathtt{wait}_2) = (0, 0, 1, 1, 1, 0, 0)$.

A token in places $\mathtt{idle}_i$, $\mathtt{wait}_i$ and $\mathtt{CS}_i$ means that process $\mathtt{p}_i$ is idle, waiting to access a critical resource, and in a critical section, respectively. Transitions $\mathtt{request}_i$, $\mathtt{enter}_i$, and $\mathtt{free}_i$ are fired when process $\mathtt{p}_i$ requests a critical resource, enters a critical section, and frees a critical resource, respectively.

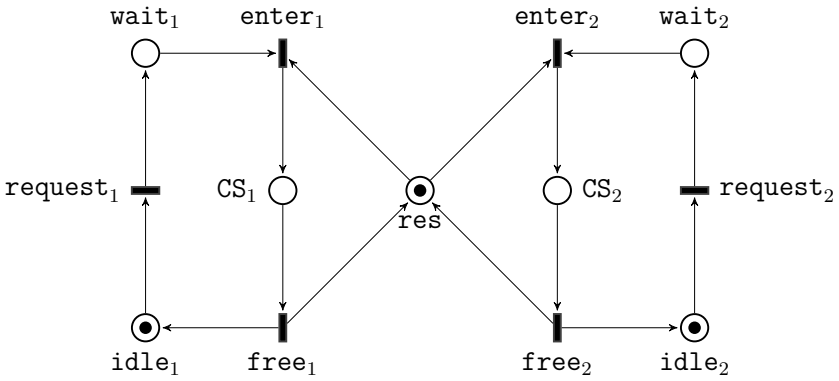


Figure 2.1: A PN models mutual exclusion.

2.3 Petri Net Structure

A Petri net is a bipartite directed graph that consists of places, depicted as circles, transitions, depicted as bars, and arcs connecting either place to transition or transition to place.

Definition 2.1. Petri Net. A Petri net is a 4-tuple $\mathcal{N} = \langle P, T, F, M_0 \rangle$ where

- P is a finite and non-empty set of places,
- T is a finite and non-empty set of transitions disjoint from P,
- $F : (P \times T) \cup (T \times P) \longrightarrow \mathbb{N}$ is a flow relation for a set of arcs,
- $M_0 : P \longrightarrow \mathbb{N}$ is an initial marking.

Some authors refer to the unmarked net $\mathcal{N} = \langle P, T, F \rangle$ as the Petri net structure and refer to the marked net $\mathcal{N}' = \langle P, T, F, M_0 \rangle$ as the system. Furthermore, if the net marking is partially provided, then the Petri net is called a family [CDF91].

Example: Consider PN H shown in Fig. 2.1. Its formal definition is given by $H = \langle P, T, F, M_0 \rangle$ where

- $P = \{\texttt{idle}_1, \texttt{wait}_1, \texttt{CS}_1, \texttt{idle}_2, \texttt{wait}_2, \texttt{CS}_2, \texttt{res}\}$,
- $T = \{\texttt{request}_1, \texttt{enter}_1, \texttt{free}_1, \texttt{request}_2, \texttt{enter}_2, \texttt{free}_2\}$,
- $F(\texttt{idle}_1, \texttt{request}_1) = 1, F(\texttt{request}_1, \texttt{wait}_1) = 1$, etc.
- $M_0(\texttt{CS}_1, \texttt{CS}_2, \texttt{idle}_1, \texttt{idle}_2, \texttt{res}, \texttt{wait}_1, \texttt{wait}_2) = (0, 0, 1, 1, 1, 0, 0)$.

Definition 2.2. Preset and Postset. The inputs and outputs, called also preset and postset respectively, of places and transitions can be defined formally as follows.

- The preset of place p, denoted by $\bullet p$, is given by $\bullet p = \{t \in T | F(t, p) > 0\}$.
- The postset of place p, denoted by $p\bullet$, is given by $p\bullet = \{t \in T | F(p, t) > 0\}$.
- The preset of transition t, denoted by $\bullet t$, is given by $\bullet t = \{p \in P | F(p, t) > 0\}$.
- The postset of transition t, denoted by $t\bullet$, is given by $t\bullet = \{p \in P | F(t, p) > 0\}$.

Example: Considering PN H shown in Fig. 2.1, we have, for instance,

1. $\bullet\texttt{idle}_1 = \{\texttt{free}_1\}$ and $\texttt{idle}_1\bullet = \{\texttt{request}_1\}$.
2. $\bullet\texttt{enter}_1 = \{\texttt{wait}_1, \texttt{res}\}$ and $\texttt{enter}_1\bullet = \{\texttt{CS}_1\}$.

2.4 Dynamic Behavior of Petri Nets

Previously, we have considered the static part of PNs. In the present section, we deal with the dynamic evolution of PN marking controlled by two rules called "enabling rule" and "firing rule". Both enabling and firing rules are specified through arc multiplicities (weights) and place markings. As for arcs, the enabling rule depends on input arcs of a transition, while the firing rule considers both input and output arcs.

Firing a transition in PN models an occurrence of its corresponding event. Before firing any transition, we check if its pre-conditions hold, in such case, the transition is said to be *enabled*.

A transition t is enabled if each place in its preset contains tokens, at least, as many as the multiplicity of the arc connecting both of them.

Definition 2.3. Enabling Rule. Transition t is enabled at marking M, denoted by $M[t\rangle$, iff

$$M(p) \geq F(p, t), \forall\ p \in \bullet t.$$

Example: Consider PN H shown in Fig. 2.1. Transition $\texttt{request}_1$ is enabled since we have $\bullet\texttt{request}_1 = \{\texttt{idle}_1\}$, $M(\texttt{wait}_1) = 1$, (i.e., place $\texttt{idle}_1$ contains one token), and $M(\texttt{wait}_1) \geq F(\texttt{idle}_1, \texttt{request}_1)$.

Definition 2.4. Firing Rule. The firing of transition t enabled at marking M yields new marking M', denoted by $M[t\rangle M'$, such that

$$M'(p) = M(p) + F(t,p) - F(p,t),\ \forall\ p \in P.$$

Firing transition t (i) removes from each place in its preset as many tokens as the multiplicity of the arc connecting both of them, and produces to each place in its postset as many tokens as the multiplicity of the arc connecting both of them.

Example: Consider PN H shown in Fig. 2.1. At initial marking M_0, transitions $\texttt{request}_1$ and $\texttt{request}_2$ are fireable. In fact, the direct arc from $\texttt{idle}_1$ to transition $\texttt{request}_1$ means that the pre-condition of transition $\texttt{request}_1$ is place $\texttt{idle}_1$. This place contains one token, that is, the pre-conditions of $\texttt{request}_1$ hold, hence it can fire (i.e., the corresponding event can occur). Firing $\texttt{request}_1$ removes a token from $\texttt{idle}_1$ and puts one token inside $\texttt{wait}_1$, since we have an arc from $\texttt{request}_1$ to $\texttt{wait}_1$, which models the post-conditions of firing $\texttt{request}_1$. Similarly to transition $\texttt{request}_2$ with respect to places $\texttt{idle}_2$ and $\texttt{wait}_2$.

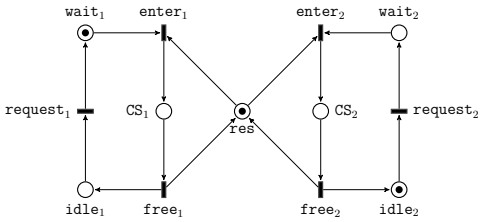
PN H after firing $\texttt{request}_1$ is shown in Fig. 2.2a. Firing $\texttt{request}_1$ at M_0 causes enabling of transition $\texttt{enter}_1$ at new marking M_1. Firing the latter transition at M_1 leads to new marking illustrated in Fig. 2.2b. In Fig. 2.2c, transition $\texttt{enter}_2$ cannot fire, since its pre-conditions are $\texttt{wait}_2$ and $\texttt{res}$ such that $\texttt{res}$ does not contain any token which means that pre-conditions of $\texttt{enter}_2$ do not hold. After firing $\texttt{free}_1$ at marking depicted in Fig. 2.2c, the obtained marking is depicted in Fig. 2.2d in which $\texttt{enter}_2$ becomes enabled.

Definition 2.5. Transition Sequence. A transition sequence $\sigma = t_1, t_2, \ldots, t_n$ is fireable starting from marking M_1 iff there exists a sequence of markings $M_1, M_2, \ldots, M_n$ such that $M_i[t_i\rangle, \forall\ i \in \{1, 2, \ldots, n\}$.

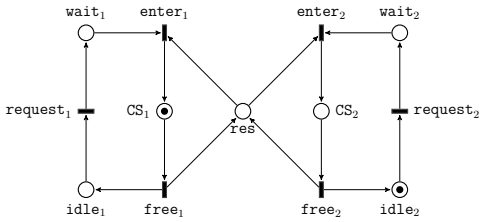
$M_1[\sigma\rangle M_{n+1}$ denotes the firing of transition sequence σ starting from M_1 and M_{n+1} is reachable from M_1 by firing σ.

Example: Consider Table 2.1, where the initial marking is denoted by s_0. Firing sequence $\sigma = \texttt{request}_1, \texttt{enter}_1, \texttt{request}_2, \texttt{free}_1, \texttt{enter}_2, \texttt{free}_2$ is fireable starting from s_0, since there exists a marking sequence $s_0, s_2, s_5, s_7, s_1, s_3$, such that $s_0[\texttt{request}_1\rangle s_2[\texttt{enter}_1\rangle$ $s_5[\texttt{request}_2\rangle s_7[\texttt{free}_1\rangle s_1[\texttt{enter}_2\rangle s_3[\texttt{free}_2\rangle$.

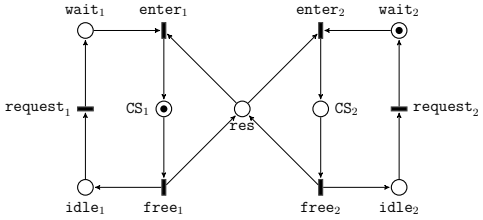
Since PNs structure is static and firing transitions changes only the marking, which is considered as the dynamic part of PNs, the state evolution can be modeled as a graph where its nodes correspond to reachable markings, and its edges correspond to fired transitions. Such a graph is called the reachability graph. Fig. 2.3 shows the reachability graph of PN H depicted in Fig. 2.1, and their corresponding values are listed in Table 2.1.



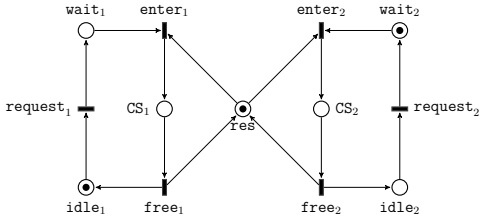
(a) Firing $\texttt{request}_1$.



(b) Firing $\texttt{request}_1$ then $\texttt{enter}_1$.



(c) Firing $\texttt{request}_1$ $\texttt{enter}_1$ $\texttt{request}_2$.



(d) Firing $\texttt{request}_1$ $\texttt{enter}_1$ $\texttt{request}_2$ $\texttt{free}_1$.

Figure 2.2: Firing transitions in PN H at M_0.

Definition 2.6. Reachability Set. The Reachability set of a PN having initial marking M_0, denoted by $RS(M_0)$, is defined as follows $RS(M_0) = \{M | M_0[\sigma\rangle M \ \wedge \ \sigma$ is a fireable sequence$\}$.

Note that (i) σ can be empty, that is, $M_0 \in RS(M_0)$, (ii) $RS(M_0)$ can be infinite, and (iii) an initial marking must be completely provided in order to compute the reachability set, that is, this computation is not possible neither for PN structure nor for PN family [CDF91, Mar+94].

Example: Consider PN H depicted in Fig. 2.1. Its reachability set is shown in Table 2.1, that is, $RS(M_0) = \{s_0, s_1, \ldots, s_7\}$.

Definition 2.7. Reachability Graph. The Reachability graph of a PN $\mathcal{N} = \langle P, T, F, M_0\rangle$, denoted by $RG(M_0) = \langle V, E\rangle$, is a labeled directed graph, where

i) $V = RS(M_0)$, and

ii) $E = \{(M_i, t, M_j) | M_i \in RS(M_0) \wedge M_j \in RS(M_0) \wedge t \in T\}$.

Example: Consider PN H depicted in Fig. 2.1. Its reachability graph is shown in Fig. 2.3, that is, $RG(M_0) = \langle V, E\rangle$ where:

i) $V = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$, and

ii) $E = \{(s_0, \texttt{request}_2, s_1), (s_0, \texttt{request}_1, s_2), (s_1, \texttt{enter}_2, s_4), (s_1, \texttt{request}_1, s_3), \ldots\}$.

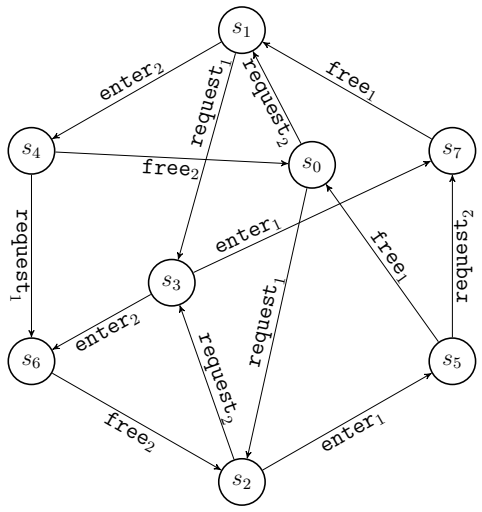


Figure 2.3: Reachability graph of H.

	CS_1	CS_2	idle_1	idle_2	res	wait_1	wait_2
s_0	0	0	1	1	1	0	0
s_1	0	0	1	0	1	0	1
s_2	0	0	0	1	1	1	0
s_3	0	0	0	0	1	1	1
s_4	0	1	1	0	0	0	0
s_5	1	0	0	1	0	0	0
s_6	0	1	0	0	0	1	0
s_7	1	0	0	0	0	0	1

Table 2.1: Reachable markings of H.

2.5 Petri Net Analysis

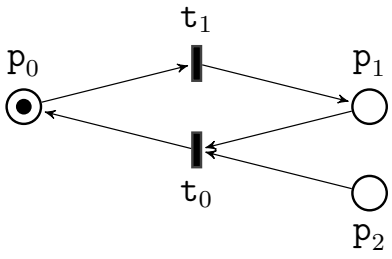
Besides its modeling power, a major reason for using Petri nets is its decision power. Numerous analysis techniques have been developed for PNs verification. Indeed, Petri nets support the analysis of many properties and problems associated with concurrent systems [Mur89]. The properties can be analyzed either based on reachability graph, called behavioral properties, or independently from reachability graph, called structural properties.

In this section, we discuss only important properties and their analysis problems. For further reading about properties analysis, the reader is referred to [Pet81, Mur89, Mar+94, EN94, GV01, Rei12].

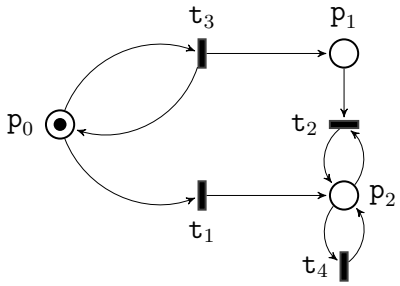
2.5.1 Petri Net Properties

Reachability

The reachability problem for Petri nets consists of deciding if a marking M is reachable from the initial marking. It has been proven that this problem is decidable [Kos82, May84, Reu90,



(a) t_0 is L0-live and t_1 is L1-live.



(b) t_2 is L2-live, t_3 is L3-live and t_4 is L4-live.

Figure 2.4: Different levels of liveness.

Lam92]. The reachability problem is a central concept in Petri nets since most problems can be converted into reachability problems.

Liveness and Deadlock-freedom

A Petri net is said to be live if every transition can always fire again in the future at any reachable marking. It has been shown that the liveness problem is decidable since it is recursively equivalent to the reachability problem [AK76].

This kind of liveness is a strong property. Thus, it is relaxed to different levels of liveness [Mur89, Pet77]. A transition in a Petri net having initial marking M_0 is

- **L0-live (or dead)** if it can never be enabled at any reachable marking.
- **L1-live** if it can be enabled at least once in certain firing sequence starting from M_0.
- **L2-live** if it can fire, at least, k times in certain firing sequence starting from M_0, where k is a positive integer number.
- **L3-live** if it appears infinitely often in certain firing sequence starting from M_0.
- **L4-live (or live)** if it is L1-live for any reachable marking from M_0.

A Petri net is Lk-live if all transitions are Lk-live, such that $k \in 0, 1, 2, 3, 4$.

Example: Petri nets in Fig. 2.4 shows different levels of liveness, where t_0 in Fig. 2.4a is dead; and transitions t_1, t_2, t_3 and t_4 in Fig. 2.4b are L1-, L2-, L3- and L4-live, respectively.

Boundedness

A Petri net is bounded if its reachable set is finite. It has been proved that boundedness is decidable [KM69]. In fact, a Petri net is unbounded iff there exists a reachable marking M and transition sequence σ such that $M[\sigma\rangle M + L$, where L is a non-zero marking [EN94].

A Petri net is *k-bounded* if, for any reachable marking, there is at most k tokens in any place. A Petri net is safe if it is 1-bounded.

If a place in a Petri net represents a buffer, it would be interesting to verify whether this place is bounded or safe to guarantee that there will be no overflows in the buffer.
Example: PNs shown in Figs.2.4a and 2.4b are safe and non-bounded, respectively.

Home state and Reversibility

A marking of a Petri net is a home state if it can be reached from all reachable markings. Authors of [Fru86, BE16] have shown that home state problem is decidable. A Petri net is said to be reversible if its initial marking is a home state.
Example: Marking $M(\mathtt{p}_0, \mathtt{p}_1, \mathtt{p}_2) = (0, 0, 1)$ is a home state for Petri net shown in Fig. 2.4b, such that is can be reached by firing $\mathtt{t}_1$ once, and firing $\mathtt{t}_2$ until it becomes no longer enabled. Petri net in Fig. 2.1 is reversible as shown in its reachability graph depicted in Fig. 2.3.

Conservation

A Petri net is said to be conservative iff the number of tokens in any reachable marking remains constant, i.e., after any transition firing the number of consumed tokens equals the number of produced tokens. This property would be useful to show that the resources (modeled by tokens) are neither created nor destroyed during the state evolution of the system.
Example: Reachable markings of Petri net depicted in Fig. 2.5a are $(1, 1, 0, 0), (0, 0, 1, 1)$, $(1, 0, 0, 1)$, and $(0, 1, 1, 0)$ (the marking of $\mathtt{p}_0$, $\mathtt{p}_1$, $\mathtt{p}_2$ and $\mathtt{p}_3$).

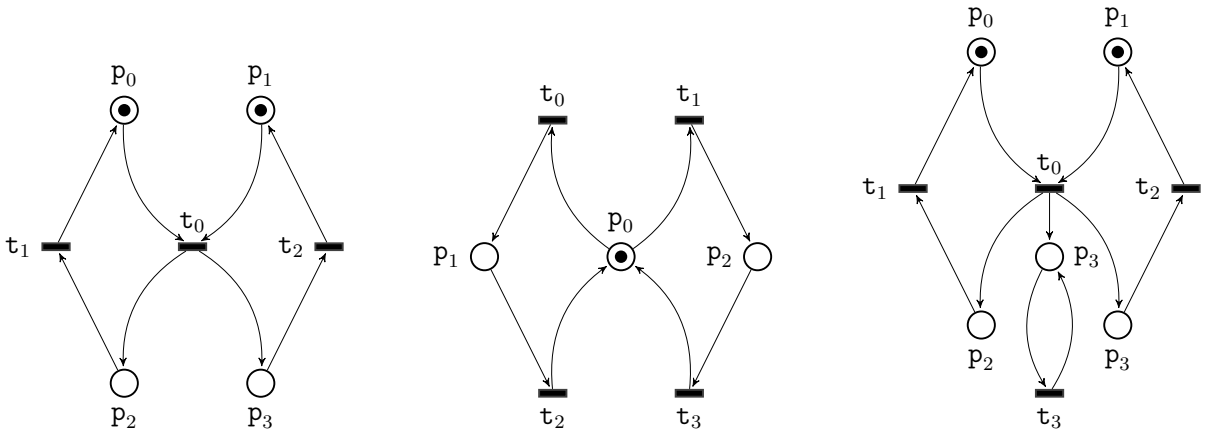
Persistence

A Petri net is persistent if any two different transitions t_1 and t_2 are enable at any reachable marking M, then firing of t_1 will not disable t_2, and vice-versa. Authors of [Gra80, May81, Mül81] have proved that this problem is decidable.
Example: Consider Petri net shown in Fig. 2.5a. At initial marking only transition $\mathtt{t}_0$ is enabled. Firing this transition leads to new marking $M_1(\mathtt{p}_0, \mathtt{p}_1, \mathtt{p}_2, \mathtt{p}_3) = (0, 0, 1, 1)$ at which both transitions $\mathtt{t}_1$ and $\mathtt{t}_2$ are enabled. Firing $\mathtt{t}_1$ at M_1 will not disable $\mathtt{t}_2$, and vice-versa.

Fairness

There are two types of fairness: bounded-fairness and unconditional fairness. Two transitions are in a bounded-fair relation if the occurrence number of one is bounded while the other does not yet fire. A PN is bounded-fair if any two transitions are in a bounded-fair relation.

A firing sequence is unconditionally fair if it is either finite or all transitions appear infinitely often in it. A PN is unconditionally fair if all fireable sequences from its initial marking are unconditionally fair.



(a) Conservative, persistent and bounded-fair Petri net.

(b) Unfair Petri net.

(c) Uncontionally fair Petri net.

Figure 2.5: Conservation, persistence and fairness.

Example: Consider Petri net shown in Fig. 2.5a. All firing sequences starting from initial marking are words generated from the following regular expression $(\mathtt{t}_0(\mathtt{t}_1\mathtt{t}_2|\mathtt{t}_2\mathtt{t}_1))^*$, i.e., are concatenations of $\mathtt{t}_0\ \mathtt{t}_1\ \mathtt{t}_2$ and/or $\mathtt{t}_0\ \mathtt{t}_2\ \mathtt{t}_1$. Thus, this Petri net is bounded-fair.

Petri net depicted in Fig. 2.5b is unfair. In fact, both transitions $\mathtt{t}_0$ and $\mathtt{t}_2$ can fire, sequentially, infinitely without any firing of neither $\mathtt{t}_1$ nor $\mathtt{t}_3$. As for PN illustrated in Fig. 2.5c, it is unconditionally fair but not bounded-fair since place p_3 is an unbounded place making the occurrence number of $\mathtt{t}_3$, without firing other transition, unbounded as well.

Mutual exclusion

There are two types of mutual exclusion dealing with the impossibility of simultaneous sub-markings or firing concurrency. Two places p and q are mutually exclusive if both of them cannot be marked at the same marking, i.e., places p and q are mutually exclusive iff $\forall M \in RS(M_0), M(p) \times M(q) = 0$. Two transitions are mutually exclusive if they cannot be both enabled at any reachable marking.

Example: Consider Petri net shown in Fig. 2.5b. Its reachable markings are $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$ (the values in each vector are the markings of places $\mathtt{p}_0$, $\mathtt{p}_1$ and $\mathtt{p}_2$, respectively). Hence, each two places in this Petri net are mutually exclusive.

Furthermore, at any reachable marking listed above, there exists one and only one fireable transition. Thus, each two transitions in this Petri net are mutually exclusive.

2.5.2 Temporal Logic

To consider a large set of properties in Petri nets, researchers have studied decidability issues using *model checking* [BK08].

Model-checking refers to a wide range of techniques that check whether a given property formalized by a *temporal logic* is verified by a given system modeled, often, by a transition system. The model checker confirms that either the system satisfies the property or violates it. In the latter case, a counter-example is provided.

Among the properties that can be checked by model checking, there are two important types of properties, namely, the liveness and safety properties [GV01]. Roughly speaking, a liveness property (eventually) must be satisfied in the future by all future executions, i.e., "good things" do happen. For instance, if a process asks for a critical resource, it will access to it in the future. A safety property (always) must be satisfied in each reachable state in the future, i.e., "bad things" do not happen. For instance, a critical resource will never be used by two processes or more at the same time. It has been shown in [AS87] that any property can be decomposed as a conjunction of safety and liveness properties.

Aforementioned, usually model checker requires formalizing properties as formulae of temporal logic and describing systems as transition systems. In the following, we present briefly transition systems and temporal logic.

Definition 2.8. Transition System. A transition system TS is a tuple $\langle S, T, E, I, \mathcal{A}, \mathcal{L}\rangle$, where:

- S is a set of states and T is a set of actions,
- $E \subseteq S \times T \times S$ is a transition relation on S,
- $I \subseteq S$ is a set of initial states,
- $\mathcal{A}$ is a set of atomic propositions,
- $\mathcal{L} : S \longrightarrow 2^{\mathcal{A}}$ is a labeling function that assigns truth value to atomic propositions at each state, that is, $a \in \mathcal{A}$ is true at $s \in S$ iff $a \in \mathcal{L}(s)$.

In PNs context, S, T and E are obtained from the reachability graph, I contains a single initial marking, and the atomic propositions are conditions on place markings.
Example: Consider transition system TS depicted in Fig. 2.6 of Petri net H shown in Fig.2.1. TS is defined as follows. $TS = \langle S, T, E, I, \mathcal{A}, \mathcal{L}\rangle$, where:

- S is the set of reachable states of H, that is $S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$,
- $T = \{\texttt{request}_1, \texttt{request}_2, \texttt{enter}_1, \texttt{enter}_2, \texttt{free}_1, \texttt{free}_2\}$,

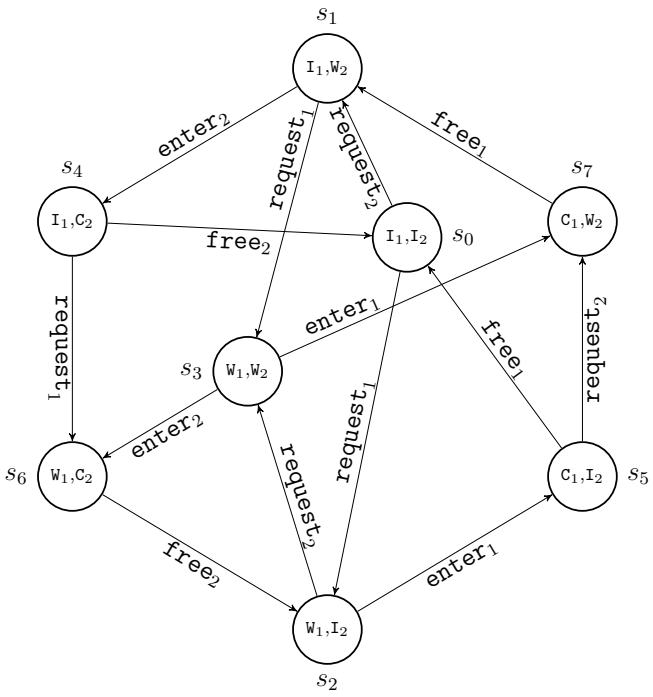


Figure 2.6: Transition system of H shown in Fig. 2.1, where $\mathtt{I}_i$, $\mathtt{W}_i$ and $\mathtt{C}_i$ denote $(\mathtt{idle}_i = 1)$, $(\mathtt{wait}_i = 1)$ and $(\mathtt{CS}_i = 1)$ for each $i \in \{1, 2\}$, respectively.

- $E = \{(s_0, \mathtt{request}_2, s_1), (s_0, \mathtt{request}_1, s_2), (s_1, \mathtt{enter}_2, s_4), (s_1, \mathtt{request}_1, s_3), \dots\}$,
- $I = \{s_0\}$,
- $\mathcal{A} = \{(\mathtt{idle}_1 = 1), (\mathtt{idle}_2 = 1), (\mathtt{wait}_1 = 1), (\mathtt{wait}_2 = 1), (\mathtt{CS}_1 = 1), (\mathtt{CS}_2 = 1)\}$, such that $(\mathtt{idle}_1 = 1)$ means $\mathtt{idle}_1$ is marked by one token. Similarly to the other places,
- $\mathcal{L}$ is the labeling function such that $\mathcal{L}(s_0) = \{(\mathtt{idle}_1 = 1), (\mathtt{idle}_2 = 1)\}$, $\mathcal{L}(s_1) = \{(\mathtt{idle}_1 = 1), (\mathtt{wait}_2 = 1)\}$, $\mathcal{L}(s_2) = \{(\mathtt{wait}_1 = 1), (\mathtt{idle}_2 = 1)\}$, etc.

Definition 2.9. Finite Path, Maximal Path, and Path. A finite path is a state sequence $s_0 s_1 \dots s_n$, where $\exists\ (s_{i-1}, t, s_i) \in E$, such that $t \in T$, for all $i \in \{1, 2, \dots, n\}$. A maximal path π of a transition system with no terminal state is an infinite sequence of states $s_0 s_1 s_2 \dots$, such that $\forall\ i, \exists\ t \in T, (s_i, t, s_{i+1}) \in E$. A maximal path $\pi = s_0\ s_1\ s_2 \dots$ is a path, if $s_0 \in I$.

Example: Consider transition system TS depicted in Fig. 2.6. $s_0 s_2 s_5 s_7$ is a finite path, $s_2 s_5 s_7 s_1 s_4 s_0\ s_1 \dots$ is a maximal path, and $s_0 s_2 s_5 s_7 s_1 s_4 s_0 \dots$ is a path.

Definition 2.10. Trace. The trace of a maximal path $\pi = s_0 s_1 s_2 \dots$ is defined as $trace(\pi) = \mathcal{L}(s_0)\mathcal{L}(s_1)\mathcal{L}(s_2) \dots$. That is, the trace registers the valid propositions in each state of π.

Example: The trace of $s_0 s_2 s_5 s_7 s_1 s_4 s_0 \dots$ is $\{\mathtt{I}_1, \mathtt{I}_2\}\{\mathtt{W}_1, \mathtt{I}_2\}\{\mathtt{C}_1, \mathtt{I}_2\}\{\mathtt{C}_1, \mathtt{W}_2\}\{\mathtt{I}_1, \mathtt{W}_2\}\{\mathtt{I}_1, \mathtt{C}_2\}$ $\{\mathtt{I}_1, \mathtt{I}_2\} \dots$.

As for temporal logics, there are two categories, namely, linear time logic (LTL) and branching time logic (CTL). In this dissertation, we will restrict our attention to the linear temporal logic. A comprehensive formal description of model checking can be found in [BK08].

Definition 2.11. LTL Formula. An LTL formula is inductively defined as follows.

$$\Phi ::= \text{true} \mid a \mid \neg\Phi \mid \Phi_1 \vee \Phi_2 \mid \mathbf{X}\Phi \mid \Phi_1 \ \mathbf{U} \ \Phi_2$$

where $a \in \mathcal{A}$ is an atomic proposition.

Example: All the following formulae are LTL formulae.

- $\Phi_1 ::= (\mathtt{CS}_1 = 1)$,
- $\Phi_2 ::= \neg(\mathtt{CS}_1 = 1) \vee \neg(\mathtt{CS}_1 = 1)$,
- $\Phi_3 ::= \mathbf{X}(\neg(\mathtt{CS}_1 = 1) \vee \neg(\mathtt{CS}_1 = 1))$,
- $\Phi_4 ::= (\mathtt{wait}_2 = 1)\mathbf{U}(\mathtt{CS}_2 = 1)$.

Informally, formula $\mathbf{X}\Phi$ holds for a maximal path π if its second state satisfies Φ. For instance, consider path $\pi_0 = s_0 s_2 s_5 s_7 s_1 s_4 s_0 \ldots$. Formula Φ_3 holds for π_0 since $(\neg(\mathtt{CS}_1 = 1) \vee \neg(\mathtt{CS}_1 = 1))$ is satisfied in s_2.

As for formula $\Phi_1\mathbf{U}\Phi_2$, it holds for a maximal path π if there exists a state s satisfies Φ_2, such that each prior state thereof satisfies Φ_1. For instance, consider maximal path $\pi_1 = s_7 s_1 s_4 s_0 \ldots$. Formula Φ_4 holds for π_1, since formula $(\mathtt{wait}_2 = 1)$ holds for s_7 and s_1; and formula $(\mathtt{CS}_2 = 1)$ is satisfied in s_4. As for maximal path $\pi_2 = s_7 s_1 s_3 s_7 s_1 s_3 s_7 \ldots$, formula Φ_4 does not hold.

Operators *eventually* (denoted by $\mathbf{F}$) and *always* (denoted by $\mathbf{G}$) can be expressed via operator $\mathbf{U}$, such that $(\mathbf{F}\Phi \equiv \mathbf{true}\mathbf{U}\Phi)$ and $(\mathbf{G}\Phi \equiv \neg\mathbf{F}\neg\Phi)$.

2.5.3 Analysis Methods

In literature, Petri net analysis methods can be categorized as: *enumeration* or *net-driven*.

The first category consists on the computation of the reachability graph (totally or partially). If the reachability graph is finite, it can be used for a proof system or for decision procedures, where the most important properties are decidable. On the other hand, we can use the *coverability tree* for unbounded systems, however, important properties such as liveness and reachability are not decidable [Pet81]. Basically, enumeration based methods are applicable to any Petri net class, however due to computational complexity, their use is limited to small nets [Mur89].

To overcome state space explosion problem, net-driven approaches reason on the net structure to extract useful information about the net behavior.

Two different approaches are: structure theory and net transformations. The former is based on graph theory and/or linear algebra. The latter transforms Petri net to another one (typically by reduction) while preserving the properties to be analyzed. The resulting models are expected to be easier to analyze [GV01].

Analysis by transformation transforms a net system $\mathcal{S}$ into a net system $\mathcal{S}'$ preserving a set of properties Π in such a way that verifying set Π in $\mathcal{S}'$ becomes easier than in $\mathcal{S}$ such that $\mathcal{S}'$ may belong to a Petri net subclass for which state enumeration can be avoided.

A special class of transformation methods is reduction methods in which a net system $\mathcal{S}$ is transformed into a net system $\mathcal{S}'$ (eventually using a reduction sequence) in such a way that $\mathcal{S}'$ is smaller than $\mathcal{S}$ preserving a set of properties Π of $\mathcal{S}$.

In structural analysis techniques, the net behavior is reasoned from its structure, and its initial marking is considered as a parameter. We can find two subcategories of structural analysis:

1. *Linear algebra*: Using matrix equations, they allow fast verification without the necessity of enumeration in certain cases.

2. *Graph-based techniques*: They are effective in analyzing some subclasses of Petri nets.

Although net-driven techniques are powerful, in many cases they are applicable only to special subclasses of Petri nets [Mur89]. For further information about these analysis techniques, the reader is referred to [Pet81, Mur89, GV01].

2.6 Stochastic Petri Nets

In this section, we present a class of Petri net enriched by temporal concepts, namely stochastic Petri nets (SPNs). Indeed, Petri nets, in their basic definition, do not involve any concept of time quantification. The notion of time is abstract in basic Petri nets, i.e., only logical relationships in terms of time such as transition t_1 fires *before* transition t_2, transition t_3 fires *after* transition t_4, etc. are allowed. Thus, performance evaluation of systems using basic Petri nets is not possible.

On the other hand, the time concept is viral in certain real applications whose efficiency is a crucial aspect. Hence, time-augmented Petri nets are introduced. In general, there are two possible ways to do this [BK02]:

- **Timed places Petri nets (TPPNs)**: In this class of Petri nets, tokens fired onto a place p are unavailable to all its output transitions for a certain time. Once this time has elapsed the tokens become available.

- **Timed transitions Petri nets (TTPNs)**: In this class of Petri nets, once a transition becomes enabled it does not fire immediately, instead, it fires after a certain time.

Time-augmented Petri nets are classified also depending upon time nature. In fact, if time is deterministic, then such Petri nets are called "timed Petri nets". On the other hand, if the firing time is a random variable following a certain distribution law, then they are called "stochastic Petri nets". Different families of stochastic Petri nets can be defined based on their distribution laws of the firing time.

Moreover, the nature of stochastic Petri nets depends also on other firing characteristics [Mic01], namely:

Selection policy: Given a set of enabled transitions at a certain marking, there are two policies concerned with how tokens are reserved until the transition firing:

1. *Preselection policy.* First, (i) an enabled transition reserves all tokens it needs so that these tokens are not available for other transitions. Then, (ii) it waits for its firing time to elapse. Finally, (iii) it fires immediately according to the firing rule of Petri nets.

2. *Race policy.* First, (i) an enabled transition waits until its firing time interval has elapsed. Then, (ii) it fires immediately according to the firing rule of Petri nets provided it is still enabled at that time, i.e., the required tokens are not already consumed by another transition.

Service policy: Given an enabled transition at a certain marking at which it can fire k times, there are three policies concerned with how many times should the transition be fired when its firing time has elapsed:

1. *Single-server.* One single firing is performed. It means that transition can only offer one service at time,

2. *Infinite-server.* k firings are performed. It means that transition can ensure any number of simultaneous services,

3. *Multiple-server.* $Min(k, deg(t))$ firings are performed. It means that transition t can ensure $deg(t)$ simultaneous services at maximum.

Memory policy: Assume that each timed transition is associated with a timer. When a timed transition becomes enabled, its corresponding times is set to an initial value and then starts to decrement. When the timer reaches the value zero, the corresponding transition fires. Given a set of enabled transitions at a certain marking, there are three policies concerned with how should firing a transition influence the timers of the other transitions:

1. *Resampling.* At each firing, the timers of all the timed transitions are discarded (restart mechanism),

2. *Enabling memory.* At each firing, the timers of all the disabled timed transitions are restarted. As for timed transitions that are still enabled, their corresponding timers hold their present value (continue mechanism),

3. *Age memory.* At each firing, the timers of all the timed transitions hold their present values (continue mechanism).

In the following, we start by providing a brief introduction of stochastic processes. Then, the basic concepts of Markov chains are presented. Finally, we consider two major classes of stochastic Petri nets, namely, stochastic Petri nets having exponential law, generalized stochastic Petri nets, and their quantitative properties.

2.6.1 Stochastic Process

Stochastic processes are used to model the evolution of systems that exhibit probabilistic behaviors. They are defined by enumerating possible reachable states by the modeled systems and the probabilities upon time of transitions between these states.

Mathematically, a stochastic process is a family of random variables $\mathcal{X}(t)$ defined over the same probability space taking values in a set S, where parameter t denotes time and used to index each random variable and S is the state space of the process [BK02]. Stochastic processes can be classified according to the state space or the nature of time.

If the state space is discrete then the process is called *discrete-state process* or chain, whereas if the state space is continuous, then it is called *continuous-space process*. Analogically, if the time is continuous then the process is called *continuous-time process*, otherwise, it is *called discrete-time process*.

2.6.2 Markov Process

A particular class of stochastic processes, called Markov process, has found a wide range of applications in the research community. A Markov process is a stochastic process whose evolution depends only on the present, without being concerned with its evolution history. This is known as *Markov property*, as well as, "memoryless".

Let $P[\mathcal{X}(t) = x]$ denote the probability that the stochastic process will have value x at time t and $P[\mathcal{X}(t) = x|\mathcal{X}(t_n) = x_n]$ denote the probability that the stochastic process will have value x at time t such that it had value x_n at time t_n. Formally, a Markov process is a stochastic process $\{\mathcal{X}(t)\}$ whose conditional probability density function is such that

$$P[\mathcal{X}(t) = x|\mathcal{X}(t_n) = x_n, \mathcal{X}(t_{n-1}) = x_{n-1}, \ldots, \mathcal{X}(t_0) = x_0] = P[\mathcal{X}(t) = x|\mathcal{X}(t_n) = x_n]. \quad (2.1)$$

where $t > t_n > t_{n-1} > \cdots > t_0$.

If the state space of a Markov process is discrete, then it is called *Markov chain* (MC). If the time is continuous (discrete), then is called continuous time MC (CTMC) (respectively, discrete time MC (DTMC)).

Furthermore, if the future evolution of a Markov process is independent of particular instant t_n in Eq. 2.1 so that is completely determined by knowing the present state, then Markov process is said to be *time-homogeneous*. That is, the Markov process is invariant to shifts in time. For a time-homogeneous Markov process, the following condition holds:

$$P[\mathcal{X}(t+s) = x | \mathcal{X}(t_n + s) = x_n] = P[\mathcal{X}(t) = x | \mathcal{X}(t_n) = x_n]. \tag{2.2}$$

In CTMCs (either time-homogeneous or not) the probabilities of transitions between states are given as functions of time. The amount of time for which the process stays in a state before moving to another one is called the *sojourn time* or *holding time* which is exponentially distributed. Indeed, the exponential distribution is the only continuous distribution that satisfies the property of memoryless [BK02].

For the remainder of this thesis, we consider only discrete-state space, time-homogeneous Markov processes.

2.6.3 Stochastic Petri Nets having Exponential Law

A stochastic Petri net having exponential law $\mathcal{N}$ is a Petri net augmented by a function Λ that assigns to each transition t_i in $\mathcal{N}$ a firing delay λ_i which is exponentially distributed. The distribution of random variable $\mathcal{X}_i$ of the firing delay of transition t_i is given by

$$F_{\mathcal{X}_i}(x) = 1 - e^{-\lambda_i x}. \tag{2.3}$$

In this dissertation, we refer to timed transitions Petri nets with race, single server and resampling mode; and exponentially distributed stochastic firing time by simply stochastic Petri nets (SPNs).

Definition 2.12. Stochastic Petri Net. An SPN is a 5-tuple $\mathcal{N} = \langle P, T, F, M_0, \Lambda \rangle$ where

- P is a finite and non-empty set of places,
- T is a finite and non-empty set of transitions disjoint from P,
- $F : (P \times T) \cup (T \times P) \longrightarrow \mathbb{N}$ is a flow relation for a set of arcs,
- $M_0 : P \longrightarrow \mathbb{N}$ is an initial marking,
- $\Lambda : T \longrightarrow \mathbb{R}^+$ is a function that associates to each transition $t_i \in T$ an exponentially distributed firing time λ_i.

Example. Consider SPN $\mathcal{N}$ shown in Fig. 2.7. Both transitions $\texttt{request}_1$ and $\texttt{request}_2$ are enabled at initial marking $M_0(\texttt{CS}_1, \texttt{CS}_2, \texttt{idle}_1, \texttt{idle}_2, \texttt{res}, \texttt{wait}_1, \texttt{wait}_2) = (0, 0, 1, 1, 1, 0, 0)$. Before firing, $\texttt{request}_1$ waits until a certain time has elapsed. This time is exponentially distributed with rate λ_0, i.e., the average time for $\texttt{request}_1$ to fire is $\frac{1}{\lambda_0}$. Similarly to transition $\texttt{request}_2$ with respect to rate λ_3.

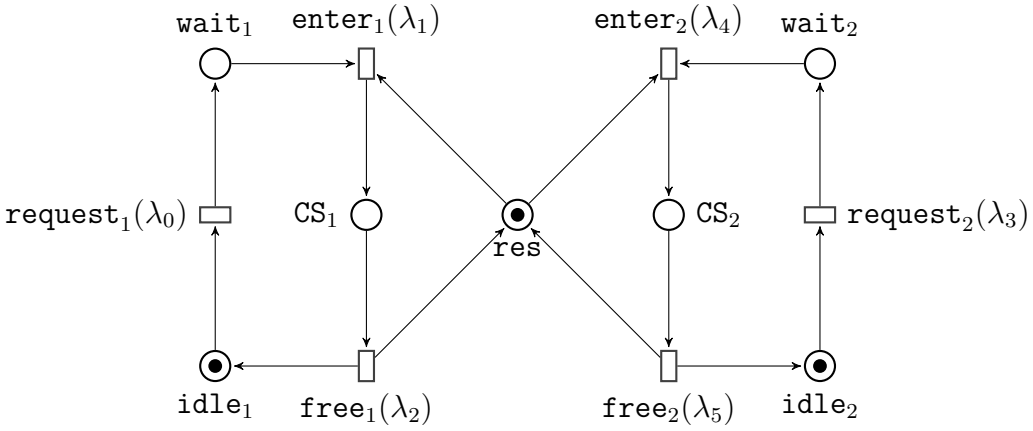


Figure 2.7: SPN $\mathcal{N}$ models mutual exclusion.

The nature of SPNs does not modify the basic behavior of the underlying non-timed model. This is useful since it is possible to study this class of SPNs using the basic model properties as well as the available theoretical results [Mar+94]. Indeed, the enabling and firing rules of basic Petri nets are preserved in SPNs. That is, once $\texttt{request}_1$ has fired, $\mathcal{N}$ marking becomes $M_2(\texttt{CS}_1, \texttt{CS}_2, \texttt{idle}_1, \texttt{idle}_2, \texttt{res}, \texttt{wait}_1, \texttt{wait}_2) = (0, 0, 0, 1, 1, 1, 0)$. As well, firing $\texttt{request}_2$ at M_0 leads to marking $M_2(\texttt{CS}_1, \texttt{CS}_2, \texttt{idle}_1, \texttt{idle}_2, \texttt{res}, \texttt{wait}_1, \texttt{wait}_2) = (0, 0, 1, 0, 1, 0, 1)$.

Considering probabilistic aspects of SPNs, the next marking depends on which transition fires first. Given n enabled transitions $t_1, t_2, \ldots, t_n$ having rates $\lambda_1, \lambda_2, \ldots, \lambda_n$ at marking M, the probability that t_i fires first is given by

$$P[t_i \text{ fires first at } M] = \frac{\lambda_i}{\lambda_1 + \lambda_2 + \cdots + \lambda_n}. \tag{2.4}$$

Using Eq. 2.4, the probability that $\texttt{request}_1$ fires first at M_0 equals $\frac{\lambda_0}{\lambda_0+\lambda_3}$ provided that only transitions $\texttt{request}_1$ and $\texttt{request}_2$ are enabled at M_0.

