Compositional Solution of Stochastic Process Algebra Models

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der
Rheinisch-Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften
genehmigte Dissertation

vorgelegt von

Dipl. Inf. (Univ.) Henrik Bohnenkamp
aus
Minden (Westfalen)

Berichter: Universitätsprofessor Dr. Ir. Boudewijn R. Haverkort
Universitair Hoofddocent Dr. Ir. Joost-Pieter Katoen

Tag der mündlichen Prüfung: 13. Februar 2002

Abstract

This dissertation is about the solution of Markovian stochastic process algebra (SPA) models and the avoidance of the state-space explosion problem. We try to answer the question whether the compositionality of SPA models can be exploited to overcome the largeness problems appearing when evaluating such models.

First, instead of a global view, we take up a *local* view, *i.e.,* we focus on components of SPA models, and derive some general results about the relation between components. We identify waiting times, throughputs, and branching probabilities as the three quantities that should be known for a compositional performance evaluation strategy.

Then, we consider a special class of SPA processes that describe semi-Markov processes. SPA processes in this class are suitable to be solved by a very efficient new technique. An important step in applying this technique is the computation of the mean value of the maximum of phase-type distributed random variables. A naive approach for a computation would require exponential space, but we present an efficient algorithm of polynomial complexity in time and space in the number of considered random variables.

Finally, we consider a true-concurrency semantics for SPA models and investigate its use for an efficient solution of SPA models. We identify three important quantities to express performance measures in this semantics. Unfortunately, as we will show, only for very restricted cases the true-concurrency-view on SPA models allows the actual *computation* of measures.

Zusammenfassung

Diese Dissertation behandelt Lösungsverfahren für Markovsche stochachstische Prozeßalgebren mit dem Ziel, das Problem der Zustandsraumexplosion zu vermeiden. Wir untersuchen die Frage, ob die Kompositionalität von SPA-Modellen zur Vermeidung der Zustandsraumexplosion ausgenutzt werden kann.

Zuerst konzentrieren wir unsere Untersuchungen auf die Komponenten eines SPA-Modells und leiten einige allgemeine Ergebnisse her, die Aufschluss über die stochastischen Abhängigkeiten zwischen diesen Komponenten geben. Wir identifizieren Wartezeiten, Durchsätze und Verzweigungswahrscheinlichkeiten als die drei wichtigsten Quantitäten, die die Abhängigkeiten zwischen Komponenten beschreiben, und die es erlauben, Leistungsmaße auszudrücken.

Wir untersuchen dann eine spezielle Klasse von SPA-Prozessen, die Semi-Markov-Prozesse beschreiben. SPA-Modelle in dieser Klasse können sehr effizient gelöst werden. Ein wichtiger Teilschritt dieser Technik ist dabei die effiziente Berechnung des Mittelwertes des Maximums von phasenverteilten Zufallsvariablen. Ein naiver Ansatz würde sofort zu einem exponentiellem Anwachsen zumindest der Lösungszeit führen, abhängig von der Anzahl der Zufallsvariablen. Wir stellen jedoch ein Verfahren vor, dessen Speicherbedarf und Lösungszeit nur polynomial mit der Anzahl der Zufallsvariablen wachsen.

Schließlich betrachten wir eine *true-concurrency*-Semantik für SPA, die auf Ereignis-Strukturen basiert. Letztere ermöglichen eine explizite Darstellung von Parallelität und Lokalität. Wir untersuchen, ob diese Semantik Vorteile bietet für die effiziente Lösung von SPA-Modellen. Wir identifizieren drei grundlegende Quantitäten, mit denen sich Leistungsmaße in Ereignis-Strukturen darstellen lassen. Bedauerlicherweise zeigt ein anderes Ergebnis, daß nur unter starken Einschränkungen Leistungsmaße auch tatsächlich berechnet werden können.

Acknowledgements

Many people have helped me to finish this thesis. First to mention is my doctoral adviser, Boudewijn Haverkort. Not only that he actually *did* advise me (something that should not be taken as granted at german universities) and therefore helped me a great lot with all the difficulties that come with the writing of a dissertation; not only that he gave me constant encouragement to finish it; not only that he stoically endured my occasional fits of exaggerated self-criticism. Most importantly, he conducted the activities in his group with an open mind and created a very pleasant atmosphere, which was the perfect environment to grow the sprout that was to become this thesis. I am very glad that I was allowed to be part of his group, the "Laboratory for Distributed Systems and Performance Evaluation".

I am grateful to Joost-Pieter Katoen, my co-adviser, who pointed out many weaknesses of earlier versions of this thesis, and who gave many valuable hints to improve its contents. I thank Rom Langerak, who helped me with the event structure part of this thesis, and who convinced me that it is also absolute all-right and even valuable to write about failures. Also Holger Hermanns has helped me, for example during several long-distance calls, to understand some of the more peculiar properties of weak congruences.

My former colleagues, Alexander Bell, Ramin Sadre, Rachid El Abdouni Khayari, and Lucia Cloth, have helped me to find more and more time for the thesis, by gradually taking over my several obligations I had in the group. Moreover, we had some entertaining and deep discussions, not only about the work, but also about important topics, as, for example, the superiority of Solaris over Linux.

My special thanks go, however, to the "Baaders", Sebastian Brandt, Ani Turhan, Carsten Lutz, and of course my dear friend Uli Sattler. With the deafening noise that was coming out of their offices quite frequently they have given me plenty of opportunities to go over, to complain a bit and look reproachful, to sit down, to chat for a while with them, and to praise them silently for the distraction they have provided me with. I have enjoyed it immensely.

I thank you all very much.

Contents

Part I

In the Beginning

Chapter 1

Introduction

Model-based performance evaluation of computer- and communication systems has a long history. Agner K. Erlang (1878–1929) was the first person who used contemporary mathematics to model parts of the developing telephone networks stochastically and to solve some dimensioning problems *a priori*. He founded the theory of queues and published his most important results already in 1909 and 1917 [51, 52].

Since then, the research in the field of performance evaluation has led to a rich theory, heavily influenced by the theory of stochastic processes, which is based on the seminal work of Andrei A. Markov (1856–1922) from 1907 [101]. Performance models are generally described as stochastic processes. Since direct modelling of complex systems in terms of stochastic processes is merely impossible, over the time, more abstract formalisms have been invented to describe performance models. The better known formalisms are queues, queueing networks [83, 57], stochastic automata networks [119], and stochastic Petri nets (SPN) [114, 103].

In this dissertation, we consider the most recent development in the field of stochastic modelling formalisms: *Stochastic Process Algebras* (SPAs). SPAs are formalisms in which a distributed systems is described by means of a simple formal language with formal semantics. SPAs are Process Algebras (PAs) with stochastic extensions. PAs are a family of well-understood formalisms to model distributed systems, to verify their behaviour, and to check for certain properties, *e.g.,* deadlock-freeness. The smallest acting unit in a process algebra is the *process*, and complex processes are composed of simpler ones. This is commonly referred to as *compositionality*, and it is one of the most attractive features of process algebras. Compositionality gives process algebra specifications a *modular* and *hierarchical* structure and allows to treat parts of complex systems in isolation.

The basic activity of a process is an *action*. In *stochastic* process algebras, actions are equipped with a distribution function, which describes the *execution time* of the action stochastically. SPAs are suitable to describe functional as well as stochastic behaviour in one single specification. Therefore, they belong in two worlds: the modelled systems can be checked for design flaws with formal verification methods, and they can be checked

for performance and dependability properties by investigating the underlying stochastic process. A single model can give information about, *e.g.,* deadlock-freeness and liveness, *and* about, *e.g.,* throughputs, bottlenecks, availability, and the reliability of the considered system.

Up to the current day, SPAs are the only compositional formalisms that allow the user to integrate functional and stochastic information into *a single* model.

Performance Analysis with Stochastic Process Algebras

Many different SPAs have been defined until now and some of them describe very general types of stochastic processes. Such models can usually only be evaluated by discrete-event simulation [111, 128]. One of the major drawbacks of simulation as evaluation technique is that usually many simulation runs are required to obtain statistically valid measures. Moreover, each run usually requires a long time to complete.

Markov processes are a restricted class of stochastic processes which, contrary to most other stochastic processes, enjoy the nice property that they can be analysed numerically. There are many SPAs which are restricted such that the underlying stochastic process is always Markovian. Consequently, such models can be evaluated numerically. One of the standard tasks in the performance evaluation of Markovian SPA models is to derive an explicit representation of the underlying Markov process. This is usually just a reachability analysis of the state space of the considered SPA model, and the result is a matrix of real numbers: the *generator matrix*. The evaluation of a Markov process requires only the solution of a system of linear equations, which is defined by the generator matrix.

The advantage of a numerical evaluation over a simulation is that the results are usually much faster obtained and are statistically accurate. One of the disadvantages of Markovian modelling is that the size of the generator matrix can become very large—larger than computers can possibly handle.

Complexity Problems

The derivation and numerical analysis of Markov processes underlying SPA specifications is complex: the size of the state space usually grows exponentially in the number of (sub-)processes of the SPA specification. The reason for this is that SPA (sub-)processes describe *independent* activities of one or another kind. In the worst case, the state space of an SPA process is the cross-product of all its sub-processes. So, if a process comprises n sub-processes, and if each of these sub-processes has at most k states in its individual state space S_i, for $i = 1, \ldots, n$, then the state space of the whole process would be of order $\mathcal{O}(k^n)$. The exponential growth of the state space in the size of the model is referred to as *state space explosion* problem[1], and it makes it sometimes impossible to assess the

[1] Sometimes also referred to as *largeness problem*.

performance characteristics of even moderately sized models. Responsible for the state space explosion is the *global* view on the system : a state of the overall process comprises all states of the individual components (the local states), and all possible combinations of the local states have to be taken into account.

Aims and Overview

This dissertation is concerned with solving Markovian SPA models while avoiding the state space explosion problem. The structure that is inherently available in SPA specifications raises the question whether this structure can not be exploited in the stochastic analysis of the model. Processes are *composed* of *smaller* processes, *i.e.,* its *components*. We investigate possibilities to carry out computations on the smaller processes to obtain measures for the considered overall model. Such a procedure should have a memory consumption that grows only polynomially or even only linearly in the number of components. It would be ideal if the computation time would also be only polynomial or linear in the number or components. Such a procedure does not (yet) exist for SPAs, and as appealing this idea might be, it is by no means obvious how such a procedure should work.

In this dissertation, we will take first steps towards the direction of a solution method for SPA processes that exploit the compositionality of process descriptions. We proceed along the lines below.

A Stochastic Process Algebra. Before we can start our investigations, we must agree upon a formalism that is suitable to convey the ideas to be developed. To do so, we define the stochastic Markovian process algebra $\mathcal{YAWN}$. Using this SPA, we will describe systems of nearly independent components that interact by means of *synchronisation*. A synchronisation happens at a point on the time scale which is defined to be the earliest time where all components that are meant to participate in the synchronisation are actually ready to do so. Once a synchronisation has happened, all participating components again proceed independently from each other. The semantics of $\mathcal{YAWN}$ will be given in terms of labelled transition systems.

What Is Going on Inside? The usual perspective to look at SPA models is the global view, represented by the global Markov process. Now that we have decided to focus on components, we have to adjust the perspective. Therefore, instead of a global view, we will take up a *local* view. To do so, we will describe components and the performance measures that can be derived for them. Moreover, we will describe the relation between components and the complete system processes as well as the relation to other components. We will describe the local view in terms of the transition systems that are defined by the semantics of $\mathcal{YAWN}$.

SPA and Semi-Markov Processes. Continuous-time stochastic processes have a notion of state and time. A continuous-time Markov process has the memoryless property, *i.e.,* the future of such a stochastic process depends only on its current state, at all times. The class of semi-Markov processes is a strict generalisation of the class of Markov processes. In general, being memoryless does not hold for semi-Markov processes, except at state *changes*. The future of a semi-Markov process depends only on its current state and the time that has passed since the occurrence of the last state change, and that at all times. We call the instances at which the future of a stochastic process depends only on its current state *regeneration points*. For continuous-time Markov processes, every time instance is a regeneration point. For semi-Markov processes, only state-changes are regeneration points.

Generally, synchronisations are regeneration points of a given SPA model since for the underlying Markov processes, every point in time is a regeneration point. However, this is only true for the global view. If we focus on one specific component, say, C, *i.e.,* choose the *local* perspective of C, things are different. The future of C depends on other components, due to synchronisation. If C takes part in a synchronisation at time t, then it knows, in the moment of the synchronisation, about the state of other participants. Hence, C's future does not depend on the history of these components before t. However, in general there are also components that do not participate in the synchronisation with C. Nevertheless, they could influence C in later synchronisations at times $t' > t$, *i.e.,* the future of C generally depends on them. C does not know in which particular state these components are at time t. Whichever state it is, it depends on t, and therefore, also the future of C depends on t. From the perspective of a component, a synchronisation is only a limited regeneration point, because the future of a participating component is independent of its own past and of the past of other participants of the synchronisation, but not of the past of the components that did not take part in the synchronisation.

We will consider the case when synchronisations are *always* regeneration points, global *and* local. This will lead us to an interesting class of SPA processes, for which we can compute local performance measures in a much more efficient way than by the usual steady-state analysis. Moreover, we will show that the state space explosion problem does not exist for this class.

Event Structures. We have distinguished between the global and the local view. The methods to express locality are complicated, since locality is not a natural notion for transition systems. There are, however, formalisms known in which locality, and therefore also concurrency, is explicitly expressed. The formalisms are used as semantical basis for *e.g.,* process algebras and Petri nets. Due to their explicit notion of concurrency they are called *true-concurrency* semantics.

One well-known class of formalisms of the true-concurrency type are *event structures* [116, 50, 18, 94, 84]. Event structures represent the evolution of a system by a partially ordered set of events. Events denote the occurrence of a most basic activity of the considered formalism (*e.g.,* the firing of a transition or the execution of an action). The partial

order represents the causal relationship between events. The relation between mutually-exclusive events is explicitly represented by a conflict relation. Stochastic extensions of event structures exist and are straightforward [84, 126, 38].

We will consider an event structure semantics for $\mathcal{YAWN}$. We will reconsider the concept of locality and the results that we have already derived before.

Related Work

The largeness problem does not only occur with SPA models. Queueing networks, stochastic Petri nets, stochastic activity networks—they all share this problem. Many approaches exist that aim at overcoming the largeness problem. In this section, we will present a classification of these approaches. We still restrict ourselves to Markovian models, *i.e.,* models which stochastic behaviours can be described by finite state Markov processes, and we consider only methods to obtain steady-state measures.

The ultimate goal of the analysis of performance models is to obtain performance measures that can be interpreted within these models. A straightforward approach to obtain these measures is to explicitly derive the generator matrix of the Markov chain that describes the stochastic behaviour of the considered model, to solve it, and to compute the desired performance measures of the model from the probability distribution thus obtained. This approach allows us to derive most steady-state performance measures for a model.

This general approach, however, is also the most vulnerable to the largeness problem. We will now describe two different classes of approaches that try to avoid this problem.

Methods to Represent the Generator Matrix

The goal of methods in this category is to compute the steady-state probability distribution of the Markov chain underlying the considered model. These approaches do not actually tackle the state space explosion problem itself, but the problem that generator matrices of such Markov chains do not fit into the main memory of a computer. Thus, these techniques do not avoid the problem, but rather circumvent the necessity to hold the complete generator matrix in the memory. These approaches differ in whether they compute exact steady-state probabilities or approximations, and whether they rely on a special structure of the considered generator matrix or not. In the following, we describe some of these approaches in more detail.

Brute force approach: The most straightforward way to tackle the largeness problem is to invest in hardware, *i.e.,* in memory. Several approaches exist to implement the derivation and solution algorithm for the generator matrix on large machines, possibly taking the advantage of having several processors into account. This has been done, for example, by Allmaier et al. [3] and Knottenbelt et al. [90, 89, 91].

Based on work of Ciardo et al. [28], Haverkort, Bell and Bohnenkamp have considered clusters of personal computers as the hardware basis [63, 7] and have developed a distributed algorithm for deriving and solving large generator matrices.

Time scale decomposition: Generator matrices are said to be nearly complete decomposable (NCD), if they have a block diagonal structure, as depicted in Figure 1.1, and if the blocks $\mathbf{A}_1, \ldots, \mathbf{A}_n$ have entries that differ from the rates in the off-diagonal blocks *in orders of magnitude*. This structure ensures that, once a set of states is entered that is described by, say, block $\mathbf{A}_i$, then the probability to leave this set of states is much smaller than the probability to stay in this set. NCD matrices occur

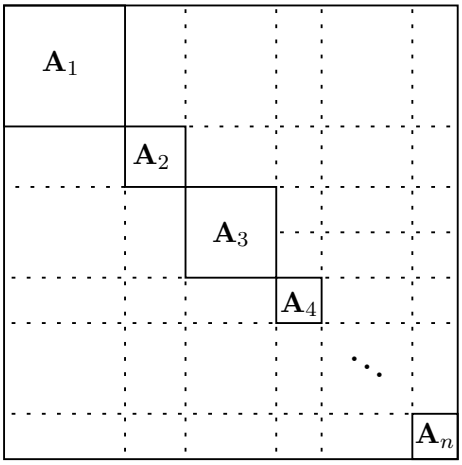


Figure 1.1: Block structure of an NCD matrix

frequently in dependability models: the "large" rates usually describe the operation of a system in a normal mode, whereas the "small" rates describe the possibilities of a breakdown, which usually occur rather seldomly. NCD matrices can be decomposed as follows: the blocks $\mathbf{A}_1, \ldots, \mathbf{A}_n$ are considered in isolation, and steady-state solutions are computed for them. The off-diagonal blocks are used to define a superordinated Markov chain that describes the transitions between the diagonal blocks. This Markov chain must be solved, too, and all results have then to be combined to yield the (approximated) steady-state probabilities of the original Markov chains. The quality of the solution depends on the order of magnitude in which the "large" and "small" rates differ.

The fact that especially NCD matrices are very suitable to be solved in this "one-step divide-and-conquer" fashion goes back to the work of Simon and Ando [131]. Based on this work, the so-called *time scale decomposition* of performance models, as, *e.g.,* for queueing networks [40], and stochastic Petri nets [4, 14], has been developed.

Kronecker representation: Plateau et al. proposed Kronecker algebra to represent global Markov chains [119]. She used *stochastic automata networks* (SAN), a formalism

quite similar to SPA, where sub-entities act independently from each other and interact occasionally. The generator matrix of an SAN is not represented explicitly, but implicitly by means of expressions in the corresponding Kronecker algebra. Such a representation still suffers from the state space explosion problem, since still all states are considered. However, the representation of this state space on a computer requires memory that grows only linearly in the number of components. The time complexity, however, remains the same, *i.e.,* grows exponentially in the number of components.

Several approaches exist to transfer the results for SANs to other formalisms, *e.g.,* stochastic Petri nets [47].

Product-form solution: Two important properties of Markov chains that imply a very efficient solution method are *reversibility* and *quasi-reversibility*. Especially for some classes of Markovian queueing networks, *e.g.,* Gordon-Newell-Networks [57], and BCMP networks [6], the properties have been proven to be most valuable. Reversibility and quasi-reversibility allow to derive performance measures analytically, *i.e.,* not by numerical techniques, but by closed mathematical formulae. There is no need to construct the state space, *i.e.,* the generator matrix of the Markov chain underlying a model with product-form. However, since product-form solutions do allow the derivation of Markov chain probabilities (and it is still necessary to obtain them to derive measures for, *e.g.,* queueing networks), the structure of the state space must be known in advance. For closed queueing systems, for example, the state space can usually be described by all vectors of non-negative integers, which entries sum up to a certain constant.

Since product-form solutions are very attractive, there were also approaches to define classes of other performance models with underlying reversible or quasi-reversible Markov processes.

Symbolic methods: The most recent and very promising approaches to represent the generator matrix of the considered Markov process is based on the work of Bryant on Binary Decision Diagrams (BDDs) [23, 24]. Bryant has introduced (ordered) BDDs as an efficient means for the representation and manipulation of Boolean functions, which can be represented as a set of strings of fixed length of zeroes and ones. Such set of strings can be represented in a directed acyclic graph, where each path in this graph represents one string. The graph is constructed in a way such that isomorphic sub-graphs are amalgamated.

The idea to represent state spaces in terms of OBDDs is based on an encoding of states and transitions as binary functions (*i.e.,* as a set of strings of zeroes and ones). It has been demonstrated that the BDD representation of a state space can be very memory efficient: Markov chains can be represented in the memory of a simple PC, for which a solution would require centuries.

Initial work for stochastic transition systems has been done by Siegle [129, 130], based

on earlier work of Clarke et al. [37], and Hachtel et al. [59]. The latest approach, based on a mutation of BDDs, called MDDs (Multi-valued decision diagrams), and Kronecker techniques is from Ciardo et al. [29, 30, 32].

All approaches that we have presented in this section have at least one of the two following limitations:

1. The steady-state probability vector of the considered Markov process must be represented explicitly in the memory of the computer (except, perhaps, for product-form solutions). The size of this solution vector grows exponentially in the size of the model.

2. The derivation of the steady-state distribution takes usually an amount of time that also grows exponentially with the size of the model.

Hence, although the state space of the considered Markov process might be represented very efficiently by the described approaches, the practical applicability is still restricted by the memory requirements of the solution vector and the time one wants to wait until a result is computed.

Methods Based on Model Modification

In this section, we will consider a class of approaches which do tackle the problem on a higher level. Instead of finding a way to compute steady-state measures of the overall Markov chain, the model itself is manipulated before computations take place. The aim of this approach is to actually *reduce* the complexity of the solution task.

Inherent to such an approach is that the *model* is modified. As a consequence, the measures that are eventually derived from it differ possibly from that of the original model. Moreover, the manipulation can alter the structural properties of the model in such a way that only a subset of those performance measures can be obtained that could be derived at least theoretically from a steady-state analysis of the original, overall Markov process.

A typical example for a model modification approach is the *decomposition* of the model: the model is decomposed in several sub-models, which are then solved in isolation. If the model is, for example, a Petri net, then a decomposition approach would cut the net in smaller subnets, which are then perhaps solved in isolation by the usual steady-state analysis. Generally, it will be the case that only measures that are valid for the subnets can be derived. Measures that relate to two or more different subnets, however, can not be computed, since subnets are treated independently from each other. Measures that relate to more than one sub-net are usually not expressible by these independent measures.

Example 1.1

We illustrate this with an informal example. We assume a small dependability model (it is not important *how* it is modelled), that comprises a machine which is either operational or out of order, and a repair person. If the machine breaks down, the repair person comes to repair the machine and to set it back to operational. The repair person, however, can have vacations, which possibly result in a delay of the repair. A measure of interest now could be the probability that the repair person has her holidays when the machine breaks down. With an ordinary steady-state analysis, this probability can be easily obtained. However, when we decompose a model of this system and consider the machine and the repair person as different components, then all we could probably derive are the probability that the machine breaks down and the probability that the repair person has her holidays—but not the conditional probability that the repair person has her holidays when the machine breaks down.

To summarise, model manipulation techniques might reduce the set of measures that can be obtained for models.

We discuss now in more detail three model manipulation techniques which are based on *decomposition* and *aggregation*.

FES method: An early approach that is based on decomposition and aggregation has been developed for queueing networks [27]. The technique decomposes a closed queueing network into an *FES sub-network* (FES stands for Flow Equivalent Server) and the *complement sub-network*. The FES sub-network is short-circuited to form a closed network again. Then, for different job populations $k = 1, \ldots, K$, the throughputs $T(k)$ across the short-circuit of the FES sub-network are computed. The complement sub-network is also closed again, where the FES sub-network is replaced by a single, load-dependent queue (the *FES node*). The parameters of the queue are derived from the throughputs $T(k)$. The complement sub-network with FES node is then solved by the usual methods available for load-dependent queueing networks.

