Numerical and statistical approaches for model checking of stochastic processes

Hilal Djafri

2012 DENS 0025

THÈSE DE DOCTORAT
DE L'ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

Présentée Par
Monsieur Hilal Djafri

Pour obtenir le grade de
DOCTEUR DE L'ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

Domaine
Informatique

Sujet de la thèse :

Numerical and Statistical Approaches for Model Checking of Stochastic Processes

Soutenue le 19 juin 2012 devant le jury composé de

Hind Castel	Maître de Conférences Telecom SudParis	Examinateur
Giuliana Franceschinis	Professeur Università del Piemonte Orientale Italie	Rapporteur
Stefan Haar	Directeur de Recherche INRIA Saclay - Ile-de-France	Examinateur
Serge Haddad	Professeur ENS de Cachan	Directeur de thèse
Patrice Moreaux	Professeur Université de Savoie	Rapporteur
Nihal Pekergin	Professeur Université Paris-Est Créteil Val de Marne	Examinateur

Abstract

We propose in this thesis several contributions related to the quantitative verification of systems. This discipline aims to evaluate functional and performance properties of a system. Such a verification requires two ingredients: a formal model to represent the system and a temporal logic to express the desired property. Then the evaluation is done with a statistical or numerical method.

The spatial complexity of numerical methods which is proportional to the size of the state space of the model makes them impractical when the state space is very large. The method of stochastic comparison with censored Markov chains is one of the methods that reduces memory requirements by restricting the analysis to a subset of the states of the original Markov chain. In this thesis we provide new bounds that depend on the available information about the chain.

We introduce a new quantitative temporal logic named Hybrid Automata Stochastic Logic (HASL), for the verification of discrete event stochastic processes (DESP). HASL employs Linear Hybrid Automata (LHA) to select prefixes of relevant execution paths of a DESP. LHA allows rather elaborate information to be collected *on-the-fly* during path selection, providing the user with a powerful mean to express sophisticated measures. In essence HASL provides a unifying verification framework where temporal reasoning is naturally blended with elaborate reward-based analysis. We have also developed COSMOS, a tool that implements statistical verification of HASL formulas over stochastic Petri nets.

Flexible manufacturing systems (FMS) have often been modeled by Petri nets. However the modeler should have a good knowledge of this formalism. In order to facilitate such a modeling we propose a methodology of compositional modeling that is application oriented and does not require any knowledge of Petri nets by the modeler.

Résumé

Les systèmes matériels et logiciels sont de plus en plus présents dans la vie quotidienne mais aussi de plus en plus complexes. Le concepteur ou l'utilisateur d'un système se pose deux questions principales : *le système fait-il ce qu'il est censé faire ? Combien de temps le système prend-il pour exécuter une tâche particulière ?* La première question, consiste à vérifier si le système satisfait une propriété fonctionnelle. Les techniques de vérification formelle [Wan04] apportent une réponse à cette question. Dans la deuxième question, il s'agit d'évaluer un paramètre ou un indice de performance du système. La discipline de l'évaluation des performances [HLR00] permet d'effectuer cette tâche. Pendant longtemps, les deux disciplines se sont développées indépendamment l'une de l'autre. Mais quelques difficultés commencent à apparaître dans les deux disciplines. En effet, de nos jours, l'évaluation des performances a besoin d'exprimer des indices plus élaborés. Les techniques de vérification peuvent exprimer ces indices rigoureusement et procèdent à leur évaluation automatiquement. D'autre part, les techniques de vérification doivent être mises à jour pour aborder les systèmes probabilistes. Clairement, les deux disciplines sont complémentaires et leur unification engendre ce qui est appelé La *vérification quantitative* [Kwi07].

Plusieurs approches ont été développées dans le contexte de la vérification formelle : tests, simulation, preuve de programmes et model checking. Depuis le travail de Emerson et Clarke [EC80], le model checking a reçu un intérêt particulier de la part de la communauté scientifique à la fois sur les plans théorique et pratique. Étant donnés un modèle et une propriété, la technique du model checking explore la totalité de l'espace d'état du modèle pour vérifier si la propriété est satisfaite par le modèle. Si la propriété n'est pas satisfaite un contre exemple est généré. L'utilisation du model checking présente deux avantages principaux par rapport aux autres méthodes de la vérification formelle : (1) la procédure est complètement automatique, (2) un contre exemple est généré si jamais le système échoue à satisfaire la propriété en question [BK08].

La première étape pour la vérification d'un système à travers la méthode du model checking consiste à représenter le système par un modèle formel, en général, un *système de transition* qui n'est qu'un graphe orienté où les nœuds

représentent les états du systèmes et les arcs représentent les changements d'états du système. D'autres informations peuvent être rajoutées sur les nœuds ou/et arcs [BK08]. Dans la deuxième étape, la propriété que l'on souhaite vérifier est exprimée à l'aide d'une logique temporelle adéquate. Finalement, à la troisième, un algorithme de model checking évalue la propriété sur le modèle. En pratique, cette étape consiste à exécuter un model checker. Ce dernier renvoie : *oui* si le modèle satisfait la propriété ou *non* si la propriété n'est pas satisfaite avec éventuellement un contre exemple.

L'évaluation des performances a connu un développement rapide avec l'avènement des ordinateurs et des réseaux de télécommunication. Cette discipline a comme objectif principal le calcul de certains paramètres de performance du système. Le temps moyen de réponse d'un réseau de télécommunication est un exemple de paramètres que l'on souhaite souvent évaluer. La méthodologie de la discipline consiste à calculer la distribution stationnaire ou transitoire du système. Ces dernières seront exploitées pour calculer les paramètres désirés. Pour ce faire, la discipline fait appel aux méthodes analytiques ou à la simulation à événements discrets. Quant aux modèles formels les plus répandus, on cite : la théorie des files d'attente [Ste09], les réseaux d'automates stochastiques [Pla85] et les réseaux de Petri stochastiques [ABC+95].

Vérification Quantitative

Pour appliquer la vérification quantitative sur un système stochastique, il est indispensable de représenter ce système par un modèle formel. Les chaines de Markov sont certainement les plus connues parmi les modèles stochastiques grâce à leur simplicité, leur capacité de modélisation et la richesse de la littérature qui leur a été consacrée par la communauté scientifique. Nous distinguons les chaînes de Markov à temps discret (DTMC) et les chaînes de Markov à temps continu (CTMC). La modélisation d'un système directement par une chaîne de Markov s'avère quelquefois fastidieuse voire impossible lorsque la taille de l'espace d'état dépasse quelque dizaines d'états. Il est alors nécessaire de faire appel aux formalismes haut niveaux tels que : Les réseaux de Petri stochastiques, les réseaux d'automates stochastiques ou les algèbres de processus stochastiques [HK01].

La propriété à évaluer doit être exprimée par une logique temporelle probabiliste. Plusieurs logiques ont été proposées dans la littérature. Elles sont en général une extension de deux logiques principales : la logique temporelle linéaire (LTL) [Pnu77] et logique temporelle arborescente (CTL) [CE82, QS82]. PLTL [Var85] (resp. PCTL [HJ94, HS86]) est une extension probabiliste de LTL (resp. CTL) pour les DTMCs. La logique stochastique continue (CSL) [ASVB96, ASVB00] a été définie comme une extension probabiliste de CTL pour les CTMCs. CSL (resp. PCTL) a été ensuite étendue à CSRL [BHHK00a] (resp. PRCTL [AHK03])

pour prendre en compte les récompenses dans les CTMCs (resp. DTMCc). asCSL [BCH+07] est une autre extension de CSL qui permet de spécifier des CTMCs avec des actions et des états étiquetés. Dans la logique CSL^{TA} [DHS09] la formule s'exprime comme un automate temporisé déterministe à horloge unique. Il a été démontré que CSL^{TA} est strictement plus expressive que CSL et asCSL. Une autre logique qui utilise les automates pour exprimer la formule appelée DTA [CHKM09] a été introduite. Dans cette logique l'automate est à horloges multiples, ce qui fait de DTA plus expressive que CSL^{TA}.

Nous distinguons deux familles de méthodes du modèle checking stochastique. La première regroupe les méthodes dites numériques. Cette famille de méthode fournit la valeur *exacte* de la probabilité recherchée ce qui est considéré comme leur principal atout. Les techniques d'analyse numérique sont utilisées pour cette catégorie. Néanmoins, plusieurs inconvénients surgissent. Le premier est dû à l'explosion de l'espace d'état ce qui rend l'analyse des grands systèmes difficile ou impossible. Le deuxième inconvénient est liée à la nature du processus stochastique engendré par le modèle. En effet, ces méthodes ne peuvent pas traiter les processus markoviens ou semi-markoviens.

Les méthodes statistiques est la deuxième famille du modèle checking stochastique. Leur principe consiste à générer un nombre suffisant de trajectoires ensuite d'estimer la probabilité désirée. On en distingue deux méthodes : (1) *L'estimation*, qui renvoie une estimation statistique de probabilité recherchée tout en construisant un intervalle (appelé *intervalle de confiance*) contenant la valeur exacte, pas surement mais avec une certaine probabilité ; (2) Le *test d'hypothèse,* dont le but est de comparer la probabilité recherchée, *sans la calculer* à un seuil prédéfini.

Les méthodes statistiques utilisent des techniques issues de la statistique mathématique telles que l'estimation et le test d'hypothèse mais aussi la simulation à événements discrets. Ces méthodes présentent plusieurs avantages : elles ne sont pas gourmandes en mémoire, leur complexité spatiale est proportionnelle au nombre de composants et non pas à l'espace d'état, il est même possible de traiter des systèmes infinis. L'autre avantage est qu'elle ne sont pas restreintes aux modèles markoviens, il suffit de pouvoir représenter le modèle par un système stochastique à événements discrets.

Toutefois ces méthodes ne sont pas sans inconvénients. En effet, l'obtention de résultats avec un niveau de confiance élevé et un intervalle étroit nécessite un temps de calcul très important. D'autre part, l'évaluation de formules stationnaires nécessite un traitement spécifique.

Questions Ouvertes

Nous décrivons dans cette section les trois questions ouvertes que nous avons abordées dans notre travail.

1. L'explosion de l'espace d'état dans les méthodes numériques. Il y a eu plusieurs approches qui ont proposées pour apporter solution à ce problème. Ces approches ne résolvent pas le problème définitivement, mais contribuent de manière significative à réduire la complexité spatiale. Certaines méthodes interviennent en amont, citons en quelques-unes : les méthodes d'agrégation d'états dans les chaînes de Markov [BHHK00b, BHHK03b], le produit tensoriel dans les les réseaux d'automates stochastiques [BKKT02], les formes produits dans les réseaux de Petri stochastiques [HMSM05] ou l'exploitation de la comparaison stochastique dans les chaînes de Markov [PY05]. Une autre méthode qui intervient en aval consiste à utiliser les diagrammes de décision binaires [CFM+97].

2. Les limitations dans les logiques existantes. La première limitation est dûe au manque d'approche unificatrice qui peut à la fois effectuer des tâches de model checking et d'évaluation des performances dans un même formalisme. La deuxième limitation est liée à l'expressivité de ces logiques. En effet, ces logique peuvent effectuer uniquement deux type d'évaluations : (1) Une évaluation booléenne qui dépends de la validité de la formule sur le chemin et (2) une somme cumulée exprimée sur les récompenses des actions et états. Mais, il n'est pas possible d'effectuer certaines opérations compliquées mais utiles telles que le minimum, le maximum, l'intégration ou la valeur moyenne sur un chemin.

3. La difficulté à exploiter des formalismes haut-niveau par le modélisateur. Les formalismes haut niveau sont des modèles formels qui ne correspondent pas à des approches pratiques des problème traités. Le modélisateur préfère toujours les formalismes qui correspondent à ses cas pratiques. De plus, un formalisme intuitif contribue à réduire les erreurs au cours de la phase de modélisation.

Les Contributions

Nous proposons dans le cadre de cette thèse plusieurs contributions

La **première contribution** rentre dans le cadre de la réduction de l'explosion de l'espace d'état pour les méthodes numériques. L'objectif est de construire des bornes stochastiques pour les chaînes de Markov censurées (CMC). Une CMC n'observe qu'un sous ensemble de l'espace d'état de la chaîne originale. Les CMCs sont très utiles quand on veux traiter une chaîne de Markov avec un espace d'état très large ou quand cette chaîne n'est connue que partiellement. En effet, la construction de bornes sur la chaîne censurée permet de borner certaines mesures dans la chaîne originale. La technique a été déjà introduite dans [FPY07a] où un algorithme appelé DPY a été proposé dans le but de construire des bornes pour une CMC. Dans notre travail, nous montrons que DPY est optimal et nous proposons plusieurs schémas de bornes qui dépendent de l'information partielle disponible ou que l'on veuille exploiter.

Dans la deuxième contribution nous introduisons une nouvelle logique ap-

plée "Hybrid Automata Stochastic Logic (HASL)". Cette logique rentre dans le cadre du model checking statistique. Une formule dans cette logique est alors exprimée à l'aide de deux composantes : un automate hybride linéaire et une expression définie sur les variables de l'automate. Avec HASL, on peut vérifier une large classe de modèles stochastiques appelée Discrete Events Stochastic Process (DESP). HASL élimine les limitations d'autres logiques. Elle unifie dans le même formalisme plusieurs disciplines : model checking, sûreté de fonctionnement, évaluation des performances. D'autre part, HASL étend l'expressivité des logiques existantes. En effet, il est possible d'effectuer certaines opérations sur les variables à la volée et pendant la génération d'une trajectoire, typiquement, le minimum, le maximum, l'intégral ou la valeur moyenne. Enfin, puisque nous utilisons une approche statistique, la propriété sans mémoire n'est pas requise, donc, on peut (en principe) traiter n'importe quelle distribution de temps. Nous avons aussi conçu un outil appelé COSMOS qui évalue une formule HASL sur un réseau de Petri généralisé.

Dans la **troisième contribution** nous proposons une nouvelle approche compositionnelle pour modéliser les ateliers flexibles (FMS) en utilisant les réseaux de Petri. Notre choix de FMS est dû à leur importance dans l'industrie. Dans notre approche un FMS est modélisé par composants en spécifiant les classes de composants à utiliser. Ce qui permet un modélisateur de construire son modèle comme s'il construit un FMS réel à l'usine. Il aura à manipuler des unités de chargement et de transport ainsi que des machines. Ainsi, la phase de modélisation consiste sélectionner les composants qui se trouvent dans une boîte à outils prédéfinie. Ensuite le modélisateur procèdera à l'assemblage des composants via leurs interfaces.

Remerciements

Je tiens tout d'abord à remercier mon directeur de thèse Serge Haddad pour m'avoir encadré tout au long de cette thèse, pour ses conseils si pertinents et si précieux et pour nos discussions très bénéfiques.

I would like to thank Giuliana Franceschinis and Patrice Moreaux for giving me the honor of reviewing my thesis and for their comments and constructive criticism. Je remercie aussi Hind Castel, Stefan Haar et Nihal Pekergin d'avoir accepté de participer au jury de ma thèse.

Je présente aussi mes remerciements à tous mes co-auteurs qui m'ont fait l'honneur de travailler avec moi : Ana Bušić, Paolo Ballarini, Jean-Michel Founeau, Marie Duflot et Nihal Pekergin.

I also thank Susanna Donatelli and her team for accepting me for a period of internship, for their availability and their warm welcome.

Mes remerciements vont aussi à tous les membres du laboratoire LSV.

Enfin, j'adresse mes remerciements à toute ma famille pour son soutien indéfectible.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Software and hardware systems are increasingly present in every day life but also increasingly complex. The designer or user of a system asks two main questions: *The system does what it is supposed to do? How long does the system take to execute a particular task?* The first question, consists of verifying if the system satisfies a functional property. Techniques of formal verification [Wan04] can respond to such questioning. The second question is to evaluate a parameter or a performance index of the system. The discipline of performance evaluation [HLR00] is related to this task. For many years the two disciplines have been developed independently from each other. But some weaknesses start to appear in both disciplines. Indeed, nowadays the performance evaluation requires more elaborate indices. Verification techniques can express such complex indices rigorously and evaluate them automatically. On the other hand, verification techniques must be updated to tackle with probabilistic systems. So the two disciplines are complementary, and their unification leads what is called the *quantitative verification* [Kwi07].

Many approaches are developed for formal verification such as: test, simulation, program proving and model checking. Since the work of Emerson and Clarke [EC80], model checking has received particular interest from the scientific community both on the theoretical and practical levels. Given a model and a property, the model checking technique explores the full state space of the model to check whether the given property is satisfied by the model. If the property is not satisfied counter examples will be generated. There are two main advantages of using model checking compared to other formal verification methods: it is fully automatic and it provides a counter example whenever the system fails to satisfy a given property [BK08].

The first step to check a system by mean of a model checking method consists of representing the system by a formal model, in general, a *transition system (TS)*. A TS is basically a directed graph where nodes represent states and edges

represent the state changes of the system. Additional information can be added on the nodes and/or the edges [BK08]. Many types of TS are defined in the literature. In the second step of a model checking procedure, the property to be checked is formally expressed using an adequate temporal logic. In the third step an algorithm of model checking evaluates the property on the model. In practice, this step consists of running a model checker. The later produces the following output: *yes* if the model satisfies the property and *no* otherwise with possibly a counter-example.

Performance evaluation has been developed especially with the rise of computer and telecommunications networks. It aims to calculate the so-called performance parameters of a system. An example of a performance parameter is the average time of response of a telecommunication network. In performance evaluation the steady-state or transient distribution of the system is computed and then used to evaluate the desired parameter. Analytic methods and simulation are the two main techniques of performance evaluation. Formal models such as, queuing networks [Ste09], stochastic automata networks (SAN) [Pla85] and stochastic Petri nets (SPN) [ABC$^+$95] are among the most widespread formalisms.

Quantitative Verification

To apply quantitative verification on a stochastic system, it is necessary to represent such system by a formal model. Markov chains models are certainly the most known stochastic models for their simplicity, their modeling power and the wealth of literature that has been devoted to them by the scientific community. We distinguish the discrete time Markov chain (DTMC) and continuous time Markov chains (CTMC). Direct modeling of a system by a Markov chain can be tedious or even impossible when the state space of this system exceeds a few dozen states. So one describes such systems by high level formalisms such as: stochastic Petri nets, stochastic automata networks or stochastic process algebras [HK01].

The property to be evaluated should be expressed by a probabilistic temporal logic. Many stochastic logics were introduced in the literature. In general, they are an extension of two main temporal logics: Linear Temporal Logic (LTL) [Pnu77] and Computational Tree Logic (CTL) [CE82, QS82]. LTL is path-based while CTL is state-based. PLTL [Var85] (resp. PCTL [HJ94, HS86]) is a probabilistic extension of LTL (resp. CTL) for DTMCs. The Continuous Stochastic Logic (CSL) [ASVB96, ASVB00] was defined as a probabilistic extension of CTL for CTMCs. CSL (resp. PCTL) is then extended to CSRL [BHHK00a] (resp. PRCTL [AHK03]) to take into account rewards for CTMCs (resp. DTMCc). Another extension of CSL named asCSL [BCH$^+$07], for specification of CTMCs with both action-and state-labels. In the logic CSL^{TA} [DHS09], the formula is

expressed as deterministic timed automaton with a single clock. It is shown that CSL^{TA} is strictly more expressive than CSL and asCSL. Another logic using timed automaton called DTA [CHKM09] appears. In DTA the automaton is a multiple clock one, which makes DTA more expressive than CSL^{TA}.

There are two families of methods for stochastic model checking. The first includes the so-called numerical methods. It provides the exact value of the probability we search which is considered as their main advantage. These methods use numerical analysis techniques. However, they have several drawbacks. The first one is due to the explosion of the state space which makes the analysis of large systems difficult or impossible. The second drawback concerns management of the trace relevant information. In fact, these methods can only handle with markovian and semi-markovian models.

Statistical methods represent the second family of stochastic model checking methods. Their principle is to generate a sufficient number of trajectories and estimate the desired probability. There are two families of methods. The estimation method can estimate the probability and build an interval (called confidence interval) containing the actual value not surely but with a certain probability called confidence level. The method of hypothesis testing proceed to a statistical test that compares the probability that the property is satisfied to a predefined threshold without computing the considered probability.

These methods make use of mathematical statistical techniques such as estimation and hypothesis testing and also the discrete events simulation. The interest of statistical methods is that they are not greedy in memory, their spatial complexity is proportional to the number of system components and not to the state space. One can even handle infinite systems. The other advantage, is that they are not restricted to markovian models, it is sufficient that the model can be represented as a stochastic discrete event system. Unfortunately these methods also have disadvantages. Indeed, obtaining results with high confidence and a reduced confidence interval requires lengthy computation time. On the other hand, evaluating steady state formulas requires a specific treatment.

Open Issues

Here we describe three of the main open issues that we have addressed in our work.

1. State space explosion in numerical methods. There were several approaches that have been proposed to address that problem. They do not provide a final solution, but they significantly reduce the space complexity. Some approaches are applied upstream. Here are some of the numerous proposed methods: the aggregation of states in Markov chains [BHHK00b, BHHK03b], tensor product in stochastic automata networks [BKKT02], product form in stochastic Petri nets [HMSM05], the exploitation of stochastic comparisons [PY05]. An-

other approach is applied downstream. It consists of using Multi-Binary Decision Diagrams (MTBDDs) [CFM+97].

2. Limitations of existing logics. the first limitation of existing logics is the lack of a unifying approach than can do model-checking and performance evaluation within the same formalism. The second limitation is a problem of expressiveness. Indeed these logics, for a given path, can only do two kinds of evaluation: (1) a boolean evaluation that depends on the validity of the formula over the path and (2) an accumulated sum expressed as state or action rewards. But it not possible to do some complicated but useful operations like the minimum, the maximum, the integration or the mean value over a path.

3. Difficulty of adoption of high-level formalism by modelers. High-level formalisms are formal models. They do not correspond to practical approaches to problems. A modeler always prefers formalisms that ressembles to its own practice. Furthermore an "intuitive" formalism contributes to reduce errors during the modeling phase.

Contributions

We present several contributions in this thesis.

The **first contribution** is within the scope of reducing state space explosion in numerical methods. The goal is to establish stochastic bounds for censored Markov chains. Such a chain observes only a subset of states in the original chain. Censored chains are useful when faced to a very large chain or when the system is only partially known. Indeed, the construction of bounds for a censored chain allows to obtain bounds on measures of the original chain. This technique was already proposed in [FPY07a]. In that work an algorithm was proposed, called DPY, to build bounds for censored Markov chains. Here we show first that DPY is optimal and we propose several alternative schemes of bounds depending on the (partial) information related to the chain.

The **second contribution** introduces a new logic called Hybrid Automata Stochastic Logic (HASL). This statistical logic consists of two components: a linear hybrid automaton and an expression defined on the variables of the automaton. Such a logic allows to verify a large class of stochastic model called Discrete Events Stochastic Process. HASL eliminates the limitations of other logics. It unifies several disciplines: model checking, dependability and performance evaluation in a unique formalism. HASL extends the expressiveness of existing logics Such as CSL, $CSRL$ $asCSL$, CSL^{TA}, and DTA. With HASL it is possible to do some operation on variables on the fly along the generation of a path, typically the minimum, the maximum, the integration, the mean value. Finally because we use a statistical approach the markovian property is not required so (in principle) we can handle any time distribution. We have developed a tool, named COSMOS, evaluating HASL formulas on Generalized

Stochastic Petri Nets (GSPN).

The **third contribution** proposes a new compositional approach to the modeling of Flexible Manufacturing Systems (FMS) using Petri nets. The choice of FMS is due to their importance in industry. In our approach the FMS is modeled piece-wise by specifying the classes of components to be used. So the modeler builds his model as a real FMS in the factory. He deals with loading units, transporters and machines. The modeling phase consists of selecting each component from a predefined toolkit and specifying the parameters of each component. The second phase consists of assembling these components via their interface.

Organization

This thesis is divided into three parts.

- **Part I.** This part is devoted to the state of the art of stochastic models and quantitative verification. In chapter 2, we present the standard results related to the analysis of Markov chains. We also introduce stochastic Petri nets. Chapter 3 surveys quantitative verification. In particular, we present the numerical and statistical methods.

- **Part II.** This part concerns numerical methods. In chapter 4, we design stochastic bounds for censored Markov chains.

- **Part III.** This part concerns statistical methods. In chapter 5, we define and study the logic HASL. In chapter 6, we describe the COSMOS tool. We give the main interface, algorithms, internal structure and we also do some numerical experiments. In chapter 7, we detail our approach for compositional modeling of FMS.

We finally conclude and give some perspectives in chapter 8.

Part I

State of the Art

Chapter 2

Stochastic Petri Nets

2.1 Introduction

One of the main interests of Petri nets is to combine qualitative analysis (*i.e.* the property verification) and quantitative one (*i.e.* performance evaluation) [FN78, Mol81, RR98a, RR98b]. By comparison, concurrency models like process algebra [Hil96] have only recently been extended with stochastic features and if first results are promising, there are still more research about performance evaluation of stochastic Petri nets. Similarly, the usual models for performance evaluation like queueing networks [Kle75] do not include synchronization mechanisms and adding them by ad hoc constructions do not reach the generality and the simplicity of concurrency modeling by Petri nets.

Stochastic Petri nets have been introduced in a pragmatic way at the end of the seventies, in order to take benefit from the evaluation methods of Markov chains. This approach leads to immediate results but occults the semantical features underlying the definition of stochastic Petri nets and cannot be easily generalized to different probability distributions.

Here we present Stochastic Petri nets in a different way. First we introduce a general definition of discrete-event stochastic process. Then we specialize this definition and present different families of processes starting from the simplest one, Discrete-time Markov chain, to a very expressive one, Markov Renewal Process, where analysis is still possible. For every family we describe how performance evaluation can be done. We omit the programming features related to numerical computations. Indeed, these features are not specific to stochastic Petri nets and are covered by excellent books [Ste94, BGdMT98]. Once these families are introduced, we are in position to cover the different versions of stochastic Petri nets.

Afterwards we develop the key points of a stochastic semantic for Petri nets.

This includes the specification of a random variable associated with the firing delay of a transition, the choice criteria between enabled transitions, the handling of the firing degree in the samplings of the random variable associated with a transition and the memorization of the previous samplings, once the firing is performed. Then we restrict the type of distributions, which leads to stochastic processes previously studied. Among the different families of stochastic nets, Petri nets with exponential and immediate distributions, called generalized stochastic Petri nets, are considered as the standard model [ABC+95]. We indicate, for this model, how to to compute the stationary distribution based on the reachability graph (when it is finite).

The basic algorithms have a complexity of the same magnitude order as the reachability graph size for the simple models and greater for models with more general distributions. Thus the more elaborated techniques split in two families: the first one aims at obtaining a complexity smaller than the size of the graph (*e.g.* by restricting the class of Petri nets) and the second one aims at obtaining the same order of complexity than the size of the graph but for extended models.

The last section describes some of these methods in order to emphasize the diversity of the approaches.We do not detail here simulation since it is one of the topics of this thesis and will be developed later. Those covered in this chapter are:

- the research of a product form: a formula that expresses the stationary probability of a marking including the net parameters and the place marking as variables of the formula. This method illustrates the extension of a technique first applied in queueing networks.

- a resolution method for nets with an only one unbounded place. The application of this method shows that conditions on the structure of Markov chains can be naturally translated in terms of Petri nets.

- a method that takes advantage of a net decomposition based on the tensorial product of matrices.

- a method to handle phase-type stochastic Petri nets. Again this method uses tensorial product illustrating the fact that a generic method can be applied in very different contexts.

For more readings on stochastic Petri nets, we recommend book [ABC+95] and book chapters [HM09a, HM09b, HM09c].

2.2 Stochastic processes

2.2.1 A stochastic model for discrete events systems

We assume that the reader is familiar with the basic probability concepts. For more details the interested reader may consult [Fel68, Fel71, Tri82].

Notations

- $\Pr(E)$ is the probability of event E, while $\Pr(A \mid B)$ is the probability of A given B.
- The term $almost$, in an expression like *almost everywhere* or *almost surely*, means with probability 1.
- $\mathbb{R}$ (resp. $\mathbb{R}^+, \mathbb{R}^{+*}$) denotes the real numbers (resp. non negative and strictly positive reals). If x is a real, then $\lfloor x \rfloor$ denotes its integer part.
- If $E \subseteq \mathbb{R}$ then $Inf(E)$ (resp. $Sup(E)$) denotes the lower (resp. upper) bound of E.

Given a discrete event dynamic system (DES), its execution is characterized by a (possibly infinite) sequence of events $\{e_1, e_2, \ldots\}$ and associated interval of time between successive events in the sequence. Only the events can change the state of the system. Formally, the stochastic behaviour of a DES is defined by two families of random variables:

- $S_0, \ldots, S_n, \ldots$ defined over the (discrete) state space of the system, denoted as S. S_0 is the system initial state and S_n for $n > 0$ is the state after the n^{th} event. The occurrence of an event does not necessarily modify the state of the system, and therefore S_{n+1} may be equal to S_n.
- $T_0, ..., T_n, ...$ defined over $\mathbb{R}^+$, where T_0 is the time interval before the first event and T_n for $n > 0$ is the time interval between the n^{th} and the $(n+1)^{th}$ event. Please note that this interval may be null (*e.g.* a sequence of assignment instructions can be modelled as instantaneous with respect to complex data base transactions involving some input/output activity). When $T_n = 0$, one says that S_n is a *vanishing* state.

If the initial distribution of variable S_0 is concentrated on a single state s, we say that the process starts in s (*i.e.* $\Pr(S_0 = s) = 1$).

A priori there is no restriction whatsoever on the two families of random variables, but, for the stochastic processes that we shall study in the following,

we assume that a discrete event system cannot execute an infinite number of actions in a finite amount of time. that is to say:

$$\sum_{n=0}^{\infty} T_n = \infty \text{ almost surely} \tag{2.1}$$

The above property allows to define the state of the system at a given time instant. let $N(\tau)$ be the random variable defined by:

$$N(\tau) \equiv min(\{n \mid \sum_{k=0}^{n} T_k > \tau\})$$

according to equation (2.1), $N(\tau)$ is defined *almost everywhere*. As exemplified in figure 2.1, $N(\tau)$ can have jumps of size bigger than one. The state $X(\tau)$ of the system at time τ, is then simply $S_{N(\tau)}$. Observe that different stochastic processes can lead to the same family $\{X(\tau)\}_{\tau \in \mathbb{R}^+}$. This means that w.r.t a state-based semantics such processes are equivalent.

The diagram of figure 2.1 represents a possible *execution* of the process and shows the interpretation of each random variable defined above. In the execution the process is initially in state s_4, where it stays until, at time τ_0, it moves to state s_6. At time $\tau_0 + \tau_1$, the system visits, in zero time, the states s_3 and s_{12}, ending up in state s_7, where it stays for a certain amount of time. The use of $X(\tau)$ in continous time, hides the vanishing states s_3 and s_{12} visited by the process.

The performance evaluation of a discrete event system can be based on two complementary approaches:

- Analysis under transient behaviour, that is to say, the computation of performance measures which are function of the time passed since the start of the system. This kind of analysis is well suited for studying the system behaviour in the initialization phase, or for studying systems with final states. Classical applications of transient analysis can be found in the studies aimed at assessing the dependability and reliability of systems [Mey80, TMWH92].

- Analysis in steady state, that is to say, the computation of performance measures which takes only into account the stationary behaviour of the system, that may be reached after a transient initial phase.

The analysis in steady state makes sense only if such a stationary behaviour exists; a condition that can be expressed as follows, denoting $\boldsymbol{\pi}(\tau)$ the distribution of $X(\tau)$:

$$\lim_{\tau \to \infty} \boldsymbol{\pi}(\tau) = \boldsymbol{\pi} \tag{2.2}$$

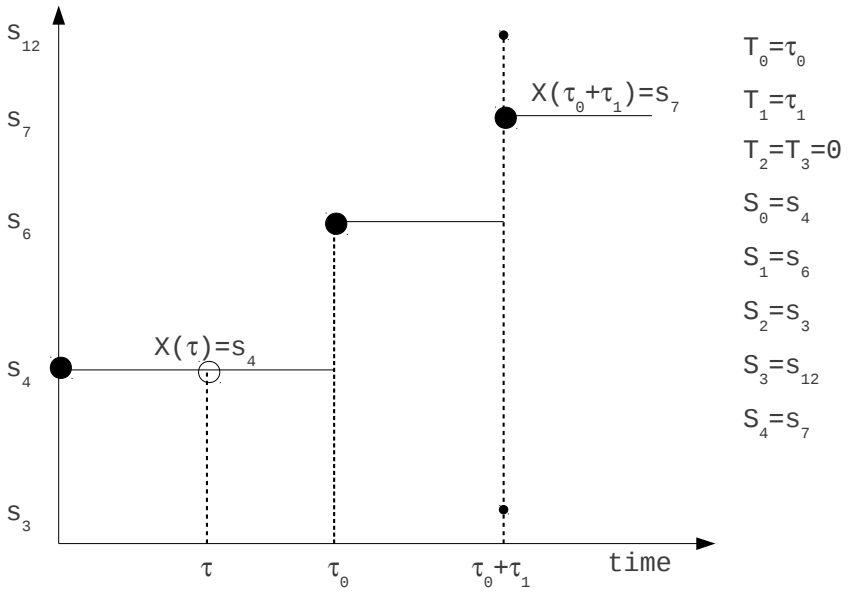


Figure 2.1: an execution of the stochastic process

where $\boldsymbol{\pi}$ is also a distribution, called the *steady-state distribution*.

The transient and steady-state distributions are the basis for the computation of *performance indices*. Examples of indices are the steady state probability that a server is up and running, the probability that at time τ a connection has been established or the mean number of clients waiting for a service. To abstract from the definition of the single performance index, we introduce the concept of *reward function*, a function f defined on the set of states of the discrete event system and with value onto $\mathbb{R}$. Given a distribution $\boldsymbol{\pi}$, the quantity $\sum_{s \in s} \boldsymbol{\pi}(s) \cdot f(s)$ represents the measure of the performance index defined by f.

If f takes values over $\{0, 1\}$, we can consider f as the definition of an *atomic proposition* ϕ which is satisfied in state s if $f(s) = 1$ and false otherwise. In the following we shall indicate with $\mathcal{P}$ the set of atomic propositions and with $s \vDash \phi$, with s a state and ϕ an atomic proposition, the fact that s verifies (or satisfies) ϕ. In this context, if $\boldsymbol{\pi}$ is a distribution, the quantity $\sum_{s \vDash \phi} \boldsymbol{\pi}(s)$ represents the measure of the index defined by ϕ.

2.2.2 Discrete time Markov chains

Presentation. The analysis of the behaviour of a general DES may be intractable or even impossible. Thus one needs to study families of simple DES. The sim-

Figure 2.2: a (finite) DTMC

plest one is the family of Discrete Time Markov Chains (DTMC). A DTMC is a stochastic process with the following characteristics:

- the time intervals T_n are constant and equal to 1.
- the next state depends only on the current state (what is called Markovian property), and the transition probability among states remains constant over time[1]:

$$\Pr(S_{n+1} = s_j \mid S_0 = s_{i_0}, ..., S_n = s_i) =$$
$$\Pr(S_{n+1} = s_j \mid S_n = s_i) = p_{ij} \equiv \mathbf{P}[i, j]$$

and we shall freely mix the two notations p_{ij} and $\mathbf{P}[i, j]$ for the transition probability.
Thus a DTMC is defined by its initial distribution $\boldsymbol{\pi}_0$ and matrix $\mathbf{P}$.

Example. Figure 2.2 represents on the right, matrix P of a DTMC with three states. On the left, a graph whose vertices are states and edges weighted by the non null transition probabilities between states is depicted. In addition to the graphical interest of this representation, analysis of this graph will provide useful information on the DTMC (see later on).

Transient and steady state behaviour of a DTMC. We now recall several classical results on the analysis of DTMC: the results will be explained in an intuitive manner, a full mathematical treatment of the topic being out of the scope of this chapter.

The transient analysis is rather simple: the change of state takes place at time instants $\{1, 2, \ldots\}$, and given an initial distribution $\boldsymbol{\pi}_0$ and the transition probability matrix $\mathbf{P}$, we have that $\boldsymbol{\pi}_n$, the distribution of X_n (*i.e.* the state of the chain at time n) can be expressed as $\boldsymbol{\pi}_n = \boldsymbol{\pi}_0 \cdot \mathbf{P}^n$, which is computed using a basic recurrence scheme.

[1] which justifies that these chains are sometimes called *time homogeneous*.

To analyze the asymptotic behaviour of a DTMC we need to investigate a bit further the DTMC behaviour, in particular we shall classify states as follows:

- A state s is said to be *transient* if the probability of returning to s after a visit is strictly less than 1. As a consequence, the probability of $\Pr(X_n = s)$ goes to zero as n tends to infinity. A state is said to be *recurrent* if it is not transient.

- A recurrent state s is said to be *null recurrent* if the mean time between two successive visits to s is infinite. Intuitively, a null recurrent state will be visited at intervals whose mean duration goes to infinity and therefore the probability of visiting s will also tends towards 0.

- A recurrent state s is *non null recurrent* if the mean time between two successive visit to s is finite. If a steady state distribution exists, then it is concentrated on the set of non null recurrent states.

Let us formalize the concept of the graph associated with a DTMC:

- the set of nodes is the set of states of the chain;

- there is an arc from s_i to s_j if $p_{ij} > 0$.

If the graph is strongly connected, *i.e.* there is a single *strongly connected component* (SCC), then the chain is said to be *irreducible*. In an irreducible DTMC, all states have the same status.

Example. Figure 2.3 represents an infinite irreducible DTMC with probability p to go "backward" and $1 - p$ to go "forward". When $p < 0.5$ all states are transient: on the long run the distribution goes "forward". When $p = 0.5$ all states are null recurrent : on the long run the distribution is more and more equally distributed on the states (except state 0) and then goes to 0 for every state. When $p > 0.5$ all states are non null recurrent since it can be proved the mean time to return to state 0 is finite.

Periodicity. The DTMC of figure 2.3 exhibits a pathologic behaviour. Starting from state 0, at even (resp. odd) instants the chain is in an even (resp. odd) state. Thus there is no hope to have a steady-state behaviour. This phenomenon is due to the *periodicity* of the graph.

The periodicity of an irreducible chain is the greatest integer k such that the states can be partitioned into subsets $S_0, S_1, \ldots, S_{k-1}$ with the requirement that from the states in S_i the chain moves in one step only to states which are in $S_{(i+1) \bmod k}$. Observe that the periodicity is well defined even in the case of infinite Markov chains since it cannot be greater than the length of any cycle

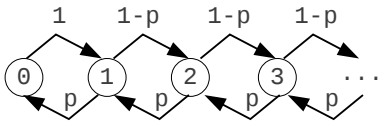


Figure 2.3: an infinite irreducible DTMC

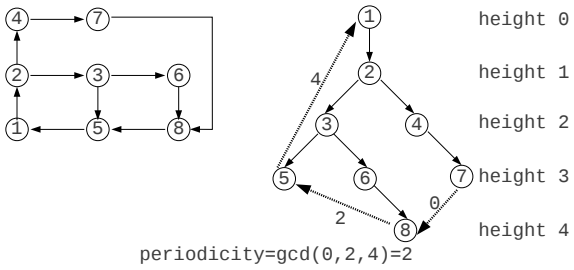


Figure 2.4: Example of the computation of a DTMC periodicity

of the graph (and there is a last one since the chain is irreducible). When the period is 1, the chain is said to be *aperiodic*.

The periodicity of a finite chain can be computed by a linear time algorithm w.r.t. the size of the graph. The algorithm builds a directed tree that covers all nodes of the chain, using any arbitrary strategy (breadth-first, depth-first, etc) that allows to label every node u with its "heigth" $h(u)$. During the traversal, one associates with every arc (u, v) of the graph a weight $w(u, v) = h(u) - h(v) + 1$: as a result all the arcs that are part of the covering tree have a null weight. The periodicity of the graph is then the greatest common divisor (gcd) of the arcs of non null weight and can be computed on the fly. The formal proof of correctness, that we do not develop here is based on the two following observations. Periodicity is the gcd of the length of the elementary circuits of the graphs, and this length is equal to the sum of the weight of the arcs of the circuit. The application of the algorithm to the example is illustrated in figure 2.4.

The following theorem characterises a situation which ensures the existence of a steady-state distribution.

Theorem 2.1. *An irreducible, aperiodic chain (also called* ergodic*) has a steady-state distribution, and such a distribution is* independent from the initial distribution.

Steady-state analysis of finite DTMC. When the DTMC is finite, the analysis is significantly simplified. First the status of the states only depends on the structure of the graph and more precisely on the SCCs. States of *bottom* SCCs (*i.e.* without exit arcs, BSCC) are non null recurrent while other states are transient. Observe that every BSCC constitutes an irreducible subchain.

First assume that the graph is strongly connected. From theorem 2.1, we know that there is a unique steady-solution. So we take the limit of the equation $\boldsymbol{\pi}_{n+1} = \boldsymbol{\pi}_n \cdot \mathbf{P}$. as n goes to infinity (which is mathematically sound) and we get $\boldsymbol{\pi} = \boldsymbol{\pi} \cdot \mathbf{P}$. Thus $\boldsymbol{\pi}$ is the single distribution which is a solution for:

$$\mathbf{X} = \mathbf{X} \cdot \mathbf{P} \wedge \mathbf{X} \cdot \mathbf{1} = 1 \tag{2.3}$$

where $\mathbf{1}$ denotes the column vector of all 1. Furthermore in equation (2.3), we can omit an arbitrary column of $\mathbf{P}$ since the sum of the equations related to $\mathbf{P}$ ($\sum_i(\sum_j p_{ij})X_i = \sum X_i$) is always fulfilled.

Example. The steady-state distribution of the DTMC of figure 2.2 fulfils the following equations:

$$\pi_1 = 0.3\pi_1 + 0.2\pi_2 \;\; \pi_2 = 0.7\pi_1 + 0.8\pi_3 \;\; \pi_3 = \pi_2 \; \pi_1 + \pi_2 + \pi_3 = 1$$

with solution $(\frac{1}{8}, \frac{7}{16}, \frac{7}{16})$

Equation (2.3) can be solved with a direct method like a Gaussian elimination. However if the size of the system is large, iterative methods are more effective. The simplest one iterates over $\mathbf{X} \leftarrow \mathbf{X} \cdot \mathbf{P}$ [Ste94].

We now consider the almost general case, with the single remaining assumption that the BSCCs (denoted as $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$) are aperiodic with steady-state distribution $\{\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_k\}$. In this case also the chain has a steady-state distribution (which now depends on the initial distribution), given by $\boldsymbol{\pi} = \sum_{i=1}^{k} \Pr(\text{of reaching } \mathcal{C}_i) \cdot \boldsymbol{\pi}_i$.

To compute the probability of reaching a BSCC we condition on being in a initial state: $\Pr(\text{of reaching } \mathcal{C}_i) = \sum_{s \in S} \boldsymbol{\pi}_0(s) \cdot \boldsymbol{\pi}'_{\mathcal{C}_i}(s)$ where $\boldsymbol{\pi}'_{\mathcal{C}_i}(s) = \Pr(\text{of reaching } \mathcal{C}_i \mid X_0 = s)$. If $\mathbf{P}_{T,T}$ is the submatrix of the transition matrix limited to transient states, and if $\mathbf{P}_{T,i}$ is the submatrix from transient states towards the states of $\mathcal{C}_i$, then:

$$\boldsymbol{\pi}'_{\mathcal{C}_i} = (\sum_{n \geq 0} (\mathbf{P}_{T,T})^n) \cdot \mathbf{P}_{T,i} \cdot \mathbf{1} = (\mathbf{Id} - \mathbf{P}_{T,T})^{-1} \cdot \mathbf{P}_{T,i} \cdot \mathbf{1}$$

where $\mathbf{Id}$ is the identity matrix.

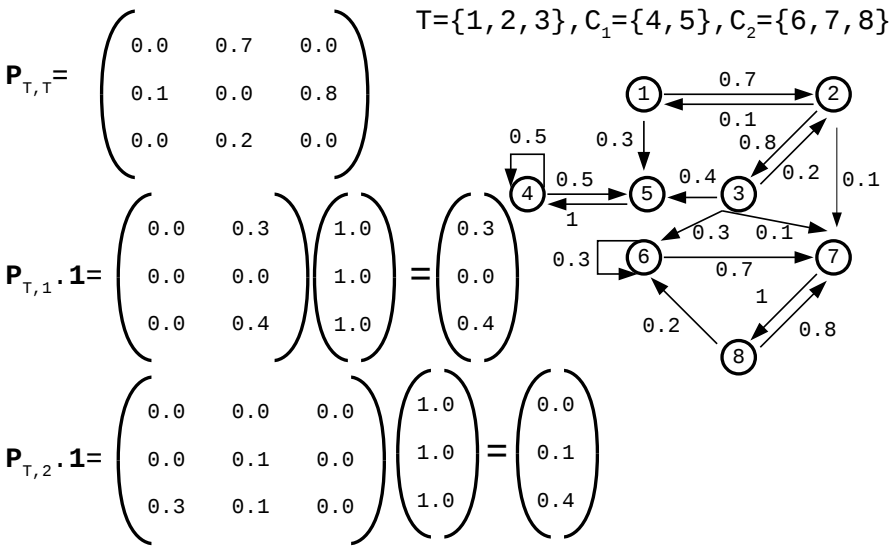


Figure 2.5: Matrices involved in the computation of reachability probabilities

The first equality is obtained by conditioning on the length of all possible paths that leads to $\mathcal{C}_i$, while the second one is an immediate consequence of the finitess of $\sum_{n\geq 0}(\mathbf{P}_{T,T})^n$.

Example. Figure 2.5 exhibits the decomposition of a chain w.r.t. SCC and the matrices involved in the computation of the reachability probabilities.

In the sequel, in case of a BSCC composed by a single state s (*i.e.* $\mathbf{P}[s,s]=1$), we will say that s is an *absorbing* state.

2.2.3 Continuous time Markov chain

Presentation. While DTMC is an appropriate model when focusing on the probabilistic behaviour of the system disregarding time, it is necessary to look for a model also presenting the memoryless property in a continuous time setting. A Continuous Time Markov Chain (CTMC) has the following characteristics:

- the time interval T_n is a random variable distributed as a negative exponential, whose rate depends only on the state X_n. That is to say:

$$\Pr(T_n \leq \tau \,|\, X_0 = s_{i_0}, ..., X_n = s_i, T_0 \leq \tau_0, ..., T_{n-1} \leq \tau_{n-1}) =$$

$$\Pr(T_n \leq \tau \,|\, X_n = s_i) = 1 - e^{\lambda_i \cdot \tau}$$

- The next state depends only on the current state, and the transition probabilities remain constant[2] over time:

$$\Pr(X_{n+1} = s_j \mid X_0 = s_{i_0}, ..., X_n = s_i, T_0 \leq \tau_0, ..., T_{n-1} \leq \tau_{n-1}) =$$

$$\Pr(X_{n+1} = s_j \mid X_n = s_i) = p_{ij} \equiv \mathbf{P}[i, j]$$

The DTMC defined by $\mathbf{P}$ is called *embedded chain*. It observes the change of state, independently of the time elapsed in the state. A CTMC state is said to be absorbing if it is absorbing in the embedded DTMC.

Transient and steady-state behaviour of a CTMC. In a continuous time Markov chain at any time the evolution of a DES is completely determined by its current state, due to the memoryless property of the exponential distribution.

In particular, the process is fully characterized by the initial distribution $\boldsymbol{\pi}(0)$, matrix $\mathbf{P}$ and by the rates λ_i. Let $\boldsymbol{\pi}(\tau)$ be the distribution of $Y(\tau)$ and write $\pi_k(\tau) = \boldsymbol{\pi}(\tau)(s_k)$. If δ is small enough, the probability of more than one event occurring in the interval τ and $\tau + \delta$ is very small and can be neglected, and the probability of a change from state k to state k' is approximately equal to $\lambda_k \cdot \delta \cdot p_{kk'}$ (by definition of exponential distribution).

$$\pi_k(\tau + \delta) \approx \pi_k(\tau) \cdot (1 - \lambda_k \cdot \delta) + \sum_{k' \neq k} \pi_{k'}(\tau) \cdot \lambda_{k'} \cdot \delta \cdot p_{k'k}$$

From which we derive:

$$\frac{\pi_k(\tau + \delta) - \pi_k(\tau)}{\delta} \approx \pi_k(\tau) \cdot (-\lambda_k) + \sum_{k' \neq k} \pi_{k'}(\tau) \cdot \lambda_{k'} \cdot p_{k'k}$$

and finally:

$$\frac{d\pi_k}{d\tau} = \pi_k(\tau) \cdot (-\lambda_k) + \sum_{k' \neq k} \pi_{k'}(\tau) \cdot \lambda_{k'} \cdot p_{k'k}$$

Let us define matrix $\mathbf{Q}$ as: $q_{kk'} = \lambda_k \cdot p_{kk'}$ for $k \neq k'$ and $q_{kk} = -\lambda_k (= -\sum_{k' \neq k} q_{kk'})$. We rewrite the previous equation as:

$$\frac{d\boldsymbol{\pi}}{d\tau} = \boldsymbol{\pi} \cdot \mathbf{Q} \tag{2.4}$$

Matrix $\mathbf{Q}$ is called *infinitesimal generator* of the CTMC.

According to equation (2.4) the infinitesimal generator completely specifies the evolution of the system. Although this equation clearly establishes the

[2] Also in this case we say that the chain is *time homogeneous*

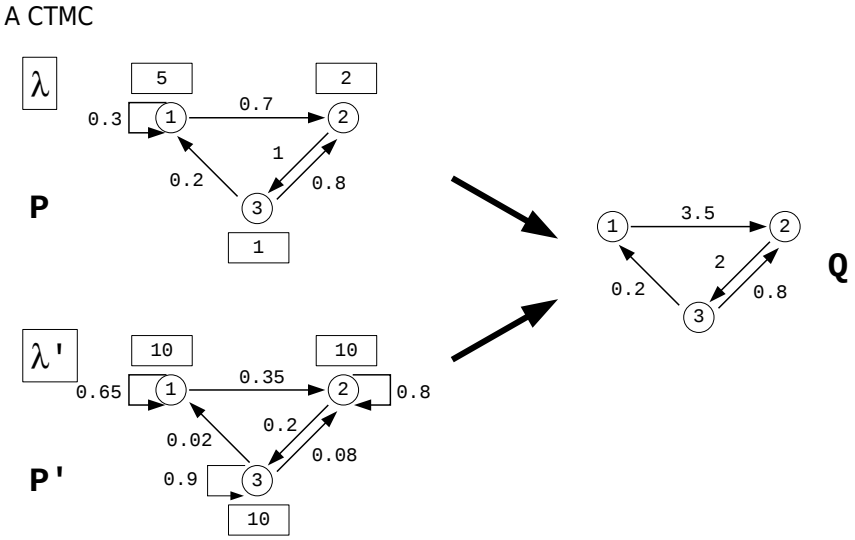


Figure 2.6: Two equivalent CTMCs: a non uniform and a uniform one

memoryless property of the CTMC, it does not give any direct mean of computing the transient behaviour of a CTMC. A possible method, called *uniformisation*, has been defined in [Jen53], and it is based upon the construction of a second Markov chain which is equivalent to the first one from a probabilistic point of view. This chain is built as follows. Let's choose a value $\mu \geq Sup(\{\lambda_i\})$, and assume that this is the parameter of the exponential distribution of the time until the next change of state, whatever the current state is (from which the term *uniform*). The change of state is defined by the transition matrix $\mathbf{P}^\mu$ defined by: $\forall i \neq j, \mathbf{P}^\mu[s_i, s_j] = (\mu)^{-1} \cdot \lambda_i \cdot \mathbf{P}[s_i, s_j]$. The computation of the infinitesimal generator of such a chain shows immediately that it is equal to the infinitesimal generator of the first CTMC, which implies that, if we disregard transitions, the two CTMCs describe the same stochastic process. We illustrate such a transformation in figure 2.6.

We compute the transient distribution $\boldsymbol{\pi}(\tau)$ as follows. We first compute the probability of being in state s at time τ, knowing that there have been n changes of state in the interval $[0, \tau]$. This probability is obtained through the embedded Markov chain, and precisely as $\boldsymbol{\pi}(0) \cdot (\mathbf{P}^\mu)^n$. Afterwards we "condition" it through the probability of having n changes of state, knowing that the time between two successive changes follows an exponential distribution. This

probability is given by $e^{-\mu\cdot\tau}\cdot(\mu\cdot\tau)^n/n!$, from which we obtain:

$$\boldsymbol{\pi}(\tau)=\boldsymbol{\pi}(0)\cdot\left(e^{-\mu\cdot\tau}\sum_{n\geq 0}\frac{(\mu\cdot\tau)^n(\mathbf{P}^\mu)^n}{n!}\right) \tag{2.5}$$

Although there is an infinite sum, in practice the sum converges rather quickly, and the sum can be stopped once the precision required is greater than $e^{-\mu\cdot\tau}\cdot(\mu\cdot\tau)^n/n!$.

We now consider the asymptotic behaviour of a CTMC. Again, the simplest way is to study the embedded chain, which, as observed when explaining uniformization, it is not unique. Let us build a DTMC as follows. Choose $\mu > Sup(\{\lambda_i\})$, since the inequality is strict, it is true that, for each state s, $\mathbf{P}^\mu[s,s]>0$ and therefore each BSCC of this chain is ergodic. As a consequence, a single stationary distribution exists, that measures the steady state probability of the occurrence of a state. Since the uniform chain has the same mean sojourn time in each state, equal to $(1/\mu)$, this also gives the stationary distribution of the CTMC.

In the particular case (rather frequent) in which the embedded chain is ergodic, this distribution can be computed through the solution of the equation $\mathbf{X}=\mathbf{X}\cdot\mathbf{P}^\mu$, and $\mathbf{P}^\mu=\mathbf{I}+(1/\mu)\mathbf{Q}$. The distribution is therefore the unique solution of the equation:

$$\mathbf{X}\cdot\mathbf{Q}=0 \quad \text{et} \quad \mathbf{X}\cdot\mathbf{1}^T=1 \tag{2.6}$$

By analogy, we then say that the CTMC is ergodic. Observe that while in the infinite case, the uniformisation technique may be not applicable (when $Sup(\{\lambda_i\})=\infty$), most of the results for DTMC still hold. In particular, theorem 2.1 is also valid without any requirement about periodicity.

2.2.4 Beyond CTMCs

DTMC and CTMC are characterized by the memoryess properties: in DTMC the next state depends only on the current state and in CTMC also the sojourn time enjoys the same type of property: knowing how long the system has been in a state does not influence how long it will take before a change of state takes place. In the following we introduce three different ways of removing the memoryless condition of sojourn times, that lead to stochastic processes whose solution is more expensive, but still affordable in many practical cases, and that may model in a more accurate manner the system under study.

The first extension leads to Vanishing CTMC, which allows the sojourn time of a subset of states to be equal to zero. The second extension leads to Semi-

Markov Processes, where states may have any distribution, as far as the behaviour of the process, as observed upon a change of state, is still a DTMC. The last extension leads to Markov Renewal Processes, where the requirement is simply that the behaviour of the system, as observed upon *some* of the changes of states, is still a DTMC.

Vanishing CTMCs

Presentation. In this chapter we have stated that a stochastic process is a CTMC if it has an embedded DTMC that describes the change of state and if sojourn times are exponentially distributed. Here we extend this concept to Vanishing CTMC (VCTMC), which are stochastic processes in which the sojourn time is either exponentially distributed or exactly zero. A sojourn time of zero is typically used for whose states in which the system takes a logical decision, or for which the sojourn time is much smaller than the other states, so that it can be considered negligible (with the additional advantage of avoiding numerical instability problems). In a VCTMC the states are partitioned into *vanishing* states (with sojourn time equal to 0), and *tangible* states (with exponentially distributed sojourn time).