Given the following:

1. the next marking of an SPN depends only on its current marking,
2. the state space of SPNs is discrete,
3. the probabilities of transitions between markings are given as a function exponentially distributed time,

4. and the probability of changing from a marking M to some other marking is independent of the sojourn time spent in M.

Then, an SPN describes a time-homogeneous CTMC that is isomorphic to its reachability graph. Thus, quantitative analysis of SPNs can be carried out by analyzing the corresponding CTMC. The latter is obtained straightforwardly by computing the reachability graph of a given SPN based on the same algorithm used for underlying PN where the transition rate between two markings (states) is the rate of the corresponding fired transition in SPN.
Example. Consider CTMC illustrated in Fig. 2.8.

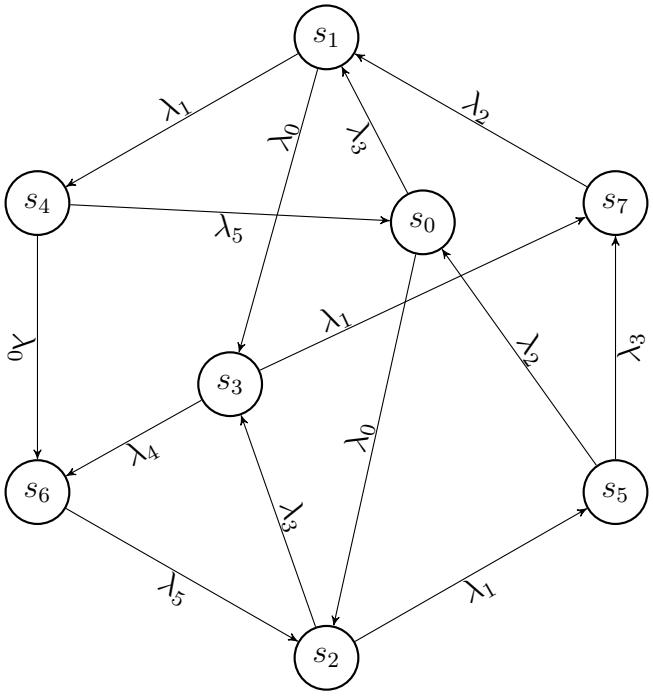


Figure 2.8: Corresponding CTMC of SPN $\mathcal{N}$ shown in Fig. 2.7, where states $s_0, \ldots, s_7$ are listed in Table 2.1.

This CTMC can be obtained by computing the reachability graph of SPN $\mathcal{N}$ depicted in Fig. 2.7 where the label of any transition between two markings is the rate of the corresponding fired transition. For instance, marking s_1 is obtained from s_0 by firing transition $\texttt{request}_2$, thus the rate of going from s_0 to s_1 is the rate of transition $\texttt{request}_2$, i.e., λ_3.

The next step in quantitative analysis of SPNs is to compute a *matrix of transition rates* $Q{=}(q_{ij})$ of order $n = |RS(SPN)|$, i.e., its order is the number of reachable markings. The element q_{ij} is the transition rate between corresponding marking M_i and M_j, i.e., the rate fired transition t_k such that $M_i[t_k\rangle M_j$. The element q_{ii} is computed such that $\sum_n^{j=1} q_{ij} = 0$, that is, $q_{ii} = -\sum_n^{j=1} q_{ij,i\neq j}$.
Example. Consider CTMC depicted in Fig. 2.8. Its matrix of transition rates Q is given as follows.

$$Q = \begin{bmatrix} -(\lambda_3+\lambda_0) & \lambda_3 & \lambda_0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -(\lambda_0+\lambda_1) & 0 & \lambda_0 & \lambda_1 & 0 & 0 & 0 \\ 0 & 0 & -(\lambda_3+\lambda_1) & \lambda_3 & 0 & \lambda_1 & 0 & 0 \\ 0 & 0 & 0 & -(\lambda_4+\lambda_1) & 0 & 0\lambda_4 & \lambda_1 & \\ \lambda_5 & 0 & 0 & 0 & -(\lambda_5+\lambda_0) & 0 & \lambda_0 & 0 \\ \lambda_2 & 0 & 0 & 0 & 0 & -(\lambda_2+\lambda_3) & 0 & \lambda_3 \\ 0 & 0 & \lambda_5 & 0 & 0 & 0 & -\lambda_5 & 0 \\ 0 & \lambda_2 & 0 & 0 & 0 & 0 & 0 & -\lambda_2 \end{bmatrix}.$$

Given a matrix of transition rates, the following step is computing the steady-state distribution $\Pi = [\pi_1, \pi_2, \ldots, \pi_n]$, i.e., the probability of being in reachable markings $(s_1, s_2, \ldots, s_n)$, respectively. To this end, we resolve the following equation system

$$\Pi \times Q = 0, \text{ such that } \sum_{n}^{i=1}(\pi_i) = 1. \tag{2.5}$$

Example. Consider CTMC depicted in Fig. 2.8. Its steady-state distribution $\Pi = [\pi_0, \ldots, \pi_7]$ is given as follows.

$$[\pi_0, \pi_1, \ldots, \pi_7] \quad \times \quad \begin{bmatrix} q_{00} & \cdots & q_{07} \\ \vdots & \ddots & \vdots \\ q_{70} & \cdots & q_{77} \end{bmatrix} = [0,0,0,0,0,0,0], \text{ such that } \sum_{7}^{i=0}(\pi_i) = 1.$$

2.6.4 Quantitative Properties

After computing the steady-state distribution $\Pi = [\pi_1, \pi_2, \ldots, \pi_n]$, we can then evaluate the following quantitative properties [BK02, Mar+94].

Probability of being in a subset of markings: Let $B \subseteq RS(SPN)$ be some markings of interest. Then, the probability of being in a state of the corresponding subset is given by:

$$P[B] = \sum_{s_i \in B} \pi_i. \tag{2.6}$$

Mean number of tokens: Let $B(p, n)$ denote the subset of $RS(\mathcal{N})$ for which the number of tokens in place p is n, and m_p denote the maximum number of tokens that can present on place p. Then, the average number of tokens in place p is given by:

$$\bar{m} = \sum_{m_p}^{n=1}(nP[B(p,n)]). \tag{2.7}$$

Probability of firing transition t_j: Let EN_t denote the subset of $RS(\mathcal{N})$ in which a given transition t is enabled. Then, the probability r that an observer, who looks randomly into the SPN, sees transition t firing next is given by:

$$r = \sum_{s_i \in EN_t} \pi_i \left(\frac{\lambda_t}{-q_{ii}} \right) = \sum_{s_i \in EN_t} \pi_i P[t \text{ fires first at } s_i]. \tag{2.8}$$

Throughput at a transition t: The throughput at a transition t is given by:

$$d = \sum_{s_i \in EN_t} \pi_i \lambda_t. \tag{2.9}$$

2.7 Generalized Stochastic Petri Nets

Aforementioned, the firing delay of any transition in SPNs is stochastic and exponentially distributed to model time-consuming activities. Nevertheless, not all activities in real systems are so. For instance, the verification of a logical condition can be proceeded in no time. Hence, the second type of transition, called immediate, is introduced into SPNs, in order to separate the two aspects in the modeling paradigm. Moreover, the introduction of immediate transitions in SPNs does not increase, significantly, the analysis complexity [Mar+94]. Stochastic Petri nets in which transitions are either immediate or timed are called *generalized SPNs* (GSPNs).

In GSPNs, immediate transitions (depicted as filled bars) fire with priority over timed transitions (depicted as unfilled bars), in zero time, as soon as they become enabled. In fact, enabling an immediate transition disables all timed transitions, i.e., timed transitions in a GSPN are disabled if there exists an enabled immediate transition. Considering firing rule, GSPNs follow the same firing rule of basic Petri nets.

Consider GSPN $\mathcal{G}$ with marking M depicted in Fig. 2.9. Transitions $\texttt{enter}_1$ and $\texttt{enter}_2$ are immediate transitions, and the other transitions are timed. In fact, transition $\texttt{enter}_1$ models the verification of a logical condition (if process $\texttt{p}_1$ is waiting for critical resource and the critical resource is free). This action is not a time-consuming activity, therefore it is natural to model it by an immediate transition that fires in zero time. Similarly to transition $\texttt{enter}_2$ with respect to process $\texttt{p}_2$. At marking M, only transition $\texttt{enter}_1$ is enabled (which disables all timed transitions including $\texttt{request}_2$). Firing $\texttt{enter}_1$ consumes one token from place `res` and one token from place $\texttt{wait}_1$; and produces one token into place $\texttt{CS}_1$. After firing $\texttt{enter}_1$, transition $\texttt{request}_2$ becomes enabled.

GSPNs represent an extension of SPNs, where transitions are divided into two sets: a set T_1 of timed transitions and a set T_2 of immediate transitions.

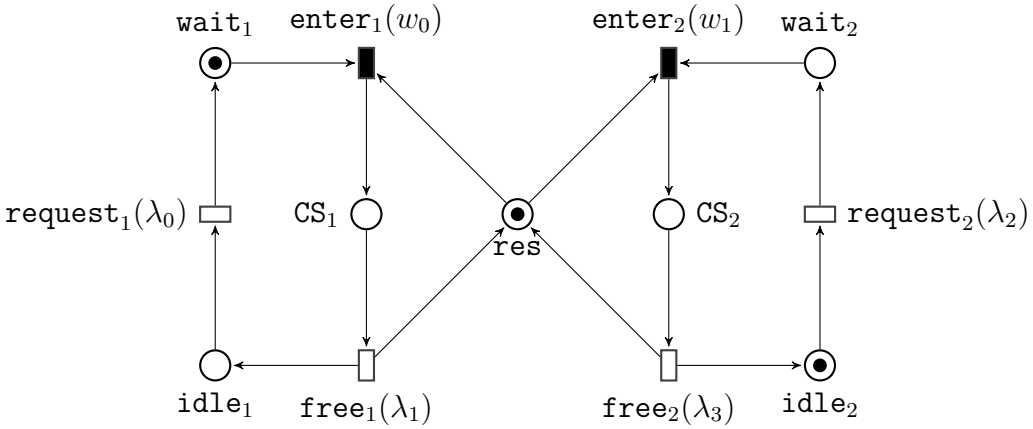


Figure 2.9: GSPN $\mathcal{G}$ models mutual exclusion.

Definition 2.13. Generalized Stochastic Petri Net. A generalized stochastic Petri net is 7-tuple $\mathcal{G} = \langle P, T, F, M_0, T_1, T_2, \Lambda \rangle$ where:

1. P is a finite and non-empty set of places,
2. T is a finite and non-empty set of transitions disjoint from P,
3. $F : (P \times T) \cup (T \times P) \longrightarrow \mathbb{N}$ is a flow relation for a set of arcs,
4. $M_0 : P \longrightarrow \mathbb{N}$ is an initial marking,
5. $T_1 \subseteq T$ is a set of timed transitions,
6. $T_2 \subset T$ is a set of immediate transitions, such that $T_1 \neq \emptyset, T_1 \cap T_2 = \emptyset, T_1 \cup T_2 = T$,
7. $\Lambda : T \to \mathbb{R}^+$, where $\Lambda(t_i)$ is a firing rate/weight of t_i.

In GSPNs, firing an immediate transition t at a marking M_v yielding a marking M'_v is performed in zero time, hence, if an observer that looks to the net marking will not see M_v since the latter will be *vanished* instantaneously. In other words, when the net reaches marking M_v in any instant d, transition t fires immediately and the new net marking becomes M'_v in same instant d, thus the sojourn time in marking M_v is null. Such markings are called *vanishing markings*. A vanishing marking is a marking that enables an immediate transition.

On the other hand, a *tangible marking* is a marking that either enables a timed transition or is a dead-end marking. The stochastic process sojourns in such markings are exponentially distributed. Therefore, these markings are not left immediately.

Another concept raises from introducing immediate transition which is *timeless trap*. A GSPN is said to have a timeless trap if it can reach a marking that enables an infinite fireable sequence of immediate transitions. Indeed, when a GSPN is in a timeless trap, it means that no timed transition can be fired in the future.

Consider stochastic process illustrated in Fig. 2.10, where its states are listed in Table 2.2. Marking s_1 and s_2 are vanishing markings. Indeed, both of them enable immediate transitions. Since s_1 and s_2 are the only vanishing markings and the only markings that describe states in which a process is waiting to access a free critical resource, then the probability that a random observer sees such situation is null.

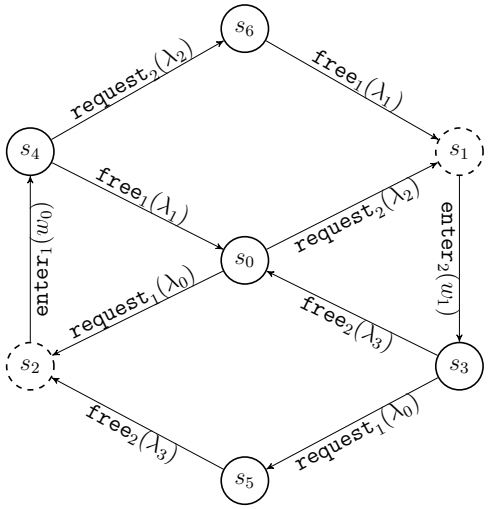


Figure 2.10: Corresponding stochastic process of GSPN $\mathcal{G}$ shown in Fig. 2.9.

	$\texttt{CS}_1$	$\texttt{CS}_2$	$\texttt{idle}_1$	$\texttt{idle}_2$	res	$\texttt{wait}_1$	$\texttt{wait}_2$
s_0	0	0	1	1	1	0	0
s_1	0	0	1	0	1	0	1
s_2	0	0	0	1	1	1	0
s_3	0	1	1	0	0	0	0
s_4	1	0	0	1	0	0	0
s_5	0	1	0	0	0	1	0
s_6	1	0	0	0	0	0	1

Table 2.2: Reachable markings of $\mathcal{G}$.

2.7.1 Embedded Markov Chain

GSPNs do not describe a continuous-time Markov process. Indeed, sojourn time in vanishing marking is null and in tangible markings is exponentially distributed, hence we do not use sojourn time to analyze GSPNs, instead we consider the probability of going from a marking M (either vanishing or tangible) to marking M'. These probabilities are given by

$$P[M \longrightarrow M'] = \frac{\sum_{t_i \in \{M[t_i\rangle M'\}} \Lambda(t_i)}{\sum_{t_j \in EN(M)} \Lambda(t_j)}. \tag{2.10}$$

where $EN(M)$ denotes the set of enabled transitions at marking M.

These probabilities are independent of the sojourn time in marking M. Therefore, we analyze the embedded Markov chain of the corresponding stochastic process of GSPNs.

Let $\hat{T}$ denote tangible state set, $\hat{V}$ vanishing state set, $m = |\hat{V}|$, and $n = |\hat{T}|$. Using embedded Markov chain, the *transition probability matrix* is given as follows [BK02]:

$$P = \left[\begin{array}{ccc|ccc} c_{11} & \cdots & c_{1m} & d_{11} & \cdots & d_{1n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ c_{m1} & \cdots & c_{mm} & d_{m1} & \cdots & d_{mn} \\ \hline e_{11} & \cdots & e_{1m} & f_{11} & \cdots & f_{1n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ e_{n1} & \cdots & e_{nm} & f_{n1} & \cdots & f_{nn} \end{array}\right]. \tag{2.11}$$

where

$$c_{ij} = P[M_i \longrightarrow M_j | Mi \in \hat{V} \wedge M_j \in \hat{V}], \quad d_{ij} = P[M_i \longrightarrow M_j | Mi \in \hat{V} \wedge M_j \in \hat{T}],$$
$$e_{ij} = P[M_i \longrightarrow M_j | Mi \in \hat{T} \wedge M_j \in \hat{V}], \quad f_{ij} = P[M_i \longrightarrow M_j | Mi \in \hat{T} \wedge M_j \in \hat{T}].$$

That is, the upper left side describes the transition probabilities between vanishing states, the upper right side describes transition probabilities from vanishing states to tangibles one, the bottom left side describes transition probabilities from tangibles states to vanishing ones, and finally, the bottom right side describes the transition probabilities between tangibles states.

Example: Assume mean firing rates $\lambda_0 = 6, w_0 = 300, \lambda_1 = 3, \lambda_2 = 2, w_1 = 300$ and $\lambda_3 = 1$. Solving the equation system we obtain the following steady-state distribution:

$$\begin{array}{rc} & \begin{array}{ccccccc} s_1 & s_2 & s_0 & s_3 & s_4 & s_5 & s_6 \end{array} \\ \text{Vanishing states} \left\{\begin{array}{c} s_1 \\ s_2 \end{array}\right. & \left[\begin{array}{ccccccc} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{array}\right] \\ \text{Tangible states} \left\{\begin{array}{c} s_0 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \end{array}\right. & \left[\begin{array}{ccccccc} \frac{2}{8} & \frac{6}{8} & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{7} & 0 & 0 & \frac{6}{7} & 0 \\ 0 & 0 & \frac{3}{5} & 0 & 0 & 0 & \frac{2}{5} \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \end{array}.$$

The steady-state distribution $\tilde{\Pi} = [\tilde{\pi}_1, \tilde{\pi}_2, \ldots, \tilde{\pi}_{m+n}]$ – of being in states (either vanishing or tangible) $s_1, s_2, \ldots, s_{m+n}$, respectively – of the embedded Markov chain is given by

$$\tilde{\Pi} \times P = \tilde{\Pi}, \text{ such that } \sum_{m+n}^{i=1} (\tilde{\pi}_i) = 1. \tag{2.12}$$

From this steady-state distribution, we can compute the steady-state distribution of the original stochastic process as follows:

$$\pi_i = \begin{cases} \dfrac{\tilde{\pi}_i \times \Big(\sum\limits_{t_j \in ET(M_i)} w_j\Big)^{-1}}{\sum\limits_{M_k \in \hat{T}} \tilde{\pi}_k \times \Big(\sum\limits_{t_j \in ET(M_k)} w_j\Big)^{-1}} & \text{if } M_i \in \hat{T} \\ 0 & \text{otherwise.} \end{cases} \tag{2.13}$$

Given the steady-state distribution of tangible states, we can compute quantitative properties presented in Section 2.6.4. Note that throughput and firing probability concern only timed transitions.

Example: Consider GSPN $\mathcal{G}$ illustrated in Fig. 2.9.

The steady-state distribution $\tilde{\Pi} = [\tilde{\pi}_1, \tilde{\pi}_2, \tilde{\pi}_0, \tilde{\pi}_3, \tilde{\pi}_4, \tilde{\pi}_5, \tilde{\pi}_6]$ of its embedded Markov chain is calculated by resolving the following equation system

$$[\tilde{\pi}_1, \tilde{\pi}_2, \tilde{\pi}_0, \tilde{\pi}_3, \tilde{\pi}_4, \tilde{\pi}_5, \tilde{\pi}_6] \quad \times \quad \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \frac{2}{8} & \frac{6}{8} & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{7} & 0 & 0 & \frac{6}{7} & 0 \\ 0 & 0 & \frac{3}{5} & 0 & 0 & 0 & \frac{2}{5} \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = [\tilde{\pi}_1, \tilde{\pi}_2, \tilde{\pi}_0, \tilde{\pi}_3, \tilde{\pi}_4, \tilde{\pi}_5, \tilde{\pi}_6].$$

such that $\sum_6^{i=0}(\tilde{\pi}_i) = 1$.

That is,

$$[\tilde{\pi}_1, \tilde{\pi}_2, \tilde{\pi}_0, \tilde{\pi}_3, \tilde{\pi}_4, \tilde{\pi}_5, \tilde{\pi}_6] \quad \times \quad \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ \frac{2}{8} & \frac{6}{8} & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{1}{7} & 0 & 0 & \frac{6}{7} & 0 & 1 \\ 0 & 0 & \frac{3}{5} & 0 & 0 & 0 & \frac{2}{5} & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = [\tilde{\pi}_1, \tilde{\pi}_2, \tilde{\pi}_0, \tilde{\pi}_3, \tilde{\pi}_4, \tilde{\pi}_5, \tilde{\pi}_6, 1].$$

Solving the equation system, we obtain the following steady-state distribution of the embedded Markov chain:

$$\tilde{\pi}_1 = P[(0,0,1,0,1,0,1)] = 0.1450,$$
$$\tilde{\pi}_2 = P[(0,0,0,1,1,1,0)] = 0.1884,$$
$$\tilde{\pi}_0 = P[(0,0,1,1,1,0,0)] = 0.0960,$$
$$\tilde{\pi}_3 = P[(0,1,1,0,0,0,0)] = 0.1450,$$
$$\tilde{\pi}_4 = P[(1,0,0,1,0,0,0)] = 0.1884,$$
$$\tilde{\pi}_5 = P[(0,1,0,0,0,1,0)] = 0.1242,$$
$$\tilde{\pi}_6 = P[(1,0,0,0,0,0,1)] = 0.1130.$$

Using Eq.2.13, we obtain the following steady-state distribution $[\pi_0, \pi_1, \pi_2, \pi_3, \pi_4]$, probabilities of being in tangible states s_0, s_3, s_4, s_5 and s_6 respectively, of the original stochastic process:

$$\pi_0 = P[(0,0,1,1,1,0,0)] = 0.0427,$$
$$\pi_1 = P[(0,1,1,0,0,0,0)] = 0.0830,$$
$$\pi_2 = P[(1,0,0,1,0,0,0)] = 0.1507,$$
$$\pi_3 = P[(0,1,0,0,0,1,0)] = 0.4975,$$
$$\pi_4 = P[(1,0,0,0,0,0,1)] = 0.2261.$$

2.8 Conclusion

In this chapter, we have presented Petri nets that constitute the basic form of other classes of Petri nets. Their intuitive aspects of modeling have been discussed. As well, the formal modeling and verification based on Petri net are presented.

Stochastic Petri nets that provide a common extension of Petri nets combining qualitative and quantitative verification are introduced. Finally, several of their underlying concepts such as Markov chains, steady-state distribution, etc. are illustrated via certain examples.

In spite of their modeling and decision power, Petri nets face several shortcomings in the design and analysis of *dynamic-structure systems*. Hence, it is required to introduce dynamic structures to Petri nets in order to model/verify such systems in a natural way. In the next chapter, we are interested in the extension of Petri nets onto dynamic-structure formalisms.

Chapter 3

Reconfiguration in Petri Nets

3.1 Introduction

Nowadays, many discrete-event systems (DESs) are becoming increasingly complex, structurally dynamic and variably interconnected. These systems are designed to be able to change their structure and/or topology, at run-time, by adding/removing interconnections, objects, or even subsystems, to accommodate new circumstances/requirements. Ongoing studies on this class of systems focus on their key feature, namely, the reconfigurability [Bre+14] that must be performed at run-time (i.e., dynamic reconfigurability) [JEB16].

In fact, many researchers have used Petri nets to study such systems [Mar+94, Lat+18]. In the literature, PNs have been extended to several classes that are proposed and applied to model and verify reconfigurable systems.

GSPNs and SPNs, presenting common extensions of PNs, are important and versatile tools [Mar+94] that fit well with the behavior of DESs at different stages of development [LZB15, Čap17, Sim+18, Lat+18]. However, PNs (low- or high-level) are not suitable in the modeling and verification of advanced systems having dynamic structures [CC18]. Indeed, systems supporting volatile environments, continuous variations, and reconfigurable structures are expected to be extremely complex [Chr+13].

In fact, modeling reconfigurable systems with basic PNs (i.e., non-reconfigurable) makes the designer's tasks even more complicated and thus the resulting models will often be very large, complicated and difficult to assimilate. The analysis of such models can only be more complicated too. Indeed, basic PNs are characterized by their rigid structure impeding modeling, verification, simulation and visualization of the dynamic structure of this class of systems. To overcome this issue, researchers introduce dynamic structures into PNs, thus expanding the standard formalism [PK18].

On the other hand, rule-based graph transformations [EP04] offer an intuitive and mathematically based graphical framework for modeling the reconfigurations of PN structures. Nevertheless, increasing the modeling power of a formalism decreases its decision power. Therefore, extensions proposed in the literature on reconfigurability in PNs try to find a trade-off between modeling and verification levels.

The idea behind reconfigurable PNs is to get closer to real dynamic systems and to offer realistic models reflecting the inherent aspects of these systems. However, these gains in modeling are to the expense of analysis level such that some, or even all, properties become undecidable. This last gap has not prevented the development of this category of formalisms and considerable research is being conducted at the analysis level.

In the following, we introduce graph transformation systems in Section 3.2. Then, we present the common reconfigurable extensions of PNs in Sections 3.3–3.8. Different choices and properties of these extensions are discussed in Section 3.9. Finally, we conclude by classifying these extensions according to their modeling/verification limits in Section 3.10.

3.2 Graph Transformation Systems

Graph transformation systems (GTSs) offer an intuitive framework for the modeling of dynamic-structure systems. GTSs consist of a start graph G_0 that models an initial configuration, i.e. initial structure, of a system $\mathcal{G}$ and a set of rewriting rules $\mathcal{R}$ that expresses the structure evolution, i.e., a system reconfiguration. Each transformation rule consists of a left-hand side (LHS) and a right-hand side (RHS).

Assume that a graph G is the current configuration of $\mathcal{G}$, roughly speaking a rule is applicable when an occurrence of its LHS is located in G. This occurrence is located by a graph morphism that maps the LHS of a rule into a sub-graph of G. Applying a rule removes the occurrence of its LHS from G and adds its RHS to G. In some GTSs, an interface graph is provided to describe some parts to be preserved and to specify the attachment of RHS to the remaining part of the reconfigured graph [Kön04].

Definition 3.1. Graph Transformation System. A graph transformation system (GTS) $\mathcal{G} = \langle G_0, \mathcal{R} \rangle$ consists of a start graph G_0 and a set of rewriting rules $\mathcal{R}$. A graph G is generated by $\mathcal{G}$ if G is obtained by applying a set of rules in $\mathcal{R}$ to G_0.

In the following, we present briefly two graph transformation approaches, namely double-pushout (DPO) and net rewriting system (NRS).

3.3 Double-Pushout Approach for Petri Nets

In the field of graph transformation, the double-pushout (DPO) approach is considered as the most prominent technique which offers a theoretically founded and tool-supported framework in a rigorous way [KL18].

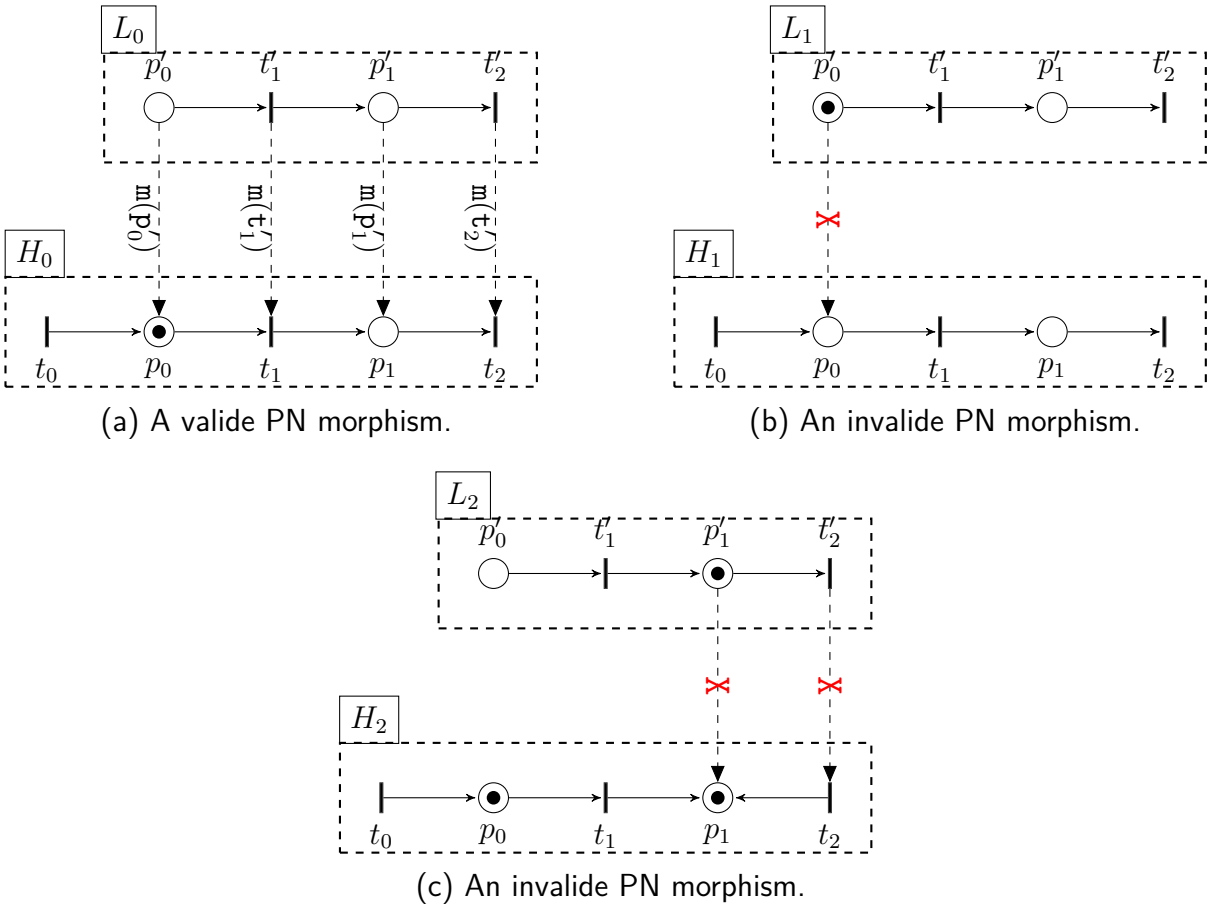
DPO approach uses graph morphisms that map transitions to transitions and places to places, so that if place p is mapped to place p', then a mapping between their preset and postset must exist (similarly to transitions).

Definition 3.2. PN morphism. Let $G = \langle P_G, T_G, F_G, M_G^0 \rangle$ and $H = \langle P_H, T_H, F_H, M_H^0 \rangle$ be two PNs. A Petri net morphism $\varphi : G \to H$ is a pair of mappings $\varphi_P : P_G \to P_H, \varphi_T : T_G \to T_H$, where $\forall\ t \in T_G$ and $\forall\ p \in P_G$, it holds the following [HEM05, EHP06]:

- $F_G(p,t) = F_H(\varphi_P(p), \varphi_T(t))$,
- $F_G(t,p) = F_H(\varphi_T(t), \varphi_P(p))$,
- $M_G^0(p) \leq M_H^0(\varphi_P(p))$.

Example. Consider PNs shown in Fig. 3.1.

An occurrence of PN L_0 is located in H_0 such that p'_0, t'_1, p'_1 and t'_2 in L_0 are mapped to p_0, t_1, p_1 and t_2 in H_0, respectively. On the other hand, place p'_0 in L_1 cannot be mapped to p_0 in H_1, since $M_{L_1}(p'_0) \not\leq M_{H_1}(p_0)$. As well, place p'_1 and transition of L_2 cannot be mapped to place p_1 and transition t_2 of H_2, since $F_{L_2}(p'_1, t'_2) \neq F_{H_2}(p_1, t_2)$.



(a) A valide PN morphism.

(b) An invalide PN morphism.

(c) An invalide PN morphism.

Figure 3.1: PN morphisms.

Definition 3.3. Transformation Rule. In DPO, transformations are rule-based where each graph transformation rule ω consists of three graphs L, I, R ($\omega = \langle L \xleftarrow{\varphi_l} I \xrightarrow{\varphi_r} R \rangle$); and two PN morphisms φ_l and φ_r, such that:

- L is called the left-hand side (a structure to be deleted),
- I is called the common interface,
- R is called the right-hand side (a structure to be inserted),
- $\varphi_l : I \to L$ is a PN morphism that maps I to L,
- $\varphi_r : I \to R$ is a PN morphism that maps I to R.

Given a rule $\omega = \langle L \xleftarrow{\varphi_l} I \xrightarrow{\varphi_r} R \rangle$, all elements in LHS not belonging to image of φ_l are called *obsolete* and all elements in RHS side not belonging to image of φ_r are called *fresh*.

Informally, applying a rule $\omega = \langle L \xleftarrow{\varphi_l} I \xrightarrow{\varphi_r} R \rangle$ to a PN G means to substitute some elements of its left-hand side L by some elements of its right-hand side R. The first step in applying ω is to find a match of L in G by a morphism m. If PN morphisms m, φ_l and φ_r are injective, then, we remove the images of obsolete nodes of L (resulting in a net called context) and insert fresh elements of R, this transformation is denoted by $G \overset{\omega,m}{\Longrightarrow} H$, such that H is the resulting PN. The removed elements are called obsolete and the inserted ones are called fresh. In this thesis, we limit our attention to injective PN morphisms.

A DPO transformation can be presented graphically by a diagram. Fig. 3.2 depicts an example of a DPO-based PN transformation.

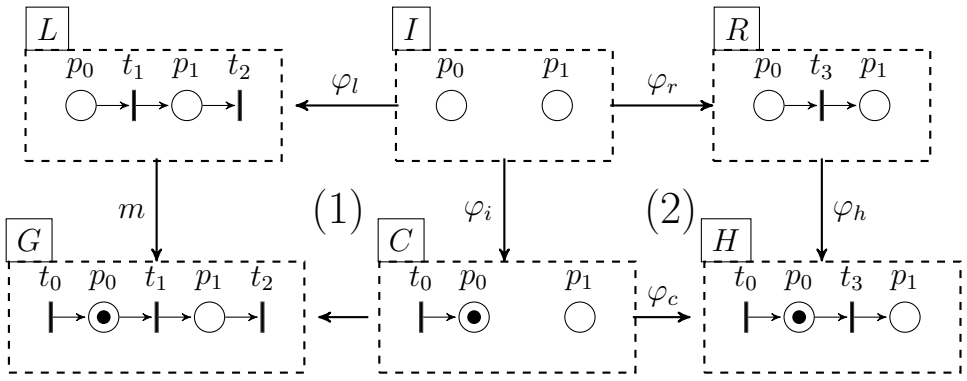


Figure 3.2: DPO diagram.

In fact, a DPO transformation consists of two *graph gluing constructions* (conventionally called *pushout*). Intuitively, the first pushout (as depicted by Box (1) in Fig. 3.2) determines the existing of a (unknown) PN C (called *context*) that can be glued with L over common interface I, where resulting PN is G, denoted by $G = C +_I L$. If such PN exists, then such a rule can be applied to G. The second pushout (as highlighted by Box (2) in Fig. 3.2) determines resulting PN H by gluing C with R over common interface I, denoted by $H = C +_I R$.

PN $H = \langle P_H, T_H, F_H, M_H^0 \rangle$ obtained by a graph gluing is given as follows.

1. $P_H = P_C \uplus (P_R \setminus P_I)$.
2. $T_H = T_C \uplus (T_R \setminus T_I)$.
3. F_H and M_H^0 are given by Eqs.(3.1) and (3.2), respectively.

$$F_H(v,w) = \begin{cases} F_C(v,w) & \text{iff } v,w \in P_C \cup T_C \\ F_R(v,w) & \text{iff } v,w \in P_R \cup T_R \\ 0 & \text{otherwise} \end{cases} . \tag{3.1}$$

$$M_H^0(p) = \begin{cases} M_C^0(p) & \text{iff } p \in P_C \setminus P_R \\ M_R^0(p) & \text{iff } p \in P_R \setminus P_C \\ M_C^0(p) + M_R^0(p) - M_I^0(p) & \text{otherwise} \end{cases} . \quad (3.2)$$

Let us consider the rule shown in Fig. 3.2. An occurrence of L is located in G by a PN morphism m, where $m(p_0) = p_0, m(t_1) = t_1, m(p_1) = p_1$, and $m(t_2) = t_2$. Then, a context net C is defined by deleting images of obsolete nodes of L (which are $\{t_1, t_2\}$) from net G. Afterward, we insert fresh nodes of R (which are $\{t_3\}$) and nodes of C into H.

Finally, flow function and net marking of new net H are computed according to Eqs.(3.1) and (3.2), respectively. For instance, (i) the connection between t_0 and p_0 in H is equal to the connection between them in C, since both t_0 and p_0 belong to net C, (ii) the connection between t_3 and p_1 in H is equal to the connection between them in R, since both t_3 and p_1 belong to net R, (iii) $M_H^0(p_0) = M_C^0(p_0) + M_R^0(p_0) - M_I^0(p_0) = 1$, etc.

3.4 Net Rewriting Systems

In order to improve the expressiveness of the basic PNs formalism, net rewriting systems (NRSs) are introduced in [LO04a, LO04b] by merging Petri nets with GTSs. NRS formalism is similar to DPO for PNs except that (i) LHS and RHS must be unmarked Petri nets, (ii) all nodes in LHS are obsolete, (iii) all nodes in RHS are fresh, and (iv) the interface is modeled as transfer relations rather than a PN.

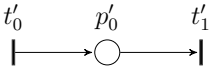
Definition 3.4. Net Rewriting System. A net rewriting system (NRS) $\mathcal{N} = \langle G_0, M_0, \mathcal{R} \rangle$ consists of a start unmarked PN G_0, an initial marking M_0, and a set of rewriting rules $\mathcal{R}$.

Definition 3.5. Rewriting Rule. A rewriting rule $r \in \mathcal{R}$ is a structure $r = \langle L, R, I, O \rangle$, such that:

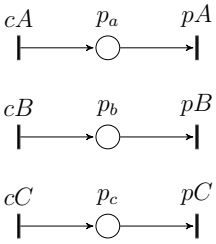
- $L = \langle P_L, T_L, F_L \rangle$ is an unmarked PN called left-hand side,
- $R = \langle P_R, T_R, F_R \rangle$ is an unmarked PN called right-hand side,
- $IR \subseteq (P_L \times P_R) \cup (T_L \times T_R)$ is a binary relation called interface relation,
- $I \subseteq (P_L \times P_R) \cup (T_L \times T_R)$ is a binary relation called input interface relation,
- $O \subseteq (P_L \times P_R) \cup (T_L \times T_R)$ is a binary relation called output interface relation.

Example. Consider NRS rule r_0 depicted in Fig. 3.3. Their LHS and RHS are unmarked PN shown in Figs 3.3a and 3.3b, respectively. Its interface relations relates transition t_0' to transitions cA, cB and cC; transition t_1' to transitions pA, pB and pC; and place p_0' to places p_a, p_b and p_c.

$$I = \Big\{(t'_0, cA), (t'_0, cB), (t'_0, cC)\Big\}.$$



(a) LHS L_0.



(b) RHS R_0.

$$IR = \Big\{(t'_0, cA), (t'_0, cB), (t'_0, cC), (p'_0, p_a), (p'_0, p_b), (p'_0, p_c), (t'_1, pA), (t'_1, pB), (t'_1, pC)\Big\}.$$

$$O = \Big\{(t'_1, pA), (t'_1, pB), (t'_1, pC)\Big\}.$$

(c) Interface relations.

Figure 3.3: NRS rule r_0.

Similarly to DPO, applying a rule $r = \langle L, R, I, O\rangle$ to a PN G removes the image of left-hand side L and add the right-hand side R. To apply r, first we find a match $\mathcal{M}$ of L in G by a morphism m without considering the marking since L is an unmarked PN. The node of G not belonging to $\mathcal{M}$ constitute the context graph. Then, we check the following conditions, (i) a node n of match $\mathcal{M}$ may belong to postset of a node in the context graph iff n belongs to I, and (ii) a node n' of match $\mathcal{M}$ may belong to preset of a node in the context graph iff n' belongs to O. After replacing the match M by R, the marking M_H and the flow function F_H of the obtained net H is defined by the following equations.

Applying rewriting rule $r = \langle L, R, , I, O\rangle$ on PN G with marking M leads to new PN $H = \langle P_H, T_H, F_H, M^0_H\rangle$. Marking M^0_H of H is given by

$$M^0_H(p) = \begin{cases} M(p) & \text{if } p \notin P_R \\ \displaystyle\sum_{p' \in \pi(p)} M(m(p')) & \text{if } p \in P_R \end{cases}. \tag{3.3}$$

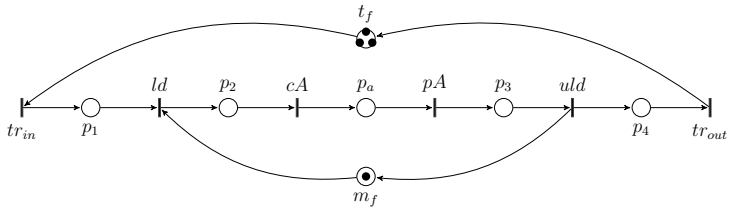
where $\pi(p) = \{p' | \exists (p', p) \in I \cup O\}$ and m is the morphism that maps L to $\mathcal{M}$.

For each couple of nodes $(v, w) \in (P_H \times T_H) \cup (T_H \times P_H)$, $F_H(v, w)$ is defined as follows.

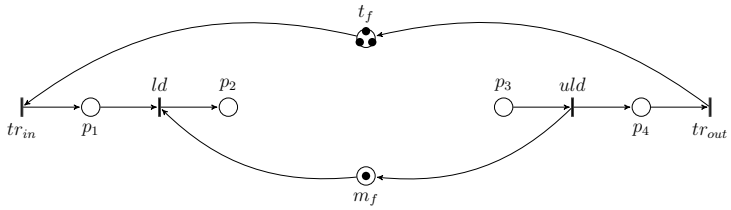
$$F_H(v,w) = \begin{cases} F_G(v,w) & \text{if } v \notin N_R \wedge w \notin N_R \\ F_G(v,w) & \text{if } v \in N_R \wedge w \in N_R \\ \displaystyle\sum_{w' \in \pi_I(w)} F_G(v, m(w')) & \text{if } v \notin N_R \wedge w \in N_R \\ \displaystyle\sum_{v' \in \pi_O(v)} F_G(m(v), w) & \text{if } v \in N_R \ \wedge w \notin N_R \end{cases}. \tag{3.4}$$

where $N_R = P_R \cup T_R$, $\pi_I(p) = \{p' | \exists (p', p) \in I\}$ and $\pi_O(p) = \{p' | \exists (p', p) \in O\}$.

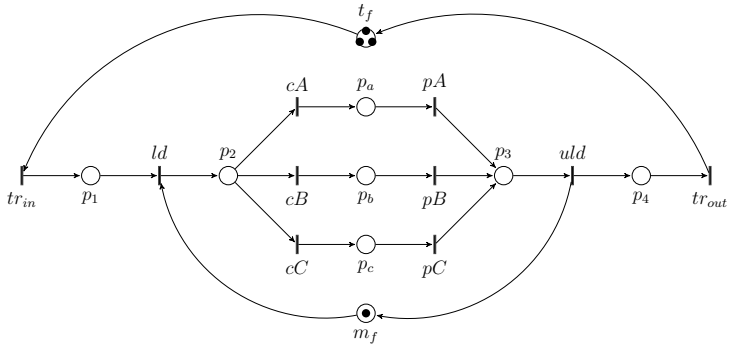
Example. Consider r_0 application depicted in Fig. 3.4 to G_0 shown in Fig. 3.4a.



(a) Source PN G_0.



(b) Intermediate PN G'_0 after removing image of L_0.



(c) Target PN H.

Figure 3.4: An application of NRS rule r_0 to G_0.

First, a morphism m matches L_0 depicted in Fig. 3.3a to a sub-graph $m(L_0)$ in G_0 such that $m(t'_0) = cA, m(p'_0) = p_a$ and $m(t'_1) = pA$. Then, this match is removed from G_0 yielding to the intermediate graph shown in Fig. 3.4b. Finally, we add and connect R_0 depicted in Fig. 3.3b to the intermediate graph.

Consider the connection $F_H(p_2, cB)$ from place p_2 to transition cB. According to, Eq.(3.4), $F_H(p_2, cB) = \sum_{w' \in \pi_I(cB)} F_G(p_2, m(w'))$, since p_2 does not belong to R_0, whereas cB does it. We have $\pi_I(cB) = \{t'_0\}$, hence $F_H(p_2, cB) = F_G(p_2, m(t'_0)) = F_H(p_2, cB) = F_G(p_2, cA) = 1$. As for the marking of place p_c, it is computed according to Eq.(3.3) as follows. $M^0_H(p_c) = \sum_{p' \in \pi(p_c)} M(m(p'))$, since $p_c \in P_{R_0}$. We have, $\pi(p_c) = \{p'_0\}$, hence $M^0_H(p_c) = M(m(p'_0)) = M(p_a) = 0$.

Note that r_0 can be applied differently to G_0. Indeed, L_0 can be matched to other sub-graphs in G_0. For instance, there is another morphism m' matching t'_0, p'_0 and t'_1 in L_0 to tr_{in}, p_1, and ld, respectively.

3.5 Self-Modifying Nets

In [Val78b, Val78a], Valk introduced Self-Modifying nets (SM-nets) in order to increase the modeling power of Petri nets and discussed their decidability issues. SM-nets are considered as the first extension that introduced reconfigurability to Petri nets. In fact, SM-nets are able to modify their own enabling and firing rules by allowing to arcs to be weighted either by a natural number or in terms of the marking of a place belonging to the net.

Consider SM-net depicted in Fig. 3.5a in which the weight of the arc ongoing from place v to transition t_1, as well as the arc from transition t_1 to place w, equals the marking of place u. Hence, firing t_1 consumes three tokens (the current marking of place u) from place v and deposits, as well, three tokens at place w. Consider Fig. 3.5b, firing t_0 yields a new marking in which u is marked by two tokens. Hence, firing t_1 consumes two tokens from v and deposits two tokens at w.

Therefore, the enabling and firing rules of transition t_1 are modified after the firing t_0. Note that if the marking of place u is zero, transition t_1 is enabled, since the weight of the arc from v to t_1 becomes zero, that is, t_1 becomes a source transition (a source transition is a transition whose preset is an empty set).