The approach yields exact results if the considered original queueing network has a product-form solution. Otherwise, the results are approximations.

Decomposition and Response Time Preservation: Stochastic marked graphs (SMGs) are a special class of stochastic Petri nets which only allow zero or one token to be in a place and which are decision free. For SMGs, Campo, Silva, Jungnitz et al. have worked on decomposition techniques based on *Response Time Preservation* (RTP) [2], which has much similarities to the FES method described above: SMGs are cut in subnets along some cutting lines [26]. For each piece, a closed sub-net is constructed, where the respective other sub-nets are replaced by a single transition, respectively. A superordinate SMG is constructed which describes the dependencies

among sub-nets. The modified sub-nets, together with the superordinate net, are solved and special export parameters from the sub-nets are mutually imported in the respective other nets. The nets are repeatedly solved, each time with updated export/import parameters. The procedure stops when the imported parameters converge. The measures obtained are local to the sub-nets.

Near-Independence: Ciardo and Trivedi have proposed a method to decompose stochastic reward nets (a special kind of stochastic Petri nets) into *nearly-independent* parts, to define stochastic measures that describe the dependencies between the parts and to solve the parts (possibly involving an fixed-point iteration) [31]. The difference to the previous method is that no restriction on the structure of Petri nets is assumed, and that the decomposition heavily depends on the attribute *near-independent*. The less different sub-parts of an SRN do interact, the more they are suitable to be considered in isolation. The perfect case would be when there was no interaction at all, but that would be a trivial case not worth any consideration. The problem with this approach is that it is very difficult to identify nearly-independent parts of an SRN. This requires intuition and skill.

Adaption to SPA

For SPA, several of the above approaches have been adapted or exploited to overcome the state space explosion also for SPA models. A complete survey can be found in [74].

1. Harrison and Hillston have defined classes of SPA processes such that the underlying Markov process is reversible [77]. Thomas and Gilmore have defined a class of SPA processes that is quasi-reversible [134]. Both approaches allow product-form solution of the underlying Markov process.

2. Mertsiotakis and Hillston have developed a method to exploit near-decomposability by time-scale decomposition of SPA processes [72, 108].

3. Buchholz has defined a semantics for an SPA that is based on the Kronecker approach of Plateau for SANs [25]. A similar approach has been chosen by Hillston and Kloul [75].

4. Mertsiotakis and Silva developed an algorithm that is based on the RTP approach described above [107, 108].

This Dissertation

The approaches so far developed for the decomposition of SPA models have been all more or less motivated by existing approaches for other formalisms. The general method for

all of them is to find a class of SPA models that has the same or similar properties that are required for an analysis by the already known techniques. The requirements that have to be met by the specifications such that the considered Markov process has the required property to be exploited, are sometimes a bit artificial, when formulated for SPAs.

In this dissertation, we go a different way. We focus strictly on stochastic process algebras. While in other formalisms compositionality is an artificial concept, it is inherent in stochastic process algebras. Thus, the invention of a *genuine*, component based solution approach for stochastic process algebras that takes advantage of the compositionality is very promising.

Compared to the approaches presented in the previous sections, our view on SPA models is very similar to that chosen by Ciardo and Trivedi [31] (although we will never try to adapt their solution approach): components of an SPA specification can be seen as nearly-independent parts of the whole system. It is our belief that this view on SPA models is the most natural one, since the identification of nearly-independent parts of the system is already given by the compositionality of the specification.

Outline of this Dissertation

This dissertation assumes knowledge of basic probability theory and the theory of Markov processes. In Appendix A and B, the most common notations, definitions, and results for the formalisms used are summarised.

We begin in Chapter 2 with a concise review of the theory of process algebras, its stochastic extensions, and the solution techniques available for them.

In Chapter 3, we introduce the SPA $\mathcal{YAWN}$, which we will use in the subsequent chapters.

In Chapter 4, we introduce local measures for the components of a $\mathcal{YAWN}$ specification. Moreover, we identify the stochastic dependencies between the different components of a $\mathcal{YAWN}$ process. Finally, we characterise *waiting times* and show that they are the measures on that a component-wise steady-state analysis could be based upon.

In Chapter 5, we identify a class of processes that can be solved very efficiently without the need of constructing the complete state space. We show that the memory requirements for this approach grow only linearly with the number of components. Therefore, the state explosion problem is avoided for this class of processes. We show that waiting times for this class of processes can also be derived. All derived quantities for this approach are exact. However, only local measures can be computed.

In Chapter 6, we introduce an event structure semantics for $\mathcal{YAWN}$. Then, we define first some important stochastic measures in terms of the event structures, then we consider the possibility to actually compute them. Our results, unfortunately, show that an approach based on event structures does not help to derive performance measures more efficiently.

In Chapter 7, we summarise the thesis, discuss the achievements and tie up some loose ends.

Chapter 2

Stochastic Process Algebra

In this chapter, we will give a short introduction into the theory of process algebras and stochastic process algebras. Nothing in this chapter is original. Most of the presented material is from the books of Baeten et al. [5], Milner [110], Hennessy [65], and the work from Hillston [73], Hermanns [66], and many others.

2.1 Process Algebra in a Nutshell

Process calculi are formalisms which are designed to describe reactive systems. Such descriptions (or *specifications*) are expressed by means of a simple formal language with well-defined semantics. Among the first and best-known representants of process calculi are CSP [78, 79], CCS [109, 110], and ACP [5]. Research on all of them has begun in the early 80's.

The approach to describe reactive systems by means of a language was inspired by the research on the field of functional programming languages. There, it has been shown that a simple calculus, the λ-calculus, is expressive enough to compute all functions that can also be computed by a Turing machine. On basis of the λ-calculus programming languages have been defined, like LISP.

One of the aims for the invention of process calculi (among others) was to create a simple formalism that can capture all important aspects of reactive systems, as the λ-calculus can for computable partial functions. Methods and means are very similar to those that have been used in the area of functional programming. Research has been fruitful, and as one result, the process calculus CSP served as a basis for the programming language OCCAM, which was particularly well suited to describe programs on tightly-coupled multi-processor systems (transputer). Also based on CSP, Brinksma and others developed the specification language LOTOS [21, 22], which eventually has been standardised by the ISO [82].

2.1.1 The Structure of a Process Calculus

The entities on which all process calculi are based upon are so-called *actions*. Actions denote the possibility of the occurrence of an activity. The most basic non-trivial activity is the *execution* of an action. Actions are *atomic*, *i.e.,* they either occur completely or not at all. Process calculi are formalisms which define how the atomic actions can be combined to describe more complex behaviour. We say that the behaviour thus described forms a *process*. A certain set of combinators has been developed with the purpose to define the behaviour of complex processes as a combination of more simple ones. Each process calculus has its own special set of operators, but they all can be classified. We describe this classification in the next paragraph.

The most simple imaginable behaviour is when nothing can happen. This behaviour must be expressible. Some process calculi, like e.g. CCS, introduce the special process stop, the process that never does anything. Other process algebras define the actions themself as the most basic processes, *i.e.,* if a is an action, then a is also the process that can execute the action a. The process that does nothing is then represented by the empty string, ϵ. Some process calculi have a special notion of deadlock as well as for successful termination of a process. ACP, for example, defines the special process δ, which denotes the process which runs in a deadlock ("with a crunching sound", as figuratively described in [8]).

On top of the basic processes, other processes can be defined inductively. The operators used for this can be classified as follows:

Sequential composition and prefixing. As already mentioned above, in ACP the most basic process is the *action*. Hence, to describe the successive execution of actions, an operator for sequential composition, $\cdot;\cdot$, of processes is needed. On the other hand, if we regard the stop process as the only basic process, we must consider the actions as *operators on processes*, *i.e.,* if P is a process then $a(P)$ is a process that can perform an action a and then behaves as P. Normally, to spare the parentheses, $a(P)$ is written as $a.P$.

Choice. The purpose of choice operators is to denote the possibility of choice between different behaviours. Choice is often denoted by $+$, and if P, Q are processes, then $P + Q$ denotes the process that can either behave as P or as Q. There are different ways how a choice can be resolved, a decision can be made. We can distinguish between *external* choice, *i.e.,* the choice of an alternative is influenced externally, or *internal* choice, *i.e.,* the process decides for itself which alternative is chosen. In some process calculi, external and internal choice are explicitly expressed by different operators (*e.g.,* in [98]). In other calculi, only one operator is defined and whether the choice has to be seen as external or internal depends on the situation.

Recursion. With the sequential and choice operators, only finite behaviour can be described. Since reactive systems generally never terminate, a formalism to describe such systems should have a facility to describe infinite behaviour. In the context of

process calculi there are two approaches to do so: *defining equations* and *recursion operators*.

For the first approach, a set $CONST$ of so-called *process constants* is needed. A constant $C \in CONST$ is assigned a process by means of a defining equation: $C \stackrel{\text{def}}{=} a.C'$ is an example for such a defining equation. It states that C behaves as $a.C'$. C' can itself be a process constant or a process. Whenever a process constant C occurs in a process, its behaviour is determined by the process on the right side of its defining equation. Defining equations can be recursive, *i.e.,* the constant to be defined can occur within the expression that it defines. The defining equation $A \stackrel{\text{def}}{=} a.A$ is an example. Intuitively, A is supposed to be the process that can execute an infinite number of a actions. We can say that the process that can make an infinite number of a is the solution of this recursive equation[1]. There is no process *expression* which really expresses this behaviour—at least, not a finite one. Obviously, recursive equations define processes only implicitly, *i.e.,* their solution can not be represented by a finite process term which contains only prefix and choice operators. Nevertheless, for some cases it is desirable to have such a finite representation, and a satisfying workaround has been invented. So-called recursion operators $recX$ for $X \in VAR$ must be defined, where VAR is a set of *process variables*. The solution of the recursive defining equation $X \stackrel{\text{def}}{=} P$ is then explicitly denoted as $recX : P$. For the above example that means that $recX : P$ behaves like the process $P[recX : P/X]$, where $P[recX : P/X]$ is the process term where simultaneously all occurrences of X in P are syntactically replaced by $recX : P$.

As long as only a finite number of (recursive) equations is employed to express the (non-terminating) behaviour of a process, the both forms to define infinite behaviour are equivalent in the sense that a finite system of defining equations can be represented by a $recX$ expression, and each $recX$ expression can be converted to a finite system of defining equations.

Parallel Composition. The *parallel operators* are the operators that make process calculi powerful formalisms for the description of reactive systems. With the parallel operator it is possible to define entities that can *interact*, either by simple synchronisation or even by communication with value passing. It is, of course, also possible to model independent parallel execution of processes. The major differences between process calculi can be found in the definition of the parallel operators. As examples we consider CCS and LOTOS.

The CCS style of parallel composition assumes a special structure of the set of actions. For each action a, there is a complementary action $\bar{a}$. We call a a *send* and $\bar{a}$ a *receive* action. Then, if we put the process $P \stackrel{\text{def}}{=} a.\mathsf{stop}$ and $Q \stackrel{\text{def}}{=} \bar{a}.\mathsf{stop}$ in parallel, *i.e.,* if we

[1] Not all recursive equations have a unique solution. For example, the equation $X \stackrel{\text{def}}{=} X$ has infinitely many of them. Generally, in an recursive equation, all occurrences of the recursion variables must be guarded, *i.e.,* prefixed with an action, to guarantee the existence of a unique solution of the equation.

consider the process $R \stackrel{\text{def}}{=} P|Q$, then both P and Q can communicate: both processes execute a and $\bar{a}$ simultaneously. a is also said to be a *channel* between P and Q. Synchronisation in CCS is a two-party matter, *i.e.,* no third process can take part in it.

In LOTOS, things are different. There we have no complementary actions. The communication is not necessarily between only two parties. The parallel operator is equipped with a subset of the action set, the *synchronisation set*. This set contains all those actions on which the combined processes have to synchronise over.

Example 2.1

If we consider the processes $P \stackrel{\text{def}}{=} a.b.c.\mathsf{stop}$ and $Q \stackrel{\text{def}}{=} d.b.e.\mathsf{stop}$, then $R \stackrel{\text{def}}{=} P\|_b Q$ denotes a process in which both P and Q can perform the actions a and d independently from each other, but must then synchronise over action b in order to proceed. After the synchronisation has taken place, both P and Q can proceed again independently from each other, *i.e.,* the can perform the c action and the e action, respectively.

Example 2.2

If we consider the process $(P\|_b Q_1)\|_b Q_2$, where $Q_1 \stackrel{\text{def}}{=} Q_2 \stackrel{\text{def}}{=} Q$, then all three processes can start independently from each other, but have all to take part in the synchronisation over b before they can execute their respective last action.

A slight restriction of the LOTOS synchronisation is used in CSP, where the synchronisation set is the intersection of the actions that occur in the processes put in parallel.

Hiding and restriction. Another important class of operators is that of the *hiding* or *restriction* operators. The purpose of these is to mark the scope of actions which should never again take part in synchronisations. To do so, a special action is introduced, which is often denoted as τ or $\mathbf{i}$: the *internal* action. If we reconsider Example 2.2, process Q_2 could be inhibited from participating in the synchronisation over b that P and Q_1 are already involved in. The idea is to restrict the scope of a synchronisation. This is done by means of the postfix operator $\backslash H$, where H is a subset of the action set. The effect of the hiding operator is that all actions in H are *hidden* away: they are no longer visible from the outside. Then, a process where P and Q_1 do synchronise and Q_2 proceeds independently from both can be expressed as $(P\|_b Q_1) \setminus \{b\}\|_{\{\}} Q_2$.

Another purpose of the hiding operator is that of *abstraction*. Once the actions of a process are hidden, they are no longer visible from the outside and do not longer play a role. The internals of a specification are hidden away.

2.1.2 Syntax and Semantics

In the previous section we have introduced the main modelling facilities that come with process calculi. The description was most incomplete and informal. In this section, we want to show how specifications, written down in a process calculus language, can be given a precise, unambiguous meaning. A specification in a process calculus is a syntactic entity; it is a word from a certain language defined by some grammar. To give meaning to these terms normally means to assign a certain mathematical structure to the specification, which somehow describes the aspects of the behaviour that the writer of the specifications has in mind. Many different mathematical structures have already been used to give meaning. Nevertheless, there is one structure that serves as a standard semantics for almost every process calculus introduced so far: the *labelled transition system*.

Definition 2.3 A labelled transition system is a tuple (S, Lab, T), where S is a set of *states*, Lab is a set of *labels*, and $T \subseteq (S \times \mathit{Lab} \times S)$ is a set of *labelled transitions*.

The states denote the states of a process, and the transitions denote the possible state changes of the process. The assignment from process term to transition system is normally done by a concept introduced by Plotkin [120], the *structured operational semantics* (SOS).

We want now give an explicit definition of syntax and semantics of an process calculus that is similar to LOTOS. The first thing to do is to define the set of actions that are executed by processes. By Com we denote the set of *visible* actions. Visible actions are those actions that can be influenced from the outside by means of synchronisation. Then we define one special action, $\mathbf{i} \notin \mathit{Com}$, the *internal* action. Action $\mathbf{i}$ is an action that can never be influenced from the outside: it is never possible to synchronise over $\mathbf{i}$. We define $\mathit{Act} = \mathit{Com} \cup \{\mathbf{i}\}$ to be the set of all actions.

Now we are ready to define a formal language for our process algebra, the language of all process descriptions:

Definition 2.4 The language $\mathcal{L}_{\mathcal{PA}}$ is that defined by the following grammar:

$$P \longrightarrow \mathsf{stop} \mid a.P \mid P + P \mid P \|_S P \mid P \setminus H \mid A$$

where $a \in \mathit{Act}$, $S \subseteq \mathit{Act}$, $H \subseteq \mathit{Com}$, and $A \in \mathit{CONST}$.

The structural operational semantics for this language is now given by a labelled transition system. The state space S is a set of processes, the set of labels are the actions (*i.e.*, $\mathit{Lab} = \mathit{Act}$). The state space as well as the transitions are derived by means of a set of derivation rules. If (P, a, P') is a transition, then we write $P \xrightarrow{a} P'$.

The SOS rules take the form

$$\frac{PREMISES}{CONCLUSION}(\text{side conditions}).$$

The premises and conclusions make statements about the membership of labelled transitions in the transition system to be defined. The side conditions give further requirements for the applicability of the rule. There are special rules without premises, which are the axioms of the SOS. Usually, there is only one axiom in an SOS, namely

$$\frac{}{a.P \xrightarrow{a} P}.$$

Actually, this is an axiom *pattern*, since $P \in \mathcal{L}_{\mathcal{PA}}$ is an arbitrary process term. It states that

> for all processes that are syntactically of the form $a.P$ for $P \in \mathcal{L}_{\mathcal{PA}}$,
>
> $$a.P \xrightarrow{a} P$$
>
> is a valid transition from $a.P$ to P.

The next rule is not an axiom pattern, but a derivation rule:

$$\frac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'}$$

It states:

> Whenever $P \xrightarrow{a} P'$ is a valid transition, then $P + Q \xrightarrow{a} P'$ is a valid transition as well.

The other rules are read in a similar fashion and hence we list them only uncommented in Table 2.1.

If we have given a process $P \in \mathcal{L}_{\mathcal{PA}}$, then the SOS rules allow us to derive from P all transition which originate from P. This has be done by repeated application of the SOS rules. Such repeated applications of derivation rules form a *derivation tree*.

$$\frac{}{a.P \xrightarrow{a} P}$$

$$\frac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'} \qquad \frac{Q \xrightarrow{a} Q'}{P + Q \xrightarrow{a} Q'}$$

$$\frac{P \xrightarrow{a} P'}{A \xrightarrow{a} P'}(A \stackrel{\text{def}}{=} P) \qquad \frac{P\{recX : P/X\} \xrightarrow{a} P'}{recX : P \xrightarrow{a} P'}$$

$$\frac{P \xrightarrow{a} P'}{P \|_S Q \xrightarrow{a} P' \|_S Q}(a \notin S) \qquad \frac{Q \xrightarrow{a} Q'}{P \|_S Q \xrightarrow{a} P \|_S Q'}(a \notin S)$$

$$\frac{P \xrightarrow{a} P' Q \xrightarrow{a} Q'}{P \|_S Q \xrightarrow{a} P' \|_S Q'}(a \in S)$$

$$\frac{P \xrightarrow{a} P'}{P \setminus H \xrightarrow{\tau} P' \setminus H}(a \in H) \qquad \frac{P \xrightarrow{a} P'}{P \setminus H \xrightarrow{a} P' \setminus H}(a \notin H)$$

where $a \in Act$, $S \subseteq Com$, $H \subseteq Com$, and $X \in VAR$.

Table 2.1: $\mathcal{L}_{\mathcal{PA}}$ structural operational semantic rules

Example 2.5

We consider the process

$$P \stackrel{\text{def}}{=} a.b.\mathsf{stop} + b.c.\mathsf{stop} \|_a (a.c.\mathsf{stop} \|_a a.d.\mathsf{stop})$$

For P, we can derive the following derivation tree:

$$\frac{\dfrac{\dfrac{}{a.b.\mathsf{stop} \xrightarrow{a} b.\mathsf{stop}}}{a.b.\mathsf{stop} + b.c.\mathsf{stop} \xrightarrow{a} b.\mathsf{stop}} \quad \dfrac{\dfrac{}{a.c.\mathsf{stop} \xrightarrow{a} c.\mathsf{stop}} \quad \dfrac{}{a.d.\mathsf{stop} \xrightarrow{a} d.\mathsf{stop}}}{a.c.\mathsf{stop} \|_a a.d.\mathsf{stop} \xrightarrow{a} c.\mathsf{stop} \|_a d.\mathsf{stop}}}{a.b.\mathsf{stop} + b.c.\mathsf{stop} \|_a a.c.\mathsf{stop} \|_a a.d.\mathsf{stop} \xrightarrow{a} b.\mathsf{stop} \|_a c.\mathsf{stop} \|_a d.\mathsf{stop}}$$

Please note that P can execute more than one action (namely b), hence there are more derivation trees for P.

The semantics for this PA is then a transition system that contains exactly all those transitions that can be derived by the rules of Table 2.1.

2.1.3 Process Algebra

Till now we have only written about process *calculi* as tools to describe concurrent systems. But generally, these formalisms are called process *algebra*. The main purpose of the formal modelling of concurrent systems is not only to describe them, but to reason about them. One of the most important questions is whether two descriptions of the same system are equal. This requires a formal definition of what equality between process descriptions means. There are many ways to define *equality* between processes, each one a little different, one more discriminating, and the other more levelling.

If a notion of equality does already exist, it is necessary to find a way to prove the equality of two descriptions.

There have always been two approaches to solve these problems, which in some sense are complementary to each other. For both approach it is necessary to define the objects on which all reasoning is focussed on: a language, which describes processes.

Algebraic approach. The first approach starts with the definition of an algebra on the set of all process descriptions. This means, that a set of equations on the terms is defined, which is initial in the sense that they are sufficient to prove whether two arbitrary processes are equal or not. This is nothing else then the definition of an abstract algebra with a carrier set and a set of axioms. Generally, this approach would be sufficient to decide between the equality of processes (if this question is decidable at all! This is generally not the case.). But this might be cumbersome. It is hence sometimes more elegant to find a concrete model for the algebra, *i.e.,* to define a semantics for it that is fully abstract with respect to to the equality defined by the algebraic laws [137]. Fully abstractness means that iff two processes t_1 and t_2 are equal in the algebra, then their semantics, the associated mathematical objects $[\![t_1]\!]$ and $[\![t_2]\!]$ are identical. If the semantic objects are easy to identify or to distinguish, then the decision of equality or difference of two process descriptions is much easier.

Example 2.6

The first version of CSP was a process algebra with a *trace semantics*: the meaning of the considered processes is given by means of the set of action sequences (*traces*) that a process can execute. This semantics shows that the roots of process algebra is classical automata theory and formal languages. As was shown, a trace semantics is not general enough. For example it is not possible to determine the deadlock–freeness of processes.

Behavioural approach. The language is given a semantics. Generally this means, for each process description a mathematical object is derived which is meant to describe

the behaviour of the process. Then, by reasoning on the semantic objects, an equivalence relation on the process terms is derived which is meant to be the equality that is looked for. The equality is required to have the plug-in replacement property, *i.e.,* it must be a congruence with respect to the operators of the considered process algebra.

Once the equivalence is chosen and proven to be a congruence, the process calculus is ready to become a process algebra. To do so, a set of axioms must be defined which induces a notion of equality on the set of processes terms. The equality must coincide exactly with the congruence relation defined before.

In the following, we will describe one of the most common congruences that is used to define equality on processes: *bisimulations*.

2.1.4 Bisimulation

Bisimulations [118] are the most common congruences for process algebras, and for nearly every one a notion of bisimulation has been defined. In the following definition we introduce bisimulation for processes in $\mathcal{L}_{\mathcal{PA}}$.

Definition 2.7 A relation $\mathrm{R} \subseteq \mathcal{L}_{\mathcal{PA}} \times \mathcal{L}_{\mathcal{PA}}$ is a strong bisimulation, if $P \mathrm{R} Q$ implies for all $a \in Act$:

– $\forall P' \in \mathcal{L}_{\mathcal{PA}}$: if $P \xrightarrow{a} P'$ then $\exists Q' \in \mathcal{L}_{\mathcal{PA}} : Q \xrightarrow{a} Q'$ and $P' \mathrm{R} Q'$

– $\forall Q' \in \mathcal{L}_{\mathcal{PA}}$: if $Q \xrightarrow{a} Q'$ then $\exists P' \in \mathcal{L}_{\mathcal{PA}} : P \xrightarrow{a} P'$ and $P' \mathrm{R} Q'$

Two processes P and Q are said to be strongly bisimilar ($P \leftrightarroweq Q$) if there is a bisimulation R such that $P \mathrm{R} Q$.