Definition 2.1. *A* CTMC with vanishing states *(VCTMC) is a quadruple* $\mathsf{V} = (\mathsf{D}, S_T, S_V, \Lambda)$ *where* $\mathsf{D} = (S, \mathbf{P}, \text{INIT}, L)$ *defines a labelled DTMC with a finite set* S *of states, such that* $S_T \cup S_V = S$ *and* $S_V \cap S_T = \emptyset$*, and* $\Lambda = \{\Lambda_s | s \in S\}$ *is a set of random variables which describe the sojourn time in states of* D*, with the constraint that* Λ_s *is deterministically zero if* $s \in S_V$ *(vanishing state), and is exponentially distributed otherwise (tangible state).*

If S_n is the random variable that describes the state of D after the n-th state change and if T_n is the random variable that describes the sojourn time in state S_n then, given states $s, s' \in S$ and duration $\delta \in \mathbb{R}_{\geq 0}$, we can define the following stochastic behaviour of V:

$$P\{S_{n+1} = s', T_n \leq \delta | S_n = s, S_{n-1} = s_{n-1}, S_{n-1} \leq \tau_{n-1}, \cdots, Y_1, S_0 = s_0\} = \begin{cases} \mathbf{P}(s,s') \cdot (1 - e^{-\lambda_s \cdot \delta}) & \text{if } s \in S_T \\ \mathbf{P}(s,s') & \text{if } s \in S_V \wedge \delta = 0 \\ 0 & \text{if } s \in S_V \wedge \delta > 0 \end{cases}$$

where λ_s is the rate of the exponential distribution associated with the random variable Λ_s of a tangible state $s \in S_T$.

For a VCTMC $\mathsf{V} = (\mathsf{D}, S_T, S_V, \Lambda)$, we consider the probability matrix $\mathbf{P}$ of D partitioned in the following way:

$$\mathbf{P} = \left| \begin{array}{cc} \mathbf{P}_{VV} & \mathbf{P}_{VT} \\ \mathbf{P}_{TV} & \mathbf{P}_{TT} \end{array} \right|$$

where $\mathbf{P}_{VV}$ contains transition probabilities from a vanishing state to a vanishing state, $\mathbf{P}_{VT}$ from a vanishing state to a tangible state, etc. Assumption 2.1 implies that $\sum_{n\to\infty} \mathbf{P}_{VV}^n$ is finite; that is, the mean number of visits within the set of vanishing states without ever reaching a tangible state is finite. This condition can be checked on the directed graph of D by (1) merging the tangible states in a state without outgoing edges and (2) checking that this state is the only BSCC of the modified graph.

Solution process. Since a VCTMC is a continuous stochastic process both the transient and steady state probabilities for the vanishing states is equal to zero (there is a null probability of finding, in the long run or at time t, a system in a state in which it stays for a zero amount of time). The solution process then concentrates on how to compute the probabilities of tangible states. This can be done through the construction and solution of an "equivalent" CTMC built only on the set of tangible states. Here by equivalent we mean "with the same transient and steady state solution of tangible states". There are different ways of computing the transient and steady-state probabilities of a VCTMC. One of them will be presented in the section devoted to generalized stochastic Petri nets.

Semi-Markovian processes

Presentation. Here we describe a restricted notion of semi-Markovian process since it is enough for our purposes and allows a simplified presentation of the computation of steady-state distributions. A semi-Markovian process is an extension of CTMC where sojourn time in states may have any distribution. This process has the following characteristics:

- The time interval T_n is a random variable that only depends on state S_n. Otherwise stated:

$$\Pr(T_n \leq \tau \,|\, S_0 = s_{i_0}, ..., S_n = s_i, T_0 \leq \tau_0, ..., T_{n-1} \leq \tau_{n-1}) =$$

$$\Pr(T_n \leq \tau \,|\, S_n = s_i) = \Pr(D_i \leq \tau)$$

  where D_i is a random variable with a finite mean, denoted d_i.

- The state following the current state only depends on this state and transition probabilities are constant over time:

$$\Pr(S_{n+1} = s_j \,|\, S_0 = s_{i_0}, ..., S_n = s_i, T_0 \leq \tau_0, ..., T_{n-1} \leq \tau_{n-1}) =$$

$$\Pr(S_{n+1} = s_j \mid S_n = s_i) \equiv p_{ij} = \mathbf{P}[i, j]$$

Observe that here again, the sequence of states (S_n) constitutes a DTMC embedded in the process. So the reachability probabilities, that only depend on this DTMC, can still be computed.

Steady-state analysis. Here we only state a sufficient condition for the existence of the stationary distribution which covers the most frequent cases. First we assume that the embedded chain is irreducible with a distribution solution of $\mathbf{X} \cdot \mathbf{P} = \mathbf{X}$ and that one of the distribution D_i is not arithmetic (*i.e.* it is not concentrated on a arithmetic sequence in $\mathbb{R}^+$).

As in CTMC, the entrances in an arbitrary state (say s_i) can constitute a renewal process. Given some state, the fact that it occurs infinitely only depends on transition probabilities p_{ij} and is ensured by our first hypothesis. The mean return time must be carefully examined. Indeed, every visit in s_i gives place to a sojourn with mean time d_i. Thus although the mean number of visits before a return is finite, the mean return time could be infinite. Let us call $\boldsymbol{\pi}'$ ($\pi'_k = \boldsymbol{\pi}'[s_k]$) the distribution solution of equation [2.3]. Then the mean number of visits of s_k between two visits of s_i is $\frac{\pi'_k}{\pi'_i}$. Consequently, the mean return time to s_i is equal to:

$$d_i + \sum_{k \neq i} d_k \cdot \frac{\pi'_k}{\pi'_i} = \frac{1}{\pi'_i} \cdot \sum_k d_k.\pi'_k$$

Otherwise stated, the existence of a stationary distribution is ensured if $\sum_k d_k.\pi'_k$ is finite. Since D_i is not arithmetic, one easily deduces that the distribution of return is not arithmetic.

With the same reasoning, one concludes that the ratio $\frac{\pi_k}{\pi_i}$ corresponds to the mean sojourn time in s_k between two returns in s_i divided by the mean sojourn time in s_i (of course this computation requires to choose a reference state s_i with a sojourn time strictly greater than zero.

Observe that the way we have proceeded also allows some distributions D_i to be concentrated in 0.

Markov Regenerative processes

Presentation *Markov Regenerative Processes* (MRP) are a class of non-Markovian stochastic processes that is characterized by a sequence of time instants called *regeneration points* in which the process loses its memory, i.e. the age of any non-exponential events is 0. The behavior between these points is then described by a time-limited stochastic process, while the process, observed at regeneration points, is still a DTMC.

The definition of an MRP requires first the identification of a sequence of states in which the process loses its memory.

Definition 2.2 (Markov renewal sequence). *Let $\mathcal{S}$ be a finite discrete state space. A sequence of bivariate random variables $\{\langle Y_n, T_n\rangle \mid n \in \mathbb{N}\}$ is called a* Markov renewal sequence *(MRS) with* regeneration points $Y_n \in \mathcal{S}$ *encountered at* renewal times $T_n \in \mathbb{R}_{\geq 0}$ *iff:*

- $0 = T_0 < T_1 < T_2 < \dots$
- $P\{Y_{n+1} = j, T_{n+1} - T_n \leq \tau \mid Y_n = i, T_n, \dots, Y_0, T_0\} =$

$= P\{Y_{n+1} = j, T_{n+1} - T_n \leq \tau \mid Y_n = i\} =$ (Markov property *for* Y_n)

$= P\{Y_1 = j, T_1 \leq \tau \mid Y_0 = i\}$ (Time homogeneity)

The process Y_n is a discrete-time Markov chain, called the *embedded Markov chain* (EMC). Conversely, the process T_n is not a Markov renewal sequence, since the times $T_{n+1} - T_n$ are not i.i.d., but depend on Y_n.

On the renewal sequence an MRP can then be defined as follows.

Definition 2.3 (Markov regenerative process). *A stochastic process $\{X_\tau \mid \tau \geq 0\}$ is a* Markov regenerative process *(MRP) if there exists an MRS $\{\langle Y_n, T_n\rangle \mid n \in \mathbb{N}\}$ of random variables such that all the conditional finite dimensional distributions of $\{X(T_n + \tau) \mid \tau \geq 0\}$ given $\{X_u \mid 0 \leq u \leq T_n, Y_n = i\}$ are the same of $\{X_\tau \mid \tau \geq 0\}$ given $Y_0 = i$, so that:*

$$Pr\{X_{T_n+\tau} = j \mid X_u, 0 \leq u \leq T_n, Y_n = i\} \;=\; Pr\{X_\tau = j \mid X_0 = i\}$$

The process behavior $\{X_\tau \mid T_n \leq \tau < T_{n+1}\}$ between two regeneration points Y_n and Y_{n+1} is described by a continuous time process, called the *subordinated process* of Y_n. We only consider the class of time-homogeneous MRP where the subordinated process is a CTMC, called the *subordinated Markov chain* (SMC) of state Y_n.

Note the difference between Markov Regenerative and semi-Markov process: in semi-Markov processes all the states are regeneration points, so the state spaces of the process and of the embedded DTMC are exactly the same, while in MRP only a subset of the states are regenerative, and the embedded DTMC is built on a smaller number of states than the MRP itself.

2.3 Stochastic Petri nets

2.3.1 Petri nets

A Petri net is a formal model of dynamical system where the state of the system is characterized by the number of *tokens* in *places* and where a *transition* needs tokens to be consumed in some places and then produces tokens in some places.

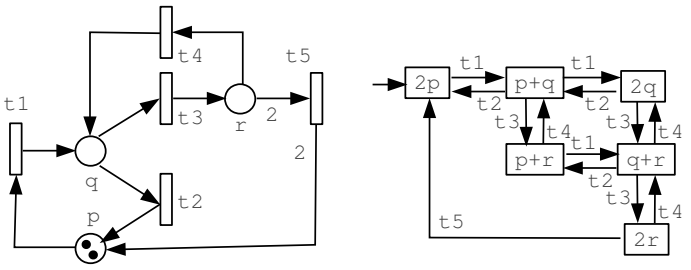


Figure 2.7: a Petri net and its reachability graph

Definition 2.4. *A Petri net $\mathcal{N} = (P, T, \mathbf{Pre}, \mathbf{Post}, m_0)$ is defined by:*

- *P, a finite set of places,*
- *T, a finite set of transitions with $P \cap T = \emptyset$,*
- **Pre** *and* **Post**, *the pre and post incidence matrices indexed by $p \times T$ ranging over $\mathbb{N}$,*
- *$m_0 \in \mathbb{N}^P$, the initial marking.*

A net is usually represented by a bipartite graph. The vertices are the places, drawn as circles, and the transitions, drawn as rectangles. There is an edge between place p (resp. transition t) and transition t (resp. place p) labelled by $\mathbf{Pre}(p,t)$ (resp. $\mathbf{Post}(p,t)$) if $\mathbf{Pre}(p,t) > 0$ (resp. $\mathbf{Post}(p,t) > 0$). When a label is equal to 1, it is omitted. The initial marking of a place $m_0(p)$ is represented by $m_0(p)$ tokens in p, drawn as filled circles. The left part of Figure 2.7 illustrates the graphical representation of a net.

A state in a Petri net is a *marking*, i.e. and item of $\mathbb{N}^P$. The incidence matrix $\mathbf{W}$ is defined by $\mathbf{W} = \mathbf{Post} - \mathbf{Pre}$. The firing rule defined below describes the semantics of a net.

Definition 2.5. *Let $\mathcal{N}$ be Petri net, m be a marking and t be a transition. Then:*

- *t is* enabled *(or* fireable*) i m if:*
  $\forall p \in P \; m(p) \geq \mathbf{Pre}(p,t)$
  which is denoted by $m[t\rangle$ or $m \xrightarrow{t}$

- *When enabled, the firing of t leads to a marking m' defined by:*
  $\forall p \in P\ m'(p) = m(p) + \mathbf{Post}(p,t) - \mathbf{Pre}(p,t) + \mathbf{W}(p,t)$
  which is denoted by $m[t\rangle m'$ or $m \xrightarrow{t} m'$

Let m be a marking and $\sigma = t_1 \dots t_n$ be a sequence of transitions. Then σ is fireable from m if there exists markings $m_1, \dots, m_n = m'$ such that for all $1 \leq i \leq n$, one has $m_{i-1}[t_i\rangle m_i$. One says that σ is a *firing sequence* from m and that m' is reachable from m. One notes $m[\sigma\langle m'$ or $m[\xrightarrow{\sigma} m'$. The *reachability graph* of a net $\mathcal{N}$ is defined by:

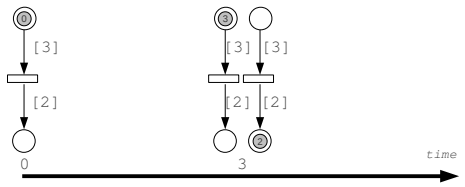
- The vertices of the graph are the markings reachable from m (this set is called the *reachability set* of $\mathcal{N}$),
- There is an edge between m and m' labelled by t if and only if $m[t\rangle m'$.

The right part of Figure 2.7 is the reachability graph of the net on the left part. Generally a marking m is represented as bag over the set of places $\sum_{p|m(p)>0} m(p)\ p$ with $m(p)$ omitted when equal to 1 (*e.g* $2p + 3q + r$).

2.3.2 Stochastic Petri nets with general distributions

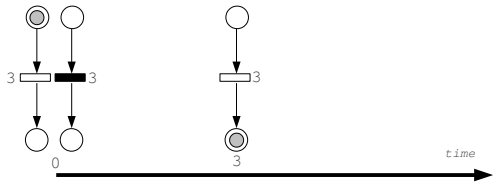
In order to enlarge Petri nets in such a way that its formal semantics should be a stochastic process, we must address the following preliminary question: how to introduce time in nets? There are at least three ways to do so.

A token-based semantic. First, we can add an age for every token. When time elapses, the age of every token is incremented by the associated duration. Furthemore the input and output arcs of a transition are now labelled by a bag of intervals specifying the age of the tokens that are allowed to be consumed and the possible age of tokens that will be produced. The behaviour of such a net is illustrated below.

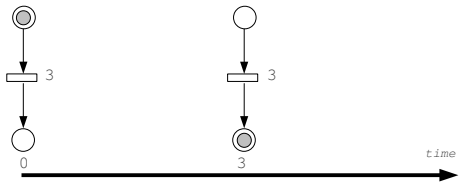


A duration-based semantic. Second, we can specify a duration for every transition. Once a transition is selected (with no time elapsing), the tokens are consumed and after the duration associated with the transition has elapsed, the

tokens are produced. Observe that with this semantic, except at selection instants, the marking of the net may not correspond to a reachable marking of the untimed net. The behaviour of such a net is illustrated below.



A delay-based semantic. Last, we can specify a delay for every transition. Once a transition is enabled, the delay must elapse before it can be fired but its tokens could be consumed by an earlier firing of some other transition. The behaviour of such a net is illustrated below.



Usually one chooses the third alternative; so the stochastic feature of Petri nets is introduced by considering that a transition has a random firing delay (taking values in $\mathbb{R}^+$). The different families of stochastic Petri nets are defined by restricting the type of distributions. For the moment, we do not make any hypothesis on distributions. However the definition of distributions is not sufficient to characterize the stochastic process. We are going to successively study the problems related to this characterization[3].

Choice policy.

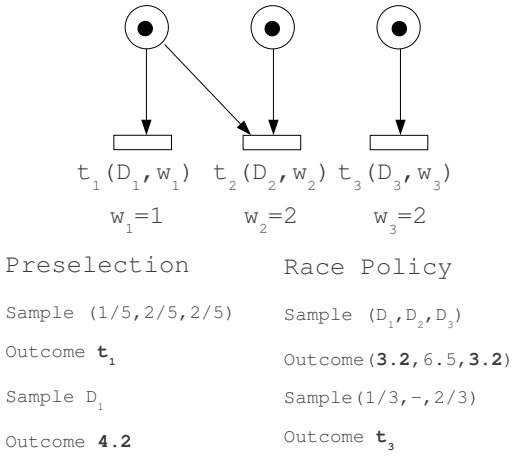
Given any marking, we need to determine the next transition to fire among the fireable ones. There are two possible strategies:

- a probabilistic choice w.r.t. a distribution associated with the subset of fireable transitions. This is a *preselection* since the choice takes place before the sampling of the delay.

[3]Most of the parameters of the process can depend on the current marking. For sake of simplicity, we will not mention it in this chapter.

- an independent sampling for every delay followed by the choice of the shortest delay. In case of equal delays, one also performs a probabilistic choice called *post-selection*.

The second solution is always chosen because on the one hand it corresponds to a more natural modeling and on the other hand since with the help of immediate transitions, preselection can be simulated by post-selection. Observe that except if the distributions are continuous (which excludes the case of equal samplings), one needs to specify the distributions of selections. We illustrate the two possible choice policies below (D_i is the random delay of transition t_i and w_i its weight for the probabilistic choice). With preselection policy, the outcome of the sample is t_1 and then the sample of the delay for t_1 is 4.2. With the race policy, after sampling the three delay distributions, two of the outcomes (D_1 and D_3) have the minimal value 3.2. Thus with a post-selection policy, one selects t_3.



Service policy.

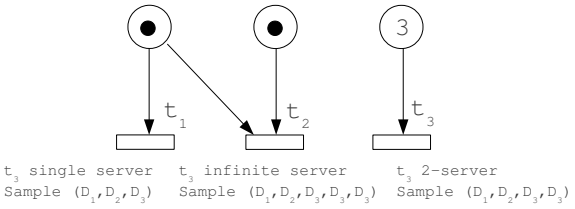
If a transition has an enabling degree $e > 1$ at marking m (i.e. $e = max(x \in \mathbb{N} \mid \forall p \in P\ x\mathbf{Pre}(p,t) \leq m(p))$), one can consider that the marking *provides* e clients to the transition viewed like a server. So when sampling the delay, three options are possible depending on the event modeled by the transition:

- a single sampling is performed, the transition offers only one service at a time (*single-server* policy)

- e samplings are performed, the transition is a "parallel" server (*infinite-server* policy)

- $Min(e, deg(t))$ samplings are performed, the transition can offer at most $deg(t)$ simultaneous services; this case generalizes the other ones with $deg(t) = 1$ or ∞ (*multiple-server* policy). The modeller must specify $deg(t)$ for every transition.

We illustrate below the effect of these policies on the sampling process.



Memory policy.

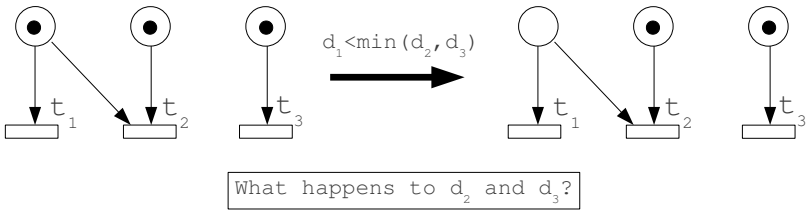
Once transition t is fired, what becomes the sampling that has not be chosen for another transition t'?

The first possibility consists in forgetting the sampling that has been performed. If transition t' remains fireable, this leads to a new sampling (*resampling memory*). With such a semantic, t could model the failure of a service specified by t'.

The second possibility consists in memorizing the sampling decremented by the sampling of t (the remaining time), but only if t' remains fireable (*enabling memory PRD* (*Preemptive Repeat Different*)). If t' is disabled, this mechanism models a time-out t' disarmed by t.

The third possibility is as the previous one for a transition still fireable but let the sampling unchanged if t' is disabled. This sampling will be used again when t' will be fireable (mode *enabling memory PRI* (*Preemptive Repeat Identical*)). A disabled transition t' could model a job aborted by t that should be restarted.

The fourth possibility consists in memorizing the sampling decremented by the sampling of t. A disabled transition t' could model a job suspended by t (*age memory* also called *PRS* (*Preemptive ReSume*)).



We illustrate some of these policies on the above example.

Resampling Memory. Samplings d_2 and d_3 are forgotten.

Enabling Memory, PRD. Sampling d_3 is kept and decremented ($d'_3 = d_3 - d_1$). Sampling d_2 is forgotten.

Age Memory. Samplings d_2 and d_3 are kept and decremented ($d'_3 = d_3 - d_1$ $d'_2 = d_2 - d_1$). Sampling d'_2 is frozen until transition t_2 becomes again enabled.

To complete this policy, we must take into account the case of multiple-server transitions, which requires to choose which samplings should be memorized, decremented or forgotten. The simplest solution is a FIFO policy for samplings. The last performed sampling is the first forgotten. Other policies (like suspend or forget the client the least engaged) are not necessarily compatible with some analysis methods.

Once these three policies are defined, the stochastic process is fully determined. We now focus on the distributions for transition delays.

2.3.3 Stochastic Petri nets with exponential distributions

In the basic model [FN85, Mol81] every transition t has an exponential distribution with rate μ_t.

Let us examine the stochastic process generated by a stochastic Petri net with policy *single-server*. Let m be some marking, $t_1, \ldots, t_k$ the fireable transitions from m. Let us note μ_i for μ_{t_i}. One can check that:

- the sojourn time is an exponential with rate $\mu_1 + \cdots + \mu_k$
- the probability to pick t_i as the next firing is equal to $\frac{\mu_i}{\mu_1+\cdots+\mu_k}$ and it is independent from the sojourn time in the marking.
- the distribution of the remaining firing delay of t_i if t_j is fired is equal to the initial distribution (memoryless property)

Otherwise stated, only the new marking determines the future behavior of the stochastic process. Thus it is a continuous time Markov chain, isomorphic to the reachability graph of the Petri net, whose all parameters are given by states (*i.e.* the markings). This reasoning is also valid for other service policies.

If the graph is finite formula (2.5) gives the transient behavior of the net and if furthermore it has a single BSCC then the resolution of equation (2.6) provides the stationary distribution of the net.

Using the stationary distribution, other performance indices can be computed as the mean throughput (number of firings per time units) of transitions given by:

$$\overline{\chi}_k = \sum_{m \text{ reachable}} \pi_m.\text{services}(m, t_k).\mu_k \tag{2.7}$$

where $\text{services}(m, t_k)$ indicates the number of clients in state m served by transition t_k; this number depends on the enabling degree and the service policy of the transition.

2.3.4 Generalized stochastic Petri nets

Modelling delays with exponential distributions is **reasonable** when:

- Only mean value information is known about distributions.
- Exponential distributions (or combination of them) are enough to approximate the "real" distributions.

Modelling delays with exponential distributions is **not reasonable** when:

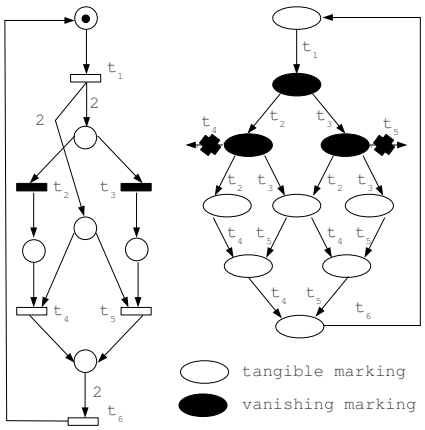
- The distribution of an event is known and is poorly approximable with exponential distributions like a time-out of 10 time units.
- The delays of the events have different magnitude orders like executing an instruction versus performing a database request. In this case, the 0-Dirac distribution is required.

Let us focus on the latter case. Modeling an algorithm or a protocol requires to represent choices, loops and other control structures. These actions are logical operations and have a negligible duration w.r.t. a data transmission for instance. Modeling them by an exponential distribution with a high rate is unsatisfactory since, on the one hand the choice of the rate is arbitrary and on the other hand numerical computations suffer from values with very different magnitude order. To overcome this difficulty, *immediate transitions* (*i.e.* with a distribution concentrated in 0) have been introduced. In this new model [ABC84], called

GSPN for *Generalized Stochastic Petri Net*, the markings are partitioned in two categories: the tangible markings from which no immediate transition is fireable and the vanishing markings. Since exponential transitions almost surely have a non null delay, immediate transitions have (implicit) priority over exponential ones.

Let us observe that:

- Weights are required for immediate transitions since two such transitions will always have the same null delay.
- Since exponential transitions will never be fired when an immediate transition is enabled, one obtains a *restricted* reachability graph corresponding to the embedded DTMC. This restriction is described in the example below.



Let us examine the stochastic process generated by a GSPN from a given marking m. If m is tangible then the process is identical to the one of a Markovian SPN. Let us examine the case of a vanishing marking; there is at least one fireable immediate transition. *Almost surely* the sampling of exponential transitions is greater than 0. Thus the choice of the transition is done by a post-selection between immediate transitions. Since the delay of immediate transitions is null and the distributions of other transitions are without memory, the remaining delay are identical to the initial delays and the state of the process only depends on the new marking.

So this is a semi-Markovian process whose sojourn times in tangible markings follow an exponential distribution and sojourn times in vanishing markings are null. The transition probabilities (matrix $\mathbf{P}$) are obtained either from the rates, or from parameters of post-selection.

The analysis of semi-Markovian process is applicable here. However, in this particular case, an improvement is possible since the stochastic process is a VCTMC. We recall here in the context of nets, the principles of such an analysis. Observe that in the stationary distribution, the vanishing markings have a null occurrence probability. Thus one wants to eliminate them before the resolution of the embedded chain. To this aim, one considers the process as a CTMC whose states are the tangible markings. We need to compute the transition probabilities between these states. So we decompose matrix $\mathbf{P}$ in sub-matrices:

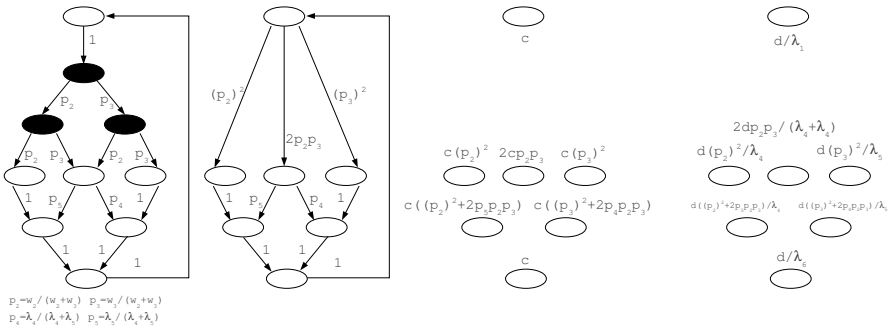
- $\mathbf{P}_{VV}$, transitions between vanishing markings
- $\mathbf{P}_{TT}$, transitions between tangible markings
- $\mathbf{P}_{VT}$, transitions from vanishing markings to tangible markings
- $\mathbf{P}_{TV}$, transitions from tangible markings to vanishing markings

Reasoning on the number of encountered vanishing markings, when going from a tangible marking to another tangible marking, one checks that the new transition matrix $\mathbf{P}'$ is given by:

$$\mathbf{P}' = \mathbf{P}_{TT} + \sum_{n=0}^{\infty} \mathbf{P}_{TV}.(\mathbf{P}_{VV})^n.\mathbf{P}_{VT} = \mathbf{P}_{TT} + \mathbf{P}_{TV}.(\mathbf{Id}_{VV} - \mathbf{P}_{VV})^{-1}.\mathbf{P}_{VT}$$

where $\mathbf{Id}_{VV}$ is the identity matrix on vanishing markings.

When $\mathbf{Id}_{VV} - \mathbf{P}_{VV}$ is not invertible, this means that the process has a pathological behaviour (*i.e.* a non null probability to infinitely remain in the vanishing states) which does not fulfill assumption 2.1. Otherwise the two expressions can be used to compute $\mathbf{P}'$. We illustrate the full computation on the example below. We start with the embedded full discrete-time Markov chain on the left. Then as a first step we eliminate the vanishing states and compute the transition probabilities (shown in the next figure). In this particular case, since there is no loop through vanishing states, the computation is immediate. Then the steady-state visit distributions are computed (the result is shown on the third figure). At last we rescale the distribution with the sojourn time in order to get the steady-state distribution as shown on the last figure.



2.4 Advanced Analysis Methods for Stochastic Petri Nets

In this section, we present methods different from the resolution of a finite Markov chain. Such methods are useful when: (1) either the considered subclass of nets allows a more efficient computation of the steady state-distribution, (2) either the chain is infinite, (3) or the stochastic process is not a Markov chain.

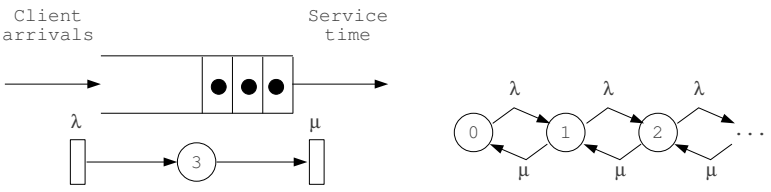
2.4.1 Research of a product form

In this section, we search for an explicit expression of the steady-state distribution which avoids to build the reachability graph. Such an approach has been first developed in queuing networks. So we will present it while modelling queuing networks by Petri nets and then move to the corresponding net theory.

A (Markovian) queue is specified by:

- an interarrival time which is an exponential distribution with parameter λ,
- a service time which is an exponential distribution with parameter μ.

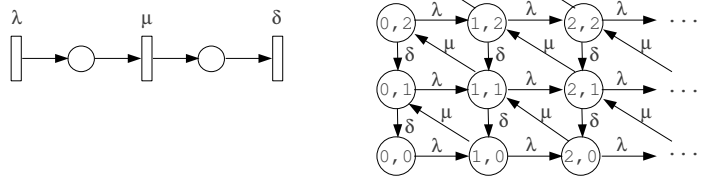
It can be modelled by a SPN presented below with its associated CTMC.



Let $\rho = \frac{\lambda}{\mu}$ be the *utilization* of the queue.

- The steady-state distribution π_∞ exists iff $\rho < 1$.
- The probability of n clients in the queue is $\pi_\infty(n) = \rho^n(1-\rho)$. Observe that this is an explicit expression of the distribution w.r.t. λ and μ.

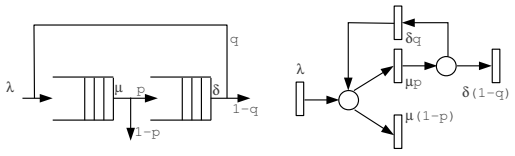
We have presented below a Petri net (and its associated CTMC) modelling two queues in tandem, i.e. the clients are served by the first server and then by the second one.



While this is a very simple case of queuing network, the associated Markov chain is more complex than the one corresponding to two isolated queues. However assume that $\rho_1 = \frac{\lambda}{\mu} < 1$ and $\rho_2 = \frac{\lambda}{\delta} < 1$ (the stability condition). Then:

- The steady-state distribution π_∞ exists.
- The probability of n_1 clients in the first queue and n_2 clients in the second queue is $\pi_\infty(n_1, n_2) = \rho_1^{n_1}(1-\rho_1)\rho_2^{n_2}(1-\rho_2)$.
- It is the product of the steady-state distributions corresponding to two isolated queues!

Let us now consider the general case of an open queuing network. In such a network, when a client leaves a queue, it randomly chooses between leaving the network or entering any queue of the network. The probability distribution of this choice only depends on the queue that the client leaves. Again this kind of networks can easily be modelled by SPNs (merging the exit of a queue with the random choice). We have illustrated it on the example below. When the client leaves the first queue, it can enter the second queue (with probability p) or exit the network. Similarly when the client leaves the first queue, it can enter the first queue (with probability q) or exit the network.



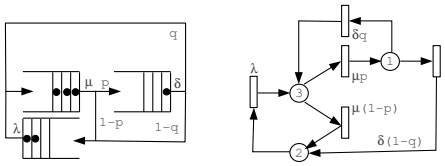
In order to analyze it, we first study the possible steady-state flow of clients through the queues.

- Define the (input and output) flow through the first (resp. second) queue as γ_1 (resp. γ_2).
- Then (assuming that a steady-state exists) $\gamma_1 = \lambda + q\gamma_2$ and $\gamma_2 = p\gamma_1$. Thus $\gamma_1 = \frac{\lambda}{1-pq}$ and $\gamma_2 = \frac{p\lambda}{1-pq}$.

Now assume $\rho_1 = \frac{\gamma_1}{\mu} < 1$ and $\rho_2 = \frac{\gamma_2}{\delta} < 1$ (the stability condition).

- The steady-state distribution π_∞ exists.
- The probability of n_1 clients in the first queue and n_2 clients in the second queue is $\pi_\infty(n_1, n_2) = \rho_1^{n_1}(1-\rho_1)\rho_2^{n_2}(1-\rho_2)$.
- It is still the product of the steady-state distributions corresponding to two isolated queues!

A closed queuing network can be seen as an open queuing network where a fixed number of clients are initially present in the system and never leave it. As illustrated below, there is no additional difficulty to model it.



In order to analyze it, the key concept is the visit ratio (up to a constant) of the queues by a client.

- Define the visit ratio flow of queue i as v_i.
- Then $v_1 = v_3 + qv_2$, $v_2 = pv_1$ and $v_3 = (1-p)v_1 + (1-q)v_2$.
  Thus $v_1 = 1$, $v_2 = p$ and $v_3 = 1 - pq$.

Since the number of clients is constant there is no stability condition. However the relative load of a file can be obtained as the visit ratio divided by the service rate of the queue: $\rho_1 = \frac{v_1}{\mu}$, $\rho_2 = \frac{v_2}{\delta}$ and $\rho_3 = \frac{v_3}{\lambda}$.

- The steady-state probability of n_i clients in queue i is: $\pi_\infty(n_1, n_2, n_3) = \frac{1}{G}\rho_1^{n_1}\rho_2^{n_2}\rho_3^{n_3}$ (with $n_1 + n_2 + n_3 = n$)
- where G the normalizing constant can be efficiently computed by dynamic programming.

Let us develop this last point and introduce:

$$G(m,k) = \sum_{\sum_{i=1}^{k} n_i = m} \prod_{i=1}^{k} \rho_i^{n_i}$$

for $m \leq n$ and $k \leq q$ with q the number of queues.
Observe that we are looking for $G(n,q)$. Decomposing the sum w.r.t. the number of clients in the kth queue, the equations leading the dynamic programing algorithm are:

$$G(0,k) = 1 \qquad G(m,1) = \rho_1^m$$

$$G(m,k) = G(m,k-1) + \rho_k G(m-1,k) \text{ for } m > 0 \text{ and } k > 1$$

Summarizing, a (single client class) queuing network can easily be represented by a Petri net. Such a Petri net is a *state machine*: every transition has at most a single input and a single output place. So there is a central question: can we define a more general subclass of Petri nets with a product form for the steady-state distribution? The answer is positive and we introduce now this subclass.

The principles of *Product-Form Stochastic Petri Nets* (PFSPN) are the following ones:

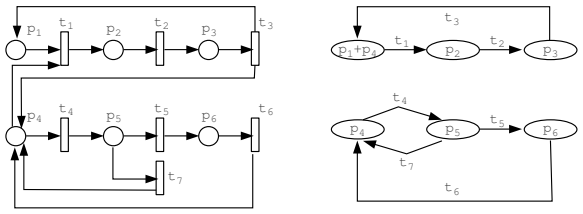
- Transitions can be partionned into subsets corresponding to several classes of clients with their specific activities.
- Places model resources shared between the clients.
- Client states are implicitly represented.

These principles are formalized by two requirements. In order to express the first one, we introduce the *resource graph*.

- The vertices are the input and the ouput bags of the transitions, *i.e.* vectors $\mathbf{Pre}(-,t)$ and $\mathbf{Post}(-,t)$ for transition t.

- Every transition of the net t yields a graph transition ${}^\bullet t \xrightarrow{t} t^\bullet$.

Below we present a Petri net and its resource graph.



We are now in position to specify the first requirement.

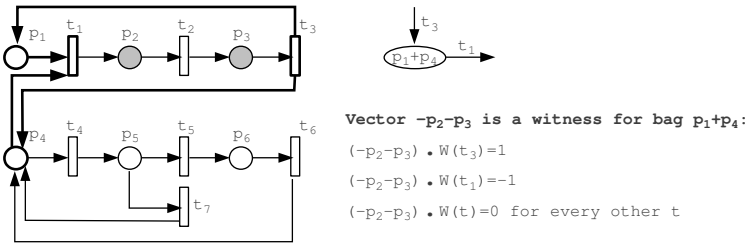
First requirement.
The connected components of the resource graph are strongly connected.

From a modelling point of view, the connected components of the graph correspond to client classes.

In order to express the second one, we introduce the *witnesses* of bags. Let b be a bag.

- Let $In(b)$ (resp. $Out(b)$) the transitions with input (resp. output) b.
- Let v be a place vector, v is a *witness* for b if:
  - $\forall t \in In(b)\ v \cdot W(t) = -1$ (where $W(t)$ is the incidence of t)
  - $\forall t \in Out(b)\ v \cdot W(t) = 1$
  - $\forall t \notin In(b) \cup Out(b)\ v \cdot W(t) = 0$

In other words, given m the current marking the quantity $\sum_{p \in P} v(p)m(p)$ is increased (resp. decreased) when a transition produces (resp. consumes) bag b. The firing of other transitions let unchanged this quantity. Intuitively speaking, a witness counts (up to some constant) the number of clients in the implicit state corresponding to bag b. Below we show a bag with its witness.



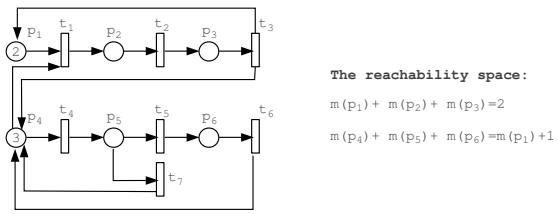
Second requirement. Every bag has a witness.

Given a Petri net, checking whether this net fulfills the two requirements can be done in polynomial time. For instance, the first requirement could be obtained by construction of the connected components of the unoriented version of the resource graph and then verifiying that every component in the oriented version of the resource graph is strongly connected (by Tarjan's algorithm). The witnesses of a bag are defined by linear equations and the existence (and computation if there exists one) can be checked by Gauss's algorithm.

Once a witness has been computed per bag, the expression of the product-form is obtained as follows.

- Assume the requirements are fulfilled, with $w(b)$ the witness for bag b.
- Compute the ratio visit of bags $v(b)$ on the resource graph.
- The output rate of a bag b is $\mu(b) = \sum_{t|\bullet t=b} \mu(t)$ with $\mu(t)$ the rate of t.
- Then: $\pi_\infty(m) = \frac{1}{G} \prod_b \left(\frac{v(b)}{\mu(b)}\right)^{w(b)\cdot m}$

The normalizing constant can be efficiently computed if the reachability space is characterized by linear place invariants. as it is the case for our example.

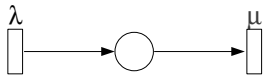


Bibliographic remarks. A first general condition for existence of product form for SPNs was established in [HPTvD90] as a conjunction of the first requirement and another requirement involving the rates of the transition. Thus the characterization was not purely structural. Later on the authors of [HMSS01, HMSS05] establish a (fully structural) necessary and sufficient condition that such a net admits a product form whatever its stochastic parameters (the one we have presented here). Recently in [MN10], another equivalent characterization was proposed but where the membership problem can be decided more efficiently. As observed before, the presence of invariants characterizing the reachability space greatly simplifies the computation of the normalizing constant using a dynamic

programming algorithm [SB93]. However until very recently all the product form Petri nets with such a characterization were in fact product-form queing networks models. In [HMN11], a large subclass of product form Petri nets is exhibited where the computation of the normalizing constant can be done by dynamic programming *without requiring a characterization of the state space by linear invariants*. To conclude, methods based on product form have a weak computational complexity but they are applicable on models whose components have simple synchronizations.

2.4.2 Unbounded Petri nets

In this section we show that it is still possible to compute the steady-state distribution of a Petri net with a single unbounded place [FN89, Hav95]. Below we present again the Petri net corresponding to a queue in order to explain how the steady-state distribution of this net ($x_i = \left(\frac{\lambda}{\mu}\right)^i (1 - \frac{\lambda}{\mu})$, with x_i be the steady-state probability of i tokens in the place) is related to the special form of the infinitesimal generator.



Q the infinitesimal generator

$$\begin{pmatrix} -\lambda & \lambda & 0 & 0 & 0 & 0 & \dots \\ \mu & -(\lambda+\mu) & \lambda & 0 & 0 & 0 & \dots \\ 0 & \mu & -(\lambda+\mu) & \lambda & 0 & 0 & \dots \\ 0 & 0 & \mu & -(\lambda+\mu) & \lambda & 0 & \dots \\ \dots & & & & & & \end{pmatrix}$$

Observe that the infinitesimal generator is a tridiagonal matrix with repeated items on the diagonals. Thus the steady-state equations are:

1. $x_0\lambda - x_1\mu = 0$
2. $\forall i \geq 1\ x_{i-1}\lambda - x_i(\lambda + \mu) + x_{i+1}\mu = 0$
3. $\sum_{i\in\mathbb{N}} x_i = 1$ (the normalizing equation).

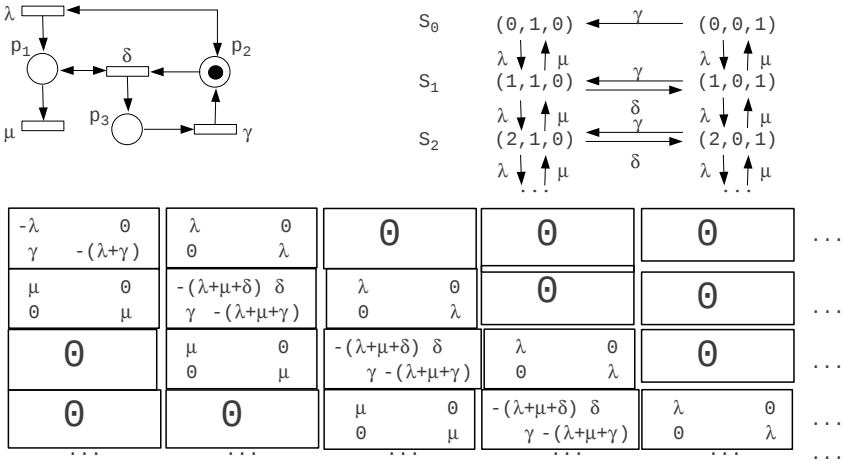
We now develop a analysis of these equations which is not the standard one but has the advantage that it can be generalized. Assume there exists a positive ρ such that:

- $\lambda - \rho(\lambda + \mu) + \rho^2\mu = 0$
- $y_0\lambda - y_1\mu = 0$ and $y_0\lambda - y_1(\lambda + \mu - \rho\mu) = 0$ are the steady-state equations of a two-state Markov chain.

- $\sum_{i\in\mathbb{N}} \rho^i$ is finite.

Then we solve the two-state Markov chain which fixes the value of y_0 and y_1. For all $i \geq 1$ we define $y_{i+1} \equiv y_i\rho$. It is routine to check that the vector $Y \equiv (y_i)_{i\in\mathbb{N}}$ satisfies all the steady-state equations of the system except the normalizing one. However due to our third hypothesis, we can normalize Y and obtains the steady-state distribution. In our case such a ρ exists: $\rho = \lambda/\mu$.

We now develop the theory illustrating it by the more complex example given below. Place p_1 is the single unbounded place and we enumerate the states in the infinitesimal generator ordered by their values on p_1. We observe that the generator still presents the same pattern as before but now w.r.t. to a block decomposition. We say that the matrix is *block tridiagonal* and we will exploit it to express the steady-state distribution.



The results will be presented in a slightly restricted way by stating when necessary additional assumptions. These assumptions could be omitted at the price of a more involved theory. Our first assumption is that arcs around the unbounded place are weighted by 1 (as in this example). Then we decompose the infinite state space S of the chain as follows. For all $i \geq 0$, S_i denotes the set of reachable markings with i tokens in p_1. Index i is called the *level* of a marking in S_i. Furthermore S_i can be defined as $\{i\} \times U$ where U is the projection on the bounded places of the reachable markings. Again for simplicity, we assume that the set of projections of the reachable markings does not depend on the level. Moreover we assume that the associated Markov chain is ergodic.

Since there is a finite number of rates, we can apply the uniformization technique and reason over the embedded discrete time Markov chain of the uniform

version of the chain which has the same steady-state distribution. We observe that after uniformization, the tridiagonal feature is preserved. Once the unbounded place is marked it does not restrict the behaviour of the net. So the transition matrix $\mathbf{P}$ of the embedded chain can be expressed by five matrices:

- $\mathbf{A}_0$ is the transition submatrix from S_i to S_{i+1} ($i \geq 1$).
- $\mathbf{A}_1$ is the transition submatrix from S_i to S_i ($i \geq 1$).
- $\mathbf{A}_2$ is the transition submatrix from S_i vers S_{i-1} ($i \geq 1$).
- $\mathbf{B}_0$ is the transition submatrix from S_0 to S_1 (in the example $\mathbf{B}_0 = \mathbf{A}_0$).
- $\mathbf{B}_1$ is the transition submatrix from S_0 to S_0.

Denoting X_i be the steady-state probability vector of markings with i tokens in p_1, the steady-state equations are:

- $X_0 = X_0 \cdot \mathbf{B}_1 + X_1 \cdot \mathbf{A}_2$, $X_1 = X_0 \cdot \mathbf{B}_0 + X_1 \cdot \mathbf{A}_1 + X_2 \cdot \mathbf{A}_2$
- $\forall i \geq 2\ X_i = X_{i-1} \cdot \mathbf{A}_0 + X_i \cdot \mathbf{A}_1 + X_{i+1} \cdot \mathbf{A}_2$
- $\sum_{i \in \mathbb{N}} ||X_i|| = 1$

In order to solve these equations, we introduce some finite stochastic process that starts in marking (i, u) ($i \geq 1$) and performs as the chain until it reaches a level less or equal than i. For instance, if after the first transition it does not reach level $i + 1$, it is immediately stopped. We do not consider the stopping marking as belonging to this process. We observe that the processes starting in (i, u) and (i', u) with $i' > i$ are the same up to the level translation $i' - i$; so we consider the process starting from marking $(1, u)$.

We now introduce useful quantities. Let $i \geq 1$, $V_i^{(n)}[u, u']$ is the probability that the process (starting from $(1, u)$) has reached time n and that its current marking is (i, u'). Thus matrix $V_1^{(0)}$ is the identity matrix while matrix $V_1^{(n)}$, for $n > 0$, is the null matrix. $N_i[u, u']$ is the mean number of visits of the process (starting from $(1, u)$) to marking (i, u'). Since the process is ergodic, matrix N_i is well defined.

We already have an immediate matrix equation relating these matrices:

$$\forall i \geq 1\ N_i = \sum_{n \in \mathbb{N}} V_i^{(n)}$$

In order to deduce another equation, we remark that to visit state (i, u) ($i > 2$) the process must reach at least time $i - 1$ and must visit at least once level

$i-1$. Considering the last visit to this lower level, we observe that from this last visit the process behaves at the original one (w.r.t. a translation). Thus:

$$\forall i > 2 \; V_i^{(n)} = \sum_{m=i-2}^{n-1} V_{i-1}^{(m)} V_2^{(n-m)}$$

Now we apply the equation related to the N_i's to derive a similar equation for $i > 2$:

$$N_i = \sum_{n \in \mathbb{N}} \sum_{m=i-2}^{n-1} V_{i-1}^{(m)} V_2^{(n-m)} = \sum_{m \geq i-2}^{\infty} V_{i-1}^{(m)} \sum_{n>m} V_2^{(n-m)}$$

$$= \sum_{m \in \mathbb{N}} V_{i-1}^{(m)} \sum_{n \in \mathbb{N}} V_2^{(n)} = N_{i-1} N_2$$

Thus $N_i = (N_2)^{i-1}$.

We now relate these visit probabilities to the transition submatrices. Given a visit at time $n > 0$ at level 2, we reason about the possible levels $1, 2, 3$ at time $n-1$:

$$V_2^{(n)} = V_1^{(n-1)} \mathbf{A}_0 + V_2^{(n-1)} \mathbf{A}_1 + V_3^{(n-1)} \mathbf{A}_2$$

Now we sum over all the instants $n > 0$:

$$N_2 = N_1 \mathbf{A}_0 + N_2 \mathbf{A}_1 + N_3 \mathbf{A}_2$$

Using the fact that $N_1 \equiv Id$ is the identity matrix, that $N_3 = (N_2)^2$ and denoting by R (which is analogous to the ρ of the first example) the matrix N_2, one obtains:

$$R = \mathbf{A}_0 + R\mathbf{A}_1 + R^2 \mathbf{A}_2$$

We have established the first condition we were looking for. Since the Markov chain is ergodic we know that the auxiliary process (starting from $(1, u)$) that we have introduced will stop with probability 1. Decomposing w.r.t. the stopping instant we obtain:

$$\sum_{s \in S_0} \mathbf{B}_0[u, s] + \sum_{u' \in U} \mathbf{A}_1[u, u'] + \sum_{u' \in U} \sum_{u'' \in U} \left((\sum_{n \geq 1} V_2^{(n-1)}[u, u'] \right) \mathbf{A}_2[u', u''] = 1$$

Thus the sum of item of every row of matrix $\mathbf{B}_0 + \mathbf{A}_1 + R\mathbf{A}_2$ is equal to 1. Consequently the system of equations below is the one of some Markov chain and admits a steady-state distribution (Y_0, Y_1).

$$Y_0 = Y_0 \cdot \mathbf{B}_1 + Y_1 \cdot \mathbf{A}_2 \text{ and } Y_1 = Y_0 \cdot \mathbf{B}_0 + Y_1 \cdot (\mathbf{A}_1 + R\mathbf{A}_2)$$

This is the second condition we were looking for. Now for $i \geq 1$, define $Y_{i+1} \equiv Y_i R$, using the previous equations the vector Y fulfills all the equations (except the normalizing one). We observe that $(\sum_{i \in \mathbb{N}} R^i)[u, u']$ counts the mean number of visits of states $\{(i, u') \mid i \geq 1\}$ starting from $(1, u)$ before stopping. Since the Markov chain is ergodic, this number is finite and so $||Y||$ is also finite (the third condition). We can normalize it and obtains the steady-state distribution.

There are different ways to compute matrix R. A first possibility is to observe that R is the (upper) limit of R_n defined by $R_n \equiv \sum_{k \leq n} V_2^{(k)}$ which fulfill $R_0 = 0$ and $R_{n+1} = A_0 + R_n A_1 + (R_n)^2 A_2$ (due to the previous equations). This proves that R is the smallest positive solution of $Z = \mathbf{A}_0 + Z\mathbf{A}_1 + Z^2\mathbf{A}_2$ and leads to an iterative computation which halts when enough precision has been obtained.

A second possibility is to use the equation $R = \mathbf{A}_0 + R\mathbf{A}_1 + R^2\mathbf{A}_2$. By ergodicity of the Markov chain, matrix $Id - \mathbf{A}_1$ is invertible. Furthermore $(Id - \mathbf{A}_1)^{-1} = \sum_{n \in \mathbb{N}} \mathbf{A}_1^n$ is non negative. Let us define $R'_0 \equiv 0$ and for all n, $R'_{n+1} \equiv (A_0 + R'^2_n \cdot A_2)(Id - \mathbf{A}_1)^{-1}$. Then by induction:

- R'_n is a increasing sequence of non negative matrices.
- $R'_n \leq R$

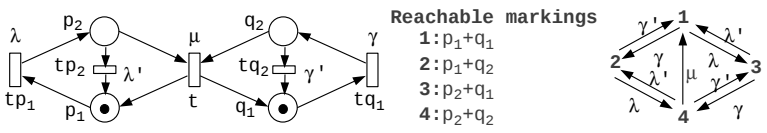
Thus the sequence R'_n is bounded and so converges to the smallest positive solution of $Z = \mathbf{A}_0 + Z\mathbf{A}_1 + Z^2\mathbf{A}_2$ which is exactly R.

This procedure has also been successfully employed to approximate bounded Petri nets where a place reaches huge values by a Petri net with a single unbounded Petri net. If furthermore the system fulfills a set of conditions for quasi-reversibility [Kel79], this approximation becomes an exact result [Hav93].

2.4.3 Composition of stochastic Petri nets

An usual way to cope with the complexity of systems is to analyze their structure in order to design more efficient algorithms than the usual ones [Pla85]. In this section, we show that when a SPN is obtained as a composition of two SPNs, such a method is possible [Don94].

Let us examine the net below. It can been seen as the composition of two subnets $(p_1, p_2, tp_1, tp_2, t)$ and $(q_1, q_2, tq_1, tq_2, t)$ synchronized by transition t.



The infinitesimal generator Q described below presents some regularities that can be emphasized by an additive decomposition between the behaviours generated by the local transitions and the one generated by the synchronized transition.

$$\begin{pmatrix} -(\gamma+\lambda) & \gamma & \lambda & 0 \\ \gamma' & -(\gamma'+\lambda) & 0 & \lambda \\ \lambda' & 0 & -(\gamma+\lambda') & \gamma \\ \mu & \lambda' & \gamma' & -(\gamma'+\lambda'+\mu) \end{pmatrix}$$

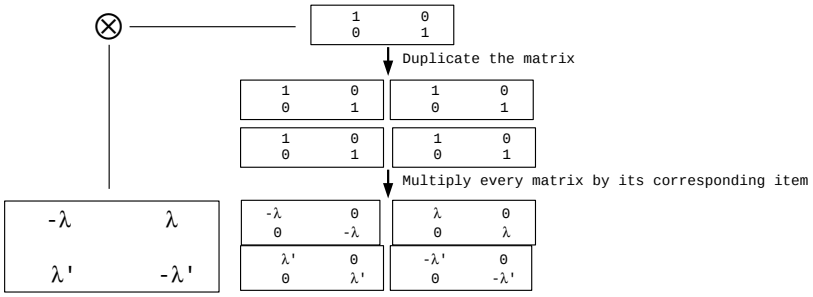
$$Q = \begin{pmatrix} -\lambda & 0 & \lambda & 0 \\ 0 & -\lambda & 0 & \lambda \\ \lambda' & 0 & -\lambda' & 0 \\ 0 & \lambda' & 0 & -\lambda' \end{pmatrix} + \begin{pmatrix} -\gamma & \gamma & 0 & 0 \\ \gamma' & -\gamma' & 0 & 0 \\ 0 & 0 & -\gamma & \gamma \\ 0 & 0 & \gamma' & -\gamma' \end{pmatrix} + \mu \cdot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 \end{pmatrix}$$

In the general case, the infinitesimal generator can be decomposed as the sum of:

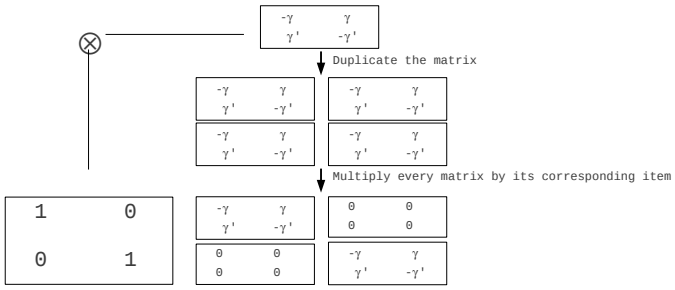
- one matrix per subnet Q_i *(here two components)*
- one matrix per every synchronized transition B_t with coefficients in $\{-1, 0, 1\}$ multiplied by the rate of the transition *(here a single transition)*

In order to exploit the regularity of these matrices, we introduce a key operation: the tensorial product. Given two matrices, A and B with dimension $m \times n$ and $m' \times n'$, the tensorial product $A \otimes B$ is built as follows. First one builds a matrix with dimension $mm' \times nn'$ by duplicating mn times matrix B. Then the block indexed by (i, j) is multiplied by coefficient $A[i, j]$. So $A \otimes B[(i, i'), (j, j')] = A[i, j]B[i', j']$.