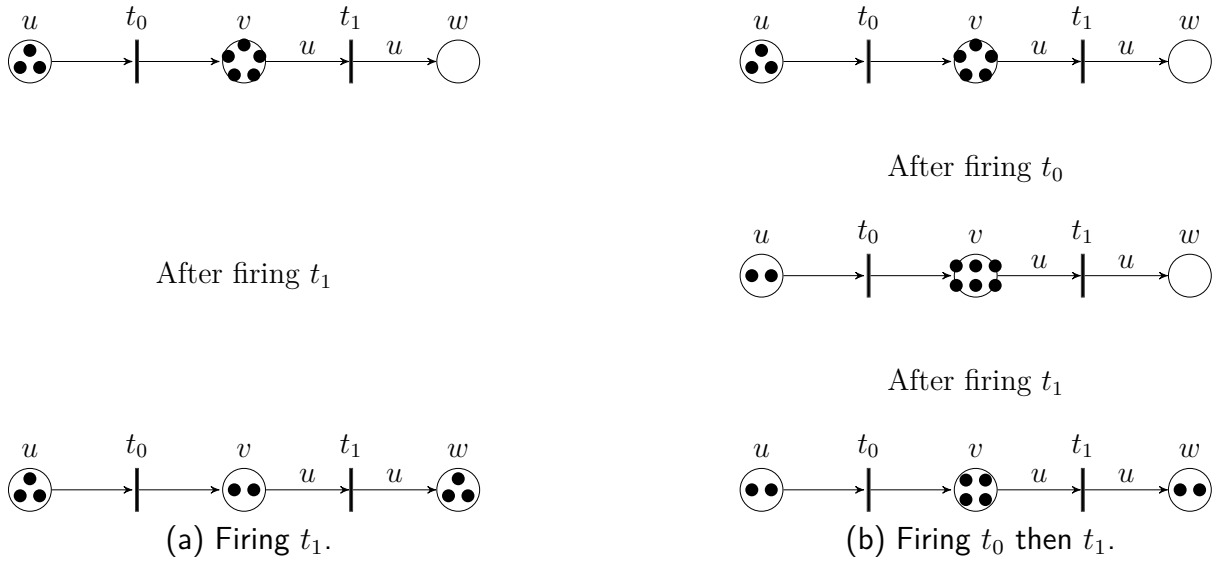


Figure 3.5: Self-modifying nets.

Definition 3.6. Self-Modifying Nets . A Self-Modifying net is a 4-tuple $\mathcal{SM} = \langle P, T, F, M_0 \rangle$ where

- P is a finite and non-empty set of places,
- T is a finite and non-empty set of transitions disjoint from P,

- $F : (P \times P_1 \times T) \cup (T \times P_1 \times P) \longrightarrow \mathbb{N}$ is a flow relation for a set of arcs, where $P_1 = P \cup \{1\}$ and $1 \notin P$,
- $F(x, 1, z) = m$ means that the multiplicity (weight) of the arc from x to z is m, and $F(x, y, z) = n$ means that the multiplicity of the arc from x to z is $n \cdot M(y)$, where $M(y)$ is the current marking of place y,
- $M_0 : P \longrightarrow \mathbb{N}$ is an initial marking.

Example. Consider the SM-net depicted in Fig. 3.5. Its formal definition $\mathcal{SM} = \langle P, T, F, M_0 \rangle$ can be given as follows.

- $P = \{u, v, w\}$,
- $T = \{t_0, t_1\}$,
- $F(u, 1, t_0) = 1, F(t_0, 1, v) = 1, F(v, u, t_1) = 1, F(t_1, u, w) = 1$, otherwise equals zero,
- $M_0(u) = 3, M_0(v) = 5$ and $M_0(w) = 0$.

Dufourd *et al.* have presented several sub-classes of SM-nets in [DFS98] and discussed the decidability issues of their properties. As well, they have shown that reachability, place-boundedness, boundedness, coverability, etc. are undecidable for SM-nets.

3.6 Reconfigurable Petri Nets

Reconfigurable Petri nets (RPNs), a subclass of NRSs, are proposed in [BLO03, LO04a, LO04b] which have the same modeling and decision power of PNs. They have proposed to explicitly the reconfiguration in structurally dynamic systems.

The reconfigurations in RPNs are expressed via a set of rewriting rules that can only modify the flow relations (i.e., sets of places and transitions are still unchanged after applying any rule) depending on net marking. In fact, RPNs can be seen as a merging of PNs with SM-nets.

Definition 3.7. Reconfigurable Petri Nets. An RPN is a structure $N = \langle P, T, \mathcal{R}, \gamma_0 \rangle$ where :

1. $P = \{p_0, \ldots, p_n\}$ is a non-empty and finite set of places,
2. $T = \{t_0, \ldots, t_m\}$ is a non-empty and finite set of transitions, such that $P \cap T = \emptyset$,
3. $\mathcal{R} = \{r_0, \ldots, r_k\}$ is a finite set of rewriting rules,
4. γ_0 is an initial configuration of net N.

Definition 3.8. Rules in RPNs. A rule r is a structure $r = \langle D, {}^\bullet r, r^\bullet, \mathbb{C}, \mathbb{M} \rangle$, such that:

1. $D \subseteq P$, such that D is the set of places whose connections will be changed by rule application,
2. ${}^\bullet r : (D \times T) \cup (T \times D) \to \mathbb{N}$ describes the flow relations of places in D before applying r,
3. $r^\bullet : (D \times T) \cup (T \times D) \to \mathbb{N}$ describes the flow relations of places in D after applying r,
4. $\mathbb{C} \subseteq D$, is the set of control places, and $\mathbb{M}$ is the required minimum marking of places of $\mathbb{C}$ so that the rule can be applied.

A rule r is applicable to N iff:

- The flow relations in N of places in D must be identical to ${}^\bullet r$,
- $\forall p \in \mathbb{C}$, $M(p) \geq \mathbb{M}(p)$, such that M is the current marking of N.

Example. Fig. 3.6a shows an example of an RPN N. This RPN is defined as $N = \langle P, T, \mathcal{R}, \gamma_0 \rangle$, where:

1. $P = \{p_0, p_1\}$, $T = \{t_0, t_1, t_2\}$, and γ_0 shown in Fig. 3.6a,
2. $\mathcal{R} = \{r = \langle D, {}^\bullet r, r^\bullet, \mathbb{C}, \mathbb{M} \rangle\}$, such that:
   - $D = P$, $\mathbb{C} = \{p_0\}$, $\mathbb{M}(p_0) \geq 3$,
   - ${}^\bullet r(t_i, p_i) = 1$ for $i = 0, 1$, ${}^\bullet r(p_j, t_{j+1}) = 1$ for $j = 0, 1$ and, otherwise, ${}^\bullet r(\cdot, \cdot) = 0$,
   - $r^\bullet(t_0, p_0) = 1$, $r^\bullet(p_0, t_2) = 1$ and, otherwise, $r^\bullet(\cdot, \cdot) = 0$.

Both functions ${}^\bullet r$ and $r^\bullet$ can be denoted simply by: $p_0(t_0 - t_1) + p_1(t_1 - t2) \triangleright p_0(t_0 - t_2) + p_1(\emptyset)$. Fig. 3.6b depicts the obtained configuration after applying rule r.

(a) Before reconfiguration. (b) After reconfiguration.

Figure 3.6: Reconfiguration after applying r.

In fact, the reconfigurations in RPNs are restricted to the topology level in order to be able to transform RPNs to equivalent basic PNs. Actually, the PNs are equipped with a set of algorithms and methods used to verify their qualitative properties either based on

its structure (i.e., structural analysis) or based on its reachability graph (i.e., behavioral analysis). Hence, RPNs are transformed into equivalent PNs, so that their analysis can be performed through analyzing their equivalent PNs.

Assume an RPN $N = \langle P, T, \mathcal{R}, C_0 \rangle$. RPN N can be reconfigured by adding and/or removing arcs. These transformations take place in order to enable, disable and/or modify conditions/effects (resp. pre/postconditions) of some actions (resp. transitions) in the reconfigurable system (resp. model). Hence, its equivalent G_e must be able to model these transformations and their effects, as well, the switching between configurations. The scratch of the transformation algorithm is given in the following. For further information, the reader is referred to [LO04a, LO04b].

Let $C_0, C_1, \ldots, C_n$ be a set of configurations, such that C_0 is an initial configuration and $C_1, \ldots, C_n$ the configurations obtained by applying rules in $\mathcal{R}$ to C_0.

To create an equivalent PN G_e, we clone all places in P with their initial marking in G_e. We associate to each configuration C_i a place $\mathring{p}_i$. A token in place $\mathring{p}_i$ signifies that the current configuration is C_i.

Considering the transitions. For each couple "transition t and configuration C_i", we add a transition t^i representing transition t as well as we preserve its flow relations in C_i. It is indispensable to connect t^i to $\mathring{p}_i$ with a self-loop in order to make t^i enable only if the current configuration is C_i. The purpose of duplicating t is to model the change in its pre/postconditions through reconfigurations.

Finally, each rule r is represented by a transition r_i^j, where C_i and C_j are their source and target configurations, respectively. The preconditions of transition r_i^j are: (i) the required marking for applying rule r, and one token in place $\mathring{p}_i$ (corresponding to source configuration C_i). The postconditions of transition r_i^j are: (i) the required marking for applying rule r, and one token in place $\mathring{p}_j$ (corresponding to target configuration C_j). In fact, firing transition r_i^j does not change the marking of places except those corresponding to source and target configurations.

Example. Consider PN shown in Fig. 3.7 obtained by transforming RPN N to a PN.

This PN is constructed such that

1. It contains all places in P of RPN N such that $P = \{p_0, p_1\}$,

2. Two places $\mathring{p}_0$ and $\mathring{p}_1$ corresponding to two configurations C_0 and C_1 depicted in Figs. 3.6a and 3.6b, respectively, are added such that initial marking of $\mathring{p}_0$ is one, since C_0 is the initial configuration,

3. A copy t_j^i of each transition t_j in each configuration C_i, is added such that t_j^i is connected to places according to the connections of t_j in C_i. For instance, t_1^0 models t_1 at

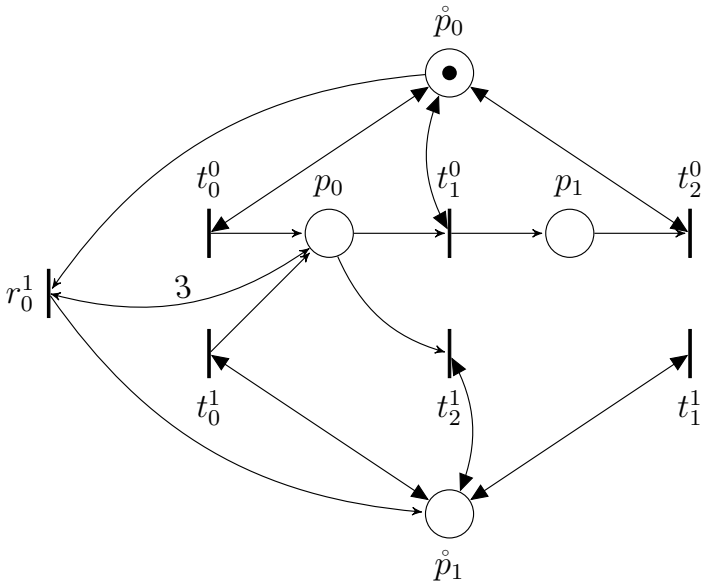


Figure 3.7: Equivalent PN to RPN N.

configuration C_0, hence, t_1^0 is connected by an arc ongoing from place p_0 to it, and an arc ongoing from it to place p_1,

4. Each transition t_j^i is connected to place $\mathring{p}_i$ by a self-loop in order to enable it only when the current configuration is C_i. For instance, transition t_1^0 is connected by a self-loop to place $\mathring{p}_0$ which enables it only when the current configuration is C_0,

5. Transition r_0^1 models rule r, such that a self-loop connecting r_0^1 with place p_0 that models the required marking of place p_0 to apply rule r. Furthermore, the arc from $\mathring{p}_0$ to r_0^1 means that its source configuration is C_0, and the arc from r_0^1 to $\mathring{p}_1$ means that its target configuration is C_1.

3.7 Improved Net Rewriting Systems

Aforementioned, net rewriting systems (NRSs) is an extension of PNs allowing modeling dynamic structures. Nevertheless, it does not consider the verification aspects. Hence, improved net rewriting systems (INRSs) are proposed in [LDM05, LDM08, Li+08, LDM09, Li+09] to allow reconfiguring live, bounded and reversible (LBR) PNs while preserving their LBR properties, i.e., the obtained PN after any reconfiguration is as well an LBR PN. Hence, it is no more required to re-verify these properties in the obtained PN after each reconfiguration. INRSs combine (i) NRSs as underlying formalism, (ii) well-behaved nets (WBNs) [SM83] to describe interface relations between left- and right hand sides, and (iii) certain subclasses of PNs to define left- and right-hand sides of rules.

In fact, WBNs were proposed to abstract or refine PNs (replace a transition by a subnet or vice-versa), while preserving certain properties such as liveness and boundedness. This approach is extended in INRSs to consider reconfiguration by replacing a subnet by another one while preserving LBR properties. Any subnet used in reconfiguration must belong to a set called *net block type library*. This set consists of certain subclasses of PNs [Mur89] such as state machines, marked graphs, etc. Some examples of net blocks are shown in Fig. 3.8. As for rule application, it is performed as specified in NRS approach.

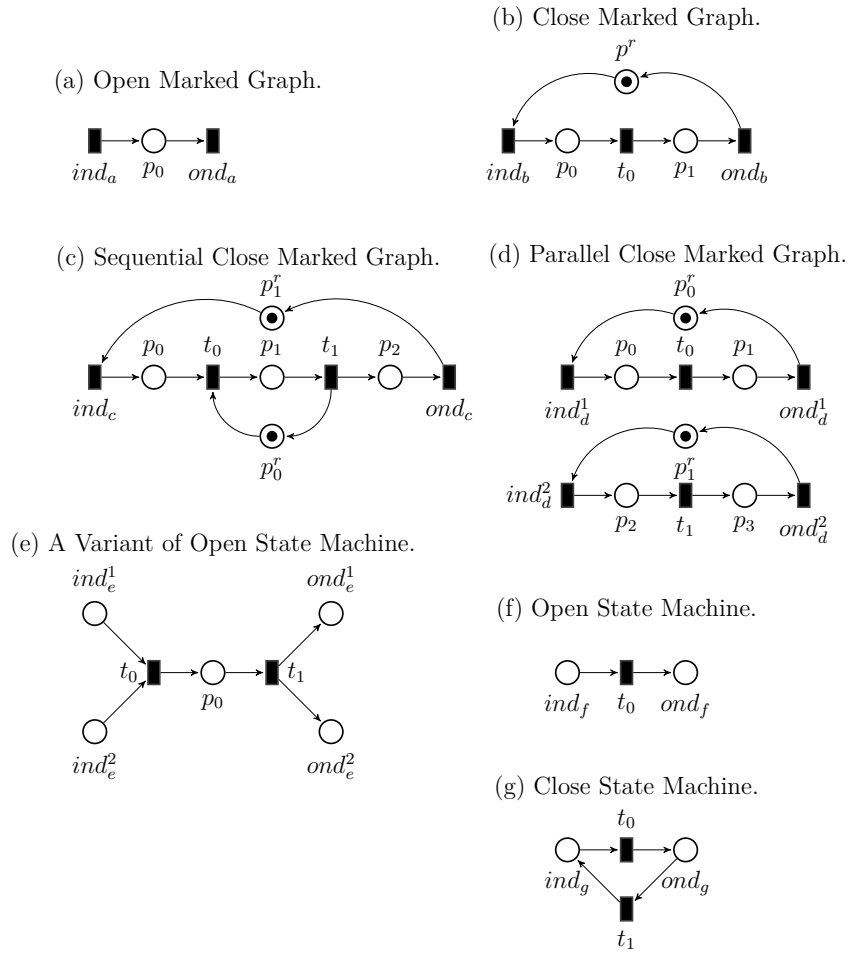


Figure 3.8: Net block library.

Definition 3.9. Improved Net Rewriting System. An INRS is triple-tuple $\mathcal{N} = \langle G_0, \mathcal{R}, \mathcal{L} \rangle$, such that

- $G_0 = \langle P, T, F, M_0 \rangle$ is a PN modeling an initial configuration,
- $\mathcal{R}$ is a finite set of rewriting rules,
- $\mathcal{L}$ is a net block type library for constraining the types of left- and right-hand sides of rewriting rules.

Definition 3.10. Rules in INRSs. A rule is a 4-tuple $r = \langle L, R, I, O \rangle$, such that

1. L and R (left- and right-hand side, respectively, of r) are two PNs belonging to $\mathcal{L}$,
2. I and O are input and output interface relations as defined in NRS formalism,
3. I and O are sets of either places or transitions.

Let $m(I_L)$ be a set of images of nodes in input interface of left-hand side of rule r and $m(O_L)$ be a set of images of nodes in output interface of right-hand side of rule r. A rule r is applicable to a given PN, in addition to conditions specified in NRS formalism, iff:

- If a node in the context graph belongs to the preset of a node in $m(I_L)$, then it must belong to the preset of each node in $m(I_L)$,
- If a node in the context graph belongs to the postset of a node in $m(O_L)$, then it must belong to the postset of each node in $m(O_L)$.

3.8 Other Extensions

Reconfigurable Object Nets. Reconfigurable object nets (RONs) are introduced in [HEM05] as high-level nets with "nets and rules as tokens". Actually, RONs extend the paradigm "nets as tokens" presented in [Val98, Val01, Val04]. RONs formalism distinguishes two classes of tokens (token-nets and token-rules).

As well, RONs formalism distinguishes between two net levels, namely, the system level and the token level. The system level is given by a high-level net and rule system. In system level, a place may contain token-nets or token-rules. The marking of the system level shows the distribution of nets and rules at different places. The firing behavior of the system level describes the movement of token-nets from place to place, as well as, structure changes of token-nets.

Obviously, a token-net can fire autonomously without being moved or transformed in order to describe a marking change at the token-level. A token-net firing can be synchronized with the occurrence of system transition, i.e., the token-net is moved or transformed iff some token-net transition occurs.

Evolving Petri Nets. The authors in [CC18] develop an emulator that encodes Petri nets and transformations as symmetric nets (SNs) [Chi+93]. Places and transitions are encoded as colors and arcs are represented by markings of specific places in SNs. Therefore, the changes in PN structure can be modeled by changing the marking of these specific places. However, the change in the set of places and transitions is not allowed since they are encoded as colors (in SNs, the color set is unchangeable).

Reconfigurable Discrete-Event Control Systems. Reconfigurable discrete-event control systems (R-DECSs) formalism is introduced in [Zha+13] to model reconfigurability in PNs. In this formalism, an R-DECS encompasses all possible reconfigurations in which a reconfigurable system can be. That is an R-DECS is a union of discrete-event control system models each of which describes a possible reconfiguration of the modeled system. The reconfiguration is emulated via transferring the system state from a source configuration to a target configuration.

Considering the verification level, an optimized analysis method is proposed in such a way that only modified parts are analyzed which allows avoiding the verification of the whole system after each reconfiguration.

3.9 Trade-off between Expressiveness and Calculability in PN-based Reconfigurable Formalisms

Although enriching Petri net by reconfigurability increases their modeling power, it decreases their decision power. Hence, the proposed extensions in literature dealing with reconfigurability in PNs try to find a trade-off between modeling and decision power. From this perspective, we can distinguish two main directions. On the one hand, researchers develop preprocessing techniques that encode or unfold graphs and transformation rules into existing formalisms in order to take advantage of off-the-shelf tools [CC18, LO04a, Zha+13]. Although they can model reconfigurations in a natural way, actually these approaches do not increase the modeling power, since they depend on the expressiveness of underlying formalisms and especially do not allow structurally unbounded graphs [RSV04]. For instance, classical model checkers use a fixed number of propositions, which impedes modeling infinite-structure systems [Ren08].

On the other hand, certain techniques execute graph rewriting systems and compute the state-space for model checking, yet an upper artificial bound is still needed [KR06]. To mitigate this issue, some approaches compute either (under) approximations of system behaviors, so that any property that holds in under-approximated model is satisfied in its original system, or (over) approximations which include all system behaviors, and eventually additional behaviors not belonging to the original system [CR12]. Nevertheless, a property that does not hold in an under-approximated model may hold in its original system and a property that holds in an over-approximated model may do not hold in its original system [BCK08].

Finally, in another direction INRSs enforce particular properties such as liveness, boundedness and reversibility after each reconfiguration. Hence, these properties are decidable whatever the number of obtained configurations (even infinite). However, INRSs are re-

stricted to (i) ordinary, live, bounded and reversible PNs and (ii) restricted forms of reconfigurations, which decreases severely their expressiveness.

A recap of this discussion is illustrated in Fig. 3.9.

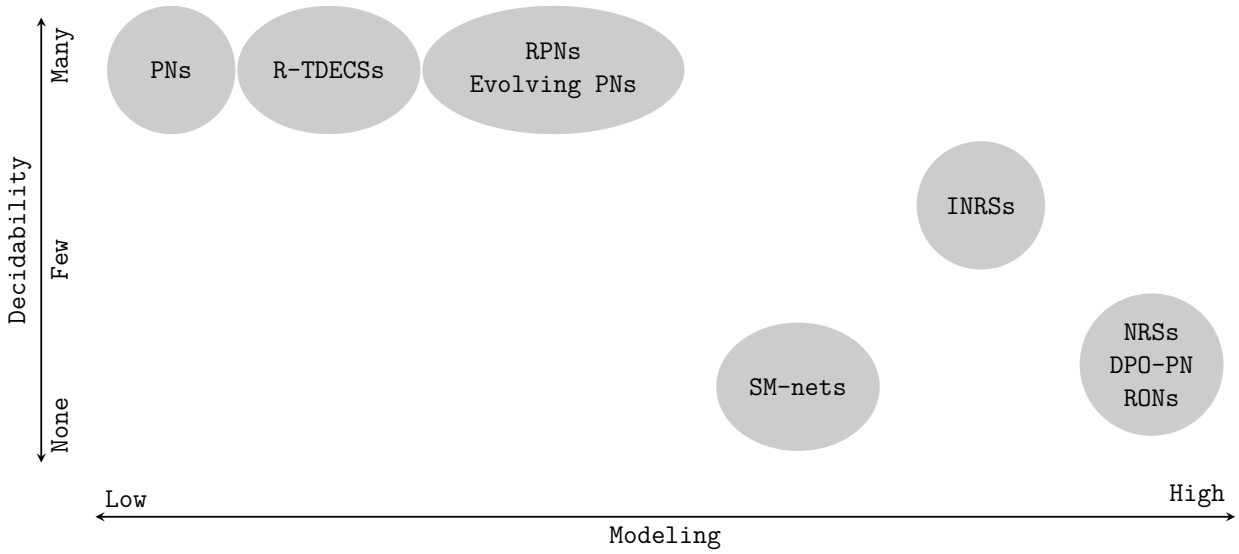


Figure 3.9: Compromise between modeling and decidability.

3.10 Conclusion

In this chapter, we have presented some formal methods proposed to enrich Petri nets with reconfigurability. These methods can be divided into three categories:

1. The first category considers only the reconfigurability modeling by allowing any reconfiguration form, such as DPO and NRS, in the expensive of decidability issues. In this category, the properties are decidable if the reachability graph is bounded. Yet, the spatial and temporal complexities are a very challenging issue due to behavioral method (based on reachability graph) complexity combined with graph transformation complexity,

2. The second category consists of formalisms that have the same modeling power of Petri nets, such as RPN and evolving Petri nets. They are used in the modeling of systems that can only have dynamic topologies without modifying the sets of places and transitions. This restriction intends to allow encoding or transforming graphs and transformation rules into existing formalisms in order to preserve decision power, as well as, to take advantages of off-the-shelf tools,

3. Finally, the last category tries to find a trade-off between modeling and decision power, such as INRSs. In this category, the reconfiguration forms are restricted in order to

preserve the original properties of a reconfigurable net. Hence, two advantages arise: (i) it is not required to verify these properties after each reconfiguration which decreases largely the verification complexity, and (ii) these properties are decidable even if the number of obtained configurations is infinite.

In this thesis, we are interested in the second and the third categories. In fact, we propose the following.

1. In **Chapters 4** and **5**, we propose two formalisms, called GSPNs with rewritable topology (GSPNs-RT) and dynamic GSPNs (D-GSPNs) respectively, that extend GSPNs by allowing modeling reconfiguration forms and transforming GSPNs-RT and D-GSPNs, respectively, to equivalent GSPNs. These transformations can take place when the original models dispose of a finite number of configurations, in order to preserve the decision power of GSPNs, as well as, to take full advantages of off-the-shelf tools proposed to GSPNs verification (e.g., **GreatSPN** [Baa+09], **PIPE** [DKS09]),

2. In **Chapter 6**, we extend GSPNs to a reconfigurable formalism, called reconfigurable GSPNs (RecGSPNs), that supports a wider range of possible structural changes than allowed in INRSs, as well as, allows to any reconfigurable GSPN to dispose of an infinite number of configurations while preserving the decidability of several important properties.

Part II

Contributions

Chapter 4

GSPNs with Rewritable Topology: A Trivial Approach

4.1 Introduction

New developed software/hardware systems are open, dynamic, and flexible (as the case of reconfigurable wireless sensor networks, data centers, etc.). Often, the reliability is indispensable in these kinds of systems. Formal methods (as GSPNs) are used to formally verify the reliability in critical systems. Nevertheless, the rigid structure of GSPNs restricts their use in the study of reconfigurable systems.

In this chapter, we present our earlier work, called GSPNs with rewritable topology (GSPNs-RT), that extends the formalism provided in [LO04b] (c.f. Section 3.6) to consider reconfigurability in GSPNs and to offer a suitable tool for the formal modeling and verification of reconfigurable systems. The new GSPNs-RT formalism combines GSPNs formalism, a set of transformation rules, and an algorithm used to transform a given GSPN-RT into an equivalent GSPN. Hence, we can apply the classical verification methods proposed for GSPNs to the obtained equivalent net in the verification phase.

The rest of this chapter is organized as follows. In Section 4.2, we provide the formal definition of the proposed formalism and we present an algorithm that transforms GSPNs-RT to GSPNs. In Section 4.3, we prove the equivalence between a given GSPN-RT and its equivalent GSPN obtained by the transformation. In Section 4.4, the feasibility of the new proposed formalism is demonstrated through an illustrative example. Finally, we conclude the chapter by a conclusion.

4.2 GSPNs with rewritable topology

Consider the following straightforward application of reconfigurable Petri nets (RPNs) (cf. Section 3.6) to GSPNs. Fig. 4.1a shows an example of RPN N to be reconfigured, and Fig. 4.1b depicts the obtained configuration after applying reconfiguration. RPN N is defined as follows. $N = \langle P, T, \mathcal{R}, \gamma_0 \rangle$, where:

1. $P = \{p_0, p_1, p_2\}$, $T = \{t_0, t_1, t_2, t_3, t_4\}$, $\gamma_0 = \Gamma$ highlighted in Fig 4.1a,
2. $\mathcal{R} = \{r = \langle D, {}^\bullet r, r^\bullet, \mathbb{C}, \mathbb{M} \rangle\}$, such that: $D = \{p_1, p_2\}$, $\mathbb{C} = \{p_0\}$, $\mathbb{M}(p_0) \geq 3$, ${}^\bullet r(p_1, t_3) =^\bullet r(t_3, p_2) = 1$, and $r(p_1, t_3)^\bullet = r(t_3, p_2)^\bullet = 0$.

This reconfiguration intends to neutralize transition t_3 in the second configuration, since the firing of t_3 does not update the net marking.

Although transition t_3 is neutralized, its presence affects the quantitative properties of the obtained net. Consider GSPNs Γ_1 and Γ_1' depicted in Fig. 4.2, they do not have the same quantitative properties. Actually, Γ_1 contains a "timeless trap" [Mar+94], since immediate transition t_3 is consistently enabled (therefore all timed transitions in Γ_1 are consistently disabled).

(a) Before reconfiguration (b) After reconfiguration

Figure 4.1: Reconfiguration of RPN N.

(a) Γ_1 (b) Γ_1'

Figure 4.2: Counterexample 1.

Furthermore, let us consider two GSPNs Γ_2 and Γ_2' shown in Fig. 4.3, they do not have the same quantitative properties, as well. For example, the probability that t_0 fires first at $M_0(p_0, p_1) = (0, 0)$ equals $\lambda_0/(\lambda_0 + \lambda_1)$ in Γ_2, and equals $\lambda_0/\lambda_0 = 1$ in Γ_2', where λ_0 and λ_1 are the firing delays of t_0 and t_1, respectively.

Hence, these two situations must be considered in the reconfiguration of GSPNs.

(a) Γ_2 (b) Γ_2'

Figure 4.3: Counterexample 2.

4.2.1 Formal definition

We define a GSPN-RT as a tuple $N = \langle P, T, \mathcal{R}, \gamma_0 \rangle$, where

1. $P = \{p_0, \ldots, p_n\}$ is a non-empty and finite set of places,
2. $T = \{t_0, \ldots, t_m\}$ is a non-empty and finite set of transitions, $P \cap T = \emptyset$,
3. $\mathcal{R} = \{r_0, \ldots, r_k\}$ is a finite set of rewriting rules,
4. γ_0 is an initial configuration of N modeled by a GSPN.

Rule $r \in \mathcal{R}$ is a tuple $r = \langle D, {}^\bullet r, r^\bullet, \mathbb{C}, \mathbb{M}, V \rangle$, such that

1. $D \subseteq P$,
2. ${}^\bullet r : (D \times T) \cup (T \times D) \to \mathbb{N}$ is a flow function of places in D at N before applying r,
3. $r^\bullet : (D \times T) \cup (T \times D) \to \mathbb{N}$ is a flow function of places in D at N after applying r,
4. $\mathbb{C} \subseteq D$ is a set of control places,
5. $\mathbb{M}$ is a required minimum marking of places of $\mathbb{C}$ so that rule r can be applied,
6. V is the firing weight of rule r.

Actually, GSPNs formalism is equipped with a set of algorithms and methods used to verify its qualitative/quantitative properties based on either its structure (i.e., structural analysis) or its reachability marking graph (i.e., behavioral analysis). Motivated by these advantages, we propose to unfold GSPNs-RT to equivalent GSPNs in order to analyze them using the well-known verification methods proposed for GSPNs. In the proposed method, we adapt the unfolding algorithm proposed in RPNs [LO04b] to deal with GSPNs-RT.

In fact, GSPN-RT N can be reconfigured by adding and/or removing arcs. These transformations take place in order to enable, disable and/or modify conditions/effects (resp. pre/postconditions) of some actions (resp. transitions) in the system. Hence, equivalent G_e must be able to model these transformations and their effects. Moreover, the switching between configurations must be modeled. In this scope, we reuse the algorithm proposed in RPNs and adapt it to consider the cases discussed above in Fig. 4.2 and Fig. 4.3. The steps of the algorithm are given as follows.

Let $\mathcal{G} = \{C_0, C_1, \ldots, C_n\}$ be the configurations set of GSPN-RT N (to be transformed), such that $C_0 = \gamma_0$ is an initial configuration and $C_1, \ldots, C_n$ are configurations obtained by applying reconfiguration rules to C_0.

Let $\mathring{P} = \{\mathring{p_0}, \mathring{p_1}, \ldots, \mathring{p_n}\}$ be a set of places corresponding to $C_0, C_1, \ldots, C_n$, respectively.

Let $\mathring{M}^0 = (1, 0, \ldots, 0)$ be an initial making of $\mathring{p}_0, \mathring{p}_1, \ldots, \mathring{p}_n$, respectively.

Let $G_e = (P_e, T_e, F_e, M_{e_0}, T_{e_1}, T_{e_2}, \Lambda_e)$ be an equivalent GSPN of GSPN-RT N.

We consider the following notations:

- X_Y: denotes the concatenation of two vectors X and Y.
- $M_e = M\_\mathring{M}^i$: denotes a reachable marking of G_e, where M is the marking of places in P, $\mathring{M}$ is the marking of places in $\mathring{P}$ corresponding to configurations in $\mathcal{G}$, and $M_e(\mathring{p}_0) = 1$.
- $\dot{t}^i$: denotes the representation of transition $t \in T$ in equivalent GSPN G_e with the same flow relations in C_i.

- $F_{C_i}(n, \cdot)$: denotes the post-conditions of node n in C_i.
- $F_{C_i}(\cdot, n)$: denotes the pre-conditions of n in C_i.
- $F_e(P, t)$: denotes the sub-pre-conditions of t with nodes in P in equivalent GSPN G_e.
- $F_e(t, P)$: denotes the sub-post-conditions of t with nodes in P in equivalent GSPN G_e.
- ${}^\bullet t_C$: denotes the pre-set of t in configuration C.
- $t^\bullet_C$: denotes the post-set of t in configuration C.
- $[M_0\rangle^{0} t_0[M_1\rangle^{0} t_1 \ldots [M_x\rangle r_0^i[M_y\rangle^{\mathrm{i}} t_q \ldots [M_z\rangle^{\mathrm{j}} t_p$ denotes a fireable sequence of transitions and rules in a GSPN-RT N, where $[M_0\rangle^{\mathrm{i}} t_0[M_1\rangle^{\mathrm{i}} t_1 \ldots [M_x$ denotes a fireable sequence of transitions in configuration C_i, and $[M_x\rangle r_k^j[M_j$ denotes an applicable rule r from configuration C_k at marking M_x to configuration C_j.

To create equivalent GSPN G_e, firstly we clone all places in P with their initial marking in set of places P_e of GSPN G_e. Secondly, places in $\mathring{P}$ corresponding to configurations are appended to P_e. We note that a token in $\mathring{p}_i$ means that the current reconfiguration of the system is C_i, such that $\Sigma_{i=0}^{n} M_e(\mathring{p}_i) = 1$, since the system can not be in two configurations or more, simultaneously.

Considering the unfolding of transitions, two cases are considered. First, for all couple "transition t and configuration C_i", if $({}^\bullet t_{C_i} \neq \emptyset) \vee (t^\bullet_{C_i} \neq \emptyset)$ (i.e., t is not neutralized), then we add transition $\dot{t}^i$ representing t to set T_e, as well as, preserving its flow relations with $p \in P$ in C_i. It is indispensable to connect $\dot{t}^i$ to $\mathring{p}_i$ with a self-loop in order to make $\dot{t}^i$ enabled only if the current reconfiguration is C_i. The purpose of duplicating t is to model the change in its pre/postconditions via reconfiguration. Second, if ${}^\bullet t_{C_i} = t^\bullet_{C_i} = \emptyset$ (i.e., t is neutralized at C_i), then we omit t which means that there is no representation of $t \in T_{C_i}$ in G_e.

Finally, each reconfiguration rule $r = \langle D, {}^\bullet r, r^\bullet, \mathbb{C}, \mathbb{M}, V\rangle$ in GSPN-RT $N = \langle P, T, \mathcal{R}, \gamma_0\rangle$ is represented by transition r_i^j, where C_i and C_j are the source and target configurations, respectively. In GSPN-RT N, reconfiguration rule r can be applied when the flow relations in the current configuration C_i of places in D is identical to ${}^\bullet r$ and $M(p) \geq \mathbb{M}(p), \forall p \in \mathbb{C}$ (here M is the current marking of C_i). Accordingly, the preconditions of transition r_i^j in $G_e = (P_e, T_e, F_e, M_{e_0}, T_{e_1}, T_{e_2}, \Lambda_e)$ are: (i) $M_e(p) \geq \mathbb{M}(p), \forall p \in \mathbb{C}$ and (ii) $M_e(\mathring{p}_i) = 1$, where M_e is the current marking of G_e. Hence, the unfolding of rule r requires the following

- Add a self-loop between transition r_i^j and each place $p \in \mathbb{C}$, such that $F_e(p, r_i^j) = F_e(r_i^j, p) = \mathbb{M}(p)$.
- Add an arc from $\mathring{p}_i$ to r_i^j and an arc from r_i^j to $\mathring{p}_j$ (i.e., $F_e(\mathring{p}_i, r_i^j) = F_e(r_i^j, \mathring{p}_j) = 1$) to model the switching from configuration C_i to configuration C_j.

4.3 Proofs

The aim of this section is to prove the equivalence between GSPN-RT N and GSPN G_e obtained by the application of the algorithm described above. To prove the equivalence, we consider the following lemmas and theorem.

Lemma 1. *If the current configuration is C_i (i.e., $M_e(\mathring{p}_i) = 1$) and transition t in C_i is fireable at M (in net N), then $\dot{t}^i$ (the representation of t at configuration C_i in G_e) is fireable at $M\_\mathring{M}^i$ in equivalent net G_e.*

Lemma 2. *If the current configuration is C_i (i.e., $M_e(\mathring{p}_i) = 1$) and reconfiguration rule r is applicable from C_i at M (in net N), then transition t^i is fireable at $M\_\mathring{M}^i$, where t^i is the corresponding transition to r (in equivalent net G_e).*

Proof. First, we prove Lemma 1. Let T^i be a set of transitions $t^i_j \in T_e$ which are the unfolding of transitions $t_j \in T_{C_i}$. If the current configuration is C_i (i.e., $M_e(\mathring{p}_i) = 1$), then t is disabled if $t \notin T^i$, since it is connected by a self-loop with $\mathring{p}_{j\neq i}$ which its current marking $M_e(\mathring{p}_j) = 0$. Also, if $\dot{t}^i$ is the representation of $t \in T_{C_i}$, then we have by definition $(F_e(\dot{t}^i, P) = F_{C_i}(t, \cdot)) \wedge (F_e(P, \dot{t}^i) = F_{C_i}(\cdot, t)) \wedge (F_e(\mathring{p}_{j,j\neq i}, \dot{t}^i) = 0) \wedge (F_e(\mathring{p}_i, \dot{t}^i) = 1)$, which means that if t in C_i is fireable at M, then $\dot{t}^i$ is fireable at $M\_\mathring{M}^i$.

The proof of Lemma 2 is given as follows. If transition $t \in T_e$ is not a representation of any transition in C_i, does not model any reconfiguration rule from C_i, and the current configuration is C_i, then t is disabled since it is connected by self-loop with $\mathring{p}_{j,j\neq i}$ which its current marking is $M_e(\mathring{p}_j) = 0$. Also, if transition $r^*_i \in T_e$ models reconfiguration rule r from configuration C_i at marking $\mathbb{M}$, then we have by definition the preconditions of r^*_i are $M_e(p) \geq \mathbb{M}(p), \forall p \in \mathbb{C}$ and $M_e(\mathring{p}_i) = 1$, which means that if r is applicable from C_i at $\mathbb{M}$, then r^*_i is fireable at $M\_\mathring{M}^i$, where $M(p) \geq \mathbb{M}(p), \forall p \in \mathbb{C}$. □

***Theorem* 4.1: Equivalence**

Any fireable sequence of transitions (which are not neutralized) and rules in GSPN-RT N is a fireable sequence in equivalent G_e obtained by the unfolding algorithm, and vice versa.

Proof. From the previous proof, it is obvious to deduct that if $[M_0\overset{0}{\rangle}t_0[M_1\overset{0}{\rangle}t_1 \ldots [M_x\rangle r^i_0[M_y\overset{i}{\rangle}t_q \ldots [M_z\overset{j}{\rangle}t_p$ is a fireable sequence in GSPN-RT N, then $[M_0\_\mathring{M}^0\rangle t^0_0[M_1\_\mathring{M}^0\rangle t^0_1 \ldots M_x\_\mathring{M}^0\rangle r^i_0$ $[M_y\_\mathring{M}^i\rangle t^i_q \ldots [M_z\_\mathring{M}^j\rangle t^j_p$ is a fireable sequence in G_e, where t^i_j is the representation of transition $t_j \in T_{C_i}$ in G_e, and vice-versa. □

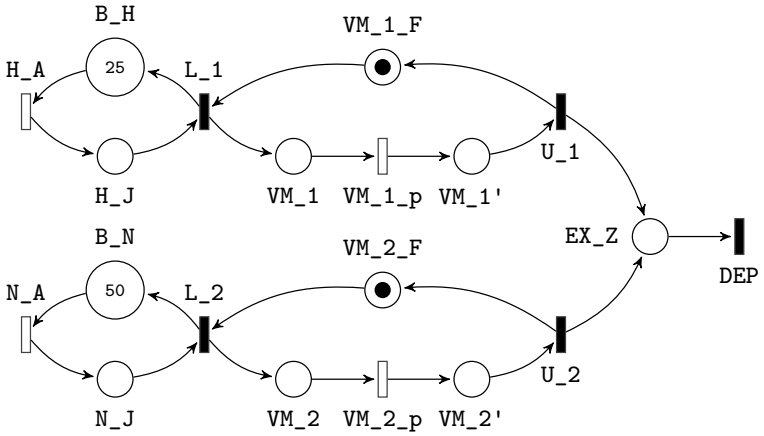


Figure 4.4: First configuration.

4.4 Illustrative Example

In this section, we demonstrate the application of the proposed formalism on a server in a data center. Firstly, we describe the structure and behavior of the system, then we apply the proposed transformation algorithm to obtain the equivalent GSPN and evaluate its performance. Finally, we conclude by a discussion about the obtained results.

Let us consider a server composed of two virtual machines $\texttt{VM}_1$, $\texttt{VM}_2$, a buffer $\texttt{buf}_\texttt{h}$ with capacity of 25 spaces for jobs with high priority, a buffer $\texttt{buf}_\texttt{n}$ with a capacity of 50 spaces for jobs with normal priority, and an exit zone EXZ in which the results are deposited to be send to the corresponding clients. Virtual machine $\texttt{VM}_1$ (resp. $\texttt{VM}_2$) loads a job from buffer $\texttt{buf}_\texttt{h}$ (resp. from $\texttt{buf}_\texttt{n}$) once it is idle, it processes the job, and it puts the results into the exit zone EXZ.

The server has two configurations: C_0 and C_1. In both configurations C_0 and C_1, virtual machine $\texttt{VM}_1$ processes high-priority jobs. As for virtual machine $\texttt{VM}_2$, it processes normal-priority jobs if the current reconfiguration is C_0. If the number of high-priority waiting jobs exceeds 15, the system switches to configuration C_1, such that the receiving of normal-priority jobs is stopped, virtual machine $\texttt{VM}_2$ stops treating normal-priority jobs and it starts processing high-priority jobs. At first, the system's configuration is C_0 depicted in Fig. 4.4.

The interpretation of places and transitions of C_0 are given as follows.

1. Places:

   (a) B_H (resp. B_N): The number of tokens, inside this place, models the number of available spaces in buffer $\texttt{buf}_\texttt{h}$ (resp. $\texttt{buf}_\texttt{n}$).

   (b) H_J (resp. N_J): The number of tokens in H_J (resp.N_J) models the number of high-priority (resp. normal-priority) waiting jobs.

(c) VM_1 (resp. VM_1'): A token in VM_1 (resp. VM_1') means that virtual machine $\texttt{VM}_1$ has begun (resp. finished) processing a job.

(d) VM_2 (resp. VM_2'): A token in VM_2 (resp. VM_2') means that virtual machine $\texttt{VM}_2$ has begun (resp. finished) processing a job.

(e) VM_1_F (resp. VM_2_F): A token in VM_1_F (resp. VM_2_F) means that $\texttt{VM}_1$ (resp. $\texttt{VM}_2$) is idle.

(f) EX_Z: The number of tokens in EX_Z models the number of waiting finished jobs in exit zone EXZ.

2. Transitions:

(a) H_A (rate=λ_{ha}): A high-priority job is loaded in buffer $\texttt{buf}_\texttt{h}$.

(b) N_A (rate=λ_{na}): A normal-priority job is loaded in buffer $\texttt{buf}_\texttt{n}$.

(c) L_1 (resp. U_1): Virtual machine $\texttt{VM}_1$ loads (resp. unloads) a job from buffer $\texttt{buf}_\texttt{h}$ (resp. to exit zone EXZ).

(d) L_2 (resp. U_2): Virtual machine $\texttt{VM}_2$ loads (resp. unloads) a job from buffer $\texttt{buf}_\texttt{n}$ (resp. to exit zone EXZ).

(e) VM_1_p (rate=λ_{vm1p}) (resp. VM_2_p (rate=λ_{vm2p})): Virtual machine $\texttt{VM}_1$ (resp. $\texttt{VM}_2$) is processing a job.

(f) DEP (rate=λ_{dep}): A finished job leaves the exit zone.

When the number of high-priority waiting jobs exceeds 15 (i.e., $M(\texttt{H\_J}) \geq 15$), the server switches to configuration C_1 depicted in Fig. 4.5, thus the receiving of normal-priority jobs is stopped and virtual machine $\texttt{VM}_2$ starts processing high-priority jobs. C_1 is obtained by applying rule $r_0 = \langle D_0, {}^\bullet r_0, r_0^\bullet, \mathbb{C}_0, \mathbb{M}_0, V_0 \rangle$ to C_0, where:

1. $D_0 = \{\texttt{B\_H}, \texttt{B\_N}, \texttt{H\_J}, \texttt{N\_J}\}$.

2. ${}^\bullet r_0$: $\texttt{B\_H}(\texttt{L\_1} - \texttt{H\_A}) + \texttt{H\_J}(\texttt{H\_A} - \texttt{L\_1}) + \texttt{B\_N}(\texttt{L\_2} - \texttt{N\_A}) + \texttt{N\_J}(\texttt{N\_A} - \texttt{L\_2})$.

3. $r_0^\bullet$: $\texttt{B\_H}(\texttt{L\_1} + \texttt{L\_2} - \texttt{H\_A}) + \texttt{H\_J}(\texttt{H\_A} - \texttt{L\_1} - \texttt{L\_2}) + \texttt{B\_N}(\emptyset) + \texttt{N\_J}(\emptyset)$.

4. $\mathbb{M}_0(\texttt{H\_J}) = 15$.