Informally, we can characterise bisimilarity as follows: two processes P, Q are considered to be bisimulation equivalent, if there is a bisimulation R such that, if $P \mathrm{R} Q$, both processes can execute the same sequences of actions, and after the execution, can stop in states P', Q' such that $P' \mathrm{R} Q'$. Strong bisimilarity $\leftrightarroweq$ is an equivalence relation on $\mathcal{L}_{\mathcal{PA}}$, and it can be shown that it is also a congruence with respect to all language operators.

Another important bisimulation congruence is *weak* congruence [110]. The most important thing about this congruence is that it abstracts from the execution of internal actions.

2.2 Stochastic Process Algebra

Stochastic process algebras have been invented in the early 90's. The research on the subject is stimulated by Herzog [69, 70], although earlier approaches seemed to have been proposed already in 1985 [117].

We will keep this section short and describe only the general picture, since in Chapter 3 an SPA is introduced in greater detail.

2.2.1 Introduction

The main idea of stochastic process algebra is to incorporate quantitative information in a qualitative process algebra model. In the approaches proposed so far, the quantitative information is given in terms of distribution functions or random variables, which denote the duration of an action. Those durations are specified together with an action.

Although the aim of the first SPAs was already to provide support for generally timed actions, the approaches where practically not feasible. As a consequence, research has focussed on exponentially distributed durations of actions, which made it possible to describe the stochastic behaviour of SPA processes by Markov processes. The first representatives of Markovian stochastic process algebras were PEPA [73], EMPA [12], MPA [25], and MTIPP [67]. The most recent representant of Markovian process algebra is IMC [66]. Some SPA are non-Markovian, first of all to mention ♠ (Spades) [41], and Interactive Generalised Markov-Chains [19]. ♠ is designed as a modelling language for simulation models.

Why should a system developer bother with stochastic process algebra? This has three good reasons.

1. In the development phase of distributed systems, be it computer systems or communication systems, the performance evaluation of a newly designed system often comes very late in the design process. The design might be functionally correct and a prototype might already exist, but if the performance of the system is assessed too late, the developers might experience a bad surprise, if their system does not meet the performance criteria that it should. In such a case, the system design must often be started from scratch. Such a scenario lets it appear reasonable to consider the performance aspects as early as possible in the design trajectory.

2. Performance evaluation is based on models. If one wants to evaluate the performance of a system in an early design stage, a model of the considered system has to be created. If the same developer wants to ensure that the design of his system is correct from a qualitative point of view, he probably has to define another model of his system, which he then must verify or check.

   With SPA, only one model must be defined. This has immense advantages:

- Only one model has to developed, which saves time.
- The integrated approach ensures that qualitative as well as quantitative descriptions relate to the same system.
- The performance measures can be interpreted in the qualitative model.

3. Stochastic process algebras allow to describe the qualitative, logical, as well as the quantitative, temporal behaviour of distributed systems in a single formalism. The qualitative description is nothing else but an "ordinary" process algebra description of the considered system. Hence, all techniques and tools for verification and model checking that have been developed for process algebra yet can also be applied to stochastic process algebra. The delay specifications define implicitly an stochastic process which can be evaluated, either analytically, numerically, or by simulation, depending on the properties of the underlying stochastic process.

The main idea of SPAs is to enhance actions with a notion of *duration*. Durations are described stochastically by means of distribution functions. In *Markovian* SPA, only exponential distributions are considered as delay distributions. An exponential distribution function is uniquely determined by its *rate*, so it is possible to accompany an action simply with a rate to specify the desired duration distribution. A nice property of exponential distributions is that their mean value is $1/\lambda$. A typical, although simple, SPA process is then $P \stackrel{\text{def}}{=} (a, \lambda).P'$, where a is an action and λ the rate of the action duration. P has to be understood as the process that executes action a, which takes an exponentially distributed random amount of time with mean value $1/\lambda$, and which then behaves as P'.

Since the rate is the only information that has to be added to an action, we can define the syntax of a simple SPA as follows:

Definition 2.8 The language $\mathcal{L}_{\mathcal{SPA}}$ is defined by the following grammar:

$$P \quad \longrightarrow \quad \mathsf{stop} \quad | \quad X \quad | \quad (a, \lambda).P \quad | \quad P + P \quad | \quad P \|_S P \quad | \quad P \setminus H$$

where $a \in Act$, $\lambda \in \mathbb{R}^+$, $S \subseteq Act$, $H \subseteq Com$ and $X \in VAR$.

As we see, only the syntax of the prefix operators change: actions are now accompanied by a positive real value, the rate of the duration distribution.

We now give a further example to illustrate the behaviour of SPA processes.

Example 2.9

The process $P \stackrel{\text{def}}{=} (a, \lambda).P' + (b, \mu).P''$ for $P', P'' \in \mathcal{L}_{\mathcal{SPA}}$ executes either action a and behaves then as P', or action b and behaves then as P''. Assuming, that the environment has no influence (for example, by means of synchronisation), the actions a and b are here in a "race condition". The decision which action is chosen is determined by

the length of their respective duration: the quicker one wins. Although this sounds complicated, the choice can be described stochastically in terms of the rates λ and μ: the duration until one of the both actions is executed is still exponentially distributed with parameter $\lambda + \mu$. The probabilities which of the both actions is executed can be expressed in terms of the rates, and they are equal to $\frac{\lambda}{\lambda+\mu}$ for action a and $\frac{\mu}{\lambda+\mu}$ for b. Again, we assume that an action is atomic: it is executed either completely, or not at all. Hence, the action that looses a choice leaves no traces.

The recursion and hiding operators are used as in non-stochastic process calculi. The parallel composition of processes, however, is more difficult to treat. Although the most Markovian process algebras employ a CSP/LOTOS style synchronisation (from the functional point of view), they differ in the way in which durations are assigned to synchronising actions. In fact, the major differences between the different stochastic process algebras is how the temporal aspects for synchronisation are treated.

2.2.2 Variants of SPA

The established Markovian SPAs all use a parallel composition that is related to that of LOTOS: processes that are combined can proceed independently from each other, unless the action to be executed is member of a given set of actions on which all participating processes have to synchronise over. Such a synchronising action should, for consistency reasons, also have an associated duration distribution, represented by a rate. A synchronising action describes an activity that is executed by all participants. It is convenient to think of this as a *cooperation* between the participating processes, as coined by Hillston [73]. The problem is now to derive a rate that is appropriate to be assigned to such an synchronised action.

In Figure 2.1, the problem is illustrated: we have two processes, $P \stackrel{\text{def}}{=} (a, \lambda).P'$ and $Q \stackrel{\text{def}}{=} (a, \mu).Q'$, and we consider the process $(a, \lambda).P' \|_a (a, \mu).Q'$. Clearly, $(a, \lambda).P \|_a (a, \mu).Q$ should be able to make an a-transition, but which rate should be assigned? For the moment all we want to assume is that it should be a value that is derived from the rates λ and μ by means of some function f. Intuitively, a cooperation should start, if all individual components that must be involved in the cooperation are ready to do so, and it should end when all components participating in the cooperation are ready with their local contribution. If $X_1, X_2, \ldots, X_n$ are exponentially distributed random variables which express the duration of the participants of a cooperation, the duration of the cooperation itself should then be expressed as $\max\{X_1, X_2, \ldots, X_n\}$. Hence, we could consider to assign $1/E[\max\{X_1, X_2, \ldots, X_n\}]$ as the rate of the cooperation. For the example in Figure 2.1, then $f(\lambda, \mu) = f_{\max}(\lambda, \mu) = E[\max\{X_\lambda, X_\mu\}]^{-1}$, where X_λ and X_μ are exponentially distributed random variables with rate λ and μ, respectively.

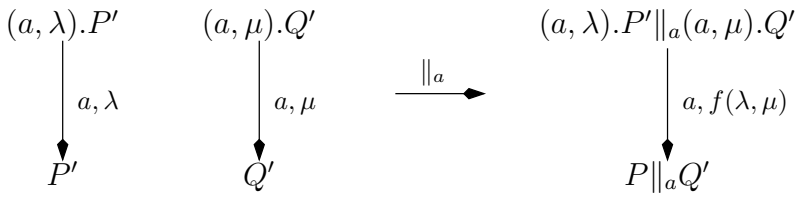


Figure 2.1: Cooperation between processes

However appealing this idea seems to be, it does not work. As has been discovered by Götz [58], the function f must be associative (*i.e.*, $f(x,(f(y,z)) = f(f(x,y,),z))$ and distributive with respect to $+$ ($f(x,y+z) = f(x,y) + f(x,z)$). The function $f_{\max}$, however, does *not* respect the associativity of the parallel operator.

The difference between the different Markovian process algebras lies in the definitions of the function f. For TIPP, $f = f_{\text{TIPP}}$ is the ordinary real number multiplication. For PEPA, the definition of $f = f_{\text{PEPA}}$ is a bit more involved. We define the apparent rate $r_a(P)$ of a process P with respect to action a as follows:

$$\begin{aligned}
r_a((b,\lambda) &= \begin{cases} \lambda & \text{if } b = a \\ 0 & \text{otherwise.} \end{cases} \\
r_a(P+Q) &= r_a(P) + r_a(Q) \\
r_a(P \setminus H) &= \begin{cases} r_a(P) & \text{if } a \notin H \\ 0 & \text{otherwise.} \end{cases} \\
r_a(P\|_S Q) &= \begin{cases} r_a(P) + r_a(Q) & \text{if } a \notin S \\ \min\{r_a(P), r_a(Q)\} & \text{otherwise.} \end{cases}
\end{aligned}$$

Then, the cooperation of two PEPA components is defined by the rule

$$\frac{P \xrightarrow{(a,\lambda)} P' \; Q \xrightarrow{(a,\mu)} Q'}{P\|_S Q \xrightarrow{(a,R)} P'\|_S Q'} (\text{for } a \in S),$$

where

$$R = f_{\text{PEPA}}(\lambda,\mu) = \frac{\lambda}{r_a(P)} \frac{\mu}{r_a(Q)} \min\{r_a(P), r_a(Q)\}.$$

A more thorough introduction in the treatment of synchronisation in stochastic process algebras and a discussion about the different approaches chosen in the Markovian case can be found in [20].

2.2.3 Delays vs. Durations

Hermanns and Rettelbach have proposed a different view on the quantitative extensions of SPA [68, 66]. The specified times are not seen as durations of actions, but as *delays*. Actions themselves are considered as *timeless*. For the prefix, choice, recursion, and hiding operators this does not make any difference to the other paradigm. However, for parallel composition, this has important consequences: synchronisation takes place between timeless actions and is therefore timeless as well. There is no cooperation anymore, but only synchronisation.

In [66], this concept has been generalised to a strict separation between timed and untimed actions: timed actions are always local to a component in the sense that they can not be used for synchronisations. Timed actions are not visible from the outside. Untimed actions, on the other hand, are visible (although there are exceptions) and they can be synchronised with other visible actions.

Even though the renunciation of cooperation does eliminate the problem to find a "reasonable" rate for the synchronising actions, this approach does have its own peculiarity: *nondeterminism*.

Nondeterminism is a very important concept in (classical) process algebra. It allows to specify behaviours which inner mechanisms are not yet fully known or are not required to be implemented in full detail. In such cases, a model is said to be underspecified. On the other hand, in stochastic process algebra, nondeterminism is a nuisance: a nondeterministic SPA model is *underspecified* and it is not possible to analyse it, since parameters are missing. We will comment on this problem in Section 3.1.5 in more detail.

2.2.4 Equivalences

As for the non-stochastic case, also for stochastic process algebras congruences have to be defined. Generally, the defined congruences do combine the pure, *functional* bisimulation, as defined in Definition 2.7, with the concept of *ordinary lumpability*, which is a concept known from the area of continuous time Markov chains [87]. In Section 3.1.4 we will find examples for bisimulations of this type.

2.3 Performance Evaluation with SPA

As described in Section 2.1.2, the semantics of process algebra specifications is given in terms of transition systems. For the Markovian SPA that do not distinguish between timed and untimed actions, like TIPP and PEPA, all transitions are labelled with actions as well as rates. The transition system completely describes the CTMC of the considered SPA process, and only the action labels and self-loops must be eliminated from it (self-loops do

not have influence on the behaviour of a CTMC). From the resulting transition system a generator matrix of the CTMC can be obtained most easily.

For the Markovian SPA that distinguishes between timed and untimed actions, there has more to be done: the untimed transitions have to be eliminated from the transition system.

For IMC, this is done in several steps. We assume an IMC process P, from which we want to derive a Markov chain. We also assume that there is a transition system T for P has been derived already. The steps are:

Hiding of all visible actions. Instead of P, $P \backslash Com$ is considered, *i.e.,* all visible actions are hidden away. All action labels in T are hence converted to the internal action, τ.

State space reduction. Then, for T a transition system, T' is derived that is weakly bisimilar to T and that is *minimal*, *i.e.,* there is no transition system T'' that is weakly bisimilar to T and that has less states than T'.

If P is completely specified and does not show nondeterministic behaviour, then T' does only contain timed transitions and can be converted to a CTMC.

Once a CTMC is derived from an SPA transition system, a transient or steady-state probability distribution can be derived, as described in Appendix B.3.2.

2.3.1 Expressing Performance Measures

All performance measures of an Markovian SPA model can be expressed in terms of the steady-state solution of the underlying CTMC. However, before measures can be computed, they must be specified. For this purpose, *rewards* [80] have proven to be valuable additions of Markov models [64]. Let $\{X_t\}$ be a CTMC with state space S. Following [135], a reward specification is a function $r : S \longrightarrow \mathbb{R}$, that assigns real values, the *rewards*, to the CTMC states. The most interesting steady-state reward measures is the *expected reward* of the Markov chain, which is defined as

$$E[M] = \sum_{i \in S} r_i \pi(i),$$

where $\pi(i)$ is the steady-state probability of state i of the CTMC.

Since rewards are a well accepted approach to specify measures of interest, they should also be used in the area of performance analysis with SPA. Till now, two fundamentally different approaches for the specification of rewards for SPA models have been proposed. The first [34, 36, 35] uses a separate language based on a temporal logic to assign rewards to states. The second [9, 11] assigns rewards in the course of the state space construction to *transitions* (although, after the generation, the rewards are interpreted for states).

The fact that we can describe all relevant steady-state performance measures by means of rewards does allow us to neglect the actual derivation of performance measures for SPA

models. It is sufficient to concentrate on the methods to derive the steady-state probability vector, since this task is completely independent from the question how more expressive performance measures can be derived from them.

2.4 Conclusions

In this chapter, we gave an introduction in the basic concepts of process algebras and stochastic process algebra. We have discussed different variants of process algebras, methods for the definition of semantics, the role of congruences, and finally, the stochastic extensions of process algebras that are known until now.

The development of process algebras with stochastic extensions is not over yet. The most recent approach, the modelling language MoDeST [42], integrates the benefits of several different formalisms for the modelling of stochastic, probabilistic, and non-deterministic systems. The purpose of the language is to provide an easy-to-use formalism with well-defined formal semantics, that can be evaluated either by ordinary or probabilistic verification, model-checking, numerical performance analysis or simulation.

Chapter 3

Stochastic Process Calculus $\mathcal{YAWN}$

In this chapter, we will define a formalism for the stochastic and functional specifications of distributed systems: YAWN-MPA(**Y**et **A**nother **W**ell-defined **N**ondeterministic **M**arkovian **P**rocess **A**lgebra), abbreviated $\mathcal{YAWN}$. $\mathcal{YAWN}$ is a stochastic process algebra very similar to IMC, defined by Hermanns [66], but there are some differences of technical nature. All properties of, all techniques developed for and all results derived from IMC should, however, also hold for $\mathcal{YAWN}$.

The purpose of this chapter is to create a vehicle which is suitable to transport the ideas that we will develop in the subsequent chapters. Where appropriate, we will refer to the results of Hermanns [66] or other authors. Only special properties of $\mathcal{YAWN}$ are treated in more detail.

Outline of this Chapter. In Section 3.1, we introduce syntax, semantics and equivalences of $\mathcal{YAWN}$. In Section 3.2 we define how we derive continuous-time Markov chains from $\mathcal{YAWN}$ processes. In Section 3.3, we conclude the chapter.

3.1 $\mathcal{YAWN}$

In this section, we introduce the basic definitions for $\mathcal{YAWN}$. In Section 3.1.1, we define the action set, which has some unusual properties, compared with other SPA. In Section 3.1.2, we define the transition systems that we will use to define the semantics of $\mathcal{YAWN}$. In Section 3.1.3, the syntax and operational semantics of $\mathcal{YAWN}$ is introduced. In Section 3.1.4, we introduce two different notions of bisimulation, which we need later in the thesis. Since $\mathcal{YAWN}$ is similar to IMC, nondeterminism is also an issue for $\mathcal{YAWN}$. We comment on this in Section 3.1.5. In Section 3.1.6, we define a class of *finite* $\mathcal{YAWN}$ processes. We conclude with an example in Section 3.1.7.

3.1.1 Actions

As already described in the previous chapter, *actions* are the basic entities of process algebras. By *Com* we denote the set of *visible actions*, as defined in Section 2.1.2.

Internal Actions

We define **iset** to be the set of *internal actions*. **iset** is defined as

$$\mathbf{iset} = \{\mathbf{i}_a \mid a \in Com\} \cup \{\mathbf{i}_\tau\}$$

Each internal action corresponds directly to a visible action from *Com*, except $\mathbf{i}_\tau$, which purpose is described later. The reason to define a whole set of internal actions for $\mathcal{YAWN}$, where other SPA can rely on only one, is the following: the hiding operator of process algebras usually renames visible actions to one internal action. Once an action is hidden, it is no longer possible to say where the hidden action came from. Later, in Chapter 5, we need this information. Therefore, we will define the hiding operator in a way that an action a will be converted to internal action $\mathbf{i}_a$. An action $\mathbf{i}_a$ has all the properties of an "ordinary" internal action (*i.e.,* synchronisations over $\mathbf{i}_a$ are not possible), but nevertheless, it is still possible to see where it came from, since the former action name gets preserved.

We will often refer to *the* internal action $\mathbf{i}$, which has to be understood as a wild card for an arbitrary element of **iset**. As usual, $\mathbf{i}$ is said to be *the* internal action, which, contrary to the actions from *Com*, is considered to be not observable.

We define $Act = Com \cup \mathbf{iset}$.

Timed Action

Another special action needed for $\mathcal{YAWN}$ is $\mathbf{t}$. Where all actions from *Act* are considered to have *no duration*, $\mathbf{t}$ is an action that explicitly denotes the passing of time, although the time itself is not specified. How the time is assigned to a $\mathbf{t}$-action will be shown in Definition 3.1.

We define $Com_{\mathbf{t}} = Com \cup \{\mathbf{t}\}$ and $Act_{\mathbf{t}} = Act \cup \{\mathbf{t}, \mathbf{i}_{\mathbf{t}}\}$. The internal action $\mathbf{i}_{\mathbf{t}}$ is needed, since we will allow to hide timed transitions. We will come back to this issue in Section 3.1.3.

3.1.2 Generalised Markovian Transition Systems

The semantics of $\mathcal{YAWN}$ processes is given in terms of transition systems. The transition systems we use for this are defined as follows:

Definition 3.1 A *generalised Markovian transition system* (GMTS) is a tuple $(S, A, T, \mathcal{R})$, where

- S is a set of states;
- A is a set of labels;
- $T \subseteq S \times A \times S$ is a set of labelled *transitions*;
- $\mathcal{R} : T \longrightarrow \mathbb{R}^+ \cup \{\infty\}$ is a function that assigns transition rates to transitions.

Typical elements of S are $s, s', s'', s_1, s_2, \ldots$, and typical elements of T are $t, t', t'', t_1, t_2, \ldots$. Transitions labelled with $\mathbf{t}$ are meant to be exponentially distributed time delays. The function $\mathcal{R}$ specifies the rates of the distributions. A GMTS is said to be *properly timed*, if, whenever $t \in T$ with $t = (s, a, s')$ and $a \in Act$ (*i.e.,* for all actions $a \neq \mathbf{t}$), then $\mathcal{R}(t) = \infty$. Hence, all internal or visible actions are considered to have no duration, which is expressed by assigning them an infinite rate. .

Definition 3.2 We call a GMTS $(S, A, T, \mathcal{R})$ together with a state $s \in S$ (starting state) a *generalised Markovian process* (GMP). We denote a GMP by a five-tuple $(S, A, T, \mathcal{R}, s)$.

Some Useful Notations

To access information from a GMTS more easily, we introduce the following notation. Let $G = (S, A, T, \mathcal{R})$ be a GMTS.

- The set S is said to be the *state space* from G (abbreviated as $SP(G)$).
- If $t = (s, a, s') \in T$, then we define
  - $src(t) = s$ (*source state* of t);
  - $dst(t) = s'$ (*destination state* of t);
  - $lbl(t) = a$ (*label* of t).

  When a transition t has label a then we say that t is an a-transition.
- We adopt the common notation for transitions: if there is a transition $(s, a, s') \in T$, then we write $s \xrightarrow{a} s'$
- A *finite path* of length $n-1$ (for $n > 0$) through the transition system is an alternating sequence of states and labels $s_1, a_1, s_2, a_2, s_3, a_3, s_4, a_4, s_5, \ldots, a_{n-1}, s_n$ such that for all i with $1 \leq i < n$ holds: $s_i \xrightarrow{a_i} s_{i+1}$. We say that state s' is *reachable* from s, if there is a $n \in \mathbb{N}$ and a path of length n such that $s = s_1$ and $s' = s_{n+1}$.

- We write $s \overset{\mathbf{i}}{\twoheadrightarrow\!\!\twoheadrightarrow} s'$, if s' is reachable from s via a path σ where the actions of σ are all equal to $\mathbf{i}$. Since paths can be of length 0, the trivial case is included, *i.e.,* $s \overset{\mathbf{i}}{\twoheadrightarrow\!\!\twoheadrightarrow} s$ for all $s \in S$.

- Accordingly, we write $s \overset{\mathbf{t}}{\twoheadrightarrow\!\!\twoheadrightarrow} s'$, if s' is reachable from s via a path σ where the actions of σ are all equal to $\mathbf{t}$. The trivial case is included, *i.e.,* $s \overset{\mathbf{t}}{\twoheadrightarrow\!\!\twoheadrightarrow} s$ for all $s \in S$.

- For $a \in Com$, we write $s \overset{a}{\twoheadrightarrow\!\!\twoheadrightarrow} s'''$, if there are states s', s'' such that $s \overset{\mathbf{i}}{\twoheadrightarrow\!\!\twoheadrightarrow} s'$, $s' \xrightarrow{a} s''$, and $s'' \overset{\mathbf{i}}{\twoheadrightarrow\!\!\twoheadrightarrow} s'''$.

- A transition $t \in T$ is said to be an *immediate* or *untimed* transition, if $lbl(t) \in Act$. t is said to be *timed*, if $lbl(t) = \mathbf{t}$.

- t is said to be a *hidden* or *internal* transition, if $lbl(t) = \mathbf{i}$.

- If $t = (s, a, s') \in T$ then $\boldsymbol{\alpha}(t) = \{a\}$, if $a \in Com$, else $\emptyset$. Hence, $\boldsymbol{\alpha}(t) \neq \emptyset$, if $lbl(a)$ is visible. For G, the GMTS in question, we define

$$\boldsymbol{\alpha}(G) = \bigcup_{t \in T} \boldsymbol{\alpha}(t).$$

  $\boldsymbol{\alpha}(G)$ is the set of all actions of GMTS G that are visible.