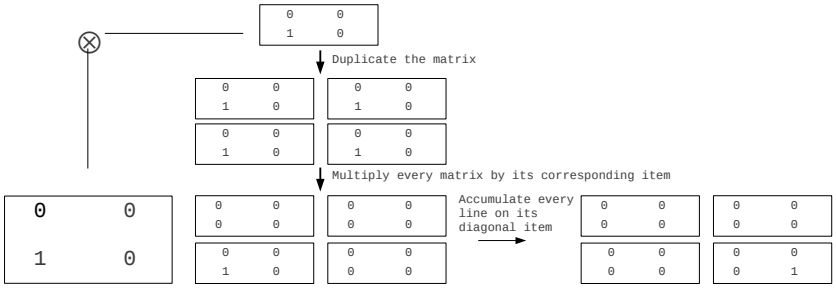
Once this operation has been defined, we observe that Q_1, the matrix associated with the first component fulfils: $Q_1 = Ql_1 \otimes Id$ where Ql_1 is the generator of the local Markov chain of the first component and Id, the identity matrix, witnesses the independence from the second component.



Q_2, the matrix associated with the second component fulfils: $Q_2 = Id \otimes Ql_2$ where Ql_2 is the generator of the local Markov chain of the second component; and Id, the identity matrix, witnesses the independence from the first component.



B_t, the matrix associated with the synchronized transition fulfils: $B_t = B_{t,1} \otimes B_{t,2} - \mathbf{D}(B_{t,1} \otimes B_{t,2})$ where $B_{t,i}$ is the indicator of local state change due to t in the ith component and $\mathbf{D}$ is the matrix operator summing the items of a line in the diagonal item.



Let us recall that the iterative computation of the steady-state distribution proceeds as follows: (1) select any initial distribution π_0, (2) iterate $\pi_{n+1} = \pi_n(Id + \frac{1}{c} \cdot Q)$ with c any value greater than $\max_i(|Q[i,i]|)$ and (3) stop when the successive values are enough close.

Observe that the iterative approach is based on the product of a vector by a matrix. When $C = A \otimes B$ computing $v \cdot C$ can be done without computing C thus saving space and more efficiently thus saving time. We do not develop the algorithm but illustrate below on an example with two matrices A, B with dimension 2×2 in such a way that the generalization is straightforward [BCDK00].

$$\begin{pmatrix} x_{11} & x_{12} & x_{21} & x_{22} \end{pmatrix} \begin{pmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{11} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{11} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{pmatrix}$$

$$= \begin{pmatrix} (x_{11}a_{11} + x_{21}a_{21})b_{11} + (x_{12}a_{11} + x_{22}a_{21})b_{21} \\ (x_{11}a_{11} + x_{21}a_{21})b_{12} + (x_{12}a_{11} + x_{22}a_{21})b_{22} \\ (x_{11}a_{12} + x_{21}a_{22})b_{11} + (x_{12}a_{12} + x_{22}a_{22})b_{21} \\ (x_{11}a_{12} + x_{21}a_{22})b_{12} + (x_{12}a_{12} + x_{22}a_{22})b_{22} \end{pmatrix}$$

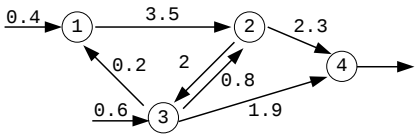
$$= \begin{pmatrix} z_{11}b_{11} + z_{12}b_{21} \\ z_{11}b_{12} + z_{12}b_{22} \\ z_{21}b_{11} + z_{22}b_{21} \\ z_{21}b_{12} + z_{22}b_{22} \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} z_{11} \\ z_{12} \end{pmatrix} \cdot B \\ \begin{pmatrix} z_{21} \\ z_{22} \end{pmatrix} \cdot B \end{pmatrix}$$

where $\begin{pmatrix} z_{11} & z_{21} \end{pmatrix} = \begin{pmatrix} x_{11} & x_{21} \end{pmatrix} \cdot A$ and $\begin{pmatrix} z_{12} & z_{22} \end{pmatrix} = \begin{pmatrix} x_{12} & x_{22} \end{pmatrix} \cdot A$

Thus in this particular case, the algorithm consists of (1) decomposing the vector x in two subvectors (x_{11}, x_{21}) and (x_{12}, x_{22}), (2) multiply them by A giving two subvectors (z_{11}, z_{21}) and (z_{12}, z_{22}), (3) switch some components leading to subvectors (z_{11}, z_{12}) and (z_{21}, z_{22}) and (4) multiply them by B and concatenate the results leading to $x \cdot A \otimes B$.

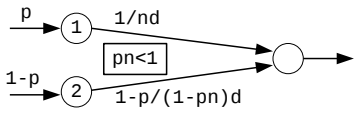
2.4.4 Phase-type stochastic Petri nets

A phase-type distribution [Neu81] is defined by a Markov chain with an absorbing state (*i.e.* without successor) and an initial distribution. If F denotes the distribution then $F(t)$ is the probability to be in the absorbing state at time t. Using our analysis of Markov chains, F is a probability distribution if and only if the absorbing state is the single BSCC of the graph associated with the chain. The states of the chain (except the last one) are called stages.



It has been established that in some sense, every distribution is a limit of phase-type distributions [Cox55]. For instance, an exponential distribution is a phase-type distribution with a single stage and an immediate distribution is a phase-type distribution without stage. A deterministic distribution with duration d is approximated by an Erlang distribution with n consecutive stages whose rate is $\frac{n}{d}$. The mean value of the absorbing time is d and the variance of the absorbing time is $\frac{d^2}{n}$; so it goes to 0 when n goes to ∞ (the coefficient of variation is $\frac{1}{\sqrt{n}}$).

As another example, let d be the mean of some distribution with a great coefficient of variation. We approximate such a distribution by some *hyper-exponential* distribution defined by a probabilistic choice between exponential distributions with different rates. Thus the mean value of the absorbing time is the weighted average of mean values of these distributions. The coefficient of variation is always greater than 1. For instance, such a distribution can be approximated by a three-state hyperexponential distribution (with two parameters p, n).



So the phase-type stochastic Petri nets (PH-SPN) have a great expressive power [Cum85, CBB89]. However, such a net generates a stochastic process of the same kind as the one of GSPN. Indeed, the sampling of a phase-type distribution can be seen as a random sampling of the choice of the first stage, a sampling of the exponential distribution of the stage, a new random sampling of the choice of the next stage, etc. until one reaches the absorbing state. So,

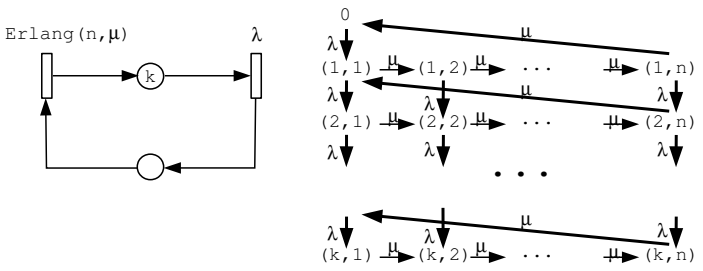
rather than considering transition firings as the events of the SED, one selects a more elementary step: the stage change of the distributions. This requires to complete the state of the SED. A state is defined by:

- a marking,
- for every transition, a descriptor which includes a sequence of samplings not yet used to fire the transition. For every sampling, one memorizes its current stage. If the transition works with the *enabling memory* policy, the number of firings is exactly the number of services offered by the transition. If it works with the *age memory* policy, this number can be greater since one takes into account the suspended services.

Every step that reaches an absorbing state is an *external* transition since it updates the marking. The new descriptor is computed w.r.t. the different policies of the net. The *internal* transitions let the marking unchanged and in the descriptor a single sampling is updated.

One builds the semi-Markovian process as a reachability graph starting from the initial state and *firing* the internal and external transitions. More elaborate constructions are possible by noting that, for instance, some markings lead to the same set of descriptors.

We illustrate this construction on the net below which has a transition with an Erlang distribution and a transition with an exponential distribution. Both transitions follow the single-server policy. So the descriptor associated with the non exponential transition is nothing more than the current stage of the distribution. On the left part of the figure, the associated Markov chain is presented. Since the sum of tokens in the net remains constant, The marking is denoted by the number of tokens in the input place of the Erlang transition. The external transitions are represented by thick lines.



However the problem is the number of states of this process which has the same magnitude order as the product of the size of the reachability space and

of the number of descriptors. Fortunately, as may be observed on the example, the Markov chain of the a PH-SPN presents regularities.

In order to exploit these regularities we proceed as follows (for more details see [HMC97, DHM98]). Markings with same enabled phase-type transitions are grouped. Let $\mathcal{M}$ be such a class of markings with $t_1, \ldots, t_d$, the corresponding enabled transitions. Then the corresponding states say $\mathcal{S}_\mathcal{M}$ in the Markov chain are $(m, q_1, \ldots, q_d)$ with $m \in \mathcal{M}$ and q_i being a stage of the distribution of t_i. Here for simplicity, we assume enabling memory and single-server policies.

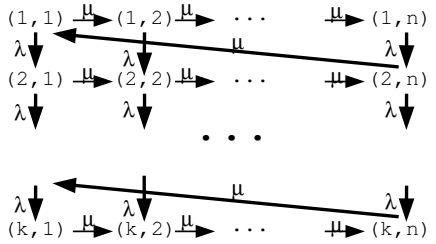
The goal of this decomposition is, given two classes $\mathcal{S}_\mathcal{M}$ and $\mathcal{S}_{\mathcal{M}'}$, to obtain a tensorial expression of the block of the infinitesimal generator Q indexed by $\mathcal{S}_\mathcal{M} \times \mathcal{S}_{\mathcal{M}'}$. Then the vector matrix multiplication involved in the iterative method for computing steady-state distribution is performed block per block where inside a block the operation is performed as in the previous section saving space and time.

The matrices occurring in the tensorial expression depend :

1. either on the change of marking by exponential transitions or phase-type transitions when reaching absorbing states *(external transitions)*

2. or on phase-type transitions that do no reach absorbing state *(internal transitions)*.

We illustrate it on our example. First we observe that there are two classes depending on the enabling of the Erlang distribution. We focus on the class $\mathcal{S}_\mathcal{M}$ for which the Erlang transition is enabled and on the block $\mathcal{S}_\mathcal{M} \times \mathcal{S}_\mathcal{M}$ which corresponds to the subgraph depicted below.



This block can be expressed as:

$$Q' = E \otimes Id + Id \otimes L + \mu(F \otimes G - \mathbf{D}(F \otimes G))$$

- where E corresponds to the firing of the exponential transition and Id represents the fact that this firing let unchanged the stage of the Erlang distribution;
- where L corresponds to the internal change of the stage and Id represents the fact that the marking is unchanged;
- and F, G correspond to the firing of the phase-type transition: F changes the marking and G moves the current stage to the initial stage.

We have detailled the matrices below.

$$E = \begin{pmatrix} -\lambda & \lambda & 0 & 0 & \dots & 0 \\ 0 & -\lambda & \lambda & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & 0 & -\lambda & \lambda \\ 0 & \dots & 0 & 0 & 0 & 0 \end{pmatrix} \quad L = \begin{pmatrix} -\mu & \mu & 0 & 0 & \dots & 0 \\ 0 & -\mu & \mu & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & 0 & -\mu & \mu \\ 0 & \dots & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$F = \begin{pmatrix} 0 & 0 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & 1 & 0 & 0 \\ 0 & \dots & 0 & 0 & 1 & 0 \end{pmatrix} \quad G = \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & \dots & 0 & 0 & 0 & 0 \\ 1 & \dots & 0 & 0 & 0 & 0 \end{pmatrix}$$

Chapter 3

Quantitative Verification of Stochastic Models

3.1 Introduction

Hardware and software systems are more and more pervasive in every day life and, therefore, there is an obvious demand for these systems to meet the functional and performance requirements that the users expect. Automatic verification methods are a possible, and doable, way to increase our level of confidence in the systems that we design and produce, both in terms of functionality (what the system does) and performance (how long does it take). In the previous chapter, we have put the focus on performance evaluation but here we give an overview of quantitative verification, a topic that unifies performance evaluation and verification, namely verification methods that take into account the randomness of systems work with a model of the system which is a stochastic process. In order to limit the complexity of the verification process, these stochastic processes are often either Discrete Time Markov Chains (DTMC) or Continuous Time Markov Chains (CTMC), usually automatically generated by some higher level formalism like stochastic Petri nets or stochastic process algebras.

Historically the functional verification and the evaluation of performance of an application have been considered as two distinct steps of the system development and verification process: each steps had its own moddel and associated verification techniques. In the last fifteen years instead we have seen the flourishing of a discipline that aims at taking simultaneously into consideration both aspects and that is often referred to as probabilistic verification or, more appropriately, of verification of probabilistic systems. The moving force of the discipline is the need of being able to evaluate the probability of a property ex-

pressed as a logic formula. To show why this is an important need, we recall a classical example from system reliability.

Consider a system whose states can be partitioned in three classes: W, the states in which the system works properly, D, the states in which the system is still working, although in a degraded mode, and F, the states in which the system is not working (failure states). The system can evolve from W states to D or F states, and from D to F states. A classical reliability measure for such a system is the probability of being in a F state within a given time interval I. classical performance and reliability methods can be easily applied to compute such probability.

If instead we ask for a slightly more refined question, as the probability of failing within I, *given that the system has not passed through a degraded mode of operation*, then we need to express and compute the probability of reaching an F state within I, passing only through W states. A temporal logic (as CSL for example) has temporal operators that allow a simple, and semantically well-founded definition for the above property. In this particular case the formula is: $P_{\leq p}(W\ \mathcal{U}^I F)$ where p is the upper limit of the probability of such an event as fixed by the designer.

This chapter presents the two main themes of probabilistic verification: the temporal logic to express probabilistic verification properties and the techniques to verify such properties for Markov chains (and some more general stochastic models). This chapter is divided in two parts. The first part is devoted to numerical verification, *i.e.* the probabilities (or other related quantities) to be computed are obtained through a numerical algorithm. In the second part, such quantities are computed through samplings with the help of simulations, and this technique is called statistical verification.

3.2 Numerical Verification

3.2.1 Verification of Discrete Time Markov Chain

Temporal logics for Markov chains

We consider a « probabilistic » extension of the CTL^* logic, that is named $PCTL^*$. The syntax of this logic is defined inductively upon state formulas and paths formulas.

Definition 3.1. *Let $\mathcal{P}$ be the set of atomic propositions.*
A $PCTL^*$ state formula (relative to $\mathcal{P}$) is defined by:

E_1: *If $\phi \in \mathcal{P}$ then ϕ is a $PCTL^*$ state formula;*

E_2: *If ϕ and ψ are $PCTL^*$ state formulas then $\neg\phi$ and $\phi \wedge \psi$ are $PCTL^*$ state formulas;*

E_3: *If φ is a $PCTL^*$ path formula, $a \in [0,1]$ is a rational number, and $\bowtie \in \{=, \neq, < , \leq, >, \geq\}$ then $P_{\bowtie a}\varphi$ is a $PCTL^*$ state formula.*

A path formula of $PCTL^*$ (relative to $\mathcal{P}$) is defined by:

C_1: *A $PCTL^*$ state formula is a $PCTL^*$ path formula;*

C_2: *if φ and θ are $PCTL^*$ path formulas, then $\neg\varphi$ and $\varphi \wedge \theta$ are $PCTL^*$ path formulas;*

C_3: *If φ and θ are $PCTL^*$ path formulas, then $\mathcal{X}\varphi$ and $\varphi\mathcal{U}\theta$ are $PCTL^*$ path formulas.*

Two subsets of the $PCTL^*$ formulas are of particular interest. The first subset is called $PCTL$ (by analogy with CTL) and it is built using only the rules E_1, E_2, E_3, C_3' where C_3' is defined as « If ϕ and ψ are $PCTL$ state formulas, the $\mathcal{X}\phi$ and $\phi\mathcal{U}\psi$ are $PCTL$ path formulas ». The second subset is called $PLTL$ (by analogy with LTL) and it is built only on the rules E_1, E_3, C_1', C_2, C_3 where C_1' is « If $\varphi \in \mathcal{P}$ then φ is a $PLTL$ state formula ».

We now explain how to evaluate the truth value of a $PCTL$, $PLTL$, or $PCTL^*$ formula.

The semantics of formulas is given in the following. We consider a Markov chain $\mathcal{M}$ whose states are labeled by a subset of atomic propositions. We indicate with s a state of the chain and with $\sigma = s_0, s_1, \ldots$ an infinite path in the graph associated to the chain. We denote σ_i the suffix $s_i, s_{i+1}, \ldots$, and $\mathcal{M}, s \models \phi$ the satisfaction of state formula ϕ by state s and $\sigma \models \varphi$ the satisfaction of path formula φ by path σ.

Definition 3.2. *Let $\mathcal{M}$ be a Markov chain, s a state of the chain, and σ a path of the chain.*
The satisfaction of the state formula ϕ by s is inductively defined by:

- *if $\phi \in \mathcal{P}$ then $\mathcal{M}, s \models \phi$ iff s is labeled by ϕ;*
- *if $\phi \equiv \neg\psi$ then $\mathcal{M}, s \models \phi$ iff $\mathcal{M}, s \not\models \psi$;*
- *$\phi \equiv \psi_1 \wedge \psi_2$ then $\mathcal{M}, s \models \phi$ iff $\mathcal{M}, s \models \psi_1$ and $\mathcal{M}, s \models \psi_2$;*
- *If $\phi \equiv P_{\bowtie a}\varphi$ then $\mathcal{M}, s \models \phi$ iff $\Pr(\{\sigma \models \varphi\} \mid s_0 = s) \bowtie a$.*

The satisfaction of a path formula φ by σ is inductively defined by:

- *If φ is a state formula, then $\sigma \models \varphi$ iff $\mathcal{M}, s_0 \models \varphi$;*
- *If $\varphi \equiv \neg\theta$ then $\sigma \models \varphi$ iff $\sigma \not\models \theta$;*
- *If $\varphi \equiv \theta_1 \wedge \theta_2$ then $\sigma \models \varphi$ iff $\sigma \models \theta_1$ and $\sigma \models \theta_2$;*
- *If $\varphi \equiv \mathcal{X}\theta$ then $\sigma \models \varphi$ iff $\sigma_1 \models \theta$;*
- *If $\varphi \equiv \theta_1 \mathcal{U} \theta_2$ then $\sigma \models \varphi$ iff $\exists i\ \sigma_i \models \theta_2$ and $\forall j < i\ \sigma_j \models \theta_1$.*

This semantic implicitly assumes that the set of paths that verify a formula is measurable. This hypothesis is justifiable, as can be proved through basic results of measure theory, but this goes beyond the scope of this chapter.

Verification of $PCTL$ formulas

Given a DTMC and a $PCTL$ formula ϕ the verification algorithm proceeds by evaluating bottom up the sub-formulas of the syntactic tree of ϕ, from the leaves up to the root. At each step the algorithm evaluates a sub-formula considering as atomic propositions the operands of the most external operator (of the sub-formula associated to the tree node considered).

Considering the syntax of $PCTL$ the formulas to be considered are: $\neg\psi, \psi \wedge \chi, P_{\bowtie a}\mathcal{X}\psi, P_{\bowtie a}\psi\mathcal{U}\chi$ where ψ and χ are (formulas transformed into) atomic propositions. We now provide an informal explanation of the algorithm and its correctness.

$\boxed{\phi = \neg\psi}$ The algorithm labels with ϕ each state not labeled with ψ.

$\boxed{\phi = \psi \wedge \chi}$ The algorithm labels with ϕ each state labeled with ψ and χ.

$\boxed{\phi = P_{\bowtie a}\mathcal{X}\psi}$ The algorithm computes the probability p_s of reaching in a single step a state labeled with ψ, with $p_s \equiv \sum_{s' \models \psi} \mathbf{P}[s, s']$ where $\mathbf{P}$ is the transition matrix of the DTMC. State s is then labeled with ϕ iff $p_s \bowtie a$.

$\boxed{\phi = P_{\bowtie a}\psi\mathcal{U}\chi}$ The algorithm computes the probability of reaching a state labeled by χ, passing only through states labeled by ψ. Let p_s be such a probability. If $s \models \chi$ then $p_s = 1$; if $s \not\models \chi$ and $s \not\models \psi$ then $p_s = 0$. In all other cases, p_s is computed on a transformed DTMC: all the states described above are made absorbing, and then the probability of reaching χ from s in the new chain. Since each χ state is a BSCC, such a probability can be computed as explained in the previous chapter, and illustrated in figure 3.1. State s is then labeled with ϕ iff $p_s \bowtie a$.

Aggregation of Markov chains

In order to establish the correctness of the verification algorithm of PLTL, we recall the notions of aggregation in Markov chains. The aggregation of finite

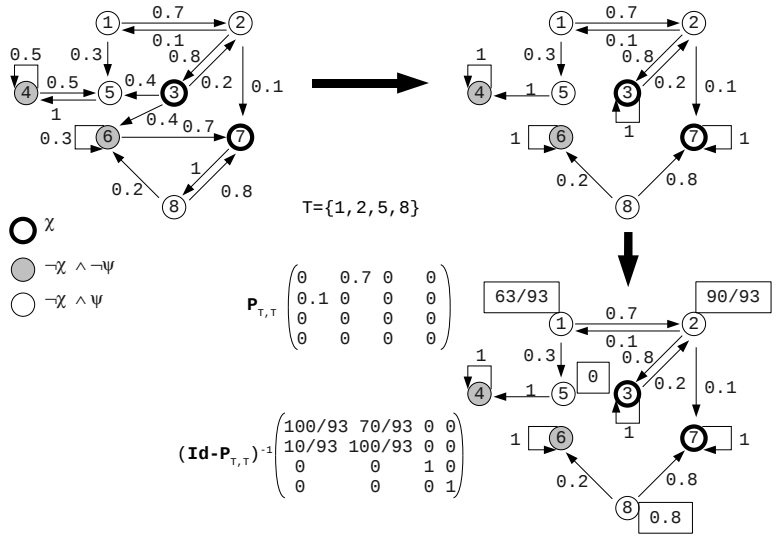


Figure 3.1: Computation of $P_{\bowtie a}\psi\mathcal{U}\chi$

Markov chains is an efficient method when one is faced to huge chains [KS60]. Its principe is simple: substitute to a chain, an "equivalent" chain where each state of the lumped chain is a set of states of the initial chain. There are different versions of aggregation depending on whether the aggregation is sound for every initial distribution (*strong aggregation*) or for at least one distribution (*weak aggregation*). We simultaneously introduce aggregation for DTMCs and CTMCs. We note π_0 the initial distribution of the chain and X_n (resp. X_τ) the random variable describing the state of the DTMC (resp. CTMC) at time n (resp. τ). $\mathbf{P}$ is the transition matrix of the DTMC and $\mathbf{Q}$ is the infinitesimal generator of the CTMC.

Definition 3.3. *Let $\mathcal{M}$ be a DTMC (resp. a CTMC) and $\{X_n\}_{n\in\mathbb{N}}$ (resp. $\{X_\tau\}_{\tau\in\mathbb{R}^+}$) the family of corresponding random variables. Let $\{S_i\}_{i\in I}$ be a partition of the state space. Define the random variable Y_n for $n \in \mathbb{N}$ (resp. Y_τ for $\tau \in \mathbb{R}^+$) by $Y_n = i$ iff $X_n \in S_i$ (resp. $Y_\tau = i$ iff $X_\tau \in S_i$). Then:*

- $\mathbf{P}$ *(resp.* $\mathbf{Q}$*) is* strongly lumpable *w.r.t.* $\{S_i\}_{i\in I}$
  iff there exists a transition matrix $\mathbf{P}^{lp}$ (resp. an infinitesimal generator $\mathbf{Q}^{lp}$) s.t
  $\forall\pi_0$ $\{Y_n\}_{n\in\mathbb{N}}$ (resp. $\{Y_\tau\}_{\tau\in\mathbb{R}^+}$) is a DTMC (resp. CTMC)
  with transition matrix $\mathbf{P}^{lp}$ (resp. with infinitesimal generator $\mathbf{Q}^{lp}$).

- $\mathbf{P}$ *(resp.* $\mathbf{Q}$*) is* weakly lumpable *w.r.t.* $\{S_i\}_{i\in I}$
  iff $\exists \pi_0$ $\{Y_n\}_{n\in\mathbb{N}}$ *(resp.* $\{Y_\tau\}_{\tau\in\mathbb{R}^+}$*) is a DTMC (resp. CTMC).*

While a characterization of the strong aggregation by examination of the transition matrix or the infinitesimal generator is easy, the search of a weak aggregation is much harder [Led60]. So we introduce exact aggregation, a simple case of weak aggregation.

Definition 3.4. *Let* $\mathcal{M}$ *be a DTMC (resp. a CTMC) and* $\{X_n\}_{n\in\mathbb{N}}$ *(resp.* $\{X_\tau\}_{\tau\in\mathbb{R}^+}$*) the family of corresponding random variables. Let* $\{S_i\}_{i\in I}$ *be a partition of the state space. Define the random variable* Y_n *for* $n \in \mathbb{N}$ *(resp.* Y_τ *for* $\tau \in \mathbb{R}^+$ *) by* $Y_n = i$ *iff* $X_n \in S_i$ *(resp.* $Y_\tau = i$ *iff* $X_\tau \in S_i$*). Then:*

- *A initial distribution* π_0 *is equiprobable w.r.t.* $\{S_i\}_{i\in I}$
  if $\forall i \in I, \forall s, s' \in S_i, \pi_0(s) = \pi_0(s')$.

- $\mathbf{P}$ *(resp.* $\mathbf{Q}$*) is* exactly lumpable *w.r.t.* $\{S_i\}_{i\in I}$
  iff there exists a transition matrix $\mathbf{P}^{lp}$ *(resp. an infinitesimal generator* $\mathbf{Q}^{lp}$*) s.t.*
  $\forall \pi_0$ *equiprobable* $\{Y_n\}_{n\in\mathbb{N}}$ *(resp.* $\{Y_\tau\}_{\tau\in\mathbb{R}^+}$*) is a DTMC (resp. CTMC)*
  with transition matrix $\mathbf{P}^{lp}$ *(resp. with infinitesimal generator* $\mathbf{Q}^{lp}$*)*
  and π_n *(resp.* π_τ*) is equiprobable w.r.t.* $\{S_i\}_{i\in I}$.

Exact and strong aggregations have simple characterizations [Sch84] stated in the next proposition.

Proposition 3.1. *Let* $\mathcal{M}$ *be a DTMC (resp. a CTMC) and* $\mathbf{P}$ *(resp.* $\mathbf{Q}$*) the corresponding transition matrix (resp. the corresponding infinitesimal generator). Then:*

- $\mathbf{P}$ *(resp.* $\mathbf{Q}$*) is strongly lumpable w.r.t.* $\{S_i\}_{i\in I}$ *iff*
  $\forall i, j \in I\ \forall s, s' \in S_i \sum_{s''\in S_j} \mathbf{P}[s, s''] = \sum_{s''\in S_j} \mathbf{P}[s', s'']$
  (resp. $\sum_{s''\in S_j} \mathbf{Q}[s, s''] = \sum_{s''\in S_j} \mathbf{Q}[s', s'']$*)*

- $\mathbf{P}$ *(resp.* $\mathbf{Q}$*) is exactly lumpable w.r.t.* $\{S_i\}_{i\in I}$ *iff*
  $\forall i, j \in I\ \forall s, s' \in S_i \sum_{s''\in S_j} \mathbf{P}[s'', s] = \sum_{s''\in S_j} \mathbf{P}[s'', s']$
  (resp. $\sum_{s''\in S_j} \mathbf{Q}[s'', s] = \sum_{s''\in S_j} \mathbf{Q}[s'', s']$*)*

Proof
We prove the first point and let to the reader the similar proof of the second point.

Assume that the condition is fulfilled, let π_n the distribution of X_n at time n.
Define $\mathbf{P}^{lp}[i, j] = \sum_{s'\in S_j} \mathbf{P}[s, s']$ for an arbitrary $s \in S_i$ (well defined using the condition). Then:
$\sum_{s\in S_i} \pi_{n+1}(s) = \sum_{s\in S_i} \sum_j \sum_{s'\in S_j} \pi_n(s')\mathbf{P}[s', s] =$

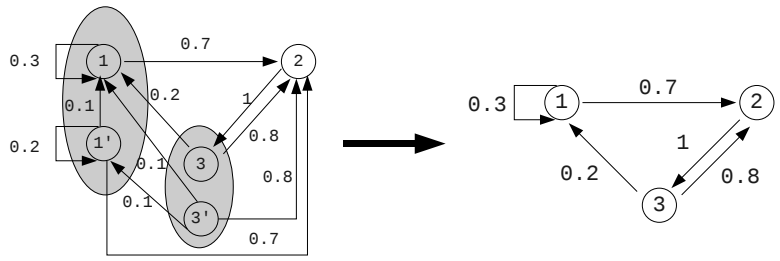


Figure 3.2: An example of strong aggregation in a DTMC

$\sum_j \sum_{s' \in S_j} \pi_n(s') \sum_{s \in S_i} \mathbf{P}[s', s] = \sum_j (\sum_{s' \in S_j} \pi_n(s')) \mathbf{P}^{lp}[j, i]$
This establishes that the condition is sufficient.

Assume now that the condition is not fulfilled,
$\exists i, j \in I \; \exists s, s' \in S_i \sum_{s'' \in S_j} \mathbf{P}[s, s''] \neq \sum_{s'' \in S_j} \mathbf{P}[s', s'']$
Let $\pi_{0,s}$ and $\pi_{0,s'}$ be the initial point distributions for s and s'. These two distributions lead to the same Y_0. Then:
$\sum_{s'' \in S_j} \pi_{1,s}(s'') = \sum_{s'' \in S_j} \mathbf{P}[s, s''] \neq \sum_{s'' \in S_j} \mathbf{P}[s', s''] = \sum_{s'' \in S_j} \pi_{1,s'}(s')$
This proves that matrix $\mathbf{P}^{lp}$ cannot exist.

◊◊◊

Figure 3.2 illustrates the concept strong aggregation in case of a DTMC.

When the condition of strong aggregation is fulfilled the transition matrix (resp. the infinitesimal generator) of the lumped chain can be directly computed from the transition matrix (resp. from the infinitesimal generator) of the initial chain as stated by the next proposition (immediate consequence of the proof of proposition 3.1).

Proposition 3.2. *Let $\mathcal{M}$ be a DTMC (resp. a CTMC) strongly lumpable w.r.t. $\{S_i\}_{i \in I}$. Let $\mathbf{P}^{lp}$ (resp. $\mathbf{Q}^{lp}$) be the transition matrix (resp. the infinitesimal generator) associated with the lumped chain then:*

$\forall i, j \in I, \forall s \in S_i, \mathbf{P}^{lp}[i, j] = \sum_{s' \in S_j} \mathbf{P}[s, s']$ *(resp.* $\mathbf{Q}^{lp}[i, j] = \sum_{s' \in S_j} \mathbf{Q}[s, s']$*)*

As for strong aggregation, in case of exact aggregation the transition matrix (resp. the infinitesimal generator) of the lumped chain can be directly computed from the transition matrix (resp. from the infinitesimal generator) of the initial chain. Observe that starting with an initial distribution equidistributed over the states of every subset of the partition, at any time the distribution is

equidistributed. Consequently, if the DTMC (resp. the CTMC) is ergodic, its stationary distribution is equidistributed over the states of every subset of the partition. Otherwise stated, knowing the transition matrix (resp. the infinitesimal generator) of the lumped chain, one can compute its stationary distribution, and deduce (by *local* equidistribution) the stationary distribution of the initial chain. This last step is impossible with strong aggregation which does not ensure equiprobability of states inside a subset.

Proposition 3.3. *Let $\mathcal{M}$ be a DTMC (resp. a CTMC) which is exactly lumpable w.r.t. $\{S_i\}_{i\in I}$. Let $\mathbf{P}^{lp}$ (resp. $\mathbf{Q}^{lp}$) be the transition matrix (resp. the infinitesimal generator) associated with the lumped chain, then:*

- $\forall i,j \in I, \forall s \in S_j\ \mathbf{P}^{lp}[i,j] = (\sum_{s'\in S_i} \mathbf{P}[s',s]) \times (|S_j|/|S_i|)$
  (resp. $\mathbf{Q}^{lp}[i,j] = (\sum_{s'\in S_i} \mathbf{Q}[s',s]) \times (|S_j|/|S_i|)$*)*
- *If* $\forall i \in I, \forall s,s' \in S_i, \pi_0(s) = \pi_0(s')$ *then*
  $\forall n \in \mathbb{N}$ *(resp.* $\forall t \in \mathbb{R}^+$*),* $\forall i \in I, \forall s,s' \in S_i$, $\pi_n(s) = \pi_n(s')$ *(resp.* $\pi_\tau(s) = \pi_\tau(s')$*),*
  where π_n (resp. π_τ) is the probability distribution at time n (resp. τ)
- *If $\mathbf{P}$ (resp. $\mathbf{Q}$) is ergodic and π is its stationary distribution then*
  $\forall i \in I, \forall s,s' \in S_i, \pi(s) = \pi(s')$

Verification of $PLTL$ formulas

Given a DTMC $\mathcal{M}$ and a $PLTL$ formula ϕ, by definition ϕ is either an atomic proposition, or $P_{\bowtie a}\varphi$ where φ is a path formula built on the operators $\mathcal{X}, \mathcal{U}$ and on atomic propositions. The first case is straightforward, while we describe the second case in the following.

As in the previous case, the evaluation proceeds by evaluating the subformulas of φ in the order given by a bottom-up visit of the syntactical tree of the formula. Here after each subformula evaluation transforms both the formula and the DTMC such that at the end the formula becomes an atomic proposition whose evaluation is straightforward. The evaluated subformula φ' is substituted by the atomic proposition $[\varphi']$ in the formula itself.

The transformation of the DTMC is more complex. We describe it in the following for the most complex case of a subformula $\varphi' \equiv \psi \mathcal{U} \chi$. Every state s such that $0 < \Pr(\sigma \models \varphi' \mid s_0 = s) < 1$ of the original DTMC is duplicated into s^y, labeled by the propositions labeling s and $[\varphi']$ and s^n labeled by the propositions labeling s. All other states are labeled according to the value of the same probability formula, either 0 or 1. The above probabilities are computed with the same procedure as for $PCTL$. S_o will denote the states that are not duplicated.

The transition probability matrix of the new DTMC is defined as follows:

- The transition probability between states of S_0 is left unchanged as well.
- For all duplicated states, let $py(s) = \Pr(\sigma \models \varphi' \mid s_0 = s)$ and $pn(s) = 1 - py(s)$. The probability to move from a state s' of the original chain to a state s^y (resp. s^n) is the probability of moving from s' to s in the original chain, multiplied by $py(s)$ (resp. $pn(s)$).
- From states s^y (resp. s^n) the chain can only move towards duplicated states s'^y (resp. s'^n) or towards states s' of the original chain such that $py(s') = 1$ (resp. $pn(s') = 1$). The associated transition probabilities are defined by $\mathbf{P}'[s^y, s'^y] = \mathbf{P}[s, s']py(s')/py(s)$ and $\mathbf{P}'[s^y, s'] = \mathbf{P}[s, s']/py(s)$, similarly for the states s^n.

To complete the definition of the transformed chain we need to define the initial probability of a state s^y (resp. s^n) given that the system starts in state s. This conditional probability is given by $py(s)$ (resp. $pn(s)$). Consequently, $\pi'_0(s^y) = py(s)\pi_0(s)$ and $\pi'_0(s^n) = pn(s)\pi_0(s)$.

Observe that $\mathbf{P}'$ is indeed a transition matrix. We prove it only for a relevant case.
$\sum_{s' \in S_o} \mathbf{P}'[s^y, s'] + \sum_{s' \in S \setminus S_o} \mathbf{P}'[s^y, s'^y] =$
$\frac{1}{py(s)} \left(\sum_{s' \in S_o, py(s')=1} \mathbf{P}[s, s'] + \sum_{s' \in S \setminus S_o} \mathbf{P}[s, s']py(s') \right)$
Examining a step of the chain, one observes that the expression between parentheses is the probability $py(s)$.

We show the DTMC transformation caused by subformula $\psi \mathcal{U} \chi$ in figure 3.3.

The correctness of this construction is established using the following lemmas.

Notations. Define the abstraction mapping abs from states of $\mathcal{M}'$ s.t. $abs(s^y) = abs(s^n) = s$ and $abs(s) = s$ for every $s \in S_o$. Define the stochastic process $\mathcal{M}^{abs}$ whose state space is the one of $\mathcal{M}$ obtained by the abstraction abs applied on $\mathcal{M}'$. The following lemma is the key point for the correctness of the algorithm.

Lemma 3.1. *The stochastic process $\mathcal{M}^{abs}$ is a weak aggregation of the process $\mathcal{M}'$ (w.r.t. the initial distribution π'_0) and it is identical to the Markov chain $\mathcal{M}$.*

Proof
Let us note π_n (resp. π'_n) the distribution of $\mathcal{M}$ (resp. $\mathcal{M}'$) at time n. We prove by recurrence on n that:
$\forall s \in S_o \; \pi_n(s) = \pi'_n(s)$ and $\forall s \in S \setminus S_o \; \pi'_n(s^y) = \pi_n(s)py(s) \wedge \pi'_n(s^n) = \pi_n(s)pn(s)$

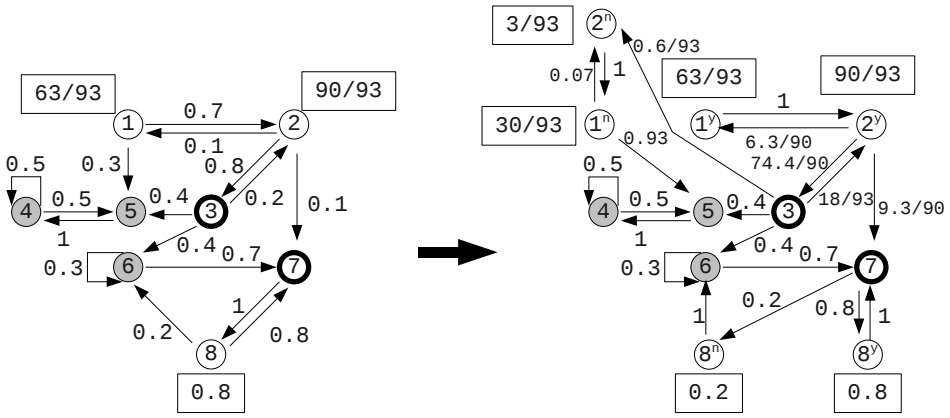


Figure 3.3: CTMC transformation for $PLTL$

For $n = 0$, this is due to the definition of π'_0. Assume that the equations are fulfilled for n. Let us prove it for $n+1$. We only handle the case of a state s^y and let to the reader the other cases.
$\pi'_{n+1}(s^y) = \sum_{s' \in S_o} \pi'_n(s')\mathbf{P}'[s, s^y] + \sum_{s'^y \in S_o} \pi'_n(s'^y)\mathbf{P}'[s'^y, s^y]$
$= \sum_{s' \in S_o} \pi_n(s')\mathbf{P}[s', s]py(s) + \sum_{s'^y | s' \in S \setminus S_o} \pi_n(s')py(s')\mathbf{P}'[s', s]\frac{py(s)}{py(s')}$
$= py(s)\left(\sum_{s' \in S_o} \pi_n(s')\mathbf{P}[s', s] + \sum_{s' \in S \setminus S_o} \pi_n(s')\mathbf{P}'[s', s]\right) = py(s)\pi_{n+1}(s)$

The result is then immediate since in $\mathcal{M}^{abs}$, $\forall s \in S \setminus S_o\ \pi^{abs}_n(s) = \pi'_n(s^y) + \pi'_n(s^n)$.

$\Diamond\Diamond\Diamond$

We note $\mathcal{M}'$ the transformed chain. A path is said *normal* if it ends in a BSCC containing a state of S_o and visits infinitely often all the states of this BSCC.

Lemma 3.2. *The set of normal paths has measure 1 in $\mathcal{M}$ and in $\mathcal{M}'$.*

Proof
Let us recall that a random path has a probability 1 to meet a BSCC and to visit infinitely often its states. Assume that there exists a BSCC (say C) in $\mathcal{M}$ or $\mathcal{M}'$ that does not include a state of S_o.

- If C belongs to $\mathcal{M}$, this means that all states of C fulfill $\neg\chi \wedge \psi$. which leads to $pn(s) = 1$ for every state s of C. Thus $C \subseteq S_o$ which is a contradiction.

- If C belongs to $\mathcal{M}'$, C only includes duplicate states. Let us pick one of this state duplicated from an original state s.

  1. This state is s^y. In $\mathcal{M}$, there is a path from s to a state s' fufilling χ such that every intermediate state fulfills $\neg\chi \wedge \psi$. This path yields a path in $\mathcal{M}'$ from s^y to s'. Consequently $s' \in C$, which is a contradiction.
  2. This state is s^n. In $\mathcal{M}$, there is (1) either a path from s to a state s' fufilling $\neg\chi \wedge \neg\psi$ such that every intermediate state fulfills $\neg\chi \wedge \psi$, (2) or a path from s to a state s' belonging to a BSCC whose every state of this BSCC fulfills $\neg\chi \wedge \psi$ and such that every intermediate state of the path fulfills $\neg\chi \wedge \psi$. In both cases, this path yields a path in $\mathcal{M}'$ from s^n to s'. Consequently $s' \in C$, which is a contradiction.

$\Diamond\Diamond\Diamond$

Let φ'' be a subformula of φ where φ' occurs. Let us note $\varphi''(\varphi' \leftarrow [\varphi'])$, the formula φ'' in which φ' has been substituted by the atomic proposition $[\varphi']$.

Lemma 3.3. *For every subformula φ'' of φ where φ' occurs, and for every normal path σ of $\mathcal{M}'$, $\sigma \models \varphi''(\varphi' \leftarrow [\varphi'])$ iff $\sigma \models \varphi''$. Consequently for a random path σ, $\Pr(\sigma \models \varphi''(\varphi' \leftarrow [\varphi']) \Leftrightarrow \varphi'') = 1$.*

Proof
The proof is done by induction on the size of φ''.

The basis case corresponds to $\varphi'' = \varphi'$.
Assume that $\sigma \models \varphi'$. This means there exists a prefix of σ such that $\neg\chi \wedge \psi$ is fulfilled by the intermediate states and χ is fulfilled by the last state. This path can be transformed by abstraction into a path of $\mathcal{M}$. Consequently one observes that no state s of the original path belongs to S_o with $py(s) = 0$. Let us show that for every state of this prefix (1) either this state belongs to S_o, say s and $py(s) = 1$ (2) or this state is a state of the kind s^y. Assume by contradiction that this claim is false and let us consider the last state of the prefix that does not fulfill the claim. This state cannot be the last state since such a state fulfills χ. Due to the previous observation, this state is of the kind s^n but (1) either its successor belongs to S_o, say s, and $py(s) = 1$, (2) or its successor is of the kind s^y. In both cases, this contradicts the construction de $\mathcal{M}'$. Thus the first state of the path fulfills $[\varphi']$.

Assume that $\sigma \models [\varphi']$. The first state of σ is either a state belonging to S_o, say s, and $py(s) = 1$, or a state of the kind s^y. An immediate successor of a state s^y can only be a state belonging to S_o, say s, and $py(s) = 1$, or a state of the kind s^y. Since σ is normal, there exists a finite prefix of σ whose intermediate

states are of the kind s^y (and thus fulfill ψ) and the last state belongs to S_o, say s and $py(s) = 1$. If $s \models \chi$ then $\sigma \models \varphi'$. Otherwise $s \models \psi$ and every successor s' of s belongs to S_o with $py(s') = 1$. Iterating this process, either one obtains a finite prefix of σ that witnesses the satisfaction of φ', or σ reaches a BSCC of $\mathcal{M}'$ such that every state s (which is infinitely often visited by σ) belongs to S_o with $py(s) = 1$ and $s \models \neg\chi$. Let us pick some state s of this BSCC and in $\mathcal{M}$ a finite path from s whose intermediate states fulfill ψ and whose last state s' fulfills χ, one obtains that s' belongs to this BSCC which is a contradiction.

For the inductive case, one first observes that a suffix of normal path is a normal path. We only develop the proof for the case $\varphi'' \equiv \theta \,\mathcal{U}\, \theta'$. Let $\sigma = s_0, s_1, \ldots$ be a normal path,

$$\begin{array}{c}
\sigma \models \varphi'' \\
\textit{iff} \\
\text{there exists } i \text{ s.t. } s_i, s_{i+1}, \ldots \models \theta' \text{ and for all } j < i,\ s_j, s_{j+1}, \ldots \models \theta \\
\textit{iff due to the inductive hypothesis and the previous observation} \\
\text{there exists } i \text{ s.t. } s_i, s_{i+1}, \ldots \models \theta'(\varphi' \leftarrow [\varphi']) \text{ and for all } j < i, \\
s_i, s_{j+1}, \ldots \models \theta(\varphi' \leftarrow [\varphi']) \\
\textit{iff} \\
\sigma \models \varphi''(\varphi' \leftarrow [\varphi'])
\end{array}$$

$\Diamond\Diamond\Diamond$

Observe that the previous lemma applies to the case $\varphi'' = \varphi$.

We establish now the correctness of the algorithm.

Theorem 3.1. *Let σ (resp. σ') be a random path of $\mathcal{M}$ (resp. $\mathcal{M}'$). Then:*

$$\Pr_{\mathcal{M}}(\sigma \models \varphi) = \Pr_{\mathcal{M}'}(\sigma' \models \varphi(\varphi' \leftarrow [\varphi']))$$

Proof
$\Pr_{\mathcal{M}}(\sigma \models \varphi) = \Pr_{\mathcal{M}^{abs}}(\sigma^{abs} \models \varphi)$
(lemma 3.1)
$= \Pr_{\mathcal{M}'}(\sigma' \models \varphi)$
Indeed the truth value of φ for a path σ' depends only on its abstraction σ^{abs}.
$= \Pr_{\mathcal{M}'}(\sigma' \models \varphi(\varphi' \leftarrow [\varphi']))$
(lemma 3.3)

$\Diamond\Diamond\Diamond$

Verification of $PCTL^*$

Given a DTMC and a formula ϕ of $PCTL^*$, the verification algorithm proceeds again through a bottom-up visit of the syntactical tree of the formula ϕ by evaluating the subtrees of ϕ that correspond to $PLTL$ formulas, substituting each verified subformula with an atomic proposition. In each step of the algorithm what needs to be evaluated is a formula of $PLTL$.

3.2.2 Verification of Continuous Time Markov Chain

Performance evaluation of systems is usually defined in a continuous context. We open this subsection with a discussion on the limits of classical performance indices, that justify the introduction of a temporal logics for performance evaluation.

Limitations of standard performance indices

The classical performance evaluation indices, recalled in the previous chapter, provide a set of important information to a system designer, but they do not capture all performance aspects of a system. As an example we consider some performance indices aimed at assessing the dependability of a system.

- *Instantaneous availability* is related to transient behavior: it represents the probability at time τ of service availability.
- *Steady-state availability* is related to steady-state behavior: it is represents the probability of service availability in steady-state.
- *Interval availability*: it represents the probability of having the service always available between time τ and τ'.
- *Steady-state interval availability*: it is the steady-state probability that the service is continuously available between two instants of time. Because we are considering the steady-state behavior, such probability does not depend on the specific points in time, but only on the duration of the interval limited by the two points.
- *Steady-state simultaneous availability and reactivity*: it is the steady-state probability that, upon a request, the system is continuously working until the service is completed and the response time does not exceed a predefined threshold.

While the first two properties can be directly and easily computed from the transient and steady-state probabilities, the computation of the other properties is more involved. It is feasible to devise, for each property, an ad-hoc computation for the probability of interest, but it is more convenient to define a general logics that can express complex performance properties, and for which a general algorithm can be designed.

A temporal logics for continuous time Markov chains

The temporal logics CSL ("Continuous Stochastic Logic") that we are going to define is an adaptation of the CTL logics ("Computation Tree Logic" [EC80]) to CTMC. The logics allow to express formulas that *are evaluated over states*, and that are built with the following syntax (in the definition we follow the approach proposed in [BHHK03a]).

Definition 3.5. *A CSL formula is inductively defined by:*

- *If $\phi \in \mathcal{P}$ then ϕ is a CSL formula;*
- *If ϕ et ψ are CSL formula then $\neg\phi$ and $\phi \wedge \psi$ are CSL formulas;*
- *If ϕ is a CSL formula, $a \in [0,1]$ is a real number, $\bowtie \in \{<, \leq, >, \geq\}$ then $S_{\bowtie a}\phi$ is a CSL formula ;*
- *If ϕ and ψ are CSL formulas, $a \in [0,1]$ is a real number, $\bowtie \in \{<, \leq, >, \geq\}$ and I is an interval of $\mathbb{R}_{\geq 0}$ then $P_{\bowtie a}\mathcal{X}^I\phi$ and $P_{\bowtie a}\phi\mathcal{U}^I\psi$ are CSL formulas.*

The first two definitions are standard CTL formulas, and we do not explain them here in more details. The formula $S_{\bowtie a}\phi$ is satisfied by a state s of the CTMC if, given that the initial state of the chain is s, the cumulative steady-state probability p of the states that satisfy ϕ, verifies $p \bowtie a$. This evaluation is well-defined, since, in a finite CTMC, a steady-state distribution always exists. If the CTMC is ergodic the evaluation of the formula does not depend on the specific state s.

An execution of a stochastic process satisfies $\mathcal{X}^I\phi$ if the first change of state takes place within the interval I and leads to a state that verifies ϕ. A state s satisfies $P_{\bowtie a}\mathcal{X}^I\phi$ if the probability p of the executions of the stochastic process that start in s and satisfy $\mathcal{X}^I\phi$ verifies $p \bowtie a$.

An execution of a stochastic process satisfies $\phi\mathcal{U}^I\psi$ if it exists a time instant $\tau \in I$ such that ψ is true at τ and for all preceding time instants ϕ is true. A state s satisfies $P_{\bowtie a}\phi\mathcal{U}^I\psi$ if the probability p of the executions that starts in s and satisfy $\phi\mathcal{U}^I\psi$ verifies $p \bowtie a$.

Using CSL, the availability and dependability properties informally defined before can be expressed in more formal terms as:

- *Instantaneous availability* guarantee of 99%:

$$P_{\geq 0.99} true \mathcal{U}^{[\tau,\tau]} disp$$

where $disp$ is an atomic proposition that indicates that the service is available.

- *Steady-state availability* guarantee of 99%:

$$S_{\geq 0.99} disp$$

- *Interval availability* guarantee of 99%:

$$P_{<0.01} true \mathcal{U}^{[\tau,\tau']} \neg disp$$

- *Steady-state interval availability* guarantee of 99%:

$$S_{<0.01} true \mathcal{U}^{[\tau,\tau']} \neg disp$$

- *Steady-state simultaneous availability and reactivity* guarantee of 99% with latency of at most 3 time units:

$$S_{\geq 0.99}(req \Rightarrow P_{\geq 0.99}(disp \mathcal{U}^{[0,3]} ack))$$

where req is the atomic proposition that indicates that a request has been received, and ack is an atomic proposition that indicates that the service has been delivered. Note that the two 99% requirements do not have the same meaning. The condition on the internal operator is a condition on the executions that starts in a particular state, while the condition on the outer operator is a global requirement on all the states of the chain, weighted by their steady-state probabilities.

Verification algorithm

Given a CTMC and a CSL formula ϕ, the algorithm evaluates the formula starting from the inner formulas and proceeding from inner to outer formulas, following bottom-up the syntactical tree of the formula ϕ and labeling each state with the sub-formulas satisfied in that state. At each step, the algorithm evaluates a formula by considering as atomic propositions the operands of the most external operator. The algorithm can be therefore explained considering one operator at a time.

$\boxed{\phi = \neg\psi}$ The algorithm labels with ϕ each state which is not labeled with ψ.

$\boxed{\phi = \psi \wedge \chi}$ The algorithm labels with ϕ every state labeled with both ψ and χ.
$\boxed{\phi = S_{\bowtie a}\psi}$ The algorithm computes the steady state distribution of the CTMC with initial probability concentrated in s (the stochastic process starts in s) as explained in the previous chapter). The probability of all states labeled with ψ are then summed up and the algorithm labels with ϕ the state s if the sum, let it be p, verifies $p \bowtie a$. Note that for all the states of a BSCC a single computation is needed: indeed either all states of the BSCC satisfy ϕ or none of them does. Similarly, if the CTMC has a single stationary distribution, then the truth value of the formula does not depend on the state.
$\boxed{\phi = P_{\bowtie a}\mathcal{X}^I\psi}$ The occurrence of a transition in a state s in within the interval I and the fact that the state reached upon the transition satisfies ψ are two independent events, and therefore the probability of the paths that satisfy the formula can be computed as the product of the probabilities of the two events. Let $I = [\tau, \tau']$; we assume a closed interval, without loss of generality (since we are in a continuous domain the fact of including or not the bounds of the interval in the computation does not influence the result). Let $\mathbf{Q}$ the infinitesimal generator of the CTMC, and $\mathbf{P}$ the matrix of the embedded DTMC. The probability of the first event is $e^{\tau\mathbf{Q}[s,s]} - e^{\tau'\mathbf{Q}[s,s]}$, while the probability of the second is $\sum_{s' \vDash \psi} \mathbf{P}[s, s']$.
$\boxed{\phi = P_{\bowtie a}\psi\mathcal{U}^I\chi}$ The evaluation of this formula requires transient analysis of a CTMC obtained from the original CTMC by some simple transformations. If X is a CTMC, then we shall indicate with X^ϕ the chain obtained by making absorbing all states of X that verify ϕ. In order to simplify the presentation, we consider as separate cases the various type of intervals.

- $\phi = P_{\bowtie a}\psi\mathcal{U}^{[0,\infty[}\chi$. In this case the executions of the chain on which we accumulate the probability should never leave the states that verify ψ, until a state that verifies χ is reached, without any constraint in time. temps. In other words, we are interested in the behavior of the chain from its initial state until it enters a state that satisfies $\neg\psi \vee \chi$. Let's consider the chain $X^{\neg\psi\vee\chi}$. If a BSCC of this chain contains a state that verifies χ then the probability that we are interested in is 1 for all states of the BSCC (since all states of a BSCC are recurrent), if no such a state exists in the BSCC, then the probability is 0. Let's call "good" a BSCC associated with a probability 1. This probability only depend on the embedded chain of $X^{\neg\psi\vee\chi}$ and its computation has already been described in the previous chapter.

- $\phi = P_{\bowtie a}\psi\mathcal{U}^{[0,\tau]}\chi$. In this case the execution of the process must visit only states that verify ψ until a state that satisfies χ s reached, and this event should happen at time τ at the latest. In other words, the probability is accumulated along the paths until a state that verifies $\neg\psi \vee \chi$ is reached.

We need therefore to compute the following probability $\Pr(X^{\neg\psi\vee\chi}(\tau) \vDash \chi \mid X^{\neg\psi\vee\chi}(0) = s)$.