5. V_0 is a firing weight of r_0.

Once the number of high-priority waiting jobs is less than five, the processing and the receiving of normal-priority jobs can be restarted. The configuration C_1 is reconfigured to C_0 by applying rule $r_1 = \langle D_1, {}^\bullet r_1, r_1^\bullet, \mathbb{C}_1, \mathbb{M}_1, V_1 \rangle$, where:

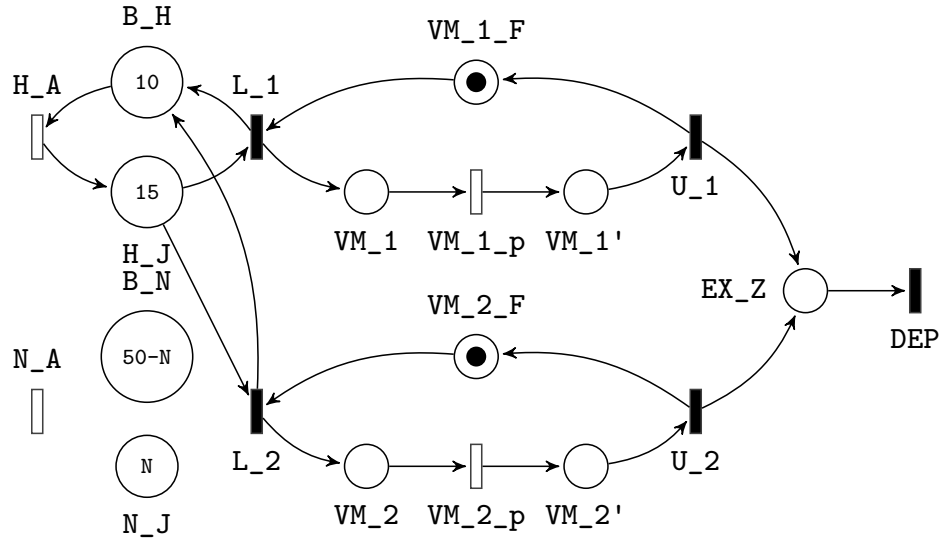


Figure 4.5: GSPN model for configuration C_1.

1. $D_1 = \{\texttt{B\_H}, \texttt{B\_N}, \texttt{H\_J}, \texttt{N\_J}\}$.

2. ${}^\bullet r_1 : \texttt{B\_H}(\texttt{L\_1} + \texttt{L\_2} - \texttt{H\_A}) + \texttt{H\_J}(\texttt{H\_A} - \texttt{L\_1} - \texttt{L\_2}) + \texttt{B\_N}(\emptyset) + \texttt{N\_J}(\emptyset)$.

3. $r_1^\bullet : \texttt{B\_H}(\texttt{L\_1} - \texttt{H\_A}) + \texttt{H\_J}(\texttt{H\_A} - \texttt{L\_1}) + \texttt{B\_N}(\texttt{L\_2} - \texttt{N\_A}) + \texttt{N\_J}(\texttt{N\_A} - \texttt{L\_2})$.

4. $\mathbb{C}_1 = \{\texttt{B\_H}\}$.

5. $\mathbb{M}_1(\texttt{B\_H}) = 20$.

6. V_1 is a firing weight of r_1.

Now, we consider the verification aspect. To analyze the system properties, first we create equivalent GSPN G_e of GSPN-RT $N = \langle P, T, \mathcal{R}, C_0 \rangle$ described above, where:

1. $P = \{\texttt{B\_H}, \texttt{B\_N}, \texttt{H\_J}, \texttt{N\_J}, \texttt{VM\_1}, \texttt{VM\_1'}, \texttt{VM\_1\_F}, \texttt{VM\_2}, \texttt{VM\_2'}, \texttt{VM\_2\_F}, \texttt{EX\_Z}\}$.

2. $T = \{\texttt{H\_A}, \texttt{N\_A}, \texttt{L\_1}, \texttt{VM\_1\_p}, \texttt{U\_1}, \texttt{L\_2}, \texttt{VM\_2\_p}, \texttt{U\_2}, \texttt{DEP}\}$.

3. C_0 is an initial configuration shown in Fig. 4.4.

4. $\mathcal{R} = \{r_0, r_1\}$ is a set of reconfiguration rules given above.

Applying rules in $\mathcal{R}$ to N yields $\mathcal{G} = \{C_0, C_1\}$ the finite configurations set, where C_0 (resp. C_1) is depicted in Fig. 4.4 (resp. Fig. 4.5).

Equivalent $G_e = (P_e, T_e, F_e, M_{e0}, T_{e_1}, T_{e_2}, \Lambda_e)$ can be constructed using the algorithm described previously. To obtain $P_e = P \cup \mathring{P}$, we create $\mathring{P} = \{\mathring{p}_0, \mathring{p}_1\}$ where $\mathring{p}_0$ and $\mathring{p}_1$ are corresponding to C_0 and C_1, respectively. The initial marking of places in $\mathring{P}$ is $M_{e0}(\mathring{p}_0, \mathring{p}_1) = (1, 0)$.

As for transitions, we give two examples of N_A and L_2, the other transitions can be added and connected by the same way. Transition N_A is connected by no arc in C_1 which means that transition N_A is neutralized at this configuration. Therefore, transition N_A has a unique representation in G_e ($T_e \longleftarrow T_e \cup \{\texttt{N\_A}^0\}$) and the connections between N_A and $p \in P$ is the same as in C_0, so $F_e(\texttt{N\_A}^0, \texttt{N\_J}) = F_e(\texttt{B\_N}, \texttt{N\_A}^0) = 1$, otherwise is 0. As for connections between N_A and $p \in \mathring{P}$, we have $F_e(\texttt{N\_A}^0, \mathring{p}_0) = F_e(\mathring{p}_0, \texttt{N\_A}^0) = 1$ and $F_e(\texttt{N\_A}^0, \mathring{p}_1) = F_e(\mathring{p}_1, \texttt{N\_A}^0) = 0$ which deactivates transition $\texttt{N\_A}^0$ when there is no token marking place $\mathring{p}_0$, indicating that the current configuration is C_1.

Taking into account transition L_2, it has no empty pre/postconditions in C_0 and C_1. Accordingly, transition L_2 has two representations in G_e as follows: (i) $\texttt{L\_2}^0$ ($T_e \longleftarrow T_e \cup \{\texttt{L\_2}^0\}$) which its flow relations are $F_e(\texttt{L\_2}^0, \texttt{B\_N}) = F_e(\texttt{L\_2}^0, \texttt{VM\_2}) = F_e(\texttt{N\_J}, \texttt{L\_2}^0) = F_e(\texttt{VM\_2\_F}, \texttt{L\_2}^0) = F_e(\texttt{L\_2}^0, \mathring{p}_0) = F_e(\mathring{p}_0, \texttt{L\_2}^0) = 1$, otherwise is 0, and (ii) $\texttt{L\_2}^1$ ($T_e \longleftarrow T_e \cup \{\texttt{L\_2}^1\}$) which its flow relations are $F_e(\texttt{L\_2}^1, \texttt{B\_H}) = F_e(\texttt{L\_2}^1, \texttt{VM\_2}) = F_e(\texttt{H\_J}, \texttt{L\_2}^1) = F_e(\texttt{VM\_2\_F}, \texttt{L\_2}^1) = F_e(\texttt{L\_2}^1, \mathring{p}_1) = F_e(\mathring{p}_1, \texttt{L\_2}^1) = 1$, otherwise is 0.

Finally, we add two transitions r_0^1 and r_1^0 to model the reconfiguration from C_0 to C_1 and from C_1 to C_0, respectively. As an example, we consider r_0^1 which models the switching to C_1 when the number of high-priority waiting jobs exceeds 15 and the current configuration is C_0. Therefore, $F_e(r_0^1, \texttt{H\_J}) = F_e(\texttt{H\_J}, r_0^1) = 15$ and $F_e(\mathring{p}_0, r_0^1) = F_e(r_0^1, \mathring{p}_1) = 1$.

Obtained equivalent GSPN G_e is illustrated in Fig. 4.6.

To verify the qualitative/quantitative properties, we used tool **PIPE2** [DKS09]. G_e is live, bounded and reversible, other qualitative properties can be investigated using either behavioral or structural analysis. Given the different rates illustrated in Table 4.1, the simulation of the server is performed. The obtained results are shown in Figs. 4.7 and 4.8.

Fig. 4.7 depicts the probability that the current configuration of the server is C_0 or C_1 under different cases. In the second case, the probability that the current configuration is C_1, is the highest among other cases. The high rate of receiving high-priority jobs in Case 2 increases the probability that buffer $\texttt{buf}_\texttt{h}$ contains more than 15 jobs. Consequently, the server switches to configuration C_1 more frequently.

As shown in Fig. 4.8, virtual machine $\texttt{VM}_\texttt{2}$ utilization is higher than that of virtual machine $\texttt{VM}_\texttt{1}$ in Case 1. This is caused by (i) the high arrival rate of normal-priority jobs (which are treated only by $\texttt{VM}_\texttt{2}$) which increases virtual machine $\texttt{VM}_\texttt{2}$ utilization, and (ii) the low arrival rate of high-priority jobs which decreases virtual machine $\texttt{VM}_\texttt{1}$ utilization.

Cases	λ_{ha}	λ_{vm1p}	λ_{vm2p}	λ_{na}
Case (1)	3	3	3	3
Case (2)	7	3	3	3
Case (3)	5	4	3	1

Table 4.1: Entry of the different evaluations.

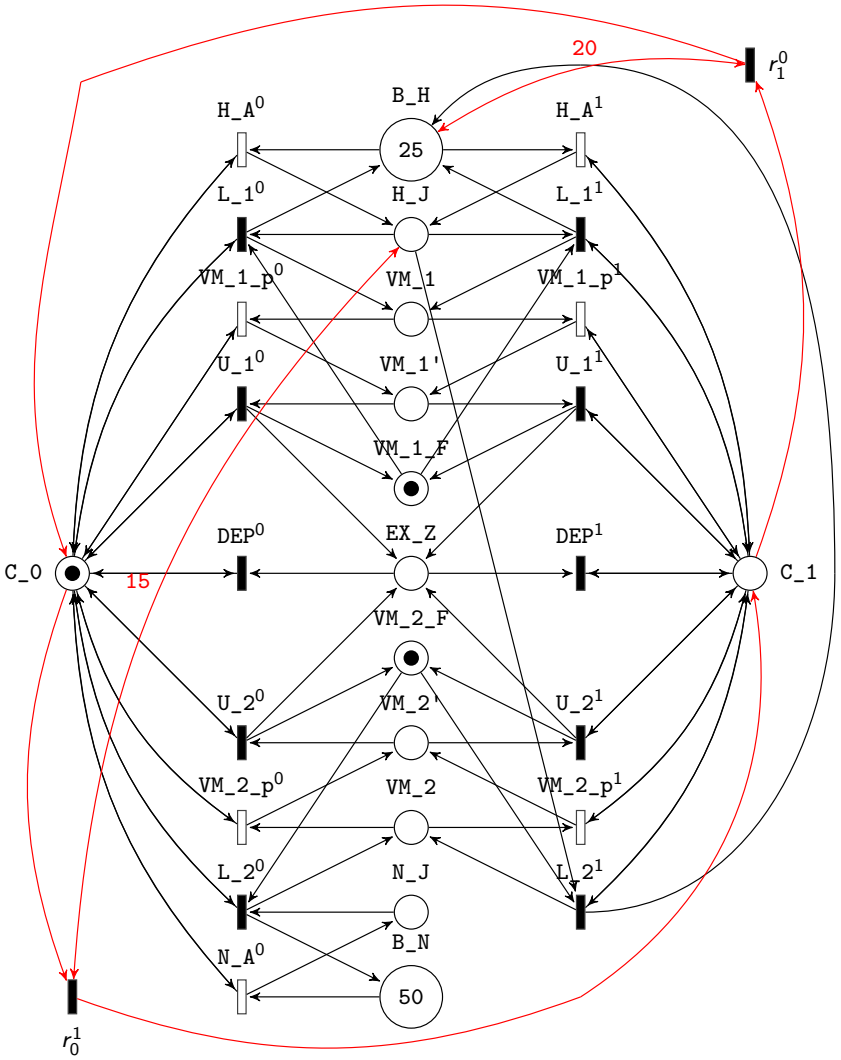


Figure 4.6: Equivalent GSPN G_e.

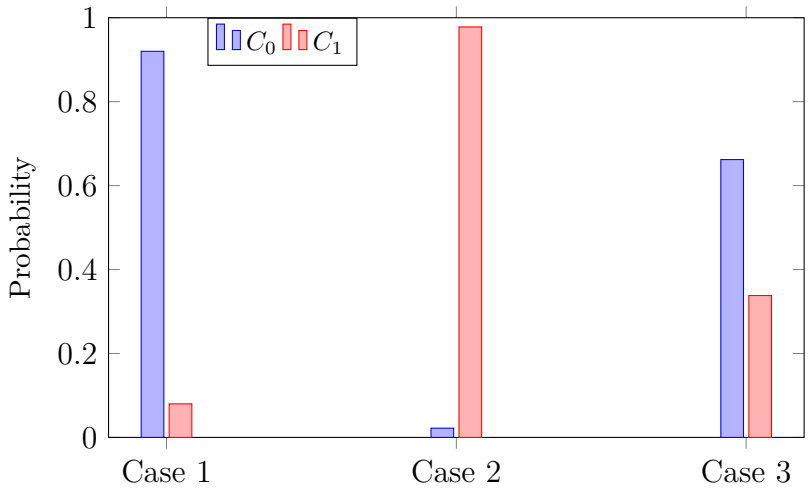


Figure 4.7: Probability that the current configuration is C_0 or C_1.

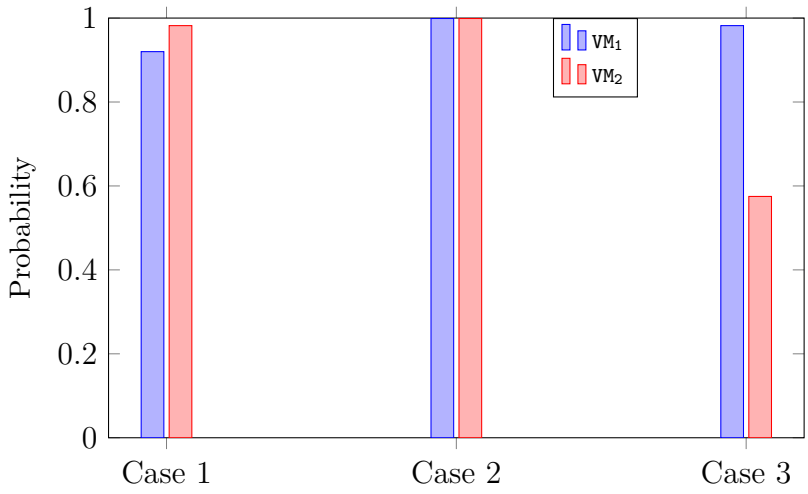


Figure 4.8: Probability that $\texttt{VM}_1/\texttt{VM}_2$ is working.

4.5 Conclusion

In this chapter, we have proposed an extension of GSPNs that allows the reconfiguration of GSPNs. The proposed approach extends reconfigurable Petri nets proposed in [LO04a] to deal with GSPNs, where qualitative and quantitative properties can be investigated.

The new GSPNs-RT formalism combines the GSPNs formalism, a set of transformation rules, and an algorithm used to transform a given GSPN-RT into an equivalent GSPN.

In the following chapter, instead of considering only dynamic topologies, we extend GSPNs-RT approach to consider more reconfiguration forms, namely, dynamic sets of places and transitions.

Chapter 5

Dynamic Generalized Stochastic Petri Nets

5.1 Introduction

Enriching PNs with reconfigurability at the modeling level requires developing new analysis methods. On the other hand, PNs already enjoy a rich panoply of well-established and optimized verification algorithms. To take full advantage of this characteristic, researchers have proposed extensions enriching PNs with reconfigurability that can be either encoded or transformed into basic PNs [LO04a, CC18, CBC18].

However, to the best of our knowledge, proposed formalisms in the literature allow only dynamic topology, i.e., sets of places and transitions cannot be changed. This last restriction was motivated by the need to allow verification of dynamic-structure nets through their encoding or transformation into basic Petri nets. However, this restriction limits severely the modeling power of these formalisms.

Our primary contributions in this line of research were reconfigurable SPNs [TKB17b] and GSPNs with rewritable topology [TKB17a] extending formalism presented in [LO04a] (c.f. Section 3.6) to cope with reconfigurability in SPNs and GSPNs, respectively. However, the reconfiguration is still limited to the topology level.

In this chapter, we propose a new formalism, called dynamic generalized stochastic Petri nets (D-GSPNs), that allows to model dynamic sets of places and transitions, as well as, keeping the possibility to transform D-GSPNs into GSPNs, for verification purposes. The obtained GSPNs preserve the stochastic behaviors of dynamic ones, allowing the use of the panoply of verification methods and tools proposed for GSPNs in D-GSPNs analysis.

In this regard, the reconfiguration in our proposed formalism is modeled via the well-known double-pushout (DPO) approach [Kön+18] which offers a theoretically founded and tool-supported framework for graph transformation.

As well, this chapter provides a new algorithm (inspired from [LO04a]), which transforms D-GSPNs into equivalent GSPNs. This algorithm computes the set of reachable configurations of a given D-GSPN, and if this set is finite, then transforms it into an equivalent GSPN preserving the stochastic behavior of the original D-GSPN. Therefore, qualitative/quantitative properties of D-GSPNs can be analyzed by applying analysis methods, supported by off-the-shelf tools [Amp14, BMS16], to their equivalent GSPNs.

The rest of this chapter is organized as follows. In Section 5.2, D-GSPNs formalism is proposed and its formal definition is exposed. The transformation of D-GSPNs into GSPNs is described in Section 5.3. Then, we present the qualitative/quantitative verifications, and we detail the proofs in Sections 5.4 and 5.5, respectively. Section 5.6 demonstrates the applicability of the proposed formalism to a reconfigurable system. Finally, the chapter is concluded in Section 5.7.

5.2 Dynamic GSPNs

In this section, first we present the formal definition of D-GSPNs formalism. Thereafter, we develop an algorithm that transforms D-GSPNs into GSPNs, then we describe its qualitative/quantitative verification. Finally, we prove the equivalence between D-GSPNs and their transformations into GSPNs.

5.2.1 Formal definition

To model reconfiguration rules in D-GSPNs via DPO approach, we extend the definition of PN morphism given in Section 3.3 to GSPN morphism as follows.

Definition 5.1. (GSPN morphism) A GSPN morphism from G to H is a pair of mappings $\varphi_P : P_G \to P_H, \varphi_T: T_G \to T_H$, such that $\forall\ p \in P_G$ and $\forall\ t \in T_G$, it holds that:

1. $F_G(p,t) = F_H(\varphi_P(p), \varphi_T(t))$ and $F_G(t,p) = F_H(\varphi_T(t), \varphi_P(p))$,
2. $M_G(p) \leq M_H(\varphi_P(p))$, and $\Lambda_G(t) = \Lambda_H(\varphi_T(t))$,
3. If t is timed, then so is $\varphi_T(t)$, and vice-versa.

Definition 5.2. (Dynamic GSPN) We define a D-GSPN as a pair $\mathcal{D} = \langle G_0, \mathcal{R} \rangle$, where:

- G_0 is a GSPN modeling an initial configuration,
- $\mathcal{R} = \{\omega_0, \ldots, \omega_m\}$ is a set of rules.

Definition 5.3. (Transformation rule) A transformation rule ω is written as a couple $\langle L \xleftarrow{\varphi_l} I \xrightarrow{\varphi_r} R, \lambda \rangle$, such that:

1. L is a left-hand side, I is a common interface, and R is a right-hand side,
2. $\varphi_l : I \to L$ is a GSPN morphism that maps I to L,
3. $\varphi_r : I \to R$ is a GSPN morphism that maps I to R,
4. If a place p belongs to L and R, then it must belong to I,
5. λ_i is an exponentially distributed application rate of rule ω.

Let G be a configuration of D-GSPN $\mathcal{D}$, a rule $\omega = \langle L \xleftarrow{\varphi_l} I \xrightarrow{\varphi_r} R, \lambda \rangle$ is applicable to G at marking M, iff:

1. there exists a matching of left-hand side L in G,
2. all immediate transitions in G are disabled at M.

As for rule application, we proceed as usual with respect to DPO approach.
Let us consider a D-GSPN $\mathcal{D}_0 = \langle H_0, \mathcal{R}_0 \rangle$, such that:

1. H_0 is depicted in Fig. 5.1, where its initial marking $M^0(p_0, p_1) = (0, 3)$,
2. $\mathcal{R}_0 = \{\omega_1, \omega_2\}$, such that ω_1 and ω_2 are shown in Figs. 5.1 and 5.2, respectively.

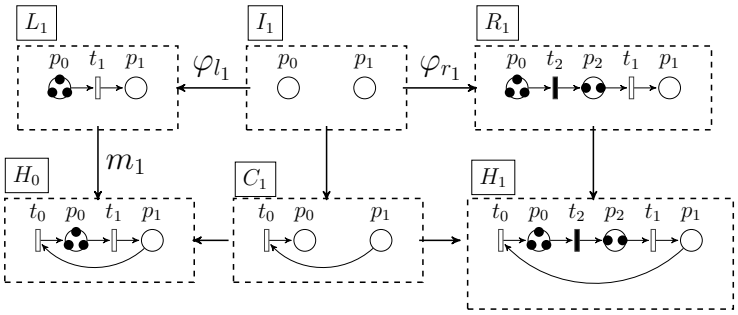


Figure 5.1: Rule ω_1.

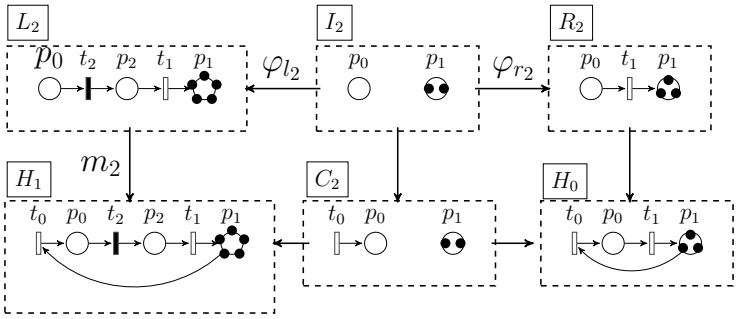


Figure 5.2: Rule ω_2.

Rule ω_1 is applicable to configuration H_0 of $\mathcal{D}_0$ at marking $M'(p_0, p_1) = (3, 0)$, since:

1. an occurrence of L_1 is located by morphism m_1 in H_0,
2. all immediate transitions are disabled at marking M'.

After applying ω_1, $\mathcal{D}_0$ changes its configuration towards GSPN H_1 depicted in Fig. 5.1. Similarly, rule ω_2, illustrated in Fig. 5.2, is applicable to H_1 shown in the bottom-left of Fig. 5.2 and the obtained configuration is H_0.

5.3 D-GSPNs transformation towards GSPNs

Generalized stochastic Petri nets are equipped with a panoply of algorithms, methods and tools used to analyze their qualitative/quantitative properties. To take full advantages of

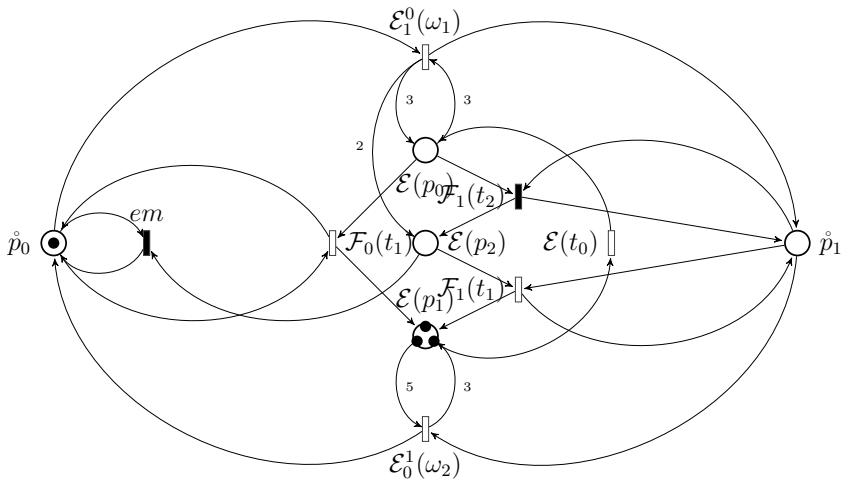


Figure 5.3: Equivalent GSPN $\mathcal{H}_0$ to D-GSPN $\mathcal{D}_0$.

this characteristic, we transform D-GSPNs into equivalent GSPNs, so that the off-the-shelf verification methods proposed for GSPNs can be still used for the D-GSPNs analysis.

Actually, a D-GSPN $\mathcal{D}$ is reconfigured by adding/removing nodes (places/transitions) and/or arcs. These transformations take place in order to introduce new behaviors/structures to $\mathcal{D}$. For this purpose, it is required that GSPN $\mathcal{H}$ obtained by transforming net $\mathcal{D}$ can emulate these transformations and preserves the stochastic behaviors of $\mathcal{D}$.

In this scope, we provide an algorithm that computes an equivalent GSPN $\mathcal{H}$ for any given D-GSPN $\mathcal{D}$ having a finite number of configurations. We start by providing the necessary explanations for the reader to understand how the transformation works. To make the steps more understandable, we apply each step to dynamic net $\mathcal{D}_0 = \langle H_0, \{\omega_1, \omega_2\}\rangle$, presented in Figs. 5.1 and 5.2. The equivalent GSPN $\mathcal{H}_0$ obtained via the transformation is shown in Fig. 5.3.

For a given configuration G_i, let us consider the following:

1. P_{G_i} denotes its set of places.
2. T_{G_i} denotes its set of transitions.
3. F_{G_i} denotes its flow function.
4. $M_{G_i}^0$ denotes its initial marking.
5. Λ_{G_i} denotes a function associating firing delay/weight to transitions of G_i.

For a given D-GSPN $\mathcal{D} = \langle G_0, \mathcal{R}\rangle$, let us consider the following.

1. G_0 is an initial configuration of $\mathcal{D}$.

2. $\mathcal{R} = \{\omega_0, \ldots, \omega_m\}$ is a finite set of rules.

3. $\mathcal{G} = \{G_0, \ldots, G_n\}$ is a finite set of GSPNs (the set of all possible configurations of $\mathcal{D}$) obtained by applying rules to G_0.

4. $\mathcal{A} = \{(G_i, \omega, G_j) | G_i, G_j \in \mathcal{G} \text{ and } G_i \overset{\omega}{\Rightarrow} G_j\}$ is a set of applicable transformations to $\mathcal{D}$, such that ω belongs to $\mathcal{R}$.

5. $\mathcal{P}$ denotes the set of all places of all configurations of $\mathcal{G}$, formally $\mathcal{P} = \bigcup_{G_i \in \mathcal{G}} P_{G_i}$.

6. $\mathcal{T}$ denotes the set of all transitions of all configurations of $\mathcal{G}$, formally $\mathcal{T} = \bigcup_{G_i \in \mathcal{G}} T_{G_i}$.

In order to preserve behaviors of a given D-GSPN $\mathcal{D}$, its transformation towards $\mathcal{H} = \langle P, T, F, M^0, T_1, T_2, \Lambda \rangle$ uses a set of morphisms $\{\mathcal{F}_0, \ldots, \mathcal{F}_n\}$, which map configurations of $\mathcal{D}$ into $\mathcal{H}$. A morphism $\mathcal{F}_i$: $P_{G_i} \cup T_{G_i} \to P \cup T$ maps places and transitions of configuration G_i into places and transitions of $\mathcal{H}$, such that following condition holds: for any pair of nodes $(x, y) \in (P_{G_i} \times T_{G_i}) \cup (T_{G_i} \times P_{G_i})$, $F_{G_i}(x, y) = F(\mathcal{F}_i(x), \mathcal{F}_i(y))$. Thus, each $\mathcal{F}_i$ preserves flow function F_{G_i} in $\mathcal{H}$.

To create an equivalent GSPN $\mathcal{H}$ to a D-GSPN $\mathcal{D}$, we proceed as follows.

***Step 1 (adding equivalent places)*:** for each place $p \in \mathcal{P}$, insert an equivalent place, denoted by $\mathcal{E}(p)$, in P (initially empty), such that if $p \in P_{G_0}$, then $M^0(\mathcal{E}(p)) = M^0_{G_0}(p)$; otherwise, $M^0(\mathcal{E}(p)) = 0$. Consider D-GSPN $\mathcal{D}_0$, we have: $\mathcal{P} = \{p_0, p_1, p_2\}$, set of equivalent places, denoted by $\mathcal{E}(\mathcal{P})$, becomes $\mathcal{E}(\mathcal{P}) = \{\mathcal{E}(p_0), \mathcal{E}(p_1), \mathcal{E}(p_2)\}$ and initial marking of equivalent places is $M^0(\mathcal{E}(p_0), \mathcal{E}(p_1), \mathcal{E}(p_2)) = (0, 3, 0)$.

***Step 2 (adding places to emulate configurations)*:** Create $\mathring{P}$ a set of places $\{\mathring{p}_0, \ldots, \mathring{p}_n\}$ associated with a set of configurations $\mathcal{G} = \{G_0, \ldots, G_n\}$. Hence, each place $\mathring{p}_i \in \mathring{P}$ is associated with configuration G_i, respectively. A token in $\mathring{p}_i$ means that current configuration of $\mathcal{D}$ is G_i, thus $\Sigma_{i=0}^{n} M(\mathring{p}_i) = 1$, since $\mathcal{D}$ cannot be in two configurations at same time. Considering again $\mathcal{D}_0$, it has two possible configurations H_0 and H_1 (illustrated in Fig. 5.1), which are associated with places $\mathring{p}_0$ and $\mathring{p}_1$, respectively. Accordingly, $\mathring{P} = \{\mathring{p}_0, \mathring{p}_1\}$, $M^0(\mathring{p}_0, \mathring{p}_1) = (1, 0)$ and $P = \{\mathcal{E}(p_0), \mathcal{E}(p_1), \mathcal{E}(p_2)\} \cup \mathring{P}$.

***Step 3 (adding equivalent transitions)*:** We consider two cases:

Case (1): For each transition t that does not change its parameters (preset, postset, rate and/or type) in any configuration, we insert an equivalent transition $\mathcal{E}(t)$ of t into T having identical rate/type. As well, we preserve its preset and postset, so that for all place p of G_0, if there exists an arc from t to p (from p to t), then we add an arc from $\mathcal{E}(t)$ to $\mathcal{E}(p)$ (from $\mathcal{E}(p)$ to $\mathcal{E}(t)$). Formally, $F(\mathcal{E}(t), \mathcal{E}(p)) = F_{G_0}(t, p)$ and $F(\mathcal{E}(p), \mathcal{E}(t)) = F_{G_0}(p, t)$, for all $p \in P_{G_0}$. Let $\mathcal{E}(t) = \mathcal{F}_i(t), \forall\ i \in \{0, \ldots, n\}$. Consider transition t_0 in (Fig. 5.1), it does not change its parameters either in H_0 or H_1, therefore we insert a transition $\mathcal{E}(t_0)$ in T, such that $F(\mathcal{E}(t_0), \mathcal{E}(p_0)) = F_{H_0}(t_0, p_0)$ and $F(\mathcal{E}(p_1), \mathcal{E}(t_0)) = F_{H_0}(p_1, t_0)$.

Case (2): In order to model the change in the structure of D-GSPN $\mathcal{D}$ with respect to transitions, we duplicate each transition as often as it appears in configurations of $\mathcal{D}$. This duplication of transitions means to model its various parameters (type, rates, presets and/or postsets).

For each transition t_j of each configuration G_i, we insert an equivalent transition $\mathcal{F}_i(t_j)$ of t_j into T having identical rate/type. As well, we preserve its preset and postset, so that for all place p of G_i, if there exists an arc from t_j to p (from p to t_j), then we add an arc from $\mathcal{F}_i(t_j)$ to $\mathcal{E}(p)$ (from $\mathcal{E}(p)$ to $\mathcal{F}_i(t_j)$). Formally, $F(\mathcal{F}_i(t_j), \mathcal{E}(p)) = F_{G_i}(t_j, p)$ and $F(\mathcal{E}(p), \mathcal{F}_i(t_j)) = F_{G_i}(p, t_j)$, for all $p \in P_{G_i}$. Finally, we connect $\mathcal{F}_i(t_j)$ with $\mathring{p}_i$ by a self-loop, thus $\mathcal{F}_i(t_j)$ is disabled if the current configuration is not G_i. Consider transition t_1 of H_0 (Fig. 5.1), we insert a transition $\mathcal{F}_0(t_1)$ in T, such that $F(\mathcal{F}_0(t_1), \mathcal{E}(p_1)) = F_{H_0}(t_1, p_1)$, $F(\mathcal{E}(p_0), \mathcal{F}_0(t_1)) = F_{H_0}(p_0, t_1)$, and $F(\mathcal{F}_0(t_1), \mathring{p}_0) = F(\mathring{p}_0, \mathcal{F}_0(t_1)) = 1$.

Step 4 (adding transitions emulating rules): Each possible application $(G_i, \omega, G_j) \in \mathcal{A}$ of rule ω is modeled by a transition denoted by $\mathcal{E}(\omega_j^i)$, such that:

1. insert timed transition $\mathcal{E}(\omega_j^i)$ into T. Formally, $T \longleftarrow T \cup \{\mathcal{E}(\omega_j^i)\}$,
2. $F(\mathring{p}_i, \mathcal{E}(\omega_j^i)) = F(\mathcal{E}(\omega_j^i), \mathring{p}_j) = 1$. That is, firing $\mathcal{E}(\omega_j^i)$ removes a token from $\mathring{p}_i$ (associated with configuration G_i) and adds a token to $\mathring{p}_j$ (associated with configuration G_j), which models switching from G_i to G_j,
3. for each fresh place $p \in P_R \setminus P_I$, let $F(\mathcal{E}(\omega_j^i), \mathcal{E}(p)) = M_R^0(p)$. That is, firing $\mathcal{E}(\omega_j^i)$ initializes $\mathcal{E}(p)$, which emulates an addition of p,
4. for each obsolete place $p \in P_L \setminus P_I$, let $F(\mathcal{E}(p), \mathcal{E}(\omega_j^i)) = M_L^0(p)$, which models preconditions of applying rule ω. As well, add an immediate transition em that empties $\mathcal{E}(p)$ from tokens to emulate deletion of p as follows: (i) add an arc from $\mathcal{E}(p)$ to em, i.e., $F(\mathcal{E}(p), em) = 1$, and (ii) connect em with $\mathring{p}_j$ by a self-loop, which allows em to start emptying $\mathcal{E}(p)$ when current configuration becomes G_j which does not contain place p,
5. for each interface place $p \in P_I$, let $F(\mathcal{E}(p), \mathcal{E}(\omega_j^i)) = M_L^0(p)$ and $F(\mathcal{E}(\omega_j^i), \mathcal{E}(p)) = M_R^0(p)$,
6. $\Lambda(\mathcal{E}(\omega_j^i)) = \lambda$, where λ is an application rate of ω.

Note that, any timed transition emulating a rule application is disabled, if an immediate transition emptying a place is enabled, due to GSPN nature. Hence, any emulation of reconfiguration is not allowed until each transition that empties an obsolete place image is no longer enabled. This is an indispensable behavior. In fact, the latter guarantees that the equivalent net does never emulate adding a place with additional tokens than specified by rule ω. That is, if

1. previous configuration of the dynamic net was G_i which contains place p,
2. current configuration is G_j which does not contain p,
3. and dynamic net is reconfigured towards a configuration G_k that may contain place p (as a fresh place), such that its marking n is computed by Eq. 3.2.

Then, equivalent net $\mathcal{H}$ must remove all tokens from place p, before emulating a reconfiguration from G_j towards G_k (otherwise, if p still contain tokens, then its marking will not be equal to n). This behavior is guaranteed by Step (4).

For instance, application (H_1, ω_2, H_0) of rule ω_2 is modeled by transition $\mathcal{E}_0^1(\omega_2)$ shown in Fig. 5.3, such that:

1. $F(\mathring{p}_1, \mathcal{E}_0^1(\omega_2)) = F(\mathcal{E}_0^1(\omega_2), \mathring{p}_0) = 1$, to model the switching from H_1 to H_0,
2. as for interface place p_1, we have: $F(\mathcal{E}(p_1), \mathcal{E}_0^1(\omega_2)) = M_{L_2}^0(p_1) = 5$ and $F(\mathcal{E}_0^1(\omega_2), \mathcal{E}(p_1)) = M_{R_2}^0(p_1) = 3$,
3. as for obsolete place p_2, we have: $F(\mathcal{E}(p_2), em) = 1$ to model its deletion, such that em is an immediate transition and it is connected with place $\mathring{p}_0$ (associated with configuration H_0) with a self-loop.

As for application (H_0, ω_1, H_1) of rule ω_1, it is modeled by transition $\mathcal{E}_1^0(\omega_1)$ shown in Fig. 5.3, such that:

1. $F(\mathring{p}_0, \mathcal{E}_1^0(\omega_1)) = F(\mathcal{E}_1^0(\omega_1), \mathring{p}_1) = 1$, to model switching from H_0 to H_1,
2. as for fresh place p_2, we have: $F(\mathcal{E}_1^0(\omega_1), \mathcal{E}(p_2)) = M_{R_1}^0(p_2) = 2$,
3. as for interface place p_0, we have: $F(\mathcal{E}(p_0), \mathcal{E}_1^0(\omega_1)) = M_{L_1}^0(p_0) = 3$ and $F(\mathcal{E}_1^0(\omega_1), \mathcal{E}(p_0)) = M_{R_1}^0(p_1) = 3$.

5.4 Qualitative/Quantitative Analysis of D-GSPNs

This section develops qualitative/quantitative verification of a given D-GSPN $\mathcal{D}$. This analysis is derived from that of equivalent GSPN $\mathcal{H}$ of $\mathcal{D}$.

We consider only some important properties. We have the following.

- a transition t of $\mathcal{D}$ is live, iff there exists an equivalent live transition thereof.
- a rule $r \in \mathcal{R}$ is live, iff there exists a live transition $\mathcal{E}(\omega_j^i)$ that models ω.
- a place p of $\mathcal{D}$ is k-bounded, iff $\mathcal{E}(p)$ is k-bounded.

- $\mathcal{D}$ is k-bounded (respectively, safe), iff $\mathcal{H}$ is k-bounded (respectively, safe).
- there exists a timeless trap in $\mathcal{D}$, iff there exists a timeless trap in $\mathcal{H}$.
- **Probability of having n tokens at a place:** The probability of having n tokens at place p of $\mathcal{D}$ is equal to the probability of having n tokens at its equivalent place $\mathcal{E}(p)$ of $\mathcal{H}$.
- **Mean number of tokens:** Average number of tokens at place p_i equals average number of tokens at its equivalent place $\mathcal{E}(p_i)$.
- **Probability of firing transition:** Probability pr_t, that transition t of $\mathcal{D}$ fires next is given by

$$pr_t = \sum_{t \in T_{G_i}} pr(\mathcal{F}_i(t)). \tag{5.1}$$

  where $pr(\mathcal{F}_i(t))$ is a probability that transition $\mathcal{F}_i(t)$ fires next in $\mathcal{H}$.
- **Throughput:** Throughput d_t at transition t of $\mathcal{D}$ is given by

$$d_t = \sum_{t \in T_{G_i}} d(\mathcal{F}_i(t)). \tag{5.2}$$

  where $d(\mathcal{F}_i(t))$ is throughput at transition $\mathcal{F}_i(t)$ in GSPN $\mathcal{H}$.

5.5 Proofs

In this section, we prove the equivalence between any given D-GSPN $\mathcal{D}$ and GSPN $\mathcal{H}$ obtained by applying the transformation algorithm presented above. We prove that if a sequence of transitions and rules $\sigma = t_0\ t_1 \ldots\ t_n\ \omega$ is fireable at marking $\mathcal{M}(p_0, p_1, \ldots, (p_m)) = (w_0, w_1, \ldots, w_m)$ in configuration G_i in $\mathcal{D}$, then transition sequence $\sigma' = \mathcal{F}_i(t_0)\ \mathcal{F}_i(t_1) \ldots\ \mathcal{F}_i(t_n)\ \mathcal{E}^i(\omega)$ is fireable at marking $M(\mathcal{E}(p_0), \mathcal{E}(p_1), \ldots, \mathcal{E}(p_m), \mathring{p}_i) = (w_0, w_1, \ldots, w_m, 1)$ in $\mathcal{H}$, and vice versa. That is, GSPN $\mathcal{H}$ preserves the behaviors of $\mathcal{D}$.

As shown in Figs. 5.4 and 5.5 depicting reachability graphs of GSPN $\mathcal{H}_0$ and D-GSPN $\mathcal{D}_0$, respectively, if state s_j is reached from s_i by $\mathcal{F}_k(t)$, then state e_j is reached from e_i by t, such that $e_i(p_l) = s_i(p_l)$, for all $p_l \in G_k$ (similarly to s_j and e_j). Consider switching from H_1 to H_0 modeled by transition from s_{14} to s_0 listed in Table 5.1 as well as by transition from e_{14} to e_0 listed in Table 5.2, that deletes place p_2. In fact, marking of p_2 at s_{14} is equal to zero. Hence, there is no token to be consumed by transition em (the role of em is removing all tokens from place p_2, when the current configuration becomes H_0). Consequently, in this specific case, both reachability graphs have the same number of states and shape.

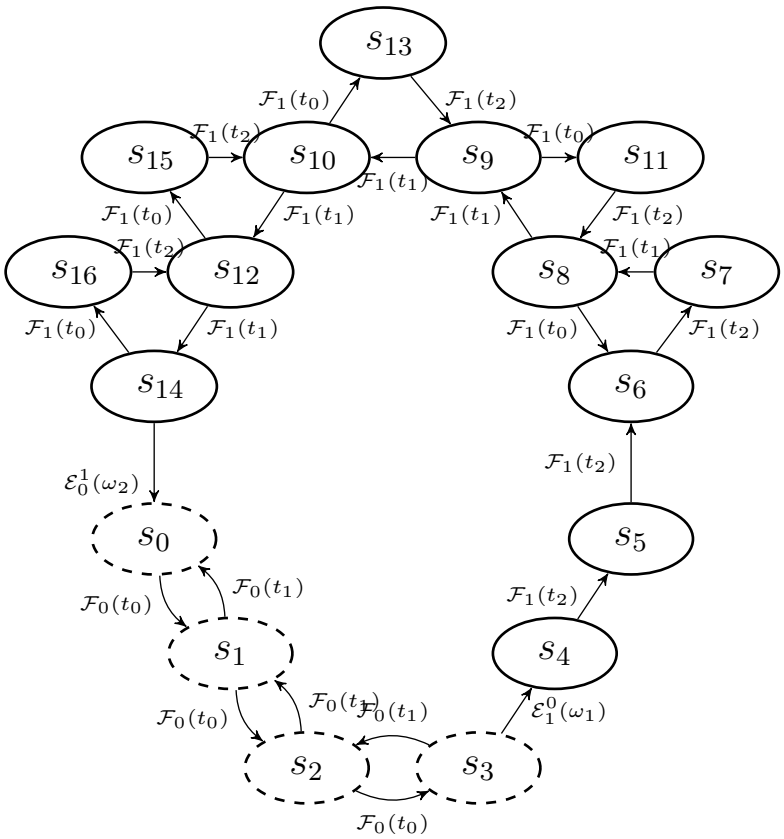


Figure 5.4: Reachability graph of equivalent GSPN $\mathcal{H}_0$ to $\mathcal{D}_0$, where marking of $\mathring{p}_0$ (resp. $\mathring{p}_1$) equals one in dashed (resp. solid) states.

By Step (1) of the proposed algorithm, sub-initial marking $\mathcal{M}(\mathcal{E}(p_0), \mathcal{E}(p_1), \ldots, \mathcal{E}(p_m))$ equals $M^0_{G_0}(p_0, p_1, \ldots, p_m)$, where $m = |P_{G_0}|$. Furthermore, for any transition $t \in T_{G_0}$, $F(\mathcal{E}(p_i), \mathcal{F}_0(t)) = F_{G_0}(p_i, t)$, for all $i \in \{0, \ldots, m\}$, hence if t is fireable at $M^0_{G_0}$, then $\mathcal{F}_0(t)$ is fireable at $\mathcal{M}$. Moreover, for any transition $t \in T_{G_0}$, $F(\mathcal{F}_0(t), \mathcal{E}(p_i)) = F_{G_0}(t, p_i)$, for all $i \in \{0, \ldots, m\}$, hence if firing t at $M^0_{G_0}$ yields M', then firing $\mathcal{F}_0(t)$ at $\mathcal{M}$ yields $\mathcal{M}'$, such that $\mathcal{M}'(\mathcal{E}(p_0), \mathcal{E}(p_1), \ldots, \mathcal{E}(p_m))$ equals $M'(p_0, p_1, \ldots, p_m)$. As for transformation rules, by Step (3), if a rule ω is applicable to configuration G_0 at marking M yielding a configuration G_i, where $M(p_j) \geq M^0_L(p_j)$, for all $p_j \in P_L$, then the transformation algorithm creates an arc from each equivalent place to p_j of P_L to transition $\mathcal{E}(\omega^0_i)$, such that arc weight is $M^0_L(p_j)$, to model pre-conditions of ω. Consequently, if a transition sequence $t_0\ t_1 \ldots\ t_n$ is fireable at marking $M(p_0, p_1, \ldots, p_m) = (w_0, w_1, \ldots, w_m)$ of configuration G_0 and followed by an application of a rule ω, then transition sequence $\mathcal{F}_0(t_0)\ \mathcal{F}_0(t_1) \ldots\ \mathcal{F}_0(t_n)\ \mathcal{E}^0_i(\omega)$ is fireable at marking $\mathcal{M}(\mathcal{E}(p_0), \mathcal{E}(p_1), \ldots, \mathcal{E}(p_m), \mathring{p}_0) = (w_0, w_1, \ldots, w_m, 1)$ in G, and vice versa.