The states and transitions of GMP G can be classified as follows:

- The set of *synchronising states* $\mathcal{S}_{syn}(G) \subseteq S$ of G is defined as

$$\mathcal{S}_{syn}(G) = \left\{ s \in S \mid \exists s' \in S, \exists a \in Com : s \xrightarrow{a} s' \right\}$$

- A state s of G is called *stable*, if for all $t \in T$ with $src(t) = s$ we have $lbl(t) = \mathbf{t}$ and $\mathcal{R}(t) < \infty$[1].

- All states that are not stable are said to be *immediate* or *vanishing*.

- A *synchronising transition* is a transition with an action label that is visible. The set of synchronising transitions $\mathcal{T}_{syn}(G) \subseteq T$ is defined as

$$\mathcal{T}_{syn}(G) = \{t \in T \mid \boldsymbol{\alpha}(t) \neq \emptyset\}$$

[1] In the area of Generalised Stochastic Petri Nets, such states are called *tangible*.

3.1.3 Syntax and Semantics of $\mathcal{YAWN}$

We define now the language in which $\mathcal{YAWN}$ processes will be specified. We first define the set L of all process algebra expressions. An expression $P \in L$ is said to be closed iff every process variable, say X, occurring in P occurs within the scope of a $recX$ operator, and if every process constant is defined by a defining equation.

Definition 3.3 ($\mathcal{L}_{\mathcal{YAWN}}$) Let L be the language defined by the following grammar:

$$\mathsf{P} \longrightarrow \mathsf{stop} \mid X \mid A \mid a.\mathsf{P} \mid [\lambda].\mathsf{P} \mid \mathsf{P} + \mathsf{P} \mid recX : \mathsf{P} \mid \mathsf{P} \setminus H \mid \mathsf{P} \parallel_S \mathsf{P}$$

where $X \in VAR$, $A \in CONST$, $a \in Com \cup \{\mathbf{i}_\tau\}$, $\lambda \in \mathbb{R}^+$, $H \subseteq Com_{\mathbf{t}}$ and $S \subseteq Com$. Then $\mathcal{L}_{\mathcal{YAWN}} \subseteq L$ is the set of all closed process algebra expressions. Elements of $\mathcal{L}_{\mathcal{YAWN}}$ are said to be $\mathcal{YAWN}$ *processes*.

$CONST$ is a set of process constants and VAR is a set of process variables (*cf.* Section 2.1.1). a can be an action from Com, or the special internal action $\mathbf{i}_\tau \in \mathbf{iset}$. Note that we have unusual hiding operators, $\cdot \setminus H$. The set H is allowed to contain the timed action $\mathbf{t}$, which means, that we allow to hide the timed action. We will discuss this in detail after Definition 3.4.

We assume that the operators have the following precedence: prefix $>$ recursion $>$ hiding $>$ choice $>$ parallel composition, *i.e.,* prefix has precedence over recursion, recursion over hiding, etc. Parentheses can be used to circumvent these rules. If we have more than two processes combined (as, for example, in $P_1 + P_2 + P_3$ or $P_1 \| _S P_2 \| _{S'} P_3$ for $P_i \in \mathcal{L}_{\mathcal{YAWN}}$, $i = 1, 2, 3$) then we assume a left-associative evaluation order: $P_1 + P_2 + P_3$ and $P_1 \| _S P_2 \| _{S'} P_3$ are assumed to be equal to $(P_1 + P_2) + P_3$ and $(P_1 \| _S P_2) \| _{S'} P_3$, respectively. These rules determine a unique evaluation order, which later will become especially important for the application of SOS rules.

Please note that the $\mathcal{YAWN}$ language comes with bells and whistles: we allow to define recursion by means of process constants, and by $recX$ operators with process variables. The only reason for this is to have a more convenient syntax for $\mathcal{YAWN}$.

Frequently, we have to compare elements of the $\mathcal{YAWN}$ language syntactically. For two terms $P, Q \in \mathcal{L}_{\mathcal{YAWN}}$, we define $P \equiv Q$, iff P and Q are syntactically equal.

Since we have already described the informal meaning of the operators in Chapter 2, we proceed now immediately with the semantics. This, again, is given in the style of an SOS. The derivation rules are listed in Table 3.1.

The semantics of all $\mathcal{YAWN}$ processes is described by a GMTS $G_{\mathcal{YAWN}}$. The definition of $G_{\mathcal{YAWN}}$ comes in two stages. In the first stage, the actual transition system of the GMTS is constructed, *i.e.,* the set of transitions T. In the second stage, the function $\mathcal{R}$ is defined.

$$1)\quad \frac{}{a.P \xrightarrow{a} P} \qquad 2)\quad \frac{}{[\lambda].P \xrightarrow{\mathbf{t}} P}$$

$$3)\quad \frac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'} \qquad 4)\quad \frac{Q \xrightarrow{a} Q'}{P + Q \xrightarrow{a} Q'}$$

$$5a)\quad \frac{P\{recX : P/X\} \xrightarrow{a} P'}{recX : P \xrightarrow{a} P'} \qquad 5b)\quad \frac{P \xrightarrow{a} P'}{A \xrightarrow{a} P'}\ (A \stackrel{\text{def}}{=} P)$$

$$6)\quad \frac{P \xrightarrow{a} P'}{P \|_S Q \xrightarrow{a} P' \|_S Q}(a \notin S) \qquad 7)\quad \frac{Q \xrightarrow{a} Q'}{P \|_S Q \xrightarrow{a} P \|_S Q'}(a \notin S)$$

$$8)\quad \frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{a} Q'}{P \|_S Q \xrightarrow{a} P' \|_S Q'}(a \in S)$$

$$9)\quad \frac{P \xrightarrow{a} P'}{P \setminus H \xrightarrow{\mathbf{i}_a} P' \setminus H}(a \in H) \qquad 10)\quad \frac{P \xrightarrow{a} P'}{P \setminus H \xrightarrow{a} P' \setminus H}(a \notin H)$$

Table 3.1: SOS rules for $\mathcal{YAWN}$

First Stage

Definition 3.4 ($G_{\mathcal{YAWN}}$, Stage I) We define $G_{\mathcal{YAWN}} = (\mathcal{L}_{\mathcal{YAWN}}, Act_{\mathbf{t}}, T, \mathcal{R})$ to be the least GMTS that satisfies the rules of Table 3.1.

Actually, we must define $G_{\mathcal{YAWN}}$ always relative to a set of defining equations that gives meaning to process constants. Since definitions of process constants can differ, the definition of $G_{\mathcal{YAWN}}$ is only unique with respect to these equations. In the following we will always make sufficiently clear which set of defining equations we consider, such that confusion shall never be possible.

Unusual for this semantics of $\mathcal{YAWN}$ is that we allow the *hiding of timed transitions*. This feature allows us to strip all timing information from a $\mathcal{YAWN}$ process, leaving behind a process that shows only functional behaviour. As a consequence, we later will be able to compare $\mathcal{YAWN}$ processes *only* with respect to their functional behaviour. Under normal circumstances, though, it will neither be necessary nor wise to use this feature; only in Chapter 5 we will find an application for it.

Second Stage

In the second stage, we define the function $\mathcal{R}$. To do so, we have to introduce an auxiliary concept. Each transition of $G_{\mathcal{YAWN}}$ is derived by successive application of the derivation rules of Table 3.1. The applied rules form a derivation tree (*cf.* [110]). We are now interested in timed transitions, *i.e.,* those transitions with label $\mathbf{t}$. It is possible that for a transition more than one derivation tree exists. For example, the transition

$$[12].\mathsf{stop} + [10].\mathsf{stop} \xrightarrow{\mathbf{t}} \mathsf{stop}$$

is derived by

1. Rule 2) (axiom), then Rule 3):

$$3) \quad \frac{[12].\mathsf{stop} \xrightarrow{\mathbf{t}} \mathsf{stop}}{[12].\mathsf{stop} + [10].\mathsf{stop} \xrightarrow{\mathbf{t}} \mathsf{stop}}$$

2. Rule 2) (axiom), then Rule 4):

$$4) \quad \frac{[10].\mathsf{stop} \xrightarrow{\mathbf{t}} \mathsf{stop}}{[12].\mathsf{stop} + [10].\mathsf{stop} \xrightarrow{\mathbf{t}} \mathsf{stop}}$$

If $t \in T$ is a transition, then we define DT_t to be the set of all derivation trees of t. For all timed transitions of $G_{\mathcal{YAWN}}$, the derivation tree starts with exactly one axiom of type 2). We define the multiset

$$DVR_t = \{\!|\lambda \mid [\lambda].P \xrightarrow{t} P \text{ is the axiom of a derivation tree } d \in DT_t|\!\}$$

and let $\rho(t) = \sum_{\lambda \in DVR_t} \lambda$ be the sum of all values in DVR_t. For the above example,

$$t = [12].\mathsf{stop} + [10].\mathsf{stop} \xrightarrow{\mathbf{t}} \mathsf{stop},$$

we have $DVR_t = \{\!|10, 12|\!\}$ and the sum of the rates is $\rho(t) = 22$.

We are now ready to enter the second stage of the definition of $G_{\mathcal{YAWN}}$:

Definition 3.5 ($G_{\mathcal{YAWN}}$, Stage II) The function $\mathcal{R}$ of $G_{\mathcal{YAWN}}$ is defined as follows:

$$\mathcal{R} : \begin{cases} T & \longrightarrow & \mathbb{R}^+ \cup \{\infty\} \\ t & \longmapsto & \begin{cases} \infty & \text{if } lbl(t) \in Act \\ \rho(t) & \text{if } lbl(t) = \mathbf{t} \end{cases} \end{cases}$$

A Bit more Notation

The *derivatives* of a term $P \in \mathcal{L}_{\mathcal{YAWN}}$ are all those elements of $\mathcal{L}_{\mathcal{YAWN}}$ that can be reached by a path starting with P. We call the set of these terms $Reach(P)$, the *reachability set*.

With the reachability set of a process $P \in \mathcal{L}_{\mathcal{YAWN}}$, a GMP which describes the behaviour of P is implicitly defined.

Definition 3.6 For $P \in \mathcal{L}_{\mathcal{YAWN}}$, $[\![P]\!] = (Reach(P), Act_t, T_P, \mathcal{R}_P, P)$ is said to be the *global GMP* of P, where $T_P = T \cap (Reach(P) \times Act_{\mathbf{t}} \times Reach(P))$ and $\mathcal{R}_P = \mathcal{R}\big|_{T_P}$.

It is obvious that $[\![P]\!]$ contains exactly the information to describe the behaviour of P.

To keep notation simple, we write $\boldsymbol{\alpha}(P)$ instead of $\boldsymbol{\alpha}([\![P]\!])$ to denote the set of visible actions of a global GMP $[\![P]\!]$. We say that P is *non-interactive* if $\boldsymbol{\alpha}(P) = \emptyset$.

3.1.4 Notions of Equivalence

In this section, we define under which circumstances we assume two $\mathcal{YAWN}$ processes to be equivalent (and substitutive) in their behaviour. Since GMP are very similar to IMC transition systems, we can adopt the definitions from [66].

The congruences we are going to define are strong Markovian bisimulation and weak Markovian congruence.

Strong Markovian Bisimulation

To define strong Markovian bisimulation, we first need a function γ_M that sums up all rates from transitions that start in a single state s and end in some state in a set C.

Definition 3.7 (γ_M) Let $(S, A, T, \mathcal{R})$ be a GMTS and for $s \in S$ and $C \subseteq S$, let

$$T_C^s = \{t \mid t \in \{s\} \times \{\mathbf{t}\} \times C\}.$$

Then the function γ_M is defined as

$$\gamma_M : \begin{cases} S \times 2^S & \longrightarrow \quad \mathbb{R} \\ (s, C) & \longmapsto \quad \sum_{t \in T_C^s} \mathcal{R}(t). \end{cases}$$

Example 3.8

Consider a GMTS with states s, s_1, s_2, s_3, s_4 and the state sets $C_1 = \{s_1, s_2\}$ and $C_2 = \{s_3, s_4\}$. In Figure 3.1, we see transitions going from s to s_i for $i = 1, \ldots, 4$.

Then,

$$\gamma_M(s, C_1) = \lambda + \mu \quad \text{and} \quad \gamma_M(s, C_2) = 2\lambda.$$

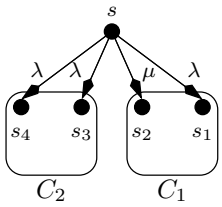


Figure 3.1: Illustration of Example 3.8

Definition 3.9 (Strong Bisimulation) An equivalence relation $\mathrm{R} \subseteq \mathcal{L}_{\mathcal{YAWN}} \times \mathcal{L}_{\mathcal{YAWN}}$ is a *strong Markovian bisimulation* iff $P \ \mathrm{R} \ Q$ implies for all $a \in Act_{\mathbf{t}}$ and all equivalence classes C of R:

1. if $P \xrightarrow{a} P'$ then $Q \xrightarrow{a} Q'$ and $P' \ \mathrm{R} \ Q'$;
2. if $P \not\xrightarrow{\mathbf{i}}$ then $\gamma_M(P, C) = \gamma_M(Q, C)$.

P and Q are said to be strongly Markovian bisimulation equivalent ($P \sim Q$) iff there is a strong Markovian bisimulation R such that $P \ \mathrm{R} \ Q$.

It can be shown that strong Markovian bisimulation equivalence is a congruence with respect to all language operators. For a proof we refer to [66].

Expansion Law

The following law expresses the most basic principle of the operational semantics of process algebras. It states that for each parallel composition of "sums" of processes P (where the choice operator takes the role of the sum here) there exists a process P' such that $P \sim P'$ and P' is the "sum" of parallel compositions. This means that parallelism is not represented explicitly, but encoded by the choice operator.

Lemma 3.10 (Expansion Law) Let

$$P = \sum_I [\lambda_i].P_i + \sum_J a_j.P_j$$

and

$$Q = \sum_K [\mu_k].Q_k + \sum_L b_l.Q_l$$

where i, j, k, l range over the respective index sets I, J, K, L. Let $S \subseteq Com$. Then

$$\begin{aligned} P\|_S Q \quad \sim \quad & \sum_I [\lambda_i].(P_i\|_S Q) + \sum_{a_j \notin S} a_j.(P_j\|_S Q) \\ & + \sum_K [\mu_k].(P\|_S Q_k) + \sum_{b_l \notin S} b_l.(P\|_S Q_l) \\ & + \sum_{\substack{A\cap B\cap S \\ a_j = b_l}} a_j.(P_j\|_S Q_l), \end{aligned}$$

where $A = \{a_j | j \in J\}$, $B = \{b_l | l \in L\}$.

Weak Markovian Congruence

The weak Markovian congruence does abstract from internal actions. To treat internal transitions properly, we need the following definition:

Definition 3.11 ($C^{\mathbf{i}}$) Let $(S, A, T, \mathcal{R})$ be a GMTS and $C \subseteq S$. The *internal backward closure* $C^{\mathbf{i}}$ is defined as

$$C^{\mathbf{i}} = \{\ s' \in S \mid \exists s \in C : s' \stackrel{\mathbf{i}}{\twoheadrightarrow\!\!\!\!\twoheadrightarrow} s\}$$

A first approach towards the definition of weak Markovian congruence is *weak Markovian bisimulation*.

Definition 3.12 (Weak Markovian Bisimulation) An equivalence relation R with $\mathrm{R} = \subseteq \mathcal{L}_{\mathcal{YAWN}} \times \mathcal{L}_{\mathcal{YAWN}}$ is called a weak Markovian bisimulation iff $P \mathrm{R} Q$ implies for all $a \in Act$ and all equivalence classes C of R

1. $P \stackrel{a}{\twoheadrightarrow\!\!\!\!\twoheadrightarrow} P'$ implies $Q \stackrel{a}{\twoheadrightarrow\!\!\!\!\twoheadrightarrow} Q'$ for some $Q' \in \mathcal{L}_{\mathcal{YAWN}}$ with $P' \mathrm{R} Q'$.
2. $P \stackrel{\mathbf{i}}{\twoheadrightarrow\!\!\!\!\twoheadrightarrow} P'$ and $P' \not\stackrel{\mathbf{i}}{\longrightarrow}$ imply $\gamma_M(p, C^{\mathbf{i}}) = \gamma_M(q, C^{\mathbf{i}})$.

P and Q are called *weakly Markovian bisimulation equivalent* ($P \approx Q$) if there is a weak Markovian bisimulation R such that $P \mathrm{R} Q$.

In [66], Hermanns has shown that $\approx$ is a congruence for all IMC operators (and hence also for $\mathcal{YAWN}$), except for the choice operator. The reasons for this are well known due to Milner [110], and the deficiency is fixed with the following definition:

Definition 3.13 (Weak Markovian Congruence) P and Q are said to be *weakly Markovian congruent* ($P \simeq Q$) iff for all $a \in Act$, all $C \in \mathcal{L}_{\mathcal{YAWN}}/\approx$:

1. $P \stackrel{a}{\longrightarrow} P'$ implies $Q \stackrel{a}{\twoheadrightarrow\!\!\!\!\twoheadrightarrow} Q'$ for some $Q' \in \mathcal{L}_{\mathcal{YAWN}}$ and $P' \approx Q'$;

2. $Q \xrightarrow{a} Q'$ implies $P \stackrel{a}{\twoheadrightarrow\twoheadrightarrow} P'$ for some $P' \in \mathcal{L}_{\mathcal{YAWN}}$ and $P' \approx Q'$;

3. P stable $\implies \gamma_M(P, C) = \gamma_M(Q, C)$;

4. P stable $\iff$ Q stable.

Theorem 3.14 Weak congruence is a congruence with respect to prefix, choice, parallel composition and recursion.

PROOF: We refer to [66]. →•

Weak Markovian congruence can be used to minimise the state space of a process. A minimal GMP (with respect to weak Markovian congruence) is one which state space does not contain states s, s' such that $s \simeq s'$. Lifting this property to the set of process terms yields the notion of *minimal representant*.

Definition 3.15 (Minimal Representant) Let $P \in \mathcal{YAWN}$. A minimal representant of P with respect to weak Markovian congruence is a process $P_{\min} \in \mathcal{YAWN}$ such that $P \simeq P_{\min}$ and there is no process $P' \in \mathcal{YAWN}$ such that $P' \simeq P$ and $\#Reach(P') < \#Reach(P_{\min})$.

Note that $P_{\min}$ is not necessarily unique.

Example 3.16

Consider $P \stackrel{\text{def}}{=} a.b.\mathsf{stop} + b.a.\mathsf{stop}$ and $Q \stackrel{\text{def}}{=} a.\mathsf{stop} \| b.\mathsf{stop}$. $P \simeq Q$ and both P and Q are minimal. However, their state space is completely different.

Although there can be different minimal representants of a process P, when we abstract from the syntactic properties of the states, we find that two weakly congruent minimal representant have an isomorphic transition system.

A Note on Hiding

As mentioned before, we allow the hiding operators to hide away timed transitions. Unfortunately, this breaks the congruence property of $\simeq$ with respect to hiding. We show this by means of the following example.

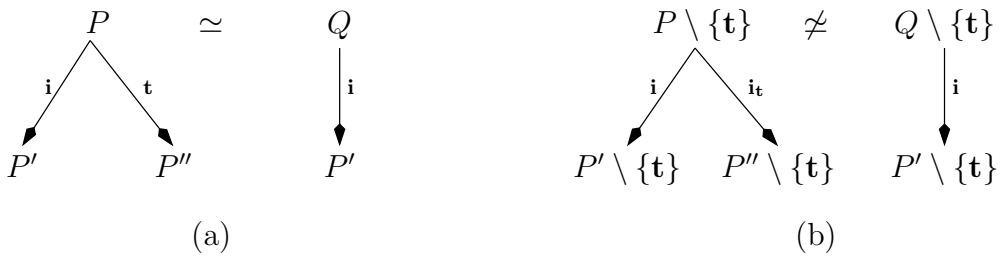


Figure 3.2: Branching Structure of P and Q (Example 3.17)

Example 3.17

We consider the process $P = \mathbf{i}.P' + [\lambda].P''$ and $Q = \mathbf{i}.P'$ (see Figure 3.2 (a)). Without loss of generality we assume that $P' \not\simeq P''$. Certainly, $P \simeq Q$. However, generally $P \setminus \{\mathbf{t}\} \not\simeq Q \setminus \{\mathbf{t}\}$. The reason is that implicitly immediate transitions in choice with timed transitions have priority in execution (maximal-progress assumption). This is the reason why $P \simeq Q$ — the process P'' can never be reached from P and hence the complete branch can be eliminated without loss. On the other hand, if we hide the timed transitions, then the label is turned from $\mathbf{t}$ to $\mathbf{i_t}$. Since we have not agreed on any precedence on different internal actions, both branches for $P \setminus \{\mathbf{t}\}$ in Figure 3.2 (b), left part, have the same priority. Hence, $P \longrightarrow P''$ has become a legal state change. $Q \setminus \{\mathbf{t}\}$, on the other hand, does not have the option to evolve into P''.

Generally, the lack of the congruence property for an operator is not desirable. However, in this case the only application of hiding-of-timed-transitions will occur in Section 5.2. There, we exclude explicitly the possibility of a choice between a timed and an internal action.

3.1.5 Nondeterminism

Nondeterminism is an important concept in process algebra. It allows to specify behaviours of which the inner mechanisms are not fully known or which are not required to be implemented in full detail, or to describe different behaviours from which the environment (*i.e.*, other components) can choose from. A nondeterministic model is also said to be *underspecified*.

Example 3.18

A communication protocol allows to send packets from a sender to a receiver. Packets are acknowledged by the receiver, hence the sender is receiving acknowledgement

messages. Packets from sender to receiver might get lost. In this case the receiver informs the sender with an error message.

On a certain level of abstraction, there is no difference between an error message and an acknowledgement: both are packets that are received by the sender. Both, however, require different treatment. So the receiver part of the sender could be modelled as follows:

$$\begin{aligned} \mathit{Sender} \stackrel{\text{def}}{=} \; & \texttt{send\_packet}.\mathit{Sender} \\ & + \texttt{receive\_ack}.\mathit{TreatAck} \\ & + \texttt{receive\_err}.\mathit{TreatErr} \end{aligned}$$

Here, the actions `receive_ack` and `receive_err` denote the acceptance of an inbound acknowledgement or error packet. The process constants *TreatAck* and *TreatErr* are not defined here, but denote the appropriate handling of acknowledgement and error packages. The nondeterminism in the example is the fact that here the three actions `send_packet`, `receive_ack` and `receive_err` are in choice. In this case, the nondeterminism could be resolved by the environment, since a synchronisation could decide which action to choose.

With the advent of SPAs which differentiate between timed and synchronising actions (as IMC), nondeterminism can become a problem. To compute a steady-state solution for an SPA model, the model must not be underspecified. It is necessary to describe a system in such detail that nondeterminism does not occur. This, however, is not always possible: there are situations where nondeterministic choices sneak in through the backdoor and can not be eliminated anymore. Nondeterminism can be introduced unconsciously by parallel composition of two or more processes.