- $\phi = P_{\bowtie a}\psi\mathcal{U}^{[\tau,\tau]}\chi$. In this case the execution of the process must stay in within states that verify ψ during the interval $[0,\tau]$ and it must verify χ at time τ. The case of a change of state at τ is not considered since the probability of this event is zero. The probability to be computed is equal to $\Pr(X^{\neg\psi}(\tau) \vDash \psi \wedge \chi \mid X^{\neg\psi}(0) = s)$.
- $\phi = P_{\bowtie a}\psi\mathcal{U}^{[\tau,\infty[}\chi$. In this case the execution of the process must stay in within states that verify ψ during the interval $[0,\tau]$ and then starting from the state s reached at time τ it must verify the formula $\psi\mathcal{U}^{[0,\infty[}\chi$. The probability to be computed is therefore
$\sum_{s'\vDash\psi} \Pr(X^{\neg\psi}(\tau) = s' \mid X^{\neg\psi}(0) = s) \cdot \boldsymbol{\pi}(s')$
where $\boldsymbol{\pi}(s')$ is computed using the procedure for the first case.
- $\phi = P_{\bowtie a}\psi\mathcal{U}^{[\tau,\tau']}\chi$. A similar reasoning as for the previous case leads to the following formula:
$\sum_{s'\vDash\psi} \Pr(X^{\neg\psi}(\tau) = s' \mid X^{\neg\psi}(0) = s) \cdot \Pr(X^{\neg\psi\vee\chi}(\tau'-\tau) \vDash \chi \mid X^{\neg\psi\vee\chi}(0) = s')$

3.2.3 State of the art in the quantitative evaluation of Markov chains

The field of Markov chain verification has started on the verification of DTMCs. The first approach for the verification of LTL over DTMCs (proposed in [Var85]) is conceptually very simple: the formula is translated into a Büchi automata, the non-determinism is then removed and a Rabin automata is produced. The synchronized product of this automata with the DTMC produces another DTMC, for which, using a variation of the technique explained in the previous chapter, it is possible to compute the required probability. The complexity of the computation is doubly exponential in the size of the formula. An improvement in complexity is given by the algorithm in [CY95]: a new DTMC is built iteratively from the initial DTMC, and the iteration is driven by the operators of the formula. This is the algorithm that we have presented in subsection 3.2.1. The resulting algorithm is exponential in the size of the formula, and the authors show that the algorithm has optimal complexity. A third algorithm, proposed in [CSS03], also translates the formula into a Büchi automata. Due to the particular construction followed by the algorithm, it is then possible to compute the probability associated to the formula directly on the synchronized product of the automata and of the formula. This algorithm has an optimal complexity as well, and moreover it provides better performance than the previous one in many practical cases.

A classical technique for evaluating the performance of a system consists in associating "rewards" with states and/or transitions of the chain, and in computing the mean reward or the accumulated reward at time t. Rewards are taken into account by the $PRCTL$ logics, which has been defined in [AHK03], where an evaluation algorithm is also presented.

The first relevant work on the verification of CTMCs has appeared in [ASVB96, ASVB00], where it is shown that CSL verification is decidable. The verification algorithm is extremely complex, since it does not perform the implicit approximations that we have done in the CSL verification algorithm presented in this chapter.

We should remark that verification algorithm may become impractical for large Markov chains. A possible way to solve the problem is to take advantage of a modular specification of the system, substituting a module with a smaller one, which is nevertheless equivalent with respect to the verification of the given formula. This approach has been introduced first in [BHHK03a], and it has been later generalized in [BHKW03], where various definitions of equivalence are considered.

The CSL logics that was introduced in subsection 3.2.2 has two main limitations. On one side, the path formulas are defined only in terms of atomic propositions associated to states, and not also in terms of the actions/transitions in the path. On the other side the temporal constraints on path formulas are bound to be intervals, which generates a number of limitations to the expressiveness of the temporal constraints in the formula.

In the logic CSRL introduced by [BHHK00a], CSL is extended to take into account Markov reward models, i.e. CTMCs with a single reward on states. The global reward of a path execution is then the integral of the instantaneous reward over time. In CSRL, the path operators `Until` and `Next` include also an interval specifying the allowed values for accumulated reward. Moreover new operators related to the expectation of rewards are defined. A numerical approach is still possible for approximating probability measures but its complexity is significantly increased. This formalism is also extended by rewards associated with actions [CKKP05]. CSRL is appropriate for standard performability measures but lacks expressiveness for more complex ones.

In the logic asCSL introduced by [BCH$^+$07], the single interval time constrained `Until` of CSL is replaced by a regular expression with a time interval constraint. These path formulas can now express elaborated functional requirements as in CTL* but the timing requirements are still limited to a single interval globally constraining the path execution.

In the logic CSL^{TA} introduced by [DHS09], the path formulas are defined by a single-clock deterministic time automaton. This clock can express timing re-

quirements all along the path. From an expressiveness point of view, it has been shown that CSL^{TA} is strictly more expressive than CSL and that path formulas of CSL^{TA} are strictly more expressive than those of asCSL. Finally, the verification procedure is reduced to a reachability probability in a semi-Markovian process yielding an efficient numerical procedure.

In [CHKM09], deterministic timed automata with multiple clocks are considered and the probability for random paths of a CTMC to satisfy a formula is shown to be the least solution of a system of integral equations. In order to exploit this theoretical result, a procedure for approximating this probability is designed based on a system of partial differential equations.

Observe that all of the above mentioned logics have been designed so that numerical methods can be employed to decide about the probability measure of a formula. This very constraint is at the basis of their limited expressive scope which has two aspects: first the targeted stochastic models are necessarily CTMCs; second the expressiveness of formulas is constrained (even with DTA [CHKM09], the most expressive among the logic for CTMC verification, properties of a model can be expressed only by means of clocks variables, while sophisticated measures corresponding to variables with real-valued rates cannot be considered). Furthermore observe that the evolution of stochastic logics seems to have followed two directions: one targeting *temporal reasoning* capability (evolutionary path: CSL $\rightarrow$ asCSL $\rightarrow$ $CSL^{TA}\rightarrow$ DTA), the other targeting *performance evaluation* capability (evolutionary path: CSRL $\rightarrow$ CSRL+impulse rewards). A unifying approach is currently not available, thus, for example, one can calculate the probability of a CTMC to satisfy a sophisticated temporal condition expressed with a DTA, but cannot, assess performance evaluation queries at the same time (i.e. with the same formalism).

3.3 Statistical Verification

The numerical verification approach suffers several drawbacks.

- The usual combinatory explosion has now two potential factors: the qualitative and the quantitative behavior of the system.
- The numerous techniques developed to cope with this combinatory explosion in the qualitative behavior (symmetry, partial order, decision diagrams, etc.) are still applicable but are considerably less efficient that in the discrete-event setting.
- In order to obtain validity and/or efficiency of the methods, a (semi) Markovian hypothesis is assumed that usually approximates the real behavior of

the system and may lead to large inaccuracies of the result w.r.t. the original system.

Thus a different approach [YS06] has been proposed: statistical model checking. The method is based on stochastic simulation and is quite simple: it generates samples of the behavior, i.e. paths, and simultaneously checks the satisfiability of the formula on the path sampled. Then by a simple count, it estimates (with some given confidence level) whether the satisfaction probability is below a lower threshold or above an upper threshold. This approach features the following advantages:

- One does not generate the state graph. Furthermore during generation of a path, only the current state is maintained. This greatly reduces the amount of memory required.
- The method is applicable for any stochastic process with a well defined semantics. In particular, no Markovian hypothesis is required.
- One benefits from result obtained in statistics in order to minimize the number of samples, depending on some criterion.

3.3.1 Statistical recalls

In order to understand statistical model checking, we present in this section the basics of statistics used for this method (see [BE91] for more details). As observed in the semantic of different quantitative logics, the main operation consists to compare some probability with a given threshold. From a statistical point of view, there are two ways to perform such an operation. On the one hand, one can evaluate this probability (up to some confidence interval) and then we compare it with the threshold. On the other hand without estimating this value, one simply tests whether this hypothesis is true. In both cases, the answer is equipped with a confidence level indicating the probability of an erroneous diagnosis.

Point Estimation

Estimator. We start with a random variable X and we want to evaluate $\mathrm{E}X$. A simple idea consists to sample n times the variable X and takes the average. Denoting $X_1, \ldots, X_n$ the random variables associated with these samples, one observes that this problem is a particular case of a more general problem.

Let $X_1, ..., X_n$ be independent and identically distributed (iid) random variables with $f(x \mid \theta_1, ..., \theta_k)$ their parametrized probability distribution function

(pdf), where $\theta_1, ..., \theta_k$ are the parameters of that distribution. The problem of *point estimation* is to determine for each parameter θ_i a statistic $\hat{\theta}_i = g_i(X_1, ..., X_n)$ which can be used to evaluate θ_i (i.e. assign a numerical value to θ_i) over a particular random sample $(x_1, ..., x_n)$. $\hat{\theta}_i$ is called: estimator of θ_i. In the sequel, we focus a single parameter θ and we note $\hat{\theta}^{(n)}$, the estimator for n samples. Several methods were proposed in the literature to construct estimators, the most used ones are:

- **the method of maximum likelihood.** The estimator consists to return the value θ that maximizes the density of joint distribution for $(X_1, \ldots, X_n)$. More formally:

$$\hat{\theta}^{(n)} = \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{n} f(X_i \mid \theta)$$

- **the Bayes'method.** This method assumes that the parameter θ follows some distribution giben by a density $p(\theta)$ and introduces a positive *loss function* $L(u, v)$ which is null if $u = v$. Let $\mathbf{x} = x_1, \ldots, x_n$ be the samples; one introduces a posterior density:

$$f_{\theta|\mathbf{x}}(\theta) = \frac{\prod f(x_i \mid \theta) p(\theta)}{\int \prod f(x_i \mid \theta) p(\theta) d\theta}$$

  Then $\hat{\theta}^{(n)}$ is a Bayes estimator if it minimizes the *average risk*:

$$\mathrm{E}_\theta(\mathrm{E}_{\mathbf{X}|\theta}(L(\hat{\theta}^{(n)}))) \text{ also equal to } \mathrm{E}_{\mathbf{X}}(\mathrm{E}_{\theta|\mathbf{X}}(L(\hat{\theta}^{(n)})))$$

  In particular cases, the Bayes estimator can be defined explicitly. For instance if $L(u, v) = (u - v)^2$, the Bayes estimator is:

$$\hat{\theta}^{(n)} = \int \theta d\theta f_{\theta|\mathbf{X}}(\theta) d\theta$$

- **the method of moments.** It defines an empirical distribution equidistributed over the samples and when estimating k parameters $\theta_1, ..., \theta_k$ selects the ones that make equal the k first moments of the theoretical and the empirical distribution. For instance, in case of a single parameter, the mean of the distribution, it chooses the average value of the samples $\overline{X} = 1/n \sum_{i=1}^{n} X_i$.

Since we are interested by the mean value of a (generally) unknown distribution, we choose the latter method. By construction, this estimator is *unbiased* meaning that its mean value is equal to the value of the estimated parameter.

For instance, it is also known that $S^2 = \frac{1}{n-1}\sum_{i-1}^{n}(X_i - \overline{X})^2$ is an unbiased estimator of the variance σ^2 (assuming that it exists).

An unbiased estimator gives some confidence in the estimation which is the main goal of the estimation. However several criteria have been proposed in order to assess the quality of an estimator. Here we just give two useful criteria:

- Let $\hat{\theta}^{(n)}$ and $\hat{\theta}'^{(n)}$ be two estimators of a parameter θ. $\hat{\theta}^{(n)}$ is *relatively more efficient* than $\hat{\theta}'^{(n)}$ if $Var(\hat{\theta}^{(n)}) > Var(\hat{\theta}'^{(n)})$.

- An estimator $\hat{\theta}^{(n)}$ is **simply consistent** if it stochastically converges to its parameter whatever the value of its parameter. More formally:

$$\forall\theta\ \forall\varepsilon\ \lim_{n\to\infty} \Pr(|\hat{\theta}_n - \theta| \leq \epsilon) = 1$$

  Observe that the estimator $\overline{X}$ is simply consistent as a consequence of the strong law of large numbers (in fact this law implies more: $\overline{X}$ converges almost surely to θ).

Confidence Interval. We now focus on the main information that should be computed simultaneously with the estimation, a *confidence interval* including the estimation. More precisely, given some $0 < \alpha < 1$, we look for two estimators $L^{(n)} < U^{(n)}$ such that:

$$\forall\theta\ \Pr(L^{(n)} \leq \theta \leq U^{(n)}) \geq 1 - \alpha$$

$[L^{(n)}, U^{(n)}]$ is called the *confidence interval* (CI). $1 - \alpha$ is called the *confidence level*. $U^{(n)} - L^{(n)}$ is the *width* of the CI and the desired goal is that whatever α and a fixed width, n can be determined to fulfill the above property with a width of interval not bigger than the required one.

There are three methods to obtain the estimators for the confidence interval of a parameter of a random variable whose parametrized distribution is known. Two of them are exact while the third one produces an approximate interval where the quality of approximation increases with the number of samples.

The *general method* is based on the following observation. Assume that we are given two functions $h^-(\theta)$ and $h^+(\theta)$ such that:

$$\forall\theta\ \Pr_\theta(h^-(\theta) \leq \hat{\theta}^{(n)} \leq h^+(\theta)) \geq \mathbf{1} - \alpha$$

Then θ belongs to the (random) set $\{\zeta \mid h^-(\zeta) \leq \hat{\theta}^{(n)} \leq h^+(\zeta)\}$ with a probability greater or equal than $1 - \alpha$. Let us illustrate this method with a simple case: θ is the mean value and $\hat{\theta}$, the estimator, consists of a single sampling. Assume that

the distribution $F(x|\theta)$ is decreasing with respect to θ (which often occurs when θ is the mean of the random variable) and that for every x, $\lim_{\theta\to-\infty} F(x|\theta) = 1$ and $\lim_{\theta\to\infty} F(x|\theta) = 0$. We also introduce the *strict* distribution $F^-(x|\theta) = \lim_{y\to x|y<x} F(x|\theta)$. Then choosing:

$$L = \sup(\zeta \mid F^-(\hat{\theta}|\zeta) \geq 1-\alpha) \text{ and } U = \inf(\zeta \mid F(\hat{\theta}|\zeta) \leq \alpha)$$

the reader can check that $[L, U]$ is a confidence interval with level $1-\alpha$ (non empty as soon as $\alpha < 1/2$).

The drawback of this method is its computational cost since it requires to "inverse" the distribution function w.r.t. θ. For instance, if this is the distribution of $\overline{X}$ and the parametrized distribution has no special features, this becomes quickly untractable when n the number of samples grows.

The *pivotal quantity method* is based on the existence of particular random variables called *pivotal quantity*. A pivotal quantity Q (w.r.t. a parameter θ) is a random variable function of $X_1, \ldots, X_n$ and θ whose distribution is independent of all parameters (including θ. Let us first illustrate this notion with the standard example of a normal distribution $N(\mu, \sigma^2)$ where μ is the mean and σ^2 is the variance. We use the estimators $\overline{X}$ and S^2 defined above. Then:

$$\frac{\overline{X}-\mu}{S/\sqrt{n}} \text{ and } \frac{(n-1)S^2}{\sigma^2}$$

are pivotal quantities related to μ and σ^2 whose distributions are respectively the Student distribution with $n-1$ degrees of freedom and the Chi-square distribution with $n-1$ degrees of freedom.

More generally, let a parametrized density function $f(x|\theta_1, \theta_2)$ defined by $f(x|\theta_1, \theta_2) = (1/\theta_2) f((x-\theta_1)/\theta_2)$. θ_1 is called the *location parameter* and θ_2 is called the *scale parameter*. Then the following variables a re pivotal quantities related to θ_1 and θ_2:

$$\frac{\hat{\theta}_1 - \theta_1}{\hat{\theta}_2} \text{ and } \frac{\hat{\theta}_2}{\theta_2}$$

when $\hat{\theta}_1$ and $\hat{\theta}_2$ are estimators obtained by the maximum likelihood method.

Once a pivotal quantity has been identified, it is routine to obtain the confidence interval when the distribution is a standard one by using some table of quantiles. Let us illustrate this point with the pivotal quantity related to the mean of a normal distribution. Observe that the Student distribution (say F) is symmetrical and let us denote $t = F^{-1}(1-\alpha/2)$. Then:

$$1-\alpha = \Pr(-t \leq \frac{\overline{X}-\mu}{S/\sqrt{n}} \leq t) = \Pr(\overline{X} - tS/\sqrt{n} \leq \mu \leq \overline{X} + tS/\sqrt{n})$$

Observe that all this probabilities are equal whatever μ and σ thus providing the required confidence interval.

The *approximate method* is based on the idea that for large number of samples given a distribution with μ its (parametrized) mean and σ^2 its (known) variance, there is an *asympotical* pivotal quantity for the estimator $\overline{X}$, More precisely, $\frac{\sqrt{n}}{\sigma}(\overline{X} - \mu)$ stochastically converges to the standard normal distribution $N(0, 1)$ and thus using the previous method will provide an approximate confidence interval.

Observe that we have considered that σ is known. We now develop the case of an unknown variance for the Bernoulli distribution $BIN(1, p)$ with the estimator $\overline{X}$. This case is very important since it is the one that is used by most of the statistical model checkers for evaluating a unknown probability p. Here the variance $p(1 - p)$ is unknown but depends on p. Furthermore by the law of large numbers $\overline{X}$ stochastically converges to p. Combining this with the central limit theorem we obtain that: $\frac{\sqrt{n}}{\overline{X}(1-\overline{X})}(\overline{X} - p)$ stochastically converges to the standard normal distribution $N(0, 1)$ leading to an approximate confidence interval. However the quality of this approximation is strongly related to $n \cdot \min(p, 1-p)$. Thus in case of rare event $p \ll 1$, one must be careful and chooses a huge n which may prohibit the use of statistical estimation.

Hypothesis Testing

Hypothesis testing consists to decide whether some hypothesis H_0 should be accepted or rejected in favour of an alternative hypothesis H_1. This problem is generally presented in an asymmetrical way. H_0 is the hypothesis that is currently adopted and without strong evidence, one does not want to reject it. So in the specification of the problem a (small) probability p is also introduced and one looks for a test that:

- has a probability α to reject H_0 when H_0 is true less or equal than p, called a type I error;
- minimizes the probability β to accept H_0 when H_1 is true, called a type II error; the pair (α, β) is called the *strength* of the test.

Hypotheses are often related to the value of some parameter, say μ, of a parametrized distribution. An hypothesis is said to be *simple* if it fixes the value of the parameter like $\mu = \mu_0$ where μ_0 is some constant or *composite* when it restricts the value of the parameter like $\mu > \mu_0$ or $\mu \neq \mu_0$. Observe that the error probabilities need a formal definition when they are conditioned by a composite hypothesis.

In the context of model checking in order to statistically "decide" whether some probability p is greater than some value v, one usually tests the (composite) hypothesis $p \leq v - \varepsilon$ against the (composite) hypothesis $p \geq v + \varepsilon$ where the interval $]p - \varepsilon, p - \varepsilon[$ represents an indifference region.

There are two ways to design such tests given some predefined strength: either one fixes *a priori* the number n of samplings that are required or after every sampling one decides to accept one of the hypotheses or to continue with a new sampling. We now develop these two techniques.

Fixed-Size Samples Test. Let us start with two simple hypotheses $H_0 : \theta = \theta_0$ and $H_1 : \theta = \theta_1$ with θ the parameter of a (known) distribution. Furthermore we assume that the distribution is given by a density function $f(x|\theta)$. Given a set of n samplings, we introduce the auxiliary function:

$$\lambda(x_1, \ldots, x_n) = \frac{\prod_{1 \leq i \leq n} f(x_i|\theta_0)}{\prod_{1 \leq i \leq n} f(x_i|\theta_1)}$$

The Neyman-Pearson test consists to accept hypothesis H_0 if $\lambda(X_1, \ldots, X_n) > k$ where k is chosen in such a way that $\Pr(\lambda(X_1, \ldots, X_n) \leq k|\theta_0) = \alpha$. This test is optimal in the following sense: any test that is based on n samples and such that the probability of a type I error is equal to α has a probability of a type II error greater or equal than the one of the Neyman-Pearson test. Thus the Neyman-Pearson test seems to be the most appropriate one w.r.t. the problem stated before.

Sequential Test. However the Neyman-Pearson test is only optimal among the tests with a fixed number of samples. The sequential test introduced by Wald [Wal45] consists to iterate a decision procedure after every step. More precisely at step n:

- If $\lambda(x_1, \ldots, x_n) \geq A$ then accept hypothesis H_0.
- If $\lambda(x_1, \ldots, x_n) \leq B$ then accept hypothesis H_1.
- Otherwise go the next iteration.

With $A > 1$ and $B < 1$, then whatever the hypothesis is true, this process terminates almost surely. Choosing $A = \frac{1-\beta}{\alpha}$ and $B = \frac{\beta}{1-\alpha}$, it can be shown that α' the probability of a type I error fulfills $\alpha' \leq \frac{\alpha}{1-\beta}$ and that β' the probability of a type II error fulfills $\beta' \leq \frac{\alpha}{1-\beta}$. Furthermore the inequality $\alpha' + \beta' \leq \alpha + \beta$ holds.

The comparison between the sequential test and the fixed sample test depends on the parametrized distribution. In [Wal45], the author establishes that,

in case of a normal distribution the expected number of samples is about half the fixed number of samples required for the same strength.

The case of discrete random variables is similarly handled by substituting to the density function the probabilities of the discrete distribution (for both the fixed sample test and the sequential test). At last, the composite test $\theta \leq \theta_0$ against $\theta \geq \theta_1$ can be managed by transforming these hypotheses by $\theta = \theta_0$ against $\theta = \theta_1$ if weak (and natural) conditions are satisfied by the parametrized distribution. This last point is particularly interesting for the statistical model checking.

3.3.2 Principles of Statistical Model Checking

In this standard version, statistical model-checking applies on a logic like PCTL or CSL and a Markov chain, discrete or continuous. Since one wants to solve decision problem with a statistical method, the specification of the problem must be carefully defined. Here we follow the approach of H. Younes [YS06].

First, one introduces a function δ which associates with every probability p, a precision $\delta(p)$. Then, rather than defining a single satisfaction relation (for state and path formulas), one defines two relations "strong satisfaction" and "strong falsification". These relations are defined inductively. We just give illustrating cases of these inductive definitions:

- Let $\phi \equiv \phi_1 \wedge \phi_2$ be a state formula. Then a state s strongly satisfies ϕ iff s strongly satisfies ϕ_1 and ϕ_2. Similarly s strongly falsifies ϕ iff s strongly falsifies ϕ_1 or ϕ_2. This case corresponds to a standard induction.

- Let $\phi \equiv \neg\phi_1$ be a state formula. Then a state s strongly satisfies ϕ iff s strongly falsifies ϕ_1. Similarly s strongly falsifies ϕ iff s strongly satisfies ϕ_1. This is the only case which requires simultaneous induction.

- Let $\phi \equiv \Pr_{\leq v}\varphi$ be a state formula. Then a state s strongly satisfies ϕ iff the probability p that a random path σ starting from s strongly satisfies φ fulfills $p < v - \delta(v)$. The state s strongly falsifies φ iff the probability p that a random path σ starting from s strongly falsifies φ fulfills $p > 1 - v + \delta(v)$. In this case, the function δ occurs.

- Let $\varphi \equiv \phi_1 \mathcal{U}^I \phi_2$ be a path formula. Then a path σ strongly satisfies φ iff there is a prefix $(s_0, \tau_0), \ldots, (s_k, \tau_k)$ of σ such that $\tau_k \in I$, s_k strongly satisfies ϕ_2 and for all $i < k$, s_i strongly satisfies ϕ_1. Similarly, a path $\sigma = (s_0, \tau_0), \ldots, (s_k, \tau_k), \ldots$ strongly falsifies φ iff for all $\tau_i \in I$, either s_i strongly falsifies ϕ_2 or there exists $j < i$ such that s_j strongly falsifies ϕ_1.

It is routine to check strong satisfaction and strong falsification are mutually exclusive. However, there may exist a state s and a state formula ϕ such that neither s strongly satisfies ϕ nor s strongly falsifies ϕ. Such a situation can be interpreted as a situation indifferent to the modeler or as a "don't know" value.

We are looking for a model checking algorithm that takes as input a Markov chain $\mathcal{M}$ with a distinguished state s, a state formula ϕ, two confidence levels α and β and returns a boolean answer say res such that:

- The probability that $res =$ **false** knowing that s strongly satisfies ϕ is less or equal than α.
- The probability that $res =$ **true** knowing that s strongly falsifies ϕ is less or equal than β.

The algorithm proposed in [YS06] is based on hypothesis testing and only works for the *bounded until* meaning that the interval I occurring in the operator $\mathcal{U}^I$ is finite. We present it for a formula without nested operators and then we explain how to handle the case of nested operators.

Let $\phi \equiv \Pr_{\geq v} p_1 \mathcal{U}^I p_2$ and p the probability that a random path starting from s satisfies ϕ. Given α and β, the algorithm determines the thresholds $A(\alpha, \beta)$ and $B(\alpha, \beta)$ used in the sequential hypothesis test for the hypothesis $p = v + \delta(v)$ against the hypothesis $p = v - \delta(v)$. Then it generates a random path up to a time equal to the upper bound of I. Almost surely, this trajectory has a finite number of events. Based on this finite prefix, the algorithm decides whether the path satisfies or falsifies ϕ (here it is true satisfaction or falsification). Then it applies the sequential testing and either decides or generates a new sampling.

When nested operators occur, for instance in $\phi \equiv \Pr_{\geq v} \phi_1 \mathcal{U}^I \phi_2$, the previous procedure must be adapted as follows.

- One fixes α' a bound on the accumulated probability that the procedure performs a type I error on the strong satisfaction of the subformulas of the formula and β' a bound on the accumulated probability that the procedure performs a type II error.
- Then one modifies the hypotheses related to the main formula substituting $v + \delta(v)$ by $(v + \delta(v))(1 - \alpha')$ and $v - \delta(v)$ by $1 - (1 - (v - \delta(v)))(1 - \beta')$. Observe that α' and β' must be chosen sufficiently small in order that: $(v + \delta(v))(1 - \alpha') > 1 - (1 - (v - \delta(v)))(1 - \beta')$.
- Once a finite prefix has been generated, the parameters of statistical evaluation of the subformulas for the states occurring on the prefix depend on the instant of their occurrence. For instance assume that the formula is

$\phi \equiv \Pr_{\geq v}\phi_1 \mathcal{U}^I \phi_2$ and the prefix is $(s_0, \tau_0), \ldots, (s_k, \tau_k)$ with $\tau_k \notin I$ (meaning that $\tau_{k+1} > \max(I)$). Thus to be satisfied for all $i < k$, s_i must satisfy ϕ_1 and s_k must satisfy ϕ_2. By a probabilistic reasoning, the statistical satisfaction of the subformulas must be done with parameters $(\frac{\alpha'}{n}, \beta')$.

3.3.3 State of the Art in Statistical Model Checking

Statistical model checking presents several advantages. No Markovian property of the system is required. Furthermore the model may have an infinite state space. Design of the algorithms are relatively simple since they are based on discrete-event simulation. In principle, the logics associated with statistical model-checking could be more expressive than the ones associated with numerical model-checking even if the current tools do not include such powerful logic. At last, the algorithms are easily parallelizable.

We now list some limits of the statistical model checking and current approaches to overcome these limits.

The statistical approach only works for purely probabilistic systems, i.e., those that do not have any non determinism. For instance, to the best of our knowledge, no statistical model checking has been proposed for Markovian decision processes.

As mentioned before, the *until* requires a finite interval in order to ensure that almost surely the generation of random path will terminate. The case of the *unbounded until*, i.e. $\Pr_{\bowtie v}\phi_1 \mathcal{U} \phi_2$, has been tackled by different researchers. In [SVA05a], the authors introduce a random coin with small probability of success and before each step of the simulation the coin is flipped and the simulation stopped on success. This decreases the probability of satisfaction and must be taken into account in the confidence interval. Furthermore, this algorithm requires a specialized procedure to check $\Pr_{=0}\phi_1 \mathcal{U} \phi_2$ for every state that is encountered during the simulation. Unfortunately, the procedure proposed for this special case requires the knowledge of a bound of the number of states and in general leads to generate paths with expected length of the same order as the size of the state space. Summarizing this method seems to be untractable on real case studies. A similar method is proposed in [HJB+10] whose improvement w.r.t. the former one, is based on the observation that the satisfaction of the formula $\Pr_{=0}\phi_1 \mathcal{U} \phi_2$ is independent of the exact values of the non null transition probabilities. Thus the authors propose to substitute to the chain, a modified chain with equal probabilities of transitions.

The logic CSL contains the steady-state operator which reasons about the steady-state probabilities. It is well-known that Monte-Carlo simulations only provide an approximate sampling of the steady-state distribution (by sampling

a transient distribution at some large instant). However perfect simulations based on the Propp-Wilson algorithm [PW96] provide an exact sampling of the steady-state distribution. In [EP09], the authors apply this technique to steady-state operator. Furthermore, they provide an alternative algorithm for the unbounded until that stops when "reaching the steady-state" (see the Propp-Wilson algorithm for the meaning of this expression) but they did not exhibit the confidence level of this new algorithm.

There have been several application fields of statistical model checking. This technique has been applied to verify properties of mixed-signal circuits including for instance statements about the Fourier transform of the signal which seems impossible with numerical model checking [CDL10]. Biological applications have been also intensively studied with the help of statistical model checking (see [BFMP09] for instance). This is mainly due to the fact that biological systems are often non Markovian and that the relevant properties are related to properties of the random path that cannot expressed with logics corresponding to numerical model checking. In [BBB+10], statistical model checking techniques are applied to the verification of heterogeneous applications communicating over a shared network.

3.4 Conclusion

In this chapter, we have introduced quantitative verification of stochastic models. The methods are partitioned into two families: numerical and statistical ones.

The first family evaluates the desired probability with a high precision. They are efficient for both transient and steady-state properties. But they are limited to Markovian or semi-Markovian models and they cannot deal with huge or infinite state space. The techniques associated with numerical verification are closely related to standard numerical analysis of Markov chains.

The second family provides an interval containing the correct answer with a high probability. Such methods can analyze any model with an operational stochastic semantic except for nested formulas which can only be checked on Markov chains. They use statistical techniques: discrete-event simulation, statistical estimation and/or statistical hypothesis test. They are well adapted to transient properties. But their application in two important settings needs some additional analysis which depends on the model: steady state properties and properties with very weak probability (i.e. rare event).

Part II

Numerical Methods

Chapter 4

Stochastic Bounds for Censored Markov Chains

This chapter is related to the publication [BDF10].

4.1 Introduction

Since Plateau's seminal work on composition and compact tensor representation of Markov chains using Stochastic Automata Networks (SAN), we know how to model Markov systems with interacting components and large state space [Pla85, FP88, PF91]. The main idea of the SAN approach is to decompose the system of interest into its components and to model each component separately. Once this is done, interactions and dependencies among components can be added to complete the model and obtain the transition probability (in discrete time) or transition rate matrix (in continuous time). The basic operations needed to build these matrices are Kronecker sum and product (sometimes denoted as tensor) and they are applied on local descriptions of the components and the actions. The benefit of the tensor approach is twofold. First, each component can be modeled much easier compared to the global system. Second, the space required to store the description of components is in general much smaller than the explicit list of transitions, even in a sparse representation. However, using this representation instead of the usual sparse matrix form increases slightly the time required for numerical analysis of the chain [FPS98, UD98, SAP95]. The decomposition and tensor representation has been generalized to other modeling formalisms as well: Stochastic Petri nets [Don93], Stochastic Process Algebra [HK01]. So we now have several well-founded methods to model complex systems using Markov chains with huge state space. However, the chains are so big that it is sometimes impossible to store in memory a probability vector in-

dexed by the state space. Consider for instance the model of the highly available multicomponent systems studied by Muntz [MdSeSG89]. A typical system consists of several disks, CPUs and controllers with different types of failures. The system is operational if there are enough CPU, disks and controllers available. As the number of states grows exponentially with the number of different components, the state space is huge and the up states are relatively rare. A typical system has more than 9×10^{10} states and 10^{12} transitions. It is even not possible to store a probability vector of the size of the whole state space. Thus we have to truncate the state space to analyze a smaller chain (see for instance [dSeSO92]). Few results have been published on this topic despite its considerable practical importance. A notable exception are the results obtained by Seneta [Sen06, chapter 7] on the approximation of an infinite Markov chain by a sequence of finite truncations. Here we investigate how one can study the truncation of a finite chain using the theory of Censored Markov Chains (CMC).

In section 4.2 we give some preliminaries about the censored Markov chains. In Section 4.3 we introduce an alternative decomposition of stochastic complement S_A that turns out to be natural for deriving stochastic bounds. Section 4.4 contains some definitions and basic results on stochastic orders that will be used later in the chapter. We also give an overview of existing results for censored Markov chains and we describe algorithm DPY. In Section 4.5 we prove optimality of DPY using our new decomposition from Section 4.3. Then in Section 4.6 we discuss how to use additional information on non-observed states to improve DPY bounds. Section 4.7 contains conclusions and final remarks.

4.2 Censored Markov Chains and State Space Truncation

Let $\{X_t\}_{t\geq 0}$ be a Discrete Time Markov Chain (DTMC) with a state space $\mathcal{X}$. Let $\mathcal{E} \subset \mathcal{X}$ be the set of observed states, $\mathcal{X} = \mathcal{E} \cup \mathcal{E}^c$, $\mathcal{E} \cap \mathcal{E}^c = \emptyset$. For simplicity of presentation, we assume here that the state space is finite and denote by $n = |\mathcal{E}|$ and $m = |\mathcal{E}^c|$. Assume that successive visits of $\{X_t\}_{t\geq 0}$ to $\mathcal{E}$ take place at time epochs $0 \leq t_0 < t_1 < ...$ Then the chain $\left\{X^{\mathcal{E}}\right\}_{k\geq 0} = \{X_{t_k}\}_{k\geq 0}$ is called the censored chain with censoring set $\mathcal{E}$ [ZL96]. Let P denote the transition probability matrix of chain $\{X_t\}_{t\geq 0}$. Consider the partition of the state space to obtain a block description of P:

$$P = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \tag{4.1}$$

Blocks A, B, C and D contain respectively transitions from $\mathcal{E}$ to $\mathcal{E}$, from $\mathcal{E}$ to $\mathcal{E}^c$, from $\mathcal{E}^c$ to $\mathcal{E}$ and from $\mathcal{E}^c$ to $\mathcal{E}^c$. The censored chain only observes the states in $\mathcal{E}$. It is out of the scope of this work to study how one can find a good set $\mathcal{E}$ to get the most accurate censoring process. In some special cases this can be done by analyzing the drift of a well chosen Lyapunov function (see [MT93], [Mey08, Chapter 8] for Lyapunov functions and stochastic stability, and [DHSW10] for an application to Markov population models). We study here how we can find bounds of the chain once set $\mathcal{E}$ is found.

The tensor representation of the chain can be used to generate the blocks or some elements of these blocks. Indeed, one can derive easily the set of successors or predecessors for a node taking into account the tensor representation. Such a property is the key idea for algorithm SAN2LIMSUB [FPSS06] which is based on the generation of a column of the stochastic matrix (i.e. the predecessor function) based on the tensor representation and allows the computation of a lumpable stochastic bound. Software tools dealing with Markov chains also support in general a successor function, provided by the user, which is used to build a row of the stochastic matrix. In some cases, a predecessor function is also supported by the tool.

We assume that $\mathcal{E}$ does not contain any reducible classes (so that the matrix $Id - D$ is regular). Then the transition probability matrix of the censored chain, often also called the stochastic complement of matrix A , is equal to [ZL96]:

$$S_A = A + B\left(Id - D\right)^{-1} C = A + B\left(\sum_{i=0}^{\infty} D^i\right) C \tag{4.2}$$

The second term of the right-hand side represents the probabilities of paths that return to set $\mathcal{E}$ through states in $\mathcal{E}^c$.

In many problems, the size of the initial probability matrix P makes building of the four blocks extremely time and space consuming, or even impossible. In some cases, we are able to obtain block D but it is too difficult to compute $(Id - D)^{-1}$ to finally get S_A. Deriving bounds for S_A from block A of matrix P and from some information on the other blocks is thus an interesting alternative approach and several algorithms have been proposed in the literature. Truffet [Tru97] considered the case when only block A is known. In that case, the stochastic bound is obtained by assuming that all the unknown returning transitions go to the last state of $\mathcal{E}$ (i.e. state n)

Dayar et al. [DPY06] proposed an algorithm, called DPY, for the case when blocks A and C are known. We prove here that their algorithm is optimal when we do not have any information on blocks B (transitions between observed and non-observed states) and D (transitions between the non-observed states). We consider further how to improve bounds when some additional information is

known on blocks B or D . We deal here with upper bounds but lower bounds may be computed as well. Once an upper bounding matrix is found, bounds on rewards, on steady-state and transient distributions, and on time to first visits may be derived as well (see [BF08] for some examples).

4.3 Decomposition of Stochastic Complement

Let us first fix the notation used throughout this chapter. For any $x \in \mathbb{R}$, $[x]^+ = max\{x, 0\}$. The row (resp. column) vectors are denoted by small latin (resp. greek) letters and $\mathbf{0}$ (resp. $\mathbf{1}$) denotes the vector with all components equal to 0 (resp. 1). For a vector v, v^t, denotes the transposed vector, and $diag(v)$ is the matrix whose diagonal elements are given by vector, v i.e.

$$diag(v)[i,j] = \begin{cases} v[i], & i = j \\ 0, & i \neq j \end{cases}$$

Furthermore, $\leq$ denotes the elementwise comparison of two vectors (or matrices), and $M[i,.]$ is row i of matrix M. We use the term positive vector (matrix) for a vector (matrix) whose all elements are non-negative. For any positive vector v we will denote by v^* the vector obtained from v by replacing all the zero elements by 1:

$$v^*[i] = \begin{cases} v[i], & v[i] > 0 \\ 1, & v[i] = 0 \end{cases}$$

For any positive matrix M, matrix $diag(M\mathbf{1}^t)$ is a diagonal matrix whose diagonal elements are equal to the sum of rows of matrix M. If matrix M does not contain any zero row, matrix $diag(M\mathbf{1}^t)$ is regular and $(diag\,(M\mathbf{1}^t))^{-1} M$ is the renormalized matrix of M (i.e. a matrix such that $\sum_j M[i,j] = 1$ for all i). If matrix M contains a zero row, then $diag\,(M(1)^t)$ is singular so we use instead a modified matrix $diag\left((M\mathbf{1}^t)^*\right)^{-1} M$ that is always regular. For any positive matrix M, we denote by $n(M)$ the matrix:

$$n(M) = diag\left(\left(M\mathbf{1}^t\right)^*\right)^{-1} M$$

Clearly, for any positive matrix M, matrix $n(M)$ is a positive matrix whose all rows are either stochastic or zero rows. We will use the following technical observation in our decomposition of stochastic complement:

Lemma 4.1. *For any positive matrix M:*

$$\left(diag\left(\left(M\mathbf{1}^t\right)^*\right) - diag\left(M\mathbf{1}^t\right)\right) n\,(M) = 0$$

Proof. Denote by $Z = diag\left((M\mathbf{1}^t)^*\right) - diag\left(M\mathbf{1}^t\right)$. Matrix Z is a diagonal matrix with:

$$Z[i,i] = \begin{cases} 1, & M[i,.] = \mathbf{0} \\ 0, & \text{otherwise} \end{cases}$$

The rows of matrix Z and the columns of M are orthogonal by the definition of matrix Z, so we have $ZM = \mathbf{0}$. This remains true for the renormalized matrix, thus $Zn\left(M\right) = \mathbf{0}$

□

Let us now consider the block decomposition of matrix P given by 4.1. Denote by $W = diag\left(C\mathbf{1}^t\right)$. Then clearly, for all $1 \leq i \leq m$:

$$W[i,i] = \sum_{j=1}^{n} C\left[i,j\right] = 1 - \sum_{k=1}^{m} D\left[i,k\right]$$

If there is a state $i \in \mathcal{E}^c$ without any outgoing transition to set $\mathcal{E}$, then row i of W is equal to $\mathbf{0}$ and matrix W is singular. Finally, denote by $W^* = diag\left((C\mathbf{1}^t)^*\right)$

The following proposition gives an alternative decomposition of stochastic complement. To the best of our knowledge such a representation was not previously stated even if it appears quite simple. Using this new representation we can derive new arguments to prove stochastic bounds based on comparison of stochastic vectors. Such an approach was harder with the usual representation in 4.2 as $\left(Id - D\right)^{-1}$ is a matrix of expectations.

Proposition 4.1 (Decomposition of stochastic complement).

1. *Matrix $\left(W^*\right)^{-1} C$ has rows that are either stochastic or equal to* $\mathbf{0}$
2. *Matrix $\left(Id - D\right)^{-1} W$ is stochastic.*
3. *Matrix S_A can be decomposed as:*

$$S_A = A + B\left(Id - D\right)^{-1} W \left(W^*\right)^{-1} C \tag{4.3}$$

Proof. Matrix W is always regular, so $\left(W^*\right)^{-1}$ is well defined.

1. We have $\left(W^*\right)^{-1} C = n\left(C\right)$ so the first statement is obvious.

2. We know that row of matrix $(Id - D)^{-1} C$ is equal to the conditional probability vector of entering the set $\mathcal{E}$, knowing that we initially start in $i \in \mathcal{E}^c$. Let $G = (Id - D)^{-1}$. Therefore, for all i, $\sum_k (GC)[i,k] = 1$ and:

$$\sum_j (GW)[i,j] = \sum_j G[i,j] \sum_k C[j,k] = \sum_k \sum_j G[i,j] C[j,k]$$

$$= \sum_k (GC)[i,k] = 1$$

Thus matrix $(Id - D)^{-1} W$ is stochastic.

3. Matrix S_A can be decomposed as:

$$SA = A + B (Id - D)^{-1} W^* (W^*)^{-1} C \tag{4.4}$$

We have:

$$W = diag\left(C\mathbf{1}^t\right), \quad W^* = diag\left(\left(C\mathbf{1}^t\right)^*\right), (W^*)^{-1} C = n(C)$$

Thus by Lemma 4.1:

$$W^* (W^*)^{-1} C = W (W^*)^{-1} C \tag{4.5}$$

Relations 4.5 and 4.4 imply 4.3.

□

Example 1 Consider the following block decomposition of matrix P into four blocks A, B, C and D given by:

$$A = \begin{bmatrix} 0.4 & 0.1 & 0.2 & 0.0 \\ 0.2 & 0.1 & 0.3 & 0.1 \\ 0.2 & 0.1 & 0.0 & 0.3 \\ 0.0 & 0.0 & 0.4 & 0.1 \end{bmatrix}, \quad B = \begin{bmatrix} 0.1 & 0.2 & 0.0 & 0.0 \\ 0.3 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.4 & 0.0 & 0.0 \\ 0.3 & 0.2 & 0.0 & 0.0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0.2 & 0.1 & 0.1 & 0.0 \\ 0.1 & 0.3 & 0.0 & 0.0 \\ 0.4 & 0.1 & 0.2 & 0.1 \\ 0.0 & 0.1 & 0.0 & 0.0 \end{bmatrix}, \quad D = \begin{bmatrix} 0.1 & 0.2 & 0.1 & 0.2 \\ 0.3 & 0.0 & 0.3 & 0.0 \\ 0.1 & 0.0 & 0.0 & 0.1 \\ 0.3 & 0.3 & 0.3 & 0.0 \end{bmatrix}$$

We have:

$$(Id - D)^{-1} = \begin{bmatrix} 1.374 & 0.368 & 0.340 & 0.309 \\ 0.472 & 1.136 & 0.429 & 0.137 \\ 0.198 & 0.084 & 1.090 & 0.149 \\ 0614 & 0.476 & 0.558 & 1.179 \end{bmatrix}$$

The usual description of the censored chain gives:

$$S_A = A + B \times \begin{bmatrix} 1.374 & 0.368 & 0.340 & 0.309 \\ 0.472 & 1.136 & 0.429 & 0.137 \\ 0.198 & 0.084 & 1.090 & 0.149 \\ 0614 & 0.476 & 0.558 & 1.179 \end{bmatrix} \times \begin{bmatrix} 0.2 & 0.1 & 0.1 & 0.0 \\ 0.1 & 0.3 & 0.0 & 0.0 \\ 0.4 & 0.1 & 0.2 & 0.1 \\ 0.0 & 0.1 & 0.0 & 0.0 \end{bmatrix}$$

while its new representation proved by Proposition 4.1 is:

$$S_A = A + B \times \begin{bmatrix} 0.550 & 0.147 & 0.272 & 0.031 \\ 0.189 & 0.454 & 0.343 & 0.014 \\ 0.079 & 0.034 & 0.872 & 0.015 \\ 0.245 & 0.191 & 0.446 & 0.118 \end{bmatrix} \times \begin{bmatrix} 0.500 & 0.250 & 0.250 & 0.000 \\ 0.250 & 0.750 & 0.000 & 0.000 \\ 0.500 & 0.125 & 0.250 & 0.125 \\ 0.000 & 1.000 & 0.000 & 0.000 \end{bmatrix}$$

Clearly the last two matrices are stochastic. As we now deal with stochastic matrices, we are able in the following to compute their stochastic bounds. It is possible that this new representation of censored Markov chains may have other applications as well. However they are out of the scope of this work.

4.4 Stochastic Bounds for Censored Markov Chains

We recall first the definition of strong stochastic ordering of random variables on a finite state space $\{1, ..., n\}$ (see [MS02] for more details on stochastic orders).

4.4.1 Some Fundamental Results on Stochastic Bounds

We will define operators r and v as in [DFPV06] and s for any positive $m \times n$ matrix M:

$$\forall i, j,\ r(M)[i,j] = \sum_{k=j}^{n} M[i,k] \tag{4.6}$$

$$\forall i, j,\ v(M)[i,j] = \max_{k \le i} \{r(M)[k,j]\} \tag{4.7}$$

$$\forall j,\ s(M)[j] = \max_{i} \{r(M)[i,j]\} \tag{4.8}$$

Let X and Y be two random variables with probability vectors p and q ($p[k] = P(X = k)$, $q[k] = P(Y = k)$ for all k).

Definition 4.1. *$X \preccurlyeq_{st} Y$ if $\forall j,\ \sum_{k=j}^{n} p[k] \leq \sum_{k=j}^{n} q[k]$ (i.e. $r(p) \leq r(q)$)*

Example 2 *Let $p = (0.1, 0.2, 0.5, 0.2)$ and $q = (0, 0.3, 0.4, 0.3)$, we have $X \preccurlyeq_{st} Y$ as*

$$\begin{gathered} 0.2 \leq 0.3 \\ 0.5 + 0.2 \leq 0.4 + 0.3 \\ 0.2 + 0.5 + 0.2 \leq 0.3 + 0.4 + 0.3 \\ 0.1 + 0.2 + 0.5 + 0.2 \leq 0.0 + 0.3 + 0.4 + 0.3 \end{gathered}$$

Let $\{X_t\}_{t\geq 0}$ and $\{Y_t\}_{t\geq 0}$ be two DTMC with transition probability matrices P and Q. We say that $\{X_t\}_{t\geq 0} \preccurlyeq_{st} \{Y_t\}_{t\geq 0}$ if $X_t \preccurlyeq_{st} Y_t$ for all $t \geq 0$.

Definition 4.2. *For two probability matrices P and Q, $P \preccurlyeq_{st} Q$ if $r(P) \leq r(Q)$*

Definition 4.3. *A probability matrix P is $\preccurlyeq_{st}$-monotone if for any two probability vectors p and q:*

$$p \preccurlyeq_{st} q \Rightarrow pP \preccurlyeq_{st} qP$$

We will use the following characterization of monotonicity (see [MS02] for the proof):

Proposition 4.2. *A probability matrix P is $\preccurlyeq_{st}$-monotone iff:*

$$\forall i > 1,\ P[i-1, .] \preccurlyeq_{st} P[i, .] \tag{4.9}$$

i.e. iff $v(P) = r(P)$

Sufficient conditions for comparison of two DTMC based on stochastic comparison and monotonicity can be found in [MS02]. These conditions can be easily checked algorithmically and it is also possible to construct a monotone upper bound for an arbitrary stochastic matrix P (see [DFPV06] for more details and proofs):

Proposition 4.3 (Vincent's algorithm [DFPV06])**.** *Let P be any stochastic matrix. Then the Vincent's bound is given by $Q = r^{-1}v(P)$, where r^{-1} denotes the inverse of r. Then Q is $\preccurlyeq_{st}$ -monotone and $P \preccurlyeq_{st} Q$, therefore Q is a transition probability matrix of an upper bounding DTMC.*
Furthermore, if $P_1 \preccurlyeq_{st} P_2$, then $r^{-1}v(P_1) \preccurlyeq_{st} r^{-1}v(P_2)$

Corollary 4.1 (Optimality [DFPV06])**.** *Let P be any stochastic matrix and $Q = r^{-1}v(P)$. Then Q is the smallest $\preccurlyeq_{st}$-monotone upper bound for P, i.e. if R is any stochastic matrix such that R is $\preccurlyeq_{st}$-monotone and $P \preccurlyeq_{st} R$, then $Q \preccurlyeq_{st} R$.*

Example 3 Consider matrix:

$$G = \begin{bmatrix} 0.500 & 0.250 & 0.250 & 0.000 \\ 0.250 & 0.750 & 0.000 & 0.000 \\ 0.500 & 0.125 & 0.250 & 0.125 \\ 0.000 & 1.000 & 0.000 & 0.000 \end{bmatrix}$$

Vincent's algorithm applied on matrix G provides a monotone upper bounding matrix of G equal to:

$$r^{-1}v\left(G\right) = \begin{bmatrix} 0.500 & 0.250 & 0.250 & 0.000 \\ 0.250 & 0.500 & 0.250 & 0.000 \\ 0.250 & 0.375 & 0.250 & 0.125 \\ 0.000 & 0.625 & 0.250 & 0.125 \end{bmatrix}$$

4.4.2 Comparison of Positive Matrices

We can extend Definition 4.2 to positive (not necessarily square) matrices:

Definition 4.4. *Let M_1 and M_2 be any two positive matrices of the same size. We will say that $M_1 \preccurlyeq_{st} M_2$ if $r\left(M_1\right) \preccurlyeq_{st} r\left(M_2\right)$*

We now state two simple properties that will be very useful later in Section 4.6.

Lemma 4.2. *Let M_1 and M_2 be two positive matrices such that $M_1 \preccurlyeq_{st} M_2$ and Z any positive matrix. Then,*

$$ZM_1 \preccurlyeq_{st} ZM_2$$

Proof. We have

$$r\left(ZM_1\right) = Zr\left(M_1\right) \leq Zr\left(M_2\right) = r\left(ZM_2\right)$$

Thus $r\left(M_1\right) \leq r\left(M_2\right)$ and the fact that Z is a positive matrix imply that $r\left(ZM_1\right) \leq r\left(ZM_2\right)$, i.e. $ZM_1 \preccurlyeq_{st} ZM_2$. □

Lemma 4.3. *Let Z be any positive matrix and M a positive matrix whose rows are either stochastic or equal to $\mathbf{0}$. Then:*

$$ZM \preccurlyeq_{st} \alpha r^{-1}\left(s\left(M\right)\right)$$

where $\alpha = ZM\mathbf{1}^t$.

Proof. By definition of operators r and s, for any positive matrix M we have:

$$M[i,.] \preccurlyeq_{st} r^{-1}(s(M)) \, \forall i$$

Thus, $M \preccurlyeq_{st} \mathbf{1}^t r^{-1}(s(M))$. However, this can be improved by taking into account the zero rows of matrix M. Note that:

$$\left(M\mathbf{1}^t\right)[i] = \begin{cases} 0, & M[i,.] = 0 \\ 1, & M[i,.] \neq 0 \end{cases}$$

Thus, we have:

$$M \preccurlyeq_{st} M\mathbf{1}^t r^{-1}(s(M))$$

Now by Lemma 4.2 it follows that

$$ZM \preccurlyeq_{st} ZM\mathbf{1}^t r^{-1}(s(M)) = \alpha r^{-1}(s(M))$$

□

4.4.3 Stochastic Bounds for CMC

Now we can formally state the problems we consider:

1. Given only block A, compute a matrix Q such that $S_A \preccurlyeq_{st} Q$. Is there an optimal bound (in the sense of Definition 4.5), knowing only block A ?

2. Given blocks A and C, compute a matrix Q such that $S_A \preccurlyeq_{st} Q$. Is this bound better than the one obtained knowing only block A? Is there an optimal bound knowing only blocks A and C?

3. Can some additional information on blocks B and D improve stochastic bounds for CMC?

The first question was already answered by Truffet [Tru97]. Denote by $\beta = \mathbf{1}^t - A\mathbf{1}^t$ the column vector of probability slack for matrix A. Then the bound in Truffet [Tru97] is given by:

$$T(A) = A + \beta(0, ..., 0, 1) \tag{4.10}$$

It is straightforward to see that $S_A \preccurlyeq_{st} T(A)$. Furthermore, this is the best bound one can obtain knowing only block A. More formally:

Definition 4.5. *Let M be a family of stochastic matrices. A stochastic matrix Q is an $\preccurlyeq_{st}$-upper bound for family M if:*

$$\forall P \in \mathcal{M},\ P \preccurlyeq_{st} Q$$

An $\preccurlyeq_{st}$-upper bound Q of M is optimal if:

$$Q \preccurlyeq_{st} R, \text{for any } \preccurlyeq_{st}\text{-upper bound } R \text{ of } M$$

Let $\mathcal{R}$ be the set of all stochastic matrices such that $\mathcal{E}^c$ does not contain any reducible class (so that for any matrix $Z \in \mathcal{R}$, the stochastic complement S_Z is well defined by 4.2). Then the Truffet's bound $T(A)$ in 4.10 is the optimal $\preccurlyeq_{st}$-upper bound for family:

$$\mathcal{M}(A) = \{S_Z \ : \ Z \in \mathcal{R},\ Z_{\mathcal{E},\mathcal{E}} = A\}$$

The proof is straightforward.

The second question was partially answered by Dayar et al. [DPY06]: they derived an algorithm DPY that computes a stochastic bound for S_A when blocks A and C are known. In Algorithm 4.1 we give the algorithm DPY in its original form in [DPY06]. We show in Section 4.5 that DPY bound is optimal, which then fully answers the second question. The third question will be discussed in Section 4.6.

Algorithm 4.1: $DPY(A, C)$ [DPY06]

Data: Blocks A and C
Result: Matrix Q such that $S_A \preccurlyeq_{st} Q$
begin
  $\beta = \mathbf{1}^t - A\mathbf{1}^t$
  for $j = n$ ***downto*** 1 **do**
    $H[j] = \max_{k \in \mathcal{E}^c} \left(\frac{\sum_{l=j}^{n} C[k,l]}{\sum_{l=1}^{n} C[k,l]} \right)$
    for $i = 1$ ***to*** n **do**
      $F[i,j] = \left(\beta[i] H[j] - \sum_{l=j+1}^{n} F[i,l] \right)^+$
  $Q = A + F$
  return Q

Using operators r and s as defined in relations 4.6 and 4.8, DPY can be rewritten as:

1. $H = s(n(C))$

2. $F = \beta r^{-1}(H)$
3. $Q = A + F$

Thus:

$$DPY(A, C) = A + \beta r^{-1}(s(n(C))) \tag{4.11}$$

From the above equation it obviously follows that $DPY(A, C) \preccurlyeq_{st} T(A)$. The following example shows that DPY can improve the Truffet's bounds.