Besides, the transformation algorithm creates an arc from $\mathcal{E}(\omega^0_i)$ to each equivalent place to p_j of P_R to model post-conditions of ω. Firing $\mathcal{E}(\omega^0_i)$ removes a token from $\mathring{p}_0$ and adds a token to $\mathring{p}_i$ (initially empty). If there exists some obsolete place, the algorithm adds an immediate transition to remove all tokens from its equivalent places.

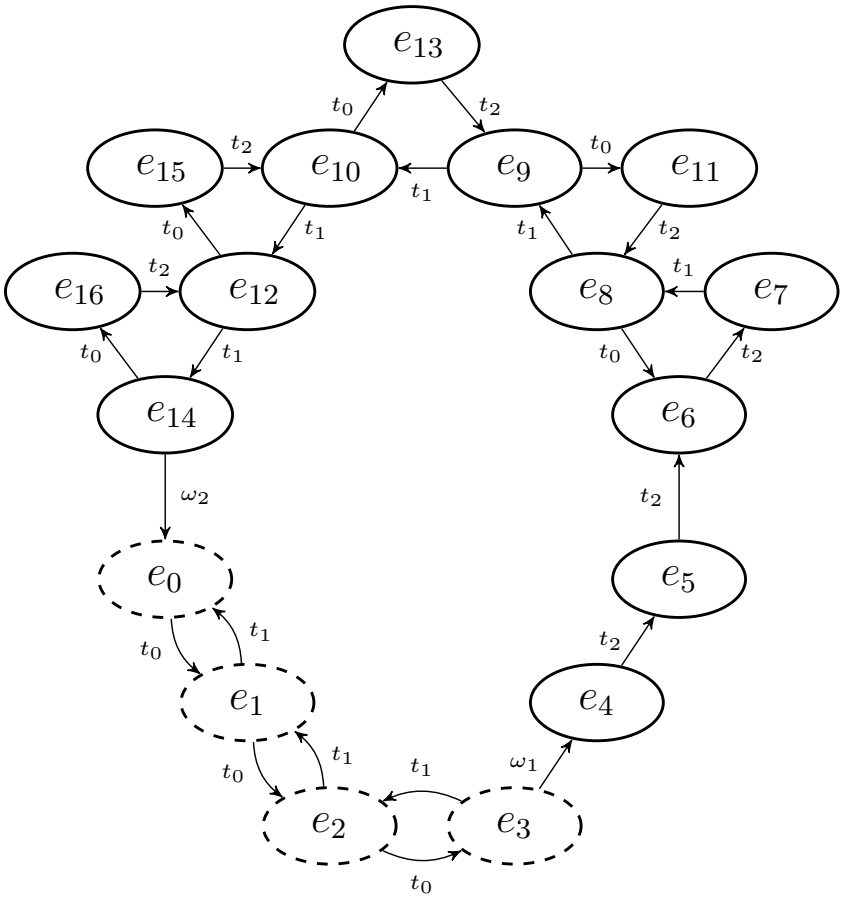


Figure 5.5: Reachability graph of D-GSPN $\mathcal{D}_0$, where dashed (resp. solid) states correspondent to H_0 (resp. H_1).

	$\mathring{p}_0$	$\mathring{p}_1$	p_0	p_1	p_2		$\mathring{p}_0$	$\mathring{p}_1$	p_0	p_1	p_2
s_0	1	0	0	3	0	s_8	0	1	0	1	4
s_1	1	0	1	2	0	s_9	0	1	0	2	3
s_2	1	0	2	1	0	s_{10}	0	1	0	3	2
s_3	1	0	3	0	0	s_{11}	0	1	1	1	3
s_4	0	1	3	0	2	s_{12}	0	1	0	4	1
s_5	0	1	2	0	3	s_{13}	0	1	1	2	2
s_6	0	1	1	0	4	s_{14}	0	1	0	5	0
s_7	0	1	0	0	5	s_{15}	0	1	1	3	1
						s_{16}	0	1	1	4	0

Table 5.1: Reachability set of $\mathcal{H}_0$.

	p_0	p_1	p_2		p_0	p_1	p_2
e_0	0	3	//	e_8	0	1	4
e_1	1	2	//	e_9	0	2	3
e_2	2	1	//	e_{10}	0	3	2
e_3	3	0	//	e_{11}	1	1	3
e_4	3	0	2	e_{12}	0	4	1
e_5	2	0	3	e_{13}	1	2	2
e_6	1	0	4	e_{14}	0	5	0
e_7	0	0	5	e_{15}	1	3	1
				e_{16}	1	4	0

Table 5.2: Reachability set of $\mathcal{D}_0$.

Similarly to the previous proof, we can prove that if sequences of transitions are fireable and rules are applicable in/to configuration G_i of D-GSPN $\mathcal{D}$ at marking M, then sequences of their equivalent transitions are fireable in GSPN $\mathcal{H}$ at marking $\mathcal{M}$, such that $\mathcal{M}(\mathcal{E}(p_j)) = M(p_j)$, for all places of P_{G_i}, after emptying equivalent places of obsolete ones, and vice versa. Hence, GSPN $\mathcal{H}$ is equivalent to D-GSPN $\mathcal{D}$.

5.6 Illustrative example

To illustrate the use of the proposed approach, we consider a reconfigurable manufacturing system (RMS) composed of three machines $\mathtt{M_1}$, $\mathtt{M_2}$ and $\mathtt{M_3}$; and three buffers $\mathtt{buf_A}$, $\mathtt{buf_B}$ and $\mathtt{buf_I}$. This RMS produces two product types, namely, A and B. Machines $\mathtt{M_1}$ and $\mathtt{M_2}$ can process items of type A and B, whereas machine $\mathtt{M_3}$ can only process product of type A.

This RMS has three possible configurations: initial, alternative 1 and alternative 2. In the initial configuration each product, either A or B, is processed first by $\mathtt{M_1}$ and then by $\mathtt{M_2}$. To produce a part A, a raw material is loaded in buffer $\mathtt{buf_A}$ having capacity of ten spaces. Then, when machine $\mathtt{M_1}$ is idle, it loads a waiting material from $\mathtt{buf_A}$, processes it and unloads it in buffer $\mathtt{buf_I}$ having capacity of five spaces. Finally, when machine $\mathtt{M_2}$ is idle, it loads an unfinished part A from $\mathtt{buf_I}$ and finishes its processing. Analogically, product B is processed, such that machine $\mathtt{M_1}$ loads raw materials from buffer $\mathtt{buf_B}$ having capacity of five spaces.

The rates of processing a product A (resp. B) by $\mathtt{M_1}$ and $\mathtt{M_2}$ are α_1 (resp. β_1) and α_2 (resp. β_2), respectively. The initial configuration, called $\mathtt{C_0}$, is shown in Fig. 5.6, and the meaning of places and transitions is given in Table 5.3.

Machine $\mathtt{M_3}$ is activated when there is a strong demand on A. There are two possible alternatives of activating this machine such that it performs either (i) the first step of processing A with rate α_3, or (ii) the second one with rate α_3'.

The activation of $\mathtt{M_3}$ according to the first alternative is modeled by applying rule $\mathtt{r_1}$ depicted in Fig. 5.7 to $\mathtt{C_0}$. Obtained GSPN $\mathtt{C_1}$ is illustrated in Fig. 5.8.

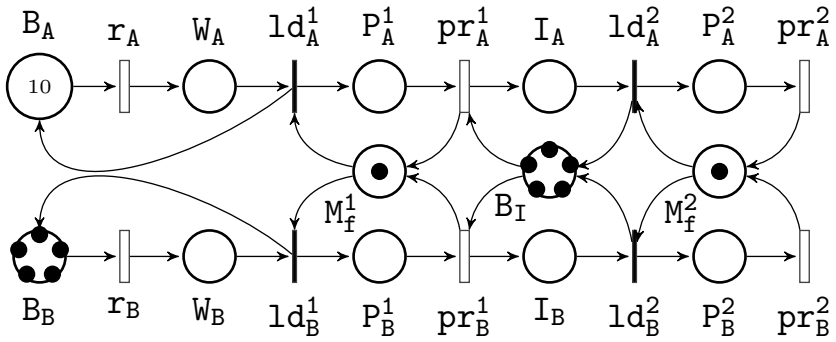


Figure 5.6: Initial configuration $\mathtt{C_0}$.

Place	**Description**
$\mathtt{B_A}$	Its number of tokens models the number of available spaces in $\mathtt{buf_A}$
$\mathtt{B_B}$	Its number of tokens models the number of available spaces in $\mathtt{buf_B}$
$\mathtt{W_A}$	Its number of tokens models the number of waiting materials in $\mathtt{buf_A}$
$\mathtt{W_B}$	Its number of tokens models the number of waiting materials in $\mathtt{buf_B}$
$\mathtt{B_I}$	Its number of tokens models the number of waiting parts in $\mathtt{buf_I}$
$\mathtt{P_A^i}$	A token inside this place means that $\mathtt{M_i}$ is treating a product A
$\mathtt{P_B^i}$	A token inside this place means that $\mathtt{M_i}$ is treating a product B
$\mathtt{I_A}$	Its marking models the number of unfinished products A in $\mathtt{buf_I}$
$\mathtt{I_B}$	Its marking models the number of unfinished products B in $\mathtt{buf_I}$
$\mathtt{M_f^i}$	A token in $\mathtt{M_f^i}$ means that $\mathtt{M_i}$ is idle
Transition	**Description**
$\mathtt{r_A}$ (resp. $\mathtt{r_B}$)	Loading a raw material in $\mathtt{buf_A}$ (resp. $\mathtt{buf_B}$)
$\mathtt{ld_A^1}$ (resp. $\mathtt{ld_B^1}$)	$\mathtt{M_1}$ loads an item from buffer $\mathtt{buf_A}$ (resp. $\mathtt{buf_B}$)
$\mathtt{ld_A^2}$ (resp. $\mathtt{ld_B^2}$)	$\mathtt{M_2}$ loads an unfinished A (resp. B) from buffer $\mathtt{buf_I}$
$\mathtt{pr_A^1}$ (resp. $\mathtt{pr_B^1}$)	$\mathtt{M_1}$ processes a product A (resp. B)
$\mathtt{pr_A^2}$ (resp. $\mathtt{pr_B^2}$)	$\mathtt{M_2}$ processes a product A (resp. B)

Table 5.3: Meaning of places and transitions in configuration $\mathtt{C_0}$.

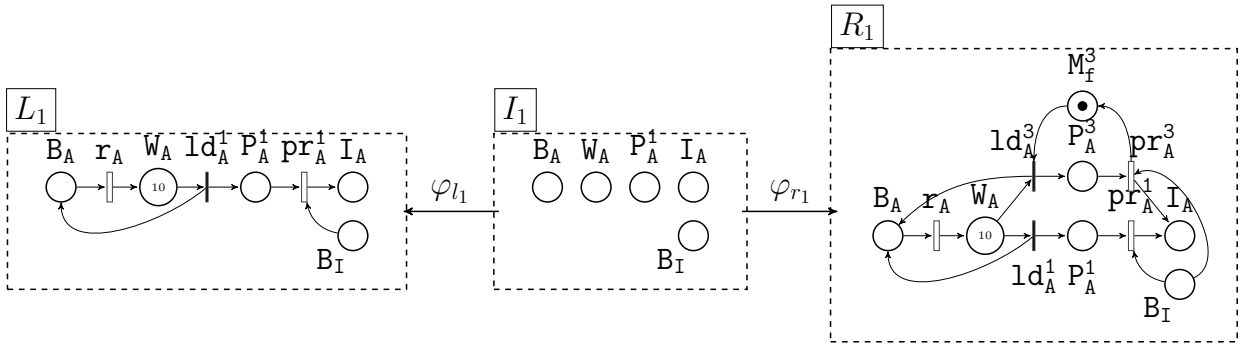


Figure 5.7: Rule $\mathtt{r_1}$.

Rule $\mathtt{r_1}$ is applied to $\mathtt{C_0}$, in addition, when the marking of place $\mathtt{W_A}$ is ten, that is, $\mathtt{buf_A}$ is full. The meanings of new places/transitions $\mathtt{ld_A^3}$, $\mathtt{M_F^3}$, $\mathtt{P_A^3}$ and $\mathtt{pr_A^3}$ is analogue to $\mathtt{ld_A^1}$, $\mathtt{M_F^1}$, $\mathtt{P_A^1}$ and $\mathtt{pr_A^1}$ w.r.t. machine $\mathtt{M_3}$. Once machine $\mathtt{M_3}$ is idle and there is no waiting raw material in $\mathtt{buf_A}$ (modeled by ten tokens in $\mathtt{B_A}$), machine $\mathtt{M_3}$ is deactivated. This deactivation is modeled by applying rule $\mathtt{r_2}$ illustrated in Fig.5.9 to GSPN $\mathtt{C_1}$. The obtained GSPN is $\mathtt{C_0}$.

The activation of $\mathtt{M_3}$ according to the second alternative is modeled by applying $\mathtt{r_3}$ depicted in Fig. 5.10 to $\mathtt{C_0}$. Obtained GSPN $\mathtt{C_2}$ is illustrated in Fig. 5.12. The meaning of new places/transitions $\mathtt{ld_A^3}$, $\mathtt{M_F^3}$, $\mathtt{P_A^3}$ and $\mathtt{pr_A^3}$ is analogue to $\mathtt{ld_A^2}$, $\mathtt{M_F^2}$, $\mathtt{P_A^2}$ and $\mathtt{pr_A^2}$ w.r.t. $\mathtt{M_3}$. Once $\mathtt{M_3}$ is idle and there is no waiting raw material in $\mathtt{buf_A}$, then $\mathtt{M_3}$ is deactivated. This deactivation is modeled by applying $\mathtt{r_4}$ illustrated in Fig.5.11 to $\mathtt{C_2}$. The obtained GSPN is $\mathtt{C_0}$.

To choose an operating mode (either alternative 1 or alternative 2) of machine $\mathtt{M_3}$, we define the two following D-GSPNs. D-GSPN $\mathcal{D}_1$ models the RMS having two configurations $\mathtt{C_0}$ and $\mathtt{C_1}$, whereas D-GSPN $\mathcal{D}_2$ models the RMS having two configurations $\mathtt{C_0}$ and $\mathtt{C_2}$. Formally, $\mathcal{D}_1 = \langle \mathtt{C_0}, \{\mathtt{r_1}, \mathtt{r_2}\}\rangle$ and $\mathcal{D}_2 = \langle \mathtt{C_0}, \{\mathtt{r_3}, \mathtt{r_4}\}\rangle$.

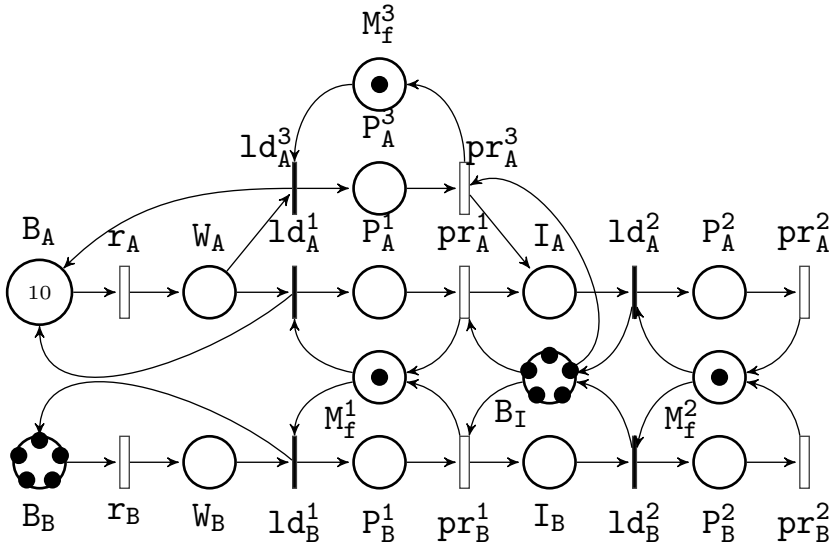


Figure 5.8: Configuration $\mathtt{C}_1$ (Alternative 1).

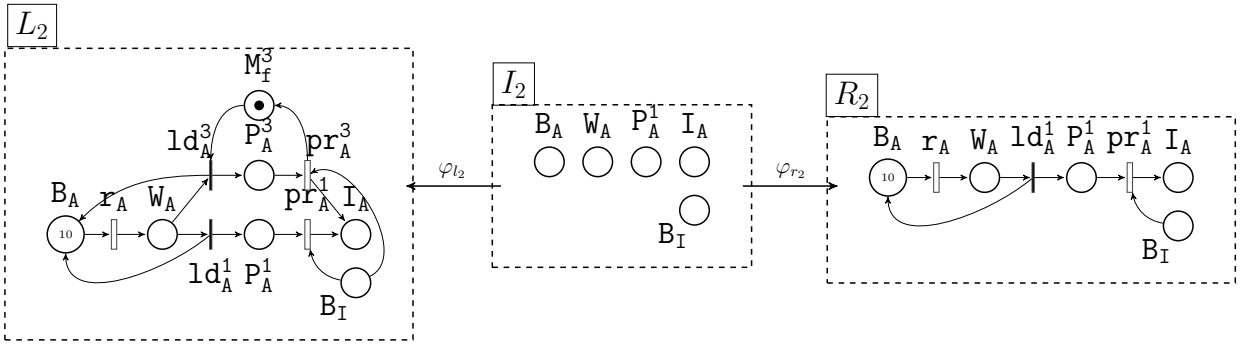


Figure 5.9: Rule $\mathtt{r}_2$.

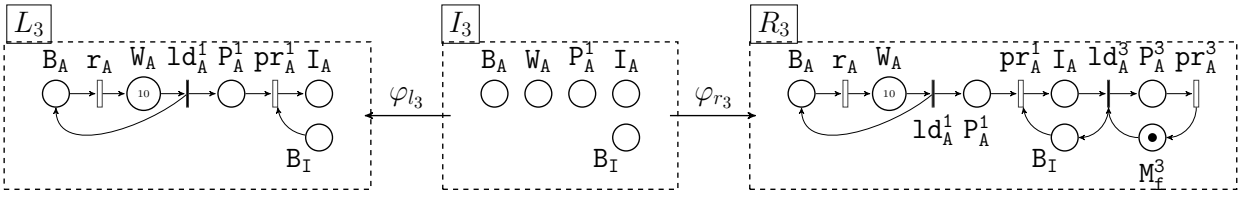


Figure 5.10: Rule $\mathtt{r}_3$.

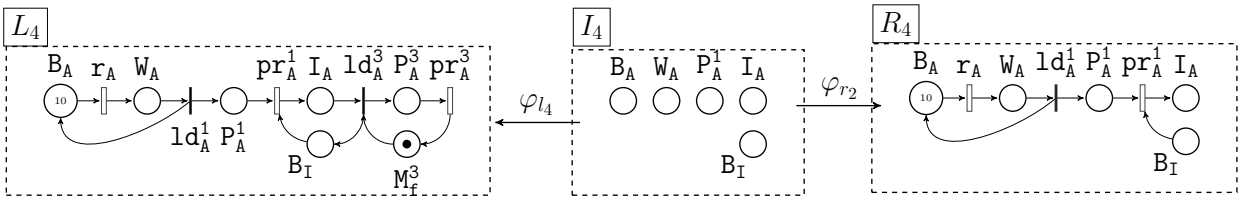


Figure 5.11: Rule $\mathtt{r}_4$.

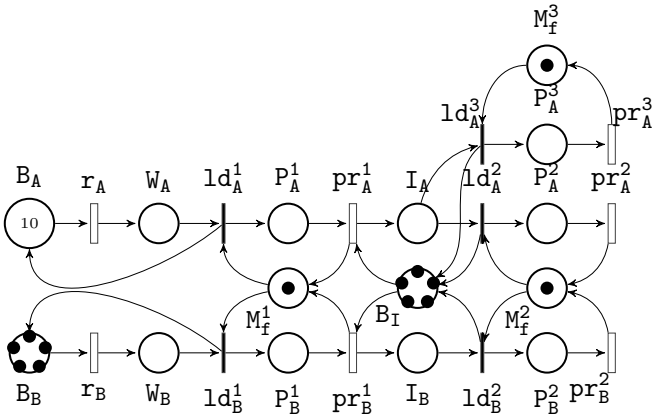


Figure 5.12: Configuration $\mathtt{C}_2$.

We apply the proposed transformation algorithm to both $\mathcal{D}_1$ and $\mathcal{D}_2$, where the obtained GSPNs are $\mathcal{H}_1$ and $\mathcal{H}_2$ shown in Figs.5.13 and 5.14, respectively.

Given the rates shown in Table 5.4, we evaluate the performances of $\mathcal{H}_1$ and $\mathcal{H}_2$. Throughput of producing A and B products are depicted in Fig 5.15. The throughput of producing A in the first case is the throughput of transition $\mathtt{pr}_\mathtt{A}^2$; and in the second case is the sum of $\mathtt{pr}_\mathtt{A}^2$ and $\mathtt{pr}_\mathtt{A}^3$ throughput. As for the throughput of producing B in both cases is the throughput of transition $\mathtt{pr}_\mathtt{B}^2$.

The results show that the throughput of A production is higher in the first case than the second, however, the throughput of B production is higher in the second case than the first.

Indeed, in the first case both machines $\mathtt{M}_1$ and $\mathtt{M}_3$ load raw materials from buffer $\mathtt{buf}_\mathtt{A}$, perform the first step of processing products of type A and put them in the intermediate buffer $\mathtt{buf}_\mathtt{I}$. This behavior puts more unfinished parts of type A in $\mathtt{buf}_\mathtt{I}$ to the detriment of unfinished parts of type B, which decreases its production throughput.

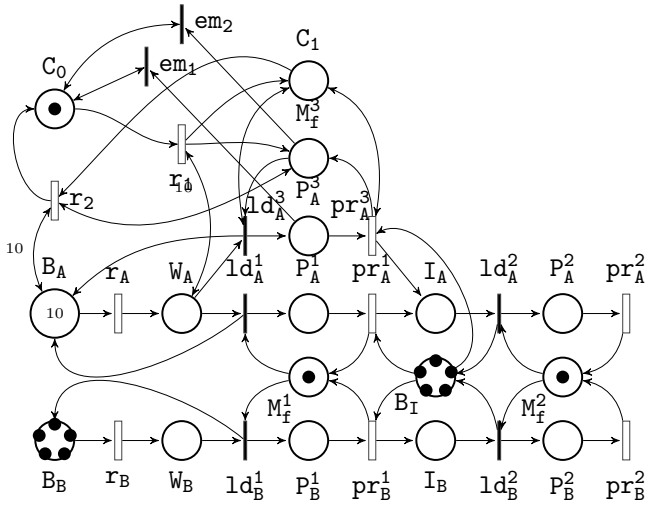


Figure 5.13: The transformation of $\mathcal{D}_1$.

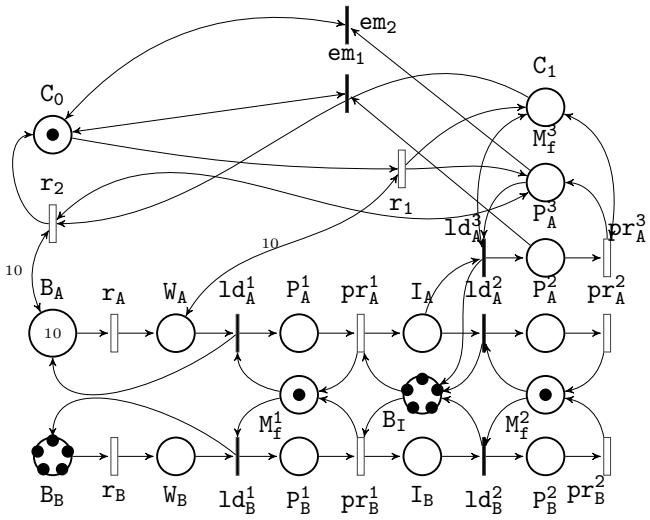


Figure 5.14: The transformation of $\mathcal{D}_2$.

Transition	Rate	Transition	Rate
$\mathtt{r_A}$	3	$\mathtt{pr}_\mathtt{B}^2$	β_2=6
$\mathtt{r_B}$	2	$\mathtt{pr}_\mathtt{A}^3$ (Alter. 1)	α_3=20
$\mathtt{pr}_\mathtt{A}^1$	α_1=10	$\mathtt{pr}_\mathtt{A}^3$ (Alter. 2)	α'_3=10
$\mathtt{pr}_\mathtt{B}^1$	β_1=2	$\mathtt{r}_1$	100
$\mathtt{pr}_\mathtt{A}^2$	α_2=3	$\mathtt{r}_2$	200

Table 5.4: Transition rates in both alternatives.

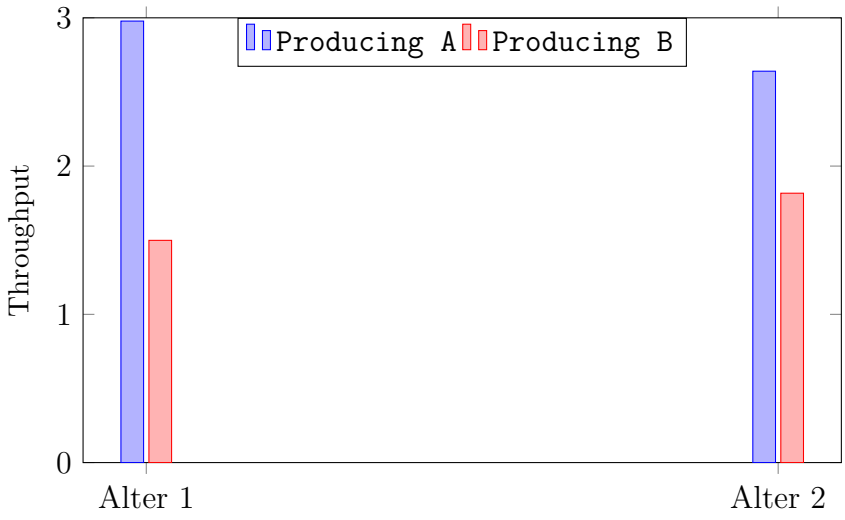


Figure 5.15: Throughput of A and B productions.

5.7 Conclusion

In this chapter, we proposed an extension of generalized stochastic Petri nets, called dynamic generalized stochastic Petri nets, that allows unrestricted reconfiguration forms while preserving generalized stochastic Petri nets decision power. Indeed, we propose to transform dynamic generalized stochastic Petri nets into generalized stochastic Petri nets, such that both qualitative and quantitative properties are still decidable using analysis methods proposed for generalized stochastic Petri nets.

However, the proposed transformation towards GSPNs can only take place when the obtained configuration set by graph transformations is finite, otherwise, transforming D-GSPNs into GSPNs may become infinite. To deal with infinite structures, we propose a new formalism in the next chapter that enables preserving several important properties after each reconfiguration which makes these preserved properties decidable whatever the number of configurations obtained by graph transformations.

Chapter 6

Reconfigurable Generalized Stochastic Petri Nets

6.1 Introduction

In the previous chapter, we have presented our formalism, called D-GSPNs, that allows modeling reconfigurability in GSPNs, as well as, transforming D-GSPNs into equivalent (basic) GSPNs for verification purpose. However, the major limitation of D-GSPNs resides in the fact that the number of configurations obtained by graph transformations must be finite. Yet, often applying transformation rules to graphs leads to structurally infinite models, and hence properties are not decidable based on classical verification techniques.

To consider infinite structure, initially, we were interested in extending INRSs [Li+09] (c.f. Section 3.7) to GSPNs and we could publish two papers [TKB17c, TKB16]. In these two extensions, each reconfiguration is expressed by a rule having left- and right-hand sides. The application of a rule implies the substitution of its left-hand side image in a given GSPN to be reconfigured by its right-hand side. These two sides must belong to particular sets in order to allow developers to reconfigure a live, bounded and reversible GSPN while preserving these three essential properties in the resulting model.

However, these reconfigurations have three major drawbacks: (i) system states are not considered (reconfigurations are done in an off-line mode), (ii) only three qualitative properties, namely liveness, boundedness and reversibility are decidable, and finally, (iii) the quantitative aspect of GSPNs is not studied.

To remedy these problems, we have proposed a new formalism, called the INRSs for GSPNs (INRSs-GSPNs) [Tig+18], which takes into account, inter alia, system states in the reconfiguration application and provides an algorithm for the property verification either qualitative or quantitative.

Using INRSs-GSPNs, designers can model reconfigurable systems using GSPNs and rewriting rules controlled by system state. Unlike our extensions described in [TKB17c, TKB16] which limit rule application to an initial marking, we associate to each reconfiguration a controller marking, that is, if the current marking of the net is less than a controller marking then the rule is not applicable. As for the verification level, an algorithm computing from the dynamic model a semi-Markov chain describing the stochastic behavior of the system is proposed. As a result, the designers can evaluate the system performance.

Nevertheless, the reconfiguration remains too limited in the INRSs-GSPNs formalism. On the one hand, only live, bounded and reversible GSPNs are concerned which limits its application field. On the other hand, left- and right-hand sides of any rule imperatively belong to a particular structure set, which can only further limit the formalism applicability.

The need to (i) relax the constraints imposed by INRSs-GSPNSs formalism, (ii) address all types of GSPNs (not only live, bounded and reversible GSPNs), and (iii) enrich the set of nets used in reconfiguration, led us to propose reconfigurable generalized stochastic Petri nets (RecGSPNs) [Tig+19].

Actually, RecGSPNs allow designers to model a wider range of possible structural changes where both sides of any rule are no longer defined by their structure, instead, they must show some behaviors allowing to use them in the reconfiguration process. Using RecGSPNs-based reconfiguration allows preserving five important properties such that liveness, boundedness, reversibility, deadlock-freedom and home state. Moreover, many properties expressed by linear time logic can be preserved after a system reconfiguration. Thus, these properties are decidable whatever the number of resulting configurations that can be infinite. This practice enjoys double advantages

1. Temporal and spatial complexity are reduced since these properties are verified only at the first configuration, hence no need to compute and explore the whole set of reachable states of all reachable configurations,

2. Often, applying transformation rules to graphs leads to structurally infinite models, and hence properties are not decidable based on classical verification techniques. In our approach, the properties mentioned above are still decidable since applying any rule will preserve them.

The remainder of this chapter is organized as follows. Section 6.2 presents the formal definition of RecGSPNs formalism. Then, properties preservation with necessary proofs and qualitative analysis are shown in Section 6.3 and Section 6.4, respectively. Section 6.5 demonstrates the use of the proposed formalism on a running example. Finally, Section 6.6 concludes the chapter.

6.2 Reconfigurable Generalized Stochastic Petri Nets

RecGSPNs introduce three major advantages: (i) any GSPN can be reconfigured at run-time while preserving several properties, such as: liveness, deadlock-free, boundedness, reversibility and home state, (ii) a wider range of possible structural changes are allowed, and (iii) RecGSPNs formalism is equipped by an algorithm that computes an isomorphic semi-Markov chain for quantitative verification.

6.2.1 Definition of RecGSPNs

A RecGSPN is composed of a GSPN, modeling an initial system configuration, and a set of rewriting rules describing possible changes in the structure of this system. Each rule r is composed of Left-hand Side (LHS), Right-hand Side (RHS) and input/output (interface) nodes, where LHS and RHS are Properties Preserving Nets (PPNs). We write $r = \langle LHS, RHS \rangle$. In the following, we present rule structure and PPN behavior by which a user can define its own rules used in a reconfiguration process.

In fact, imposing that LHS and RHS must be PPNs preserves certain behaviors of the system after a reconfiguration where LHS is replaced by RHS. Therefore, if some properties hold in the previous configuration, then they are still satisfied in the next configuration.

Intuitively, the application of rule $r = \langle LHS, RHS \rangle$ to G acts as follows. (i) r matches its LHS to a subnet of G, (ii) then it deletes this matching from G, (iii) and finally, it inserts and connects RHS to the rest of G, depending on input/output nodes.

To match LHS to a subnet of G, a morphism is used. It preserves a GSPN structure by mapping places to places and transitions to transitions so that if a transition t is mapped to a transition t', then a mapping between the preset and postset of both transitions t and t' must exist (similarly to places considering their marking) [HEM05].

Let us consider $G = \langle P_G, T_G, T_{1_G}, T_{2_G}, \Lambda_G \rangle$ as a GSPN, such that P_G denotes a set of places, T_G denotes a set of transitions, T_{1_G} denotes a set of timed transitions, T_{2_G} denotes a set of immediate transitions, and Λ_G denotes a firing rate/weight function.

Definition 6.1. (GSPNs-morphism). If G and H are to GSPNs, then a GSPNs-morphism from G to H is mapping $f : G \to H$ composed of a pair of mappings (f_P, f_T), such that $f_P : P_G \to P_H, f_T : T_G \to T_H$. f_P and f_T must satisfy the following conditions. $\forall\ p \in P_G$ and $\forall\ t \in T_G$:

- $F_G(p,t) = F_H(f_P(p), f_T(t))$ and $F_G(t,p) = F_H(f_T(t), f_P(p))$.
- $M_G(p) = M_H(f_P(p))$ and $\Lambda_G(t) = \Lambda_H(f_T(t))$.
- If $t \in T_{1_G}$, then $f_T(t) \in T_{1_H}$, otherwise $f_T(t) \in T_{2_H}$.

In addition to the well-known conditions of PNs-morphisms, a GSPNs-morphism requires mapped places to have identical marking and mapped transition to have identical rate/weight and type.

(a) g. (b) h_0. (c) h_1. (d) h_2.

Figure 6.1: Morphism.

Let us consider both GSPNs g and h_0 depicted in Figs. 6.1a and 6.1b, respectively. There exists a GSPNs-morphism f from g to h_0 that maps p, t and q in g to p_0, t_0 and q_0 in h_0, respectively. On the other hand, g cannot be matched to either h_1 or h_2 shown in Figs. 6.1c and 6.1d, respectively, since (i) marking of p_1 in h_1 does not equal marking of p in g, and (2) transition t in g is timed, while transition t_2 in h_2 is immediate.

Let us consider GSPN G_0 and rule $r = \langle lhs, rhs \rangle$ shown in Fig. 6.2. We apply rule r to GSPN G_0, as follows. Firstly, a match of lhs (LHS of r depicted in Fig. 6.2b) must be found in G_0. This match is located by morphism f that maps p_1, t_1 and q_1 in lhs to p_1, t_1 and p_2 in G_0, respectively. Secondly, we remove this match from G_0, the obtained net, called G'_0, is illustrated in Fig. 6.2d. Then, we insert rhs (RHS in r depicted in Fig. 6.2c) to G'_0, the obtained net, called G'_1, is highlighted in Fig. 6.2e. Finally, we connect rhs to G'_0. Actually, this last step depends on input/output nodes of both lhs and rhs. Given that sets of input nodes in lhs and rhs are $\{p_1\}$ and $\{p_1, p'_1\}$, respectively; and sets of output nodes in lhs and rhs are $\{q_1\}$ and $\{q_1, q'_1\}$, respectively, then each input node in rhs is connected to nodes of G'_0 identically as each input node in lhs is connected to nodes in G_0. Idem for output nodes.

In fact, all input nodes, as well as output nodes, must be identically connected to the rest of the net. The other nodes in rhs (i.e., which are neither input nor output nodes) are not connected directly, to G'_0. The resulting GSPN is G_1 depicted in Fig. 6.2f.

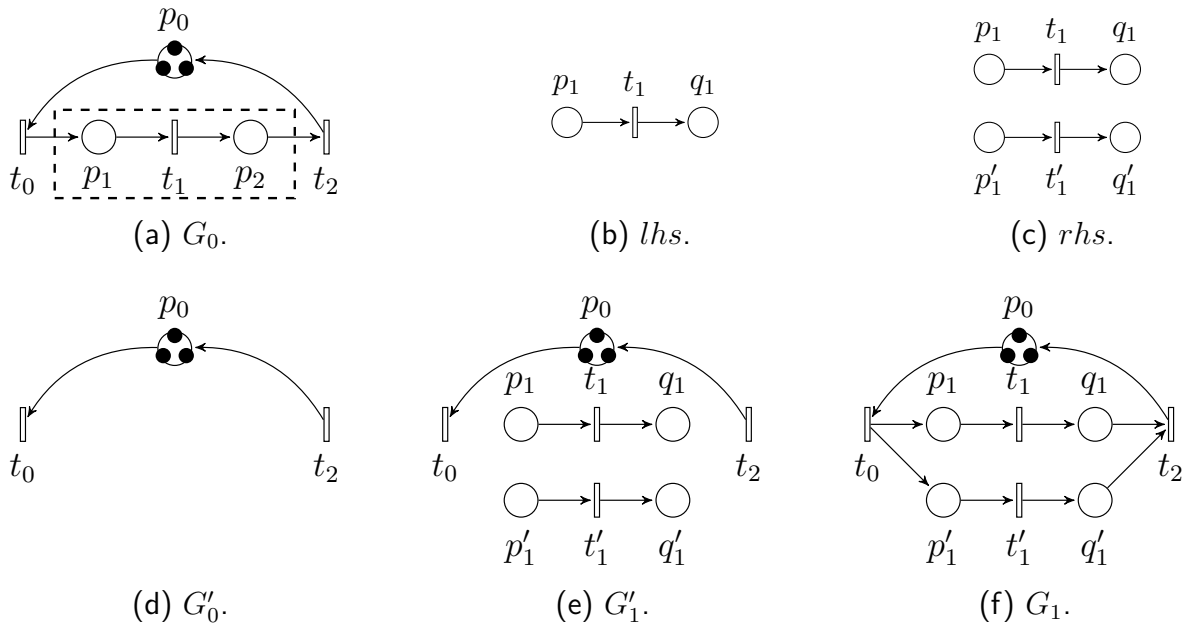


Figure 6.2: Steps of applying a rewriting rule.

Additional nets can be used to forbid a rule application if certain structures are present before applying thereof. This notion is called *Negative Application Conditions* (NACs) [LEO06]. A NAC(q) on LHS L is considered to be satisfied iff there does not exist a GSPNs-morphism that maps q in G.

Definition 6.2. (RecGSPNs). Formally, a RecGSPN is a tuple $\Psi = \langle G_0, \mathcal{R} \rangle$, where

1. $G_0 = \langle P, T, F, M_0, T_1, T_2, \Lambda \rangle$ is an initial configuration.
2. $\mathcal{R} = \{r_0, r_1, \ldots, r_m\}$ is a finite set of rewriting rules.

A rule $r_i = \langle L_i, R_i, \text{NAC}_i, I_i, O_i, V_i, m_i \rangle \in \mathcal{R}$ is defined as follows.

1. LHS L_i and RHS R_i are two Properties Preserving Nets (PPNs).
2. $\text{NAC}_i = \{\text{NAC}(q_0), \ldots, \text{NAC}(q_n)\}$ is a set of NACs on L.
3. $I_i = (I_{L_i}, I_{R_i})$ ($O_i = (O_{L_i}, O_{R_i})$, respectively) are input nodes (output nodes, respectively) of L_i and R_i, where (i) interface nodes in both sides are either timed transitions, immediate transitions, or places, and (ii) non-interface transitions in both sides are either timed or immediate.
4. If LHS does not contain timed transition, so does RHS, and vice versa.
5. $V_i \in \mathbb{R}^+$ is an application weight of r_i.
6. m_i is a marking that controls the application of rule r_i.
7. If one side (either LHS or RHS) is a single immediate transition, then the other side must not contain a timed transition.

Assume a GSPN G. Let N_G denote the set of nodes in G, ${}^\bullet v = \{w | F(w, v) \geq 1\}$ denote preset of node v, and $v^\bullet = \{w | F(v, w) \geq 1\}$ denote its postset, and $\Theta(L)$ denote set of inner nodes in L that are neither input nor output nodes.

A rule $r = \langle L, R, \text{NAC}, I, O, V, m \rangle$ is applicable to G with marking M iff

1. An occurrence g of L is located in G by a GSPNs-morphism f.
2. $\forall$ $\text{NAC}(q_i) \in \text{NAC}$, $\text{NAC}(q_i)$ is satisfied.
3. Preset and postset of each image of any inner node v of L are subsets of g. Formally, $\forall\ v \in f(\Theta(L)), {}^\bullet v \cup v^\bullet \subseteq f(L)$.
4. Input nodes images of L are connected identically to nodes not belonging to g, i.e., $\forall\ v \in N_G \setminus N_g, \forall\ w, w' \in I_L, F(v, f(w)) = F(v, f(w'))$ and $F(f(w), v) = F(f(w'), v)$. Idem for output nodes images.
5. M is greater than or equal to m. Formally, $M(p) \geq m(p), \forall\ p \in P_G$.
6. If one side (either LHS or RHS) is a single place, then the other side and G must not contain an immediate transition.

Applying rewriting rule $r = \langle L, R, \text{NAC}, I, O, V, m \rangle$ to GSPN G with marking M leads to new GSPN G'. Let F' be the flow function of G'. For each couple of nodes $(v, w) \in N_{G'} \times N_{G'}$, $F'(v, w)$ is defined as follows.

$$F'(v,w) = \begin{cases} F_G(v,w) & \text{if } v \notin N_R \wedge w \notin N_R \\ F_R(v,w) & \text{if } v \in N_R \wedge w \in N_R \\ F_G(v,f(u)) & \text{if } v \notin N_R \wedge w \in I_R \\ F_G(f(u),w) & \text{if } v \in I_R \ \wedge w \notin N_R \\ F_G(f(u'),w) & \text{if } v \in O_R \wedge w \notin N_R \\ F_G(v,f(u')) & \text{if } v \notin N_R \wedge w \in O_R \\ 0 & \text{otherwise} \end{cases} . \tag{6.1}$$

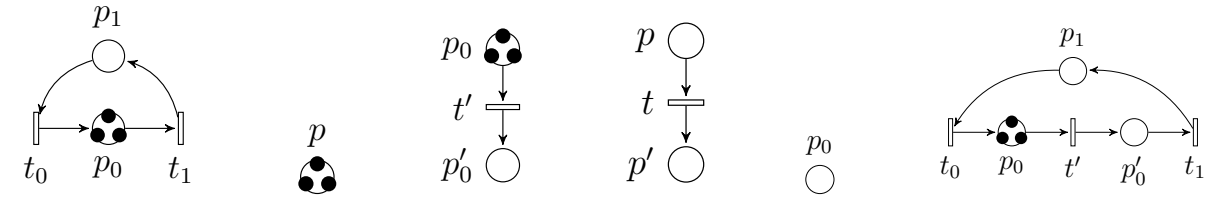
where $u \in I_L$, $u' \in O_L$ and f is a GSPNs-morphism that maps L into G.

Marking M' of G' is given by

$$M'(p) = \begin{cases} M(p) & \text{if } p \in P_G \\ M_{0_R}(p) & \text{if } p \in P_R \end{cases} . \tag{6.2}$$

where M_{0_R} is an initial marking of R.

Let us consider RecGSPN shown in Fig. 6.3, where its initial configuration C_0 is depicted in Fig. 6.3a. The net switches to configuration C_1 illustrated in Fig. 6.3f when the marking of place p_0 of C_0 is three. Once the marking of place p_1 of C_1 becomes three, this latter switches again to C_0.



(a) Configuration C_0. (b) L_0. (c) R_0. (d) L_1. (e) R_1. (f) Configuration C_1.

Figure 6.3: Reconfiguration in RecGSPNs.

Rule $r_0 = \langle L_0, R_0, \text{NAC}_0, I_0, O_0, V_0, m_0 \rangle$ models the switching from C_0 to C_1, where (i) L_0 and R_0 are shown in Figs. 6.3b and 6.3c, (ii) $I_0 = (\{p\}, \{p_0\})$, (iii) $O_0 = (\{p\}, \{p'_0\})$, (iv) $\text{NAC}_0 = \emptyset$, (v) $V_0 = 1$, and (vi) $m_0(p_0) = 3$.

Rule $r_1 = \langle L_1, R_1, \text{NAC}_1, I_1, O_1, V_1, m_1 \rangle$ models the switching from C_1 to C_0, where (i) L_1 and R_1 are shown in Figs. 6.3d and 6.3e, (ii) $I_1 = (\{p\}, \{p_0\})$, (iii) $O_1 = (\{p'\}, \{p_0\})$, (iv) $\text{NAC}_1 = \emptyset$, (v) $V_1 = 1$, and (vi) $m_1(p_1) = 3$.

Rule r_1 can be applied to C_1 when p_1 marking becomes three, since:

- There exists an occurrence g of L_1 mapped by GSPNs-morphism f_1 in C_1, where $f_1(p) = p_0$, $f_1(t) = t'$ and $f_1(p') = p'_0$.
- Condition (2) (applicability of rules) holds, since NAC_1 is empty.

- Preset and postset of inner transition t' are subsets of g.
- Condition (4) holds, since there is one input node and one output node.
- Condition (5) holds, since $M_{C_1}(p_0) \geq m(p_0)$.
- Condition (6) holds, since C_1 and L_1 contain only timed transition.

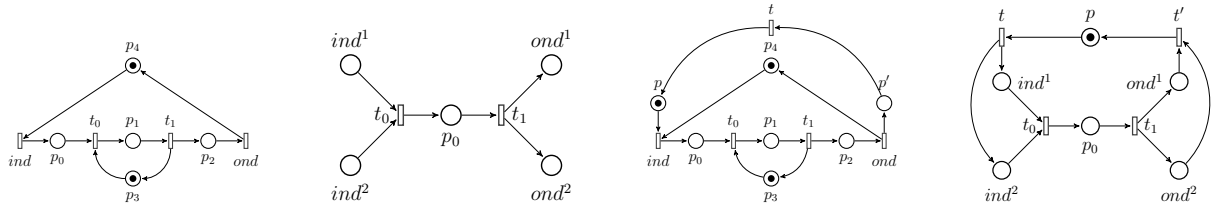
6.2.2 Properties Preserving Nets

In this subsection, we describe Properties Preserving Nets (PPNs) that are used to reconfigure GSPNs while preserving certain of their properties. Therefore, it is no more required to verify the preserved properties after each reconfiguration. Consequently, they are decidable even if the number of obtained configurations is infinite.