Example 3.19

We consider the three processes

$$\begin{aligned} P_1 &= [\lambda].a.P_1 \\ P_2 &= [\lambda].a.P_2 \\ P_3 &= [\lambda].a.P_3. \end{aligned}$$

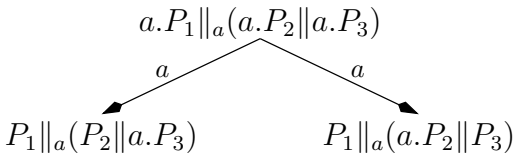
All three processes are doing basically the same and are, what is more important, deterministic. But the process

$$R = P_1 \|_a (P_2 \| P_3)$$

is nondeterministic: it can evolve to

$$R' = a.P_1 \|_a (a.P_2 \| a.P_3).$$

From this state, two a-transitions are possible:



Even by external influence, this nondeterministic choice can not be resolved: there is no way to choose between two identical actions.

This effect is called *auto-concurrency* and is an inherent problem for IMC (at least, if stochastic evaluation of the model is planned) and likewise for $\mathcal{YAWN}$.

To resolve the problem of nondeterminism and underspecification for stochastic systems, several approaches have been proposed. The simplest one is to change a nondeterministic choice in a *probabilistic* choice[2]. It is up to the modeller to distribute probability on the different branches of the nondeterministic choice. There is, however, the problem to find a criterion which allows to decide whether a chosen probability distribution is appropriate. Normally such a criterion can not be given. It is therefore common practice to distribute probability uniformly over all branches. Though this approach can be given a stochastic justification[3], the measures obtained from the model are only valid for this explicit choice to resolve nondeterminism by uniform distributions. Consequently, this approach has brought up some critique.

An alternative approach, that goes back to work of Vardi [136] has been proposed by De Alfaro [44]: instead of resolving the nondeterminism, it is retained and the model is to be seen as a *Markov decision process* [80, 123]. There are techniques to assess the (long run) average measures and bounds.

In this thesis we assume that we have a system that is not underspecified. We do not say that a system is deterministic in this case, since there are still decisions which are described stochastically. Hence, we say that a model that is not underspecified is *s-deterministic*.

In Section 3.2.1 we will come back to this issue once more.

[2]Neither IMC nor $\mathcal{YAWN}$ have mechanisms to assign probabilities to individual, immediate transitions. Nevertheless, we assume that such mechanisms can be defined and utilised. If we say it is a simple approach to assign probabilities to transitions, then we mean that the idea is simple (and appealing). However, technically the approach is by no means simple nor trivial. Several process algebras that allow for the mixture of stochastic specification of delays and explicit assignment of probabilities exist, *e.g.,* from Rettelbach [124], Katoen [84], and Bernardo et al. [13].

[3]We refer here to Bernoulli's *principle of insufficient reason* that states, following Harrison and Patel [60], that all events over a sample space should have the same probability unless there is evidence to the contrary. As an aside, this principle, combined with information theoretical considerations, has lead to the so-called *maximum entropy methods*. These are used to ensure that only that much information is derived from a model (or a measurement) as the model (measurement) actually can provide. In performance evaluation, maximum entropy methods have been used by Kouvatsos et al. [92] to derive results for general queueing networks.

3.1.6 Finiteness

To be able to derive measures from SPA models, it is generally necessary for them to have a finite state space. Although there are approaches to derive measures even for infinite state systems, they work only for a class of processes with a certain structure (Processes that have the Quasi-Birth-and-Death structure [112, 48]). Finiteness of PA models is generally not decidable, however, syntactic subclasses of the PA languages exist whose elements are only finite state and which are nevertheless expressive enough to describe realistic models. The basic trick is to disallow expressions of the form $P = recX : Q$, where Q contains one or more parallel operators. If, for example, $Q = recY : a.Y \| b.X$, then the expanded process $P\{recX : Q/X\} = recY : a.Y \| b.recX : Q$ can execute a b action and suddenly contain two processes $recY : a.Y$ running in parallel. Since the expansion of $recX : P$ can be performed an arbitrary number of times, always yielding more processes than before, the state space of P is obviously of infinite size.

In the rest of the dissertation, we will only consider so-called *communicating components*, which are guaranteed to have a finite reachability set.

Definition 3.20 (Systems of Communicating Components) Let L be the language defined by following grammar:

$$\begin{array}{lcl} \mathsf{C} & \longrightarrow & \mathsf{P} \mid \mathsf{C} \|_S \mathsf{C} \mid \mathsf{C} \setminus H \\ \mathsf{P} & \longrightarrow & \mathsf{stop} \mid X \mid a.\mathsf{P} \mid [\lambda].\mathsf{P} \mid \mathsf{P} + \mathsf{P} \mid recX : \mathsf{P} \mid A \end{array}$$

where $X \in VAR$, $A \in CONST$, $a \in Com \cup \{\mathbf{i}_\tau\}$, $\lambda \in \mathbb{R}^+$, $H \subseteq Com_{\mathbf{t}}$, and $S \subseteq Com$. The set of *systems of communicating components* is defined to be the set $\mathcal{L}_{\mathcal{CC}} \subseteq L$ of all closed expressions contained in L. Defining equations have to be of the form $A \stackrel{\text{def}}{=} P$, where P is a term described by the production P in the above grammar.

In important property of a $\mathcal{YAWN}$ process to be suitable for a stochastic analysis is *irreducibility*.

Definition 3.21 (Irreducibility) A GMTS $(S, A, T, \mathcal{R})$ is said to be *irreducible* iff for all states $s, s' \in S$ holds: s is reachable from s', *i.e.,* each state $s \in S$ can reach each other state $s' \in S$.

3.1.7 Example

In this section we develop a simple example to illustrate the concepts introduced so far. The example is a $\mathcal{YAWN}$ process that describes a Gordon-Newell queueing network [57]. The system comprises M queues of $M|M|1$ type. Since the system is closed, only a finite number K of customers circulates between the queues. Hence the queues only need finite buffer capacity with $K - 1$ places and the whole system can be described with finitely many states.

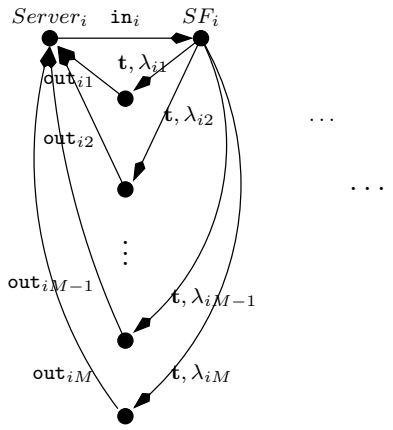


Figure 3.3: GMP describing server i

An individual queue is composed of a *server*, which does the actual work, and a *buffer*, that holds the jobs waiting for the server to become free.

The (idle) Server i can be described as follows:

$$Server_i \stackrel{\text{def}}{=} \mathtt{in}_i.SF_i,$$

where

$$SF_i \stackrel{\text{def}}{=} \sum_{j=1}^{M} [\lambda_{ij}].\mathtt{out}_{ij}.Server_i.$$

In Figure 3.3 we see a graphical representation of the GMP corresponding to the server description $Server_i$, *i.e.,* $[\![Server_i]\!]$. The server expects a signal to start a job over action $\mathtt{in}_i$ and changes state from $Server_i$ to SF_i when it arrives. State SF_i is stable: all outgoing transitions have label $\mathbf{t}$, and since we assume that all components are properly timed, they have an associated positive rate. Hence, state SF_i has a positive, exponentially distributed sojourn time, whose rate is the sum of the rates of all outgoing transitions. The mean sojourn time of this state is the equivalent of the mean service time of a queue in a GNQN. The reason to split the rate over several transitions is that a probabilistic choice has to be made, in which queue the processed job shall be enqueued when execution is done. Once one of the $[\lambda_{ij}]$ actions has been executed (say, $[\lambda_{ik}]$ for $k \in \{1, \ldots, M\}$), the server offers the job to the queue k via action $\mathtt{out}_{ik}$. In case $k = i$, the server routes the finished job back to its own queue.

The buffer of queue i is given as follows:

$$\begin{aligned} Buffer_i &\stackrel{\text{def}}{=} Buf_{i0} \stackrel{\text{def}}{=} \sum_{j=1}^{M} \mathtt{out}_{ji}.Buf_{i1} \\ &\vdots \\ Buf_{ik} &\stackrel{\text{def}}{=} \sum_{j=1}^{M} \mathtt{out}_{ji}.Buf_{i,k+1} + \mathtt{in}_i.Buf_{i,k-1} \\ &\vdots \\ Buf_{i,K-1} &\stackrel{\text{def}}{=} \mathtt{in}_i.Buf_{i,K-2} \end{aligned}$$

The corresponding GMP is depicted in Figure 3.4. The buffer of service station i is signalled

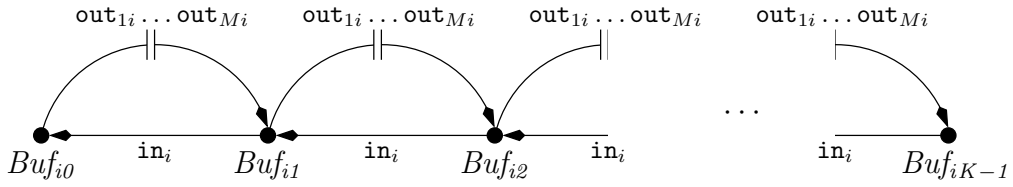


Figure 3.4: GMP describing buffer i

an incoming job via the actions $\mathtt{out}_{ji}$ for $j = 1, \ldots, M$, the same actions that have been already used in the specification of $Server_i$. The buffer can hold at most $K-1$ jobs and as soon as a job arrives the buffer tries to signal the server $Server_i$ via action $\mathtt{in}_i$ the possibility to get some work done. When we combine $Server_i$ and $Buffer_i$, we obtain the description of a complete, empty Queue, $Queue_i$:

$$Queue_i \stackrel{\text{def}}{=} (Server_i \|_{\{\mathtt{in}_i, \mathtt{out}_{ii}\}} Buffer_i) \setminus \{\mathtt{in}_i, \mathtt{out}_{ii}\}.$$

The set of visible actions of $Queue_i$ is

$$\begin{aligned} \boldsymbol{\alpha}(Queue_i) &= \{\mathtt{out}_{i1}, \ldots, \mathtt{out}_{i,i-1}, \mathtt{out}_{i,i+1}, \ldots, \mathtt{out}_{iM}\} \\ &\cup \{\mathtt{out}_{1,i}, \ldots, \mathtt{out}_{i-1,i}, \mathtt{out}_{i+1,i}, \ldots, \mathtt{out}_{Mi}\} \end{aligned}$$

Each of these actions can be seen as unidirectional channel between the different queues, over which jobs are transfered. Over the channel $\mathtt{out}_{ij}$ jobs are transfered from queue i to queue j.

A queue that is filled with K customers is described by

$$FullQueue_i \stackrel{\text{def}}{=} (SF_i \|_{\{\mathtt{in}_i, \mathtt{out}_{ii}\}} Buf_{i,K-1}) \setminus \{\mathtt{in}_i, \mathtt{out}_{ii}\}.$$

Then, a complete GNQN is described by

$$GNQN \stackrel{\text{def}}{=} (\ FullQueue_1 \|_{A_1} (Queue_2 \|_{A2} (\cdots (Queue_{M-1} \|_{A_{M-1}} Queue_M) \cdots)\ )$$

where $A_i = \{\mathtt{out}_{ij} \mid j = i+1, \ldots, M\}, i = 1, \ldots, M-1$. Process *GNQN* describes a queueing system, where, initially, K customers are in queue 1 and all other queues are empty.

3.2 GMP and Continuous-Time Markov Chains

The derivation of continuous-time Markov chains from stochastic process algebra models is a crucial step towards the numerical evaluation of the stochastic properties of the model. In this section we will define a procedure for this purpose for $\mathcal{YAWN}$.

Since $\mathcal{YAWN}$ is very similar to IMC, we will give an overview over the CTMC generation procedure of IMC in Section 3.2.1. Moreover, we justify why we do not adopt the IMC method. In Section 3.2.2, we present the procedure we have chosen for $\mathcal{YAWN}$ and which is based on an approach developed for Generalised Stochastic Petri Nets [103, 102].

The necessary notation and definitions for discrete-time and continuous-time Markov chains are summarised in Appendix B.3.

3.2.1 CTMC Derivation from IMC

Since IMC is the SPA most similar to $\mathcal{YAWN}$, it is worthwhile to describe CTMC generation for IMC and to point out why we do not regard the procedure as suitable for our purposes. The CTMC generation for IMC has two aims (besides the generation itself):

1. Minimisation of the state space by aggregating weakly bisimilar states;
2. Detection of nondeterministic behaviour.

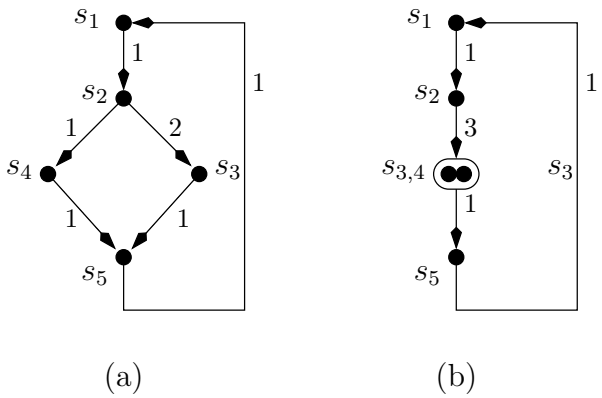
The first aim is to cushion the state space explosion. The second one is necessary to determine whether the IMC model is underspecified or not.

Minimisation has several advantages: the memory consumption is lower, and even though some models might show nondeterministic behaviour, it might get eliminated by the reduction (we call this *soft nondeterminism*, in opposition to *hard nondeterminism* that can not be eliminated by aggregation).

However, the aggregation does not come without cost. The main penalty is loss of information that could be obtained from the model. The reduction of the state space is based on aggregation of weakly congruent states. The steady-state probabilities are then computed for the aggregated states. In some cases this might be sufficient, but sometimes it is important to know how the probability mass of an aggregated state distributes on the states which form the aggregate: the probability mass can not simply be distributed equally over all states that form the aggregate.

Example 3.22

We illustrate this by means of a small Markov chain. Consider the CTMC in Figure 3.5 (a). States s_3 and s_4 are equivalent and can be aggregated, as shown in



(a) (b)

Figure 3.5: CTMC

Figure 3.5 (b). However, we can see that the throughput of state s_4, $\tau(s_4)$, is smaller than the throughput of state s_3, $\tau(s_3)$. To be more specific, $\tau(s_3) = 2{\cdot}\tau(s_4)$. The reason is that the branches outgoing from s_2 have different rates. So $\tau(s_3) = \frac{2}{3}\tau(s_2)$ and $\tau(s_4) = \frac{1}{3}\tau(s_2)$. Since the steady-state probability of a CTMC state can be expressed by its throughput and the sum of its outgoing rates, the steady-state probability of s_3 is equal to $\tau(s_3)$ and that of s_4 equal to $\tau(s_4)$.

State $s_{3,4}$ of the aggregated CTMC has throughput $\tau(s_{3,4}) = \tau(s_3) + \tau(s_4)$ and consequently the steady-state probability would be equal to $\tau(s_{3,4})$. Nothing tells us how this probability should be distributed again on the states s_3 and s_4, since we "throw away" the information of the original CTMC (*cf.* Figure 3.5 (a)).

Hence, in case that probabilities for individual states are needed, and not only for aggregates, state space reduction is not recommended. The problem is even harder, if nondeterminism comes into play: if the original model shows soft nondeterministic behaviour which has been eliminated in the reduced model, it is not possible to distribute the probability mass over the original states at all[4].

[4]Things might be different, if the pure IMC specification is superimposed by a reward structure [80]: a reward structure is a function on the state space which assigns a *reward rate* to a state. There are different approaches known to define reward structures for SPA models [34, 10, 36]. Whichever approach is chosen, an aggregation algorithm should only aggregate these states that are weakly bisimilar *and which have the same reward rate assigned.* Then, two weakly bisimilar states are either distinguished by the reward structure, or not. The point is that a reward structure brings the choice between distinguishing and aggregating weakly bisimilar states back under the control of the system modeller.

Apart from this potential problem, for our purposes it is necessary to retain the structure of a $\mathcal{YAWN}$ transition system as far as possible, since we will need this information in Chapter 4. This inhibits the use of state aggregation and elimination of nondeterminism. Though the former requirement is not really a problem, the latter forces us to make a design decision: *we can not allow a $\mathcal{YAWN}$ model to show nondeterministic behaviour, neither soft nor hard.*

3.2.2 CTMC Derivation and Evaluation for GMP

The CTMC generation from a given GMP and its evaluation is done in five steps. Let $G^u = (S^u, T^u, \mathcal{R}^u, s^u)$ be an s-deterministic (cf Section 3.1.5), non-interactive GMP with $\#S = n$ (the meaning of the superscript ^{u} is described below). The steps are:

1. Reduction of G^u to an irreducible GMP G;
2. Generation of a CTMC described by a generator matrix $\mathbf{Q}$;
3. Computation of the steady-state probabilities $\underline{\pi}$ of $\mathbf{Q}$;
4. Interpretation of $\underline{\pi}$ in G.
5. Computation of throughputs $\underline{\tau}$ of G.

We will comment on these steps in detail below.

Step 1: Reduction

Since G^u is non-interactive, T only contains **i**- and **t**-transitions. **i**-transitions denote timeless state changes of the GMP and have priority over **t**-transitions, which is a direct consequence of the definition of weak Markovian congruence (*cf.* Definition 3.13).

Example 3.23

Consider the process $R \stackrel{\text{def}}{=} \mathbf{i}.P + [\lambda].Q$ for $P, Q \in \mathcal{L}_{\mathcal{YAWN}}$. It is easy to check that $R \simeq \mathbf{i}.P$. In other words, all states of $Reach(Q)$ are reachable from R, by the definition of reachability, but the probability that this will happen is 0 (assuming $Reach(P) \cap Reach(Q) = \emptyset$). The $[\lambda]$-branch of the choice will never be chosen in the execution of R.

Hence, a GMP may contain states which are *effectively* not reachable from the starting state, since the probability that this happens is 0. In such a case the GMP would not be irreducible. Before the actual CTMC generation can take place, the GMP has to be freed from these unreachable states. This is what in the following we will call *reduction*. The simplest way to do so is to recompute the state space of G^u and eliminate all states that are effectively not reachable.

Definition 3.24 The *reduced* GMP $G = reduce(G^u) = (S, A, T, \mathcal{R}, s)$ is defined as follows: $T = T^u \cap (S \times Act_{\mathbf{t}} \times S)$ and $\mathcal{R} = \mathcal{R}^u|_T$. S is inductively defined as follows:

1. $s^u \in S$ and $s = s^u$.
2. If $p \in S$ and p is stable then, if $t \in T^u$ such that $src(t) = p$ then $dst(t) \in S$.
3. If $p \in S$ is not stable than for all[5] $t \in T^u$ with $src(t) = p$ and $lbl(t) = \mathbf{i}$: $dst(t) \in S$.
4. S is the smallest set containing all states that can be derived by the above rules.

Note that even after a GMP has been reduced by the described procedure, it it is not necessarily true that the resulting GMP is irreducible. Reduction as defined here only ensures that all states of the considered GMP are reachable from the starting state, but not vice versa.

Step 2: CTMC Generation

Let $G = (S, A, T, \mathcal{R}, s)$ be an s-deterministic, non-interactive, reduced, and irreducible GMP with $\#S = n$. To derive a generator matrix from G, we have to remove its untimed transitions. To define the generator matrix, we adopt a technique known from the area of Generalised Stochastic Petri Nets (GSPN) [103, 102]. We number the states of G from 1 to n such that $I = \{1, 2, \ldots, m\}$ (for $m \leq n$) numbers all states which are *not* stable and $J = \{m+1, \ldots, n\}$ those that are. For the rest of this section, we identify the states with their number. In the first step, we define a matrix $\mathbf{Q}' = (q'_{ij})_{n,n}$ of the following form:

$$\mathbf{Q}' = \begin{pmatrix} \mathbf{C} & \mathbf{D} \\ \mathbf{E} & \mathbf{F} \end{pmatrix},$$

where $\mathbf{C} \in \mathbb{R}^{m \times m}$ describes transitions from immediate (unstable) states to immediate states, $\mathbf{D} \in \mathbb{R}^{m \times (n-m)}$ transitions from immediate to stable states, $\mathbf{E} \in \mathbb{R}^{(n-m) \times m}$ those from stable to immediate states and finally, $\mathbf{F} \in \mathbb{R}^{(n-m) \times (n-m)}$ transitions from stable to stable states. More formally, $\mathbf{Q}'$ is defined for all $t \in T$ such that, if $src(t) = i$, $dst(t) = j$, then:

1. if $i, j \in I$ with $i \neq j$: $q'_{ij} := 1$ (sub-matrix $\mathbf{C}$);

[5] Since we assume s-determinism, this can be at most one transition.

2. if $i \in I, j \in J$: $q'_{ij} := 1$ (sub-matrix $\mathbf{D}$);
3. if $i \in J, j \in I$: $q'_{ij} := \mathcal{R}(t)$ (sub-matrix $\mathbf{E}$);
4. if $i, j \in J$, $i \neq j$: $q'_{ij} := \mathcal{R}(t)$ (sub-matrix $\mathbf{F}$).

The condition $i \neq j$ is important since self-loops do not play a role in CTMCs. Instead, we define for all stable states, *i.e.,* for $i = m+1, \ldots, n$:

$$q_{ii} = -\sum_{\substack{j=1 \\ j \neq i}}^{n} q_{ij}.$$

All other entries of $\mathbf{Q}'$ are defined to be 0. Please note that the definition of the sub-matrices $\mathbf{C}$ and $\mathbf{D}$ is based on the assumption that G is s-deterministic. Therefore, the ith row of $\mathbf{Q}'$ for $i \in I$ contains at most one 1.

We define the generator matrix $\mathbf{Q}$ of the CTMC underlying G as

$$\mathbf{Q} = \mathbf{F} + \mathbf{E}(\mathbf{I} - \mathbf{C})^{-1}\mathbf{D} \tag{3.1}$$

The proof that $\mathbf{Q}$ indeed describes the same Markov process as $\mathbf{Q}'$ can be found in [102]. The sub-matrix $\mathbf{F}$ describes the transitions between stable states. The sub-matrix $\mathbf{E}$ describes the transitions from a stable to an immediate state. Due to the irreducibility property of the GMP G, the probability to eventually enter a stable state from an immediate state is 1. However, before a stable state is reached, an arbitrary (although finite) number of immediate states can be visited. The probabilities to reach from one immediate state i the stable state j in an arbitrary number of steps is described by the (i, j)th entry in the matrix $(\mathbf{I} - \mathbf{C})^{-1}\mathbf{D}$, which occurs in (3.1). That the number of steps can be arbitrary large is reflected by the matrix $(\mathbf{I} - \mathbf{C})^{-1}$, which is a closed-form expression for the geometric sum of $\mathbf{C}$, $\sum_{\nu=0}^{\infty} \mathbf{C}^{\nu}$.

An entry at position (k, l) of matrix $\mathbf{E}(\mathbf{I} - \mathbf{C})^{-1}\mathbf{D}$ then denotes the rate with which a transition from stable state k to stable state l is made, *with a detour over the set of immediate states*. This matrix has to be added to $\mathbf{F}$ to completely specify the behaviour of G without explicit reference to the set of immediate states.

$\mathbf{Q}$ obviously has the same dimensions as $\mathbf{F}$ and its rows correspond one-to-one to the stable states of G.