Example 4 Let

$$A = \begin{bmatrix} 0.2 & 0.1 & 0.3 & 0.0 \\ 0.1 & 0.0 & 0.3 & 0.0 \\ 0.0 & 0.5 & 0.0 & 0.2 \\ 0.1 & 0.0 & 0.3 & 0.2 \end{bmatrix}, \quad B = \begin{bmatrix} 0.1 & 0.1 & 0.2 & 0.0 & 0.0 & 0.0 \\ 0.2 & 0.2 & 0.2 & 0.0 & 0.0 & 0.0 \\ 0.1 & 0.2 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.4 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0.2 & 0.1 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.2 & 0.2 & 0.0 \\ 0.2 & 0.2 & 0.1 & 0.0 \\ 0.2 & 0.1 & 0.0 & 0.0 \end{bmatrix}, \quad D = \begin{bmatrix} 0.5 & 0.2 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.2 & 0.6 & 0.2 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.2 & 0.4 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.3 & 0.0 & 0.2 & 0.0 \\ 0.0 & 0.0 & 0.1 & 0.0 & 0.0 & 0.4 \\ 0.1 & 0.0 & 0.5 & 0.0 & 0.0 & 0.1 \end{bmatrix}$$

Then the stochastic complement of A is equal to:

$$S_A = \begin{bmatrix} 0.3475 & 0.2257 & 0.3827 & 0.0440 \\ 0.3446 & 0.1900 & 0.4079 & 0.0573 \\ 0.1378 & 0.5960 & 0.0431 & 0.2229 \\ 0.2009 & 0.1227 & 0.4151 & 0.2612 \end{bmatrix}$$

The Truffet's bound $T(A)$ and the DPY bound $DPY(A, C)$ are respectively:

$$T(A) = \begin{bmatrix} 0.2 & 0.1 & 0.3 & 0.4 \\ 0.1 & 0.0 & 0.3 & 0.6 \\ 0.0 & 0.5 & 0.0 & 0.5 \\ 0.1 & 0.0 & 0.3 & 0.6 \end{bmatrix}, \quad DPY(A, C) = \begin{bmatrix} 0.28 & 0.22 & 0.4 & 0.1 \\ 0.22 & 0.18 & 0.45 & 0.15 \\ 0.06 & 0.59 & 0.075 & 0.275 \\ 0.18 & 0.12 & 0.4 & 0.3 \end{bmatrix}$$

Computing blocks A and C is based on the application of the predecessor function (i.e. the computation of a column of the stochastic matrix) for all the states in $\mathcal{E}$. Note that block D and B in Example 4 have typical properties of blocks generated by breadth first search visit algorithm based on the successor functions. Indeed, block D is block upper Hessenberg and the rightmost entries of B are 0.

Let us turn back to the algorithm, its theoretical properties and some numerical results. First, DPY Algorithm is even exact in some cases which have proved by Dayar and his co- authors in [DPY06].

Proposition 4.4. *If block C has rank 1, then the bound provided by DPY is the true solution for the stochastic complement (see [DPY06] for a proof).*

Numerical experiments of bounds computation with DPY and the algorithm proposed in [BF08] (designed for the case when block C is not completely known) when blocks A and C are known have shown that DPY always provide the most accurate bounds. Numerical evidence suggests that DPY is optimal in that case and we prove this property in the next section.

4.5 Optimality of DPY

The Markov chain we consider may be ergodic or not. Indeed, we are interested in computing transient and steady-state results but also absorption probabilities and bounds on the first passage times. Therefore we must consider the cases where the chains are irreducible or not. Proving optimality of DPY must take into account the set of matrices involved and we derive two main results which differs on the ergodicity of the matrices.

We will first show here the optimality of DPY for family

$$\mathcal{M}(A,C) = \{S_Z \ : \ Z \in \mathcal{R}, \ Z_{\mathcal{E},\mathcal{E}} = A, \ Z_{\mathcal{E}^c,\mathcal{E}} = C\}$$

of all transition probability matrices in R with given blocks A and C. The chains may be not ergodic.

Theorem 4.1 (Optimality of DPY)**.** *Matrix $DPY(A,C)$ is the optimal $\preccurlyeq_{st}$-upper bound for family $\mathcal{M}(A,C)$.*

Proof. The proof that $DPY(A,C)$ is an $\preccurlyeq_{st}$-upper bound for family $\mathcal{M}(A,C)$ was already given in [DPY06]. We prove here that $DPY(A,C)$ is the optimal bound for $\mathcal{M}(A,C)$. Consider any non-zero row j of matrix C (i.e. such that $s_j < 1$) and denote by B_j the matrix such that:

$$B_j[i,k] = \begin{cases} 0, & \text{if } k \neq j \\ \beta[i] & \text{if } k = j \end{cases}$$

Let D_j be a matrix such that:

$$D_j[i,k] = \begin{cases} 0, & \text{if } k \neq j \\ 1 - \sum_{l=1}^{n} C[i,l] & \text{if } k = j \end{cases}$$

Let Z_j be the matrix composed of blocks A, B_j, C and D_j. Then clearly $Z_j \in \mathcal{M}(A,C)$. For Z_j all the returning paths from $\mathcal{E}^c$ to $\mathcal{E}$ go by state $j \in \mathcal{E}^c$. Denote

$\tilde{C} = n(C)$ to ease the notation. Any $\preccurlyeq_{st}$-upper bound R for family $\mathcal{M}(A, C)$ satisfies in particular:

$$S_{Z_j} = A + \beta\tilde{C}[j, .] \preccurlyeq_{st} R$$

i.e. $r\left(A + \beta\tilde{C}[j, .]\right) \leq r(R)$. This is valid for all j such that $\tilde{C}[j, .] \neq 0$. Thus:

$$\max_j r\left(A + \beta\tilde{C}[j, .]\right) = \max_{j:\tilde{C}[j,.]\neq 0} r\left(A + \beta\tilde{C}[j, .]\right) \leq r(R)$$

And

$$\begin{aligned} \max_j r\left(A + \beta\tilde{C}[j, .]\right) &= r(A) + \beta s\left(\tilde{C}\right) \\ &= r(A) + \beta H = r(A + F) \end{aligned}$$

so $DPY(A, C) = A + F \preccurlyeq_{st} R$

□

Similarly, let $\mathcal{M}^e(A, C)$ be the family of all ergodic matrices in $\mathcal{M}(A, C)$.

Theorem 4.2 (Optimality of DPY for the ergodic matrices)**.** *Matrix $DPY(A, C)$ is the optimal $\preccurlyeq_{st}$-upper bound for family $\mathcal{M}^e(A, C)$.*

Proof. The main step of the proof is to show that family $\mathcal{M}^e(A, C)$ is dense within $\mathcal{M}(A, C)$, i.e. that for any $U \in \mathcal{M}(A, C)$ and for any $\epsilon > 0$ there exists $V \in \mathcal{M}^e(A, C)$ such that $\|U - V\|_1 \leq \epsilon$. □

In order to obtain an upper bound for the chain $\left\{X_k^{\mathcal{E}}\right\}_{k\geq 0}$, we can now apply Proposition 4.3 and Corollary 4.1:

Corollary 4.2. *The smallest $\preccurlyeq_{st}$-monotone upper bound for $\left\{X_k^{\mathcal{E}}\right\}_{k\geq 0}$ is given by the transition probability matrix:*

$$r^{-1}(v(DPY(A, C)))$$

Remark 4.1 (Lower bounds)**.** *Similar algorithm to compute lower bounds can be obtained using the symmetry of $\preccurlyeq_{st}$ order. The details are omitted for the sake of conciseness.*

We proved the optimality of DPY for the case when only blocks A and C are known. In the following section we consider the case when we have some additional information about blocks B and D and how we can improve the bounds taking into account this new information.

4.6 Using Additional Information

We consider in this section different assumptions on the (partial) knowledge of blocks of matrix P and we show how this can be used to improve bounds for the stochastic complement. In general, computing the bounds consists in two parts:

1. Find a deterministic part that we can obtain from A, C, and all the additional information on the model.
2. Then apply DPY to the unknown part. Thus the optimality of DPY is not sufficient in general to imply the optimality of these bounds.

4.6.1 Known Blocks A, B and C

Let us first assume that we also know block B. Computing blocks A, B and C requires that we know both the predecessor function and the successor function. Using predecessor function we get the column of the stochastic matrix for all the states in $\mathcal{E}$ (i.e. blocks A and C) while the successor function gives rows of the states in $\mathcal{E}$ (i.e. blocks A and B).

Proposition 4.5. *Assume that A, B and C are known. Then:*

$$S_A \preccurlyeq_{st} DPY\,(A + BC, C)$$

Proof. The proof is based on two steps. First we build a new expression for the stochastic complement associated with a new matrix. Then we prove that the matrix we have built is stochastic and we use DPY to obtain a bound of the stochastic complement of that matrix. Let us recall relation 4.3 and recall that as D does not contain any recurrent class we have:

$$(Id - D)^{-1} = \sum_{i=0}^{\infty} D^i = Id + D\,(Id - D)^{-1}$$

After substitution we get:

$$S_A = A + B\left(Id + D\,(IdD)^{-1}\right) W\,(W^*)^{-1}\,C$$

After simplification we obtain:

$$S_A = A + BC + BD\,(Id - D)^{-1}\,W\,(W^*)^{-1}\,C \tag{4.12}$$

Therefore we obtain S_A as the complement of matrix

$$\left(\begin{array}{c|c} A + BC & BD \\ \hline C & D \end{array}\right)$$

Simple algebraic manipulations allow to prove that this matrix is stochastic. Thus S_A is upper bounded by $DPY\,(A + BC, C)$. □

Example 5 Using as example the same blocks A, B and C already defined, we obtained a new upper bound of the stochastic complement S_A denoted as $H0$. Clearly the bound is better than the one obtained with DPY using only A and C:

$$H0 = \begin{bmatrix} 0.2980 & 0.2170 & 0.3925 & 0.0925 \\ 0.2520 & 0.1780 & 0.4350 & 0.1350 \\ 0.0740 & 0.5910 & 0.0675 & 0.2675 \\ 0.1880 & 0.1120 & 0.4000 & 0.3000 \end{bmatrix}$$

4.6.2 All Blocks are known, but $(Id - D)^{-1}$ is Difficult to Compute

Now assume that we also know D, but we cannot compute $(Id-D)^{-1}$ because of the computational complexity. This assumption is similar to the one developed in [FPY07b] where graph theoretical arguments were used to obtain bounds.

Proposition 4.6. *For any* $K \geq 0$, $S_A \preccurlyeq_{st} DPY\left(A + B\sum_{i=0}^{K} D^i C, C\right)$

The proof relies on the same technique as Proposition 4.5 and is omitted.

Example 6 Let us turn back now to the example for the same blocks and for $K = 1$ (bound $H1$) and $K = 2$ (bound $H2$).

$$H1 = \begin{bmatrix} 0.309000 & 0.220500 & 0.393250 & 0.077250 \\ 0.273200 & 0.181800 & 0.430500 & 0.114500 \\ 0.087400 & 0.591100 & 0.060750 & 0.260750 \\ 0.189600 & 0.118400 & 0.412000 & 0.280000 \end{bmatrix}$$

$$H2 = \begin{bmatrix} 0.318780 & 0.221970 & 0.390825 & 0.068425 \\ 0.290520 & 0.183780 & 0.424850 & 0.100850 \\ 0.098340 & 0.591910 & 0.056475 & 0.253275 \\ 0.194080 & 0.120320 & 0.413600 & 0.272000 \end{bmatrix}$$

For the same blocks and for $K = 1$ we have also computed the bound obtained with an algorithm [FPY07b], based on breadth first search visit of the successors of the nodes in $\mathcal{E}$. The results are clearly less accurate than the bounds we obtain with Proposition 4.6:

$$FPY1 = \begin{bmatrix} 0.256000 & 0.139000 & 0.322000 & 0.283000 \\ 0.192000 & 0.058000 & 0.324000 & 0.426000 \\ 0.042000 & 0.523000 & 0.004000 & 0.431000 \\ 0.140000 & 0.040000 & 0.340000 & 0.480000 \end{bmatrix}$$

4.6.3 Componentwise Bounds on $(Id - D)^{-1}$

Here we assume that we know the blocks A, B and C. Additionally, we know componentwise lower bounds for matrix $(Id - D)^{-1}$:

$$F \leq (Id - D)^{-1}$$

The block D is either not completely known or $(Id - D)^{-1}$ is difficult to inverse as in the cases 4.6.1 and 4.6.2. Recall that $(Id - D)^{-1} = \sum_{i=0}^{\infty} D^i$, so Proposition 4.6 can be seen as a special case for $F = \sum_{i=0}^{K} D^i$.

We introduce first a new decomposition of stochastic complement (Proposition 4.7) that we need to prove bounds in Proposition 4.9. Matrix S_A can be decomposed as:

$$S_A = A + BFC + B\left((Id - D)^{-1} - F\right) W \left(W^*\right)^{-1} C$$

Let $G = \left((Id - D)^{-1} - F\right) W$, $V = diag\left(G\mathbf{1}^t\right)$ and $V^* = diag\left((G\mathbf{1}^t)^*\right)$. Then we have an additional decomposition of stochastic complement:

Proposition 4.7. *Matrix S_A can be decomposed as:*

$$S_A = A + BFC + BV \left(V^*\right)^{-1} \left((Id - D)^{-1} - F\right) W \left(W^*\right)^{-1} C \tag{4.13}$$

Matrices $\left(W^*\right)^{-1} C = n\left(C\right)$ and $\left(V^*\right)^{-1} \left((Id - D)^{-1} - F\right) W = n\left(G\right)$ have rows that are either stochastic or equal to $\mathbf{0}$.

Proof. Matrix S_A can be written as:

$$S_A = A + BFC + BV^* \left(V^*\right)^{-1} \left((Id - D)^{-1} - F\right) W \left(W^*\right)^{-1} C \tag{4.14}$$

Recall that $\left(V^*\right)^{-1} \left((Id - D)^{-1} - F\right) W = n\left(G\right)$, $V = diag\left(G\mathbf{1}^t\right)$ and $V^* = diag\left((G\mathbf{1}^t)^*\right)$. Thus, Lemma 4.1 implies that:

$$V^* \left(V^*\right)^{-1} \left((Id - D)^{-1} - F\right) W = V \left(V^*\right)^{-1} \left((Id - D)^{-1} - F\right) W \tag{4.15}$$

Relation 4.13 now follows from 4.14 and 4.15 □

One possible way to obtain such a matrix F is to obtain first E, a lower bound of D, and then compute or approximate $(Id - E)^{-1}$ such as mentioned in the next proposition.

Proposition 4.8. *As matrix D is non negative and such that $(Id - D)$ is not singular, then for all non negative matrix $E \leq D$, we have $(Id - E)^{-1} \leq (Id - D)^{-1}$.*

The straightforward proof is omitted.

Proposition 4.9. *Assume that we know the blocks A, B, C and the matrix F such that:*

$$F \leq (Id - D)^{-1}$$

Then:

$$S_A \preccurlyeq_{st} A + BFC + \gamma r^{-1}(s(n(C)))$$

where $\gamma = \mathbf{1}^t - (A + BFC)\mathbf{1}^t$

Proof. By Proposition 4.7, we have $S_A = A + BFC + BVn(G)n(C)$, with $G = \left((Id - D)^{-1} - F\right)W$. Lemma 4.3 for $Z = BVn(G)$ and $M = n(C)$ implies:

$$BVn(G)n(C) \preccurlyeq_{st} \alpha r^{-1}(s(n(C)))$$

for $\alpha = ZM\mathbf{1}^t = BVn(G)n(C)\mathbf{1}^t$. Therefore:

$$S_A \preccurlyeq_{st} A + BFC + \alpha r^{-1}(s(n(C))) \tag{4.16}$$

After multiplying 4.13 by $\mathbf{1}^t$, we get: $\mathbf{1}^t = (A + BFC)\mathbf{1}^t + BVn(G)n(C)$, so:

$$\alpha = BVn(G)n(C)\mathbf{1}^t = \mathbf{1}^t - (A + BFC)\mathbf{1}^t = \gamma$$

□

Example 7 Consider the same block decomposition as in Example 4. Assume that we know an element-wise lower bound of D denoted as $D1$ and which is equal to:

$$\begin{bmatrix} 0.500 & 0.200 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.200 & 0.600 & 0.200 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \end{bmatrix}$$

Note that $D1$ is obtained by a breadth first search visit of the nodes in $\mathcal{E}^c$ started with nodes of $\mathcal{E}$ and limited by a depth of two nodes. We easily compute $(Id - D1)^{-1}$:

$$\begin{bmatrix} 2.500 & 1.250 & 0.250 & 0.000 & 0.000 & 0.000 \\ 1.250 & 3.125 & 0.625 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \end{bmatrix}$$

Finally the bounds of S_A are:

$$\begin{bmatrix} 0.3382 & 0.2180 & 0.3718 & 00718 \\ 0.3325 & 0.1800 & 0.3937 & 0.0937 \\ 0.1330 & 0.5920 & 0.0375 & 0.2375 \\ 0.1880 & 0.1120 & 0.4000 & 0.3000 \end{bmatrix}$$

4.6.4 Componentwise bounds on $(Id - D)^{-1}$ and C

Here we assume that we do not know the block C, but only its elementwise lower bounds. Then we can extract the known part and apply the Truffet's algorithm to the remaining part:

Proposition 4.10. *Assume we know the blocks A and B and the componentwise lower bounds of block C and matrix $(Id - D)^{-1}$:*

$$F \leq (Id - D)^{-1} \quad and \quad H \leq C$$

Then:

$$S_A \preccurlyeq_{st} T(A + BFH) = A + BFH + \delta(0, ..., 0, 1)$$

where $\delta = \mathbf{1}^t - (A + BFH)\mathbf{1}^t$

Observe that the product $\delta(0, ..., 0, 1)$ is a matrix with appropriate dimensions.

Proof. We have:

$$S_A = A + BFH + B\left(F(C - H) + \left((Id - D)^{-1} - F\right)C\right)$$

Denote by $U = F(C - H) + \left((Id - D)^{-1} - F\right)C$. After multiplying by $\mathbf{1}^t$, we get:

$$\mathbf{1}^t = (A + BFH)\mathbf{1}^t + BU\mathbf{1}^t$$

We have $\delta = \mathbf{1}^t - (A + BFH)\mathbf{1}^t = BU\mathbf{1}^t$. Now from

$$S_A = A + BFH + Bdiag\left(U\mathbf{1}^t\right)n(U) \tag{4.17}$$

and Lemma 4.3 for $Z = Bdiag(U\mathbf{1}^t)$ and $M = n(U)$ implies:

$$Bdiag\left(U\mathbf{1}^t\right)n(U) \preccurlyeq_{st} \alpha r^{-1}(s(n(U))) \tag{4.18}$$

for $\alpha = ZM\mathbf{1}^t = Bdiag(U\mathbf{1}^t)n(U)\mathbf{1}^t = BU\mathbf{1}^t = \delta$. Now from 4.17 and 4.18 we obtain:

$$S_A \preccurlyeq_{st} A + BFH + \delta r^{-1}(s(n(U))) \preccurlyeq_{st} A + BFH + \delta(0, ..., 0, 1)$$

where $\delta = \mathbf{1}^t - (A + BFH)\mathbf{1}^t$ □

Example 8 We consider again Example 4. We assume that we have been able to obtain $C1$ an element-wise lower bound of C and $D1$ an element wise lower bound of D. For the sake of simplicity we assume that $D1$ is already defined in Example 7. Assume that $C1$ is equal to:

$$C1 = \begin{bmatrix} 0.2 & 0.1 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Note that this information on $C1$ and $D1$ may be obtained from a breadth first search visit out of states in $\mathcal{E}$ but limited to two nodes (i.e. the first two nodes of $\mathcal{E}^c$ visited from $\mathcal{E}$). Finally the bounding matrix of S_A is:

$$\begin{bmatrix} 0.234 & 0.117 & 0.300 & 0.349 \\ 0.168 & 0.034 & 0.300 & 0.498 \\ 0.038 & 0.519 & 0.000 & 0.443 \\ 0.100 & 0.000 & 0.300 & 0.600 \end{bmatrix}$$

4.6.5 Decomposition

We assume now that the complement can be divided into two or more non-communicating subsets. Such an information may be provided by the tensor based representation of the DTMC (for some relations between tensor representation of the chain and its graph properties, see [FQ95]). We assume that block D has the diagonal block form:

$$D = diag\left(D_{1,1}, D_{2,2}, ..., D_{K,K}\right) = \begin{bmatrix} D_{1,1} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & D_{2,2} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & & \ddots & & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & D_{K-1,K-1} & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & D_{K,K} \end{bmatrix}$$

where $\mathbf{0}$ stands here for a matrix of appropriate size whose all elements are equal to 0. Let m_k be the size of block $1 \leq k \leq K$. Blocks B and C can also be decomposed as:

$$B = \begin{bmatrix} B_1 & B_2 & \cdots & B_K \end{bmatrix}, \quad C = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_K \end{bmatrix}$$

where B_k is an $n \times m_k$ matrix and C_k an $m_k \times n$ matrix, for $1 \leq k \leq K$

Proposition 4.11. *Let $\beta_k = B_k \mathbf{1}^t$, for all $1 \leq k \leq K$. We assume that blocks A and C are known and that we also know all β_k. Then:*

$$S_A \preccurlyeq_{st} A + \sum_{k=1}^{K} \beta_k s_k$$

where $s_k = r^{-1}\left(s\left(n\left(C_k\right)\right)\right), \quad 1 \leq k \leq K.$

Proof. We have:

$$S_A = A + \sum_{k=1}^{K} B_k \left(Id - D_{k,k}\right)^{-1} W_k n\left(C_k\right)$$

where $W_k = diag\left(C_k \mathbf{1}^t\right)$ and $n\left(C_k\right) = \left(W_k^*\right)^{-1} C_k$, $W_k^* = diag\left(\left(C_k \mathbf{1}^t\right)^*\right)$, for all $1 \leq k \leq K$. Thus by Lemma 4.3,

$$B_k \left(Id - D_{k,k}\right)^{-1} W_k n\left(C_k\right) \preccurlyeq_{st} B_k \left(Id - D_{k,k}\right)^{-1} W_k \mathbf{1}^t s_k$$

where $s_k = r^{-1}\left(s\left(n\left(C_k\right)\right)\right)$. Similarly as in the proof of Proposition 4.1, it can be shown that $\left(Id - D_{k,k}\right)^{-1} W_k$ is a stochastic matrix. We obtain, $B_k \left(Id - D_{k,k}\right) W_k \mathbf{1}^t = B_k \mathbf{1}^t = \beta_k$ and

$$B_k \left(Id - D_{k,k}\right)^{-1} W_k n\left(C_k\right) \preccurlyeq_{st} \beta_k s_k, \text{ for all } 1 \leq k \leq K$$

Thus,

$$S_A \preccurlyeq_{st} A + \sum_{k=1}^{K} \beta_k s_k$$

□

Example 9 We now slightly modify Example 4. We consider the same state space and the same block decomposition except block D which is modified as follows:

$$D2 = \left[\begin{array}{cc|cccc} 0.5 & 0.2 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.4 & 0.6 & 0.0 & 0.0 & 0.0 & 0.0 \\ \hline 0.0 & 0.0 & 0.2 & 0.4 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.3 & 0.0 & 0.2 & 0.0 \\ 0.0 & 0.0 & 0.1 & 0.0 & 0.0 & 0.4 \\ 0.0 & 0.0 & 0.5 & 0.0 & 0.1 & 0.1 \end{array}\right]$$

The exact solution for the stochastic complement and the bound $BC2$ based on Proposition 4.11 are respectively:

$$S_A = \begin{bmatrix} 0.384 & 0.228 & 0.358 & 0.030 \\ 0.417 & 0.195 & 0.358 & 0.030 \\ 0.200 & 0.600 & 0.000 & 0.200 \\ 0.201 & 0.123 & 0.415 & 0.261 \end{bmatrix}, \quad B2C = \begin{bmatrix} 0.373 & 0.227 & 0.350 & 0.050 \\ 0.407 & 0.193 & 0.350 & 0.050 \\ 0.200 & 0.600 & 0.000 & 0.200 \\ 0.180 & 0.120 & 0.400 & 0.300 \end{bmatrix}$$

The bound in Proposition 4.11 can be seen as a refinement of DPY bound (see Section 4.4, equation 4.11), using the decomposition of matrix D. Thus, as for DPY, we can similarly use additional knowledge on blocks B, C and D to improve the bounds in Proposition 4.11. For example, if we know blocks $B_1, \cdots, B_K$, then by combining the results of Subsection 4.6.1 with Proposition 4.11 we get:

$$S_A \preccurlyeq_{st} A + \sum_{k=1}^{K} (B_k C_k + \gamma_k s_k)$$

where $s_k = r^{-1}(s(n(C_k)))$ and $\gamma_k = \beta_k - B_k C_k \mathbf{1}^t$, $1 \leq k \leq K$. The proof uses similar arguments as the proofs of Proposition 4.5 and 4.11. We leave to the reader the details of the proof, as well as similar generalizations obtained by combining Proposition 4.11 with results in Subsections 4.6.2 and 4.6.4.

4.7 Conclusion

Our approach gives a theoretical framework for the partial generation of the state-space and the transition matrix of a really large Markov chain. Partial generation is often performed heuristically by software tools without any control on the accuracy of the results. If the chain is designed using an initial state and the successor function, when we stop the generation, we obtain blocks A and B. Similarly, using an initial state and the predecessor function we get blocks A and C when the partial generation is achieved. Tensor based representation [FPS98] allows to build all blocks, but it is also possible to take advantage of a partial representation to reduce the complexity of the computational algorithms. Clearly, the more information (i.e. blocks) we put in the model, the more accurate are the bounds. Similarly, when we increase the number of steps to obtain a more accurate version of the blocks (i.e. parameter K in the visit-based algorithms in Subsection 4.6.2), we also increase the tightness of the bounds. We also want to emphasize the importance of DPY algorithm, which is optimal when only A and C are known and which allows to derive better bounds when we add further useful information.

Part III

Statistical Methods

Chapter 5

Hybrid Automata Stochastic Logic

This chapter is related to the publication [BDD+11b].

5.1 Introduction

From model checking to quantitative model checking. Since its introduction [EC80], model checking has quickly become a prominent technique for verification of discrete-event systems. Its success is mainly due to three factors: (1) the ability to express specific properties by formulas of an appropriate logic, (2) the firm mathematical foundations based on automata theory and (3) the simplicity of the verification algorithms which has led to the development of numerous tools. While the study of systems requires both functional, performance and dependability analysis, originally the techniques associated with these kinds of analysis were different. However, in the mid nineties, classical temporal logics were adapted to express properties of Markov chains and a decision procedure has been designed based on transient analysis of Markov chains [BHHK03a].

From numerical model checking to statistical model checking. The numerical techniques for quantitative model checking are rather efficient when a memorylessness property can be exhibited (or recovered by a finite-state memory), limiting the combinatory explosion due to the necessity to keep track of the sampling of distributions. Unfortunately both the formula associated with an elaborated property and the stochastic process associated with a real application make rare the possibility of such pattern. In these cases, statistical model checking [YS06] is thus an alternative to numerical techniques. Roughly speaking, statistical model checking consists in sampling executions of the system (possibly synchronized with some automaton corresponding to the formula to

be checked) and comparing the ratio of successful executions with a threshold specified by the formula. The advantage of the statistical model checking is the small memory requirement while its drawback is its inability to generate samples for execution paths of potentially unbounded length.

Limitations of existing logics. However, a topic that has not been investigated is the suitability of the temporal logic to express (non necessarily boolean) quantities defined by path operators (minimum, integration, etc.) applied on instantaneous indicators. Such quantities naturally occur in standard performance evaluation. For instance, the average length of a waiting queue during a busy period or the mean waiting time of a client are typical measures that cannot be expressed by the quantitative logics based on the concept of successful execution probability like CSL [ASVB00].

Our contribution. We introduce a new formalism called Hybrid Automaton Stochastic Logic (HASL) which provides a unified framework both for model checking and for performance and dependability evaluation. A HASL formula evaluates to a real number which is defined by the expectation of a path random variable conditioned by the success of the path. The concept of conditional expectation significantly enlarges the expressive power of the logic. The proposed temporal logic is indeed a quantitative logic permitting both to check if probability thresholds are met and to evaluate complex performability measures. A formula of HASL consists of an automaton and an expression. The automaton is a Linear Hybrid Automaton (LHA), i.e. an automaton with clocks, called in this context *data variables*, where the dynamic of each variable (i.e. the variable's evolution) depends on the model states. This automaton will synchronize with the DESP, precisely selecting accepting paths while maintaining detailed information on the path through data variables. The expression is based on moments of path random variables associated to path executions. These variables are obtained by operators like time integration on data variables. HASL extends the expressiveness of automaton-based CSL like formalisms as CSL^{TA} [DHS09] and its extension to multi-clocks [CHKM09] with state and action rewards and sophisticated update functions especially useful for performance and dependability evaluation. On the other hand it extends reward enriched versions of CSL, (CSRL [BHHK00a]) with a more precise selection of path executions, and the possibility to consider multiple rewards. Therefore HASL makes it possible to consider not only standard performability measures but also complex ones in a generic manner.

A statistical verification tool (i.e. a discrete event simulator) COSMOS has been developed for this logic. We have chosen generalized stochastic Petri nets (GSPN) as high level formalism for the description of the discrete event stochas-

tic process since (1) it allows a flexible modeling w.r.t. the policies defining the process (choice, service and memory) and (2) due to the locality of net transitions and the simplicity of the firing rule it leads to efficient path generation.

Organization. In section 5.2 we describe the class of stochastic models we refer to (i.e. DESP). In section 5.3 we formally introduce the HASL logic and we provide an overview of the related work, where the expressiveness of HASL is compared with that of existing logics. Finally, in section 5.4, we conclude and give some perspectives.

5.2 DESP

We describe the class of stochastic models that are suitable for HASL verification, namely Discrete Event Stochastic Processes (DESP). Such class includes Markov chain models, the main type of stochastic models targeted by existing stochastic logics. The definition of DESP we introduce resembles that of generalized semi-Markov processes [Gly83] as well as that given in [ACD91].

Syntax. DESPs are stochastic processes consisting of a (possibly infinite) set of states and whose dynamic is triggered by a set of discrete events. We do not consider any restriction on the nature of the distribution associated with events. In the sequel $dist(A)$ denotes the set of distributions whose support is A.

Definition 5.1. *A DESP is a tuple*

$\mathcal{D} = \langle S, \pi_0, E, Ind, enabled, delay, choice, target \rangle$ *where:*

- S *is a (possibly infinite) set of discrete states,*
- $\pi_0 \in dist(S)$ *is the initial distribution on states,*
- E *is a set of events,*
- Ind *is a set of functions from* S *to* $\mathbb{R}$ *called state indicators (including the constant functions),*
- $enabled : S \to 2^E$ *are the enabled events in each state with for all* $s \in S$, $enabled(s) \neq \emptyset$.
- $delay \colon S \times E \to dist(\mathbb{R}^+)$ *is a partial function defined for pairs* (s, e) *such that* $s \in S$ *and* $e \in enabled(s)$.
- $choice \colon S \times 2^E \times \mathbb{R}^+ \to dist(E)$ *is a partial function defined for tuples* (s, E', d) *such that* $E' \subseteq enabled(s)$ *and such that the possible outcomes of the corresponding distribution are restricted to* $e \in E'$.

- $target : S \times E \times \mathbb{R}^+ \to S$ *is a partial function describing state changes through events, defined for tuples* (s, e, d) *such that* $e \in enabled(s)$.

Before giving the operational semantics of a DESP, we informally describe its items. Given a state s, $enabled(s)$ is the set of events enabled in s. For an event $e \in enabled(s)$, $delay(s, e)$ is the distribution of the delay between the enabling of e and its possible occurrence. Furthermore, if we denote d the earliest delay in some configuration of the process with state s, and $E' \subseteq enabled(s)$ the set of events with earliest delay, $choice(s, E', d)$ describes how the conflict is randomly resolved: for all $e' \in E'$, $choice(s, E', d)(e')$ is the probability that e' will be selected among E' after waiting for the delay d. The function $target(s, e, d)$ denotes the target state reached from s on occurrence of e after waiting for d time units.

We define the subset $Prop \subseteq Ind$ of *state propositions* taking values in $\{0, 1\}$. The sets Ind and $Prop$ will be used in the sequel to characterize the information on the DESP known by the automaton (LHA) corresponding to a formula. In fact the LHA does not have direct access to the current state of the DESP but only through the values of the state indicators and state propositions.

Semantics. In order to define the semantics of this class of DESPs, we consider the following policies: choice is driven by the *race policy* (i.e. the event with the shortest delay occurs first), the service policy is *single server* (at most one instance per event may be scheduled) and the memory policy is the *enabling memory* one (i.e. a scheduled event remains so until executed or until it becomes disabled). Other policies could have been selected (resampling memory, age memory, etc.). We have stuck to the most usual policies for the sake of simplicity.

Given a discrete event system, its execution is characterized by a (possibly infinite) sequence of events $\{e_1, e_2, \ldots\}$ and occurrence time of these events. Only the events can change the state of the system.

In the stochastic framework, the behaviour of a DESP is defined by three families of random variables:

- $e_1, \ldots, e_n, \ldots$ defined over the set of events E denoting the sequence of events occuring in the system
- $s_0, \ldots, s_n, \ldots$ defined over the (discrete) state space of the system, denoted as S. s_0 is the system initial state and s_n for $n > 0$ is the state after the n^{th}

event. The occurrence of an event does not necessarily modify the state of the system, and therefore s_{n+1} may be equal to s_n.

- $\tau_0 \leq \tau_1 \leq \cdots \leq \tau_n \leq \cdots$ defined over $\mathbb{R}^+$, where τ_0 is the initial instant and τ_n for $n > 0$ is the instant of the occurrence of the n^{th} event.

We start from the syntactical definition of a DESP and show how we obtain the three families of random variables $\{s_n\}_{n\in\mathbb{N}}$, $\{e_n\}_{n\in\mathbb{N}^*}$ and $\{\tau_n\}_{n\in\mathbb{N}}$. This definition is inductive w.r.t. n and includes some auxiliary families.

Notation. In the whole section, when we write an expression like $Pr(e_{n+1} = e \mid e \in E'_n)$, we also mean that this conditional probability is independent from any event Ev that could be defined using the previously defined random variables:

$$Pr(e_{n+1} = e \mid e \in E'_n) = Pr(e_{n+1} = e \mid e \in E'_n \wedge Ev)$$

The family $\{sched(e)_n\}_{n\in\mathbb{N}}$ whose range is $\mathbb{R}^+ \cup \{+\infty\}$ denotes whether the event e is scheduled at the n^{th} state and what is the current schedule. Now τ_{n+1} is defined by $\tau_{n+1} = \min(sched(e)_n \mid e \in E)$ and the family $\{E'_n\}_{n\in\mathbb{N}}$ denotes the set of events with minimal schedule: $E'_n = \{e \in E \mid \forall e' \in E, sched(e)_n \leq sched(e')_n\}$. From this family we obtain the conditional distribution of e_{n+1}: $Pr(e_{n+1} = e \mid e \in E'_n \wedge \tau_{n+1} - \tau_n = d) = choice(s_n, E'_n, d)(e)$

Now $s_{n+1} = target(s_n, e_n, \tau_{n+1} - \tau_n)$ and:

- For every $e \in E$,
  $Pr(sched(e)_{n+1} = \infty \mid e \notin enabled(s_{n+1})) = 1$
- For every $e \in E$,
  $Pr(sched(e)_{n+1} = sched(e)_n \mid$
  $e \in enabled(s_{n+1}) \cap enabled(s_n) \wedge e \neq e_n) = 1$
- For every $e \in E$ and $d \in \mathbb{R}^+$,
  $Pr(\tau_n \leq sched(e)_{n+1} \leq \tau_n + d \mid$
  $e \in enabled(s_{n+1}) \wedge (e \notin enabled(s_n) \vee e = e_n))$
  $= delay(s_{n+1}, e)(d)$

We start the induction by:

- $Pr(\tau_0 = 0) = 1$, $Pr(s_0 = s) = \pi_0(s)$
- For every $e \in E$,
  $Pr(sched(e)_0 = \infty \mid e \notin enabled(s_0)) = 1$

- For every $e \in E$ and $d \in \mathbb{R}^+$,
  $Pr(sched(e)_0 \leq d \mid e \in enabled(s_0))$
  $= delay(s_0, e)(d)$

A *configuration* of a DESP is described as a triple $(s\ ,\ \tau\ ,\ sched)$ with s being the current state, $\tau \in \mathbb{R}^+$ the current time and $sched : E \rightarrow \mathbb{R}^+ \cup \{+\infty\}$ being the function that describes the occurrence time of each scheduled event ($+\infty$ if the event is not yet scheduled). Note that, because of its definition, the evolution of a DESP is naturally suitable for discrete event simulation. However, while it can model almost all interesting stochastic processes, it is a low level representation since the set of states is explicitely described. A solution for a higher level modelling is to choose one of the formalisms commonly used for representing Markov chain models (e.g. Stochastic Petri Nets [ABC+95] or Stochastic Process Algebras [GHH02]), that can straightforwardly be adapted for representation of a large class of DESPs. It suffices that the original formalisms are provided with formal means to represent the type of delay distribution of each transition/action (function $delay$ of Definition 5.1) as well as means to encode the probabilistic choice between concurrent events (i.e. function $choice$ of Definition 5.1). However, due to syntactic restrictions, Stochastic Petri Nets and other formalisms cannot (in their original definition) capture all the functionalities of DESPs.

In the following we describe an example of DESP expressed in terms of a Generalized Stochastic Petri Net (GSPN). This model will be used in Section 5.3 for describing, through a couple of LHA examples, the intuition behind Hybrid Automata based verification. Before describing the running example, we informally outline the basis of GSPN specification (for a formal account we refer the reader to [ABC+95]), pointing out the differences between "original" GSPNs and GSPNs for representing DESPs (which we refer to as GSPN-DESP).

GSPN models. A GSPN model is a bi-partite graph consisting of two classes of nodes, *places* and *transitions*. Places may contain *tokens* (representing the state of the modeled system) while transitions indicate how tokens "flow" within the net (encoding the model dynamics). The state of a GSPN consists of a *marking* indicating the distribution of tokens throughout the places (i.e. how many tokens each place contains). Roughly speaking a transition t is enabled whenever every *input place* of t contains a number of tokens greater than or equal to the multiplicity of the corresponding (input) arc. An enabled transition may *fire*, consuming tokens (in a number indicated by the multiplicity of the corresponding input arcs) from its input places, and producing tokens (in a number indicated by the multiplicity of the corresponding output arcs) in its *output places*.

Transitions can be either *timed* (denoted by empty bars) or *immediate* (denoted by filled-in bars, see Figure 5.1). Transitions are characterized by: (1) a *distribution* which randomly determines the delay before firing it; (2) a *priority* which deterministically selects among the transitions with earliest scheduling time, the one to be fired; (3) a *weight*, that is used in the random choice between transitions scheduled the soonest with the same highest priority. With the GSPN formalism [ABC+95] the delay of timed transitions is assumed *exponentially* distributed, whereas with GSPN-DESP it can be given by any distribution. Thus a GSPN-DESP timed-transition is characterized by a tuple: $t \equiv (type, par, pri, w)$, where $type$ indicates the type of distribution (e.g. uniform), par indicates the parameters of the distribution (e.g $[\alpha, \beta]$), $pri \in \mathbb{R}^+$ is a priority assigned to the transition and $w \in \mathbb{R}^+$ is used to probabilistically choose between transitions occurring with equal delay and equal priority. Observe that the information associated to a transition (i.e. $type, par, pri, w$) is exploited in different manners depending on the type of transition. For example for a transition with a continuous distribution the priority (pri) and weight (w) records are superfluous (hence ignored) since the probability that the schedule of the corresponding event is equal to the schedule of (the event corresponding to) another transition is null. Similarly, for an immediate transition (denoted by a filled-in bar) the specification of the distribution type (i.e. $type$) and associated parameters (par) is not needed (hence also ignored). Therefore these unnecessary information are omitted in Figure 5.1)

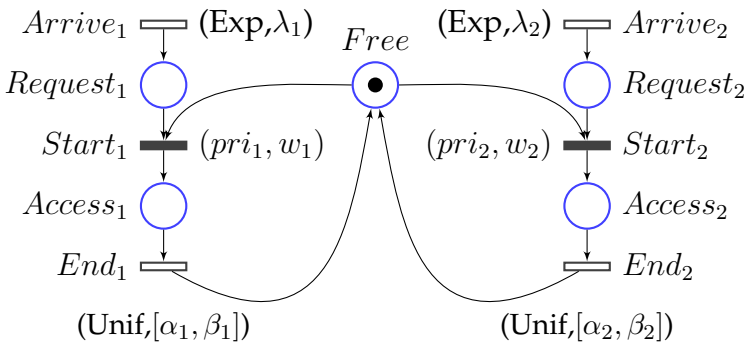


Figure 5.1: The GSPN description of a shared memory system.

Running example. We consider the GSPN model (inspired by [ABC+95]) of Figure 5.1. It describes the behavior of an open system where two classes of clients (namely 1 and 2) compete to access a shared memory (resource). Class

i-clients ($i \in \{1, 2\}$) enter the system according to a Poisson process with parameter λ_i (corresponding to the exponentially distributed timed transition $Arrive_i$ with rate λ_i). On arrival, clients cumulate in places $Request_i$ where they wait for the memory to be free (a token in place $Free$ witnessing that the memory is available). The exclusive access to the shared memory is regulated either deterministically or probabilistically by the *priority* (pri_i) and the *weight* (w_i) of immediate transitions $Start_1$ and $Start_2$. Thus in presence of a competition (i.e. one or more tokens in both $Request_1$ and $Request_2$) a class i process wins the competition with a class $j = (i \mod 2)+1$ process with probability 1 if $pri_i > pri_j$, and with probability $w_i/(w_i+w_j)$ if $pri_i = pri_j$. The occupation time of the memory by a class i client is assumed to be uniformly distributed within the interval $[\alpha_i, \beta_i]$ (corresponding to transitions End_i). Thus on firing of transition End_i the memory is released and a class i client leaves the system.

5.3 HASL

We intuitively describe the syntax and semantics of HASL before formally defining them in the next subsections. A formula of HASL consists of two parts:

- The first component of a formula is a hybrid automaton that synchronizes with an infinite timed execution of the considered DESP until some final state of the automaton is reached or the synchronization fails. During this synchronization, some data variables evolve and also condition the evolution of this synchronization.
- The second component of a formula is an expression whose operands are mainly data variables and whose operators will be described formally later in this section. In order to express path indices, they include path operators such as min and max value along an execution, value at the end of a path, integral over time and the average value operator. Conditional expectations are applied to these indices in order to obtain the value of the formula.

5.3.1 Synchronized Linear Hybrid Automata

Syntax. The first component of a HASL formula is a restriction of hybrid automata [ACHH92], namely synchronized Linear Hybrid Automata (LHA). LHA extend the Deterministic Timed Automata (DTA) used to describe properties of Markov chain models [DHS09, CHKM09]. Simply speaking, LHA are

automata whose set of *locations* is associated with a n-tuple X of real-valued variables (called data variables) and whose rate can vary.

In our context, LHA are used to synchronize with DESP paths. However they can evolve in an autonomous way: thus the symbol $\sharp$ denotes a pseudo-event that is not included in the event set E of the DESP associated with these autonomous changes. The values of the data variables $x_1, \ldots, x_n$ evolve with a linear rate depending on the location of the automaton and on the current state of the DESP. More precisely the function *flow* associates with each location a n-tuple of indicators (one for each variable), and given a state s of a DESP and a location l the flow of variable x_i in (s, l) is $\mathit{flow}_i(l)(s)$ (where $\mathit{flow}_i(l)$ is the i^{th} component of $\mathit{flow}(l)$). Our model also uses *constraints*, which describe the conditions for an edge to be traversed, and *updates*, which describe the actions taken on the data variables on traversing an edge. A *constraint* of an LHA edge is a boolean combination of inequalities of the form $\sum_{1 \leq i \leq n} \alpha_i x_i + c \prec 0$ where α_i and c are indicators (*i.e.* in Ind), $\prec$ stands for either $=$, $<$, $>$, $\leq$ or $\geq$. The set of constraints is denoted by Const. Given a location and a state, an expression of the form $\sum_{1 \leq i \leq n} \alpha_i x_i + c$ evolves linearly with time. An inequality thus gives an interval of time during which the constraint is satisfied. We say that a constraint is left closed if, whatever the current state s of the DESP (defining the values of indicators), the time at which the constraint is satisfied is a union of left closed intervals. We denote by lConst the set of left closed constraints that are used for the "autonomous" edges (i.e. those labelled by $\sharp$). An *update* is more general than the reset of timed automata. Here each data variable can be set to a linear function of the variables' values. An update U is then a n-tuple of functions $u_1, ..., u_n$ where each u_k is of the form $x_k = \sum_{1 \leq i \leq n} \alpha_i x_i + c$ where $\alpha_i \in Ind$ and c are indicators. The set of updates is denoted by Up.

Definition 5.2. *A* synchronized linear hybrid automaton *(LHA)* $\mathcal{A} = \langle E, L, \Lambda, \mathit{Init}, \mathit{Final}, X, \mathit{flow}, \rightarrow \rangle$ *comprises:*

- E*, a finite alphabet of events;*
- L*, a finite set of locations;*
- $\Lambda : L \rightarrow \mathit{Prop}$*, a location labelling function;*
- Init*, a subset of* L *called the initial locations;*
- Final*, a subset of* L *called the final locations;*
- $X = (x_1, ...x_n)$ *a* n*-tuple of data variables;*
- *a function flow* $: L \mapsto \mathit{Ind}^n$ *which associates to each location one indicator for each data variable representing the evolution rate of the variable in this location.* flow_i *denotes the projection of flow on its* i^{th} *component.*

- $\rightarrow \subseteq L \times \big((\mathsf{Const} \times 2^E) \uplus (\mathsf{IConst} \times \{\sharp\})\big) \times \mathsf{Up} \times L$, *a set of edges, where the notation* $l \xrightarrow{\gamma,E',U} l'$ *means that* $(l, \gamma, E', U, l') \in \rightarrow$.

The edges labelled with a set of events in 2^E *are called* synchronized *whereas those labelled with* $\sharp$ *are called* autonomous. *Furthermore* $\mathcal{A}$ *fulfills the following conditions.*

- **Initial determinism:** $\forall l \neq l' \in Init$, $\Lambda(l) \wedge \Lambda(l') \Leftrightarrow \texttt{false}$ [1].
- **Determinism on events:** $\forall E_1, E_2 \subseteq E \; s.t. \; E_1 \cap E_2 \neq \emptyset$, $\forall l, l', l'' \in L$, *if* $l'' \xrightarrow{\gamma,E_1,U} l$ *and* $l'' \xrightarrow{\gamma',E_2,U'} l'$ *are two distinct transitions, then either* $\Lambda(l) \wedge \Lambda(l') \Leftrightarrow \texttt{false}$ *or* $\gamma \wedge \gamma' \Leftrightarrow \texttt{false}$[1].
- **Determinism on $\sharp$:**[2] $\forall l, l', l'' \in L$, *if* $l'' \xrightarrow{\gamma,\sharp,U} l$ *and* $l'' \xrightarrow{\gamma',\sharp,U'} l'$ *are two distinct transitions, then either* $\Lambda(l) \wedge \Lambda(l') \Leftrightarrow \texttt{false}$ *or* $\gamma \wedge \gamma' \Leftrightarrow \texttt{false}$[1].
- **No $\sharp$-labelled loops:** *For all sequences* $l_0 \xrightarrow{\gamma_0,E_0,U_0} l_1 \xrightarrow{\gamma_1,E_1,U_1} \cdots \xrightarrow{\gamma_{n-1},E_{n-1},U_{n-1}} l_n$ *such that* $l_0 = l_n$, *there exists* $i \leq n$ *such that* $E_i \neq \sharp$ [3].

Discussion. The motivation for the distinction between two types of edges in the LHA is that the transitions in the synchronized system (DESP + LHA) will be either autonomous, *i.e.* time-triggered (or rather variable-triggered) and take place as soon as a constraint is satisfied, or synchronized *i.e.* triggered by the DESP and take place when an event occurs in the DESP. The LHA will thus take into account the system behavior through synchronized transitions, but also take its own autonomous transitions in order to evaluate the desired property. In order to ensure that the first time instant at which a constraint is satisfied exists, we require for the constraints on autonomous transitions to be left closed. It should also be said that the restriction to linear equations in the constraints and to a linear evolution of data variables can be relaxed, as long as they are not involved in autonomous transitions. Polynomial evolution or constraints could easily be allowed for synchronised edges for which we would just need to evaluate the expression at a given time instant. Since the best algorithms solving polynomial equations operate in PSPACE [Can88], such an extension for autonomous transitions cannot be considered for obvious efficiency reasons.

[1] These equivalences must hold whatever the interpretation of the indicators occurring in $\Lambda(l)$, $\Lambda(l')$, γ and γ'.

[2] Note that our two notions of determinism allow an autonomous and a synchronised edges to be simultaneously fireable

[3] This condition is sufficient to avoid an infinite behavior of $\mathcal{A}$ without synchronization.

The automata we consider are deterministic in the following sense: given a path σ of a DESP, there is at most one synchronization with the linear hybrid automaton. This constraint ensures the synchronized system is still a stochastic process. In the above definition, the first three conditions ensure the uniqueness of the synchronization. First there is at most an initial location that can match with the initial state of the DESP. Then when two synchronized edges start from the same (current) location, (1) either their associated sets of events are disjoint, (2) either their guards are mutually exclusive (3) or their destination locations satisfy incompatible propositions. As there must be a matching with a state in the DESP this excludes non determinism. The third condition is similar for autonomous transitions. Since our semantic gives priority of autonomous transitions over synchronized ones, there is no need to introduce additional constraints.

At last, the fourth disables "divergence" of the synchronization, i.e. the possibility of an infinity of consecutive autonomous events without synchronization.

Notations. A *valuation* ν maps every data variable to a real value. The value of data variable x_i in ν is denoted $\nu(x_i)$. Let us fix a valuation ν and a state s. Given an expression $exp = \sum_{1 \leq i \leq n} \alpha_i x_i + c$ related to variables and indicators, its interpretation w.r.t. ν and s is defined by $exp(s, \nu) = \sum_{1 \leq i \leq n} \alpha_i(s)\nu(x_i) + c(s)$. Given an update $U = (u_1, \ldots, u_n)$, we denote by $U(s, \nu)$ the valuation defined by $U(s, \nu)(x_k) = u_k(s, \nu)$ for $1 \leq k \leq n$. Let $\gamma \equiv exp \prec 0$ be a constraint, we write $(s, \nu) \models \gamma$ if $exp(s, \nu) \prec 0$. Let φ be a state proposition we write $s \models \varphi$ if $\varphi(s) = \mathbf{true}$.

Semantics.

The role of a synchronized LHA is, given an execution of a corresponding DESP, to first decide whether the execution is to be accepted or not, and also to maintain data values along the execution.

Before defining the model associated with the synchronization of a DESP $\mathcal{D}$ and an LHA $\mathcal{A}$, we need to introduce a few notations to characterize the evolution of a synchronized LHA.

Given a state s of the DESP, a non final location l and a valuation ν of $\mathcal{A}$, we define the effect of time elapsing by:
$Elapse(s, l, \nu, \delta) = \nu'$ where, for every variable x_k, $\nu'(x_k) = \nu(x_k) + \mathit{flow}_k(l)(s) \times \delta$.

We also introduce the autonomous delay $Autdel(s, l, \nu)$ by:
$Autdel(s, l, \nu) = \min(\delta \mid \exists l \xrightarrow{\gamma, \sharp, U} l' \wedge s \models \Lambda(l') \wedge (s, Elapse(s, l, \nu, \delta)) \models \gamma)$

Whenever $Autdel(s, l, \nu)$ is finite, we know that there is at least one exe-

cutable transition with minimal delay and, thanks to the "determinism on $\sharp$" of definition 5.2, we know that this transition is unique. In the following we will denote $Next(s,l,\nu)$ the target location of this first transition and $Umin(s,l,\nu)$ its update.

We now proceed to the formal definition of the DESP $\mathcal{D}'$ associated with the synchronization of a DESP $\mathcal{D} = \langle S, \pi_0, E, Ind, enabled, target, delay, choice\rangle$ and an LHA $\mathcal{A} = \langle E, L, \Lambda, Init, Final, X, \mathit{flow}, \rightarrow\rangle$.

- $S' = (S \times L \times Val) \uplus \{\bot\}$
- $\pi'_0(s,l,\nu) = \begin{cases} \pi_0(s) & \text{if } (l \in Init \wedge s \models \Lambda(l) \wedge \nu = 0) \\ 0 & otherwise \end{cases}$
  and $\pi'_0(\bot) = 1 - \Sigma_{s\in S, l\in L, \nu\in Val}\pi'_0(s,l,\nu)$.
  Note that this definition gives a distribution since, due to "initial determinism" of definition 5.2, for every $s \in S$, there is at most one $l \in Init$ such that $s \models \Lambda(l)$.
- $E' = E \uplus \{\sharp\}$
- $Ind' = \emptyset$. In fact Ind' is useless since there is no more synchronization to make.
- $enabled'(s,l,\nu) = \{e \in enabled(s)\} \cup \{\sharp\}$ if $Autdel(s,l,\nu,) \neq \infty$,
  and $enabled'(s,l,\nu) = \{e \in enabled(s)\}$ otherwise.
- $delay'((s,l,\nu),e) = delay(s,e)$ for every $e \in enabled(s)$ and, whenever $\sharp \in enabled'(s,l,\nu)$, $delay'((s,l,\nu),\sharp)$ is a dirac function with a spike for the value $Autdel(s,l,\nu)$.
- $choice'((s,l,\nu),E',d)(e) = \begin{cases} 1 \text{ if } (\sharp \in E' \wedge e = \sharp) \\ 0 \text{ if } (\sharp \in E' \wedge e \neq \sharp) \\ 0 \text{ if } e \notin E' \\ choice(s,E',d)(e) \text{ otherwise} \end{cases}$
  Again this is coherent since, as soon as $\sharp \notin E'$, then E' is a subset of $enabled(s)$ on which $choice$ is thus defined.
- For a synchronized event,
  $target'((s,l,\nu),e,d) = (target(s,e,d),l',\nu')$ if $e \in enabled(s)$, $\exists l \xrightarrow{\gamma,E',U} l'$, $target(s,e,d) \models \Lambda(l')$,
  $Elapse(s,l,\nu,d) \models \gamma$, $e \in E'$ and $\nu' = U(Elapse(s,l,\nu,d))$.
  For an autonomous event, $target'((s,l,\nu),\sharp) = (s,l',\nu')$ if $\sharp \in enabled'(s,l,\nu)$ with $l' = Next(s,l,\nu)$ and $\nu' = Umin(s,l,\nu)(Elapse(s,l,\nu,Autdel(s,l,\nu)))$.

Note that, by initial determinism, for every $s \in S$ there is at most one $l \in Init$ such that s satisfies $\Lambda(l)$. Otherwise the synchronization starts and immediately ends up in state $\perp$. Observe that there are two possible behaviors for the synchronization. Either it ends up in some final state leading to a finite *synchronizing path* or the synchronization goes over all states of the path without ever reaching a final configuration. We discuss this point in the next subsection.

Example. The two LHA of figure 5.2 and 5.3 intend to illustrate the expressiveness of HASL's LHAs. They are meant to synchronize with the shared memory system of figure 5.1. The one in figure 5.2 uses two variables. x_0 has a null flow in every location and is used to count the number of memory accesses granted. x_1 expresses the difference of memory usage between processes of class 1 and 2. It has thus flow 1 (*resp.* -1) when the memory is used by class 1 (*resp.* 2) processes, and 0 when the memory is not used. As soon as k processes have been given a memory access, the system terminates in state l_3 or l_4 depending on which process type has used the memory for the longest cumulated period. In the figure, $\sharp$ labels for autonomous transitions are omitted and label E denotes synchronization with any event.