We determine whether a net N is a PPN by constructing a GSPN G, called container net, that contains N in a particular way described below. We show that if G is LBR (live, bounded, and reversible), then N is a PPN.

Definition 6.3. (Preserving Properties Nets with transition interface). A net N having transition interface (I_N, O_N) is a PPN, if its container net G is an LBR GSPN, where:

- A timed transition t, a marked place p by one token, a not marked place p' and N are inserted into G (initially empty).
- Preset and postset of t are $\{p'\}$ and $\{p\}$, respectively.
- An arc from place p to each input transition of N is inserted, formally $p^\bullet = I_N = \{t^1_{in}, \ldots, t^n_{in}\}$. $\{t^1_{in}, \ldots, t^n_{in}\}$ is the set of input transitions in N.
- An arc from each output transition of N to place p' is inserted, formally ${}^\bullet p' = O_N = \{t^1_{out}, \ldots, t^m_{out}\}$. $\{t^1_{out}, \ldots, t^m_{out}\}$ is the set of output transitions in N.
- If p and p' are initially not marked, then G is in a deadlock state.
- Transition t fires infinitely often.



(a) N_1. (b) N_2. (c) LBR G_1 contains N_1. (d) LBR G'_1 contains N_2.

Figure 6.4: Properties preserving nets examples.

Fig. 6.4a illustrates PPN N_1, where its interface is $(\{ind\}, \{ond\})$. Fig. 6.4c highlights its container net G_1, which is an LBR GSPN. GSPN G_1 is constructed as follows. We insert timed transition t, and two places p (marked by one token) and p' (not marked) into N_1. We add an arc from t to p and from p' to t. We add an arc from p to each input node, and an arc from each output node to p'. Note that in G_1, t fires infinitely often and when p and p' are initially not marked, G_1 is deadlocked.

Definition 6.4. (Properties Preserving Nets with place interface). A net N having place interface (I_N, O_N) is a PPN, if container net G of N is an LBR GSPN, where:

- A marked place p by one token, two timed transitions t, t' and N are inserted into G (initially empty).
- Preset and postset of p are $\{t'\}$ and $\{t\}$, respectively.
- An arc from transition t to each input place of N is inserted, formally $t^\bullet = I_N = \{p^1_{in}, \ldots, p^n_{in}\}$. $\{p^1_{in}, \ldots, p^n_{in}\}$ is the set of input places of N.
- An arc from each output place of N to transition t' is inserted, formally ${}^\bullet t' = O_N = \{p^1_{out}, \ldots, p^m_{out}\}$. $\{p^1_{out}, \ldots, p^m_{out}\}$ is the set of output places of N.
- If p is initially not marked, then G is deadlocked.
- Transitions t and t' fire infinitely often.

Fig. 6.4b depicts PPN N_2 having interface $(\{ind^1, ind^2\}, \{ond^1, ond^2\})$ and Fig. 6.4d shows its container net G'_1 which is an LBR GSPN.

6.3 Preservation of properties in RecGSPNs

In this section, we prove that the reconfiguration based on RecGSPNs preserves some properties such as LBR (i.e., liveness, boundedness, and reversibility), home state, deadlock-free, and other temporal linear properties.

6.3.1 Preservation of LBR, home state, and deadlock-free

Let $r = \langle L, R, \text{NAC}, I, O, V, m\rangle$ be a rule, $G = \langle P, T, F, M_0, T_1, T_2, \Lambda\rangle$ be its host GSPN, $G' = \langle P', T', F', M'_0, T'_1, T'_2, \Lambda'\rangle$ be its target GSPN, and L_G be the occurrence of L in G mapped by a morphism f.

In fact, applying rule r to G is performed through two steps. Firstly, we locate and delete nodes of L_G from G (deleted nodes are called *obsolete*), then we insert and connect nodes of R to the rest of G (inserted nodes are called *fresh*). All nodes which are not in occurrence

L_G are preserved in target GSPN G'. These nodes are called *preserved* nodes and denoted by $\mathcal{N} = \mathcal{T} \cup \mathcal{P}$, where $\mathcal{T}$ is the set of preserved transitions and $\mathcal{P}$ is the set of preserved places. Connections between preserved nodes are preserved as well. Each node connected to the image of an input node in L preserves its preset and each node connected to the image of an output node in L preserves its postset (see Eq. 6.1).

In the sequel, we consider the following notations:

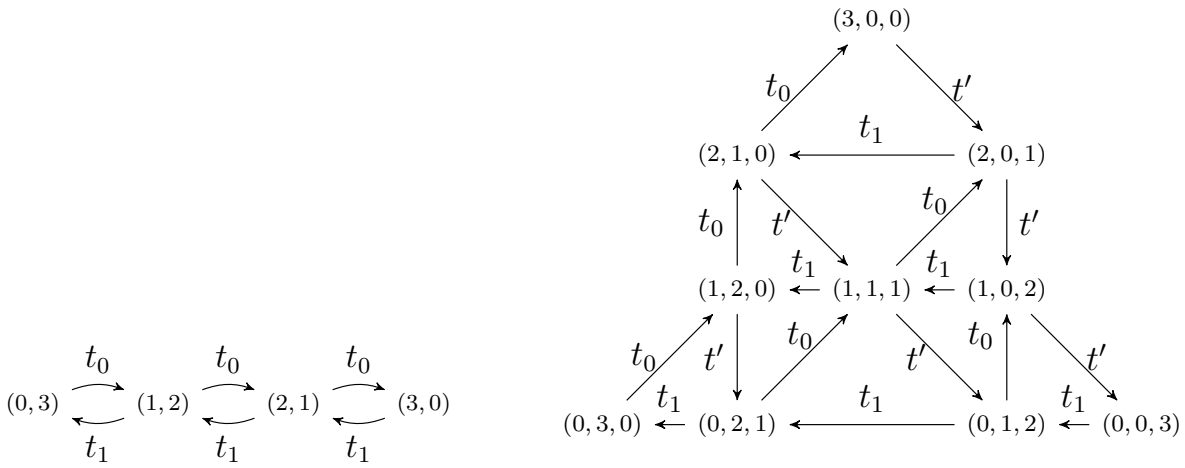
- $X.Y$=$(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m)$ denotes a concatenation of two vectors $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_m)$.
- $RM(G)$ denotes the reachable markings set of G.
- $M[\sigma(G)\rangle$ denotes fireable sequence $\sigma = t_1 t_2 \ldots t_n$ in G at marking M, such that $M[t_1\rangle M_1[t_2\rangle \ldots\ M_{n-1}[t_n\rangle$.
- $M[\sigma_1 \ldots \sigma_n(G)\rangle$ denotes a concatenation of fireable sequences in G at M.
- $\mathcal{T}(\sigma)$ denotes that all transitions in sequence σ are preserved after reconfiguration.
- $X.Y[\sigma(G)\rangle$ denotes a fireable sequence, such that $\sigma = t_1\ t_2\ ...\ t_n$ and $X.Y[t_1\rangle X_1.Y[t_2\rangle \ldots X_{n-1}.Y[t_n\rangle$. Hence, X_1, X_2 , ..., X_{n-1} are respectively the sequence of sub-markings obtained by firing sequence σ from sub-marking X. Here, Y is still not modified.
- $G(^\bullet n)$ ($G(n^\bullet)$, respectively) denotes preset (postset, respectively) of node n in G.

Denotation 1. *Let r be a rule, R be its RHS, G be its source GSPN, G' be its target GSPN, and $\mathcal{P}$ be a set of preserved places. If M' is a reachable marking in G', then M' can be written as a concatenation $M' = \mathcal{M}.M^R$, such that $\mathcal{M}$ is the marking of preserved places (where dimension$(\mathcal{M}) = |\mathcal{P}|$ and $\mathcal{M}(p) = M'(p)$, $\forall p \in \mathcal{P}$) and M^R is the marking of subnet R.*

Example. Fig. 6.5b shows the reachability graph of C_1 shown in Fig. 6.3f. Set of preserved places $\mathcal{P}$ is $\{p_1\}$ and marking $M = (3, 0, 0)$ of $p_0,\ p_1$ and p'_0 can be written as a concatenation $M = \mathcal{M}.M^R$, where $\mathcal{M} = (0)$ is the marking of preserved place $\{p_1\}$ and $M^R = (3, 0)$ is the marking of p_0 and p'_0.

Denotation 2. *Let r be a reconfiguration rule, L be its left-hand side, and G be its source GSPN. If M is a reachable marking in G, then M can be written as a concatenation $M = \mathcal{M}.M^L$ such that $\mathcal{M}$ is the marking of preserved places and M^L is the marking of subnet L_G (the occurrence of L in G).*

Example. Fig. 6.5a shows the reachability graph of C_0 shown in Fig. 6.3a. Marking $M' = (0, 3)$ of p_0 and p_1 can be written as $M' = \mathcal{M}'.M'^L$, where $\mathcal{M}' = (0)$ is the marking of p_1 and $M'^L = (3)$ is the marking of p_0.



(a) Reachability graph of C_0.

(b) Reachability graph of C_1.

Figure 6.5: Reachability graphs of both configurations C_0 and C_1, where $M_0(p_0, p_1) = (0, 3)$ is the initial marking of C_0, and $M'_0(p_0, p_1, p'_0) = (3, 0, 0)$ is the initial marking of C_1.

If the input and output nodes of RHS R are transitions, then $\forall\ t \in \mathcal{T}$, $G({}^\bullet t) = G'({}^\bullet t)$ and $G(t^\bullet) = G'(t^\bullet)$ (i.e., the preset and postset of t are preserved in G'), due to Eq. (6.1). Hence, if exists $t \in \mathcal{T}$ such that $\mathcal{M}.M^L[t\rangle\mathcal{M}'.M^L$ in G, then t is fireable in G' at $\mathcal{M}.M^R$ and the obtained marking is $\mathcal{M}'.M^R$. Therefore, we have the following lemma.

Lemma 3. *If the input and output nodes of RHS R are transitions, then each fireable sequence of preserved transitions in source GSPN G at $\mathcal{M}.M^L$ is a fireable sequence in target GSPN G' at $\mathcal{M}.M^R$.*

Proof. If the input and output nodes of R are transitions, then each preserved transition after reconfiguration preserves its preset and postset. Subsequently, if a preserved transition fires at $\mathcal{M}.M^L$ in G and yields $\mathcal{M}'.M^L$, then t fires at $\mathcal{M}.M^R$ in G' and yields marking $\mathcal{M}'.M^R$. Thus, if $t_0, t_1, \ldots, t_n$ are preserved and $\mathcal{M}.M^L[t_0\rangle\mathcal{M}_1.M^L[t_1\rangle \ldots \mathcal{M}_n.M^L[t_n\rangle$ is fireable in G, then $\mathcal{M}.M^R[t_0\rangle\mathcal{M}_1.M^R[t_1\rangle \ldots \mathcal{M}_n.M^R[t_n\rangle$ is fireable in G'. □

By Lemma 3, we mean that each fireable sequence σ in G at $\mathcal{M}.M^L$ containing only preserved transitions is also a fireable sequence in G' at $\mathcal{M}.M^R$.

In order to prove the reversibility of reconfigured nets, we divide set $RM(G')$ into two distinguishable subsets $RM_1(G')$ and $RM_2(G')$. The former contains reachable markings, where the marking of each place in RHS R equals its initial marking and the latter contains the rest of markings. We prove the reversibility for the markings of two subsets $RM_1(G')$ and $RM_2(G')$ in the two following lemmas.

Lemma 4. *If G is reversible, then the initial marking of G' is reachable from any marking $M = \mathcal{M}.M_0^R \in RM_1(G')$.*

Example. In Fig. 6.5b, the initial marking $(3,0,0)$ of C_1 is reachable from each marking in $RM_1(C_1) = \{(3,0,0)\}$.

Proof. Since G is reversible, then $\forall M \in RM(G)$, there exists a sequence σ such that $M[\sigma(G)\rangle M_0$. After reconfiguring G towards G', two cases are possible: either $\mathcal{T}(\sigma)$ (i.e., σ contains only preserved transitions) or $\neg\mathcal{T}(\sigma)$.
Case 1: If $\mathcal{T}(\sigma)$, then $M[\sigma(G)\rangle M_0$ can be written as $\mathcal{M}.M_0^L[\sigma(G)\rangle\mathcal{M}_0.M_0^L$, such that $\mathcal{M}_0.M_0^L$ is the initial marking of G. Thus, by Lemma 3, we deduct that $\mathcal{M}.M_0^R[\sigma(G')\rangle\mathcal{M}_0.M_0^R$, where $\mathcal{M}_0.M_0^R$ is the initial marking of G'.
Case 2: If $\neg\mathcal{T}(\sigma)$, then according to RecGSPNs, only the interface nodes in $f(L)$ (the image of net L inside G) can be connected to preserved nodes belonging to G (i.e., non-interface nodes in $f(L)$ can never be connected to other nodes outside $f(L)$ due to Condition (2)). Thus, sequence $\sigma = t_1...t_n$ can be divided into three sub-sequences $\sigma_1 = t_1...t_{i-1}$, $\sigma_2 = t_i...t_j$, and $\sigma_3 = t_{j+1}...t_n$, where $0 \leq i,j \leq n$, $\{t_1,...,t_{i-1},t_{j+1},...,t_n\} \subset \mathcal{T}$ and $t_k \notin \mathcal{T}$, $k \in \{i, i+1, ..., j\}$.
$\mathcal{M}.M_0^L[\sigma(G)\rangle\mathcal{M}_0.M_0^L$ can be written as $\mathcal{M}.M_0^L[\sigma_1.\sigma_2.\sigma_3(G)\rangle\mathcal{M}_0.M_0^L$ where:

- σ_1 does not update marking M_0^L: $\mathcal{M}.M_0^L[t_1\rangle\mathcal{M}_1.M_0^L \ \ldots \ \mathcal{M}_{i-2}.M_0^L[t_{i-1}\rangle \ \mathcal{M}_{i-1}.M_0^L$.
- σ_2 updates two markings $\mathcal{M}$ and M^L: $\mathcal{M}_{i-1}.M_0^L[t_i\rangle\mathcal{M}_i.M_1^L \ \ldots \ \mathcal{M}_i.M_{j-1}^L \ [t_j\rangle\mathcal{M}_j.M_0^L$, such that t_i and t_j are respectively input and output nodes in $f(L)$.
- σ_3 updates marking $\mathcal{M}_j, \mathcal{M}_{j+1}, \ldots, \mathcal{M}_{n-1}$: $\mathcal{M}_j.M_0^L[t_{j+1}\rangle\mathcal{M}_{j+1}.M_0^L \ \ldots \ \mathcal{M}_{n-1}.M_0^L[t_n\rangle$ $\mathcal{M}_0.M_0^L$.

Since we have $\mathcal{M}.M_0^L[\sigma_1(G)\rangle\mathcal{M}_{i-1}.M_0^L$ and $\mathcal{T}(\sigma_1)$, we apply Lemma 3 to deduct that $\mathcal{M}.M_0^R$ $[\sigma_1(G')\rangle \ \mathcal{M}_{i-1}.M_0^R$. Similarly to σ_3.

Let us consider σ_2. Using Eq. (6.1), if $\mathcal{M}_{i-1}.M_0^L[t_i\rangle\mathcal{M}_i.M_1^L$, then $\mathcal{M}_{i-1}.M_0^R[t_{in}\rangle\mathcal{M}_i.M_1^R$, $\forall t_{in} \in I_R$. On the other hand, the firing of any fireable sequence $t'_1 \ldots t'_m$ at $\mathcal{M}_i.M_1^R$ in G' (where $t'_{k\in\{1,...,m\}}$ is neither input nor output node) does not affect the marking of preserved places, thus $\mathcal{M}_i.M_0^R[t_{in}\rangle\mathcal{M}_i.M_1^R[t'_1\rangle \ldots \ \mathcal{M}_{m-1}.M_m^R[t_m\rangle \ \mathcal{M}_i.M_m^R[t_{out}\rangle$, such that t_{out} is an output node. Based on Eq. (6.1), we deduct that $\mathcal{M}_i.M_m^R[t_{out}\rangle\mathcal{M}_j.M_0^R$. □

Lemma 5. *If G is reversible, then the initial marking of G' is reachable from any marking $M \in RM_2(G')$.*

Example. In Fig. 6.5b, initial marking $(3,0,0)$ of C_1 is reachable from each marking in $RM_2(C_1) = \{(0,2,1),(0,1,2),(0,0,3),(1,1,1),(1,0,2),(2,0,1),\ (2,1,0),(1,2,0),(0,3,0)\}$.

Proof. Markings in $RM_2(G')$ have the form $\mathcal{M}.M^R$. Due to the reversibility property of PPNs, for each reachable marking $\mathcal{M}.M^R$ there exists σ where $\mathcal{M}.M^R[\sigma(G')\rangle\mathcal{M}'.M_0^R$. By Lemma 4, it is proved that initial marking of G' is reachable from any marking $\mathcal{M}.M_0^R$. □

***Theorem* 6.1: Reversibility**

If GSPN G' is obtained by applying r to reversible GSPN G, then G' is reversible.

Proof. Previously, it is proved that the initial marking of G' is reachable from any marking in $RM_1(G')$ and $RM_2(G')$. Hence G' is reversible since its reachable marking set $RM(G')$ is the union of subsets $RM_1(G')$ and $RM_2(G')$. □

Remarque. Note that preserving reversibility property means that any reachable configuration of a given RecGSPN N is reversible. However, RecGSPN N may become irreversible to an initial marking of an initial configuration. As well, we note that if a given RecGSPN N can infinitely often return to a reversible initial configuration, then RecGSPN N is reversible.

***Theorem* 6.2: Home state**

If GSPN G' is obtained by applying r to GSPN G having a home state, then G' has a home state.

Proof. If M_h is a home state of G, then $\forall\ M \in RM(G),\ \exists\ \sigma,\ M[\sigma(G)\rangle M_h$. Based on reversibility proof, we deduct that G' has a home state. □

***Theorem* 6.3: Boundedness**

If GSPN G' is obtained by applying r to bounded GSPN G, then G' is bounded.

Proof. According to PPNs behavior, places in R are bounded. Preserved places that preserve their preset and postset are bounded. According to Eq. 6.1, each preserved place that does not preserve its preset and/or postset is connected to images of input and/or output transitions. For each input transition t_{in} and output transition t_{out} in R, we have (i) t_{in} in G' consumes the same number of tokens consumed by $t \in f(I_L)$ in G and produces in preserved places of G' the same number of tokens produced by t in G, and (ii) t_{out} in G' produces the same number of tokens produced by $t \in f(O_L)$ in G and consumes from preserved places of G' the same number of tokens consumed by t in G. Hence, places in G' are bounded. □

Remarque. Note that in the case of infinite-structure RecGSPNs, the preservation of boundedness property means that each place remains bounded, however, the reachability graph may become infinite. In fact, if the application of transformation rules infinitely keeps adding new places to the net, then the reachability graph keeps growing, as well.

***Theorem* 6.4: Liveness**

If GSPN G' is obtained by applying r to live GSPN G, then G' is live.

Proof. If G is live, then $\forall\ t \in T,\ M \in RM(G),\ \exists\ \sigma,\ M[\sigma(G)\rangle M'[t\rangle$. Based on reversibility proof, we deduct that G' is live. □

***Theorem* 6.5: Deadlock-free**

If GSPN G' is obtained by applying r to deadlock-free GSPN G, then G' is deadlock-free.

Proof. If G is deadlock-free, then $\forall\ M \in RM(G),\ \exists\ t \in T$, such that $M[t\rangle M'$. Based on previous proofs, we deduct that G' is deadlock-free. □

6.3.2 Preservation of linear temporal properties

In the following, we describe a subset of LTL formulae that are still satisfied after a reconfiguration, if these formulae fulfill specific conditions. Four cases are discussed in the following: (i) preservation of propositional formulae satisfied, for a state, in the source GSPN, (ii) preservation of "Until" satisfied for a path in the source GSPN, (iii) preservation of LTL formulae satisfied for a path in the source GSPN, (iv) and preservation of LTL formulae satisfied for a state in the source GSPN. These preservations require that the formulae involve only specific places, called "*Strongly Preserved places*" (SP-places). This section starts by presenting the concept of strongly preserved nodes (i.e., places or transitions), before detailing properties preservation theorems and their proofs.

Definition 6.5. (Strongly preserved nodes). Let G and G' be source and target GSPNs of rule r, respectively. Sets $\widehat{\mathcal{P}}$ and $\widehat{\mathcal{T}}$ containing preserved places and transitions having preserved presets and postsets, are called *Strongly Preserved* places (SP-places) and transitions, respectively.

Example. Considering GSPN H' in Fig. 6.6b which is obtained from H (Fig. 6.6a) by replacing transition $\texttt{need}_2$ by subnet in dashed box ($\texttt{start}_2$, $\texttt{work}_2$, and $\texttt{need}'_2$). Places $\texttt{idle}_2$ and $\texttt{req}_2$ are not strongly preserved since their preset and/or presets are not preserved. On the other hand, places $\texttt{CS}_1$, $\texttt{CS}_2$, $\texttt{wait}_1$, $\texttt{wait}_2$, ... are strongly preserved.

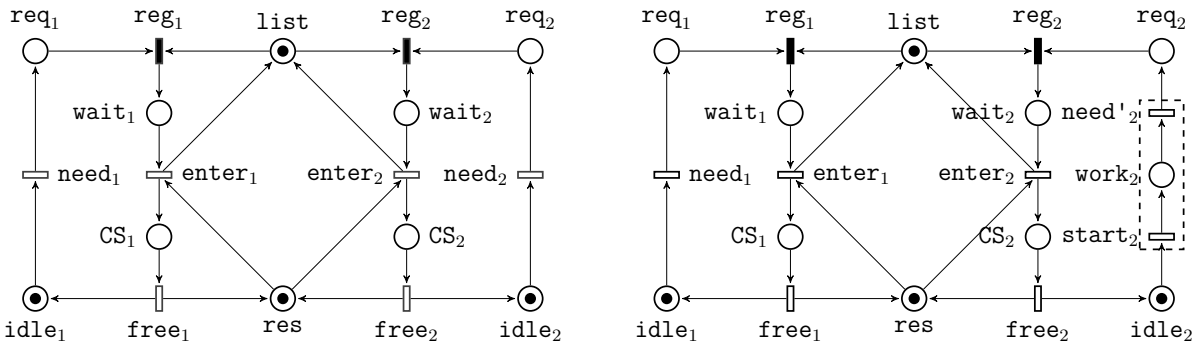
Definition 6.6. (Prompt of a path). In transition system TS, an infinite transition sequence $\sigma = t_0\ t_1 \ldots$ is called *prompt* of a maximal path $\pi = s_0\ s_1 \ldots$, if $\forall\ i, (s_i, t_i, s_{i+1}) \in E$.

Example. GSPN H in Fig. 6.6a models two processes $\texttt{p}_1$ and $\texttt{p}_2$ trying access a critical resource. A token in $\texttt{idle}_i$, $\texttt{req}_i$, $\texttt{wait}_i$ and $\texttt{CS}_i$ means that process $\texttt{p}_i$ is idle, requesting access, waiting to access a critical resource, or in a critical section, respectively. Transitions $\texttt{need}_i$, $\texttt{req}_i$, $\texttt{reg}_i$, $\texttt{enter}_i$, and $\texttt{free}_i$ are fired when process $\texttt{p}_i$ needs a critical resource, requests it, starts waiting for it, enters a critical section, or frees a critical resource, respectively. Path $\pi_0 = s_0 s_1 s_3 s_6 s_0 \ldots$ in Fig. 6.7a has prompt $\sigma_0 = \texttt{need}_2\ \texttt{reg}_2\ \texttt{enter}_2\ \texttt{free}_2 \ldots$.

Definition 6.7. (Projection of a sequence). By Lemma 3, for each fireable sequence σ in G, there exist fireable sequence σ' in G', called *projection* of σ, such that obsolete transitions are removed and sequence of fresh ones are added, and vice versa.

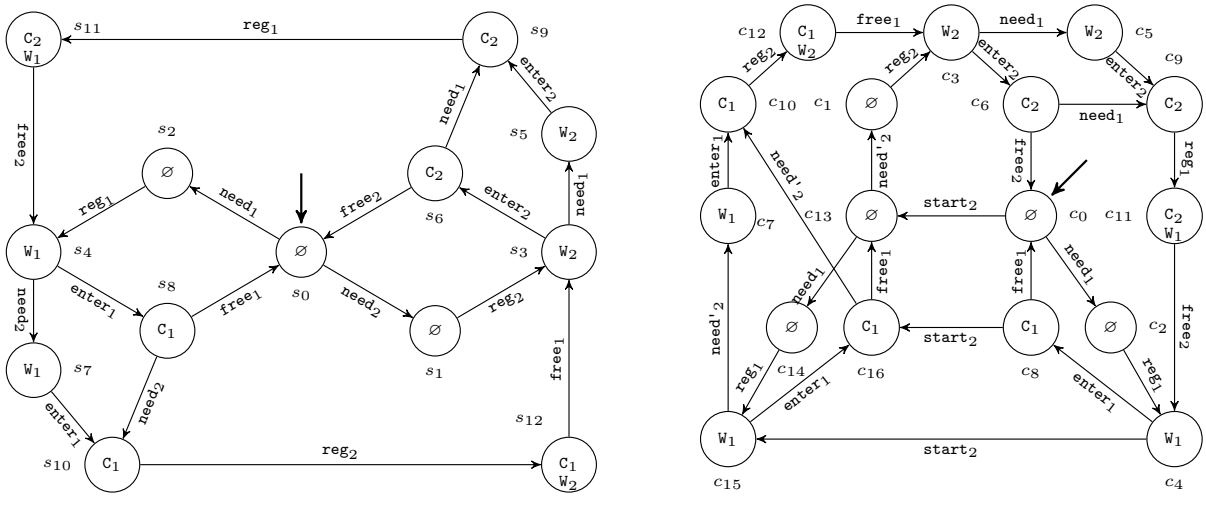
Example. For instance, $\sigma'_0 = \texttt{start}_2\ \texttt{need'}_2\ \texttt{reg}_2\ \texttt{enter}_2\ \texttt{free}_2$ starting from state c_0 in Fig. 6.7b is obtained from $\sigma_0 = \texttt{need}_2\ \texttt{reg}_2\ \texttt{enter}_2\ \texttt{free}_2$ starting from state s_0 in Fig. 6.7a by removing occurrences of obsolete $\texttt{need}_2$ in σ_0 and adding $\texttt{start}_2\ \texttt{need'}_2$ (which are fresh).

Definition 6.8. (Projection of a maximal path). Let G be a GSPN and G' the GSPN obtained by reconfiguring G, let $TS = \langle S, T, E, I, \mathcal{A}, \mathcal{L}\rangle$ and $TS' = \langle S', T', E', I', \mathcal{A}', \mathcal{L}'\rangle$ be transition systems with respect to G and G'. A maximal path $\tau = c_0 c_1 \ldots$ of TS' having trace σ' is called a *projection* of a maximal path $\pi = s_0 s_1 \ldots$ of TS having trace σ, if σ' is a projection of σ and $c_0(p) = s_0(p), \forall\ p \in \mathcal{P}$.



(a) GSPN H models mutual exclusion. (b) GSPN H' models mutual exclusion.

Figure 6.6: Mutual exclusion.



(a) Transition system of H. (b) Transition system of H'.

Figure 6.7: Transition systems of GSPNs H and H', where $\texttt{C}_i$ and $\texttt{W}_1$ denote $(\texttt{CS}_i = 1)$ and $(\texttt{wait}_i = 1)$, for each $i \in \{1, 2\}$, respectively.

Example. In Fig. 6.7b, $\tau_0 = c_0c_{13}c_1c_3c_6c_0$ is a projection of $\pi_0 = s_0s_1s_3s_6s_0$ in Fig. 6.7a.

Next, we show that if a propositional formula involving only SP-places is satisfied for a state in the source GSPN, then it is satisfied in the target GSPN for some state.

Lemma 6. *Let Φ be a propositional formula involving only strongly preserved places. If there is a state s in TS such that Φ holds for s, then for all states c in TS' such that $c(p) = s(p)$, $\forall p \in \widehat{\mathcal{P}}$ (i.e., SP-places), Φ holds for c.*

Example. Assume set of atomic propositions $\widehat{\mathcal{A}}_0 = \{\mathtt{C}_1, \mathtt{C}_2, \mathtt{W}_1, \mathtt{W}_2\}$, where $\mathtt{C}_i$ and $\mathtt{W}_i$ denote $(\mathtt{CS}_i = 1)$ and $(\mathtt{wait}_i = 1)$, for $i \in \{1, 2\}$, respectively. Atomic proposition $\mathtt{W}_2$ holds for state s_3 as well for states c_3 and c_{15}. This is because $\mathtt{wait}_2$ is a SP-place in both nets H and H'.

Proof. Let $\widehat{\mathcal{A}}$ be a set of atomic propositions involving only SP-places. Let s and c be states two belonging to transition systems TS and TS', respectively, such that $s(p) = c(p), \forall\ p \in \widehat{\mathcal{P}}$. Then, each atomic proposition a in $\widehat{\mathcal{A}}$ that holds for s, holds for c, and vice-versa, since each SP-place has identical marking at both states s and c. Therefore, if s satisfies a propositional formula Φ, then so does c. □

In the rest of this section, atomic propositions and propositional formulae involve only strongly preserved places. Example in Fig. 6.7 is still used to clarify each theorem and lemma.

In the following, we consider the satisfaction preservation of formulae of the from $\Phi_1\mathbf{U}\Phi_2$, where Φ_1 and Φ_2 are two propositional formulae, in the target GSPN for some path.

Lemma 7. *Let Φ_1 and Φ_2 be propositional formulae. If $\Phi_1\mathbf{U}\Phi_2$ holds for a path π, then it holds for its projection.*

Proof. Let Φ_1 and Φ_2 be two propositional formulae. Let $\pi = s_0s_1\ldots$ be a path and $\tau = c_0c_1\ldots$ be its projection. Let s_n be the state in which Φ_2 holds and Φ_1 holds for each state s_i, for all $i \in \{0, \ldots, n-1\}$. Path τ is a projection of π if its prompt σ' is a projection of prompt σ of π and $s_0(p) = c_0(p), \forall\ p \in \mathcal{P}$ (i.e., preserved places). Hence, if Φ_1 holds for s_0 then it holds for c_0. If states $s_1, s_2, \ldots, s_i$ are reached from s_0 by firing strongly preserved transition sequence $\widehat{\sigma}$, then states $c_1, c_2, \ldots, c_i$ are reached from c_0 by firing $\widehat{\sigma}$ too, such that $s_j(p) = c_j(p), \forall\ p \in \mathcal{P}, \forall\ j \in \{1, \ldots, i\}$ (see Lemma 3). Thus, if Φ_1 holds for $s_1, s_2, \ldots, s_i$, then it holds for $c_1, c_2, \ldots, c_i$. If $s_{i+1}s_{i+2}\ldots s_{k-1}$ is a state sequence reached from s_i by firing non-strongly preserved transition sequence and s_k is reached from s_{k-1} by firing a strongly preserved transition t, then, we have the following in the target GSPN.

- $c_{i+1}c_{i+2}\ldots c_{l-1}c_l$ is a state sequence, where $c_{i+1}, \ldots, c_{l-1}$ is reached from c_i by firing non-strongly preserved transition sequence and c_l is reached from c_{l-1} by firing a strongly preserved transition t, where $s_k(p) = c_l(p), \forall\ p \in \widehat{\mathcal{P}}$ (see Lemma 3).
- and Φ_1 holds for states $c_{i+1}, \ldots, c_{l-1}$.

As a result, if propositional formula Φ_1 holds for state sequence $s_0\ s_1 \dots s_{n-1}$ having prompt ϱ, then it holds for state sequence $c_0\ c_1 \dots c_{m-1}$ having prompt ϱ', where ϱ' is a projection of ϱ. By Lemma 3, if s_n is reached from s_0 by firing transition sequence $\varsigma = t_0\ t_1 \dots t_n$ in G, where t_n is a strongly preserved transition, then c_m is reached from c_0 by firing transition sequence $\varsigma' = t'_0\ t'_1 \dots t'_m$ in G', where $\forall\ p \in \mathcal{P}, c_0(p) = s_0(p), c_m(p) = s_n(p)$, $t_n = t'_m$, and ς' is a projection of ς. As a recap, if $\Phi_1 \mathbf{U} \Phi_2$ holds for a path π, then $\Phi_1 \mathbf{U} \Phi_2$ holds for a path τ, such that τ is a projection of π. Note that t_n must be a strongly preserved transition, otherwise, its firing does not affect the truth value of Φ_2. □

Example. In Fig. 6.7, path $\pi_0 = s_0 s_1 s_3 s_6 s_0 \dots$ and its projection $\tau_0 = c_0 c_{13} c_1 c_3 c_6 c_0 \dots$ have traces $\{\varnothing\}\{\varnothing\}\{\mathtt{W}_2\}\{\mathtt{C}_2\}\{\varnothing\} \dots$ and $\{\varnothing\}\{\varnothing\}\{\varnothing\}\{\mathtt{W}_2\}\{\mathtt{C}_2\}\{\varnothing\} \dots$, respectively. Considering formula ($\mathbf{true}\ \mathbf{U}\ \mathtt{C}_2$), it holds for both π_0 and τ_0.

In the rest of this chapter, we exclude operator $\mathbf{X}$ (i.e., next) from the definition of LTL formulae. By following Theorems 6.6, 6.7 and 6.8, we show that if a formula is satisfied for a path, a state, and transition system TS derived from the source GSPN, then it is satisfied for some path, some state, and transition TS' derived from the target GSPN, respectively.

***Theorem* 6.6: Paths**

Let Φ be an LTL formula inductively defined as follows

$$\Phi ::= \text{true} \mid a \mid \neg\Phi \mid \Phi_1 \vee \Phi_2 \mid \Phi_1\ \mathbf{U}\ \Phi_2$$

where $a \in \widehat{\mathcal{A}}$. If Φ holds for a path π, then Φ holds for its projection τ.

Proof. Let Φ_1, Φ_2 and Φ_3 be propositional formulae. Previously, it is proved that if Φ_1 involving only strongly preserved places holds for a path π of GSPN G, then it holds for path τ of GSPN G', where τ is a projection of π. Obviously, if $\neg\Phi_1$ (as well as $\Phi_1 \vee \Phi_2$) holds for π, then it holds for τ. Besides, it is proved that if $\Phi_1 \mathbf{U} \Phi_2$ holds for path π, then it holds for τ. Obviously, if $\Phi_1 \mathbf{U} \Phi_2 \mathbf{U} \Phi_3$ holds for path π, it holds for τ. Therefore, if Φ is inductively defined by $\Phi ::= \text{true} \mid a \mid \neg\Phi \mid \Phi_1 \vee \Phi_2 \mid \Phi_1\ \mathbf{U}\ \Phi_2$, where $a \in \widehat{\mathcal{A}}$ is an atomic proposition involving only strong preserved places, and Φ holds for a path π, then Φ holds for τ. Furthermore, if $\mathbf{G}\Phi$ (as well as $\mathbf{F}\Phi$) holds for π, then it holds for τ. □

Example. Considering the previous example again, where π_0 is a path in source GSPN and its projection is τ_0. Formula ($\mathbf{GF}\ \mathtt{W}_2$) $\rightarrow$ ($\mathbf{GF}\ \mathtt{C}_2$) (starvation freedom) holds for both π_0 and τ_0. Note that Boolean connectives such as $\rightarrow$ can be derived from $\neg$ and $\vee$.

***Theorem* 6.7: States**

Let Φ be an LTL formula. If Φ holds for a state s of G, then Φ holds for state c of G', such that $s(p) = c(p), \forall\ p \in \mathcal{P}$.

Proof. Recall that a formula holds for a state iff it holds for all paths starting from this state [BK08].

Let s_0 and c_0 be two states of G and G', respectively, where $s_0(p) = c_0(p), \forall\ p \in \mathcal{P}$ (i.e., preserved places). If Φ is a propositional formula that holds for s_0, then it was proved that Φ holds for c_0.

Let Φ_1 and Φ_2 be propositional formulae, where $\Phi_1\ \mathbf{U}\ \Phi_2$ holds for s_0. Let $\tau = c_0c_1\ldots$ be a path starting from c_0 of TS'. If c_1 is reached from c_0 by firing a strongly-preserved transition t, then there exists a path $\pi = s_0s_1\ldots$ of TS where s_1 is reached from s_0 by firing t and $s_1(p) = c_1(p), \forall\ p \in \widehat{\mathcal{P}}$, and vice versa. Thus, the truth value of Φ_1 and Φ_2 are preserved. If c_1 is reached from c_0 by firing a non-strongly-preserved transition t', then the truth value of Φ_1 and Φ_2 are preserved (since firing non-strongly preserved transition does not affect the truth value of propositional formulae involving only strongly preserved places), and vice versa.

That is, if there exists a path $\tau = c_0c_1\ldots$ of TS', where Φ_1 does not hold for c_k and Φ_2 does not hold for each state c_i, for all $i \in \{0,\ldots,k-1\}$, then there exists a path of TS, where $\Phi_1\ \mathbf{U}\ \Phi_2$ does not hold. Such a path does not exist since $\Phi_1\ \mathbf{U}\ \Phi_2$ holds for s_0.

Hence, if there exists a path starting from s_0 where $\Phi_1\ \mathbf{U}\ \Phi_2$ does not hold, then it means that a state for which Φ_2 holds can be never reached. Such a path is of the form $\tau = c_0c_1\ldots c_{n-1}(c_n\ldots c_m)^\omega$ (ω denotes infinite repetition) such that Φ_2 does not hold for each state c_i, for all $i \in \{0,\ldots,m\}$.

By Lemma 3, if c_n is reached from c_m by firing a preserved transition t (interface transition t', respectively), then there exists a path $\pi = s_0s_1\ldots s_{i-1}(s_i\ldots s_j)^\omega$ of TS, such that s_i is reached from s_j by firing t (t'', respectively) and Φ_2 does not hold for each state s_l, for all $l \in \{0,\ldots,j\}$. Such a path does not exist since $\Phi_1\ \mathbf{U}\ \Phi_2$ holds for s_0.

On the other hand, c_n cannot be reached from c_m by firing an inner transition t of right-hand side (see definition of *properties preserving nets*). As a result, if $\Phi_1\ \mathbf{U}\ \Phi_2$ holds for s_0, then it holds for c_0. The rest of proof can be continued as in the proof of Theorem 6.6. □

Example. Formula $((\mathbf{GF}\ \mathtt{W}_1) \rightarrow (\mathbf{GF}\ \mathtt{C}_1) \wedge (\mathbf{GF}\ \mathtt{W}_1) \rightarrow (\mathbf{GF}\ \mathtt{C}_1))$ holds for both states c_0 and s_0 shown in Figs. 6.7a and 6.7b.

***Theorem* 6.8: Transition System**

Let Φ be an LTL formula. If Φ holds for a state s of G, then Φ holds for transition system TS' having initial state c_0 with respect to G', such that $s(p) = c_0(p), \forall\ p \in \mathcal{P}$.

Proof. Recall that a formula holds for a transition system TS iff it holds for all its initial states[BK08]. We have already proved that if Φ holds for s then it holds for c_0, such that $s(p) = c_0(p), \forall\ p \in \mathcal{P}$. □

6.4 Quantitative Analysis

In this section, quantitative analysis of RecGSPNs is discussed. Similarly to GSPNs, the proposed quantitative analysis uses a semi-Markov chain that describes the stochastic behavior of the reconfigurable system. Thus, we propose a method that builds a semi-Markov chain for a given RecGSPN $\Psi = \langle G^0, \mathcal{R} \rangle$, such that the number of configurations is finite.

Let $s^i_j = (G^i, M_j)$ be the state isomorphic to marking M_j of GSPN G^i obtained by applying a rule sequence to G^0. Let $\mathcal{S}$ be the set of states in semi-Markov chain, and $\mathcal{E}$ be its set of arrows.

The semi-Markov chain for a given RecGSPN Ψ is computed as follows.

1. Insert into $\mathcal{S}$ state $s^0_0 = (G^0, M_0)$ isomorphic to initial marking M_0 of initial configuration G_0 of RecGSPN Ψ.

2. Apply or fire each applicable rule or enabled immediate transition at s^0_0. For each obtained state s', add s' in $\mathcal{S}$ and an arrow (s^0_0, a, s') in $\mathcal{E}$.

3. If there is neither an applicable rule nor an enabled immediate transition, then fire each enabled timed transition in s^0_0 and insert the obtained states and arrows, similarly to the previous step.

4. Repeat Steps (2) and (3) on each newly obtained state. Otherwise, semi-Markov chain is completely computed.

The detailed algorithm is described in Algorithm 1.

We apply this algorithm to Ψ_0 of Fig. 6.3. The obtained semi-Markov chain is shown in Fig. 6.8 and Table 6.1 lists its state space.

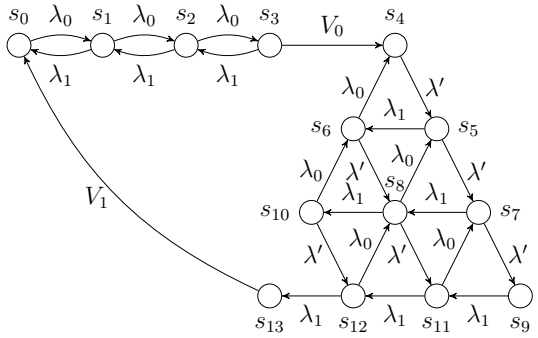


Figure 6.8: semi-Markov chain of Ψ_0, where λ_0, λ_1 and λ' are the firing rates of t_0, t_1 and t', respectively; and V_0 and V_1 are application weights of r_0 and r_1, respectively.

According to Algorithm 1, the first step consists of creating a state s_0 that represents initial marking M_0 of initial configuration C_0. In s_0 rules and immediate transitions are

Algorithm 1 Computing semi-Markov chain for a given RecGSPN

Require: $\Psi = \langle G^0, \mathcal{R} \rangle$: RecGSPN
Ensure: $\mathcal{C} = (\mathcal{S}, \mathcal{E})$: semi-Markov chain
Step 1 - Initialization:
Create state $s = (G^0, M_0)$
Mark s as non-visited
$\mathcal{S} \longleftarrow \{s\}$
$\mathcal{E} \longleftarrow \emptyset$
Step 2 - Applying rules and firing immediate transitions:
for all NV $s \in \mathcal{S}$, $A(s) \cup EI(s) \neq \emptyset$ **do**
  Mark $s = (G, M)$ as visited
  for all AR r to G at M **do**
    Apply r to obtain G' with marking M'
    if $s' = (G', M') \notin \mathcal{S}$ **then**
      $\mathcal{S} \longleftarrow \mathcal{S} \cup \{s'\}$
    end if
    if $(s, r, s') \notin \mathcal{E}$ **then**
      $\mathcal{E} \longleftarrow \mathcal{E} \cup \{(s, r, s')\}$
    end if
  end for
  for all EI t in G at M **do**
    Fire t to obtain new marking M'
    if $s' = (G, M') \notin \mathcal{S}$ **then**
      $\mathcal{S} \longleftarrow \mathcal{S} \cup \{s'\}$
    end if
    if $(s, t, s') \notin \mathcal{E}$ **then**
      $\mathcal{E} \longleftarrow \mathcal{E} \cup \{(s, t, s')\}$
    end if
  end for
end for
Step 3 - Firing timed transitions:
for all NV $s = (G, M) \in \mathcal{S}$, $ET(s) \neq \emptyset$ **do**
  Mark s as visited
  for all ET t in G at M **do**
    Fire t to obtain new marking M'
    if $s' = (G, M') \notin \mathcal{S}$ **then**
      $\mathcal{S} \longleftarrow \mathcal{S} \cup \{s'\}$
    end if
    if $(s, t, s') \notin \mathcal{E}$ **then**
      $\mathcal{E} \longleftarrow \mathcal{E} \cup \{(s, t, s')\}$
    end if
  end for
end for
if $\exists\ s \in \mathcal{S}$, s is NV and $A(s) \cup EI(s) \neq \emptyset$ **then**
  goto ***Step 2***
else
  if $\exists\ s \in \mathcal{S}$, s is NV and $ET(s) \neq \emptyset$ **then**
    goto ***Step 3***
  else
    semi-Markov chain is completely computed
  end if
end if

NV, AR, ET and EI stand for: non-visited, applicable rule, enabled timed transition, and enabled immediate transition, respectively.

not enabled, and hence we jump to **Step 3** in which t_0 is fired and obtained marking M_1 is represented by s_1, and so on. Once Ψ_0 reaches state s_{13}, rule r_1 becomes applicable, by which Ψ_0 returns to its initial configuration C_0 (transition from s_{13} to s_0 models r_1 application).

State	Config.	Marking	State	Config.	Marking
s_0	C_0	$(0,3)$	s_1	C_0	$(1,2)$
s_2	C_0	$(2,1)$	s_3	C_0	$(3,0)$
s_4	C_1	$(3,0,0)$	s_5	C_1	$(2,0,1)$
s_6	C_1	$(2,1,0)$	s_7	C_1	$(1,0,2)$
s_8	C_1	$(1,1,1)$	s_9	C_1	$(0,0,3)$
s_{10}	C_1	$(1,2,0)$	s_{11}	C_1	$(0,1,2)$
s_{12}	C_1	$(0,2,1)$	s_{13}	C_1	$(0,3,0)$

Table 6.1: State space of semi-Markov chain depicted in Fig. 6.8.