Step 3: Steady-State Probabilities

The steady-state distribution $\underline{\pi}$ of a CTMC with generator matrix $\mathbf{Q}$ is defined as the solution of the system of linear equations

$$\begin{aligned} \underline{\pi}\mathbf{Q} &= 0 \\ \underline{\pi}\,\underline{1} &= 1. \end{aligned}$$

If $\mathbf{Q}$ is derived from a GMP G by Step 2, then $\underline{\pi}$ describes the steady-state probabilities for all stable states of G. Since they already sum up to 1, the immediate states consequently carry no probability mass. The steady-state probability vector of the GMP is then denoted as

$$\underline{\pi}^G = (\underbrace{0, \ldots, 0}_{m \text{ times}}, \pi_1, \ldots, \pi_n).$$

The leading null vector of $\underline{\pi}^G$ denotes the probabilities of the immediate states of G.

Step 4: Interpretation of $\underline{\pi}$ in G

In the following, we will no longer refer to the enumeration of the states of the GMP, but write $\pi^G(s)$ to denote the steady-state probability of state s of G.

Step 5: Throughputs of GMP

The throughputs of the stable states and the respective transitions of GMP G are the throughputs that can be computed by means of the techniques described in Appendix B.3.2. If $\mathbf{\Delta} = \text{diag}(q_{m+1,m+1}, \ldots, q_{nn})^{-1}$, then the throughputs for the stable states are defined as

$$\underline{\tau} = -\underline{\pi}\mathbf{\Delta}^{-1}.$$

We can also define throughputs for immediate states, and as we will see later, we need them. They can be derived by means of the matrices $\mathbf{C}$ and $\mathbf{E}$. The vector of throughputs for immediate states is denoted as $\underline{\tau}_i$ and can be derived as

$$\begin{aligned} \underline{\tau}_i &= -\underline{\tau}\mathbf{\Delta}\mathbf{E}(\mathbf{I} - \mathbf{C})^{-1} \\ &= -(-\underline{\pi}\mathbf{\Delta}^{-1})\mathbf{\Delta}\mathbf{E}(\mathbf{I} - \mathbf{C})^{-1} \\ &= \underline{\pi}\mathbf{E}(\mathbf{I} - \mathbf{C})^{-1}. \end{aligned}$$

$\underline{\pi}\mathbf{E}$ describes the vector of throughputs from stable states to immediate states. $(\mathbf{I} - \mathbf{C})^{-1}$ is the matrix that describes the probabilities that, starting in state $i \in J$, state $j \in I$ is reached in an arbitrary number of steps. These probabilities describe how the incoming throughput $\underline{\pi}\mathbf{E}$ distributes on the states $i \in I$. We define the vector of throughputs of the states of G as $\underline{\tau}^G = \underline{\tau}_i \circ \underline{\tau}$, where "$\circ$" denotes simple vector concatenation.

As for the probabilities, later we will not refer to the enumeration of the states of the GMP, but write $\tau^G(s)$ to denote the throughput of state s.

We can also express the throughput of individual transitions. To do so, we consider a matrix $\mathbf{P}'$ that describes the branching probabilities of all states of the considered CTMC: $\mathbf{P}'$ is the probability matrix of the CTMCs embedded Markov chain. $\mathbf{P}'$ can be written as

$$\mathbf{P}' = \left(\begin{array}{c|c} \mathbf{C} & \mathbf{D} \\ \hline \mathbf{\Delta}\mathbf{E} & \mathbf{I} - \mathbf{\Delta}\mathbf{F} \end{array}\right).$$

The branching probabilities distribute the incoming throughput of a state s over the outgoing transitions of s. Then

$$\mathbf{T}^G = (\tau_{ij})_{n,n} = \underline{\tau}^G \mathbf{P}'$$

is the matrix of all transition throughputs of GMP G. If $t \in T$ and $src(t) = i$ and $dst(t) = j$, then we define $\tau^G(t) = \tau_{ij}$. Please note that $\tau^G(\cdot)$ is overloaded, *i.e.,* defined for states *and* transitions.

3.3 Conclusions

In this chapter, we have defined the stochastic Markovian process algebra $\mathcal{YAWN}$. We have first defined the transition systems, namely GMTS, that we use to give meaning to $\mathcal{YAWN}$ processes. Then we defined the syntax of $\mathcal{YAWN}$ and the operational semantics. The stochastic information is given in term of exponentially distributed delays, specified by rates assigned to transitions. We have then given two notions of congruence, *strong Markovian bisimulation equivalence* and *weak Markovian congruence*. Finally, we have demonstrated how CTMCs can be derived from GMTS.

Since IMC is the nearest relative of $\mathcal{YAWN}$ in the family of SPA, we will briefly compare both calculi. Noteworthy differences between IMC and $\mathcal{YAWN}$ are only of technical, but not of conceptual nature. Therefore, the properties of IMC should also hold for $\mathcal{YAWN}$.

The main difference is that $\mathcal{YAWN}$ uses a completely different kind of transition system to describe the semantics of processes. Rather then using heterogeneous labelled (multi-) transitions systems to describe functional and temporal behaviour, for $\mathcal{YAWN}$ we use ordinary labelled transition systems, which are equipped with rates by means of a simple function.

We will compare IMC and $\mathcal{YAWN}$ more thoroughly in the following. The major differences are also summarised in Table 3.2.

Expressiveness

One of the most important characteristics of a process algebra is its expressiveness. Although IMC and $\mathcal{YAWN}$ are of the same breed, there are minor differences regarding their expressiveness. The most important difference is that we have not only one, but many internal actions. This allows that even for $\mathcal{YAWN}$ processes in which all actions are hidden away, it is still possible to see from which visible action a hidden action originally came from. The hiding operator in $\mathcal{YAWN}$ is not transparent. This allows a more thorough analysis of the inner structure of a $\mathcal{YAWN}$ process, however, the observable behaviour of a $\mathcal{YAWN}$ process is not distinguishable from an equivalent IMC process.

We can conclude that from a $\mathcal{YAWN}$ model it is still possible to obtain information about the internal structure, where this information might be lost in IMC.

Feature	IMC	$\mathcal{YAWN}$
Syntax	Differs in minor details	
State space	Set of terms	
Actions	• Synchronising actions $a \in Com$ • Timed actions $\lambda \in \mathbb{R}$ • Internal action τ	• Synchronising actions $a \in Com$ • Special action $\mathbf{t}$ • Internal actions $\mathbf{i}_a$ for $a \in Com \cup \{\mathbf{t}\}$
Transition system	Heterogeneous (multi-)sets of transitions (*Interactive Markov Chains*)	Homogeneous set of labelled transitions (*GMTS*)
Timing information	Special labelled multi-transition relation	Function on transition set
Treatment of identical timed transitions	Explicitly represented in multi-relation	Amalgamated, with proper addition of rates
CTMC generation	Componentwise minimisation according to weak congruence. *Soft* nondeterminism allowed	Elimination of immediate transitions. Nondeterminism not allowed.

Table 3.2: Differences between $\mathcal{YAWN}$ and IMC

CTMC Generation

In Section 3.2 we have presented a way to derive CTMCs from $\mathcal{YAWN}$ s-deterministic processes that is not based on aggregation, as for IMC. We chose a method known from the area of Generalised Stochastic Petri Nets and adapted it for $\mathcal{YAWN}$. The reason to go without the advantages of aggregation is that we have to retain the structure of the global state space as much as possible. We will see this in the next chapter.

Part II

Compositional Performance Evaluation

Chapter 4

Properties of $\mathcal{YAWN}$ Processes: Local Measures and Waiting Times

In Section 2.3, several approaches have been sketched to derive performance measures for SPA models without having the problem of state space explosion. We found that product-form solutions are the most advantageous, since they allow to compute results on the component level and combine them afterwards to yield *global* performance measures. Basically, this means that a probability distribution on the *global* state space can be computed, on which most other desired performance measures depend. Global probabilities are obtained by simple multiplication, as the name suggests (a normalisation might be necessary, however).

Product-form solution techniques require, however, that the structure of the global state space is known. Otherwise there would be no way to know which local measures should be combined, and how. Stated differently, if a process is considered for which the structure of the state space is not known without generating it explicitly, such a process will not have a product-form structure. As a consequence, if we want to do a component-based analysis of a process to avoid the state space explosion, all we possibly can derive are *local* measures, *i.e.,* measures that only inform about individual components, but not about the system as a whole.

In this chapter, we will define local measures, more specifically, local steady-state probabilities, in terms of global measures. This provides us a theoretical framework to describe exhaustively the stochastic dependencies that can be found between synchronising components. As a result, we can establish criteria how local measures should be related to each other, whichever method is chosen to derive them. These criteria can therefore be used as a touchstone for local measures that have been derived by whichever solution technique. However, no computational methods are presented to actually *derive* measures.

Outline of this Chapter. In this chapter we will formally introduce some of the concepts that have already been used (informally) in the previous chapter. In particular, in

Section 4.1, we define what we mean by a *component*. After that, in Section 4.2, we introduce the concept of a *local measure* for a component in its most general way, and set it into relation to what we shall name a *global measure*. In Section 4.3, we show that the mean number of synchronisations of interacting components plays a very important role to describe the stochastic dependencies between components. In Section 4.4 we introduce the concept of waiting times. The chapter is concluded by a short summary in Section 4.5.

4.1 Processes and Components

4.1.1 Sub-Processes, Locations, and Components

In this section, we define the term *component*. Intuitively, we shall consider a component as a sub-entity that is part of a system and that communicates with other sub-entities of the same system. It is a common property of all process calculi that the system descriptions are structured from the beginning. This structure gives many hints to decide which part of the process belongs to one or another component. However, the structure of the process is a feature of the description methodology used for PA: processes are described by syntactic objects of a language, and the structure of a process is pre-determined by the structure of its syntactic description. It is straightforward to decompose such a process *description*: informally, we want to consider everything as part of one component that is syntactically enclosed in parallel operators, like $\|_S$. But this approach does not reach far enough. A decomposition of a process description yields a number of syntactic objects, which in the following we will refer to as *sub-processes*. We can assume that sub-processes are related to what we want to call "components". However, since they are proper process terms, they describe a behaviour; but this is generally not the behaviour that we want to be described. A sub-process has no notion that it is part of a whole. What is described by a sub-process is the *potential* behaviour of a component, but not the *actual* behaviour which, in fact, also depends on the influence of other components, *e.g.,* via synchronisations.

It is therefore obvious that a syntactic decomposition of a $\mathcal{YAWN}$ system description (*i.e.,* a process) is not sufficient to derive components. The decomposition must take place on the semantical level.

What we therefore do in this section is to identify the components of a system properly and to describe their actual behaviour. First, we introduce a number of concepts:

1. A *sub-process* is meant to be a syntactic entity derived by the *syntactic decomposition* of a process[1]. We will only consider processes $P \in \mathcal{L}_{\mathcal{CC}}$, as defined in Definition 3.20. Intuitively, sub-processes of a $\mathcal{YAWN}$ process are more simple $\mathcal{YAWN}$ processes which are combined by means of the parallel operators $\|_S$.

[1] As described Section 3.1.3, processes are defined to be syntactic entities.

2. A *location* is meant to be an abstract identifier of a component and its behaviour. For each location of a process there is also an unique sub-process. One can think of a location as the place where a component "lives". We will use the terms "component" and "location" interchangeably. We exploit the fact that the processes that we consider have a static structure with respect to the parallel operator. Sub-processes are enclosed within parallel operators (except the leftmost and the rightmost) and do not change their position relative to these delimiters. We can therefore identify components by their position within the term. We say that the ith position within a process P is *location* i. Then, the syntactic entity at location i of process P is said to be the ith sub-process of P.

3. The actual behaviour of a component is described by a GMP: the *local* GMP. For each location i there is a local GMP G_i. If P_i is a sub-process that belongs to location i, then G_i is a sub-GMP of $[\![P_i]\!]$, the GMP defined by the sub-process P_i. "Sub-GMP" means that state-space and transition set of G are a subset of the state-space and transition set of $[\![P]\!]$, respectively. We will see in the following how sub-GMPs are obtained.

Sub-processes are derived from processes by functions which we will refer to as *projections*. Before we can define them formally, we need to define the number of locations of a process:

Definition 4.1 ($\#loc$) Let $P \in \mathcal{L}_{\mathcal{CC}}$. Then $\#loc(P)$, the number of locations of P, is defined as

$$\#loc(P) = \begin{cases} \#loc(Q) + \#loc(R) & \text{if } P \equiv Q\|_S R \\ \#loc(Q) & \text{if } P \equiv Q \setminus H \\ 1 & \text{otherwise.} \end{cases}$$

Example 4.2

We consider the $\mathcal{YAWN}$ process

$$P \stackrel{\text{def}}{=} \underline{recX : a.b.X\|_{\{b\}} \left((recY : b.c.Y\|_{\{c\}} recZ : c.d.Z) \setminus \{c\}\right)}.^{2}$$

Then, P has 3 locations:

$$\begin{aligned} \#loc(P) &= \#loc(\underline{recX : a.b.X\|_{\{b\}}(recY : b.c.Y\|_{\{c\}} recZ : c.d.Z) \setminus \{c\}}) \\ &= \#loc(\underline{recX : a.b.X}) + \#loc(\underline{recY : b.c.Y\|_{\{c\}} recZ : c.d.Z}) \\ &= 1 + \#loc(\underline{recY : b.c.Y}) + \#loc(\underline{recZ : c.d.Z}) \\ &= 1 + 1 + 1 = 3. \end{aligned}$$

[2] In the following, we underline process terms for better readability.

4.1.2 Projection on Local States

We now introduce a function $\Downarrow_i^P$ that maps a process $P \in \mathcal{L}_{\mathcal{CC}}$ on the sub-process at location i, $1 \leq i \leq \#loc(P)$; $\Downarrow_i^P$ is said to be the ith projection of P and is defined as:

Definition 4.3 (Projection) Let $P \in \mathcal{L}_{\mathcal{CC}}$. Then, for $1 \leq i \leq \#loc(P)$:

$$\Downarrow_i^P : \begin{cases} Reach(P) & \longrightarrow \quad \mathcal{L}_{\mathcal{YAWN}} \\ P' & \longmapsto \quad \begin{cases} \Downarrow_i^Q(Q) & \text{if } P' \equiv Q\|_S R \text{ and } i \leq \#loc(Q) \\ \Downarrow_{i-\#loc(Q)}^R(R) & \text{if } P' \equiv Q\|_S R \text{ and } i > \#loc(Q) \\ \Downarrow_i^Q(Q) & \text{if } P' \equiv Q \setminus H \\ P' & \text{otherwise.} \end{cases} \end{cases}$$

If $P_i \stackrel{\text{def}}{=} \Downarrow_i^P(P)$, then certainly $\Downarrow_i^P(Reach(P)) \subseteq Reach(P_i)$. We say that $\Downarrow_i^P(Reach(P))$ is the *local state space* of component i of P.

Example 4.4

We continue Example 4.2 and derive now the sub-process at location 2 of P (we leave out the superscript ^{P} for better readability):

$$\begin{aligned} \Downarrow_2(P) &= \Downarrow_2\Big(\underline{recX : a.b.X\|_{\{b\}} \left((recY : b.c.Y\|_{\{c\}} recZ : c.d.Z) \setminus \{c\}\right)}\Big) \\ &= \Downarrow_{2-1}\Big(\underline{(recY : b.c.Y\|_{\{c\}} recZ : c.d.Z) \setminus \{c\}}\Big) \\ &= \Downarrow_1\Big(\underline{recY : b.c.Y\|_{\{c\}} recZ : c.d.Z}\Big) \\ &= \Downarrow_1(\underline{recY : b.c.Y}) \\ &= recY : b.c.Y \end{aligned}$$

We see that the fact that the action c was hidden is not taken into account by the projection $\Downarrow$.

We now want to define the set of *partial decompositions* of a term $P \in \mathcal{L}_{\mathcal{CC}}$, denoted as $pDecomp(P)$.

Definition 4.5 (Partial decompositions) Let $P \in \mathcal{L}_{\mathcal{CC}}$. The set of *partial decompositions* of P is defined as

$$pDecomp(P) = \begin{cases} pDecomp(P_1) \cup pDecomp(P_2) \cup \{P\} & \text{if } P \equiv P_1\|_S P_2 \\ pDecomp(P') \cup \{P\} & \text{if } P \equiv P' \setminus H \\ P & \text{otherwise.} \end{cases}$$

Sub-processes are partial decompositions of P, *i.e.*, $\Downarrow_i^P(P) \in pDecomp(P)$.

Example 4.6

Let $P \stackrel{\text{def}}{=} A\|_c((B\|_b C) \setminus \{b\})$, where A, B, C are process constants. Then the partial decompositions of P are

$$\begin{aligned} pDecomp(P) &= pDecomp(A) \cup pDecomp((B\|_b C) \setminus \{b\}) \cup \{P\} \\ &= \{A, P\} \cup pDecomp(B\|_b C) \cup \{(B\|_b C) \setminus \{b\}\} \\ &= \{A, P, (B\|_b C) \setminus \{b\}\} \cup pDecomp(B) \cup pDecomp(C) \cup \{B\|_b C\} \\ &= \{A, P, (B\|_b C) \setminus \{b\}, B\|_b C\} \cup pDecomp(B) \cup pDecomp(C) \\ &= \{A, P, (B\|_b C) \setminus \{b\}, B\|_b C, B, C\} \end{aligned}$$

4.1.3 Projection on Local Transitions

We are not only interested in a mapping from the global onto the local state space, but also in a mapping from the set of the global transitions to that of the respective sub-processes. We define a function $\Downarrow_i^P$ that maps the transitions of $[\![P]\!]$ on the transitions of $\left[\!\left[\Downarrow_i^P(P)\right]\!\right]$. Such a function can only be partially defined, since usually not all components will participate in one global transition. Timed transitions are always executed by single components *only*. So, when we consider a timed global transition, for example, t of $[\![P]\!]$, then there is only one location i such that $\Downarrow_i^P(t)$ is defined. For $j \neq i$, $\Downarrow_j^P(t)$ is undefined, which we denote as $\Downarrow_j^P(t) = \uparrow$.

In process calculi, the relation between global and local transitions usually can not be expressed by functions. In $\mathcal{YAWN}$this is possible. Before we explain why this works for $\mathcal{YAWN}$, we describe the problem.

In other processes calculi, some operators do merge two local transitions to one global, so that it is not possible to express by a function which components caused this global transition. The operator which is the most likely cause of this this "trouble" is the hiding operator. We illustrate this in the following example.

Example 4.7

We consider the process $P \stackrel{\text{def}}{=} a.\mathsf{stop} + b.\mathsf{stop}$, which we interpret according to the SOS given in Table 2.1. Then P has two outgoing transitions, namely $t_1 = P \xrightarrow{a} \mathsf{stop}$ and $t_2 = P \xrightarrow{b} \mathsf{stop}$. The process $P \setminus \{a, b\}$ has only one outgoing transition, $t_3 = P \setminus \{a, b\} \xrightarrow{\mathbf{i}} \mathsf{stop}$. Although $P \setminus \{a, b\}$ is a trivial process comprising only one component, it is not possible to say to which transition t_1 or t_2 the transition t_3 corresponds. This information is lost.

The problem in Example 4.7 cannot occur with $\mathcal{YAWN}$. In Section 3.1.1, we have defined a whole set of internal actions. The hiding operator of $\mathcal{YAWN}$ is defined such that a relabelled transition still retains the information which action is hidden. Each hidden transition still carries the information from which transition it was originally derived.

Example 4.8

We reconsider the process $P \stackrel{\text{def}}{=} a.\mathsf{stop} + b.\mathsf{stop}$ of Example 4.7, now interpreted according to the $\mathcal{YAWN}$ SOS in Table 3.1. Again, P has two outgoing transitions $t_1 = P \xrightarrow{a} \mathsf{stop}$ and $t_2 = P \xrightarrow{b} \mathsf{stop}$— as does the process $P \setminus \{a, b\}$. The $\mathcal{YAWN}$ semantics ensures that $[\![P]\!]$ contains the two transitions $P \setminus \{a, b\} \xrightarrow{\mathbf{i}_a} \mathsf{stop}$ and $P \setminus \{a, b\} \xrightarrow{\mathbf{i}_b} \mathsf{stop}$.

To define the function $\Downarrow_i^P$ for $P \in \mathcal{L}_{\mathcal{CC}}$ we again rely on the fact that for each transition t of $[\![P]\!]$ there is a set of derivation trees DT_t which proves that t is a legal transition. If a location i contributes to t with a local transition t', then a derivation subtree D' of a derivation tree $D \in DT_t$ will prove that there is indeed a transition t' of component i that participates in t. Our definition of a projection from global on local transitions is based on the idea to find a derivation sub-tree of the global transition which proves that the component contributes with a local transition. If there is such a derivation subtree, we define $\Downarrow_i^P(t) = t'$. If no derivation subtree for the location does exist, we define $\Downarrow_i^P(t) = \uparrow$.[3] We introduce this concept now more formally.

Definition 4.9 (Projections on Local Transitions) Let $P \in \mathcal{L}_{\mathcal{CC}}$, $[\![P]\!] = (S, A, T, \mathcal{R}, P)$, and $t \in T$. Let $P' \stackrel{\text{def}}{=} \Downarrow_i^P(src(t))$ for a location i of P and $[\![P']\!] = (S', A', T', \mathcal{R}', P')$. Let $\mathcal{T} = \{t' \in T' \mid src(t') = P'\}$ the set of all transitions of $[\![P']\!]$ with source P'. Let $DT_{loc}(P')$ contain all derivation trees for all transitions that have source state P'.

Define the function $\Downarrow_i^P : T \longrightarrow T' \cup \{\uparrow\}$ such that $\Downarrow_i^P(t) = t'$ if there is a $D' \in DT_{loc}(P')$ and a $D \in DT_t$ such that D' is a derivation sub-tree of D and proves a transition of location i. If no such D and D' exist then $\Downarrow_i^P(t) = \uparrow$.

Example 4.10

We consider the processes

$$\begin{aligned} P_1 &\stackrel{\text{def}}{=} a.c.P_1 \\ P_2 &\stackrel{\text{def}}{=} a.b.P_2 + b.a.P_2 \\ P_3 &\stackrel{\text{def}}{=} c.b.P_3 \end{aligned}$$

[3] A similar approach to this has been proposed by Priami [121]: there, the transitions are labelled with the proof trees that prove them. With this approach, all transitions can always be distinguished from each other, regardless whether the other labels differ or not.

and

$$R \stackrel{\text{def}}{=} P_1 \parallel_{\{a,c\}} (P_2 \parallel_b P_3).$$

P_i is the sub-process of location i for $i = 1, 2, 3$. Surely, $R \xrightarrow{a} c.P_1 \parallel_{\{a,c\}} (b.P_2 \parallel_b P_3)$ is a transition of $[\![R]\!]$ (which we name t for easier reference). The derivation tree D (there is only one) of t is

$$\dfrac{\dfrac{}{a.c.P_1 \xrightarrow{a} c.P_1} \qquad \dfrac{\dfrac{\dfrac{}{a.b.P_2 \xrightarrow{a} b.P_2}}{a.b.P_2 + b.a.P_2 \xrightarrow{a} b.P_2}}{P_2 \xrightarrow{a} b.P_2}}{P_2 \parallel_b P_3 \xrightarrow{a} b.P_2 \parallel_b P_3}}{R \xrightarrow{a} c.P_1 \parallel_{\{a,c\}} (b.P_2 \parallel_b P_3)}$$

What is $\Downarrow_1^R(t)$? There is a derivation sub-tree in D that proves that location i is involved in the execution of t: it is the left branch,

$$\dfrac{}{a.c.P_1 \xrightarrow{a} c.P_1},$$

hence, we can set $\Downarrow_1^R(t) = \underline{a.c.P_1 \xrightarrow{a} c.P_1}$. For location 2, we find that $\Downarrow_2^R(t) = \underline{a.b.P_2 \xrightarrow{a} b.P_2}$, since the right branch of D leads directly to a derivation of location 2. $\Downarrow_3^R(t)$ is then equal to $\uparrow$, since there are no derivation sub-trees left.