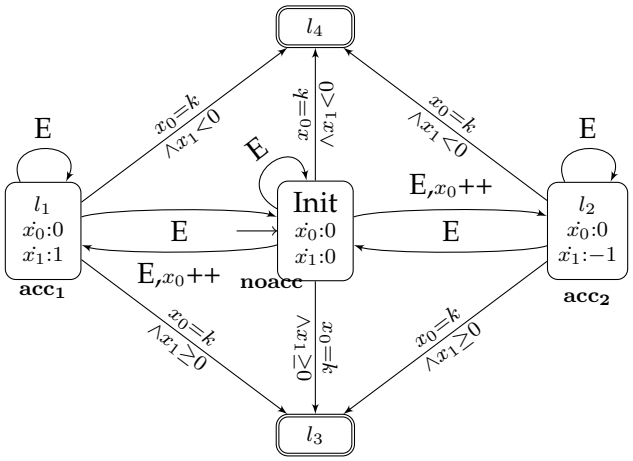


Figure 5.2: An LHA to compute the difference of memory usage

The example of figure 5.3 uses indicator dependent flows. x_1 counts the cumulated waiting time of processes of class 1 before k of them have been served. The flow $nbreq_1$ corresponds to the number of tokens in place $Request_1$ in the current marking whereas event $Serv_1$ corresponds to the firing of the bottom left transition of the SPN of figure 5.1. x_2 is the number of served processes of class 1 which is updated due to event $Serv_1$.

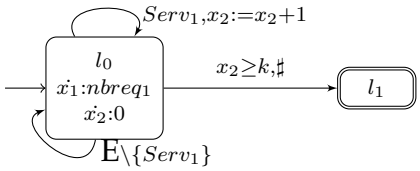


Figure 5.3: An LHA to compute the average waiting time

5.3.2 HASL expressions

The second component of a HASL formula is an expression related to the automaton. Such an expression, denoted Z, is based on moments of a path random variable Y and defined by the grammar:

$$\begin{aligned} Z &::= E(Y) \mid Z + Z \mid Z \times Z \\ Y &::= c \mid Y + Y \mid Y \times Y \mid Y/Y \mid last(y) \mid min(y) \\ &\quad \mid max(y) \mid int(y) \mid avg(y) \\ y &::= c \mid x \mid y + y \mid y \times y \mid y/y \end{aligned} \tag{5.1}$$

y is an arithmetic expression built on top of LHA data variables (x) and constants (c). Y is a path dependent expression built on top of basic path random variables such as $last(y)$ (i.e. the last value of y along a synchronizing path), $min(y)/max(y)$ (the minimum, resp. maximum, value of y along a synchronizing path), $int(y)$ (i.e. the integral over time along a path) and $avg(y)$ (the average value of y along a path). Finally Z, the actual target of HASL verification, is an arithmetic expression built on top of the first moment of Y ($E[Y]$), and thus allowing for the consideration of diverse significant characteristics of Y (apart from its expectation) as the quantity to be estimated, including, for example, $Var(Y) \equiv E[Y^2] - E[Y]^2$, $Covar(Y_1, Y_2) \equiv E[Y_1 \cdot Y_2] - E[Y_1] \cdot E[Y_2]$. Note that for efficiency reasons, in the implementation of the software tool, we have considered a restricted version of grammar (5.1), where products and quotients of data variables (e.g. $x_1 \cdot x_2$ and x_1/x_2) are allowed only within the scope of the $last$ operator (*i.e.* not with min, max, int or avg). This is because allowing products and quotients as arguments of path operators such as max, min requires the solution of a linear programming problem during the generation of a synchronized $\mathcal{D} \times \mathcal{A}$ path which, although feasible, would considerably affect the computation time.

Semantics. Given $\mathcal{D}$ a DESP and $(\mathcal{A}, Z)$ a HASL formula, we assume that with probability 1, the synchronizing path generated by a random execution path of $\mathcal{D}$ reaches a final state. This semantical assumption can be ensured by structural properties of $\mathcal{A}$ and/or $\mathcal{D}$. For instance the time bounded `Until`

of CSL guarantees this property. As a second example, the time unbounded `Until` of CSL also guarantees this property when applied on finite CTMCs where all terminal strongly connected components of the chain include a state that fulfills the target subformula of the `Until` operator. This (still open) issue is also addressed in [SVA05b, HJB+10]. Due to this assumption, the random path variables are well defined and the expression Z associated with the formula may be evaluated with expectations defined w.r.t. the distribution of a random path *conditioned by acceptance of the path*. In other words, the LHA both calculates the relevant measures during the execution and selects the relevant executions for computing the expectations. This evaluation gives the result of the formula $(\mathcal{A}, Z)$ for $\mathcal{D}$.

Example. Referring to the LHA of figure 5.2, we can consider path random variables such as $Y = last(x_1)$ (the final difference of memory usage), or $Y = avg(x_1)$ (the average along paths of such a difference). Furthermore, with a slight change of the automaton (setting x_0 to 0 (*resp.* 1) when reaching l_4 (*resp.* l_3)), $E(last(x_0))$ will give the probability to reach l_3. With the LHA of figure 5.3, we can express (an overestimation of) the average waiting time by means of $Y = last(x_1/x_2)$. An underestimation can also be computed counting the overall number of requesting processes instead of the number of served processes.

5.3.3 Expressiveness of HASL

In this subsection we first give an overview of related logics. Then we discuss the expressiveness of HASL and show how it improves the existing offer to capture more complex examples and properties, and facilitates the expression and the computation of costs and rewards.

CSL. In [ASVB00] the logic Continuous Stochastic Logic (CSL) has been introduced and the decidability of the verification problem over a finite continuous-time Markov chain (CTMC) has been established. CSL extends the *branching time* reasoning of CTL to CTMC models by replacing the discrete CTL path-quantifiers `All` and `Exists` with a continuous path-quantifier $P_{\prec r}$ ($\prec \in \{<, \leq, \geq, >\}$, $r \in [0, 1]$). Thus a CSL formula $P_{\prec r}\varphi$ expresses that the probability of CTMC paths satisfying condition φ fulfills the bound $\prec r$, where φ is, typically, a time-bounded `Until` formula. In [BHHK03a] it has been demonstrated that the evaluation of the probability measure of a (time-bounded) CSL specification corresponds to the transient analysis of a (modified) CTMC, for which efficient approximate numerical procedures exist.

CSRL. In the logic CSRL introduced by [BHHK00a], CSL is extended to take into account Markov reward models, i.e. CTMCs with a single reward on states. The global reward of a path execution is then the integral of the instantaneous reward over time. In CSRL, the path operators `Until` and `Next` include also an interval specifying the allowed values for accumulated reward. Moreover new operators related to the expectation of rewards are defined. A numerical approach is still possible for approximating probability measures but its complexity is significantly increased. This formalism is also extended by rewards associated with actions [CKKP05]. CSRL is appropriate for standard performability measures but lacks expressiveness for more complex ones.

asCSL. In the logic asCSL introduced by [BCH$^+$07], the single interval time constrained `Until` of CSL is replaced by a regular expression with a time interval constraint. These path formulas can now express elaborated functional requirements as in CTL* but the timing requirements are still limited to a single interval globally constraining the path execution.

CSL^{TA}. In the logic CSL$^{\text{TA}}$ introduced by [DHS09], the path formulas are defined by a single-clock deterministic time automaton. This clock can express timing requirements all along the path. From an expressiveness point of view, it has been shown that CSL$^{\text{TA}}$ is strictly more expressive than CSL and that path formulas of CSL$^{\text{TA}}$ are strictly more expressive than those of asCSL. Finally, the verification procedure is reduced to a reachability probability in a semi-Markovian process yielding an efficient numerical procedure.

DTA. In [CHKM09], deterministic timed automata with multiple clocks are considered and the probability for random paths of a CTMC to satisfy a formula is shown to be the least solution of a system of integral equations. In order to exploit this theoretical result, a procedure for approximating this probability is designed based on a system of partial differential equations.

Observe that all of the above mentioned logics have been designed so that numerical methods can be employed to decide about the probability measure of a formula. This very constraint is at the basis of their limited expressive scope which has two aspects: first the targeted stochastic models are necessarily CTMCs; second the expressiveness of formulas is constrained (even with DTA [CHKM09], the most expressive among the logic for CTMC verification, properties of a model can be expressed only by means of clocks variables, while sophisticated measures corresponding to variables with real-valued rates cannot be considered). Furthermore observe that the evolution of stochastic logics seems to have followed two directions: one targeting *temporal reasoning* capability (evolutionary path: CSL $\rightarrow$ asCSL $\rightarrow$ CSL$^{\text{TA}}\rightarrow$ DTA), the other targeting *performance evaluation* capability (evolutionary path: CSRL $\rightarrow$ CSRL+impulse

rewards). A unifying approach is currently not available, thus, for example, one can calculate the probability of a CTMC to satisfy a sophisticated temporal condition expressed with a DTA, but cannot, assess performance evaluation queries at the same time (i.e. with the same formalism).

HASL: a unifying approach. As HASL is inherently based on simulation for assessing measures of a model, it naturally allows for releasing the constraints imposed by logics that rely on numerical solution of stochastic models. From a modeling point of view HASL allows for targeting of a broad class of stochastic models (i.e. DESP), which includes, but is not limited to, CTMCs. From an expressiveness point of view the use of LHA allows for generic variables, which include, but are not limited to, clock variables (as per DTA). This means that sophisticated temporal conditions as well as elaborate performance measures of a model can be accounted for in a single HASL formula, rendering HASL a unified framework both for model-checking and for performance and dependability studies. Note that the nature of the (real-valued) expression Z (5.1) (characterizing the outcome of a HASL formula) generalizes the common approach of stochastic model checking where the outcome of verification is (an approximation of) the mean value of a certain measure (with CSL, asCSL, CSL^{TA} and DTA a measure of probability). *Cost functions.* It is also worth noting that the use of data variables and extended updates in the LHA enables to compute costs/rewards naturally. The rewards can be both on locations and on actions. First using an appropriate flow in each location of the LHA, possibly depending on the current state of the DESP we get "state rewards". Then by considering the *update expressions* on the edges of the LHA we can model sophisticated "action rewards" that can either be a constant, depend on the state of the DESP and/or depend on the values of the variables. It thus extends the possibilities of CSRL and its extensions [CKKP05] where only one reward function (on states and actions) is considered.

Finally we briefly discuss on the issue of nesting of probabilistic operators. Nesting of probabilistic operators, which is present in all stochastic logics discussed above, is meaningful only when an identification can be made between a state of the probabilistic system and a configuration (comprising the current time and the next scheduled events). Whereas this identification was natural for Markov chains, it is not possible with DESP and general distributions, and therefore this operation has not been considered in HASL. A similar problem arises for the steady state operator. The existence of a steady state distribution raises theoretical problems, except for finite Markov chains, but with HASL we allow for not only infinite state systems but also non Markovian behaviors. However, when the DESP has a regeneration point, various steady state properties can be computed by defining the regeneration point as a final state. For the

expressiveness of HASL, we can state that when we omit nesting and steady state properties, HASL is at least as expressive as CSRL and DTA : *Every non nested transient CSRL or DTA formula can be expressed with HASL.*

5.4 Conclusion

We have presented a new logic for expressing elaborated properties related to stochastic processes. Contrary to previous approaches, a formula of HASL returns a conditional expectation whose condition is based on acceptance by a linear hybrid automaton. Such a logic can be employed both for probabilistic validation of functional properties or for elaborated performance analysis as we have illustrated with examples. In the next chapter, we present the tool that we have developed for evaluation of HASL formulas over stochastic Petri nets.

Chapter 6

Cosmos

This chapter is related to the publication [BDD+11a].

6.1 Introduction

In order to implement a statistical model checker, different components must be developed.

- The first one is a simulator. The design of such a tool raises two different issues. There are two ways to simulate a trajectory of a formal model. On the one hand, one may design an *interpretive program* that takes as input the model and applies the semantic rules in order to generate the trajectories. On the other hand, one may generate an *execution code* that corresponds to the dynamic of the model. While the former choice is relatively easy to develop, the latter provides significantly better time performance. The second issue is related to the satisfaction of the formula by the trajectory. It can be done *offline* once the trajectory has been generated but even with efficient algorithms this increases significantly the time complexity. However performing it *online* requires (as proposed in the previous chapter) to consider that the system to be simulated is a synchronized product of the system and an automaton corresponding to the formula.
- The second component monitors the generation of the paths by stopping it when some criteria have been fulfilled. It can be: a maximal number of paths, the reaching of the confidence level associated with an interval width, etc. Generally the simplest way to monitor the simulation consists to launch some usual statistical tool.

- The third component is the parser associated with the textual description of the formal model and the formula. It is rather standard and several tools support the quick development of a parser.

The first tools that have been developed are are Ymer [You05] and Vesta [SVA05b]. They mainly differ by the way they monitor the number of paths to be generated. A detailed comparison between these tools can be found in [JKO+07]. Then classical tools have been enlarged with statistical model-checking features like PRISM [KNP11] and UPPAAL [DLL+11].

In this chapter we present COSMOS, the tool that we have developed to implement the statistical model checking of HASL formulas for models specified with Petri nets. In french, COSMOS is the acronym of: "***C**oncepts et **O**utils **S**tatistiques pour les **M**odèles **S**tochastiques* " which means statistical concepts and tools for stochastic models. We start by presenting the development environment and the main interface. Then, we present the structure and the main algorithms used in the implementation. After that, the syntax of the input files (Petri net and HASL files) is described. We also illustrate the application of COSMOS with some case studies showing the efficiency of COSMOS in terms of performance (time and memory) and the expressiveness and adequacy of HASL.

6.2 Interface and Syntax

6.2.1 Interface

The source code of COSMOS can be partitioned into three parts:

- The parsing part which consists of parser generators written in Bison and lexical analyzers written in Flex. It is about 2300 lines. The files generated by bison and flex are about 15500 lines.

- The code generation part. It is about 2700 lines. This part generates a C++ code for the input files (HASL and Petri net classes).

- The simulator part. It consists of fixed files (about 2400 lines) and variables files (HASL and Petri net classes, their sizes depend on the input files).

Installation The source code of COSMOS is completely written in the C++ programming language. Its compilation requires:

- The GNU Compiler Collection (GCC) [GCC], more precisely, the C++ GNU compiler.
- The GNU make utility [GNU].
- The Boost C++ libraries [BOO], which are used to manage the random number generation.

The current version of COSMOS runs under Linux and MAC-OS operating systems.

Execution. The execution of COSMOS requires two files as inputs: filename.gspn which describes the generalized stochastic Petri net and filename.lha which describes the HASL formula (an automaton and a set of expressions). The grammars are given in the next subsection. The generator of COSMOS parses the inputs and produces a C++ code of two classes: the Petri net class and the HASL class. The C++ compiler compiles these files and links them with other objects files (which were generated during the installation) to produce the executable file of COSMOS.

Figure 6.1 summarizes the execution of COSMOS.

Since it is easier to define the input Petri net by a graphical interface, as does GreatSPN [CFGR95, Gre], we added an option to DSPN [AD10, DSP] that allows to export nets which were drawn in GreatSPN to the input format of COSMOS.

We plan to specify our nets and automata in Coloane [COL], a generic graph editor.

Let us list the main commands of COSMOS.

- ***sim***: This is the main command. It launches the simulation.
- ***width***: It specifies the confidence interval width.
- ***level***: It specifies the confidence level.
- ***batch***: It specifies the batch size.
- ***maxpaths***: It specifies the maximum number of paths to generate.

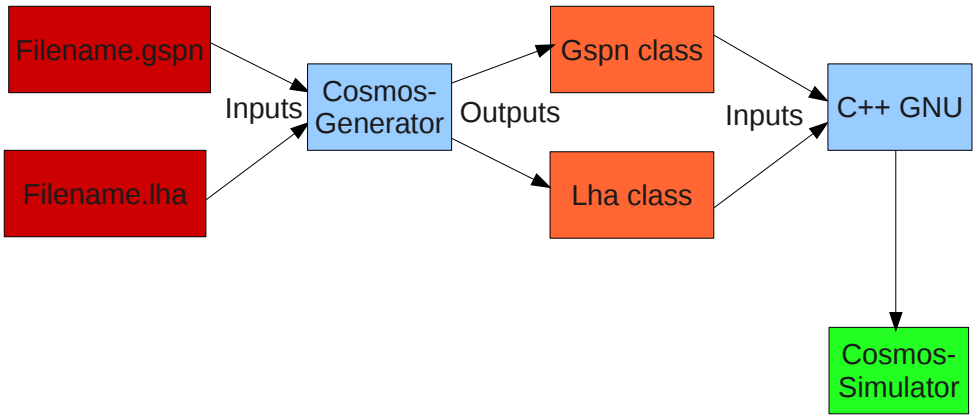


Figure 6.1: From input files to simulation.

- ***params***: It displays the current value of parameters.
- ***help***: It displays the commands.
- ***stop***: It exits COSMOS.

The results of a COSMOS execution are summarized in an output file named *filename.res*. It contains the following informations.

- a point estimation of each expression;
- the sample standard deviation;
- the number of generated trajectories and the number of successful trajectories;
- a confidence interval for each expression;
- the execution time.

6.2.2 Syntax

We first describe the syntax of some common symbols before giving the syntax of each input file.

A natural number <Integer>, a real number <Real> or string type <Str> are defined like this:

⟨Integer⟩ ::= [0-9]+| [0-9]*
⟨Real⟩ ::= ([0-9]+| [0-9]*[0-9]+)([eE][-+]?[0-9]+)?
⟨Str⟩ ::= [a-zA-Z][a-zA-Z_0-9]*

All the symbols finishing by "**Tag**" refer to a tag of an object. These symbols are string type:

- <IConstTag>: A tag of a natural number constant.
- <RConstTag>: A tag of a real constant.
- <PTag>: A tag of a Petri net place.
- <TTag>: A tag of a Petri net transition.
- <LTag>: A tag of an automaton location.
- <VTag>: A tag of an automaton variable.

It is useful to define some integer constants <IConstant> or real constants <RConstant> which can be used by other definitions:

⟨IConstant⟩ ::= "const" "int" ⟨IConstTag⟩ "=" ⟨Integer⟩ ";"
⟨RConstant⟩ ::= "const" "double" ⟨RConstTag⟩ "=" ⟨Real⟩ ";"

Some numerical attributes (marking values, transitions parameters, arcs multiplicity, variables rate, etc.) may be introduced as a function of numerical values (real and/or integer), constants and/or Petri net places. Let us give the grammar of such functions.

The first kind of functions is <RFormula> for real formula. It includes numerical values (integer or real) and constants (integer or/and real).

⟨RFormula⟩ ::= ⟨Real⟩ | ⟨RConstTag⟩ | ⟨IFormula⟩ | ⟨RFormula⟩ ⟨ArOP⟩ ⟨RFormula⟩ | "(" ⟨RFormula⟩ ")"

The second kind is <IFormula> for integer formula. It includes numerical values (integer or real) and constants (integer or/and real) but its value should always be a natural number.

⟨IFormula⟩ ::= ⟨Integer⟩ | ⟨IConstTag⟩ | ⟨IFormula⟩ ⟨ArOpRes⟩ ⟨IFormula⟩ | "(" ⟨IFormula⟩ ")" | "floor" "(" ⟨RFormula⟩ ")"

The third kind of functions is <MRFormula> for marking real formula. It includes numerical values (integer or real), constants (integer or/and real) and Petri places.

⟨MRFormula⟩ ::= ⟨PTag⟩ | ⟨RFormula⟩ | ⟨MRFormula⟩ ⟨ArOp⟩ ⟨MRFormula⟩ | "(" ⟨MRFormula⟩ ")"

The last type of functions is <MIFormula> for marking integer formula. It includes numerical values (integer or real), constants (integer or/and real) and Petri places but its value should be always a natural number.

⟨MIFormula⟩ ::= ⟨PTag⟩ | ⟨IFormula⟩ | ⟨MIFormula⟩ ⟨ArOpRes⟩ ⟨MIFormula⟩ | "(" ⟨MIFormula⟩ ")" | "floor" "(" ⟨RFormula⟩ ")"

These functions are defined with the following set of arithmetic operators called <ArOp> or its restricted version <ArOpRes>:

⟨ArOp⟩ ::= "+" | "-" | "*" | "/" | "^"
⟨ArOpRes⟩ ::= "+" | "-" | "*" | "^"

GSPN Syntax. The definition of the Petri net consists of:

⟨GSPN⟩ ::= {⟨IConstant⟩ } {⟨RConstant⟩ } ⟨NT⟩ ⟨NP⟩ ⟨PList⟩ ⟨TList⟩ ⟨InitMarking⟩ ⟨TransitionsDef⟩ [⟨InArcs⟩] [⟨OutArcs⟩] [⟨InhibArcs⟩]

In the first part, some integer and/or real constants can be declared. The size of the Petri net (number of transitions and places) are declared.

⟨IConstant⟩ ::= "const" "int" ⟨IConstTag⟩ "=" ⟨Integer⟩ ";"
⟨RConstant⟩ ::= "const" "double" ⟨RConstTag⟩ "=" ⟨Real⟩ ";"
⟨NT⟩ ::= "NbTransitions" "=" ⟨Integer⟩ ";" | "NbTransitions" "="
⟨IFormula⟩ ";"
⟨NP⟩ ::= "NbPlaces" "=" ⟨Integer⟩ ";" | "NbPlaces" "=" ⟨IFormula⟩
";"

Then, the set of transitions and places are defined.

⟨PList⟩ ::= "PlacesList" "=" "{" ⟨PTags⟩ "}" ";"
⟨PTags⟩ ::= ⟨PTag⟩ | ⟨PTags⟩ "," ⟨PTag⟩
⟨TList⟩ ::= "TransitionsList" "=" "{" ⟨TTags⟩ "}" ";"
⟨TTags⟩ ::= ⟨TTag⟩ | ⟨TTags⟩ "," ⟨TTag⟩

After that, the initial marking is given. By default, all places contain zero token.

⟨InitMarking⟩ ::= "Marking" "=" "{" ⟨Inits⟩ "}"
⟨Inits⟩ ::= ⟨Init⟩ | ⟨Init⟩ "," ⟨Inits⟩
⟨Init⟩ ::= "(" ⟨PTag⟩ "," IFormula ")"

The next step consists of a full description of the transitions. Observe that transitions which are exponentially distributed, are defined differently from those with other distributions.

⟨TransitionsDef⟩ ::= "Transitions" "=" "{" ⟨Transitions⟩ "}" ";"
⟨Transitions⟩ ::= ⟨Transition ⟩ | ⟨Transitions⟩ "," ⟨Transition⟩
⟨Transition⟩ ::= ⟨Exp⟩ | ⟨NonExp⟩

The memory and server policies have the following syntax.

<Service> :
“SINGLE” ;

“INFINITE” ;
“MULTIPLE” “(”<integer>“)”.
<Memory> :
“ENABLEDMEMORY” ;
“AGEMEMORY”.

A transition with an exponential distribution can have a marking dependent parameter. A priority and a weight are also given. Then a memory policy and a service are chosen:

⟨Exp⟩ ::= "(" ⟨TTag⟩ "," "EXPONENTIAL" "(" ⟨MRFormula⟩ ")" "," ⟨Priority⟩ "," ⟨Weight⟩ "," ⟨Memory⟩ "," ⟨Service⟩ ")"

In COSMOS, a transition with non exponential distribution cannot have marking dependent parameters. A priority and weight are also given. Then a memory policy is chosen The service policy is forced: it is single server.

⟨NonExp⟩ ::= "(" ⟨TTag⟩ "," ⟨Dist⟩ "," ⟨Priority⟩ "," ⟨Weight⟩ "," ⟨Memory⟩ ")"
⟨Dist⟩ ::= "IMMEDIATE" | "DETERMINISTIC" "(" ⟨Real⟩ ")" | "UNIFORM" "(" ⟨Real⟩ "," ⟨Real⟩ ")" "ERLANG" "(" ⟨Integer⟩ "," ⟨Real⟩ ")" | "GAMMA" "(" ⟨Real⟩ "," ⟨Real⟩ ")" | "TRIANGLE" "(" ⟨Real⟩ "," ⟨Real⟩ "," ⟨Real⟩ ")" | "GEOMETRIC" "(" ⟨Real⟩ "," ⟨Real⟩ ")" | "LOGNORMAL" "(" ⟨Real⟩ "," ⟨Real⟩ ")"

The final part consists of introducing the different matrices of the net. Observe that the arc multiplicities can be marking dependent.

⟨In⟩ ::= "InArcs" "=" "{" ⟨InArcs⟩ "}" ";"
⟨InArcs⟩ ::= ⟨InArc⟩ | ⟨InArcs⟩ "," ⟨InArcs⟩
⟨InArc⟩ ::= "(" ⟨PTag⟩ "," ⟨TTag⟩ ")" | "(" ⟨PTag⟩ "," ⟨TTag⟩ "," ⟨MIFormula⟩ ")"
⟨Out⟩ ::= "OutArcs" "=" "{" ⟨OutArcs⟩ "}" ";"
⟨OutArcs⟩ ::= ⟨OutArc⟩ | ⟨OutArcs⟩ "," ⟨OutArcs⟩
⟨OutArc⟩ ::= "(" ⟨TTag⟩ "," ⟨PTag⟩ ")" | "(" ⟨TTag⟩ "," ⟨PTag⟩ "," ⟨MIFormula⟩ ")"
⟨Inhib⟩ ::= "InhibArcs" "=" "{" ⟨InhibArcs⟩ "}" ";"

⟨InhibArcs⟩ ::= ⟨InhibArc⟩ | ⟨InhibArcs⟩ "," ⟨InhibArcs⟩
⟨InhibArc⟩ ::= "(" ⟨PTag⟩ "," ⟨TTag⟩ ")" | "(" ⟨PTag⟩ "," ⟨TTag⟩ "," ⟨MIFormula⟩ ")"

HASL Syntax. The definition of the HASL formula consists of:

⟨HASL⟩ ::= {⟨IConstant⟩ } {⟨RConstant⟩ } ⟨NL⟩ ⟨NV⟩ ⟨LList⟩ ⟨VList⟩ ⟨Expression⟩ ⟨InitLoc⟩ ⟨FinalLoc⟩ ⟨LocDef⟩ [⟨Edges⟩]

In the first part some constants can be declared. Then the number of locations and variables are defined.

⟨IConstant⟩ ::= "const" "int" ⟨IConstTag⟩ "=" ⟨Integer⟩ ";"
⟨RConstant⟩ ::= "const" "double" ⟨RConstTag⟩ "=" ⟨Real⟩ ";"
⟨NL⟩ ::= "NbLocations" "=" ⟨Integer⟩ ";" | "NbLocations" "=" ⟨IFormula⟩ ";"
⟨NV⟩ ::= "NbVariables" "=" ⟨Integer⟩ ";" | "NbVariables" "=" ⟨IFormula⟩ ";"

Afterward, locations and variables are declared:

⟨LList⟩ ::= "LocationsList" "=" "{" ⟨LTags⟩ "}" ";"
⟨LTags⟩ ::= ⟨LTag⟩ | ⟨LTags⟩ "," ⟨LTag⟩
⟨VList⟩ ::= "VariablesList" "=" "{" ⟨VTags⟩ "}" ";"
⟨VTags⟩ ::= ⟨VTag⟩ | ⟨VTags⟩ "," ⟨VTag⟩

Then the HASL expression is introduced:

⟨HAslExp⟩ ::= {⟨ExpectExp⟩ ";" }
⟨ExpectExp⟩ ::= "AVG" "(" ⟨F⟩ ")" | ⟨ExpectExp⟩ ⟨ArOp⟩ ⟨ExpectExp⟩
⟨F⟩ ::= ⟨H⟩ | ⟨F⟩ "/" ⟨RFormula⟩ | ⟨F⟩ "*" ⟨RFormula⟩ | ⟨F⟩ ⟨ArOp⟩ ⟨F⟩ | "min" "(" ⟨F⟩ "," ⟨F⟩ ")" | "max" "(" ⟨F⟩ "," ⟨F⟩ ")"
⟨H⟩ ::= "Last" "(" ⟨LX⟩ ")" | "Min" "(" ⟨LX⟩ ")" | "Max" "(" ⟨LX⟩ ")" | "Integral" "(" ⟨LX⟩ ")" | "Mean" "(" ⟨LX⟩ ")" | "Var" "(" ⟨LX⟩ ")"
⟨LX⟩ ::= ⟨term⟩ | ⟨term⟩ "+" ⟨term⟩ | ⟨term⟩ "-" ⟨term⟩

⟨term⟩ ::= ⟨VTag⟩ | ⟨Real⟩ "*" ⟨VTag⟩ | "(" ⟨MRFormula⟩ ")" "*" ⟨VTag⟩

Initial and final locations are declared:

⟨InitLoc⟩ ::= "InitialLocations" "=" "{" ⟨LTags⟩ "}" ";"
⟨FinalLoc⟩ ::= "FinalLocations" "=" "{" ⟨LTags⟩ "}" ";"

The locations are fully described. Each location is tagged with <LTag> and satisfies a property on the marking of the Petri net. At each location, the rates of the variables are given. By default, rates are set to zero.

⟨LocDef⟩ ::= "Locations" "=" "{" ⟨Ldefs⟩ "}" ";"
⟨Ldefs⟩ ::= ⟨Ldef⟩ | ⟨Ldefs⟩ "," ⟨Ldef⟩
⟨Ldef⟩ ::= "(" ⟨LTag⟩ "," ⟨MLFormula⟩ "," "(" ⟨Vrates⟩ ")" ")"

⟨MLFormula⟩ ::= "TRUE" | ⟨MRFormula⟩ ⟨CompOp⟩ ⟨MRFormula⟩
| ⟨MLFormula⟩ ⟨LogOp⟩ ⟨MLFormula⟩ | "!" "(" ⟨MLFormula⟩ ")"
⟨CompOp⟩ ::= "=" | ">" | "<" | ">=" | "<="
⟨LogOp⟩ ::= "&" | "|"

⟨Vrates⟩ ::= ⟨Vrate⟩ | ⟨Vrates⟩ "," ⟨Vrate⟩
⟨Vrate⟩ ::= ⟨VTag⟩ ":" MRFormula

Finally, the edges are defined. An edge connects a location source to a location target (<LTag> , <LTag>). Each edge is associated with a set of Petri net transitions <Actions>. If the edge is synchronized with all Petri transitions then <Actions> takes value "ALL". If the edge is not synchronized with the Petri net (i.e an autonomous edge) then <Actions> takes value "#". Each edge is associated with a set of linear constraints on automaton variable <Constraints>. If the edge is not subject to any constraint then <Constraints> takes value "#". Each edge is also associated with a set of variable updates <Ups>. If no update is required then <Ups> takes value "#".

⟨Edges⟩ ::= "Edges" "=" "{" ⟨Edefs⟩ "}" ";"
⟨Edefs⟩ ::= ⟨Edef⟩ | ⟨Edefs⟩ "," ⟨Edef⟩
⟨Edef⟩ ::= "(" "(" ⟨LTag⟩ "," ⟨LTag⟩ ")" "," ⟨Actions⟩ "," ⟨Constraints⟩

"," ⟨Updates⟩ ")"
⟨Actions⟩ ::= "#" | "{" ⟨PTags⟩ "}" | "ALL" "
" "{" ⟨PTags⟩ "}"
⟨Constraints⟩ ::= "#" | Constraint | Constraint "&" Constraints
⟨Constraint⟩ ::= ⟨LX⟩ "=" ⟨MRFormula⟩ | ⟨LX⟩ ">=" ⟨MRFormula⟩
| ⟨LX⟩ "<=" ⟨MRFormula⟩
⟨Updates⟩ ::= "{" ⟨Ups⟩ "}" ";"
⟨Ups⟩ ::= ⟨Up⟩ "," ⟨Ups⟩
⟨Up⟩ ::= ⟨VTag⟩ "=" ⟨VMRFormula⟩ | ⟨VMRFormula⟩ ::= ⟨VTag⟩
| ⟨MRFormula⟩ | ⟨VMRFormula⟩ ⟨ArOp⟩ ⟨VMRFormula⟩

6.3 Algorithms

In this section we describe the most important algorithms used by COSMOS. We start by the main algorithm in subsection 6.3.1. Then we describe how a single trajectory of the synchronized product $\mathcal{N} \times \mathcal{A}$ is generated in subsection 6.3.2. In subsection 6.3.3 we show how the event queue is managed. In subsection 6.3.4 we describe the Petri net class. In subsection 6.3.5 the automaton class is described and finally in subsection 6.3.6 we show how the HASL expression is updated.

6.3.1 Main Algorithm

Assume that we want to estimate a HASL formula:

$$(\mathcal{A}, H = AVG\left(F\left(f_1\left(L_1\left(X\right)\right), ..., f_n\left(L_n\left(X\right)\right)\right)\right))$$

on a Generalized Stochastic Petri Net $\mathcal{N}$ with respect to the statistical parameters $SP = (width, 1-\alpha, batch, maxpaths)$.

The main algorithm of COSMOS is described by algorithm 6.1. It consists of two nested *while* loops.

The first loop (line 3) launches the second loop and updates the current confidence interval width (w) (line 9) by using the the current variance (Var), the number of successful trajectories ($Ksucc$) and the normal percentile Z. The loop stops if the current width becomes less than the desired width or if the total number of generated trajectories (K) is greater than the allowed number.

The second loop (line 5) generates a successful batch of trajectories. At each iteration, it launches the procedure **SimulateSinglePath()** (see algorithm 6.2) which generates a trajectory. If the generated trajectory is not a successful one, it is omitted otherwise its value (val) is used in procedure $UpdateStatistics$ to update the expectation and the variance of H. The loop stops when it generates a batch of successful trajectories or when the total number of allowed trajectories is exceeded.

Observe that procedure $\mathbf{NormalPercentile}(\mathbf{1}-\alpha/\mathbf{2})$ computes the normal percentile with a given threshold α.

Algorithm 6.1: RunSimulation()

Data: $\mathcal{N}$; $\mathcal{A}$; H and SP
Result: $\hat{H}$ and $CI\left(\hat{H}\right)$

```
1  begin
2      Ĥ = 0;  K = 0;  Ksucc = 0;  w = ∞;  Z =
       NormalPercentile(1 − α/2)
3      while (w > width and K < maxpaths) do
4          i = 0
5          while (i < batch and K < maxpaths) do
6              (success, val) = SimulateSinglePath();  K = K + 1
7              if (success) then
8                  Ksucc = Ksucc + 1;  i = i + 1;
                   UpdateStatistics(Ĥ, val, Ksucc)
9          UpdateWidth(Var, Ksucc, Z)
10     return Ĥ and CI(Ĥ)
```

6.3.2 Single Path Generation

Now we describe how a trajectory of the synchronized product $\mathcal{N} \times \mathcal{A}$ is generated. This is done by algorithm 6.2. During its execution the algorithm essentially maintains in the memory the simulation time ($SimTime$), the Petri net marking ($Mark$), the queue of scheduled events ($Queue$), the automaton location (loc), the values of the automaton variables and the values of the functions occurring in the HASL formula.

It consists of an initialization phase (from line 2 to line 4) and a *while* loop (line 5) In the initialization phase, $SimeTime$ is set to zero, $Mark$ is initialized

to the initial marking of the Petri net, function $\mathcal{A}.EnabledInitLocation(Mark)$ selects the current location among the initial locations of the automaton according to the current marking of the Petri net. Procedure $GenerateInitialQueue()$ builds the initial queue. Function $\mathcal{A}.GetEnabled\_A\_Edge$ determines the enabled autonomous edge AE.

Algorithm 6.2: SimulateSinglePath()

Data: $\mathcal{N}$; $\mathcal{A}$; $F\left(f_1\left(L_1\left(X\right)\right), ..., f_n\left(L_n\left(X\right)\right)\right)$
Result: $success \quad and \quad val$

```
1  begin
2      SimTime = 0;  Mark = N.InitMarking;  Loc =
       A.EnabledInitLocation(Mark)
3      GenerateInitialQueue()
4      AE = A.GetEnabled_A_Edge(SimTime, Loc, Mark)
5      while (¬(Queue.IsEmpty()) or AE.index > −1) do
6          Event = Queue.top()
7          while (Event.time ≥ AE.FiringTime and AE.index > −1) do
8              A_Exp.Update(AE)
9              AE = A.GetEnabled_A_Edge(SimTime, Loc, Mark)
10         if (Event == NullEvent) then
11             SE.index = −1
12         else
13             OldMark = Mark;  N.Fire(Event.index)
14             Δt = Event.time − SimTime
15             SE =
               A.GetEnabled_S_Edge(SimTime, Event, Loc, OldMark, Mark)
16         if (SE.index < 0) then
17             return (false, val)
18         else
19             A_Exp.Update(SE)
20             Queue.Update(Event.index)
21         AE = A.GetEnabled_A_Edge(SimTime, Loc, Mark)
22     return (false, val)
```

The while loop (line 5) run if at least one the following conditions holds:

- The queue is not empty.

- There exists an enabled autonomous edge ($AE.index > -1$).

At line 6, the first event of the queue is selected.

The *while* (line 7) represents the situation where the enabled autonomous edge is scheduled before the selected event or the situation where an enabled autonomous edge exists but the queue is empty. At each iteration of the loop, the function $\mathcal{A}\_Exp.Update(AE)$ is called (see algorithm 6.3) and a new enabled autonomous edge is determined. The loop continues running until the new autonomous edge is scheduled after the selected event or there is no enabled autonomous edge.

If the queue is empty we conclude that there is no enabled synchronized edge (line 10). Otherwise, the Petri net transition corresponding to the event is fired (line 13) and function $\mathcal{A}.GetEnabled\_S\_Edge(...)$ (line 14) determines the enabled synchronized edge SE. If there is no enabled synchronized edge (line 16) then the algorithm terminates and the trajectory is unsuccessful (line 17). Otherwise function $\mathcal{A}\_Exp.Update(SE)$ is called again (line 19) and the event queue is updated according to the fired transition and the resulting marking (line 20, see algorithm 6.4).

At line 21, a new autonomous edge is determined.

If initially, the queue is empty and there is no enabled autonomous edge the algorithm does not enter the first *while* loop and terminates from line 21 with an unsuccessful trajectory.

The next algorithm updates the configuration of the automaton and the expression to evaluate. It is called by algorithm 6.2 when an autonomous or a synchronized edge ED is fired.

At line 3, the values of the variables are updated. Then the values of the HASL function are updated (line 4). After that, the simulation time is updated (line 5). The values of the variables may be updated again if there is some update function associated with the fired edge (line 6). At line 7, the current location of the automaton is updated, its new value is simply the target of the fired edge. Finally, if the new location is final (line 8), then a final update of the HASL functions is processed (line 9), the value of $F()$ is computed and the algorithm terminates with a successful trajectory.

Table 6.1 summarizes the meaning of variables and procedures used in algorithms 6.2 and 6.3

Algorithm 6.3: $\mathcal{A}\_Exp.Update()$

Data: an edge ED
Result: Update automaton and expressions values

```
1  begin
2      Δt = ED.FiringTime − SimTime
3      VarUpdates(SimTime, Δt, Loc, Mark)
4      FuncUpdates(SimTime, Δt, Mark)
5      SimTime = ED.FiringTime
6      EdgeUpdates(ED.index)
7      Loc = Edge[ED.index].target
8      if (A.IsFinal(Loc)) then
9          FuncUpdates(SimTime, 0, Mark)
10         val = F()
11         return (true, val)
```

6.3.3 Events Queue Management

The events queue is encoded as a binary min-heap structure. An event e in the heap is a tuple: $(index, time, priority, w)$ where:

- $index \in \mathbb{N}$ is the index of the associated Petri net transition.
- $time \in \mathbb{R}_+$ is the scheduled date of the transition firing. It is generated according to the delay distribution of the transition.
- $priority \in \mathbb{N}$ is the priority of the associated transition.
- $w \in \mathbb{R}_+$ is generated according to an exponential distribution with $\lambda = weight$, where, $weight$ is the weight of the associated transition. We justify later this choice.

So here the event located at the top will be the first to occur. The comparison rule between events is:

$$(i_1, t_1, pr_1, w_1) < (i_2, t_2, pr_2, w_2) \text{ iff } \begin{cases} t_1 < t_2 \\ or \\ t_1 = t_2 \quad and \quad pr_1 > pr_2 \\ or \\ t_1 = t_2 \quad and \quad pr_1 = pr_2 \quad and \quad w_1 < w_2 \end{cases}$$

Variables or procedures	Meaning
$SimTime$	The current simulation time
$Mark$	The current marking in $\mathcal{N}$
$OldMark$	The previous marking in $\mathcal{N}$
Loc	The current location in $\mathcal{A}$
$Event$	An instance of a Petri net transition t generated according to the definition of t (see 6.3.3)
$NullEvent$	Indicates that the queue is empty. ($NullEvent.index = -1, NullEvent.time = \infty$)
AE	An autonomous edge of $\mathcal{A}$. $SE.index = -1$ means that no autonomous edge is enabled
SE	A synchronized edge of $\mathcal{A}$. $AE.index = -1$ means that no synchronized edge is enabled
Δt	The time from now ($SimTime$) until the next edge firing in $\mathcal{A}$
$\mathcal{A}.EnabledInitLocation()$	Selects the enabled initial location of $\mathcal{A}$ from the set of initial locations according to the initial Petri net marking
$\mathcal{A}.GetEnabled\_A\_Edge()$	Selects the enabled autonomous edge.
$\mathcal{A}.GetEnabled\_S\_Edge()$	Selects the enabled synchronized edge.
$VarUpdates()$	Updates the values of the variables (due to time elapse of Δt).
$EdgeUpdates()$	Updates the values of the variables (updates set on the edges).
$FuncUpdates()$	Updates the values of $L_1(X), ..., L_n(X)$ and $f_1(L_1(X)), ..., f_n(L_n(X))$
$F()$	Computes the value of $F(f_1(L_1(X)), ..., f_n(L_n(X)))$
$\mathcal{A}.IsFinal()$	Checks if the current location is final
$GenerateInitialQueue()$	Initializes the queue of events
$Queue.top()$	Returns the first event in the queue
$Queue.IsEmpty()$	Checks if the queue is empty
$Queue.Update()$	Updates the events queue
$\mathcal{N}.Fire()$	Fires a Petri net transition

Table 6.1: Main variables and procedures

Let us justify the use of the exponential distribution to randomly select an event among those scheduled at the same time and with same priority. Let consider the following set of events:

$$\{e_1(i_1, t, pr, weight_1), e_2(i_2, t, pr, weight_2), ..., e_k(i_k, t, pr, weight_k)\}$$

The classical method to randomly select an event in this case consists to define a discrete random variable X on the space $\Omega\{i_1, ..., i_k\}$ with distribution: $Prob(X = i_j) = \frac{weight_j}{\sum_{l\in\{1,...,k\}} weight_l}$ and generate a random number according to this distribution. The main difficulty here is to find all the nodes in the heap having the same minimal delay and the same maximal priority. Instead we make a random sample per node such that whatever the set of nodes that can be selected the individual samples provide the selected node. Let us associate with the considered set of events a set of random variables $X_1, ..., X_k$ exponentially distributed with rate $\lambda_k = weight_k$. Let us consider a random sample $w_1, ..., w_k$ where each w_j is an observation of X_j. The rule selects e_j such that $w_j = \min\{w_1, ..., w_k\}$. The rule is sound since $Prob(X_j \leq \min_{l\neq j}\{X_l\}) = \frac{\lambda_j}{\sum_{l\in\{1,...,k\}} \lambda_l}$.

The traditional algorithms for adding or removing an event from the heap are implemented as they are known in the literature. Another table $PosInHeap$ is added with a fixed size equal to the number of Petri net transitions, Where $PosInHeap[t]$ is equal to the position of t in the heap or (-1) if t is not in the heap.

Algorithm 6.4 shows how the queue is updated after firing a transition t.

When the transition t is fired, the net will be in a new marking, so some transitions will be enabled, some others will be disabled and others need to be re-sampled (Marking dependent transitions). The natural way to do that is to check all the net transitions. But this is a waste of time. We will do that by building for each transition t the following sets:

- $PossiblyEnabled[t]$, the set of transitions that may be enabled after firing t;
- $PossiblyDisabled[t]$, the set of transitions that may be disabled after firing t;
- $FreeMarkDep[t]$, the set of marking dependent transitions which are not in $PossiblyEnabled[t]$ or in $PossiblyDisabled[t]$.

These sets belong to the Petri net class. Algorithm 6.7, in the next subsection, constructs these sets. This algorithm is executed once, before starting the simulation. It accelerates the update of the events queue after an event queue.

Since we also manage the age memory policy, some tables are needed:

- $ActDate$ indexed by the transitions. $ActDate[t]$ is equal to last date of activation. The value of $ActDate[t]$ is updated at the beginning of an enabling period and it is set to (-1) after t has been fired.
- $ResidualTime$ indexed by the transitions. $ResidualTime[t]$ is equal to the residual time before firing t. The value is updated at the end of an enabling period.

Age memory policy is incompatible with the marking dependent rate.

Let us describe algorithm 6.4. Assume that t is the transition which was fired. In the first part (from line 2 to line 19), the algorithm examines transitions t' that may be enabled (line 2). If t' is not enabled there is nothing to do otherwise (line 3) additional tests will be done. If t' is already in the queue (line 16) it will be re-sampled only if it is marking dependent. If t' is not in the queue,

it will be inserted as a new event. The event to insert is generated, taking in account the memory policy of the transition (from line 5 to line 15).

In the second part (from line 20 to line 29), the algorithm examines transitions t' that may be disabled (line 20). If t' is not in the queue there is nothing to do otherwise (line 21) additional tests will be done. If t' is disabled (line 22) then the associated event is deleted from the queue (line 23). If t' is enabled then it will be re-sampled only if it is marking dependent (line 27).

In the last part (from line 30 to line 36), the algorithm re-samples marking dependent transitions which are not in $PossiblyEnabled[t]$ or in $PossiblyDisabled[t]$.

6.3.4 Petri Net Algorithms

Algorithm 6.5 checks if a given transition t is enabled. Let us recall that Petri net arcs as they are considered here are marking dependent, which requires tests at lines 3 and 6 to avoid arcs valuated by 0.

Algorithm 6.6 fires a Petri net transition.

Algorithm 6.7 constructs the sets: $PossiblyEnabled$, $PossiblyDisabled$ and $FreeMarkDep$.

6.3.5 Automaton Algorithms

The most important algorithms in the automaton class are: $GetEnabled\_S\_Edge$ and $GetEnabled\_A\_Edge$ which respectively return the enabled synchronized edge and the enabled autonomous edge and when they should be traversed.

Assume that e is the first scheduled event and the automaton is in a location loc. $GetEnabled\_S\_Edge$ examines the set of synchronized edges which start from loc and contain transition t associated with e. This set is given by the data structure $TransitionLocation[loc, t] = \{s\_edge_{i1}, s\_edge_{i2}, ...\}$, built before launching the simulation. Then, an edge from this set is enabled if its constraints are satisfied by the variables at time $e.time$ and the target location is satisfied by the new marking. By determinism, at most one synchronized edge is enabled. $GetEnabled\_S\_Edge$ returns a pair $(edge\_index, e.time)$. The traversing time of the selected edge is the occurring time of the event. If no edge is enabled the pair $(-1, \infty)$ is returned.

Algorithm 6.4: Queue.Update()

```
   Data: The fired transition t
   Result: An updated heap
1  begin
2      for (t' ∈ PossiblyEnabled[t]) do
3          if (N.IsEnabled(t')) then
4              if (PosInHeap[t'] < 0) then
5                  if (t'.IsAgeMemory) then
6                      if (ActDate[t'] < 0) then
7                          Event = GenerateEvent(t', SimTime)
8                          Queue.insert(Event)
9                          ResidualTime[t'] = Event.time − SimeTime
10                     else
11                         Event = GenerateEvent(t', SimTime)
12                         Event.time = SimTime + ResidualTime[t'],  Queue.insert(Event)
13                     ActDate[t'] = SimTime
14                 else
15                     Event = GenerateEvent(t', SimTime),  Queue.insert(Event)
16             else
17                 if (t'.IsMarkingDependent) then
18                     Event = GenerateEvent(t', SimTime)
19                     Queue.replace(Event, PosInHeap[t'])
20     for (t' ∈ PossiblyDisabled[t]) do
21         if (PosInHeap[t'] > −1) then
22             if (¬N.IsEnabled(t')) then
23                 Queue.remove(PosInHeap[t'])
24                 if (t'.IsAgeMemory) then
25                     ResidualTime[t'] = SimTime − ActDate[t']
26             else
27                 if (t'.IsMarkingDependent) then
28                     Event = GenerateEvent(t', SimTime)
29                     Queue.replace(Event, PosInHeap[t'])
30     for (t' ∈ FreeMarkDep[t]) do
31         if (N.IsEnabled(t')) then
32             if (PosInHeap[t'] < 0) then
33                 Event = GenerateEvent(t', SimTime),  Queue.insert(Event)
34         else
35             Event = GenerateEvent(t', SimTime)
36             Queue.replace(Event, PosInHeap[t'])
```

Assume that the automaton is in a location loc at time $SimTime$. Then procedure $GetEnabled\_A\_Edge$ examines the autonomous edges starting from loc. The first step consists in building the set of autonomous edges starting from loc such that their target location satisfies the current marking. The second step consists in computing, for the selected edges, time intervals $[low_i, up_i]$ in which each edge can be traversed. The third step consists in eliminating edges

Algorithm 6.5: IsEnabled(t)

Data: A Petri net transition t
Result: **true** if t is enabled

```
1  begin
2      for (p ∈ Γ⁻(t)) do
3          if (In[t,p] > 0) then
4              if In[t,p] < Marking[p] then
5                  return false
6      for (p ∈ Γ^o(t)) do
7          if (Inhib[t,p] > 0) then
8              if Inhib[t,p] ≥ Marking[p] then
9                  return false
10     return true
```

Algorithm 6.6: Fire(t)

Data: A Petri net transition t
Result: Marking update

```
1  begin
2      for (p ∈ Γ⁻(t)) do
3          Marking[p] = Marking[p] − In[t,p]
4      for (p ∈ Γ⁺(t)) do
5          Marking[p] = Marking[p] + Out[t,p]
```

Algorithm 6.7: PostFiring()

Data: A Petri net $\mathcal{N}$
Result: PossiblyEnabled, PossiblyDisabled, FreeMarkDep

```
1  begin
2      for (t ∈ N.Transitions) do
3          for (p ∈ Γ⁻(t)) do
4              for (t' ∈ Γ⁺(p)) do
5                  PossiblyDisabled[t] = PossiblyDisabled[t] ∪ {t'}
6              for (t' ∈ Γ^o(p)) do
7                  PossiblyEnabled[t] = PossiblyEnabled[t] ∪ {t'}
8          for (p ∈ Γ⁺(t)) do
9              for (t' ∈ Γ⁺(p)) do
10                 PossiblyEnabled[t] = PossiblyEnabled[t] ∪ {t'}
11             for (t' ∈ Γ^o(p)) do
12                 PossiblyDisabled[t] = PossiblyDisabled[t] ∪ {t'}
13         for (t' ∈ N.Transitions) do
14             if (t'.MarkingDependent) then
15                 if (t' ∉ PossiblyEnabled[t]  and  t' ∉ PossiblyDisabled[t]) then
16                     FreeMarkDep[t] = FreeMarkDep[t] ∪ {t'}
```

having $up_i < SimTime$. In the forth step, the traversing time is computed for the remaining edges $TraversingTime = max\{low_i, SimTime\}$. Finally, the autonomous edge with the minimal $TraversingTime$ is selected as the enabled autonomous edge and a pair $(edge\_index, TraversingTime)$, is returned. Like synchronized edges and by determinism at most one autonomous edge is enabled. These steps are done edge per edge.

Let us illustrate how the time interval $[low_i, up_i]$ is computed for an autonomous edge with constraints:

$$\sum_j a_j^{(1)} x_j \bowtie b_1 \tag{6.1}$$

$$\sum_j a_j^{(2)} x_j \bowtie b_2 \tag{6.2}$$

where $\bowtie \ \in \{=, \leq, \geq\}$. First, $[low_i, up_i]$ is initialized with $[SimTime, \infty]$. Then a time interval $[low_i^1, up_j^{(1)}]$ is computed for the first equation.

Equation 6.1 can be rewritten as:

$$\forall t \geq t_0, \ \sum_j \left(a_j^{(1)} \left(t - t_0\right) r_j + x_j \left(t_0\right) \right) \bowtie b_1 \tag{6.3}$$

where, $t_0 = SimTime$, r_j is the rate of the variable x_j in the current location and $x_j\left(t_0\right)$ is the value of x_j at time t_0.

$$(6.3) \Rightarrow t \sum_j \left(a_j^{(1)} r_j \right) - t_0 \sum_j \left(a_j^{(1)} r_j \right) + \sum_j \left(a_j^{(1)} x_j(t_0) \right) \bowtie b_1 \tag{6.4}$$

If $\sum_j \left(a_j^{(1)} r_j \right) = 0$, then:

$$[low_i^1, up_j^{(1)}] = \begin{cases}]-\infty, +\infty[& If \quad -t_0 \sum_j \left(a_j^{(1)} r_j \right) + \sum_j \left(a_j^{(1)} x_j(t_0) \right) \bowtie b_1 \\ \emptyset & otherwise \end{cases} \tag{6.5}$$

If $\sum_j \left(a_j^{(1)} r_j \right) \neq 0$ let $z = \frac{b_1 + t_0 \sum_j \left(a_j^{(1)} r_j \right) - \sum_j \left(a_j^{(1)} x_j(t_0) \right)}{\sum_j \left(a_j^{(1)} r_j \right)}$ and we will distinguish two cases according to the sign of $\sum_j \left(a_j^{(1)} r_j \right)$.

(1) If $\sum_j \left(a_j^{(1)} r_j \right) > 0$ then:

$$[low_i^1, up_j^{(1)}] = \begin{cases}]-\infty, z] & if \quad \bowtie \ is \ \leq \\ [z, +\infty[& if \quad \bowtie \ is \ \geq \\ [z, z] & if \quad \bowtie \ is \ = \end{cases} \tag{6.6}$$

(2) If $\sum_j \left(a_j^{(1)} r_j\right) < 0$ then:

$$[low_i^1, up_j^{(1)}] = \begin{cases}]-\infty, z] & if \quad \bowtie \ is \ \geq \\ [z, +\infty[& if \quad \bowtie \ is \ \leq \\ [z, z] & if \quad \bowtie \ is \ = \end{cases} \tag{6.7}$$

The new time interval for the edge is now $[low_i, up_i] = [low_i, up_i] \cap [low_i^1, up_j^{(1)}]$. If at this stage $[low_i, up_i] = \emptyset$ the edge is not enabled and the other constraints are not examined.

6.3.6 HASL Expression Update

A HASL expression can be written as: $H = AVG\left(F\left(f_1\left(L_1\left(X\right)\right), ..., f_n\left(L_n\left(X\right)\right)\right)\right)$ Where, $L_i(X)$ is a linear expression on X, thus, $L_i(X) = \sum_j a_j^{(i)} x_j$. $L_i(X)$ can also be viewed as a linear variable on time t, $L_i(X) = a_i \times t + b_i$.

$f_i \in \{Last, Min, Max, Integral, Mean, Var\}$ is a HASL function.

- $Last(L_i(X))$: computes the last value of $L_i(X)$ over a trajectory.
- $Min(L_i(X))$: computes the minimum value of $L_i(X)$ over a trajectory.
- $Max(L_i(X))$: computes the maximum value of $L_i(X)$ over a trajectory.
- $Integral(L_i(X))$: computes the integral $I = \int_0^T L_i(X)dt$.
- $Mean(L_i(X))$: computes the mean value of $L_i(X)$ over a trajectory. In other words $M = \frac{1}{T}\int_0^T L_i(X)dt = \frac{I}{T}$.
- $Var(L_i(X))$: computes the variability of $L_i(X)$ over a trajectory, defined by $V = \frac{1}{T}\int_0^T \left(L_i(X) - M\right)^2 dt$ or equivalently $V = \frac{1}{T}\int_0^T \left(L_i(X)\right)^2 dt - M^2$.

F: is an arbitrary function.