Given state probability distribution $\kappa_0, \ldots, \kappa_m$ of states $s_0, \ldots, s_m$, respectively, we can compute the following.

- **Probability of having n tokens in place p_i [Mar+94]:** Let $B(p_i, n)$ be the subset of $\mathcal{S}$ in which the number of tokens in a place p_i is n. Then, the probability of having n tokens in place p_i is given by:

$$P[B(p_i, n)] = \sum_{s_j \in B(p_i, n)} \kappa_j. \tag{6.3}$$

- **Mean number of tokens [Mar+94]:** The average number of tokens in p_i is given by:

$$\mu_i = \sum_{n=1}^{\infty}(nP[B(\kappa_i, n)]). \tag{6.4}$$

- **Probability of firing transition t_j [Mar+94]:** Let EN_j be the subset of $\mathcal{S}$ in which transition t_j is enabled. Then, the probability r_j that t_j fires next is given by:

$$r_j = \sum_{s_i \in EN_j} \kappa_i P[t_j \text{ fires first at } s_i]. \tag{6.5}$$

- **Throughput at transition t_j [Mar+94]:** Throughput d_j at t_j is given by

$$d_j = \sum_{s_i \in EN_j} \pi_i \lambda_j. \tag{6.6}$$

6.5 Using RecGSPNs in Practice

In this section, we show the usage of RecGSPNs to model/verify the behavior of a running example, introduced below. The results described in this section have been obtained by using our prototypal implementation. The web page of the tool contains a technical report explaining how the tool can be used.

In this section, we describe a Reconfigurable Manufacturing System (RMS) used to illustrate and evaluate our modeling formalism. The considered RMS is composed of four machines (M_1, M_2, M_3 and M_4), and a central buffer with a capacity of 50 raw product pieces. In this RMS, an activated machine can: (i) load raw material from the central buffer when it is idle, (ii) process raw material, and (iii) unload worked product. The four machines produce the same type of product.

The RMS has three modes: high, middle, and low production mode. In high production mode (HM), machines M_1, M_2 and M_4 are activated. As for middle production mode (MM), machines M_1, M_2 and M_3 are activated. Whereas in low production mode (LM), only machines M_1 and M_2 are activated. M_1 is the fastest machine in this RMS, however, it fails when it produces 50 products without enough rest, and α the cost of its repairing is highly expensive. Therefore, M_3 and M_4 are activated, mutually exclusive, so that M_1 can get enough rest. M_3 works, with cost β, if the number of raw materials in the central buffer exceeds threshold s, whereas M_4 works, with cost θ, when M_1 has produced n products without rest.

The initial configuration of this RMS is LM modeled by GSPN G^0 depicted in Fig. 6.9. The interpretation of places and transitions of G^0 is given as follows.

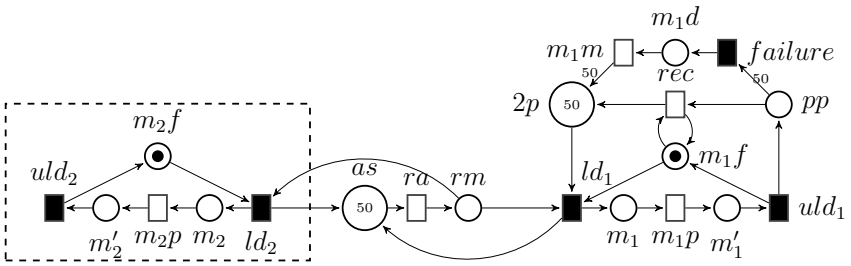
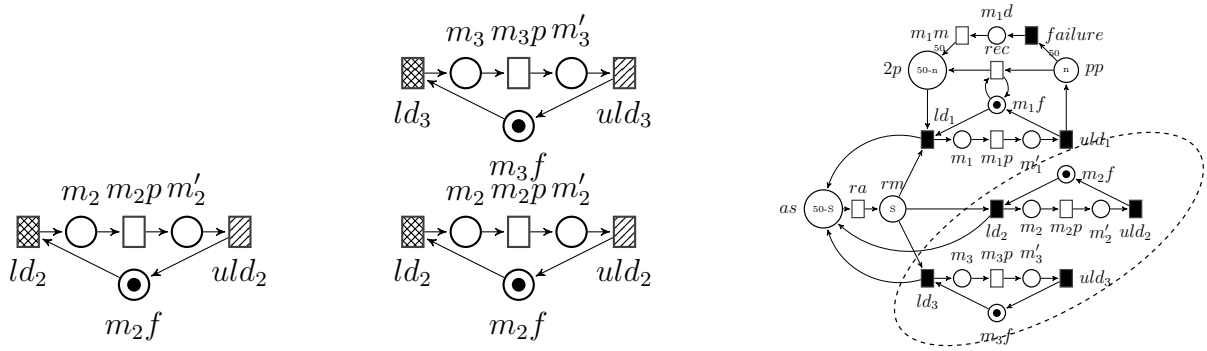


Figure 6.9: GSPN model of LM.

1. as: Its marking models the number of free spaces in the central buffer.
2. rm: Its marking models the number of raw materials in the central buffer.
3. m_i: A token in m_i means that machine M_i has begun processing.
4. m_i': A token in m_i' means that machine M_i has finished processing.
5. m_if: A token in m_if means that machine M_i is idle.
6. m_1d: A token in m_1d means that machine M_1 is down.
7. pp: The number of tokens inside pp represents the number of products produced by M_1 without rest.
8. $2p$: The number of tokens in $2p$ represents the number of products that can be produced by M_1 before it fails, if it does not get enough rest.
9. ra ($\Lambda(ra) = 5$): Raw material is loaded in the central buffer.
10. ld_i ($W(ld_i) = 1$): M_i loads an item from the central buffer.
11. m_ip ($\Lambda(m_1p) = 5$ and $\Lambda(m_2p) = 1$): Machine M_i is processing.
12. uld_i ($W(uld_i) = 1$): M_i unloads a product.
13. $failure$ ($W(failure) = 1$): M_1 fails.
14. m_1m ($\Lambda(m_1m) = 0.2$): M_1 is under repair.
15. rec ($\Lambda(rec) = 0.5$): M_1 has got enough rest.

First, we reconfigure G^0 to G^1 shown in Fig. 6.10c which models the RMS in its middle production mode (MM). G^1 is obtained by applying rule $r_1 = \langle L_1, R_1, \mathrm{NAC}_1, I_1, O_1, V_1, mark_1 \rangle$ (see Fig. 6.10), to G^0 as follows. First, an occurrence g, subnet in dashed box in Fig. 6.9, of its LHS L_1 depicted in Fig. 6.10a is mapped by a morphism f_1, where $\mathrm{NAC}_1 = \emptyset$, input

(output, respectively) nodes of L_1 and R_1 are depicted by ▩ (▨, respectively), $V_1 = 1$ and $mark_1(rm) = s$. Then, g is substituted by RHS R_1 depicted in Fig. 6.10b. The interpretation of new added places and transitions m_3, ld_3, m_3p, etc. is analog to the meaning of m_1, ld_1, m_1p, etc. respectively, where $W(ld_3) = 1$, $\Lambda(m_3p) = 2$ and $W(uld_3) = 1$.



(a) L_1 (R_2) (b) R_1 (L_2) (c) G^1: GSPN model of MM.

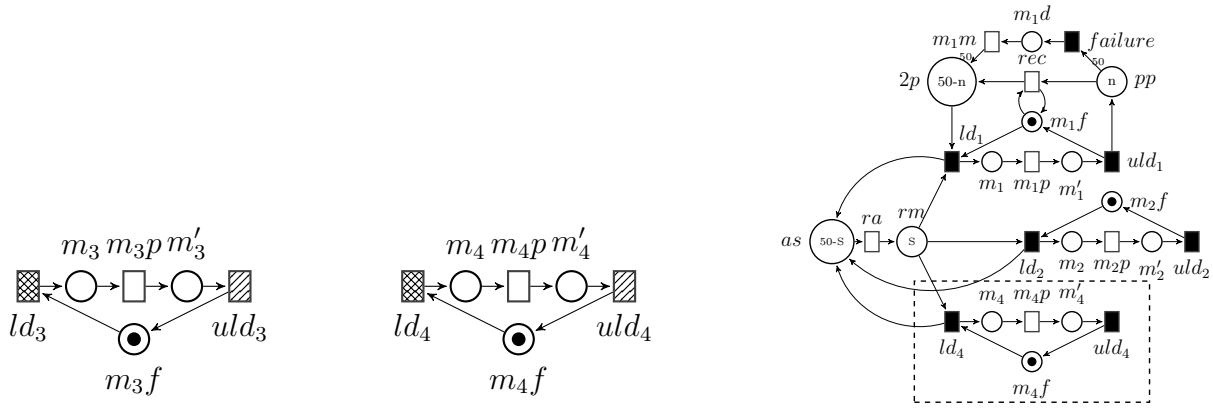
Figure 6.10: Left and right-hand sides of r_1.

Once there are $(55 - s)$ available spaces in the central buffer and M_3 is idle, M_3 is deactivated. This reconfiguration is obtained by applying rule $r_2 = \langle L_2, R_2, \text{NAC}_2, I_2, O_2, V_2, mark_2 \rangle$ to G^1 (Fig. 6.10c) and the resulting GSPN is G^0, where L_2 and R_2 are shown in Figs. 6.10b and 6.10a, respectively, occurrence g' of L_2 in G^1 is subnet in dashed ellipse in Fig. 6.10c, $\text{NAC}_2 = \emptyset$, input (output, respectively) nodes of L_2 and R_2 are depicted by ▩ (▨, respectively), $V_2 = 1$ and $mark_2(as, m_3f) = (55 - s, 1)$.

In the following, we consider RMS HM. This production mode is activated when the current configuration is G^1 and M_1 has produced n products without enough rest. This reconfiguration is modeled by G^2 depicted in Fig. 6.11c. G^2 is obtained by applying rule $r_3 = \langle L_3, R_3, \text{NAC}_3, I_3, O_3, V_3, mark_3 \rangle$ to G^1, which models M_4 activation and M_3 deactivation, where L_3 and R_3 are shown in Figs. 6.11a and 6.11b, respectively, occurrence of L_3 in G^1 is the bottom-most subnet in dashed ellipse in Fig. 6.10c, $\text{NAC}_3 = \emptyset$, input (output, respectively) nodes of L_3 and R_3 are depicted by ▩ (▨, respectively), $V_3 = 1$ and $mark_3(m_3f, pp) = (1, n)$.

The interpretation of new added places and transitions m_4, ld_4, m_4p, etc. are analog to the meaning of m_1, ld_1, m_1p, etc. respectively, where $W(ld_4) = 1$, $\Lambda(m_4p) = 3$ and $W(uld_4) = 1$.

Once M_1 has got enough rest (modeled by $(55-n)$ tokens at $2p$) and M_4 is idle, M_4 is deactivated. This reconfiguration is obtained by applying $r_4 = \langle L_4, R_4, \text{NAC}_4, I_4, O_4, V_4, mark_4 \rangle$ to G^2 and the resulting GSPN is G^1, where L_4 and R_4 are shown in Figs. 6.11b and 6.11a, respectively, occurrence of L_4 in G^2 is the subnet in dashed box in Fig. 6.11c, $\text{NAC}_4 = \emptyset$, input (output, respectively) nodes of L_4 and R_4 are depicted by ▩ (▨, respectively), $V_4 = 1$ and $mark_4(m_4f, 2p) = (1, 55 - n)$.



(a) L_3 (R_4) (b) R_3 (L_4) (c) G^2: GSPN model of HM.

Figure 6.11: Left and right-hand sides of r_3.

Considering performance evaluation, we studied the values of thresholds s and n that minimize $\mathcal{C}$ the total cost of repairing machine M_1 and the utilization of machines M_3 and M_4. We measured this cost under different values of s and n such that $s, n \in \{5, 10, \ldots, 45\}$.

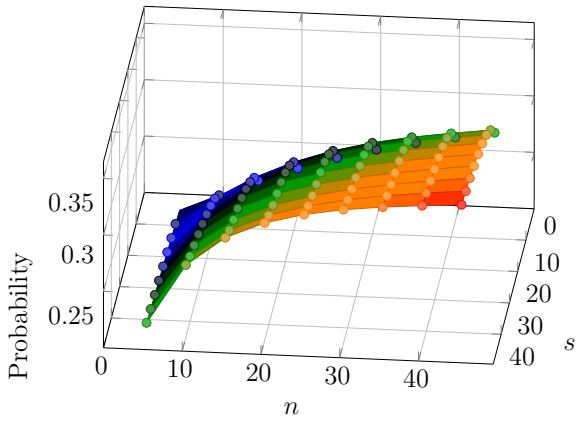
We have developed a basic tool that has as inputs a GSPN model and a set of rules. This tool is used to simulate the system described above. Given the different values of s and n, many semi-Markov chains are generated. The probabilities that M_1 is down (probability that place m_1d contains a token), M_3 is working (probability that place m_3 contains a token) and M_4 is working (probability that place m_4 contains a token) are computed according to Eq. 6.3 and shown in Fig. 6.12.

As depicted in Fig.6.12b, at couple $(s, n) = (5, 45)$ the probability that M_3 is working is the highest. Indeed, M_3 is deactivated when the number of raw materials is less than $55 - s$ or machine M_1 has produced n products without enough rest, which means that increasing s and decreasing n decreases the probability that M_3 is working, as well.

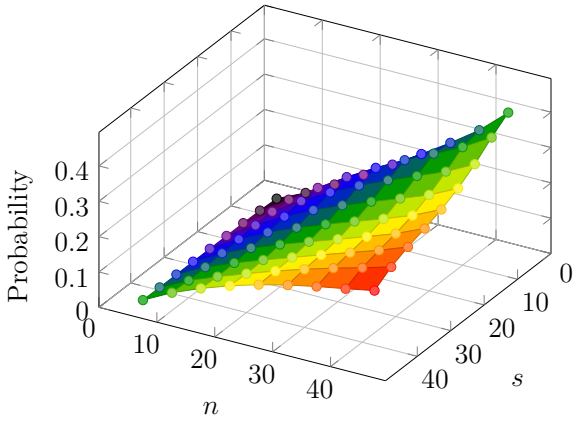
At $(s, n) = (5, 5)$ in Fig. 6.12c, the probability that M_4 is working is the highest. Actually, M_4 is activated when M_3 is active and machine M_1 has produced n products without enough rest. Hence, increasing s and n decreases the probability that M_4 is working, also.

As illustrated in Fig. 6.12a, the probability that M_1 is down is more influenced by n than s. In fact, M_1 fails when it has produced n products without enough rest, whatever the number of raw materials, thus increasing or decreasing s does not affect really the probability that M_1 is down.

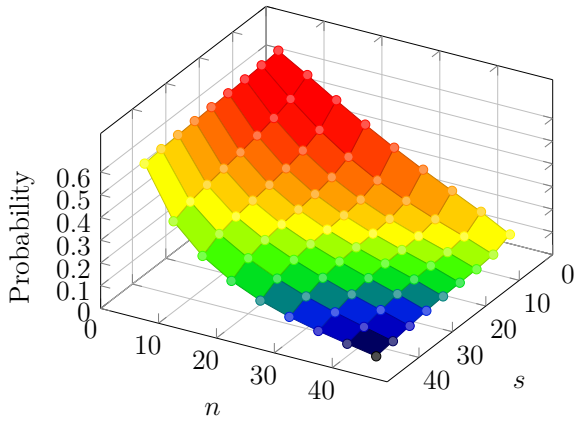
Given these probabilities, we compute total cost $\mathcal{C} = \alpha \times P[m_1d = 1] + \beta \times P[m_3 = 1] + \theta \times P[m_4 = 1]$, where $P[m_1d = 1]$, $P[m_3 = 1]$ and $P[m_4 = 1]$ are the probabilities that M_1 is down, M_3 is working and M_4 is working, respectively. The obtained results are illustrated in Fig. 6.12d, where $(s, n) = (35, 45)$ is the couple of values that minimizes cost $\mathcal{C}$ such that $\alpha = 60$, $\beta = 9$ and $\theta = 20$.



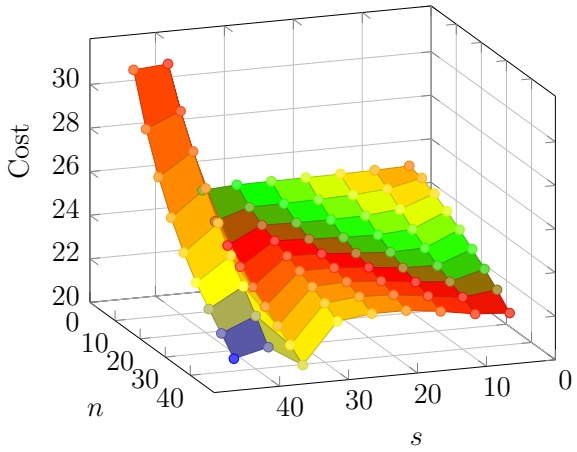
(a) Probabilities that M_1 is down.



(b) Probabilities that M_3 is working.



(c) Probabilities that M_4 is working.



(d) Total cost $\mathcal{C}$, where $\alpha = 60$, $\beta = 9$ and $\theta = 20$.

Figure 6.12: Performance evaluation of the RMS.

6.6 Conclusion

In this chapter, we tackled the problem of reconfiguration in GSPNs. We have presented a formalism that defines particular nets and rules that are exploited to transform reconfigurable nets while preserving several interesting properties after each reconfiguration, hence these properties are decidable whatever the number, even infinite, of obtained configurations. In addition to this, several LTL (after reconfiguration) properties are preserved which may decrease the effort required during formal verification activities.

Furthermore, we have presented an algorithm that computes semi-Markov chain describing stochastic behavior of a given reconfigurable GSPN.

Chapter 7

Discussion

7.1 Introduction

Several extensions have been proposed for enriching Petri nets with reconfigurability to get closer to real dynamic systems and to offer realistic models that reflect the inherent aspects of these systems. However, these gains in modeling are to the expense of analysis level where the techniques are often limited and some properties become undecidable. This last gap has not prevented the development of this category of formalisms and considerable research is being conducted at the analysis level.

In this chapter, we compare the proposed approaches with the current state-of-the-art. We start showing new qualitative modeling aspects provided by RecGSPNs and D-GSPNs formalisms. Then, a quantitative comparison shows how the proposed approaches optimize time and memory consumption in the verification phase.

7.2 Qualitative Aspects

The net rewriting system (NRS) presented in [LO04b] is a formalism for the modeling of dynamic changes in the structure of Petri nets. In NRSs approach, a configuration is modeled by a PN and a reconfiguration is described as a graph rewriting rule. However, no method is given to verify the basic properties of Petri nets. Analyzing such a rewriting system requires computing the corresponding transition system by executing the underlying GTS and using model-checking techniques in case that the obtained transition system is bounded.

Reconfigurable Petri nets (RPNs) [LO04a] represent a subclass of NRSs where a reconfiguration is restricted to the flow function (i.e., there is no change in the sets of places and transitions), thus properties of Petri nets are decidable in the RPNs. The verification process is based on an algorithm that converts RPNs to equivalent basic Petri nets such that one can apply the well-known verification methods of Petri nets. The equivalent basic Petri nets are generated via the duplication of the transitions as many as the number of configurations, which increases considerably the model size. As well, some transformation rules may yield graphs having isolated transitions (their both preset and postset are empty) in order to model the deactivation of the corresponding actions (see Fig. 3.6b). The transformation of these isolated transitions appear in equivalent nets as superfluous transitions (see transition t_1^1 in Fig. 3.7) whose firing does not have any signification. Furthermore, the transformation algorithm connects all transitions of each configuration C_i by self-loops with single place (corresponding ti C_i) that can only contain at most one token (see Fig. 3.7). Hence, obtained model cannot model concurrency between transitions although they are maybe concurrent in the original RPN. Using read arcs [Vog97] may improve such situation.

INRSs approach [Li+09] is a subclass of NRSs restricting NRS reconfiguration forms to preserve specific properties after each net reconfiguration. In INRS, three basic properties

hold: liveness, boundedness, and reversibility (LBR). This practice yields two advantages. First, preserving net properties allows avoiding their verification after each reconfiguration which reduces significantly the space and time complexity of the verification process. Second, LBR properties are still decidable even the modeled system is structurally unbounded, i.e., the number of obtained configurations is infinite. However, PNs considered by INRSs must be ordinary, live, bounded and reversible which is a strong constraint. Moreover, the reconfiguration forms are too restricted which as well restricts the applicability of this approach.

Considering time in Petri nets, reconfigurable timed net condition/event systems (R-TNCESs) [Zha+13] and generalized R-TNCESs (GR-TNCESs) [Khl+15] are proposed to deal with the formal modeling and verification of reconfigurable discrete event control systems. In both former formalisms, a reconfigurable system is modeled by the union of multi timed net condition/event systems (TNCESs) in one R-TNCES, where each TNCES represents one system configuration. An R-TNCES (as well as a GR-TNCES) is not reconfigurable but the reconfiguration is emulated by (i) switching from one TNCES sub-model (corresponding to source configuration) to another one (corresponding to target configuration), and (ii) transferring the system state of the previous configuration to the target configuration. The verification process, based on model checking, of R-TNCES avoids the repetitive verification of unchanged system parts after a reconfiguration. However, encompassing all configurations in one net yields complex models that can only complicate the verification process.

The authors in [CC18] develop an emulator that encodes PNs and their transformations in symmetric nets (SNs) [Chi+93]. Places and transitions are encoded as colors and arcs are represented as markings of specific places in SNs. Therefore, the changes in PN structure can be modeled by changing the marking of these specific places. However, the change in the set of places and transitions is not allowed since they are encoded as colors (in SNs, the color set is unchangeable). Although this formalism has the same modeling power of SNs, it allows (i) modeling reconfigurability in a natural way, and (ii) using the existing tools in verification. Nevertheless, the used emulator contains a high number of immediate transitions leading to an explosion of vanishing markings which makes the verification phase based on existing tools more complicated [CC18].

Reconfigurable SPNs (R-SPNs) [TKB17b] and GSPNs with Rewritable Topology (GSPNs-RT) [TKB17a] extend RPNs to deal with reconfigurability in SPNs and GSPNs, respectively. Yet, the reconfiguration remains limited to arcs (places and transitions cannot be modified). As well, the obtained nets (via transformation towards basic SPNs or GSPNs) may not preserve the concurrency modeled in the original reconfigurable nets.

The authors in [Tig+18] use INRS-based approach to deal with the reconfiguration of live, bounded and reversible (LBR) GSPNs, however, the reconfiguration is too restricted since it uses specific reconfigurations and considers only ordinary LBR GSPNs.

Although several extensions tackle reconfigurability in GSPNs, RecGSPNs formalism differs in two main aspects: (i) more reconfiguration behaviors are allowed instead of those proposed in [CC18, TKB17a, Tig+18], and (ii) thanks to RecGSPNs, several basic properties (e.g., boundedness, reversibility, etc.) can be preserved after reconfiguration, under some conditions, hence these properties are decidable even if the number of obtained configurations is infinite.

As for D-GSPNs, they allow modeling any reconfiguration form while transforming D-GSPNs toward GSPNs is still possible when the number of obtained configurations is finite. However, the transformation algorithm computes conventional GSPNs (i.e., do not contain either inhibitor, read, transfer, or reset arcs [DFS98]) leading to the same shortcomings discussed above concerning R-SPNs and GSPNs-RTs. Another modeling issue is that reconfiguration can be only timed (i.e., immediate reconfigurations modeling immediate switchings are not allowed). Adding immediate reconfigurations to D-GSPNs formalism brings the same interests of adding immediate transitions to SPNs models. In fact, immediate reconfigurations are not allowed in D-GSPNs formalism in order to be able to erase all tokens of place that models an obsolete place before switching to another configuration (see proofs in Subsection 5.5). Adding immediate reconfiguration to D-GSPNs formalism requires as well adding reset arcs to the target nets, thus obtained GSPNs become no longer conventional.

A comparison of modeling/verification features is given in Table 7.1 and Fig. 7.1.

Formalism	Modeling				Verification			
	Modeling SUS	+/- P/T	+/- Arcs	GSPNs	QLV of SUS	QLV of SBS	QNV of SUS	QNV of SBS
NRS [LO04b]	✔	✖	✔	✖	✖	✖	✖	✖
RPN [LO04a]	✖	✖	✔	✖	✖	✔	✖	✖
R-TNCES [Zha+13]	✖	✖	✖	✖	✖	✔	✖	✔
INRS [Li+09]	✔	✔	✔	✖	✔	✔	✖	✖
Evolving PN [CC18]	✖	✖	✔	✔	✖	✔	✖	✔
R-SPNs [TKB17b]	✖	✖	✔	✖	✖	✔	✖	✔
GSPNs-RT [TKB17a]	✖	✖	✔	✔	✖	✔	✖	✔
D-GSPNs	✖	✔	✔	✔	✖	✔	✖	✔
RecGSPNs	✔	✔	✔	✔	✔	✔	✖	✔

where +/-, SUS, SBS, P/T, QLV and QNT stand for adding/removing, structurally-unbounded systems, structurally-bounded systems, places/transitions, qualitative verification, and quantitative verification, respectively.

Table 7.1: A comparison of modeling/verification features.

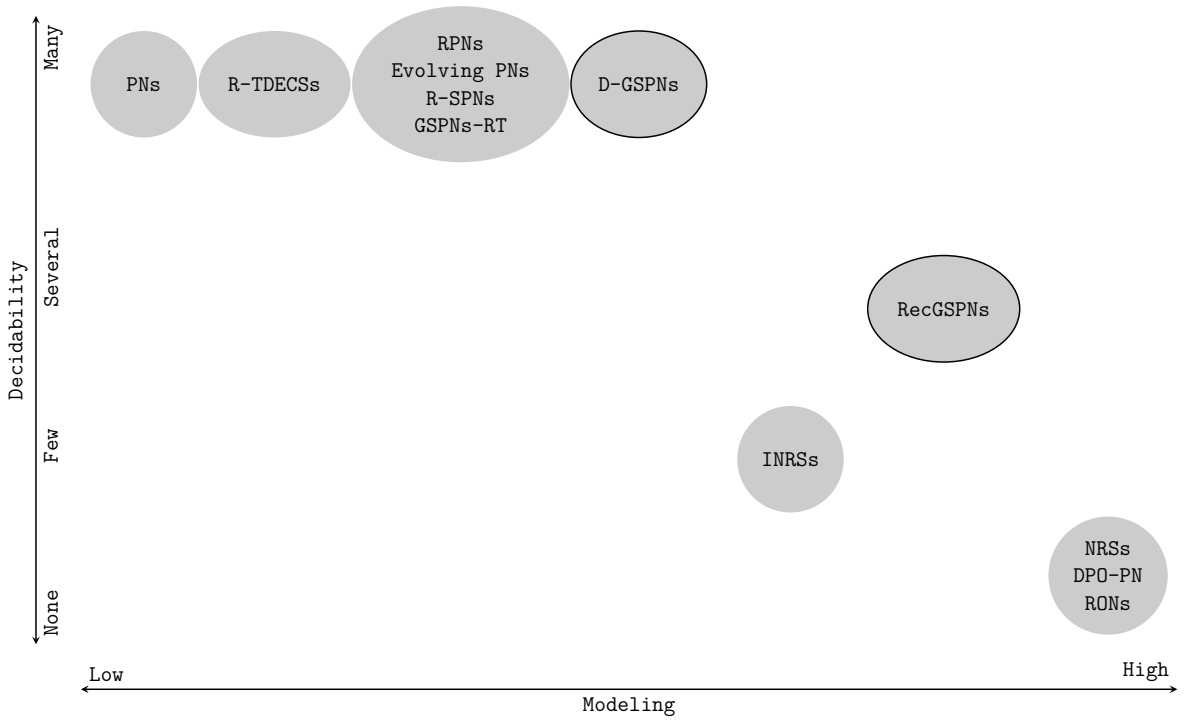


Figure 7.1: Modeling versus Decidability

7.3 Quantitative Aspects

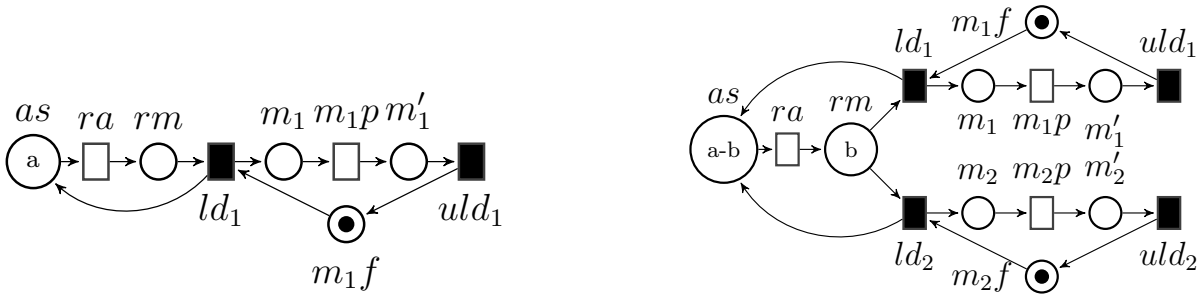
Here, we provide a quantitative comparison between our proposals and existing approaches. In order to show the efficiency of D-GSPNs and RecGSPNs, we consider three factors, namely, the model size, spatial complexity and computation time.

To the best of our knowledge, there is no method that allows computing quantitative properties of infinite-structure reconfigurable systems using neither SPNs nor GSPNs. Therefore, this quantitative comparison considers only finite-structure reconfigurable systems.

First, we elaborate the required models describing a reconfigurable system based on: (i) non-reconfigurable formalism, (ii) an approach that transforms dynamic GSPNs to GSPNs, and (iii) a formalism that computes Markov chain while executing graph transformations, namely, basic GSPNs, D-GSPN, and RecGSPNs, respectively. Then, we compare the size of obtained models (i.e., transitions, places, arcs, etc.). Finally, we compute Markov chains isomorphic to obtained models and discuss their spatial/temporal complexity.

This comparison is conducted on a reconfigurable system composed of machine M_1 permanently active and $n-1$ machines each of which denoted by M_i, $i \in \{2, \ldots, n\}$. Machine M_i is activated when the number of raw materials in the buffer exceeds a threshold. Each machine M_i is modeled by q_i nodes (i.e., places and transitions).

The initial configuration containing M_1 is illustrated in Fig. 7.2a, and the second configuration containing M_1 and M_2 is depicted in Fig. 7.2b.



(a) Initial configuration. (b) Second configuration.

Figure 7.2: GSPN models of initial and second configurations.

7.3.1 Factor 1: Model size

To model reconfigurable systems, *classical approaches* use basic GSPNs without explicit reconfiguration (e.g., **GreatSPN** [Baa+09]). Indeed, the classical tools simulate the reconfiguration by establishing a complete model that encompasses all possible configurations. The complete model contains a set of sub-models (SMs) each of which represents one possible configuration. Hence, the complexity of models increases considerably. RecGSPNs approach avoids the construction of such complex models since the reconfiguration is expressed via a set of rules.

In a classical approach, if initial configuration C_0 is modeled by a sub-model composed of k nodes, then sub-model of second configuration C_1 (composed of M_1 and M_2) contains $(k + q_2)$ nodes. Hence, the sub-model of j^{th} configuration contains $(k + q_2 + \cdots + q_j)$ nodes. Consequently, the entire model contains $(k + (k + q_2) + (k + q_2 + q_3) + \cdots + (k + \Sigma_{i=2}^{n} q_i) + Q_n)$ nodes, where Q_n is the number of nodes connecting n sub-models. Moreover, a reconfiguration from C_i to C_j is modeled by the state migration from subnet modeling C_i to subnet modeling C_j. This practice enlarges the model size since it duplicates original nodes as many as they appear in each configuration.

Fig. 7.3 depicts a configuration containing two machines according to classical approach, such that (i) places c_0 and c_1 are associated with C_0 and C_1, respectively, (ii) a token inside c_0 (c_1, respectively) means that the current configuration is C_0 (C_1, respectively), (iii) transitions r_0 and r_1 model the switching from C_0 to C_1 and from C_1 to C_0, respectively, and (iv) transitions depicted by ■ (▨, respectively) reconfigure the net state from C_0 to C_1 (from C_1 to C_0, respectively).

Based on D-GSPNs formalism, the equivalent model used to analyze this reconfigurable system contains $((\sum_{i=1}^{nc} |T_{C_i}| + |P_{C_i}|) + nc + |\mathcal{R}|) + |EM|$ nodes, where P_{C_i} and T_{C_i} are the sets of places and transitions of configuration C_i, respectively, $\mathcal{R}$ is the set of rules, EM is the set of emptying transitions, and nc is the number of configurations. Fig. 7.4 shows a configuration containing two machines according to D-GSPNs approach.

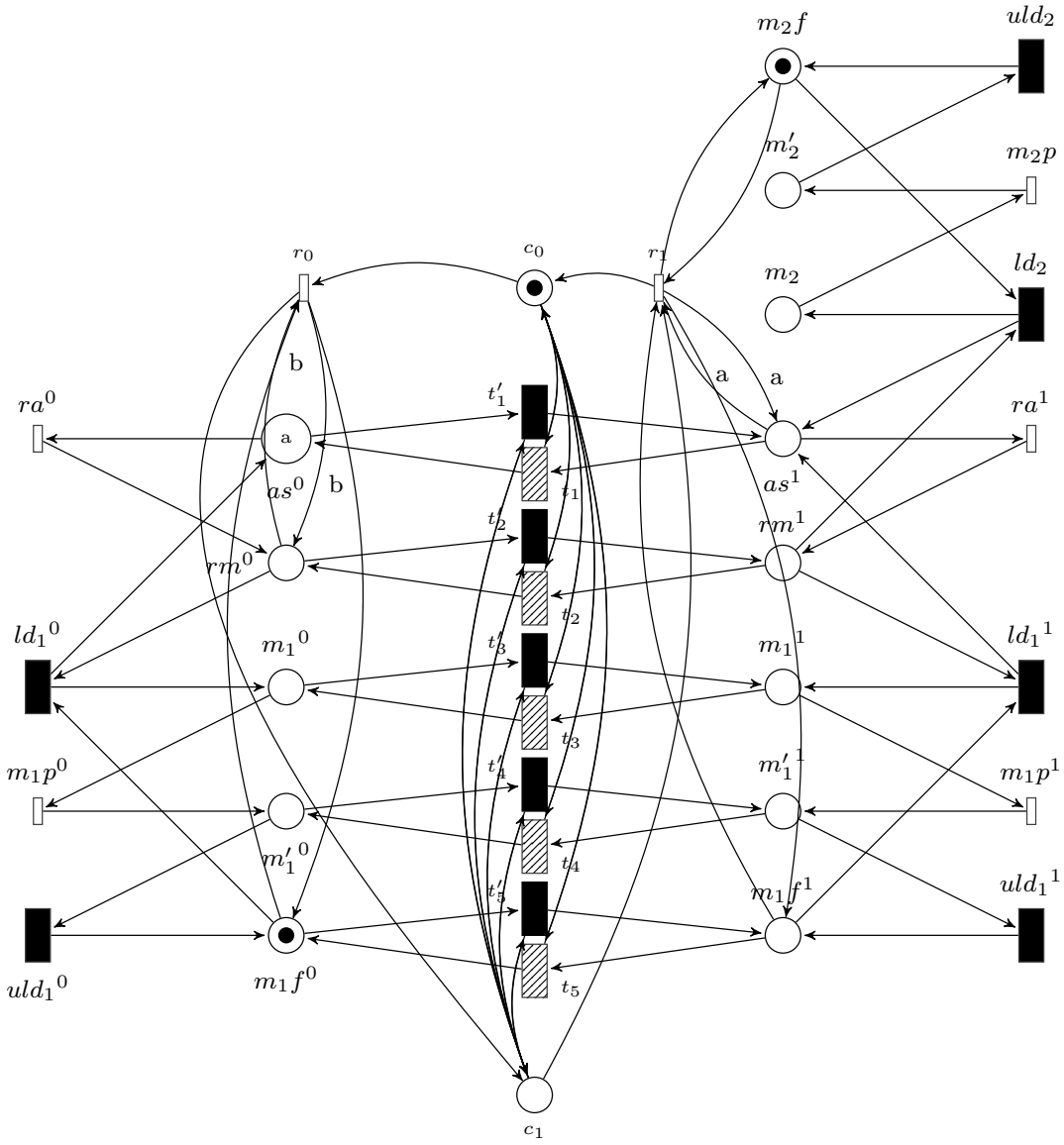


Figure 7.3: Model of the reconfigurable system having two machines based on classical approaches.

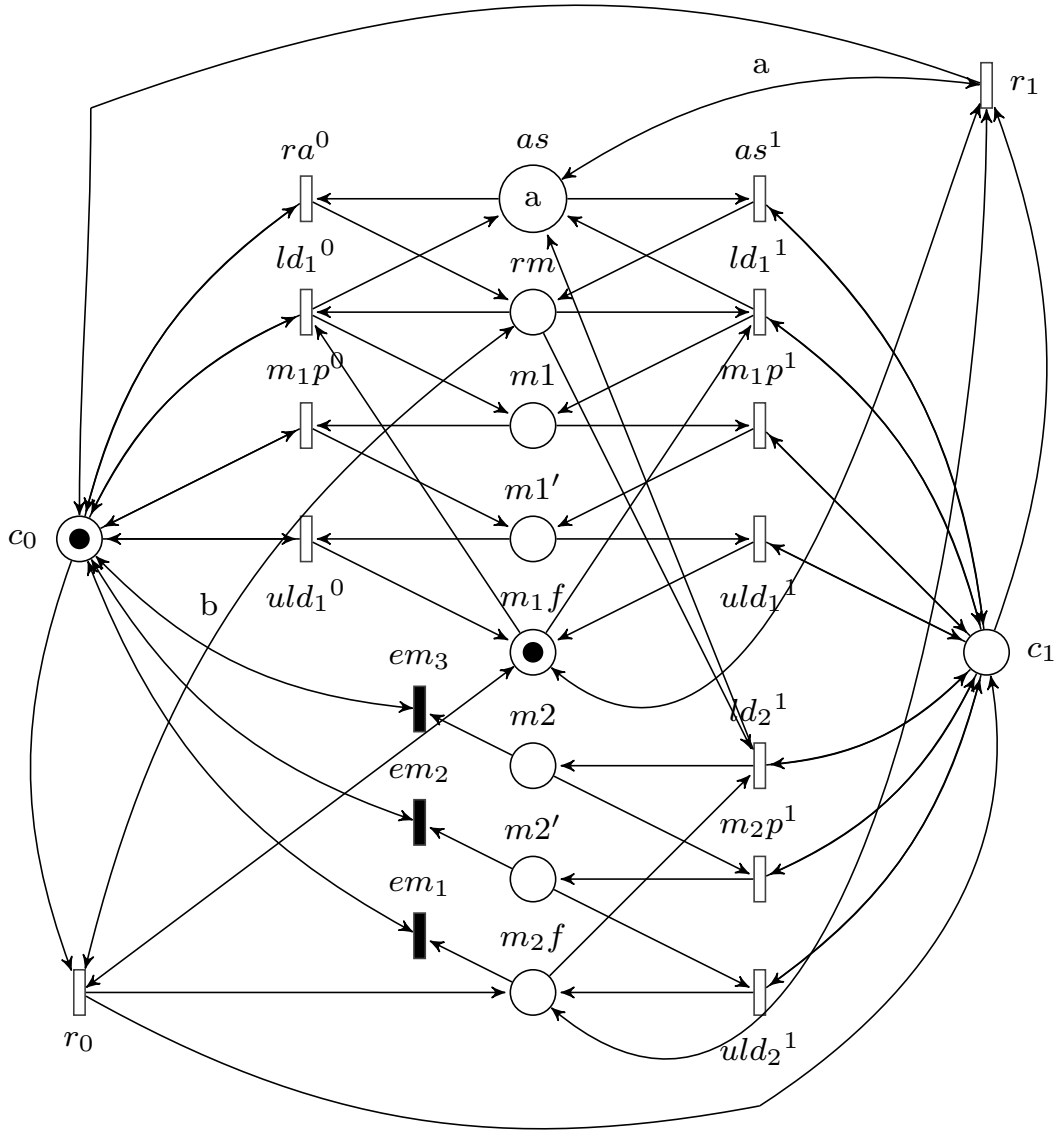


Figure 7.4: Model of the reconfigurable system having two machines based on D-GSPNs approach.

In RecGSPNs, any configuration used in the analysis contains at most $(k + \Sigma_{i=2}^{n}\ q_i)$ nodes, where k is the number of nodes in the initial configuration, q_i is the number of nodes in configuration C_i, and n is the number of configurations. Fig. 7.5 depicts a comparison between RecGSPNs versus existing approaches according to the model size of the described reconfigurable system having n machines, for all $n \in \{2, \ldots, 10\}$.

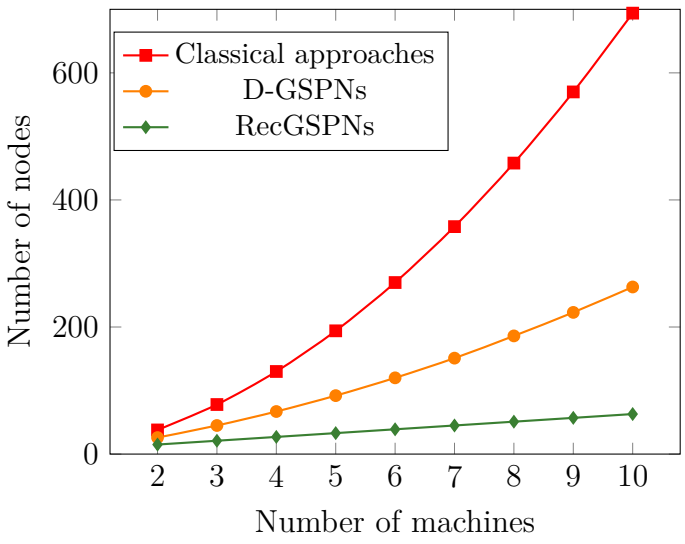


Figure 7.5: Model size.

7.3.2 Factor 2: Markov chain spatial complexity

Now, we consider spatial complexity of Markov chains isomorphic to models elaborated above.

Let NS be the number of reachable states and NM be the number of states modeling the state migration, then we have (i) in case of classical approaches, Markov chain contains NS states, (ii) in RecGSPNs approach, it contains $(NS - NM)$ states. It means that the size of Markov chain in terms of states obtained by RecGSPNs approach is less than or equal to that in existing approaches. Therefore, the spatial complexity of quantitative verification is reduced.

In order to illustrate the advantage of RecGSPNs at this level, we compare the size of Markov chains obtained by the three approaches on four cases of the reconfigurable system presented previously, where the buffer contains ten spaces. In each case, various numbers of machines are used.

Note that D-GSPNs formalism does not allow modeling immediate reconfigurations, that is any reconfiguration is modeled as a timed transition. Thus, immediate transitions in the original models are replaced by timed ones in order to allow firing the transitions that model reconfiguration rules when it is necessary, which enlarges the state space. Recall that using immediate transitions reduces the state space [Mar+94].

As for models obtained by classical approaches, we use **PIPE** [DKS09] to compute the Markov chains. The obtained results are given in Table. 7.2. In the third case where the reconfigurable system contains four machines, **PIPE** cannot compute the Markov chain due to the state explosion problem.

# of Mach.	RecGSPNs	D-GSPNs	Classical approach
2	100 states	133 states	1963 states
3	220 states	435 states	11340 states
4	421 states	1432 states	//
5	743 states	9226 states	//

Table 7.2: Number of states in Markov chains obtained according to the number of machines, where # stands for number.

7.3.3 Factor 3: Markov chain time complexity

Finally, we consider the time required to compute Markov chains discussed above. We run experiments on Intel Core i5-8500 processor with 16 GB of memory, where Java Virtual Machine (the tool is developed in Java) was started with a maximum size of 2 GB. We took measurements several times to compute the (average) required computing time. Fig. 7.6 shows the required computing time of Markov chains according to different approaches.

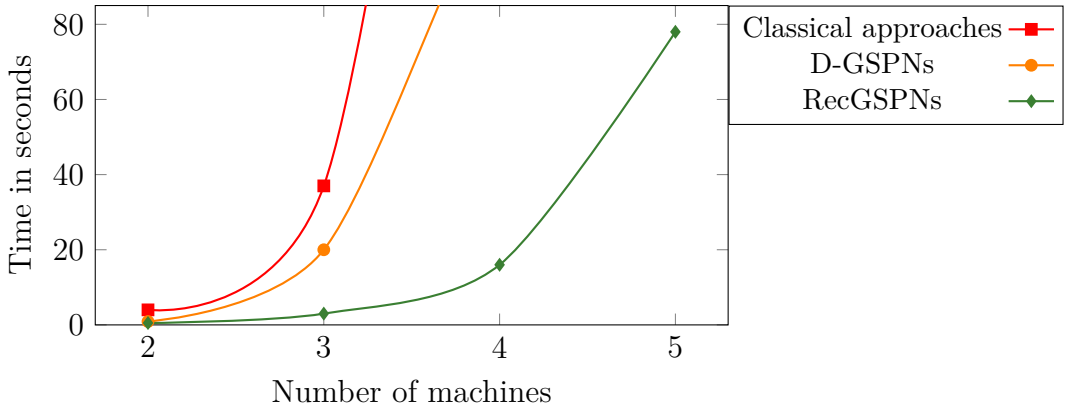


Figure 7.6: Time to compute Markov chain.

Changing hardware/software specifications (e.g., programming language, processor or memory size) may change the obtained results. Yet, the gap between RecGSPNs and the other approaches in terms of computation time will remain in favor of RecGSPNs.

In fact, for classical approaches, the temporal complexity is very high due to the state migration (from configuration to another which adds additional states to Markov chains) and the large size of the model (which requires extra time to be explored). As for D-GSPNs, the

need to replace immediate transitions in the original models by timed ones increases the time complexity as well.