Even though the definition of the projection on transitions, as given in Definition 4.9, has been designed very carefully, there are still some open problems. In the next example we will see that there are situations where a (global) transition can be mapped on two transitions of different components, even though the both components might be completely unrelated. This is a situation that we would like to avoid, since it is contrary to the assumption that different, unsynchronising local transitions correspond to different global ones.

Example 4.11

We consider the process

$$R \stackrel{\text{def}}{=} recX : a.X \| recX : a.X.$$

The GMP $[\![R]\!]$ has only one state, R, and only one transition, the self-loop $R \xrightarrow{a} R$. However, there are two derivation trees for this process, even though they are not easy to distinguish: the first is

$$6) \dfrac{5a) \dfrac{a.recX : a.X \xrightarrow{a} recX : a.X}{recX : a.X \xrightarrow{a} recX : a.X}}{recX : a.X \| recX : a.X \xrightarrow{a} recX : a.X \| recX : a.X}$$

and the second is

$$7)\ \frac{5a)\ \dfrac{a.recX : a.X \xrightarrow{a} recX : a.X}{recX : a.X \xrightarrow{a} recX : a.X}}{recX : a.X \| recX : a.X \xrightarrow{a} recX : a.X \| recX : a.X}$$

The difference is that the first derivation tree uses SOS Rule 1), Rule 5a), and then Rule 6) (*cf.* Table 3.1), whereas the second derivation tree uses SOS Rule 1), Rule 5a), and then Rule 7). Consequently, the left location as well as the right one are described by the transition $R \xrightarrow{a} R$, even though they are completely independent.

We want to exclude such effects from our further considerations, and we show now that we can avoid these situations.

There are several possibilities why two transitions can be amalgamated, although the should not. First of all, this can only happen when self-loops are involved. Otherwise, the state change would already be sufficient to distinguish the transitions in question.

We describe now the different scenarios that can occur by means of examples:

- Two (or more) processes P_1 and P_2 have two *timed* self-loops which are amalgamated in $P_1 \|_S P_2$. In the Markovian case, timed self-loops are irrelevant for stochastic evaluation of a process, hence they can be eliminated from the start.

- Two (or more) processes P_1 and P_2 have two untimed self-loop transitions t_1 and t_2 with $lbl(t_1) = lbl(t_2) = a \in Com$. If we now consider the process $P \stackrel{\text{def}}{=} P_1 \|_S P_2$ with $a \in S$, we can assume without loss of generality that $[\![P]\!]$ has a self-loop transition t with label $lbl(t) = a$ such that $\Downarrow_1^P(t) = t_1$ and $\Downarrow_2^P(t) = t_2$. But this is correct, since both transition t_1 and t_2 actually participate in t. Hence, the situation of Example 4.11 does not belong to this case.

- Two (or more) processes P_1 and P_2 are combined to a process $R \stackrel{\text{def}}{=} P_1 \|_S P_2$ with $lbl(t_1) = lbl(t_2) \notin S$ such that t_1 and t_2 are amalgamated. This is the situation of Example 4.11, where $P_1 \stackrel{\text{def}}{=} P_2 \stackrel{\text{def}}{=} recX : a.X$ and $S \stackrel{\text{def}}{=} \emptyset$. Now, we consider the processes $P_1' = P_2' = recX : a.a.X$. It is easy to see that $P_1 \sim P_1'$ and $P_2 \sim P_2'$. Consequently, $P_1 \| P_2 \sim P_1' \| P_2'$. But the latter process is nondeterministic, since $P_1' \| P_2' \xrightarrow{a} a.recX : a.a.X \parallel P_2'$ and $P_1' \| P_2' \xrightarrow{a} P_1' \parallel a.recX : a.a.X$. Hence, we must understand the situation in Example 4.11 as a form of nondeterminism that is not so easy to detect, but is forbidden due to our requirement that we only consider s-deterministic processes.

Only the last item is hence relevant to us, and since it is detectable we can exclude it from our considerations.

Local GMP

By means of the projections we can now define the local GMPs of a process:

Definition 4.12 Let $P \in \mathcal{L}_{\mathcal{CC}}$ and $[\![P]\!] = (S, Act_{\mathbf{t}}, T, \mathcal{R}, P)$, i a location of P, $P_i = \Downarrow_i^P(P)$, and $[\![P_i]\!] = (S_i, Act_{\mathbf{t}}, T_i, \mathcal{R}_i, P_i)$. The local GMP of location i of P is then the GMP

$$G_i^P = (S_i^P, Act_{\mathbf{t}}, T_i^P, \mathcal{R}_i^P, \Downarrow_i^P(P))$$

where

- $S_i^P = \Downarrow_i^P(S)$;
- $T_i^P = \Downarrow_i^P(T) \setminus \{\uparrow\}$;
- $\mathcal{R}_i^P = \mathcal{R}_i\big|_{T_i^P}$.

In GMP $[\![P]\!]$ for $P \in \mathcal{L}_{\mathcal{YAWN}}$, there is always a tight relationship between the individual states and the complete transition system itself: states are processes, and a state s has exactly that transition encoded that is formed by the sub-GMP of $[\![P]\!]$ with state space $Reach(s)$. Local GMPs are different, because states do no longer have this property: in case that states of sub-process $\left[\!\left[\Downarrow_i^P(P)\right]\!\right]$ become unreachable, this is reflected in the local GMP G_i^P, but not in its states. Thus, the local GMP G_i^P describes the actual behaviour of a component more exactly than $\left[\!\left[\Downarrow_i^P(P)\right]\!\right]$. Nevertheless, the description is not accurate, as the next example shows.

Example 4.13

We consider the process

$$P \stackrel{\text{def}}{=} (recX : \mathsf{a.b}.X) \|_{\{\mathsf{a},\mathsf{b}\}} \mathsf{a.b.a.b.a.b.stop}$$

P has two locations. Obviously, P can only make a finite number of steps until it blocks forever. Nevertheless, the local GMP of location 1 can be described by the process $recX : \mathsf{a.b}.X$, which shows infinite behaviour. Hence, local GMP generally do not reflect reachability properties of components accurately.

For our purpose, local GMPs are however accurate enough.

4.1.4 Inverse Projections

Projections relate global states to local states and global transitions to local ones. We can also use both functions to reverse the direction and relate local states and transitions to global states and transitions, respectively. We will do so in Section 4.2 by means of the *inverse of the projections*. In this section, we define the notation we will use for them. We give the definitions again in two parts, first for states, then for transitions.

Definition 4.14 Let $P \in \mathcal{L}_{\mathcal{CC}}$, G_i^P as defined in Definition 4.12 for a location i of P. Then the inverse of the projection $\Downarrow_i^P$ is denoted as $\Uparrow_i^P$ and is defined as

$$\Uparrow_i^P(Q) : \left\{ \begin{array}{ccc} S_i^P & \longrightarrow & 2^{Reach(P)} \\ Q & \longmapsto & \left\{ P' \in Reach(P) \;\middle|\; \Downarrow_i^P(P') = Q \right\} \end{array} \right.$$

$\Uparrow_j^P$ obviously partitions $Reach(P)$.

The definition of inverses of the projections on the transition sets is similar:

Definition 4.15 Let $[\![P]\!] = (S, Act_{\mathbf{t}}, T, \mathcal{R}, P)$ and $G_i^P = (S_i^P, Act_{\mathbf{t}}, T_i^P, \mathcal{R}_i^P, \Downarrow_i^P(P))$ the ith local GMP as defined in Definition 4.12 for a $P \in \mathcal{L}_{\mathcal{CC}}$. Then the inverse of $\Downarrow_i^P$ is denoted as $\Uparrow_i^P$ and is defined as

$$\Uparrow_i^P : \left\{ \begin{array}{ccc} T_i^P & \longrightarrow & 2^T \\ t' & \longmapsto & \left\{ \; t \in T \;\middle|\; \Downarrow_j^P(t) = t' \; \right\}. \end{array} \right.$$

4.2 Local Measures

In this section, we will show the use of projections and inverse projections. We consider a process $P \in \mathcal{L}_{\mathcal{YAWN}}$, the sub-process $P_i \stackrel{\text{def}}{=} \Downarrow_i^P(P)$ and the local GMP G_i^P for a location i of P.

4.2.1 Local Probabilities

Let $P \in \mathcal{L}_{\mathcal{YAWN}}$. An inverse projection $\Uparrow_i^P$ for $i \in \{1, \ldots, \#loc(P)\}$ defines implicitly a partition and an equivalence relation on $Reach(P)$: $P', Q' \in Reach(P)$ are in one equivalence class, if and only if $\Downarrow_i^P(P') = \Downarrow_i^P(Q')$. We denote this equivalence as $\mathcal{E}_i^P$. Let $\pi : Reach(P) \longrightarrow [0, 1]$ be a probability measure on the state space of $[\![P]\!]$. Then we define for $C \in Reach(P)/\mathcal{E}_i^P$

$$\pi(C) = \sum_{s \in C} \pi(s). \tag{4.1}$$

Definition 4.16 Let $P \in \mathcal{L}_{\mathcal{CC}}$ such that a steady-state probability measure π on $Reach(P)$ does exist. If $\Downarrow_i^P(C) = \{Q\}$ for $C \in Reach(P)/\mathcal{E}_i^P$, then $\pi(C)$ is said to be the *local (steady-state) probability* of Q and is denoted as $\phi_i^P(Q)$.

If we define a numbering on the states of the local GMP G_i^P of P then we can write the local probabilities as vector

$$\underline{\phi_i^P} = \left(\phi_i^P(1), \phi_i^P(2), \ldots, \phi_i^P(m)\right),$$

where $m = \#S_i^P$ and $\phi_i^P(j) = \phi_i^P(R)$ if $R \in S_i^P$ has number j.

Lemma 4.17 $\underline{\phi_i^P}$ is a stochastic vector.

Proof: We only have to check that $\underline{\phi_i^P}$ is not sub-stochastic. The requirement that all elements have to be positive is trivially fulfilled.

By the definition of $\Uparrow_i^P$ (*cf.* Definition 4.14) it is easy to see that $\Uparrow_i^P(S_i^P) = S$. Hence, by means of (4.1), we accumulate the probability mass of the global state space on the local states. Since the global probabilities sum up to 1, so do the local. →•

4.2.2 Local Throughputs

Let $P \in \mathcal{L}_{\mathcal{CC}}$ and G_i^P defined as in Definition 4.12 and let us assume that the throughputs of the transitions of $[\![P]\!]$ are described by the function $\tau : S \longrightarrow \mathbb{R}$. We want to describe the throughputs of the local GMP G_i^P in dependence of τ. We do so by means of the projection $\Downarrow_i^P$ and its inverse. For $t \in T_i$, the set $\Uparrow_i^P(t)$ contains all transitions of $[\![P]\!]$ which affect a state change of sub-process i. The sum of the throughputs of these transitions, *i.e.*, the mean number of instances in which these transitions are "used", is consequently the throughput of t.

Definition 4.18 If $C = \Uparrow_i^P(t)$, we define for $t \in T_i$ the *local throughput* $\sigma_i^P(t)$ of t as

$$\sigma_i^P(t) = \sum_{t' \in C} \tau^G(t').$$

By overloading of σ_i^P we define the throughput of a state $s \in S_i$ to be

$$\sigma_i^P(s) = \sum_{\substack{t \in T_i \\ src(t)=s}} \sigma_i^P(t)$$

4.2.3 Action Throughputs

In the following we will also be interested in the overall throughput of a component in which a transition with label a is involved. In general, we name this quantity the *action throughput*, or, more specifically, the a-throughput. Again, we assume a process $P \in \mathcal{L}_{\mathcal{CC}}$, the local GMP G_i^P as defined in Definition 4.12. For $a \in Com$, the a-throughput $\theta_i^P(a)$ is defined as

$$\theta_i^P(a) = \sum_{\substack{t \in T_i \\ lbl(t)=a}} \sigma_i^P(t).$$

$\theta_i^P(a)$ is the mean number of a actions that location i of P performs in unit time. We enhance this definition on partial decompositions of P as follows.

Definition 4.19 Let $P' \in pDecomp(P)$ and $a \in Com$. Then the a-throughput $\Theta_a^P(P')$ of P' is defined as

$$\Theta_a^P(P') = \begin{cases} \Theta_a^P(P_1) + \Theta_a^P(P_2) & \text{if } P' \equiv P_1 \|_S P_2 \text{ and } a \notin S, \\ \Theta_a^P(P_1) & \text{if } P' \equiv P_1 \|_S P_2 \text{ and } a \in S, \\ \Theta_a^P(P_1) & \text{if } P' \equiv P_1 \setminus H \text{ and } a \notin H, \\ \theta_i^P(a) & \text{if } P' \equiv \Downarrow_i^P(P) \text{ for a location } i \text{ of } P \text{ and } a \in \boldsymbol{\alpha}(P'), \\ 0 & \text{otherwise.} \end{cases}$$

In the definition of Θ_a^P we have made an assumption that needs justification: in the case that $P' = P_1 \|_S P_2$ and $a \in S$, we have set $\Theta_a^P(P') = \Theta_a^P(P_1)$, *i.e.*, we have not at all taken $\Theta_a^P(P_2)$ into account. The reason for that is the following theorem.

Theorem 4.20 Let $P \in \mathcal{L}_{\mathcal{CC}}$, $P' = P_1 \|_S P_2 \in pDecomp(P)$, and $a \in S$. Then

$$\Theta_a^P(P_1) = \Theta_a^P(P_2) \tag{4.2}$$

PROOF: Due to the fact that a is an element of the synchronisation set S, both P_1 and P_2 can perform an a action only in synchrony. Hence, the mean number of instances an a-transitions is performed is equal for both P_1 and P_2. →•

Theorem 4.20 will become important in Section 4.3.

4.2.4 Branching Probabilities and Throughput Equations

For a process $P \in \mathcal{L}_{\mathcal{CC}}$, we assume that for $[\![P]\!] = (S, Act_{\mathbf{t}}, T, \mathcal{R}, P)$ the throughputs $\tau(t)$ for all transitions $t \in T$ and the respective throughputs $\tau(s)$ for $s \in S$ are determined. The branching probabilities of a state s of $[\![P]\!]$ are a probability distribution over the transition set $T_s = \{t \in T \mid src(t) = s\}$. It describes the probability $\beta(t)$ that transition t is "used"

to leave state $s = src(t)$, once it is entered. These probabilities can be described by the throughputs $\tau(t)$ for $t \in T_s$. We define for $t \in T_s$

$$\beta(t) = \frac{\tau(t)}{\sum_{t' \in T_s} \tau(t')} = \frac{\tau(t)}{\tau(s)}.$$

With the branching probabilities we can express the throughputs of the transition as

$$\tau(t) = \beta(t)\tau(s)$$

Since the throughput going into a state must be equal to the throughput leaving the state, we can formulate for each state $s \in S$ a balance equation as follows.

$$\sum_{\substack{t \in T \\ src(t)=s}} \tau(t) \quad = \quad \sum_{\substack{t' \in T \\ dst(t')=s}} \tau(t').$$

Rewritten with branching probabilities, we yield

$$\tau(s) \sum_{\substack{t \in T \\ src(t)=s}} \beta(t) \quad = \quad \sum_{\substack{t' \in T \\ dst(t')=s}} \beta(t')\tau(src(t')),$$

which is equivalent to

$$\tau(s) \quad = \quad \sum_{\substack{t' \in T \\ dst(t')=s}} \beta(t')\tau(src(t')), \tag{4.3}$$

since

$$\sum_{\substack{t \in T \\ src(t)=s}} \beta(t) = 1.$$

Matrix notation

If we assume a numbering of all states $s \in S$, we can bring the above equations in matrix form. We define $\tau_i = \tau(s)$, if state s corresponds to number i. Let $\beta_{ij} = \beta(t)$, if $src(t)$ and $dst(t)$ correspond to number i and j, respectively. If $\#S = N$, then the throughputs are described by a vector $\underline{\tau} = (\tau_1, \ldots, \tau_N)$ and the branching probabilities by a matrix $\mathbf{B} = (\beta_{ij})_{N,N}$. The system of linear equations, described by (4.3), can then be formulated as

$$\underline{\tau}\mathbf{B} = \underline{\tau}. \tag{4.4}$$

This system of linear equations is very similar to the system of global balance equations of an DTMC, and indeed: the matrix $\mathbf{B}$ is a probability matrix: the *embedded Markov chain*

of the Markov process described by the GMP $[\![P]\!]$. We call (4.4) the system of *throughput equations* of GMP $[\![P]\!]$.

As is well known, a system of linear equations like (4.4) has not full rank. In our case it is of rank $N-1$. For a DTMC, usually the condition that all probabilities sum up to one is employed to determine a unique solution. But since $\underline{\tau}$ is a vector of throughputs, this condition is of no use to yield a unique solution for (4.4). There is only the possibility to express all throughputs relatively to one arbitrarily chosen reference. If we assume a solution $\underline{\tau}$ for (4.4) and choose τ_1 as the reference throughput, we can define a vector $\underline{v}$ as

$$\begin{aligned} \underline{v} &= \frac{1}{\tau_1}\underline{\tau} \\ &= (1, v_2, \ldots, v_N) \end{aligned}$$

The vector $\underline{v} = (v_1, \ldots, v_N)$ (with $v_1 = 1$) is said to be the vector of *visit counts* and $\nu = \tau_1$ the *reference throughput*[4]. In subsequent sections we will no longer explicitly define which throughput is chosen to be the reference throughput; we will only refer to *the* reference throughput ν. Then for all throughputs τ_i can be expressed as

$$\tau_i = v_i \nu.$$

As usual, we write $v(s)$ for the visit count v_i, if $s \in S$ corresponds to number i and if we do not want to refer to an explicit numeration of state space S. For all transitions $t \in T_s$, the throughput of t can then be written as

$$\tau(t) = v(t)\nu$$

for $v(t) = \beta(t)v(s)$.

Now, we assume that $\pi : S \longrightarrow [0,1]$ is the steady-state probability distribution of $[\![P]\!]$. Let $G_i^P = (S_i, Act_{\mathbf{t}}, T_i, \mathcal{R}_i, s_i)$ for $i = 1, \ldots, \#loc(P)$ be the local GMPs for P. Definition 4.18 allows us to derive that for $t \in T_j$ and $C = \Uparrow_j^P(t)$:

$$\begin{aligned} \sigma_j^P(t) &= \sum_{t' \in C} \tau(t') \\ &= \nu \sum_{t' \in C} v(t') \end{aligned} \tag{4.5}$$

A consequence of (4.5) is that also *local* throughputs, regardless to which transition in whatever location they belong, are linearly dependent on the reference throughput ν, and therefore, also on each other.

[4]The terminology chosen for the throughput equations comes actually from queueing theory. For closed queueing networks, like Gordon-Newell queueing networks, a system of linear equations has to be solved that expresses the routing probabilities between the different queues: the so-called *traffic equations*.

4.3 Throughputs and Synchronisations

In Section 4.2.4 we have shown that the throughputs of locations can be expressed by visit counts and one global reference throughput ν. In this section we state this fact in a different way: we set the a-throughputs (for $a \in Com$) of the locations of a $P \in \mathcal{L}_{\mathcal{CC}}$ in relation to each other. This approach is based on Theorem 4.20.

4.3.1 Local Throughput Equations

We consider a process $P \in \mathcal{L}_{\mathcal{CC}}$ from which we assume that a steady-state measure $\pi : Reach(P) \longrightarrow [0,1]$ on $[\![P]\!]$ does exist. Let $P_i \stackrel{\text{def}}{=} \Downarrow_i^P(P)$ and let $G_i^P = (S_i, Act_{\mathbf{t}}, T_i, \mathcal{R}_i, P_i)$ be the local GMP of P for location $i = 1, \ldots, \#loc(P)$. For each of the G_i^P a system of throughput equations can be defined, as shown in Section 4.2.4. This system of linear equations defines visit counts $v_i(s)$ and $v_i(t)$ for each state $s \in S_i$, and each transition $t \in T_i$ of G_i^P, respectively. Each location i has also its own reference throughput, ν_i.

Location i can only interact with other locations by synchronisation over the set of visible actions, $\boldsymbol{\alpha}(G_i^P)$. Therefore, we are only interested in the a-throughputs $\theta_i^P(a)$ for $a \in \boldsymbol{\alpha}(G_i^P)$. The a-throughputs for $a \in \boldsymbol{\alpha}(G_i^P)$ and a location i are related to each other by the throughput equations defined for G_i^P. Consequently, all a-throughputs for $a \in \boldsymbol{\alpha}(G_i^P)$ for location i can be expressed by means of the reference throughput ν_i and the visit counts that are defined for G_i^P:

$$\theta_i^P(a) = \sum_{\substack{t \in T_i \\ lbl(t)=a}} v_i(t)\nu_i.$$

If we define $\theta_i^P(a)$ to be the new reference throughput, we can express the action throughputs $\theta_i^P(b)$ for $b \in \boldsymbol{\alpha}(P_i) \setminus \{a\}$ as

$$\theta_i^P(b) = \overline{ab}_i \theta_i^P(a) \tag{4.6}$$

where $\overline{ab}_i$ is a scaling parameter derived from the visit counts of i, and which is defined as

$$\overline{ab}_i = \sum_{\substack{t \in T_i \\ lbl(t)=b}} v_i(t) \Big/ \sum_{\substack{t \in T_i \\ lbl(t)=a}} v_i(t).$$

Note that the following property holds:

$$\overline{ab}_i = \frac{1}{\overline{ba}_i}.$$

If $\#(\boldsymbol{\alpha}(P_i)) = k_i$, then (4.6) defines $k_i - 1$ independent equations for the local throughput in location i. Hence, if we have $\#loc(P) = n$ locations, then there are $\sum_{i=1}^{n} k_i - n$ linearly independent equations in total.

Example 4.21

We consider the (sub-)processes $A, B, C, D, E \in \mathcal{L}_{\mathcal{YAWN}}$, which are defined as follows:

$$\begin{aligned} A &\stackrel{\text{def}}{=} a.[\alpha_1].b.[\alpha_2].b.[\alpha_3].A \\ B &\stackrel{\text{def}}{=} b.[\beta_1].a.[\beta_2].b.[\beta_3].B \\ C &\stackrel{\text{def}}{=} a.[\gamma_1].c.[\gamma_2].c.[\gamma_3].C \\ D &\stackrel{\text{def}}{=} c.[\delta_1].a.[\delta_2].d.[\delta_3].d.[\delta_4].c.[\delta_5].D \\ E &\stackrel{\text{def}}{=} d.[\varepsilon_1].a.[\varepsilon_2].d.[\varepsilon_3].E. \end{aligned}$$

Then

$$R \stackrel{\text{def}}{=} (\ A \parallel_b B\ ) \parallel_a (\ C \parallel_c (D \parallel_d E)\ )$$

is the system of communicating components that we consider in this example. A resides at location 1, B at location 2, C at 3, D at 4, and E at location 5. We compute now the visit counts for each process. Since the processes do not branch, the branching probabilities are trivial and always equal to 1. For location 1, we choose $\theta_1^R(b)$ as reference throughput. It is easy to see that for two b-transitions to be made there is one a transition. Hence, $\overline{ba}_1 = \frac{1}{2}$ and

$$\theta_1^R(a) = \overline{ba}_1 \theta_1^R(b) = \frac{1}{2}\theta_1^R(b)$$

For location 2 (choosing the b-throughput $\theta_2^R(b)$ as reference) we obtain the same result:

$$\theta_2^R(a) = \overline{ba}_2 \theta_2^R(b) = \frac{1}{2}\theta_2^R(b)$$

For location 3, the derivations are similar: choosing $\theta_3^R(c)$ as reference, we find that $\overline{ca}_3 = \frac{1}{2}$. For location 4, $\overline{da}_4 = \frac{1}{2}$ and $\overline{dc}_4 = 1$, and finally for location 5, $\overline{da}_5 = \frac{1}{2}$. Counting the visible actions that occur in the sub-processes, we find that there are 6 equations that describe the relation between the different throughputs.