The values of $L_i(X)$ are updated on the fly during the generation of a trajectory. The value of F is computed when the trajectory is achieved. In terms of probability, $X(t)$ and $L_i(X(t))$ are stochastic processes, $f_i(L_i(X))$ and $F(.)$ are random variables, and $AVG(F())$ is the expectation of $F(.)$.

The value of $Last(L_i(X))$ is the last value of $L_i(X)$. For the other functions, let us consider a trajectory generated in $[0, T]$ and denote:

- k, the k^{th} jump of the automaton (i.e an edge is traversed).
- t_k, the date of the k^{th}.
- Let us also consider k^{th} period of time $[t_k, t_{k+1}[$ where the automaton is in the same location and $OldLX_i^k$ (and LX_i^k) the value of $L_i(X)$ at the beginning (resp. at the end) of the k^{th} period. Note that the values of LX_i^k $OldLX_i^{k+1}$ are not necessary equal because of point update specified on the automaton edges.

Now let f_i^k be the value of $f_i(L_i(X))$ at the end of the k^{th} period. We recursively the value of $f_i(L_i(X))$:

$$Last_i^k = LX_i^k \tag{6.8}$$

$$Min_i^k = \begin{cases} min\{OldLX_i^k, LX_i^k\} & if \quad k = 0 \\ min\{Min_i^{k-1}, OldLX_i^k, LX_i^k\} & if \quad k > 0 \end{cases} \tag{6.9}$$

$$Max_i^k = \begin{cases} max\{OldLX_i^k, LX_i^k\} & if \quad k = 0 \\ max\{Max_i^{k-1}, OldLX_i^k, LX_i^k\} & if \quad k > 0 \end{cases} \tag{6.10}$$

$$Integral_i^k = \begin{cases} \int_{t_k}^{t_{k+1}} L_i X(t)dt & if \quad k = 0 \\ Integral_i^{k-1} + \int_{t_k}^{t_{k+1}} L_i X(t)dt & if \quad k > 0 \end{cases} \tag{6.11}$$

$$Mean_i^k = \begin{cases} \frac{1}{t_{k+1}-t_k} \int_{t_k}^{t_{k+1}} L_i X(t)dt & if \quad k = 0 \\ \left(t_k \times Mean_i^{k-1} + \int_{t_k}^{t_{k+1}} L_i X(t)dt\right) / t_{k+1} & if \quad k > 0 \end{cases} \tag{6.12}$$

$$Var_i^k = M2_i^k - (Mean_i^k)^2 \tag{6.13}$$

where,

$$M2_i^k = \begin{cases} \frac{1}{t_{k+1}-t_k} \int_{t_k}^{t_{k+1}} \left(L_i X(t)\right)^2 dt & if \quad k = 0 \\ \left(t_k \times M2_i^{k-1} + \int_{t_k}^{t_{k+1}} \left(L_i X(t)\right)^2 dt\right) / t_{k+1} & if \quad k > 0 \end{cases} \tag{6.14}$$

Finally let us show how the integrals $I_k = \int_{t_k}^{t_{k+1}} LX(t)dt$ and $I2_k = \int_{t_k}^{t_{k+1}} (LX(t))^2 dt$ are efficiently computed.

$\forall t_k \leq t \leq t_{k+1}, LX(t) = \sum_j a_j(t-t_k)r_j^k + LX(t_k) = t\sum_j a_j r_j^k - t_k \sum_j a_j r_j^k + LX(t_k)$

Let be $a_k = \sum_j a_j(t-t_k)r_j^k$ and $b_k = -t_k \sum_j a_j r_j^k + LX(t_k)$. Observe that a_k is also equal to $\frac{LX(t_{k+1})-LX(t_k)}{t_{k+1}-t_k}$.

Computing I_k

(1) If $a_k \neq 0$ then $I_k = \int_{t_k}^{t_{k+1}} (a_k t + b_k)\, dt = \frac{1}{a}\int_{a_k t_k + b_k}^{a_k t_{k+1}+b_k} y dy = \frac{1}{a}\int_{LX(t_k)}^{LX(t_{k+1})} y dy = \frac{1}{2a}\left(LX(t_{k+1})^2 - LX(t_k)^2\right)$. Then, $I_k = \frac{(t_{k+1}-t_k)(LX(t_{k+1})+LX(t_k))}{2}$.

(2) If $a_k = 0$ then $I_k = (t_{k+1} - t_k)\, LX(t_k)$.

Computing $I2_k$

(1) If $a_k \neq 0$ then $I2_k = \int_{t_k}^{t_{k+1}} (a_k t + b_k)^2\, dt = \frac{1}{a}\int_{a_k t_k + b_k}^{a_k t_{k+1}+b_k} y^2 dy = \frac{1}{a}\int_{LX(t_k)}^{LX(t_{k+1})} y^2 dy = \frac{1}{3a}\left(LX(t_{k+1})^3 - LX(t_k)^3\right)$. Then, $I2_k = \frac{(t_{k+1}-t_k)\left(LX(t_{k+1})^2+LX(t_{k+1})LX(t_k)+LX(t_k)^2\right)}{3}$.

(2) If $a_k = 0$ then $I2_k = (t_{k+1} - t_k)\, LX(t_k)^2$.

6.4 Case Studies

In this section, we perform some experiments on different models and different types of measure. Le us precise that all the experiments are done under these conditions.

- COSMOS 1.0
- PRISM 4.0
- CPU: Intel(R) Core(TM)2 Duo CPU T9400 @ 2.53GHz
- Memory: 4 G.O
- Operating system: Fedora release 15.

We add that the execution time is expressed in seconds.

6.4.1 Kanban System

The first example is a Kanban system. Kanban systems are a special kind of production systems designed to minimize the size as well as the fluctuation of in-process inventory [ABC+95]. They have been initially introduced in the Japanese industry. Let us consider the model presented in figure 6.4.1 introduced in [CT96]. It consists of 4 cells limited in size to N. In each cell i, incoming pallets are put in place Pm_i waiting for service. After service (transition Tm_i) a pallet moves to place $Pout_i$ by means of the immediate transition Tok_i and waits for outgoing or to place $Pback_i$ by means of the immediate transition $Tredo_i$ and waits to be moved to the place Pm_i. Pallets enter the system from the cell 1 (transition Tin_1) and leave it from cell 4 (transition $Tout_4$). Transition $Tsynch_{123}$ synchronizes moves between cells 1, 2 and 3 while transition $Tsynch_{234}$ synchronizes moves between cells 2, 3 and 4.

Transient measurement of the throughput. Let us compute the throughput of the transition $\mathbf{Tin_1}$ during the time interval $[0, T]$. The first goal of these experiments is to compare COSMOS to the numerical version of PRISM in term of quality of results. The second goal is to compare COSMOS with the statistical version of PRISM in term of execution time. In these experiments $N = 4$, the confidence interval level $1-\alpha = 0.99$ and the absolute confidence interval width $w = 10^{-3}$. The results of the experiments are presented in the table 6.2.

T	Throughput			Generated paths		exec-time		Ratio Prism/Cosmos	
	PRISM num	PRISM stat	COSMOS	PRISM	COSMOS	PRISM	COSMOS	total time	single path
10	0.5960	0.5960	0.5961	449273	449370	61.2	26	2.35	2.36
50	0.3511	0.3512	0.3509	69458	68510	60.2	22	2.74	2.70
100	0.3136	0.3133	0.3136	34360	34420	60.9	22	2.77	2.77
500	0.2834	0.2832	0.2834	6774	7040	60.5	22	2.75	2.86
1000	0.2796	0.2797	0.2798	3385	3410	61.5	22	2.80	2.82
5000	0.2766	0.2766	0.2769	739	720	66.3	23	2.88	2.81
10000	0.2762	0.2763	0.2760	344	370	62.1	26	2.39	2.57

Table 6.2: Throughput in a Kanban system

The results show that the numerical evaluations of PRISM are contained in the confidence intervals given by COSMOS . The results show also that COSMOS is at least 2.35 faster than the statistical version of PRISM when comparing the total time to do an experiment and when comparing the average time to generate a trajectory.

Transient analysis of a probabilistic measure. Let us compute the probability $p = Prob(\big((Pm2 + Pm3 = 0)\mathcal{U}^{[0,T]}(Pm1 = N)\big))$. This is a bounded until for-

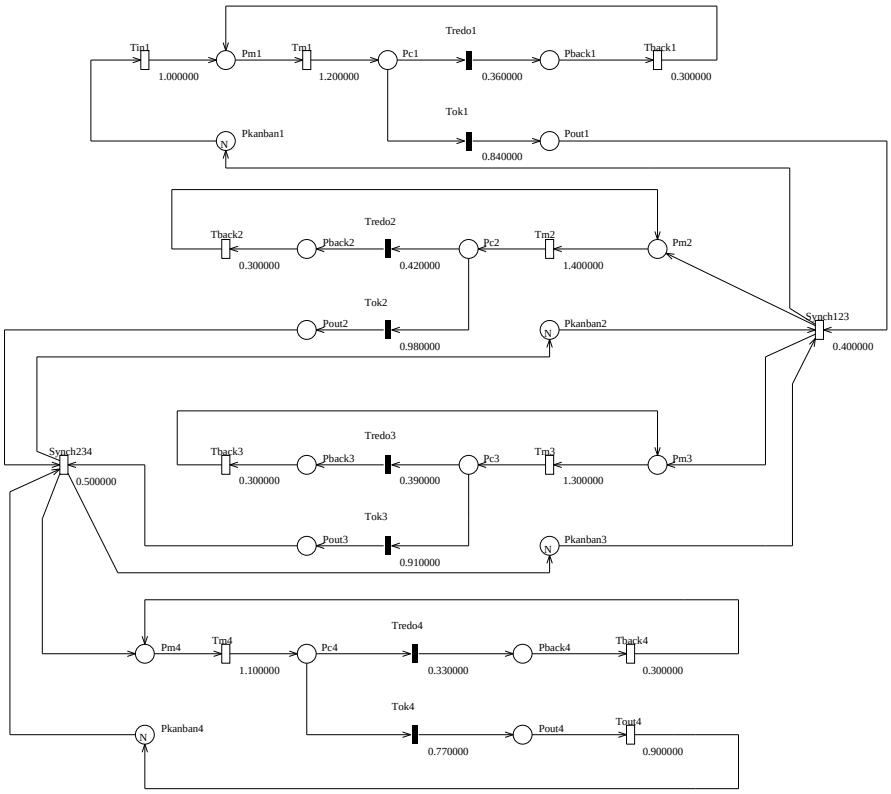


Figure 6.2: A Kanban System

mula. The goals and the parameter of these experiments are the same as the ones of the last paragraph. The results are presented in the table 6.3.

As before, the numerical evaluations of PRISM are contained in the confidence intervals given by COSMOS . COSMOS is at least 1.64 faster than the statistical version of PRISM when comparing the total time to do an experiment and when comparing the average time to generate a trajectory.

6.4.2 Fault-tolerant Cluster of Workstations

The second example is a fault-tolerant cluster of workstations. The model in figure 6.4.2 introduced in [BCH$^+$10] represents the system by a stochastic Petri net. The system consists of two sub-clusters (left and right) connected via a

T	p			Generated paths		exec-time		Ratio Prism/Cosmos	
	PRISM num	PRISM stat	COSMOS	PRISM	COSMOS	PRISM	COSMOS	total time	single path
1	0.0100	0.0102	0.0101	266867	270000	4.0	2	2.02	2.04
2	0.0467	0.0469	0.0468	1185604	1183900	23.0	14	1.64	1.64
3	0.0812	0.0815	0.0814	1986205	1983600	48.6	27	1.80	1.80
4	0.1024	0.1018	0.1024	2427320	2439700	69.1	37	1.87	1.88
5	0.1132	0.1132	0.1129	2665328	2658000	80.5	42	1.92	1.91
6	0.1183	0.1180	0.1187	2762627	2777000	87.5	47	1.86	1.87
7	0.1206	0.1207	0.1206	2817136	2814100	93.3	47	1.99	1.98
8	0.1216	0.1217	0.1216	2835033	2834100	97.1	49	1.98	1.98
9	0.1221	0.1217	0.1218	2837471	2839500	94.2	49	1.92	1.92
10	0.1222	0.1222	0.1222	2846391	2846300	97.2	49	1.98	1.98

Table 6.3: Bounded until in a Kanban system

backbone. In each sub-cluster, a set of workstations ($N1$ left, $N2$ right) are connected in a star topology to a central switch. Each component (workstation, switch, backbone) can break down. A single repair unit will then repair the failed components.

Let us precise that transitions of type *Inspect* were originally immediate. They are here considered as exponentially distributed to have an equivalent Petri net to the PRISM model proposed in [PNK].

The first goal of these experiments is to introduce the regeneration point method to estimate a steady state measure. The second goal is to show the main drawback of the numerical methods which is the memory consuming.

The regenerative method. A regeneration point for a stochastic process is an instant in time at which the future behavior of the process depends only on its state at that instant, it is independent of the evolutionary path that led to its current state [Ste09]. The idea of the regenerative method is to divide the trajectory of the system into a series of cycles according to a regeneration point. Thus the evolution of the system in a cycle is a probabilistic replica of the evolution in any other cycle [HL10]. If we associate some statistics of interest to these cycles, these statistics for the respective cycles constitute a series of independent and identically distributed observations that can be analyzed by standard statistical procedures.

The theoretical requirements for the method are that the probability of returning to the regenerative point is equal to one and the expected cycle length be finite.

The problem of the regenerative method is to find an estimate of the expectation of a random variable X of interest by using the observations of a random variable Y collected during each cycle and the random variable D representing

the size of the cycles. More precisely:

$$E[X] = \frac{E[Y]}{E[D]} \tag{6.15}$$

Let us consider n cycles, the data gathered are $y_1, y_2, ..., y_n$ and $d_1, d_2, ..., d_n$ for the respective cycles. Let us denote $\overline{Y}$ and $\overline{D}$, respectively, denote the sample averages for these two sets of data, the corresponding point estimate of $E[X]$ is:

$$\hat{E[X]} = \frac{\overline{Y}}{\overline{D}} \tag{6.16}$$

The empirical confidence interval of $E[X]$ is given by:

$$\left[\frac{\overline{Y}}{\overline{D}} - \frac{Z_{\alpha/2} \times s}{\overline{D} \times \sqrt{n}}, \frac{\overline{Y}}{\overline{D}} + \frac{Z_{\alpha/2} \times s}{\overline{D} \times \sqrt{n}}\right] \tag{6.17}$$

where:

- $s^2 = s_{11}^2 - 2\frac{\overline{Y}}{\overline{D}} s_{12}^2 + \left(\frac{\overline{Y}}{\overline{D}}\right)^2 s_{22}^2$
- $s_{11}^2 = \frac{1}{n-1} \sum_{i=1}^{n} \left(y_i - \overline{Y}\right)^2$
- $s_{22}^2 = \frac{1}{n-1} \sum_{i=1}^{n} \left(d_i - \overline{D}\right)^2$
- $s_{12}^2 = \frac{1}{n-1} \sum_{i=1}^{n} \left(y_i - \overline{Y}\right)\left(d_i - \overline{D}\right)$

In this example, we compute the probability that the repair unit is available. We consider the symmetric case $N1 = N2 = N$, the confidence interval level is set to 0.99, the relative confidence interval width is set to 10^{-2} and the PRISM resolution method is the Gauss-Seidel one. The results are presented in table 6.4.

N	Sizes		Availability		Iterations	Generated paths	Memory		Exec-time	
	|States|	|Transitions|	PRISM	COSMOS	PRISM	COSMOS	PRISM	COSMOS	PRISM	COSMOS
100	365620	1779232	0.79671	0.79585	267	31200	4.6 MB	3.7 MB	9.6	1.0
200	1451220	7078432	0.59725	0.59856	653	134800	15.7 MB	3.7 MB	87.5	6.0
300	3256820	15897632	0.39846	0.39785	1381	376700	33.3 MB	3.7 MB	388.1	24.0
400	5782420	28236832	0.20338	0.20361	3388	733500	57.8 MB	3.7 MB	1990.7	137.0
500	9028020	44096032	0.09089	0.09086	10180	1000	89.2 MB	3.7 MB	7041.0	369.0

Table 6.4: COSMOS vs Numerical Prism wrt cluster of workstations

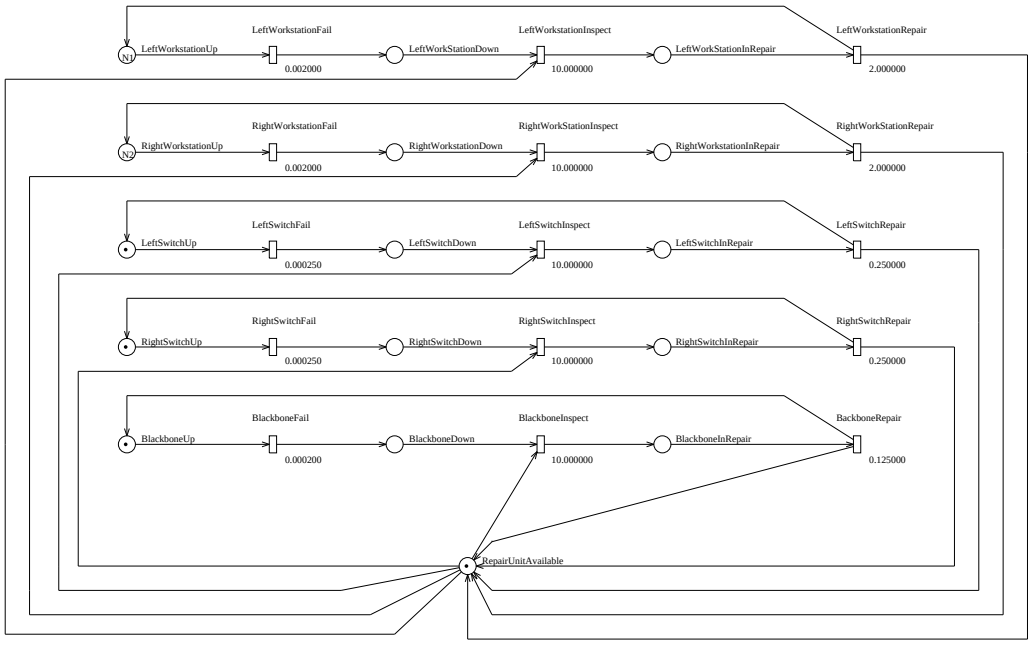


Figure 6.3: Stochastic Petri of a fault-tolerant cluster of workstations

6.4.3 $M/G/1/\infty/\infty$ Queue

In this example, we consider a queue $M/G/1/\infty/\infty$ which consists of a queue with an infinite capacity. The population of the clients is infinite and arrive according to a Poisson process. The service duration is generally distributed. The goal of theses experiments is to analyze a system with an infinite state space.

Here we compute the expected number of clients in the system. This quantity is given by the Pollaczek–Khinchine formula :

$$L = \rho + \frac{\rho^2 + \lambda^2 * Var(S)}{2(1-\rho)} \tag{6.18}$$

The confidence interval level is set to 0.99, the relative confidence interval width is set to 10^{-2}.

We have done the experiments with two different service distribution but

parameters are chosen such as the expectation and the variance of the service distributions are the same. In table 6.5, the service is distributed with a uniform distribution $U[0.2, 0.7]$. In table 6.6, the service is distributed with a gamma distribution $\Gamma(k = 9.72, \theta = 0.0462962963)$.

λ	ρ	L		Generated paths	Exec-time
		Theoretical	COSMOS		
0.8	0.36	0.4717	0.4726	439100	4
1.0	0.45	0.6530	0.6526	534600	5
1.2	0.54	0.8895	0.8894	657100	7
1.4	0.63	1.2214	1.2247	895500	10
1.6	0.72	1.7408	1.7440	1318700	17
1.8	0.81	2.7139	2.7110	2175100	35
2.0	0.90	5.3660	5.3730	4765100	122
2.2	0.99	55.0286	55.0856	58552700	15689

Table 6.5: $M/Unif/1/\infty/\infty$ Queue

λ	ρ	L		Generated paths	Exec-time
		Theoretical	COSMOS		
0.8	0.36	0.4717	0.4726	441700	4
1.0	0.45	0.6530	0.6541	528600	5
1.2	0.54	0.8895	0.8906	675400	8
1.4	0.63	1.2214	1.2228	913700	12
1.6	0.72	1.7408	1.7430	1301900	19
1.8	0.81	2.7139	2.7199	2245700	42
2.0	0.90	5.3660	5.3524	4847000	145
2.2	0.99	55.0286	54.7729	56394300	17291

Table 6.6: $M/Gamma/1/\infty/\infty$ Queue

We note that the simulation results are very close to those obtained by the theoretical formula. We also note that the average time to generate a trajectory with a gamma distribution is larger than that with a uniform distribution.

6.5 Conclusion

We have presented in this chapter the main features of COSMOS. We also performed some numerical experiments on various models to check different properties. COSMOS can deal with finite and infinite state space models. We have checked some transient properties and shown how to adapt the method of regeneration point to evaluate steady state properties. The comparison with the statistical version of PRISM shows that COSMOS is faster while handling more general properties. In the future, we plan to automatize the method of regeneration point. We also want to parallelize our tool.

Chapter 7

Compositional Modeling of Flexible Manufacturing Systems Using Petri Nets.

This chapter is related to the publication [BDD+11c].

7.1 Introduction

Analysis of Flexible Manufacturing Systems (FMS). FMS have been introduced in order to optimize different criteria of manufacturing systems. For instance, one wants to efficiently manage crashes, increase the productivity and the flexibility, etc. In such a context, a critical issue consists in evaluating these criteria and comparing different architectures before selecting the appropriate one. This implies to resort to formal models and evaluation methods.

Modeling FMS with Petri Nets. The Petri net formalism is applied in numerous application areas. Compared to other formalisms, Petri nets are appropriate to model concurrent activities (each one described by a finite automaton) sharing resources (described by additional places) and communicating via synchronization (described by transitions). Since FMS present such characteristics, they are a good candidate to be modeled and analyzed with the help of Petri nets [WD98].

Indeed several approaches have been undertaken differing w.r.t. their goals and the kind of nets used for modeling. When one is interested in qualitative properties of FMS like deadlock prevention, modeling is based on structural

subclasses of Petri nets allowing to design efficient algorithms [ECM95]. When one is interested in performance of FMS, there are (at least) two possible modeling approaches: either to substitute discrete quantities by continuous ones leading to an hybrid Petri net [BGS01] or to represent the uncertainties related to the FMS behavior by distribution probabilities leading to a stochastic Petri net [ABC+95, LCGH93]. Here we follow the latter approach.

Limitations of Petri net modeling of FMS. There are two drawbacks of Petri nets w.r.t. the modeling. First there are no net operators that would lead to a compositional modeling. In [BDK01], Petri nets have been extended with operators. This is an interesting theoretical approach but the subnets are not viewed as components. For instance, they do not own an interface and an internal part. Second, the syntax and semantics of nets may prevent modelers used to their dedicated formalism to switch to Petri nets.

Steady-state and transient-analysis of FMS. Although the vast majority of FMS stochastic modeling studies have been focused on the analysis of *steady-state*-based measures (such as, for example, *throughput*, *productivity*, *makespan*) the relevance of *transient-analysis* of FMS models has been demonstrated [NV94]. For instance, as soon as faults are modeled, transient measures like the time until FMS stopping are interesting. Furthermore it is well known that for systems presenting regenerative points (like idle states), every steady-state measure may be obtained by averaging the corresponding transient measure between two occurrences of a regenerative point.

In chapters 5 and 6, we have presented a framework with a dedicated prototype tool COSMOS for analyzing complex systems modeled by stochastic Petri nets via quantitative model-checking of formulas specified in an expressive language HASL. In this chapter, we show an high-level modeling approach for FMS developed over this framework presenting the following features.

- **A compositional framework targeted to FMS modeling.** Our framework allows to build arbitrarily large/complex models of FMS by composition of basic elements representing the elementary parts of a FMS. Following an object-oriented approach, we start with three generic classes: the load unit class, the machine class and the transportation class. Then the modeler specializes these classes in order to express the characteristics of his specific architecture. This specialization concerns both the qualitative features like the routing policy of a transporter and the quantitative features like the loading time of a unit. By instantiating such classes into components and gluing them through their interface, he finally produces the FMS architecture. During the modeling stage, Petri net patterns associated

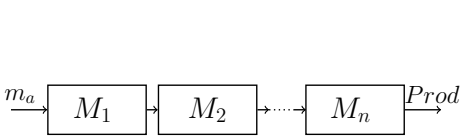
with specialized classes are automatically generated. Then these patterns are duplicated to reflect the components corresponding to the classes and linked via place merging to obtain the final stochastic Petri net. Observe that the net is managed internally so that one does not require any Petri net knowledge from the user.

- **A set of formal properties customized for analysis of FMS.** As discussed before, several specific qualitative and quantitative properties of FMS are relevant. Our framework includes a set of FMS-oriented properties described in natural language including the appropriate parameters w.r.t. the property. Once the user fixes the parameters, an HASL formula is automatically generated and later evaluated over the SPN corresponding to the FMS. Since HASL is very expressive the specification of almost all relevant properties for FMS does not present any difficulty.
- **A case study of an FMS.** We provide an FMS model, we give the corresponding compositional scheme. Then we will evaluate some HASL formulae using COSMOS.

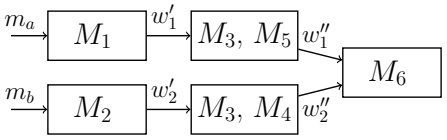
In section 7.2 we present our modeling approach. We illustrate the approach on a toy example in section 7.3 and the adequacy of HASL formulae for analysis of FMS in section 7.4. In section 7.5, we describe the analysis of a significant case study. Finally we conclude and give perspectives to this work in section 7.6.

7.2 Compositional FMS modeling using Petri Nets

We introduce a stochastic Petri net (SPN) based modeling framework for FMS, by means of which a model of an FMS can be obtained through assembling of the desired combination of *basic components*. FMS are, by nature, heterogeneous systems which may differ from one another both in the functionality of components as well as on the workflow of the production process. The compositional framework we introduce is designed to cope with the heterogeneity of FMS. For example, components for modeling of a simple *linear FMS* (whose workflow is drafted in Figure 7.1(a)), whereby the end product is obtained by processing of a single type of raw material through a line of machines connected by a conveyor belt, will be (internally) different from components for modeling of a more complex FMS (e.g. workflow drafted in Figure 7.1(b)), involving multiple types of material (m_a and m_b), machine selection (e.g. workpiece w_1' outputted by machine M_1 is delivered to either M_3 or M_5) and workpieces combination (e.g. workpieces w_1'' and w_2'' are combined by machine M_6).



(a) A simple linear FMS with single type of raw material



(b) An FMS with 2 types of raw materials, machine-selection and workpiece-combination

Figure 7.1: Different types of workflow for FMSs

7.2.1 Principles of the proposed approach

Goal. A framework for modeling and analysis of complex systems is well suited to the user when this framework allows the user to model his system as he has designed it and to analyze it with logic formulas and/or performance indices which are directly meaningful to him. This has led us to the following choices in the context of FMSs. First the modeling we propose is component-based as FMS are explicitly obtained by assembling different functional parts. In addition, the granularity of the components correspond to the architectural decomposition of the FMS meaning that there is a one to one mapping between the model components and the business components. Although our internal formalism is an SPN, the user can fully model his FMS without specifying any Petri net. Similarly, while the formula language supported by the framework is very expressive, a set of specific formula patterns corresponding to usual FMS analyses is proposed to the user. Since these patterns can be instantiated by fixing different parameters, this yields a simple and flexible way to check the model.

An overall view. The FMS modeling framework is based on the following principles:

1. It lies on the following basic component classes: *Load Unit* (representing the loading of raw materials into the system), *Machine* (representing the various phases of the actual manufacturing of workpieces) and *Transportation* (representing the movement of materials/workpieces). These component classes have different attributes reflecting their functional properties, that may be quite complex. For instance routing policy may be fixed or state-dependent and in the latter case may depend on the occupation of buffers or occurrences of failures.

2. First the modeler specializes the basic classes by fixing the values of their attributes. As in object-oriented approaches, it allows to reuse the specialized classes for different architectures of FMS sharing some identical component types.

3. Then the modeler instantiates these specialized classes in (named) components.

4. At last these components are combined using the names to bind variables occurring in the interface of the class. For instance, assume that the definition of a transporter class involves some machine variable, say X, producing the inputs of the transporter. Then, when the transporter is instantiated as a component, variable X is substituted by a machine name.

An internal view. When the set of specialized classes are specified through a (natural-language-like) syntax, the corresponding SPN subnets are automatically generated from such specifications. The SPN subnets can be seen as boxes partitioned in an *interface* and an *internal structure*. SPN components interface consists of *local* and *imported places* arranged on the edge of a box. Local places (denoted as non-filled-in circles) represent relevant aspects of a component's state (that may be imported by other components). Imported places (denoted as filled-in circles) represent relevant aspects of external components that influence the importing component behavior.

Naming of places and transitions is essential for the assembling of FMS. The name of local places can be viewed as local identifiers. Depending on their role in the component, the name can be predefined, as $idle$ in Figure 7.3, or composed by a predefined word followed by an identifier provided by the user, like in_a1 in the same figure. Here $a1$ corresponds to the name of a product and in means that this product is an input of the machine. Names of imported places are built by prefixing a local name by a variable like $X3.in\_a1$ in Figure 7.4. Observe that the set of variables occurring in a subnet corresponds to the components that will communicate with a component of this class. Since the variable are typed by their class, the compilation stage checks that the interfaces intended to be linked are compatible. Transition names are handled like local place names.

When the modeler instantiates a class into one or several (named) components, he must provide a component name per variable occurring in this class. At the net level, we need one class subnet copy per component of this class. The names of local places and transitions are prefixed by the name of the component while the name of imported places is obtained by substituting the component

names to the variables. Now the assembling of the whole net is straightforward: it consists in merging places with identical names.

In the following subsections, we illustrate the specialization of the three basic classes and the associated subnets.

7.2.2 Modeling the load unit

The Load Unit (LU) class represents the process through which raw materials are loaded (from the "external world") into an FMS. **LU's parameters** are: *i) set of loaded materials:* the type of raw-materials the whole production system depends upon; in the case of a closed-system, the number of items for each material type *ii) size of buffers:* the size of the output buffers of the LU component; *iii) loading times:* the distribution of the loading time for each type of material.

3 types of material ($a1, a2, a3$) are loaded in the FMS. Items of type ai ($1 \leq i \leq 3$) are loaded according to (delay) distribution ld_ai. An SLU1 class has a finite buffer of size s. When s items are present in the buffer loading is interrupted, and it is automatically restarted as soon as one item is withdrawn from the output buffer of SLU1.

Table 7.1: Example of (informal) specialization of the load unit class

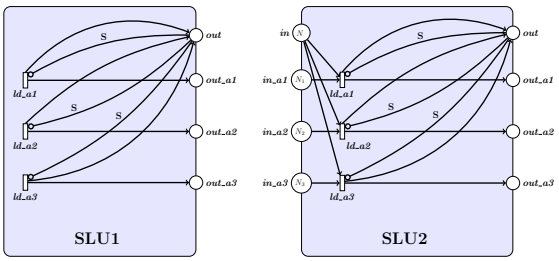


Figure 7.2: Internal structure of multi-material, buffered specialized Load Unit class for *open system* (left) and *closed system* (right)

An example of LU specialization (for an *open system*) for a FMS part named SLU1 is given in Table 7.1. The corresponding SPN component is depicted in Figure 7.2 (left), (Figure 7.2 (right) instead corresponds to a *closed system*). The **interface of the (*open system*) SLU1 class** consists of 4 (output) places corresponding to the items loaded into the FMS: place out containing the total of loaded items, place out_ai containing type ai loaded items. The **interface of the**

SLU1 class consists of 3 timed-transitions $load\_ai$ ($1 \leq i \leq 3$), representing loading of each type of piece. Note that since transitions ld_ai have no input places the underlying model is inherently infinite-state if we allow $s = \infty$. In case of a *closed system* (Figure 7.2 right) the interface contains 4 additional (input) places. They represent the initial amount of material: total amount (place in) and type ai amount (place in_ai) [1]. The control on the fullness of the $s \geq 1$ sized buffer is achieved through the inhibitor arcs connecting each "loading" transition with the (output) place out. An SLU1 class with infinite buffer capacity is obtained by removing inhibitor arcs from the components in Figure 7.2.

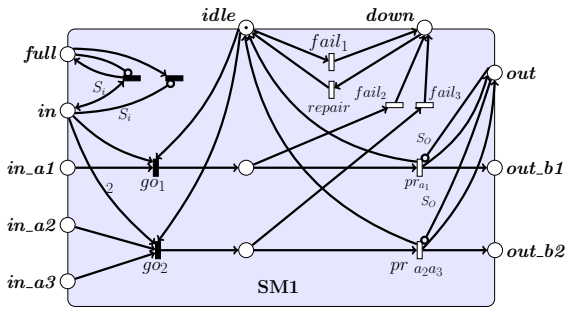


Figure 7.3: A specialized class of machine

7.2.3 Modeling the machine

The machine class (M) describes the behavior of machines processing materials and workpieces. **M's parameters** are: *i) set of input materials:* the type of materials/workpieces processed by the machine; *ii) buffers dimension:* the dimension ($\in \mathbb{N} \cup \{\infty\}$) of the input and output buffers of M; *iii) processing times:* the distribution of the processing time for each type of material/workpiece processed by M. This is given by a 3-tuple: first parameter is the set of input materials, the second is the set of produced workpieces, the third one is the distribution for this production. *iv) failure/repair times:* the distribution of failure and repairing times.

We now illustrate a possible specialization SM1 (described in table 7.2) of the machine class. The corresponding SPN component is depicted in Figure 7.3.

The ***interface of the SM1 class*** consists of 10 places (Figure 7.3): a 1-safe place $idle$ indicating the idle state of SM1 ; a 1-safe place $down$ indicating whether

[1] Note that in a *closed system* the input places of the LU component will be "connected" with the output places of the machine(s) which delivers the end product of the FMS.

Machine class SM1 processes 3 types of pieces: $a1, a2, a3$. It can process a $a1$ piece resulting in a type $b1$ piece. Another process consists in combining $a2$ and $a3$ resulting in $b2$. SM1 class is prone to failure and repairing. After a repairing, the unfinished workpieces are lost. *The input (resp. output) buffer sizes are denoted s_i and s_o*

Table 7.2: Example of (informal) specialization of the machine class

SM1 is down; a 1-safe place $full$ indicating that the input buffer of the machine is full; place in (out) representing the total number of items in the input (output) buffer of SM1 and places in_ai, $1 \leq i \leq 3$ (out_bj, $1 \leq j \leq 2$) indicating the number of items of type ai (bj) material (workpiece) in the input (output) buffer of SM1. In case of a machine producing a single type of workpiece from a single type of material, the interface of the corresponding subclass would simply consist of places $idle$, $full$, $down$, in and out.

Let us describe ***internal structure of SM1***. Transition $pra1$ and $pra2a3$ respectively represent processing of output pieces $b1$ and $b2$. Their associated distributions are obtained following the user specification. The control on the $s_o \geq 1$ sized output buffer is achieved through the inhibitor arcs connecting each "processing" transition with the (output) place out. Failures are modeled by three transitions: $fail_1$, $fail_2$ and $fail_3$ corresponding to a failure occuring respectively when the machine is idle, processing piece $b1$ or $b2$. Transition $repair$ models the repairing process.

Many other specializations of the machine class can be considered, for instance a machine could simultaneously process multiple inputs and outputs, or different materials could have different separate buffers.

7.2.4 Modeling the transportation unit

The transporter class (T) describes a transportation unit moving materials and workpieces from (a set of) *source nodes* to (a set of) *target nodes* of an FMS. Source and target nodes of a T component can be either machines or other transporters[2].

At the level of generic class T, the parameters are untyped. The types will be defined during the specialization. We now describe them informally. **T's parameters** are: *i) level of freedom* of the unit, specifying whether the trajectory is fixed or subject to change depending on the needs. *ii) transportation policy* expressing when the transporter decides to move and where. *iii) delivery time*

[2]machine-to-transporter movements are useful when modeling *continuous* transportation system such as, for example, conveyor belts transportation or rail-guided AGVs.

depending on the materials to be delivered and the location (initial or destination). Depending on the nature of the transportation unit, new parameters will appear in the specialized class. We now illustrate a possible specialization ST1 (described in table 7.3) of the transporter class. The corresponding SPN component is depicted in Figure 7.4. Note that almost all places in the interface are represented in grey as they are *imported* from other SPNs.

The ***interface of the ST1 class*** is composed of: i) 3 input places from machine $X1$, one for the total number of pieces and one for each type ($a1$ and $a2$). ii) five output places representing the input buffers of the destination machines, and the number of pieces of each type in each machine iii) six controlling places stating whether the destination machines are full, idle and down. iv) Finally place $idle$ indicates whether the transporter is free.

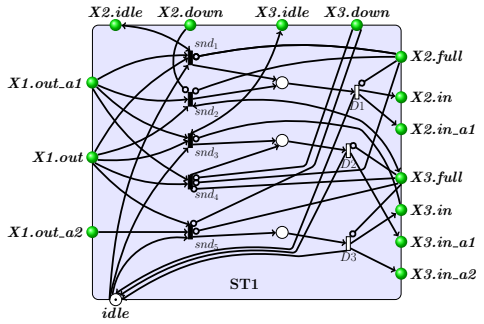


Figure 7.4: An example of *transporter class* employing selective policies for moving of pieces from multi-typed machine X_1 to limited-size buffered target machines X_2, X_3

The ***internal structure of ST1*** describes the delivery policy. The three internal places correspond to (from top to bottom) the $a1$ pieces to be delivered to $X2$, the $a1$ pieces to be delivered to $X3$ and the $a2$ pieces to be delivered to $X3$. The black transitions named $snd1$ to $snd4$ correspond to the delivering possibilities for $a1$ pieces. They are controlled by controlling places through inhibitor or regular arcs. Note that, in order to simplify the figure, the return time of the transporter is abstracted away.

7.3 A Small Example

To demonstrate the application of the compositional SPN framework we consider an example of FMS (push production) system taken from [ABC+95] (chap-

Transporter class $ST1$ moves workpieces $a1$ and $a2$ from machine $X1$ to machines $X2$ and $X3$ (to be instanciated during the linking phase). *$a1$ pieces can reach either $X2$ or $X3$; $a2$ pieces must reach $X3$.* *delivery time from $X1$ to $X2$ for $a1$ follows distribution $D1$* *delivery time from $X1$ to $X3$ for $a1$ follows distribution $D2$* *delivery time from $X1$ to $X3$ for $a2$ follows distribution $D3$* *if $X3$ is idle and not full then deliver $a1$ to $X3$,* *else if $X2$ is idle and not full, then deliver $a1$ to $X2$,* *else if $X3$ is up and not full, then deliver $a1$ to $X3$,* *else if $X2$ is up and not full then deliver $a1$ to $X2$.*

Table 7.3: Example of (informal) specialization of the transporter class

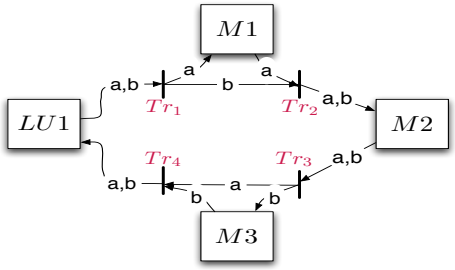


Figure 7.5: Workflow of the modeled conveyor-belt FMS with 3 machines and 2 raw-materials

ter 8). Such FMS consists of: a loading unit $LU1$ and three machines $M1, M2$ and $M3$ arranged in a linear fashion according to the workflow depicted in Figure 7.5. The FMS treats two types of material, namely a and b, which are progressively transformed into workpieces and eventually in the final products (i.e. the output of $M3$). The transportation medium is a conveyor belt, which we assume to consist of 4 adjacent belt segments (segment $LU1$-$M1$, segment $M1$-$M2$, segment $M2$-$M3$ and segment $M3$-$LU1$, each one commanded by a separate engine) The first machine in the line, i.e. $M1$, processes only material of type a (thus type b items arriving at $M1$ are bypassed to $M2$); machine $M2$ processes both type b raw material and pieces a' outputted by $M1$; finally $M3$ processes only pieces b' outputted by $M2$. The SPN components corresponding to such FMS are depicted in Figure 7.6. They consist of an LU-component (i.e. $LU1$), three M-component (i.e. $M1, M2, M3$) and four T-components (i.e. $T1, T2, T3, T4$).

In this example we assume the 4 segments of the conveyor belt to behave in a *blocking fashion*: if on reaching of a certain position an item cannot be unloaded (because the destination machine's buffer is full or because the next segment is not empty) then the belt (segment) blocks. Note that from a behavioral point of view the conveyor's segments can be distinguished into: those performing

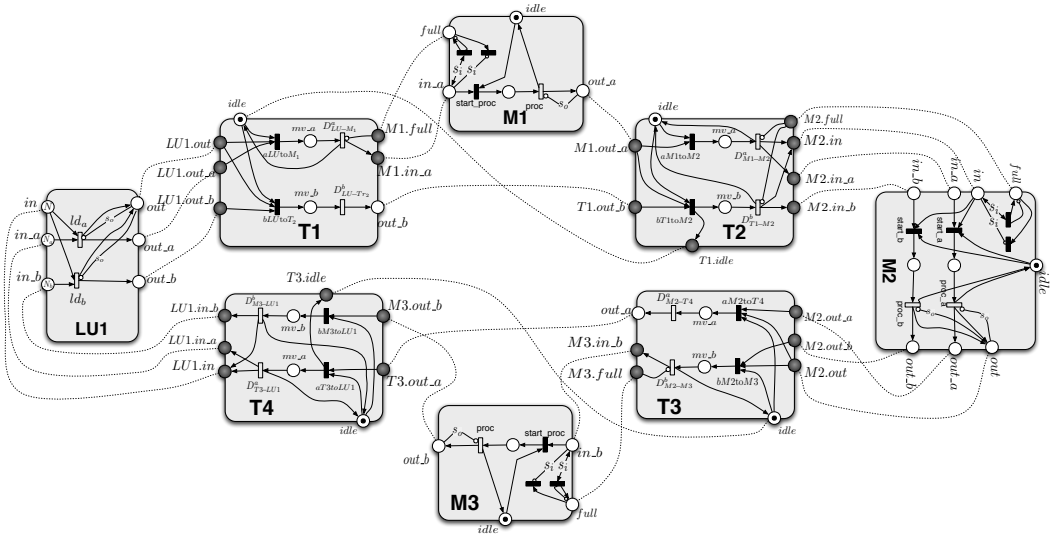


Figure 7.6: SPN components for the conveyor-belt FMS of Figure 7.5

delivering of an item (to a machine) and bypassing (to the next segment) i.e. component $T1$ and $T3$, and those performing delivering without bypassing, i.e. component $T2$ and $T4$. To better understand such a difference let's consider, for example, the (SPN) component $T1$ of Figure 7.6. If $T1$ is carrying a type a item (i.e. a token is in place $T1.mv\_a$) then it will return *idle* as soon as it drops it in the input buffer of $M1$ and this happens only if $M1$ buffer is not full (firing of transition $T1.D^a_{LU-M1}$). If $T1$ is carrying a type b item (i.e. a token is in place $T1.mv\_b$) then $T1$ will return *idle* only when $T2$ takes on the item b from $T1$, but this can happen only if $T2$ is idle: if it is carrying an item to $M3$ (i.e. a token is either in $T2.mv\_a$ or in $T2.mv\_b$) then it will be able to take on the item from $T1$ only when it has delivered the one it is carrying (i.e. firing of transition $T2.D^a_{M1-M2}$ or $T2.D^a_{T1-M2}$). In practice the dependency between a by-passing transporter (i.e. $T1$ or $T3$) with its successor (i.e. $T2$, respectively $T4$) is achieved by *importing* of a place (i.e. place *idle*) from the interface of the controlled node to the interface of the controlling one (i.e. place $T1.idle$ imported by component $T2$, and place $T3.idle$ imported by component $T4$). We consider that the unloading time of the finished workpieces in LU1 component is negligible. Thus when workpieces are returned to LU1, new raw materials are charged and new productions begin.

7.4 Fine-grained transient-analysis of FMS

7.4.1 Expressing qualitative and quantitative properties of FMS

The steady-state analysis has been the focus of many performance studies for manufacturing systems. Traditionally we are interested in customer average measures like mean fabrication time for a kind of product, time average measures like mean number of raw materials in a buffer. The relevance of transient measures for manufacturing systems has been emphasized in [NV94]. In FMSs the arrivals of raw materials to feed the input buffers, and the extraction of finished products from output buffers may be bursty. This means there may be high-activity and low-activity periods due to some external reasons like logistic problems. For such cases it is really important to observe transient behaviors which may be radically different from equilibrium (steady-state) behaviors. For instance, the buffers must be dimensioned by considering high-activity periods, and the throughput (mean number of finished products) during low-activity may be important. We can state here the case when the setting of FMS is changed, the time until the system reaches a stationary regime may be long and it may be important to observe this transient period. In FMS, the components are prone to failures or human interventions that may provoke the unavailability of some parts of the system. Such phenomena may lead to a deadlock situation or to a complete unavailability of the system. For such cases only transient measures provide some lights on FMS properties.

Using the HASL formalism, we can express interesting quantitative measures on FMS. We give here several examples. First we can characterize (and evaluate) properties related to the occupation of finite-capacity buffers, like the blocking probability for a machine, the mean time to fill $x\%$ of buffers, the mean number of pieces in buffers during a given time interval. These measures are important for an appropriate dimensioning of buffers. In order to evaluate the efficiency of the underlying FMS we are also interested in the measures related to throughput (mean number of produced workpieces per time unit), and makespan (average production time for a given production workflow). We can state for instance the probability that a certain number of workpieces are produced during a given time interval, the average time to produce a given number of workpieces. The reliability measures when some components are prone to failures can be also considered like the Mean Time To Failure (MTTF) of a component or whole system, throughput of a given production workflow between the first and the second failure.

In addition, steady state measures can also be obtained by transient analysis

when the system admits regeneration points. Indeed, the steady state measure is then the average measure between two regeneration points.

The automata of figure 7.7 illustrate some of the possibilities of HASL. The first one has two variables: x_1 is in fact a clock, reset at the occurrence of the first failure (in figure 7.3, fail should label all three transitions whose name is prefixed by $fail$), and x_2 is a counter (rate 0) that counts the number of objects processed (transition labelled out) between the two first failures. The second automaton has two variables counting the global time (x_1) and the time in state Mthre (x_2). The automaton changes state from Init to Mthre depending on whether the number of tokens in a specified buffer reaches a threshold or not. Here on the example of figure 7.6, the condition $m(M2.in) > s$ with s equal to a chosen threshold is a good candidate for indicator **thre**. After k time units, the execution reaches state End and terminates. In the third automaton, x_2 counts the number of pieces arrived so far, and x_1 the number of pieces, arrived among the k first ones, that are still waiting. The execution terminates when all of the k first pieces are being/have been served (x_1=0). Since both variables have rate 0 in each state, the rates are omitted in the figure.

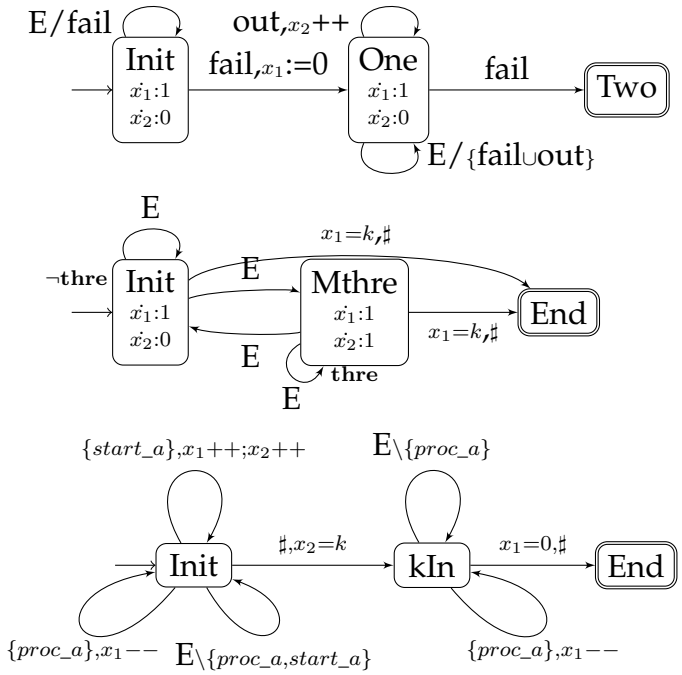


Figure 7.7: Three LHAs to compute interesting measures on FMS

The second component of a HASL formula is an expression related to the automaton. Such an expression, denoted Z, is based on moments of a path random variable Y and is defined as follows. First y is an arithmetic expression built on top of LHA data variables and constants. Then Y is a path dependent expression built on top of basic path random variables such as $last(y)$ (resp. $min(y)$, $max(y)$)i.e. the last (resp. minimum, maximum) value of y along a synchronizing path, $int(y)$ (i.e. the integral over time along a path) and $avg(y)$ (the average value of y along a path). Finally Z, the actual target of HASL verification, is an arithmetic expression built on top of the first moment of Y ($E[Y]$), and thus allowing for the consideration of diverse significant characteristics of Y including, for example, expectation, variance and covariance. Ensuring that, with probability 1, the system (SPN + LHA) will reach a final state, the expression Z associated with the formula may be evaluated with expectations defined w.r.t. the distribution of a random path *conditioned by acceptance of the path*. In other words, the LHA $\mathcal{A}$ both calculates the relevant measures during the execution and selects the relevant executions for computing the expectations. This evaluation gives the result of the formula $(\mathcal{A}, Z)$ for an SPN $\mathcal{S}$.

Given the first LHA of figure 7.7, the expected throughput between the two first failures corresponds to expression $E(last(x_2)/last(x_1))$. If we slightly modify it by considering state One as a final state, we can compute the mean time to first failure by $E(last(x_1))$ and its variance (which is often a critical parameter) by $E(last(x_1)^2) - E(last(x_1))^2$. If we consider the second LHA, the expected value of the average time (within k time units) that the input buffer of machine M is full can be computed using $E(last(x_2))$ and the ratio of the time it is full is $E(last(x_2)/last(x_1))$. For automaton 3 we can express the expected value of the mean waiting time for k products using expression $E(int(x_1)/k)$.

This logic extends the transient properties that can be expressed and verified using other stochastic logics (such as CSL, CSRL, asCSL, CSL^{TA},...) both capturing probabilistic properties of standard probabilistic model checking and also enabling to express more complex performance evaluation measures, coupled with a more precise selection of paths.

7.4.2 Automatic Generation of properties for FMS

Just as we did not want to assume that a modeller knows the Petri net formalism, this modeller should be able to verify different properties without any knowledge about hybrid automata. The goal of the automatic generation of properties for FMS is to hide this formalism and to let the user choose a property to verify in an intuitive way. The user selects a property pattern in a predefined

list (for example the mean time to fill $x\%$ of a buffer), selects the appropriate parameters (the desired buffer and the percentage) and the HASL formula (automaton + expression) is generated automatically. This generation is possible and efficient since, as we consider a predefined list of relevant properties, there is no combinatory explosion and the translation is relatively simple.

An important point to mention is that the generation of SPNs for the FMS model and of the automaton for the HASL formula are linked, since the SPN needs to be coherent in terms of labels on events and indicators.

7.5 A Complete Case Study

We give in this section a full description of the FMS case-study that we have considered. We start by giving the architecture of the system (overall view), then, we detail the description of each component (internal view). We will consider some interesting properties to evaluate followed by their numerical results and interpretations.

7.5.1 FMS Model

The system consists of a load unit, two conveyor belts and two machines. The load unit receives a unique type of materials from outside. The raw materials are oriented to two conveyor belts **TU** (figure 7.11). We consider here two routing policies (strategies) **S1** and **S2**. Load unit **LUS1** (figure 7.9) is associated with strategy S1, while, **LUS2** (figure 7.10) is associated with S2. Each conveyor belt leads the raw materials to the machine located on it downstream. We consider two types of behavior for the machines: in **M_U** (figure 7.12) the machine is supposed permanently working (never fails), in **M_UD** (figure 7.13) the machine may fail and could be repaired.

The architecture of the system is given in figure 7.8. The load unit is related to the two conveyor belts and each conveyor belt is related to a machine.

There is a small difference between the two strategies with respect to components linkage. In case of strategy S2 the load unit is not linked to the machines. So arcs from the machines to load unit do not exist.

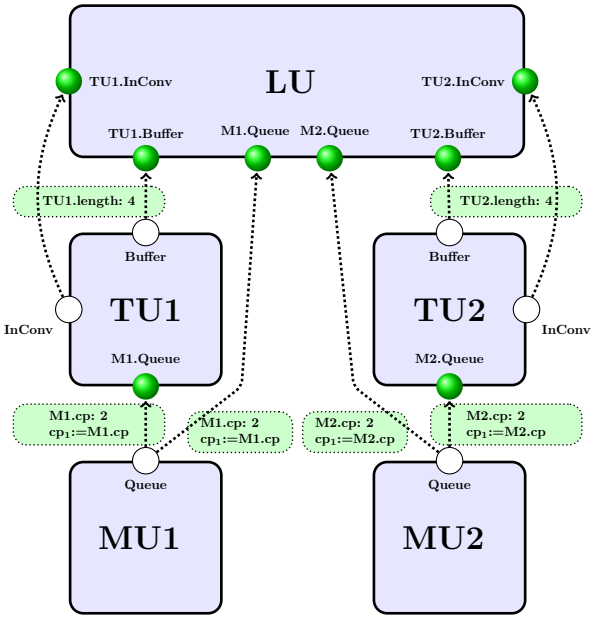


Figure 7.8: Architecture of the system

Internal Structure of the Load Units

The raw material arrive to an unbounded buffer which is represented by place *Products*. The inter-arrival times are distributed according to a uniform distribution within interval $[a1, a2]$. A uniform distribution is chosen, rather than a deterministic one, in order to represent small variabilities due to logistic problems.

When a new material arrives to place *Products*, it is immediately oriented to the buffer of one of the two conveyor belts, represented by the imported places *X1.Buffer* and *X2.Buffer*.

The imported places *X1.Buffer* and *X2.Buffer* give us the information of the quantities of materials in each conveyor belt.

The imported places *X3.Queue* and *X4.Queue* give us the information of the quantities of materials in each machine.

Let us now describe the routing policies.

Routing policies

Policy S_1.

- if one of the conveyor belts is blocked and the other is available, then put the material to the available one,
- if both conveyor belts are available and if the number of material in conveyor i is greater than a given threshold l_i while for the other conveyor is less than its own threshold l_j, put the material in the conveyor which does not exceed its threshold,
- otherwise one of the conveyor is randomly chosen with probability 0.5.

Policy S_2.

- choose the conveyor belt with the smallest number of occupied positions.
- in the case of equal number of materials in conveyors, choose randomly a conveyor with probability 0.5.