Considering RecGSPNs, their temporal complexity is much lower than that in classical approaches. Indeed, RecGSPNs do not use additional states (state migration) to emulate the reconfigurations, since RecGSPNs approach computes Markov chains directly from executing the graph transformations and firing enabled transitions.

Nevertheless, the gap between classical approaches and RecGSPNs in terms of computation time could be reduced by a modeling trick. In fact, state migration is modeled by firing transitions that transfer tokens (one token at time) from a sub-model to another one. This kind of transfer, which increases computation time of state space, can be modeled by transfer arcs [DFS98] by which each transition transfers all tokens in a single firing step. However, the use of transfer arcs decreases the model decidability [DFS98].

Moreover, this gap depends, as reported in [RSV04], as well on problem dynamic. Indeed, classical approaches give better results when system dynamic is too limited, whereas for problems having highly dynamic structures, RecGSPNs formalism is a promising approach.

Finally, we note that several properties (e.g., that involving only strongly preserved places, reversibility, liveness, etc.) are preserved after reconfiguration based on RecGSPNs, therefore no time is required to verify them. Hence, the temporal complexity of both qualitative and quantitative verifications (as presented above) are decreased.

7.4 Conclusion

In this chapter, a comparison of the proposed approaches with the existing tools has been conducted. This comparison shows the efficiency of the proposed approaches in this thesis at modeling and verification compared with classical formalisms. We have shown new qualitative modeling aspects provided by RecGSPNs formalism. As well, we have shown how RecGSPNs formalism optimizes time and memory consumption in the verification phase.

Although we have carried out a comparison through a small RMS, the spatial/temporal complexity of the verification phase based on either D-GSPNs or classical approaches was enormous. One of the reasons for this high complexity is due to graph transformations.

In fact, model checking and behavioral analysis require the computation of a transition system (reachability graph) in order to check a set of desired properties. To this end, they enumerate each reachable pair of state/configuration. Reachable configurations are computed via graph transformations that search for a match of an LHS, and eventually NACs, of each rule in each reachable configuration (graph). This searching is called graph matching problem [ZBV08] which is inherently more complex [RSV04], as well as, increases subsequently the verification process complexity.

RecGSPNs formalism based on transformations preserving some desired qualitative properties is a promising approach. Indeed, it is not required to verify a set of properties after any RecGSPNs-based reconfiguration leading to two major advantages. First, the verification complexity is largely reduced and some desired qualitative properties are still decidable even if the system is structurally unbounded.

Conclusion

Modern discrete-event systems are becoming increasingly complex, structurally dynamic and variably interconnected. These systems are designed to be able to change their structure and/or topology, at run-time, by adding/removing interconnections, objects, or even subsystems, to accommodate new circumstances/requirements.

Using (non-reconfigurable) Petri nets in modeling/analyzing of reconfigurable systems is no more sufficient. Indeed, they are unable to specify/verify, in a natural way, advanced systems having dynamic structures. Hence, several researchers have enriched PNs with dynamic structures in order to offer suitable modeling/verification approaches of dynamic systems.

In this thesis, we have presented the main reconfigurable PNs extensions. These extensions can be classified into three categories.

1. Extensions allowing modeling reconfiguration without any restriction. Yet, the verification of obtained models is based on model checking. The complexity of graph transformations along with model checking make the verification process a hard task even for bounded systems.

2. Approaches restricting the possible reconfigurations in such a way that the number of obtained models are finite. Then, they transform, encode or unfold these models into equivalent (non-reconfigurable) Petri nets. Although the PNs properties are still decidable, the modeling power is not increased.

3. Finally, approaches applying reconfigurations that preserve desired properties. Such practice avoids the verification of desired properties after each reconfiguration, hence their verification process complexity is reduced. Moreover, these properties are still decidable even if the modeled system is structurally unbounded.

Our goal in this dissertation was to introduce reconfigurability into the well-known GSPNs formalism that remedies the shortcomings in the other formalisms. The need to relax the constraints imposed by existing formalisms either on modeling or verification level led us to propose our first contribution, called GSPNs with rewritable topology (GSPNs-RT). In the latter, the sets of places and transitions are fixed, whereas the topology is dynamic. GSPNs-RT can be transformed into basic equivalent GSPNs that will be used later on to verify the GSPNs-RT properties using classical verification methods proposed for GSPNs.

As a second contribution, to take full advantage of GSPNs characteristics (i.e., modeling/decision power, analysis methods supported by off-the-shelf tools, etc.), we have proposed a new formalism that transforms dynamic GSPNs to basic GSPNs. This transformation allows us to straightforwardly exploit the methods and techniques proposed in the literature for GSPNs in dynamic GSPNs verification. This transformation can take place only when the set of reachable configurations obtained by transformation rules is finite.

To consider infinite structure, we have proposed reconfigurable generalized stochastic Petri nets (RecGSPNs). Actually, RecGSPNs offer to designers a wider range of possible structural changes where both sides of any rule are no longer defined by their structure, instead, they must show some behaviors allowing to use them in the reconfiguration process. The use of RecGSPNs-based reconfiguration preserves five important properties, namely, liveness, boundedness, reversibility, deadlock-freedom and home state. Moreover, many properties expressed by linear time logic can be preserved after each system reconfiguration. Thus, these properties are decidable whatever the number of obtained configurations that can be infinite. This practice enjoys double advantages

1. Temporal and spatial complexity are reduced since these properties are verified only at the first configuration, hence no need to compute and explore the whole set of reachable states of all reachable configurations,

2. Often, applying transformation rules to graphs leads to structurally infinite models, and hence properties are not decidable based on classical verification techniques. In our approach, several properties are still decidable since applying any rule will preserve them.

Finally, we have conducted a comparison which showed quite good characteristics of proposed formalisms in this thesis compared with existing ones. However, several issues remain to be investigated such as:

- Enriching the set of possible reconfiguration forms of RecGSPNs. Although the restrictions on reconfiguration forms in RecGSPNs are relaxed compared with either INRSs or INRSs-GSPNs, it is useful to consider reconfiguration forms that can preserve one property without considering the others. For instance, a designer may be interested in preserving the boundedness of a given system, rather than preserving other properties.

- The quantitative verification of reconfigurable systems based on RecGSPNs is still conducted in an old-fashioned manner. Indeed, it is required to compute the state space of a given reconfigurable system for performance evaluation, which is highly complex (yet, less complex than existing approaches). Knowing that a reconfiguration will increase, decrease or preserve a metric, based on the properties of the corresponding rule may avoid the computation of the state space.

- Consider the quantitative properties of structurally unbounded systems. Although we have considered the qualitative properties of such systems, their quantitative properties cannot be analyzed based on RecGSPNs.
- Consider the reconfiguration in other stochastic Petri net classes that involve other kinds of law distributions.

Appendices

Appendix A

A Prototype for RecGSPNs.

A.1 Introduction

We aim at developing a tool that deals with reconfigurable generalized stochastic Petri nets. To this end, we have developed a basic tool[1] that has as inputs a GSPN that models an initial configuration and a set of rules each of which models a possible change in the structure of the reconfigurable net. Then, our tool applies these rules to a reconfigurable net and computes its isomorphic Markov chain. Once the latter is completely constructed, the tool computes the quantitative properties such as: throughput of a transition, mean number of tokens in a place, the mean sojourn time at a marking, etc.

A.2 Description

Here, we give a basic description of developed classes, their important fields and methods.

A.2.1 `Place`

Place class allows to create an instance of a place.

Fields

```
public String IP; //This is a place label.
public int MP;  //This is a place marking.
```

Methods

```
public Place(String p,int m);
```

This method is used to create an instance of a place, where `p` is a label and `m` is a marking.

```
public boolean equals(Place p);
```

This method is used to compare two places, where `p` is the place to be compared with. It returns **true** if two places are having same label and marking.

A.2.2 `Transition`

Transition class allows to create an instance of a transition.

Fields

```
public String IT;
//This is a transition label.
public Transition.Type type;
```

[1] `https://kahloul2006.wixsite.com/laid-kahloul/reconfigurable-gspns`

```
//This is a transition type: timed or immediate.
public double rate;
//This is a transition rate/weight.
```

Methods

```
public Transition(String l, Transition.Type t, double r);
```

This method is used to create an instance of a transition, where `l` is a transition label, `t` is a transition type: timed or immediate, and `r` is a transition rate/weight.

```
public boolean equals(Transition t);
```

This method is used to compare two transitions, where `t` is a transition to be compared with. This method returns **true** if two transitions have same label, rate and type.

A.2.3 `Rule`

Rule class allows to create an instance of a rule.

Fields

```
public GSPN L;
//This is a left-hand side.
public GSPN R;
//This is a right-hand side.
public ArrayList<GSPN> NAC;
//This is a list of negative application conditions.
public String[] IL;
//This is an input interface of left-hand side.
public String[] OL;
//This is an output interface of left-hand side.
public String[] IR;
//This is an input interface of right-hand side.
public String[] OR;
//This is an output interface of right-hand side.
public double weight;
//This is a weight application of rule.
public String ID;
//This is an identificator of rule.
public Place[] activatingMarking;
//This is an activator marking that controls rule application.
```

Methods

```
public Rule(String ID, GSPN L, GSPN R, ArrayList<GSPN> NAC, String[] IL, String[]
    OL, String[] IR, String[] OR, double weight, Place[] AM);
```

This method is used to create an instance of a rule, where `ID` is an identification of rule, `L` is a left-hand side, `R` is a right-hand side, `NAC` is a list of negative application conditions, `IL` is an input interface of left-hand side, `OL` is an output interface of left-hand side, `IR` is an input interface of right-hand side, `OR` is an output interface of right-hand side, `weight` is a weight application of rule, and `AM` is an activator marking that controls rule application.

```
public boolean isInIL(String n);
```

This method is used to check whether a node belongs to input nodes of left-hand side of a rule, where `n` is a label node. It returns **true** if node `n` belongs to input nodes of left-hand side of the rule.

```
public boolean isInOL(String n);
```

This method is used to check whether a node belongs to output nodes of left-hand side of a rule, where `n` is a label node. It returns **true** if node `n` belongs to output nodes of left-hand side of the rule. Analogously to methods `isInIR` and `isInOR` with respect to right-hand side.

A.2.4 GSPN

GSPN class allows to (i) create an instance of a GSPN from a PNML file describing its structure and (ii) compute its reachability graph. As well, it allows to compute its quantitative properties, such as: mean number of tokens, token probability density, throughput, etc.

Methods

```
public GSPN(Place[] setOfP, Transition[] setOfT, int[][] pr, int[][] po);
```

This method is used to create an instance of a GSPN, where `setOfP` is a set of places, `setOfT` is a set of transitions, `pr` is presets of transitions and `po` is postsets of transitions.

```
public GSPN(String xFile);
```

This method is used to create an instance of a GSPN, where `xFile` is the path of PNML file containing the description of a GSPN created by a third-party.

```
public int getNumberOfTangibleStates();
```

This method is used to get the number of tangible states in reachability graph.

```
public int getNumberOfStates();
```

This method is used to get the number of states in reachability graph.

```
public boolean isFireable(String t, Place[] M);
```

This method is used to check whether a transition `t` is fireable at a marking `M`.

```
public void fire(String t);
```

This method is used to fire a transition `t` at the current marking of a GSPN.

```
public Place[] getMarkingAfterFiring(String t, Place[] M);
```

This method is used to compute the obtained marking after firing a transition `t` at marking `M`.

```
public JSONArray getReachabilityGraph();
```

This method is used to get a reachability graph as a JSON object.

```
public .JSONArray getMarkingsDistProba();
```

This method is used to get a marking distribution probability.

```
public JSONArray getMeanNumberOfTokens();
```

This method is used to get mean number of tokens.

```
public JSONArray getTokenProbabilityDensity();
```

This method is used to get token probability density.

```
public JSONArray getProbabilitiesFiringTransition();
```

This method is used to get firing transition probability density.

```
public JSONArray getThroughputOfTransitions();
```

This method is used to get throughput of transitions.

```
public org.json.simple.JSONArray getMeanSojournTime();
```

This method is used to get mean sojourn time.

A.2.5 `RecGSPN`

RecGSPN class allows to create an instance of a reconfigurable generalized stochastic Petri net describing its dynamic structure. As well, it allows to apply rules to reconfigurable nets and compute their quantitative properties, such as: mean number of tokens, token probability density, throughput, etc.

Methods

```
public RecGSPN(GSPN G0, ArrayList<Rule> setOfRules);
```

This method is used to create an instance of RecGSPN, where `G0` is an initial configuration, and `setOfRules` is a set of rules.

```
public boolean isApplicable(Rule r, GSPN G, Place[] M);
```

This method is used to check whether a rule `r` is applicable to a GSPN `G` at a marking `M`.

```
public GSPN getGSPNAfterApplayingRule(Rule r, GSPN G, Place[] M);
```

This method is used to compute obtained GSPN after applying `r` to GSPN `G` at marking `M`.

```
public int getNumberOfTangibleStates();
```

This method is used to get the number of tangible states in the reachability graph.

```
public JSONArray getReachabilityGraph();
```

This method is used to get reachability graph.

```
public JSONArray getMarkingsDistProba();
```

This method is used to get marking distribution probability.

```
public JSONArray getMeanNumberOfTokens();
```

This method is used to get mean number of tokens.

```
public String[][] getMeanNumberOfTokensAsMatrix();
```

This method is used to get mean number of tokens as matrix.

```
public JSONArray getTokenProbabilityDensity();
```

This method is used to get token probability density.

```
public String[][] getTokenProbabilityDensityAsMatrix();
```

This method is used to get token probability density as matrix.

```
public JSONArray getProbabilitiesFiringTransition();
```

This method is used to get firing transition probability density.

```
public String[][] getTransitionsStat();
```

This method is used to get firing transition probability density and throughputs as matrix.

```
public JSONArray getThroughputOfTransitions();
```

This method is used to get throughput of transitions.

```
public JSONArray getMeanSojournTime();
```

This method is used to get mean sojourn time.

A.3 Demonstration

In this section, we demonstrate how to model/verify (quantitatively) a reconfigurable net. First, the user can use a third-party tool to create a GSPN that models the initial configuration of the reconfigurable net. Note that the GSPN must be described by the standard format PNML as used by **PIPE** tool [DKS09]. As well, left- and right-hand sides of each rule are GSPNs that can be created by a third-party.

Let us consider a reconfigurable system composed of machine M_1 permanently active and machine M_2 which is activated when the number of raw materials in the buffer exceeds five. The initial configuration containing M_1 is highlighted in Fig. A.1. The interpretation of places and transitions is given as follows.

1. as (resp. rm): Its marking represents the number of free spaces (resp. raw materials) in the central buffer.
2. m_1 (resp. m_1'): A token in m_1 (resp. m_1') means that machine M_1 has begun (resp. has finished) processing.
3. $m_1 f$: A token in $m_1 f$ means that machine M_1 is idle.
4. ra: Raw material is loaded in the central buffer.
5. ld_1 (resp. uld_1): M_1 loads an item (resp. unloads a product).
6. $m_1 p$: Machine M_1 is processing.

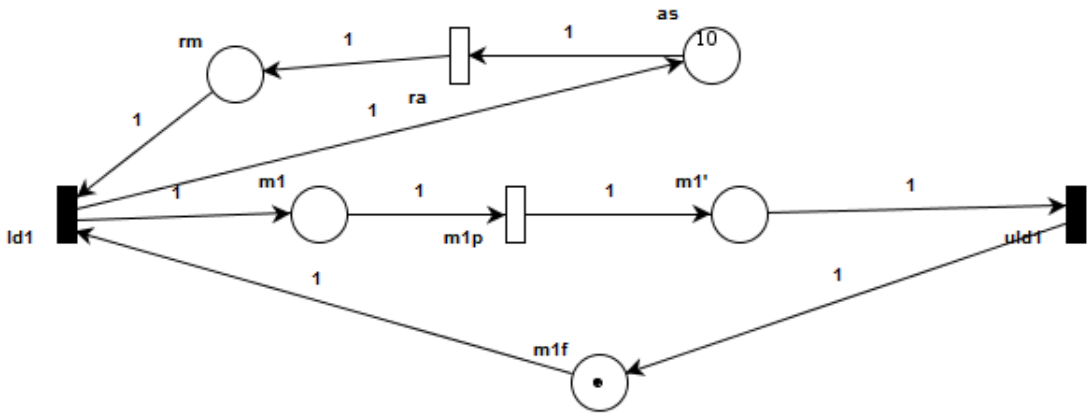


Figure A.1: Initial configuration.

Once the number of raw materials in the buffer exceeds five, machine M_2 is activated and the system switches to its second configuration. This reconfiguration is modeled by rule r_1, where its left- and right-hand sides are shown in Figs. A.2 and A.3, its input nodes are $(\{\texttt{ld1}\}, \{\texttt{ld1}, \texttt{ld2}\})$, and its output nodes are $(\{\texttt{uld1}\}, \{\texttt{uld1}, \texttt{uld2}\})$.

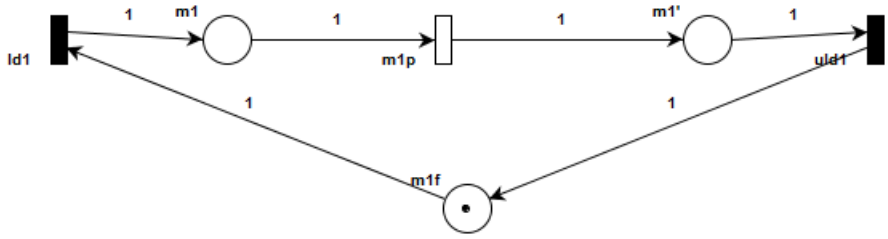


Figure A.2: Left-hand side of rule r_1 and right-hand side of rule r_2.

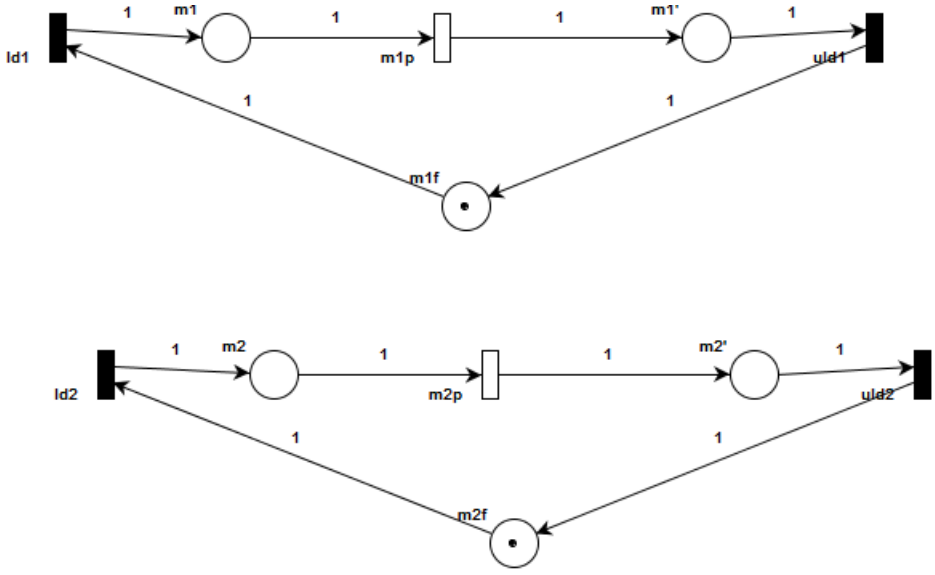


Figure A.3: Right-hand side of rule r_1 and left-hand side of rule r_2.

Consider the following code.

```
import java.util.ArrayList;
  public class Main {
     public static void main(String[] args) {
        GSPN G= new GSPN("C_0.xml"),
           L1=new GSPN("L_1.xml"),
           R1=new GSPN("R_1.xml"),
           L2=new GSPN("L_2.xml"),
           R2=new GSPN("R_2.xml");

        String[]IL1={"ld1"};
        String[]OL1={"uld1"};
        String[]IR1={"ld1","ld2"};
        String[]OR1={"uld1","uld2"};
        Place[] am1=new Place[5];
        am1[0]=new Place("as",0);
        am1[1]=new Place("rm",6);
        am1[2]=new Place("m1",0);
        am1[3]=new Place("m1'",0);
        am1[4]=new Place("m1f",1);
        ArrayList<GSPN> NAC1=new ArrayList();
        NAC2.add(L2);

        Rule r1 = new Rule("r1", L1, R1, NAC1, IL1, OL1, IR1, OR1, 2, am1);

        String[]IL2={"ld1","ld2"};
        String[]OL2={"uld1","uld2"};
        String[]IR2={"ld1"};
        String[]OR2={"uld1"};
        Place[] am2=new Place[8];
        am2[0]=new Place("as",10);
        am2[1]=new Place("rm",0);
        am2[2]=new Place("m1",0);
        am2[3]=new Place("m1'",0);
        am2[4]=new Place("m1f",1);
        am2[5]=new Place("m2",0);
        am2[6]=new Place("m2'",0);
        am2[7]=new Place("m2f",1);
        Rule r2 = new Rule("r2", L2, R2, null, IL2, OL2, IR2, OR2, 2, am2);
```

```
39
40          ArrayList<Rule> lr= new ArrayList<>();
41          lr.add(r1);
42          lr.add(r2);
43
44          RecGSPN rgspn = new RecGSPN(G,lr);
45          System.out.println(rgspn.getNumberOfGSPNs());
46          System.out.println(rgspn.getNumberOfStates());
47          System.out.println(rgspn.getMeanNumberOfTokens());
48          System.out.println(rgspn.getProbabilitiesFiringTransition());
49          System.out.println(rgspn.getThroughputOfTransitions());
50      }
51
52  }
```

L_1 (left-hand side) and R_1 (right-hand side) of r_1 are instantiated at Lines (5) and (6), respectively. Input and output nodes of L_1 are defined as arrays of String at Lines (10) and (11). As well, input and output nodes of R_1 are defined at Lines (12) and (13). Aforementioned, rule r_1 is applicable to initial configuration when the number of raw materials in the buffer exceeds five. The activator marking of rule r_1 is defined as array of `Place` at Lines (14)–(19). The instruction at Line (16) states that the marking of place rm (its marking modeling the number of raw materials in the buffer) is six. Finally, rule r_1 is instantiated at Line (23), where its set of negative application conditions contains L_2 (Fig. A.3). Indeed, r_1 is not applicable if machine M_2 is already activated.

Once the buffer is empty, the system switches to its initial configuration. This switching is modeled by rule r_2, where its left- and right-hand sides are depicted in Figs. A.3 and A.2, respectively.

L_2 (left-hand side) and R_2 (right-hand side) of r_2 are instantiated at Lines (7) and (8), respectively. Input and output nodes of L_2 are defined as arrays of String at Lines (25) and (26). As well, input and output nodes of R_2 are defined at Lines (27) and (28). Rule r_2 is applicable to second configuration when the buffer is empty. The activator marking of rule r_2 is defined as array of `Place` at Lines (29)–(37). The instruction at Line (30) states that the marking of place as (its marking modeling the number of available spaces in the buffer) is ten. Finally, rule r_2 is instantiated at Line (38), where its set of negative application conditions is empty.

Rule r_1 and r_2 are inserted into list `lr` at Lines (41) and (42) to create a set of rules.

The reconfigurable net is instantiated at Line (44), where its set of rules is `lr` and its initial configuration is `G` instantiated at Line (4).

Finally, we can compute different parameters, such as the number of obtained GSPNs after applying the set of rules to the initial configuration, the number of states in the isomorphic Markov chain, the mean number of tokens at each place, etc.

The result of execution of the above code is the following, where the weights/rates of all transitions are equal to one.

```
2//The number of obtained GSPNs.
96//The number of states in the isomorphic Markov chain
//The mean number of tokens at each place
[{"values":8.163158068311944,"id":"as"},
{"values":0.8025364509473432,"id":"m1"},
{"values":0.0,"id":"m1'"},
{"values":0.19746354905265673,"id":"m1f"},
{"values":1.8368419316880582,"id":"rm"},
{"values":0.1929087837708813,"id":"m2"},
{"values":0.0,"id":"m2'"},
{"values":0.012781191714198437,"id":"m2f"}]
//Firing transition probability
[{"values":0.37306141543396915,"id":"m1p"},
{"values":0.5602739943546192,"id":"ra"},
{"values":0.06666459021141184,"id":"m2p"}]
//Transition throughput
[{"values":0.8025364509473432,"id":"m1p"},
{"values":0.9954452347188375,"id":"ra"},
{"values":0.1929087837708813,"id":"m2p"}]
```

A.4 Conclusion

In this appendix, we have presented a tool that implements several classes used to model/verify reconfigurability in GSPNs. This tool allows to define a set of rules each of which has left- and right-hand sides. These rules are applied to an initial configuration of a reconfigurable net, and therefor an isomorphic Markov chain is computed. Once the latter is completely constructed, we can compute several quantitative properties.

In future version of this tool, we are interested to develop an integrated tool that allows to users to model GSPNs, left- and right-hand sides of rules and plot charts of different quantitative properties.

List of Publications

[Tig+19] S. Tigane, L. Kahloul, S. Benharzallah, S. Baarir, and S. Bourekkache. "Reconfigurable GSPNs: A modeling formalism of evolvable discrete-event systems". In: *Science of Computer Programming* 183 (2019).

[Tig+18] S. Tigane, L. Kahloul, S. Bourekkache, and S. Baarir. "Extending GSPNs for the modelling, analysis and performance evaluation of dynamic systems". In: *International Journal of Critical Computer-Based Systems* 8.1 (2018), pp. 25–44.

[TKB17a] S. Tigane, L. Kahloul, and S. Bourekkache. "Generalized stochastic Petri nets with rewritable topology". In: *Proc. of International Conf. on EDiS*. 2017, pp. 1–6.

[TKB17b] S. Tigane, L. Kahloul, and S. Bourekkache. "Reconfigurable stochastic Petri nets: A new formalism for reconfigurable discrete event systems". In: *Proc. of ICMIT*. 2017, pp. 301–308.

[TKB17c] S. Tigane, L. Kahloul, and S. Bourekkache. "Reconfigurable Stochastic Petri Nets for Reconfigurable Manufacturing Systems". In: *Proc. 6th Int. Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing*. 2017, pp. 383–391.

[TKB16] S. Tigane, L. Kahloul, and S. Bourekkache. "Net rewriting system for GSPN a RMS case study". In: *Proc. International Conference on Advanced Aspects of Software Engineering (ICAASE)*. 2016, pp. 38–45.

Bibliography

[Abr05] J.-R. Abrial. *The B-book: assigning programs to meanings*. Cambridge University Press, 2005.

[AS87] B. Alpern and F. B. Schneider. "Recognizing safety and liveness". In: *Distributed Computing* 2.3 (1987), pp. 117–126.

[Amp14] E. G. Amparore. "A New GreatSPN GUI for GSPN Editing and CSLTA Model Checking". In: *Quantitative Evaluation of Systems*. Ed. by G. Norman and W. Sanders. Cham: Springer International Publishing, 2014, pp. 170–173.

[AK76] T. Araki and T. Kasami. "Some decision problems related to the reachability problem for Petri nets". In: *Theoretical Computer Science* 3.1 (1976), pp. 85–104.

[Baa+09] S. Baarir et al. "The GreatSPN Tool: Recent Enhancements". In: *SIGMETRICS Perform. Eval. Rev.* 36.4 (Mar. 2009), pp. 4–9.

[BLO03] E. Badouel, M. Llorens, and J. Oliver. "Modeling Concurrent Systems: Reconfigurable Nets." In: *PDPTA*. 2003, pp. 1568–1574.

[BK08] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT press, 2008.

[BCK08] P. Baldan, A. Corradini, and B. König. "A framework for the verification of infinite-state graph transformation systems". In: *Information and Computation* 206.7 (2008), pp. 869–907.

[BMS16] S. Balsamo, A. Marin, and I. Stojic. "SPNPS: A Tool for Perfect Sampling in Stochastic Petri Nets". In: *Quantitative Evaluation of Systems*. Cham: Springer International Publishing, 2016, pp. 163–166.

[BK02] F. Bause and P. S. Kritzinger. *Stochastic Petri Nets: An Introduction to the Theory*. Vieweg+Teubner, 2002.

[BE16] E. Best and J. Esparza. "Existence of home states in Petri nets is decidable". In: *Information Processing Letters* 116.6 (2016), pp. 423–427.

[BB87] T. Bolognesi and E. Brinksma. "Introduction to the ISO specification language LOTOS". In: *Computer Networks and ISDN systems* 14.1 (1987), pp. 25–59.

[Bre+14] M. Brettel, N. Friederichsen, M. Keller, and M. Rosenberg. “How virtualization, decentralization and network building change the manufacturing landscape: An Industry 4.0 Perspective”. In: *Int. J. of Mechanical, Industrial Science and Engineering* 8.1 (2014), pp. 37–44.

[BHR84] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. “A Theory of Communicating Sequential Processes”. In: *J. ACM* 31.3 (June 1984), pp. 560–599.

[BS80] J. A. Buzacott and J. G. Shanthikumar. “Models for Understanding Flexible Manufacturing Systems”. In: *AIIE Transactions* 12.4 (1980), pp. 339–350.

[CBC18] M. Camilli, C. Bellettini, and L. Capra. “A High-level Petri Net-based Formal Model of Distributed Self-adaptive Systems”. In: *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*. ECSA ’18. Madrid, Spain: ACM, 2018, 40:1–40:7.

[Čap17] F. Čapkovič. “Petri Nets in Discrete-Event and Hybrid Systems Modelling, Analysing, Performance Evaluation and Control”. In: *Automation 2017*. Cham: Springer International Publishing, 2017, pp. 3–21.

[Cap17] L. Capra. “Stochastic Petri Nets with Changeable Layout”. In: *Proc. 5th World Conference on Information Systems and Technologies*. Springer International Publishing, 2017, pp. 831–840.

[CC18] L. Capra and M. Camilli. “Towards Evolving Petri Nets: a Symmetric Nets-based Framework”. In: *IFAC-PapersOnLine* 51.7 (2018), pp. 480–485.

[CL09] C. G. Cassandras and S. Lafortune. *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[Che+17] Y. Chen, Z. Li, A. Al-Ahmari, N. Wu, and T. Qu. “Deadlock recovery for flexible manufacturing systems modeled with Petri nets”. In: *Information Sciences* 381 (Mar. 2017), pp. 290–303.

[CDF91] G. Chiola, S. Donatelli, and G. Franceschinis. “GSPNs versus SPNs: what is the actual role of immediate transitions?” In: *Proceedings of the Fourth International Workshop on Petri Nets and Performance Models*. Dec. 1991, pp. 20–31.

[Chi+93] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. “Stochastic well-formed colored nets and symmetric modeling applications”. In: *IEEE Transactions on Computers* 42.11 (1993), pp. 1343–1360.

[Chr+13] G. Chryssolouris, K. Efthymiou, N. Papakostas, D. Mourtzis, and A. Pagoropoulos. “Flexibility and complexity: is it a trade-off?” In: *International Journal of Production Research* 51.23-24 (2013), pp. 6788–6802.

[CW96] E. M. Clarke and J. M. Wing. “Formal Methods: State of the Art and Future Directions”. In: *ACM Comput. Surv.* 28.4 (1996), pp. 626–643.

[CR12] S. A. da Costa and L. Ribeiro. “Verification of graph grammars using a logical approach”. In: *Science of Computer Programming* 77.4 (2012), pp. 480–504.

[CGR93] D. H. Craigen, S. L. Gerhart, and T. J. Ralston. *An International Survey of Industrial Applications of Formal Methods. Volume 2. Case Studies*. Tech. rep. NAVAL RESEARCH LAB WASHINGTON DC, 1993.

[DKS09] N. J. Dingle, W. J. Knottenbelt, and T. Suto. “PIPE2: A Tool for the Performance Evaluation of Generalised Stochastic Petri Nets”. In: *SIGMETRICS Perform. Eval. Rev.* 36.4 (2009), pp. 34–39.

[DFS98] C. Dufourd, A. Finkel, and P. Schnoebelen. “Reset nets between decidability and undecidability”. In: *Automata, Languages and Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 103–115.

[EHP06] H. Ehrig, K. Hoffmann, and J. Padberg. “Transformations of Petri nets”. In: *Electronic Notes in Theoretical Computer Science* 148.1 (2006), pp. 151–172.

[EP04] H. Ehrig and J. Padberg. “Graph Grammars and Petri Net Transformations”. In: *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 496–536.

[EN94] J. Esparza and M. Nielsen. “Decidability issues for Petri nets”. In: *Petri nets newsletter* 94 (1994), pp. 5–23.

[Fru86] D. de Frutos-Escrig. “Decidability of home states in place transition systems”. In: *Report of Dpto. Informatica y Automatica. Univ. Complutense de Madrid* (1986).

[GG13] H. Garavel and S. Graf. *Formal Methods for Safe and Secure Computers Systems*. Federal Office for Information Security, Bonn (Germany), 2013.

[GV01] C. Girault and R. Valk. *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications*. Berlin, Heidelberg: Springer-Verlag, 2001.

[Gra80] J. Grabowski. “The decidability of persistence for vector addition systems”. In: *Information Processing Letters* 11.1 (1980), pp. 20–23.

[Hax10] A. E. Haxthausen. “An introduction to formal methods for the development of safety-critical applications”. In: *DTU Informatics Technical University of Denmark* (2010).

[HEM05] K. Hoffmann, H. Ehrig, and T. Mossakowski. “High-Level Nets with Nets and Rules as Tokens”. In: *Applications and Theory of Petri Nets 2005*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 268–288.

[JEB16] K. Jackson, K. Efthymiou, and J. Borton. “Digital Manufacturing and Flexible Assembly Technologies for Reconfigurable Aerospace Production Systems”. In: *Procedia CIRP* 52 (2016). The Sixth International Conference on CARV, pp. 274–279.

[Jen13] K. Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use*. Vol. 1. Springer Science & Business Media, 2013.

[KV86] M. Kamath and N. Viswanadham. “Applications of Petri net based models in the modelling and analysis of flexible manufacturing systems”. In: *Proceedings. 1986 IEEE International Conference on Robotics and Automation*. Vol. 3. Apr. 1986, pp. 312–317.

[KM69] R. M. Karp and R. E. Miller. “Parallel program schemata”. In: *Journal of Computer and System Sciences* 3.2 (1969), pp. 147–195.

[KR06] H. Kastenberg and A. Rensink. “Model Checking Dynamic States in GROOVE”. In: *Model Checking Software*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 299–305.

[Khl+15] O. Khlifi, O. Mosbahi, M. Khalgui, and G. Frey. “GR-TNCES: New extensions of R-TNCES for modelling and verification of flexible systems under energy and memory constraints”. In: *Proc. 10th International Joint Conference on Software Technologies (ICSOFT)*. Vol. 1. IEEE. July 2015, pp. 1–8.

[Kön04] B. König. “Analysis and verification of systems with dynamically evolving structure”. Dissertation. Universität Stuttgart, 2004.

[Kön+18] B. König, D. Nolte, J. Padberg, and A. Rensink. “A Tutorial on Graph Transformation”. In: *Graph Transformation, Specifications, and Nets*. Cham: Springer, 2018, pp. 83–104.

[Kor+99] Y. Koren et al. “Reconfigurable Manufacturing Systems”. In: *CIRP Annals* 48.2 (1999), pp. 527–540.

[Kos82] S. R. Kosaraju. “Decidability of reachability in vector addition systems”. In: *STOC*. Vol. 82. ACM. 1982, pp. 267–281.

[KL18] G. Kulcsár and A. Lochau Malte and Schürr. “Graph-Rewriting Petri Nets”. In: *Graph Transformation*. Cham: Springer, 2018, pp. 79–96.

[LEO06] L. Lambers, H. Ehrig, and F. Orejas. “Conflict Detection for Graph Transformation with Negative Application Conditions”. In: *Graph Transformations*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 61–76.

[Lam92] J. Lambert. “A structure to decide reachability in Petri nets”. In: *Theoretical Computer Science* 99.1 (1992), pp. 79–104.

[Las+14] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann. “Industry 4.0”. In: *Business & Information Systems Engineering* 6.4 (2014), pp. 239–242.

[Lat+18] J.-I. Latorre-Biel, J. Faulín, A. A. Juan, and E. Jiménez-Macías. “Petri Net Model of a Smart Factory in the Frame of Industry 4.0”. In: *IFAC-PapersOnLine* 51.2 (2018), pp. 266–271.

[LDM09] J. Li, X. Dai, and Z. Meng. “Automatic Reconfiguration of Petri Net Controllers for Reconfigurable Manufacturing Systems With an Improved Net Rewriting System-Based Approach”. In: *IEEE Transactions on Automation Science and Engineering* 6.1 (2009), pp. 156–167.

[LDM05] J. Li, X. Dai, and Z. Meng. “Improved net rewriting systems-based rapid reconfiguration of Petri net logic controllers”. In: *31st Annual Conference of IEEE Industrial Electronics Society, 2005. IECON 2005.* IEEE. 2005.

[LDM08] J. Li, X. Dai, and Z. Meng. “Improved net rewriting system-based approach to model reconfiguration of reconfigurable manufacturing systems”. In: *The International Journal of Advanced Manufacturing Technology* 37.11-12 (2008), pp. 1168–1189.

[Li+09] J. Li, X. Dai, Z. Meng, J. Dou, and X. Guan. “Rapid design and reconfiguration of Petri net models for reconfigurable manufacturing cells with improved net rewriting systems and activity diagrams”. In: *Computers & Industrial Engineering* 57.4 (2009), pp. 1431–1451.

[Li+08] J. Li, X. Dai, Z. Meng, and L. Xu. “Improved net rewriting system-extended Petri net supporting dynamic changes”. In: *Journal of Circuits, Systems, and Computers* 17.06 (2008), pp. 1027–1052.

[Liu+18] G. Liu, P. Li, Z. Li, and N. Wu. “Robust Deadlock Control for Automated Manufacturing Systems With Unreliable Resources Based on Petri Net Reachability Graphs”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2018), pp. 1–15.

[LO04a] M. Llorens and J. Oliver. “Introducing Structural Dynamic Changes in Petri Nets: Marked-Controlled Reconfigurable Nets”. In: *Proc. of Second International Conference on Automated Technology for Verification and Analysis*. Springer, 2004, pp. 310–323.

[LO04b] M. Llorens and J. Oliver. “Structural and dynamic changes in concurrent systems: Reconfigurable Petri nets”. In: *IEEE Transactions on Computers* 53.9 (2004), pp. 1147–1158.

[LZB15] F. Long, P. Zeiler, and B. Bertsche. “Potentials of coloured Petri nets for realistic availability modelling of production systems in Industry 4.0”. In: *Proceedings of the ESREL 2015 Conference*. Vol. 7. 2015.

[LZB17] F. Long, P. Zeiler, and B. Bertsche. “Modelling the flexibility of production systems in Industry 4.0 for analysing their productivity and availability with high-level Petri nets”. In: *IFAC-PapersOnLine* 50.1 (2017). 20th IFAC World Congress, pp. 5680–5687.

[Mar+94] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. 1st. New York, NY, USA: John Wiley & Sons, Inc., 1994.

[May84] E. Mayr. “An Algorithm for the General Petri Net Reachability Problem”. In: *SIAM Journal on Computing* 13.3 (1984), pp. 441–460.

[May81] E. Mayr. “Persistence of vector replacement systems is decidable”. In: *Acta Informatica* 15.3 (1981), pp. 309–318.

[MD97] H. D. Meer and O. R. Dusterhoft. “Controlled stochastic Petri nets”. In: *Proc. of The Sixteenth Symposium on Reliable Distributed Systems, 1997.* IEEE, Oct. 1997, pp. 18–25.

[Mic01] D. Michel. *Les réseaux de Petri: Modèles fondamentaux.* Ouvrage collectif sous la direction de Michel Diaz. Hermes Science Publications, 2001.

[Möl16] D. P. F. Möller. “Digital Manufacturing/Industry 4.0”. In: *Guide to Computing Fundamentals in Cyber-Physical Systems: Concepts, Design Methods, and Applications*. Cham: Springer International Publishing, 2016, pp. 307–375.

[Mül81] H. Müller. “On the reachability problem for persistent vector replacement systems”. In: *Parallel Processes and Related Automata*. Springer, 1981, pp. 89–104.

[Mur89] T. Murata. “Petri nets: Properties, analysis and applications”. In: *Proceedings of the IEEE* 77.4 (1989), pp. 541–580.

[PK18] J. Padberg and L. Kahloul. “Overview of Reconfigurable Petri Nets”. In: *Graph Transformation, Specifications, and Nets*. Cham: Springer International Publishing, 2018, pp. 201–222.

[Pet77] J. L. Peterson. “Petri Nets”. In: *ACM Comput. Surv.* 9.3 (1977), pp. 223–252.

[Pet81] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.

[Pet62] C. A. Petri. “Kommunikation mit automaten”. 1962.

[Ram73] C. Ramchandani. “Analysis of asynchronous concurrent systems by timed Petri nets.” PhD thesis. Massachusetts Institute of Technology, 1973.

[Rec+04] L. Recalde, M. Silva, J. Ezpeleta, and E. Teruel. “Petri Nets and Manufacturing Systems: An Examples-Driven Tour”. In: *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 742–788.

[Rei12] W. Reisig. *Petri nets: An introduction*. Vol. 4. Springer Science & Business Media, 2012.

[Ren08] A. Rensink. “Explicit State Model Checking for Graph Grammars”. In: *Concurrency, Graphs and Models*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 114–132.

[RSV04] A. Rensink, Á. Schmidt, and D. Varró. “Model Checking Graph Transformations: A Comparison of Two Approaches”. In: *Graph Transformations*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 226–241.

[Reu90] C. Reutenauer. *The Mathematics of Petri Nets*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1990.

[RK15] I. Rodhe and M. Karresand. *Overview of Formal Methods in Software Engineering*. Swedish Defence Research Agency, Stockholm, 2015.

[ST96] M. Silva and E. Teruel. “A systems theory perspective of discrete event dynamic systems: The Petri net paradigm”. In: *Symposium on Discrete Events and Manufacturing Systems. CESA*. Vol. 96. 1996, pp. 1–12.

[SV90] M. Silva and R. Valette. “Petri nets and flexible manufacturing”. In: *Advances in Petri Nets 1989*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 374–417.

[Sim+18] E. Simon, J. Oyekan, W. Hutabarat, A. Tiwari, and C. Turner. “Adapting Petri nets to DES: stochastic modelling of manufacturing systems”. In: *International Journal of Simulation Modelling* 17.1 (2018), pp. 5–17.

[SM83] I. Suzuki and T. Murata. “A method for stepwise refinement and abstraction of Petri nets”. In: *J. of Computer and System Sciences* 27.1 (1983), pp. 51–76.

[TKB16] S. Tigane, L. Kahloul, and S. Bourekkache. “Net rewriting system for GSPN a RMS case study”. In: *Proc. International Conference on Advanced Aspects of Software Engineering (ICAASE)*. 2016, pp. 38–45.

[TKB17a] S. Tigane, L. Kahloul, and S. Bourekkache. “Generalized stochastic Petri nets with rewritable topology”. In: *Proc. of International Conf. on EDiS*. 2017, pp. 1–6.

[TKB17b] S. Tigane, L. Kahloul, and S. Bourekkache. “Reconfigurable stochastic Petri nets: A new formalism for reconfigurable discrete event systems”. In: *Proc. of ICMIT*. 2017, pp. 301–308.

[Tig+19] S. Tigane, L. Kahloul, S. Benharzallah, S. Baarir, and S. Bourekkache. “Reconfigurable GSPNs: A modeling formalism of evolvable discrete-event systems”. In: *Science of Computer Programming* 183 (2019).

[TKB17c] S. Tigane, L. Kahloul, and S. Bourekkache. “Reconfigurable Stochastic Petri Nets for Reconfigurable Manufacturing Systems”. In: *Proc. 6th Int. Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing*. 2017, pp. 383–391.

[Tig+18] S. Tigane, L. Kahloul, S. Bourekkache, and S. Baarir. “Extending GSPNs for the modelling, analysis and performance evaluation of dynamic systems”. In: *International Journal of Critical Computer-Based Systems* 8.1 (2018), pp. 25–44.

[Val78a] R. Valk. “On the computational power of extended Petri nets”. In: *International Symposium on Mathematical Foundations of Computer Science*. Springer. 1978, pp. 526–535.

[Val78b] R. Valk. “Self-modifying nets, a natural extension of Petri nets”. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 1978, pp. 464–476.

[Val98] R. Valk. “Petri nets as token objects”. In: *International Conference on Application and Theory of Petri Nets*. Springer. 1998, pp. 1–24.

[Val01] R. Valk. “Concurrency in communicating object Petri nets”. In: *Concurrent Object-Oriented Programming and Petri Nets*. Springer, 2001, pp. 164–195.

[Val04] R. Valk. “Object Petri nets”. In: *Lectures on Concurrency and Petri Nets*. Springer, 2004, pp. 819–848.

[Vog97] W. Vogler. “Partial order semantics and read arcs”. In: *Mathematical Foundations of Computer Science 1997*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 508–517.

[Woo+09] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald. “Formal Methods: Practice and Experience”. In: *ACM Comput. Surv.* 41.4 (2009), 19:1–19:36.

[Xu12] X. Xu. “From cloud computing to cloud manufacturing”. In: *Robotics and Computer-Integrated Manufacturing* 28.1 (2012), pp. 75–86.

[You+18] D. You et al. “Liveness Enforcement for a Class of Petri Nets via Resource Allocation”. In: *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. Oct. 2018, pp. 4369–4374.

[ZBV08] M. Zaslavskiy, F. Bach, and J.-P. Vert. “A path following algorithm for the graph matching problem”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.12 (2008), pp. 2227–2242.

[Zha+13] J. Zhang, M. Khalgui, Z. Li, O. Mosbahi, and A. M. Al-Ahmari. “R-TNCES: A novel formalism for reconfigurable discrete event control systems”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 43.4 (2013), pp. 757–772.