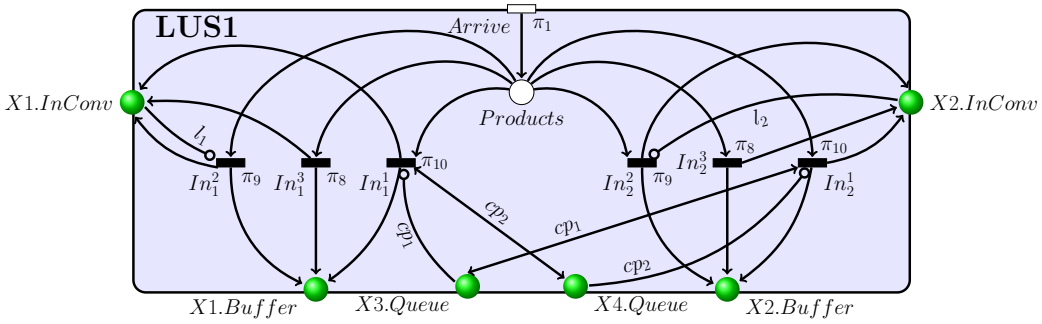


Figure 7.9: The load unit with strategy S1

Internal Structure of the Transportation Unit

The conveyor belt (component **TU**) consists of

- an unbounded input buffer, represented by places *Buffer*;

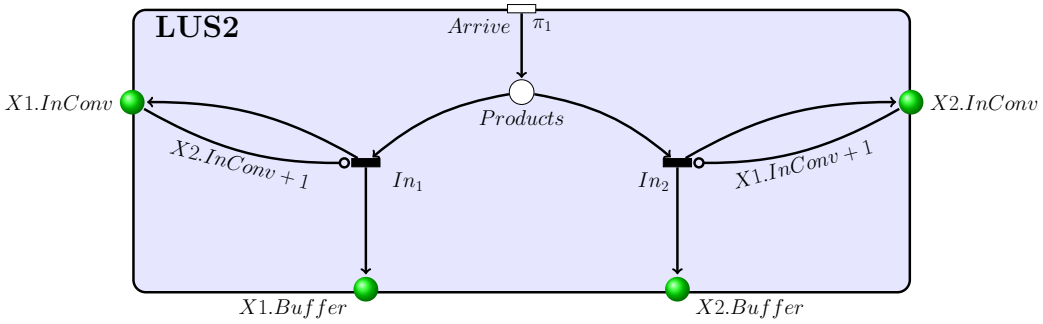


Figure 7.10: The load unit with strategy S2

- a set of equally distanced positions, represented by places Pos^j. We consider 4 positions, $j \in \{1, ..., 4\}$. Transitions Mv^j denote the movements on the conveyor. Mv^1 denotes the movement from *Buffer* to the first position of the conveyor; Mv^2 denotes the movement from the first to the second position of conveyor, and Mv^5 denotes the movement from the fourth position to the input buffer of machine located at downstream of the conveyor (represented by the imported place *X.Queue*). Transitions $Mv^j,\ j \in \{2 \cdots 5\}$ are deterministic with parameter *Tunit* which represents transport time of one pallet between two successive positions. Transition Mv^1 is immediate.

Conveyor belt drives materials to the bounded imported place *X.Queue*. When *X.Queue* reaches it capacity (*cp*) (*marking* $(X.Queue) = cp$), the conveyor belt is blocked (as ensured by inhibitor arcs of weight cp).

Internal Structure of the Machine

The service in the machine is represented by transition *Serve*. The service duration follows a log-normal distribution except in experiments reported in tables 7.7 and 7.8, where it is geometrically distributed with parameter p and step length Δ. A geometric service means that first the machine processes the product during a service period Δ and then, with probability p, the product leaves the machine successfully or, with probability $1-p$, it requests for another service period.

We consider two models. In the first one we suppose that there is no machine failure so the service is always available, we refer to this model by M_1 (component **M_U**). In the second one that we refer as M_2 (component **M_UD**),

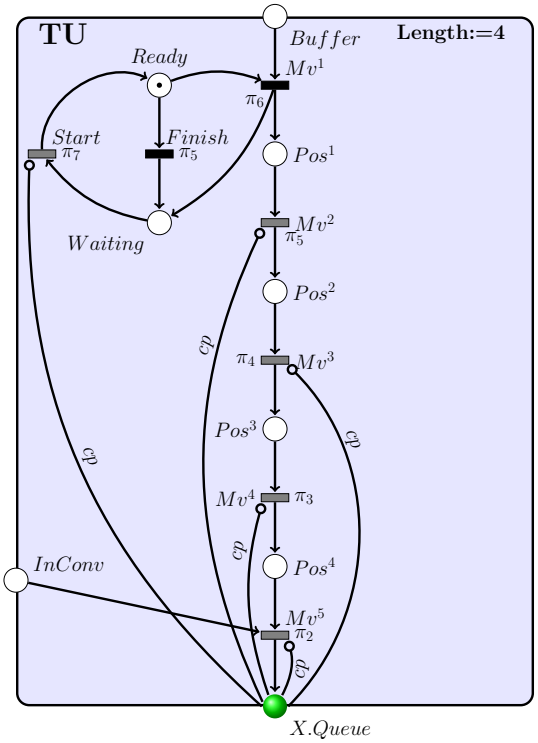


Figure 7.11: The transportation unit (conveyor belt)

we suppose a failure/repair model for each machine. The time to failures is distributed according to an *Erlang* distribution.

We assume that the failure probability increases with machine utilization and an inactive machine cannot fail. Thus we have chosen the age-memory as memory policy for transition *Fail*. The reparation time is uniformly distributed in the interval $[r1, r2]$ (transition *Repair*).

The Petri net given in figure 7.14 (resp. figure 7.16) represents a FMS with policy $S1$ (resp. policy $S2$) and non-failing machines (model M_1) while PN in figure 7.15 represents a FMS with policy $S1$ and failing/repair machines (model M_2). The PN for policy $S2$ and model M_2 is omitted here since it can be easily designed from figure 7.16 by including Fail and Repair transitions as in figure 7.15.

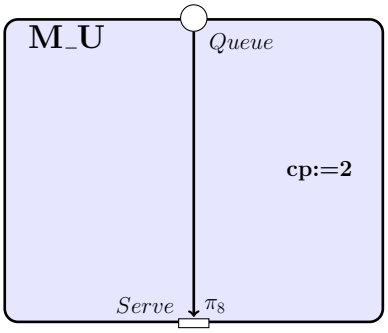


Figure 7.12: Machine without failing

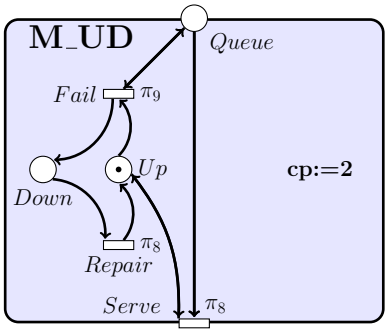


Figure 7.13: Machine fail/repair model

7.5.2 Specification of Properties

We have chosen several meaningful properties in order to assess the quality of the FMS design w.r.t. different model assumptions: routing policies, service distributions and presence of failures. We also study the behavior of COSMOS and in particular its performances and the accuracy of its results (witnessed by the width of the confidence intervals).

First we want to characterize the bottlenecks of the architecture. More precisely, a large ratio of blocking time for the conveyor may indicate that the buffer of the server should be enlarged. Furthermore if some cost is associated with the re-starting of the conveyors, decreasing this ratio can induce significant savings. So ϕ_1 (see figure 7.18) denotes the ratio of blocking time for conveyor 1. Since we study this formula in a symmetric framework the choice of the conveyor is irrelevant.

In order to support additional load due to client requests, it is important to estimate the average completion time for a product. Using Little formula (on the long run), it is equivalent to compute the expected number of products in the system which is denoted by ϕ_2 (see figure 7.19). We estimate this value depending on (1) the kind of the service distribution letting the expectation and the variance fixed and (2) the relative rate of the two machines.

The context of production of FMS implies that time is divided in periodic intervals both for logistic issues and for following the cycle of demands. This can be characterized by some threshold relative to the number of products inside an interval. Failing to meet this threshold may have dramatic consequences for the company. So ϕ_3 is the probability to produce at least K products (the threshold) in a time interval of the form $[iD, (i+1)D[$ for $0 \leq i < m$ during horizon

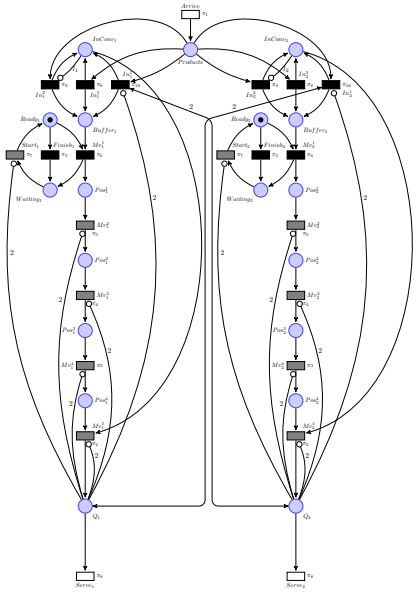


Figure 7.14: PN for policy S_1 and model M_1.

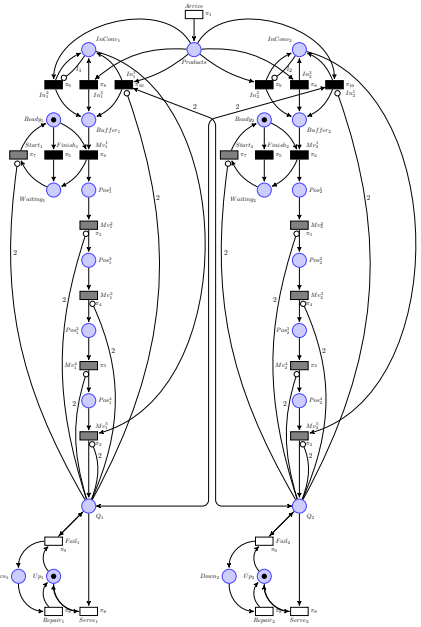


Figure 7.15: PN for policy S_1 and model M_2.

$T = mD$ (see figure 7.20).

7.5.3 Numerical Results

Unless specified otherwise, all numerical results have been obtained with a confidence interval level 0.99 and with a relative confidence interval width 10^{-3}. The transport time of pallets between two successive positions is $Tunit = 0.5$ time unit. In the following tables, T denotes the simulation horizon, S.T. denotes the simulation time in seconds, N.P. denotes the number of paths for the required accuracy of the estimation, and C.I. denotes the relative confidence interval width.

In the following experiences, the inter-arrival distribution of raw material is *Unif*$[0.45, 0.55]$. We consider two kinds of distributions for service times in machines:

- Lognormal distribution $Ln\mathcal{N}(\mu, \sigma^2)$ with scale parameter μ and shape parameter σ^2. The expectation is given by $e^{\mu+\sigma^2/2}$ and the variance by $(e^{\sigma^2} - 1)e^{2\mu+\sigma^2}$.

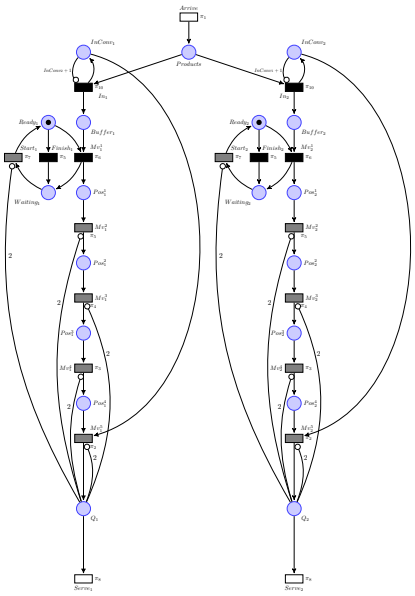


Figure 7.16: PN for policy S_2 and model M_1.

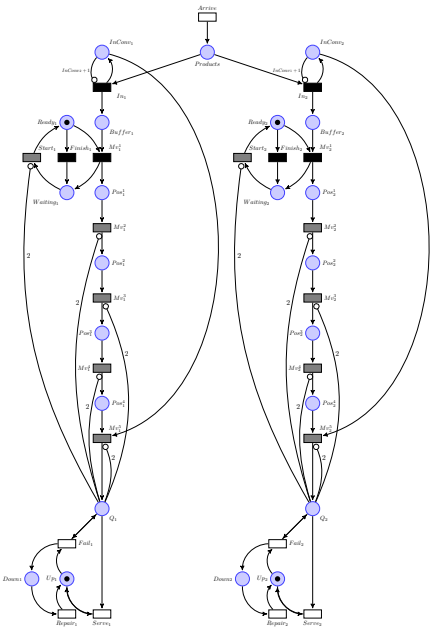


Figure 7.17: PN for policy S_2 and model M_2.

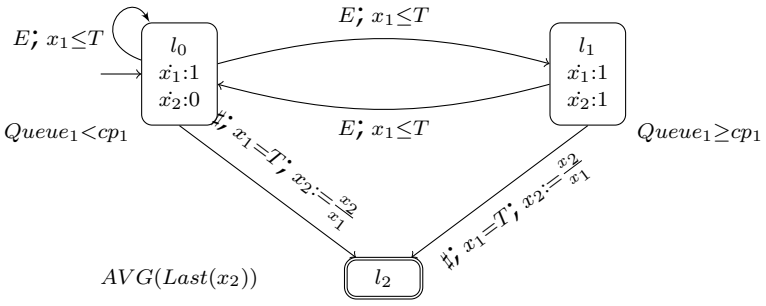


Figure 7.18: HASL formula of ϕ_1

- Geometric distribution $Geo(p, \Delta)$ with success parameter p, and service period Δ. The expectation is given by Δ/p and the variance by $\Delta^2 \frac{1-p}{p^2}$.

We first study model M_1 without machine failures.

In table 7.4, we consider property ϕ_1 with respect to horizon T under policy $S1$ with thresholds $l_1 = 3$, $l_2 = 3$ and under policy $S2$. The service rates in ma-

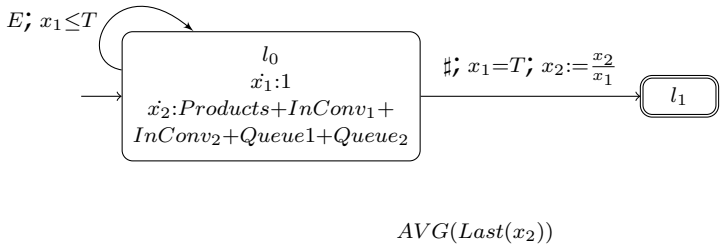


Figure 7.19: HASL formula of ϕ_2

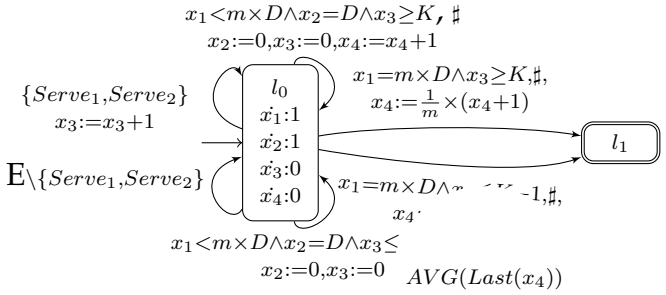


Figure 7.20: HASL formula of ϕ_3

T	$S1$			$S2$		
	ϕ_1	S.T.	N.P.	ϕ_1	S.T.	N.P.
20	0.1737	3041	7.9221e+06	0.1631	2823	8.812e+06
50	0.2185	3142	3.2186e+06	0.2125	2890	3.4693e+06
100	0.2349	3266	1.6603e+06	0.2307	2976	1.7748e+06
400	0.2475	3370	423500	0.2445	3053	448300
6400	0.2514	3406	26800	0.2488	3092	27900
25600	0.2516	3451	6800	0.2489	3059	7000
102400	0.2517	3685	1800	0.2489	3290	1900

Table 7.4: the ratio of blocking time of conveyor 1 under both policies for model M_1.

chines are symmetric and the service time distribution is $Ln\mathcal{N}(-0.683046, 0.693147)$ with expectation $1/1.4$, variance $1/1.4^2$.

We can see that, for short horizons, the sample variance is high, necessitating

a large number of generated paths. The sample variance then decreases when the simulation horizon increases, and thus the total simulation time does not increase significantly with the increase of the horizon. The relative confidence interval width is 0.0002 for all experiences in this table. We observe that the steady-state seems to be reached at $T = 6400$, and the blocking probability for conveyor 1 is slightly smaller under policy $S2$ than the blocking probability under policy $S1$.

In table 7.5 (resp. table 7.6), we present the expected number of products in the system under policy $S1$ with the thresholds $l_1 = 3$, $l_2 = 3$ (resp. policy $S2$). The service time distribution is lognormal with the same parameters as in the previous experience. We observe that the expected number of products in the system is greater under policy $S2$.

T	ϕ_2(log)	S.T.	N.P.	C.I.
10	5.1769	6	31900	0.0052
30	6.0261	7	11900	0.0060
50	6.1952	7	7600	0.0062
70	6.2670	7	5300	0.0062
100	6.3228	7	3600	0.0063
400	6.4200	8	900	0.0064
1600	6.4428	10	300	0.0056
6400	6.4473	13	100	0.0045
25600	6.4479	51	100	0.0025
102400	6.4494	203	100	0.0012

Table 7.5: the expected number of products in the system with symmetric, log-normal service distribution under policy $S1$, for model M_1.

T	ϕ_2(log)	S.T.	N.P.	C.I.
10	5.1437	5	34200	0.0051
30	6.2171	12	24200	0.0062
50	6.4420	13	15300	0.0064
70	6.5391	14	11100	0.0065
100	6.6136	15	8000	0.0066
400	6.7405	16	2100	0.0066
1600	6.7732	18	600	0.0063
6400	6.7803	24	200	0.0051
25600	6.7812	48	100	0.0042
102400	6.7830	181	100	0.0017

Table 7.6: the expected number of products in the system with symmetric, log-normal service distribution under policy $S2$, for model M_1.

In table 7.7 (resp. 7.8), we consider geometric service distribution with the same mean value and the variance as in tables 7.5 and 7.6 but with distribution *Geo*$(0.98, 0.7)$. The mean number of products in the system is slightly larger with

geometric distribution under both policies. This is due to the discrete nature of the geometric distribution.

T	ϕ_2(geo)	S.T.	N.P.	C.I.
10	5.1843	6	32200	0.0052
30	6.0491	8	14000	0.0060
50	6.2190	8	8600	0.0062
70	6.2942	8	6300	0.0062
100	6.3484	8	4300	0.0063
400	6.4435	9	1100	0.0062
1600	6.4670	10	300	0.0063
6400	6.4718	12	100	0.0055
25600	6.4745	50	100	0.0026
102400	6.4749	204	100	0.0013

Table 7.7: the expected number of products in the system with symmetric, geometric service distribution under policy $S1$, for model M_1.

T	ϕ_2(geo)	S.T.	N.P.	C.I.
10	5.1507	6	35100	0.0051
30	6.2313	11	23200	0.0062
50	6.4590	13	14900	0.0065
70	6.5596	13	10900	0.0065
100	6.6323	13	7700	0.0066
400	6.7629	14	2100	0.0067
1600	6.7935	17	600	0.0063
6400	6.7999	22	200	0.0057
25600	6.8023	43	100	0.0042
102400	6.8035	177	100	0.0020

Table 7.8: the expected number of products in the system with symmetric, geometric service distribution under policy $S2$, for model M_1.

In the following experiences, we consider lognormal service distribution and we study the impact of increasing the machines service rate. Indeed, when the response time constraints are not met for a given FMS architecture, one solution may be the replacement of one or both machines with more efficient ones. For this purpose, we evaluate different configurations in order to determine the most suitable one. In particular, we consider configurations with a fixed total service rate which is 1.5 times the original total service rate. In the asymmetric case, i.e. ϕ_2(asy), the service rate of the second machine is set to twice the rate of the first machine. In the symmetric case, i.e. ϕ_2(sym), service rates of both machines are the same and they are increased by 1.5 times the original value.

In table 7.9 we give results for ϕ_2(asy) under policy $S2$ and under policy $S1$ with different threshold values (l_1, l_2). In table 7.10 we give results for ϕ_2(sym) under policy $S1$ with $l_1 = l_2 = 3$ and under policy $S2$.

T	S2	$S1$			
-	-	(3,3)	(2,3)	(1,3)	(1,4)
10	4.9257	4.9090	4.8193	4.7790	4.7537
30	6.0416	5.8290	5.6631	5.6259	5.5548
50	6.2745	6.0173	5.8374	5.7993	5.7195
70	6.3787	6.1009	5.9133	5.8742	5.7901
100	6.4525	6.1609	5.9685	5.9299	5.8423
400	6.5865	6.2671	6.0622	6.0283	5.9355
1600	6.6193	6.2975	6.0897	6.0516	5.9581
6400	6.6271	6.3036	6.0966	6.0601	5.9625
25600	6.6298	6.3036	6.0963	6.0606	5.9651
102400	6.6281	6.3039	6.0964	6.0614	5.9645

Table 7.9: $\phi_2(asy)$: the expected number of products in the system with lognormal service distribution, asymmetric service rates ($\mu_1 = 1.4, \mu_2 = 1.4$) under policy $S2$ and under policy $S1$ with different thresholds for model M_1.

T	$S2$	$S1$
10	4.7897	4.7889
30	5.7242	5.5996
50	5.9193	5.7655
70	6.0019	5.8377
100	6.0662	5.8911
400	6.1716	5.9837
1600	6.1991	6.0071
6400	6.2094	6.0122
25600	6.2052	6.0126
102400	6.2063	6.0133

Table 7.10: $\phi_2(sym)$: the expected number of products in the system with lognormal service distribution, symmetric service rates ($\mu_1 = 2.1$, $\mu_2 = 2.1$) under policy $S2$ and under policy $S1$ with thresholds $l_1 = 3$, $l_2 = 3$ for model M_1.

By comparing results in Tables 7.5, 7.6, 7.9 and 7.10, we observe that obviously increasing the service rate reduces the mean number of products in the system. However, the improvement is not significant for policy $S1$ if the thresholds are not chosen so to compensate the effect of asymmetric service time (comparison of Tables 7.5, 7.9 (column (3,3)). On the other hand asymmetric thresholds $(l_1, l_2) = (2, 3)$, $(l_1, l_2) = (1, 3)$, $(l_1, l_2) = (1, 4)$ (last three columns in Table 7.9) provide better results. Similarly for policy $S2$, the improvement is not significant for the asymmetric service rate increase $\phi_3(asym)$ (comparison of Tables 7.6, 7.9 (column S_2)).

Finally, we compare asymmetric configuration of the $S1$ policy with asymmetric thresholds $(l_1, l_2) = (1, 4)$ (Table 7.9) against the symmetric configuration with symmetric thresholds $(l_1, l_2) = (3, 3)$ (Table 7.10). We observe that there's no significant difference (in terms of expected number of products) be-

tween these two configurations. Such results may be useful during the cost-contribution analysis of FMS designing. In the case we consider here, for example, the designer knows that investing on a single twice-faster machine (i.e. asymmetric configuration) is, performance-wise, as convenient as investing on a pair of 50% faster machines (i.e. symmetric configuration). Thus he/she can opt for either possibility based on machine costs only.

We observe that symmetric increase gives better results than the asymmetric increase for policy $S2$. Furthermore policy $S1$ is better than policy $S2$ for all experiences w.r.t. the expected number of products in the system. However the blocking probability is higher under policy $S1$. Thus we can conclude that the blocking is not a problem as long as it does not induce an extra cost.

In remaining tables, we compare models M_1 and M_2. Failures occur according to an *Erlang* distribution of 4 stages of exponential distribution with mean value 250, *Erlang*$(4, 250)$, while repair time follows a uniform distribution *Unif*$(30, 50)$. Thus the mean time to failure is 1000 while mean repair time is 40 time units. First we repeat in tables 7.11 and 7.12 the experiences of tables 7.5 and 7.6 for model M_2.

T	ϕ_2(log)	S.T.	N.P.	C.I.
10	5.1750	7	31500	0.0051
30	6.0260	7	12200	0.0060
50	6.1957	8	7300	0.0061
70	6.2690	8	5300	0.0062
100	6.3220	8	3700	0.0062
400	6.5242	1545	177900	0.0065
1600	7.5133	16710	318000	0.0075
6400	7.9791	10289	78100	0.0080
25600	8.0947	10592	19900	0.0081
102400	8.1223	10512	5000	0.0081

Table 7.11: the expected number of products in the system with symmetric, lognormal service distribution under policy $S1$, for model M_2 with failures/repairs.

In short horizons, models M_1 and M_2 have similar behaviors. For horizons $400 - 1600$, when failures begin to occur, the number of paths increases significantly and then it decreases for larger horizons. As expected, in large horizons, there are more products in the system in model M_2 with failures but, contrarily to M_1 model, it seems that policy $S2$ is better than policy $S1$ for model M_2.

In tables 7.13 and 7.14 we consider lognormal service distribution with symmetric service rates in each machine ($\mu_1 = \mu_2 = 1.4$). In table 7.13 (resp. table 7.14) we present results for property ϕ_3 with the required number of productions $K = 95$ during each time interval $D = 50$, for model M_1 (resp. for model

T	ϕ_2(log)	S.T.	N.P.	C.I.
10	5.1432	5	34300	0.0051
30	6.2164	13	24100	0.0062
50	6.4431	14	15500	0.0064
70	6.5395	15	11300	0.0065
100	6.6140	15	8200	0.0066
400	6.7935	577	79100	0.0068
1600	7.2612	2708	92300	0.0073
6400	7.4717	2344	19400	0.0075
25600	7.5202	2254	4800	0.0075
102400	7.5343	2253	1200	0.0073

Table 7.12: the expected number of products in the system with logNormal, symmetric service distribution under policy $S2$, for model M_2 with failures/repairs.

$M2$ with failures). Policy $S1$ is better than policy $S2$ for model M_1 while $S2$ is better for model M_2. Thus policy $S2$ is more robust when FMS is subject to failures.

T	$S1$			$S2$		
	ϕ_3	S.T.	C.I.	ϕ_3	S.T.	C.I.
100	0.4880	4690	0.0005	0.4815	3886	0.0005
200	0.7157	2395	0.0007	0.7111	2106	0.0007
400	0.8296	1858	0.0008	0.8254	1693	0.0008
800	0.8867	1651	0.0009	0.8831	1616	0.0008
1600	0.9145	1604	0.0009	0.9117	1458	0.0009
3200	0.9289	1523	0.0009	0.9259	1520	0.0009
6400	0.9364	1543	0.0009	0.9326	1415	0.0009
12800	0.9397	1510	0.0009	0.9364	1606	0.0009
25600	0.9415	1535	0.0009	0.9383	1406	0.0009
51200	0.9422	1515	0.0009	0.9394	1466	0.0009
102400	0.9428	1516	0.0009	0.9397	1374	0.0009

Table 7.13: the probability to complete at least $K = 95$ productions during a time interval $D = 50$ with symmetric, lognormal service distribution under both policies, for model M_1.

T	$S1$			$S2$		
	ϕ_3	S.T.	C.I .	ϕ_3	S.T.	C.I.
100	0.4881	5082	0.0005	0.4816	4646	0.0005
200	0.7146	2707	0.0007	0.7103	2394	0.0007
400	0.8224	2389	0.0008	0.8220	1966	0.0008
800	0.8615	2833	0.0008	0.8703	2091	0.0009
1600	0.8662	2979	0.0009	0.8868	2072	0.0009
3200	0.8686	2799	0.0008	0.8950	2062	0.0009
6400	0.8700	2760	0.0009	0.8992	2005	0.0009
12800	0.8704	2741	0.0009	0.9014	2105	0.0009
25600	0.8707	2882	0.0009	0.9023	1982	0.0009
51200	0.8710	2797	0.0009	0.9026	2149	0.0009
102400	0.8711	2739	0.0009	0.9030	2261	0.0009

Table 7.14: the probability to complete at least $K = 95$ productions during a time interval $D = 50$ with symmetric, lognormal service distribution under both policies, for model M_2.

7.6 Conclusion

We have presented here a compositional modeling framework for flexible manufacturing systems using stochastic Petri nets. The FMS is modeled piece-wise by specifying the classes of components to be used (loading unit, transporters, machines), specifying their parameters (type of raw material needed/produced, size of input/output buffers, transporting policy, ...) and then combining all these components via their interface.

In order to evaluate these FMS, we then use HASL the stochastic language that we have presented in chapter 5. This language enables a precise selection of successful path by synchronizing the SPN with an hybrid automaton, and then a quantitative evaluation using an expression that can express both model checking (the probability of the set of winning paths, ...) and performance evaluation (mean waiting time, ...) measures.

These two steps are meant to be facilitated for the modeler. The goal is to generate both the SPN for the FMS and the automaton + expression for the formula in an automated way, not requiring the modeler to be familiar to either type of models. We aim at an analysis that is both formal, using the COSMOS tool for evaluating HASL formulas on FMS, and user oriented, providing the user with an easy way to describe his model and specify the useful formulas.

We also applied our approach on an FMS model with several interesting features.

Chapter 8

Conclusion and Perspectives

We proposed in this thesis several contributions devoted to probabilistic verification. These contributions aim at: (1) designing efficient verification procedures, (2) obtaining a more expressive logic for expressing properties, (3) considering general stochastic processes, and (4) providing toolkits for modelers for specification and analysis.

In chapter 4, our work focused on obtaining stochastic bounds for censored Markov chains. The framework of censored Markov chains is useful in two cases: (1) when the original chain is very large and we would like to truncate it, and (2) when the original chain is only partially known. Our contribution focused on the establishment of several schemes of bounds adapted to the amount of information available on the chain. We also proved the optimality of the first proposed algorithm (DPY).

In chapter 5, we extend the scope of statistical verification. More precisely, we developed a logic, called HASL, that is, to our knowledge, the most expressive of existing logics. A formula in HASL is expressed using a linear hybrid automaton and an expression defined on the variables of the automata. Using HASL one can express path expressions thanks to appropriate operators. Moreover HASL allows a unified specification of functional and performance properties. Our verification method applies on a large class of stochastic processes called discrete events stochastic processes (DESP).

In chapter 6, we presented COSMOS, the tool that we developed to implement the statistical verification of HASL formulas over DESP. In particular, we designed several optimizations in order to decrease its execution time. We also tested it on several models and formulas.

In chapter 7, we presented a method of compositional modeling for flexi-

ble manufacturing systems (FMS) based on a toolbox of components (load unit, transporter, machine). The modeler that designs a FMS, starts by selecting the components that he needs, then he determines the parameters of each component and finally he assembles the different components via their interface to obtain his model. This approach does not require any knowledge of Petri nets by the modeler. So it facilitates the modeling task and saves time related to the lifecycle of the design process.

Perspectives

Markov decision processes (MDP) are an extension of Markov chains that also include non deterministic features represented by actions whose effect is probabilistic. Eliminating the non determinism by the choice of a strategy yields a stochastic process. So problems associated with MDP consist to find optimal strategies related to the probability of some event. MDP techniques are generally highly time and space consuming. So we plan to design approximate iterative analysis based on statistical verification.

The input formalism of COSMOS is a GSPN. This formalism is an intermediate formalism more compact than Markov chain but still lacking some mechanisms to concisely express complex systems. High-level Petri nets have been introduced to address this problem. In particular stochastic well-formed nets (SWN) allow to represent complex information on the net while exploiting the symmetries to obtain efficient performance evaluation. Our second goal is to integrate SWN in our tool COSMOS.

Faced to the computation of the probability of a rare event, statistical verification requires to generate a huge number of paths to get a satisfactory estimation. Generally this number is so large that the estimated execution time forbids its use. Several methods were proposed to address the problem, among them splitting and importance sampling methods. We plan to adapt some rare event methods for HASL and to integrate them in COSMOS.

Bibliography

[ABC84] M. Ajmone Marsan, G. Balbo, and G. Conte. A class of generalized of stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on computer systems*, 2(2):93–122, May 1984.

[ABC+95] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. J. Wiley, 1995.

[ACD91] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for probabilistic real-time systems. In *18th Int. Coll. on Automata, Languages and Programming (ICALP'91)*, volume 510 of *LNCS*, pages 115–126. Springer, 1991.

[ACHH92] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, volume 736 of *LNCS*, pages 209–229. Springer, 1992.

[AD10] Elvio Gilberto Amparore and Susanna Donatelli. Qest 2010, seventh international conference on the quantitative evaluation of systems, williamsburg, virginia, usa, 15-18 september 2010. In *QEST*. IEEE Computer Society, 2010.

[AHK03] S. Andova, H. Hermanns, and J-P. Katoen. Discrete-time rewards model-checked. In *Formal Modelling and Analysis of Timed Systems (FORMATS 2003)*, volume 2791 of *LNCS*, pages 88–103. Springer Verlag, 2003.

[ASVB96] A. Aziz, K. Sanwal, V.Singhal, and R.K. Brayton. Verifying continuous-time Markov chains. In *8th Int. Conf. on Computer Aided Verification (CAV'96)*, number 1102 in LNCS, pages 269–276, New Brunswick, NJ, USA, 1996. Springer Verlag.

[ASVB00] A. Aziz, K. Sanwal, V.Singhal, and R.K. Brayton. Model chcking continuous-time Markov chains. *ACM Transactions on Computational Logic*, 1(1):162–170, 2000.

[BBB+10] A. Basu, S. Bensalem, M. Bozga, B. Delahaye, A. Legay, and E. Sifakis. Verification of an afdx infrastructure using simulations and probabilities. In *Runtime Verification - First International Conference, RV 2010*, volume 6418 of *LNCS*, pages 330–344. Springer, 2010.

[BCDK00] P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper. Complexity of Kronecker operations on sparse matrices with applications to solution of Markov models. *INFORMS Journal on Computing*, 12(3):203–222, 2000.

[BCH+07] C. Baier, L. Cloth, B. Haverkort, M. Kuntz, and M. Siegle. Model checking action- and state-labelled Markov chains. *IEEE Transactions on Software Engineering*, 33(4):209–224, 2007.

[BCH+10] C. Baier, L. Cloth, B. Haverkort, H. Hermanns, and J-P. Katoen. Performability assessment by model checking of Markov reward models. *Formal Methods in System Design*, 36:1–36, 2010.

[BDD+11a] Paolo Ballarini, Hilal Djafri, Marie Duflot, Serge Haddad, and Nihal Pekergin. COSMOS: a statistical model checker for the hybrid automata stochastic logic. In *Proceedings of the 8th International Conference on Quantitative Evaluation of Systems (QEST'11)*, pages 143–144, Aachen, Germany, September 2011. IEEE Computer Society Press.

[BDD+11b] Paolo Ballarini, Hilal Djafri, Marie Duflot, Serge Haddad, and Nihal Pekergin. HASL: An expressive language for statistical verification of stochastic models. In *Proceedings of the 5th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS'11)*, pages 306–315, Cachan, France, May 2011.

[BDD+11c] Paolo Ballarini, Hilal Djafri, Marie Duflot, Serge Haddad, and Nihal Pekergin. Petri nets compositional modeling and verification of flexible manufacturing systems. In *Proceedings of the 7th IEEE Conference on Automation Science and Engineering (CASE'11)*, pages 588–593, Trieste, Italy, August 2011. IEEE Robotics & Automation Society.

[BDF10] A. Busic, H. Djafri, and J-M. Fourneau. Stochastic bounds for censored Markov chains. In Michele Benzi and Tugrul Dayar, editors, *Proceedings of the 6th International Workshop on the Numerical Solution of Markov Chain (NSMC'10)*, Williamsburg, Virginia, USA, September 2010.

[BDK01] E. Best, R. Devillers, and M. Koutny. *Petri net algebra*. Monographs in Theoretical Computer Science. Springer, 2001.

[BE91] L. J. Bain and M. Engelhardt. *Introduction to Probability and Mathematical Statistics (second edition)*. Duxbury Classic Series, 1991.

[BF08] A. Busic and J.-M. Fourneau. Stochastic bounds for partially generated markov chains: an algebraic approach. In *Proc. of the Fith European Performance Engineering Workshop, EPEW '08*, volume 5261 of *LNCS*, pages 227–241. Springer, 2008.

[BFMP09] P. Ballarini, M. Forlin, T. Mazza, and D. Prandi. Efficient parallel statistical model checking of biochemical networks. In *Proceedings 8th International Workshop on Parallel and Distributed Methods in Verification*, volume 14 of *EPTCS*, pages 47–61, 2009.

[BGdMT98] G. Bolch, S. Greiner, H. de Meer, and K.S. Trivedi. *Queueing networks and Markov chains*. John Wiley & Sons, New-York, 1998.

[BGS01] F. Balduzzi, A. Giua, and C. Seatzu. Modelling and simulation of manufacturing systems with first-order hybrid Petri nets. *Inter. J. of Production Research*, 39(2):255–282, 2001.

[BHHK00a] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. On the logical characterisation of performability properties. In *27th International Colloquium, ICALP 2000*, volume 1853 of *LNCS*, pages 780–792. Springer, 2000.

[BHHK00b] C. Baier, B.R.H.M. Haverkort, H. Hermanns, and J.P. Katoen. Model checking continuous-time markov chains by transient analysis. In E.A. Emerson and A.P. Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 358–372, Berlin, 2000. Springer Verlag.

[BHHK03a] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous time Markov chains. *IEEE Transactions on Software Engineering*, 29(7):524–541, July 2003.

[BHHK03b] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 29(7):2003, 2003.

[BHKW03] C. Baier, H. Hermanns, J.-P. Katoen, and V. Wolf. Comparative branching-time semantics for Markov chains. In *Concurrency Theory (CONCUR 2003)*, volume 2761 of *LNCS*, pages 492–507. Springer Verlag, 2003.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[BKKT02] Peter Buchholz, Joost-Pieter Katoen, Peter Kemper, and Carsten Tepper. Model-checking large structured markov chains, 2002.

[BOO] The boost c++ libraries. http://www.boost.org/.

[Can88] J. Canny. Some algebraic and geometric computations in pspace. In *20th ACM Symposium on Theory of Computing (STOC'88)*, pages 460–467. ACM, 1988.

[CBB89] P. Chen, S. C. Bruell, and G. Balbo. Alternative methods for incorporating non exponential distributions into stochastic timed petri nets. In *Third International Workshop on Petri Nets and Performance Models*, pages 187–197. IEEE Computer Society Press, 1989.

[CDL10] E. M. Clarke, A. Donzé, and A. Legay. On simulation-based probabilistic model checking of mixed-analog circuits. *Formal Methods in System Design*, 36(2):97–113, 2010.

[CE82] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, UK, 1982. Springer-Verlag.

[CFGR95] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. Greatspn 1.7: Graphical editor and analyzer for timed and stochastic petri nets. *Perform. Eval.*, 24(1-2):47–68, 1995.

[CFM+97] E.M. Clarke, M. Fujita, P. McGeer, K. McMillan, J. Yang, and X. Zhao. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10(2/3):149–169, 1997.

[CHKM09] T. Chen, T. Han, J.-P. Katoen, and A. Mereacre. Quantitative model checking of continuous-time Markov chains against timed automata specifications. In *Proc. of the 24th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 309–318. IEEE, 2009.

[CKKP05] L. Cloth, J.-P. Katoen, M. Khattri, and R. Pulungan. Model checking Markov reward models with impulse rewards. In *Int. Conference on Dependable Systems and Networks (DSN 2005)*, pages 722–731, 2005.

[COL] Coloane. http://move.lip6.fr/software/coloane/.

[Cox55] D. R. Cox. A use of complex probabilities in the theory of stochastic processes. *Proc. Cambridge Philosophical Society*, pages 313–319, 1955.

[CSS03] J.-M. Couvreur, N. Saheb, and G. Sutre. An optimal automata approach to LTL model checking of probabilistic systems. In *In Proc. 10th Int. Conf. Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'2003)*, volume 2850 of *LNAI*, pages 361–375. Springer Verlag, september 2003.

[CT96] G. Ciardo and M. Tilgner. On the use of Kronecker operators for the solution of generalized stochastic Petri nets. Technical report 96-35, ICASE, Institute for Computer Applications in Science and Engineering, NASA/Langley Research Center, Hampton, VA, USA, May 1996.

[Cum85] A. Cumani. Esp - a package for the evaluation of stochastic petri nets with phase-type distributed transition times. In *International Workshop on Timed Petri Nets*, pages 144–151. IEEE Computer Society Press, 1985.

[CY95] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, july 1995.

[DFPV06] T. Dayar, J.-M. Fourneau, N. Pekergin, and J.-M. Vincent. Polynomials of a stochastic matrix and strong stochastic bounds. In *Markov Anniversary Meeting*, pages 211–228. Boson Books, 2006.

[DHM98] S. Donatelli, S. Haddad, and P. Moreaux. Structured characterization of the markov chains of phase-type spn. In *10th International Conference on Computer Performance Evaluation. Modelling Techniques and Tools*, volume 1469 of *LNCS*, pages 243–254. Springer, 1998.

[DHS09] S. Donatelli, S. Haddad, and J. Sproston. Model checking timed and stochastic properties with CSL^{TA}. *IEEE Transactions on Software Engineering*, 35:224–240, 2009.

[DHSW10] T. Dayar, H. Hermanns, D. Spieler, and V. Wolf. Bounding the equilibrium distribution of markov population models. In *Proc. of the Sixth International Workshop on the Numerical Solutions of Markov Chains, NSMC '10*, pages 40–43, 2010.

[DLL+11] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, Danny Bøgsted Poulsen, Jonas van Vliet, and Zheng Wang. Statistical model checking for networks of priced timed automata. In *Formal Modeling and Analysis of Timed Systems - 9th International Conference, FORMATS 2011, Aalborg, Denmark*, volume 6919 of *Lecture Notes in Computer Science*, pages 80–96. Springer, 2011.

[Don93] S. Donatelli. Superposed stochastic automata: A class of stochastic Petri nets with parallel solution and distributed state space. *Perform. Eval.*, 18(1):21–36, 1993.

[Don94] S. Donatelli. Superposed generalized stochastic Petri nets: definition and efficient solution. In Robert Valette, editor, *Proc. of the 15th International Conference on Application and Theory of Petri Nets*, volume 815 of *LNCS*, pages 258–277. Springer–Verlag, 1994.

[DPY06] T. Dayar, N. Pekergin, and S. Younès. Conditional steady-state bounds for a subset of states in Markov chains. In *SMCtools '06*. ACM Press, 2006.

[dSeSO92] E. de Souza e Silva and P. M. Ochoa. State space exploration in Markov models. *ACM SIGMETRICS Perform. Eval. Rev.*, 20(1):152–166, 1992.

[DSP] The dspn tool. http://www.di.unito.it/ amparore/dspn-tool/.

[EC80] E. A. Emerson and E. M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, volume 85 of *LNCS*, pages 169–181, 1980.

[ECM95] J. Ezpeleta, J.M. Colom, and J. Martinez. A Petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE transactions on robotics and automation*, 4(2):173–184, 1995.

[EP09] D. El Rabih and N. Pekergin. Statistical model checking using perfect simulation. In *ATVA*, volume 5799 of *LNCS*, pages 120–134. Springer, 2009.

[Fel68] W. Feller. *An introduction to probability theory and its applications. Volume I*. John Wiley & Sons, 1968. (third edition).

[Fel71] W. Feller. *An introduction to probability theory and its applications. Volume II*. John Wiley & Sons, 1971. (second edition).

[FN78] G. Florin and S. Natkin. Évaluation des performances d'un protocole de communication à l'aide des réseaux de Petri et des processus stochastiques. In *Journées AFCET Multi-ordinateurs, multiprocesseurs en temps réel*, CNRS, Paris, France, mai 1978.

[FN85] G. Florin and S. Natkin. Les réseaux de Petri stochastiques. *TSI*, 4(1):143–160, 1985.

[FN89] G. Florin and S. Natkin. Necessary and sufficient ergodicity condition for open synchronized queuing networks. *IEEE Transactions on software engineering*, 15(4):367–380, 1989.

[FP88] J.-M. Fourneau and B. Plateau. PEPS: A package for solving complex Markov models of parallel systems. In *Proceedings of the Fourth International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 291–305, Spain, 1988.

[FPS98] P. Fernandes, B. Plateau, and W. J. Stewart. Efficient descriptor-vector multiplications in stochastic automata networks. *J. ACM*, 45(3):381–414, 1998.

[FPSS06] J.-M. Fourneau, B. Plateau, I. Sbeity, and W. J. Stewart. SANs and lumpable stochastic bounds: Bounding availability. In *Computer System, Network Performance and Quality of Service*. Imperial College Press, 2006.

[FPY07a] J. M. Fourneau, N. Pekergin, and S. Younès. Censoring markov chains and stochastic bounds. In *Proceedings of the 4th European performance engineering conference on Formal methods and stochastic models for performance evaluation*, EPEW'07, pages 213–227, Berlin, Heidelberg, 2007. Springer-Verlag.

[FPY07b] J.M. Fourneau, N. Pekergin, and S. Younès. Censoring markov chains and stochastic bounds. In K. Wolter, editor, *Formal Methods*

and Stochastic Models for Performance Evaluation, EPEW '07, volume 4748 of *LNCS*, pages 213–227. Springer, 2007.

[FQ95] J.-M. Fourneau and F. Quessette. Graphs and stochastic automata networks, Jan. 1995.

[GCC] The gnu compiler collection. http://gcc.gnu.org/.

[GHH02] Roberto Gorrieri, Ulrich Herzog, and Jane Hillston. Unified specification and performance evaluation using stochastic process algebras. *Perform. Eval.*, 50(2/3):79–82, 2002.

[Gly83] P. W. Glynn. On the role of generalized semi-Markov processes in simulation output analysis. In *Proceedings of the 15th conference on Winter simulation*, volume 1, pages 38–42, 1983.

[GNU] The gnu make. http://www.gnu.org/s/make/.

[Gre] The greatspn tool. http://www.di.unito.it/ greatspn/index.html.

[Hav93] B.R. Haverkort. Approximate performability and dependability modelling using generalized stochastic Petri nets. *Performance Evaluation*, 18(1):61–78, 1993.

[Hav95] B.R. Haverkort. Matrix-geometric solution of infinite stochastic Petri nets. In *Proc. of the International Performance and Dependability Symposium*, pages 72–81. IEEE Computer Society Press, 1995.

[Hil96] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[HJ94] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:102–111, 1994.

[HJB+10] R. He, P. Jennings, S. Basu, A. P. Ghosh, and H. Wu. A Bounded Statistical Approach for Model Checking of Unbounded Until Properties. In *Int. Conf. on Automated Software Engineering (ASE'10)*, pages 225–234. IEEE/ACM, 2010.

[HK01] J. Hillston and L. Kloul. An efficient Kronecker representation for PEPA models. In *Proceedings of the Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*, PAPM-PROBMIV '01, pages 120–135. Springer-Verlag, 2001.

[HL10] F. S. Hillier and G. J. Lieberman. *Introduction to Operations Research (ninth edition)*. McGraw-Hill, 2010.

[HLR00] Günter Haring, Christoph Lindemann, and Martin Reiser, editors. *Performance Evaluation: Origins and Directions*, London, UK, UK, 2000. Springer-Verlag.

[HM09a] S. Haddad and P. Moreaux. Stochastic Petri nets. In Michel Diaz, editor, *Petri Nets: Fundamental Models, Verification and Applications*, pages 269–302. Wiley-ISTE, 2009.

[HM09b] S. Haddad and P. Moreaux. Stochastic well-formed Petri nets. In Michel Diaz, editor, *Petri Nets: Fundamental Models, Verification and Applications*, pages 303–320. Wiley-ISTE, 2009.

[HM09c] S. Haddad and P. Moreaux. Tensor methods and stochastic Petri nets. In Michel Diaz, editor, *Petri Nets: Fundamental Models, Verification and Applications*, pages 321–346. Wiley-ISTE, 2009.

[HMC97] S. Haddad, P. Moreaux, and G. Chiola. Efficient handling of phase-type distributions in generalized stochastic Petri nets. In *18th International Conference on Application and Theory of Petri Nets*, volume 1248 of *LNCS*, pages 175–194. Springer, 1997.

[HMN11] S. Haddad, J. Mairesse, and H-T. Nguyen. Synthesis and analysis of product-form Petri nets. In *ICATPN'11*, volume 6709 of *LNCS*, pages 288–307. Springer, 2011.

[HMSM05] S. Haddad, P. Moreaux, M. Sereno, and M.Silva. Product-form and stochastic Petri nets: a structural approach. *Performance Evaluation*, 59/4:313–336, 2005.

[HMSS01] S. Haddad, P. Moreaux, M. Sereno, and M. Silva. Structural characterization and behavioural properties of product form stochastic Petri nets. In *Proc. of the 22th International Conference on Application and Theory of Petri Nets*, volume 2075 of *LNCS*, pages 164–183. Springer, 2001.

[HMSS05] S. Haddad, P. Moreaux, M. Sereno, and M. Silva. Product-form and stochastic Petri nets: a structural approach. *Perform. Eval.*, 59:313–336, 2005.

[HPTvD90] W. Henderson, C.E.M. Pearce, P.G. Taylor, and N.M. van Dijk. Closed queueing networks with batch services. *Queueing Systems*, 6(2):59–70, 1990.

[HS86] Sergiu Hart and Micha Sharir. Probabilistic propositional temporal logics. *Inf. Control*, 70(2-3):97–155, August 1986.

[Jen53] A. Jensen. Markov chains as an aid in the study of Markov processes. *Skand. Aktuarietidskrift*, 3:87–91, 1953.

[JKO+07] David N. Jansen, Joost-Pieter Katoen, Marcel Oldenkamp, Mariëlle Stoelinga, and Ivan S. Zapreev. How fast and fat is your probabilistic model checker? an experimental performance comparison. In *Hardware and Software: Verification and Testing, Third International Haifa Verification Conference, HVC 2007, Haifa, Israel*, volume 4899 of *Lecture Notes in Computer Science*, pages 69–85, 2007.

[Kel79] F.P. Kelly. *Reversibility and stochastic networks*. John Wiley & Sons, England, 1979.

[Kle75] L. Kleinrock. *Queueing systems. Volume I: Theory*. Wiley-Interscience, New-York, 1975.

[KNP11] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011.

[KS60] J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. D. Van Nostrand-Reinhold, New York, NY, 1960.

[Kwi07] M. Kwiatkowska. Quantitative verification: Models, techniques and tools. In *Proc. 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 449–458. ACM Press, September 2007.

[LCGH93] C. Lindemann, G. Ciardo, R. German, and G. Hommel. Performabilty modeling of an automated manufacturing system with deterministic and stochastic Petri nets. In *ICRA (3)*, pages 576–581, 1993.

[Led60] J. Ledoux. *Weak lumpability of finite Markov chains and positive invariance of cones*. research report INRIA-IRISA n° 2801, 1960.

[MdSeSG89] R. R. Muntz, E. de Souza e Silva, and A. Goyal. Bounding availability of repairable computer systems. *IEEE Trans. on Computers*, 38(12):1714–1723, 1989.

[Mey80] J.F. Meyer. On evaluating the performability of degradable computing systems. *IEEE Transactions on Computers*, 29(8):720–731, August 1980.

[Mey08] S. Meyn. *Control techniques for complex networks*. Cambridge University Press, Cambridge, 2008.

[MN10] J. Mairesse and H-T. Nguyen. Deficiency zero Petri nets and product form. *Fundamenta Informaticae*, 105(3):237–261, 2010.

[Mol81] M. K. Molloy. *On the integration of delay and throughput in distributed processing models*. PhD dissertation, University of California, Los Angeles, CA, USA, September 1981.

[MS02] A. Muller and D. Stoyan. *Comparison Methods for Stochastic Models and Risks*. Wiley, New York, USA, 2002.

[MT93] S. P. Meyn and R. L. Tweedie. *Markov chains and stochastic stability*. Springer–Verlag, 1993.

[Neu81] M. F. Neuts. *Matrix-geometric solutions in stochastic models - an algorithmic approach*. The John Hopkins University Press, London, 1981.

[NV94] Y. Narahari and N. Viswanadham. Transient analysis of manufacturing systems performance. *IEEE Transactions on Robotics and Automation*, 10(2):230 –244, 1994.

[PF91] B. Plateau and J.-M. Fourneau. A methodology for solving markov models of parallel systems. *J. Parallel Distrib. Comput.*, 12:370–387, 1991.

[Pla85] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *Proc. of the 1985 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, SIGMETRICS '85, pages 147–154. ACM, 1985.

[PNK] D. Parker, G. Norman, and M. Kwiatkowska. Workstation cluster. http://www.prismmodelchecker.org/casestudies/cluster.php.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

[PW96] J. G. Propp and D. B. Wilson. Exact sampling with coupled markov chains and applications to statistical mechanics. *Random Struct. Algorithms*, 9(1-2):223–252, 1996.

[PY05] Nihal Pekergin and Sana Younès. Stochastic model checking with stochastic comparison. In *Proceedings of the 2005 international conference on European Performance Engineering, and Web Services and Formal Methods, international conference on Formal Techniques for Computer Systems and Business Processes*, EPEW'05/WS-FM'05, pages 109–123, Berlin, Heidelberg, 2005. Springer-Verlag.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *Proceedings of the 5th Colloquium on International Symposium on Programming*, pages 337–351, London, UK, UK, 1982. Springer-Verlag.

[RR98a] W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic models*, volume 1491 of *LNCS*. Springer–Verlag, June 1998. Advances in Petri nets.

[RR98b] W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets II: Applications*, volume 1492 of *LNCS*. Springer–Verlag, June 1998. Advances in Petri nets.

[SAP95] W. J. Stewart, K. Atif, and B. Plateau. The numerical solution of stochastic automata networks. *European Journal of Operational Research*, 86:503–525, 1995.

[SB93] M. Sereno and G. Balbo. Computational algorithms for product form solution stochastic Petri nets. In *Proc. of the 5th International Workshop on Petri Nets and Performance Models*, pages 98–107, Toulouse, France, October 19–22 1993. IEEE Computer Society Press.

[Sch84] P. J. Schweitzer. Aggregation methods for large markov chains. In *Proceedings of the International Workshop on Computer Performance and Reliability*, pages 275–286. North-Holland, 1984.

[Sen06] E. Seneta. *Non-negative Matrices and Markov Chains (Springer Series in Statistics)*. Springer, 2006. Revised reprint of the second (1981) edition.

[Ste94] W. J. Stewart. *Introduction to the numerical solution of Markov chains*. Princeton University Press, USA, 1994.

[Ste09] William J. Stewart. *Probability, Markov chains, queues, and simulation: the mathematical basis of performance modeling*. Princeton University Press, 2009.

[SVA05a] K. Sen, M. Viswanathan, and G. Agha. On statistical model checking of stochastic systems. In *CAV*, volume 3576 of *LNCS*, pages 266–280, 2005.

[SVA05b] K. Sen, M. Viswanathan, and G. Agha. VESTA: A statistical model-checker and analyzer for probabilistic systems. In *Proc. QEST'05*, pages 251–252, 2005.

[TMWH92] K. S. Trivedi, J. K. Muppala, S. P. Woole, and B. R. Haverkort. Composite performance and dependability analysis. *Performance Evaluation*, 14(3–4):197–215, February 1992.

[Tri82] K. S. Trivedi. *Probability & statistics with reliability, queueing, and computer science applications*. Prentice Hall, Englewood Cliffs, NJ, USA, 1982.

[Tru97] L. Truffet. Near complete decomposability: Bounding the error by a stochastic comparison method. *Adv. in App. Prob.*, 29:830–855, 1997.

[UD98] E. Uysal and T. Dayar. Iterative methods based on splittings for stochastic automata networks. *European Journal of Operational Research*, 110:166–186, 1998.

[Var85] M. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *FOCS 1985*, pages 327–338, 1985.

[Wal45] A. Wald. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics*, 16(2):117–186, June 1945.

[Wan04] Farn Wang. Formal verification of timed systems: A survey and perspective. In *Proceedings of the IEEE*, page 2004, 2004.

[WD98] J. Wang and Y. Deng. Incremental modeling and verification of flexible manufacturing systems. *Journal of Intelligent Manufacturing*, 4, 1998.

[You05] H.L.S. Younes. Ymer: A statistical model checker. In *Computer Aided Verification (CAV)*, volume 3576, pages 429–433. Springer, 2005.

[YS06] H.L.S. Younes and R.G. Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Inf. Comput.*, 204(9):1368–1409, 2006.

[ZL96] Y. Zhao and D. Liu. The censored Markov chain and the best augmentation. *Jour. of App. Prob.*, 33:623–629, 1996.