4.3.2 Global Throughput Equations

The equations for one component that we have defined with Equation (4.6) are unrelated to the equations derived for the other components. But Theorem 4.20 allows us to define a set of equations that establishes a relationship between a-throughputs of different locations.

Definition 4.22 Let $P \in \mathcal{L}_{\mathcal{CC}}$. Then we define the set $Eq^P(Q)$ of *global throughput equations* of P recursively as follows:

1. if $Q = Q_1 \|_S Q_2 \in pDecomp(P)$, then

$$Eq^P(Q) = \left\{\Theta_a^P(Q_1) = \Theta_a^P(Q_2) \mid a \in S\right\} \cup Eq^P(Q_1) \cup Eq^P(Q_2)$$

2. if $Q = Q' \setminus H \in pDecomp(P)$, then

$$Eq^P(Q) = Eq^P(Q')$$

3. $Eq^P(Q) = \emptyset$, otherwise.

Note that in the first case, Q_1 and Q_2 can be sub-processes which are only partially decomposed. In that case it is necessary to expand $\Theta_a^P(Q_1)$ and $\Theta_a^P(Q_2)$ according to Definition 4.19.

If $P_i \stackrel{\text{def}}{=} \Downarrow_i^P(P)$, then, according to Definition 4.19, $\Theta_a^P(P_i) = \theta_i^P(a)$. Hence, it is obvious that the equations of $Eq^P(P)$ indeed relate the internal throughputs of the different components to each other.

Example 4.23

We continue Example 4.21 and derive the set of equations $Eq^R(R)$. We find that

$$\begin{aligned} Eq^R(R) \;=\; & \left\{\Theta_a^R(A\|_b B) = \Theta_a^R(\ C\ \|_c\ (D\ \|_d\ E))\right\} \\ & \cup\ Eq^R(A\|_b B) \\ & \cup\ Eq^R(C\|_c(D\|_d E)). \end{aligned}$$

We expand $Eq^R(A\|_b B)$ and derive

$$\begin{aligned} Eq^R(A\|_b B) \;=\; & \left\{\Theta_b^R(A) = \Theta_b^R(B)\right\} \\ & \cup\ Eq^R(A) \quad (= \emptyset) \\ & \cup\ Eq^R(B) \quad (= \emptyset). \end{aligned}$$

The expansion of $Eq^R(C\|_c(D\|_d E))$ yields

$$\begin{aligned} Eq^R(C\|_c(D\|_d E)) \;=\; & \left\{\Theta_c^R(C) = \Theta_c^R(D\|_d E)\right\} \\ & \cup\ Eq^R(C) \quad (= \emptyset) \\ & \cup\ Eq^R(D\|_d E) \\ =\; & \left\{\Theta_c^R(C) = \Theta_c^R(D\|_d E)\right\} \\ & \cup\ \left\{\Theta_d^R(D) = \Theta_d^R(E)\right\} \\ & \cup\ Eq^R(D) \quad (= \emptyset) \\ & \cup\ Eq^R(E) \quad (= \emptyset). \end{aligned}$$

Expanding the four equations by means of Definition 4.19, we obtain

$$\Theta_a^R(A) + \Theta_a^R(B) = \Theta_a^R(C) + \Theta_a^R(D) + \Theta_a^R(E) \tag{4.7}$$
$$\Theta_b^R(A) = \Theta_b^R(B) \tag{4.8}$$
$$\Theta_c^R(C) = \Theta_c^R(D) \tag{4.9}$$
$$\Theta_d^R(D) = \Theta_d^R(E) \tag{4.10}$$

The elements in $Eq^P(P)$ form a system of linear equations. Note that for all actions a that are used for synchronisation within P and for all components P_i in which a is used, there is an equation in $Eq^P(P)$ in which the throughput $\Theta_a^P(P_i)$ occurs. We have seen above that for all $a \in \boldsymbol{\alpha}(G_i^P)$, the throughputs $\theta_i^P(a)$ can be expressed by one reference throughput and the scaling parameters $\overline{aa'}_i$. Therefore, the equations of $Eq^P(P)$ can be modified such that only reference throughputs of the different components are referred to. As a consequence, the equations that are introduced as an application of rule 1 of Definition 4.22 are linearly dependent and can be expressed by only a single equation. Since in a process $P \in \mathcal{L}_{\mathcal{CC}}$ with n locations only $n-1$ parallel operators can occur, the system of linear equations $Eq^P(P)$ is of rank $n-1$.

Example 4.24

We continue Example 4.23. Using equations (4.7)–(4.10) and the local throughput equations that we have derived in Example 4.21, we can solve this system of linear equations up to one unknown. We begin with Equation (4.7). We can rewrite the left-hand side as

$$\begin{aligned}
\Theta_a^R(A) + \Theta_a^R(B) &= \Theta_a^R(A) + \overline{ba}_2\Theta_b^R(B) && (cf.\ \text{Example 4.21}) \\
&= \Theta_a^R(A) + \overline{ba}_2\Theta_b^R(A) && (\text{by (4.7)}) \\
&= \Theta_a^R(A) + \overline{ba}_2\frac{\Theta_a^R(A)}{\overline{ba}_1} \\
&= \Theta_a^R(A)\left(1 + \frac{\overline{ba}_2}{\overline{ba}_1}\right)
\end{aligned}$$

and the right-hand side as

$$\begin{aligned}
\Theta_a^R(C) + \Theta_a^R(D) + \Theta_a^R(E) &= \Theta_a^R(C) + \Theta_a^R(D) + \overline{da}_5\Theta_d^R(E) \\
&= \Theta_a^R(C) + \Theta_a^R(D) + \overline{da}_5\Theta_d^R(D) \\
&= \Theta_a^R(C) + \Theta_d^R(D)\left(\overline{da}_4 + \overline{da}_5\right) \\
&= \Theta_a^R(C) + \Theta_c^R(C)\frac{\overline{da}_4 + \overline{da}_5}{\overline{dc}_4} \\
&= \Theta_a^R(C) + \Theta_a^R(C)\frac{\overline{da}_4 + \overline{da}_5}{\overline{ca}_3\overline{dc}_4} \\
&= \Theta_a^R(C)\left(1 + \frac{\overline{da}_4 + \overline{da}_5}{\overline{ca}_3\overline{dc}_4}\right).
\end{aligned}$$

Hence,

$$\Theta_a^R(A) = \Theta_a^R(C) \cdot \frac{\overline{ca}_3\overline{dc}_4 + \overline{da}_4 + \overline{da}_5}{\overline{ba}_1 + \overline{ba}_2} \cdot \frac{\overline{ba}_1}{\overline{ca}_3\overline{dc}_4}, \tag{4.11}$$

or, using the ratios computed in Example 4.21,

$$\Theta_a^R(A) = \frac{3}{2}\Theta_a^R(C).$$

Equation (4.11) can now be used to express all other a-throughputs of R in terms of $\Theta_a^R(C)$.

4.3.3 Branching Probabilities Revisited

The throughput equations in Sections 4.3.1 and 4.3.2 can be defined without knowledge of the steady-state probabilities of the global system, if certain requirements are met. The most crucial requirement is that branching probabilities for all choices have to be known. If global throughputs are unknown, then this generally requires that choices can be only made from stable states. Only in that case the branching probabilities can be obtained to define the local throughput equations.

However, this assumption can be relaxed in some cases. We demonstrate this by means of the next example: sometimes, the throughput equations are sufficient to express the branching probabilities of a choice directly.

Example 4.25

We consider the following (sub-)processes:

$$\begin{aligned} P_1 &\stackrel{\text{def}}{=} [\lambda_1].c.c.P_1 + [\mu_1].a.P_1 \\ Q &\stackrel{\text{def}}{=} a.[\lambda].Q + b.[\mu].Q \\ P_2 &\stackrel{\text{def}}{=} c.P_3 \\ P_3 &\stackrel{\text{def}}{=} [\lambda_3].c.c.P_3 + [\mu_3].b.P_3 \end{aligned}$$

and the global process

$$R \stackrel{\text{def}}{=} (\ P_1 \parallel_c P_2\ ) \parallel_{\{a,b\}} Q.$$

Process Q has a nondeterministic choice: we have the transitions $Q \xrightarrow{a} [\lambda].Q$ and $Q \xrightarrow{b} [\mu].Q$. However, the GMP $[\![R]\!]$ is completely s-deterministic, so that a Markov chain can be derived. This means that the nondeterministic choice of Q is resolved by the other components by means of synchronisation (external choice). In particular, the components P_1 and P_3 are controlling whether Q should perform an

a or an b. If P_1 is given control over Q, then Q has to do an a, and if P_1 hands over control to P_2, then Q has to do an b. The decision whether P_1 or P_2 triggers Q or hands over control to the respective other component is made *locally* by the component in possession of the control over Q.

Hence, the branching probabilities of state Q are well-defined. The interesting fact is now that these branching probabilities are already determined by the local throughput equations of the components P_1, P_2 and Q and the global throughput equations. The global throughput equations for R are (according to Definition 4.22):

$$\Theta_c^R(P_1) = \Theta_c^R(P_2) \tag{4.12}$$
$$\Theta_a^R(P_1) + \Theta_a^R(P_2) = \Theta_a^R(Q) \tag{4.13}$$
$$\Theta_b^R(P_1) + \Theta_b^R(P_2) = \Theta_b^R(Q). \tag{4.14}$$

Equations (4.13) and (4.14) can be simplified to

$$\begin{aligned} \Theta_a^R(P_1) &= \Theta_a^R(Q) \\ \Theta_b^R(P_2) &= \Theta_b^R(Q). \end{aligned}$$

We can now introduce another equation, which expresses the ratio between the mean number of a-synchronisations and b-synchronisations:

$$\beta_a = \frac{\Theta_a^R(Q)}{\Theta_a^R(Q) + \Theta_b^R(Q)}$$

and

$$\beta_b = \frac{\Theta_b^R(Q)}{\Theta_a^R(Q) + \Theta_b^R(Q)}.$$

As is easy to see, $\beta_a = 1 - \beta_b$ and β_b are the unknown branching probabilities of Q. The global throughput equations allow us now to express $\Theta_b^R(Q)$ by means of $\Theta_a^R(Q)$: $\Theta_b^R(Q) = C \cdot \Theta_a^R(Q)$ for a certain scaling parameter C (which we could compute, but leave unspecified for sake of simplicity). Then

$$\beta_a = \frac{\Theta_a^R(Q)}{\Theta_a^R(Q) + C \cdot \Theta_a^R(Q)} = \frac{1}{1+C} \tag{4.15}$$

and

$$\beta_b = \frac{C}{1+C}.$$

This example shows that some local parameters of components are determined by the environment, *i.e.,* other components. In this case, the local parameters are the branching probabilities of Q, and they can be expressed by means of the global throughputs Θ_a^R and Θ_b^R. That means that the global throughput equations can determine parameters within components without referring to the global state space.

Now immediately the question arises, whether this is always the case: can branching probabilities be always determined by the global throughput equations? Unfortunately, *no*, as the following example shows.

Example 4.26

We consider the processes P_1, P_2, and P_3 as in Example 4.25, and

$$\begin{aligned} Q' &= a.[\lambda].Q_1 + b.[\mu].Q' \\ Q_1 &= a.[\lambda].Q' + b.[\mu].Q'. \end{aligned}$$

Q' is strongly bisimilar to process Q from Example 4.25, but we have duplicated the nondeterministic choice. An abstract view on a fragment of the GMP $[\![Q]\!]$ is shown in Figure 4.1.

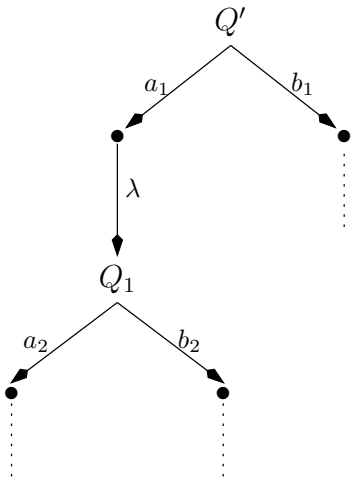


Figure 4.1: Part of $[\![Q']\!]$

We consider the process

$$R' \stackrel{\text{def}}{=} (P_1 \parallel_c P_2) \parallel_{\{a,b\}} Q'.$$

R' is, as R, s-deterministic. However, the a-throughput $\Theta_a^{R'}(Q')$ and the b-throughput $\Theta_b^{R'}(Q')$ are now distributed over the transitions labelled a_1 and a_2 and b_1 and b_2, respectively, but the ratio is unknown: at least one of the branching probabilities of Q' or Q_1 must be known. In the example of Figure 4.1, this would be sufficient, but it is no problem whatsoever to define processes that are bisimilar to Q and for which even more branching probabilities are unknown. Since the number of throughput equations that would be needed to determine these probabilities does not increase, they can obviously not be expressed by means of throughput equations only. A

consequence of all this is that throughput equations are not the only factors that govern the behaviour of components.

Nevertheless, in the example of process R', the probabilities *can* be expressed by local parameters only, without referring to the global state space of R'. The derivation is much more involved than in case of process R, as we demonstrate now.

First, we derive β_{a_2}, the probability that the a_2 transition is chosen, given that we are in state Q_1. This is the probability to make an a synchronisation under the assumption that immediately before state Q_1 has been entered, another a synchronisation has happened. This means however that component P_1 had control over component Q. That means that the probability β_{a_2} is

1. the probability that P_1 decides to let an a synchronisation happen again (with probability $\frac{\mu_1}{\lambda_1+\mu_1}$); or
2. to hand over control to P_2, receive it back immediately, and *then* do an a synchronisation (with probability $\frac{\lambda_1\lambda_3\mu_1}{(\lambda_1+\mu_1)^2(\lambda_3+\mu_3)}$); or
3. do the hand-over procedure several times (without performing a b synchronisation), and then executing action a.

In summary that means that β_{a_2} can be expressed as

$$\beta_{a_2} = \frac{\mu_1}{\lambda_1+\mu_1}\frac{1}{1-z}, \quad \text{where} \quad z = \frac{\lambda_1\lambda_3}{(\lambda_1+\mu_1)(\lambda_3+\mu_3)}.$$

The probability for transition a_1, *i.e.*, β_{a_1}, can then be derived from β_{a_2} and the quotient in Equation (4.15).

This example shows that

1. it is *sometimes* possible to express local quantities such as branching probabilities without the necessity to refer to the global Markov chain at all, but that
2. even for simple examples a profound knowledge of the behaviour of components is required.

A more practical conclusion of this example is that the rank of the system of throughput equations is not sufficient to determine whether the considered system shows nondeterminism or not.

4.4 To Synchronise Means to Wait

4.4.1 Introduction

Generally, in CTMCs it is possible to compute steady-state probabilities from throughputs and vice-versa. If $\tau(i)$ is the throughput of state i of a CTMC, then the steady-state probability for i is equal to $\tau(i)/q_i$, where q_i is the sum of the rates of all outgoing transitions of i.

We *assume* now that we know the throughputs of a local GMP G_i^P, as defined in Section 4.2.4. The question arises whether we can also derive the steady-state probabilities for G_i^P, *i.e.,* the local probabilities of location i, as we can do for CTMCs. To answer this question, we must distinguish between stable and other states. Stable states are those where all outgoing transitions are timed. Since we know the outgoing rates in such case, we can compute the steady-state probability for this state by the simple division provided in the previous paragraph.

Other states are, first of all, not stable. That means, only if they are non-synchronising, we can say what probability mass they carry: zero. The remaining states are then *synchronising*. These states usually do carry probability mass, but it is a problem to derive it. The knowledge of the throughput is not enough.

We consider a process $P \in \mathcal{L}_{\mathcal{CC}}$ and assume that a global steady-state measure $\pi : \mathit{Reach}(P) \to [0,1]$ for $[\![P]\!]$ is defined. Let $P_i \stackrel{\text{def}}{=} \Downarrow_i^P(P)$ and G_i^P be the local GMP for locations $1 \leq i \leq \#\mathit{loc}(P)$. Let $s \in \mathcal{S}_{syn}(G_i^P)$ be a synchronising state. Then $\Uparrow_i^P(s) \subseteq \mathit{Reach}(P)$ and $\phi_i^P(s) = \pi\left(\Uparrow_i^P(s)\right)$ is the local probability that component i is in state s. Even though s is synchronising, $\phi_i^P(s)$ is usually larger than 0 and consequently, the mean sojourn time for s is positive and there is a positive probability that component i is in state s.

We refer to sojourn times of synchronising states as *waiting times*: the reason why a component remains in a synchronising state is that it has to wait until the other participants in the synchronisation become ready for it.

So, we now know that synchronising states do accumulate probability mass. But this information does not help to compute the steady-state probabilities for these states, since the sojourn times of these states are generally not known locally. *It is this lack of knowledge that inhibits us from solving a local GMP like an ordinary Markov chain.*

In the following, we will characterise waiting times more thoroughly.

4.4.2 Characterisation of Waiting Times

In this section we show that the waiting time of a local synchronising state is phase-type distributed. Actually, this is no surprise: what else could be a distribution in a Markovian "environment" be, if not of phase-type? Again, we assume that $s \in \mathcal{S}_{syn}(G_i^P)$

is a synchronising state. $\Twoheaduparrow_i^P(s)$ implicitly defines a sub-GMTS of $[\![P]\!]$. We can define this as

$$W_{i,s}^P = \left(C, Act_{\mathbf{t}}, T_i\big|_C, \mathcal{R}\big|_{T_i\big|_C} \right) \tag{4.16}$$

where $C = \Twoheaduparrow_i^P(s)$ and $T_i\big|_C = T_i \cap (C \times Act_{\mathbf{t}} \times C)$. The rate function of $W_{i,s}^P$ is the restriction of $\mathcal{R}$ on $T_i\big|_C$. $W_{i,s}^P$ forms that part of the global GMP of P in which component i stays in state s. It describes the behaviour of G in which component i does not participate.

$W_{i,s}^P$ *implicitly* describes an absorbing Markov chain. Implicitly, because the absorbing states and the transitions that lead to them are not part of it. However, they can easily be retrieved from the global GMP $[\![P]\!]$: the absorbing states are all those global states which are not element of C, but which in the global GMP can be reached in one step from states in $W_{i,s}^P$.

A phase-type distribution can be represented by means of an *absorbing Markov chain* (*cf.* Appendix B) and a *starting distribution*. Since $W_{i,s}^P$ is an absorbing Markov chain, we have one part of the representation of a phase-type distribution.

The sub-GMTS $W_{i,s}^P$ is part of the global GMP G. Since G is assumed to be irreducible, the states of $W_{i,s}^P$ are positive recurrent. Hence, there is always a positive probability that the states of $W_{i,s}^P$ are eventually entered. Entering $W_{i,s}^P$ means that component i starts to wait. It is possible that $W_{i,s}^P$ can be entered via different transitions and different states. Certainly, for each of these *starting states* s' of of $W_{i,s}^P$ there exists a probability $p(s')$ that $W_{i,s}^P$ is entered via state s'.

These probabilities define a probability measure $\underline{p}$ on all states of $W_{i,s}^P$, and it is this probability distribution $\underline{p}$ that completes the representation of the waiting time distribution of state s.

We can describe the vector $\underline{p}$. To do so, we need the following theorem.

Theorem 4.27 Let $\mathbf{Q}$ be the generator matrix of an ergodic CTMC and $-\underline{q}$ the vector of diagonal elements of $\mathbf{Q}$. Let $\mathbf{E} = (e_{ij})_{n,n} = \mathbf{I} + \mathbf{Q}\mathrm{diag}(\underline{q})^{-1}$. $\mathbf{E}$ describes the EMC of $\mathbf{Q}$. Let $K \subset \{1, \ldots, n\}$, and let $\overline{K} = \{1, \ldots, n\} \setminus K$. Then, in steady-state,

$$\Pr(\text{current state is } k \in K \mid \text{a transition from } \overline{K} \text{ to } K \text{ has last occurred }) = \frac{\sum_{j \in \overline{K}} p_j e_{jk}}{\sum_{l \in K} \sum_{j \in \overline{K}} p_j e_{jl}}$$

PROOF: To prove this theorem we just have to resolve the conditional probability according

to the usual law. We can reformulate the above probabilities, assuming steady state, as

$$\Pr(\text{current state is } k \in K \mid \text{a transition from } \overline{K} \text{ to } K \text{ has last occurred }) \tag{4.17}$$

$$= \frac{\Pr(\text{current state is } k \in K \textbf{ and } \text{a transition from } \overline{K} \text{ to } K \text{ has occurred last})}{\Pr(\text{a transition from } \overline{K} \text{ to } K \text{ has occurred last})} \tag{4.18}$$

$$= \frac{\sum_{j \in \overline{K}} p_j e_{jk}}{\sum_{j \in \overline{K}} \sum_{l \in K} p_j e_{jl}}$$

$\rightarrow\bullet$

With the last theorem, we have the means to describe the starting distribution of an absorbing Markov chain describing the waiting time of a synchronising state. To use the result, however, it is necessary to define an embedded Markov chain for a GMP.

In Section 3.2.2 we have describe a reduced, irreducible GMP by a matrix

$$\mathbf{Q}' = \begin{pmatrix} \mathbf{C} & \mathbf{D} \\ \mathbf{E} & \mathbf{F} \end{pmatrix}$$

where $\mathbf{C} \in \mathbb{R}^{m \times m}$ describes transitions from immediate states to immediate states, $\mathbf{D} \in \mathbb{R}^{m \times (n-m)}$ transitions from immediate to stable states, $\mathbf{E} \in \mathbb{R}^{(n-m) \times m}$ those from stable to immediate states and finally, $\mathbf{F} \in \mathbb{R}^{(n-m) \times (n-m)}$ transitions from stable to stable states.

We define now a DTMC $\mathbf{G}$, the EMC of $\mathbf{Q}'$, as follows. Let $-\underline{f}$ be the vector of the diagonal entries of matrix $\mathbf{F}$. Then,

$$\mathbf{G} = \begin{pmatrix} \mathbf{C} & \mathbf{D} \\ \mathbf{E} \cdot \mathrm{diag}(\underline{f})^{-1} & \mathbf{F} \cdot \mathrm{diag}(\underline{f})^{-1} + \mathbf{I} \end{pmatrix}.$$

$\mathbf{G}$ is a probabilistic matrix and describes an irreducible DTMC.

Let $P \in \mathcal{L}_{\mathcal{CC}}$ with $G = (S, A, T, \mathcal{R}, P) = [\![P]\!]$ be a process with m locations. Let $G_i = (S_i, A_i, T_i, \mathcal{R}_i, \Downarrow_i(P)) = [\![\Downarrow_i(P)]\!]$ for $i = 1, \ldots, m$. Let $n = \#\mathit{Reach}(P)$. We assume that the elements of $\mathit{Reach}(P)$ are numbered from 1 to n. Let $\mathbf{G} = (g_{ij})_{n,n}$ be the EMC of the Markov process defined by P, and let $\underline{p} = (p_1, \ldots, p_n) \neq \underline{0}$ be a vector such that $\underline{p}\mathbf{G} = \underline{p}$ and $\underline{p} \cdot \underline{1} = 1$ hold. Let $s \in \mathcal{S}_{syn}(P_i)$ be a synchronising state of component i. We assume now that the numbers in the set $K \subset \{1, \ldots, n\}$ correspond to the states of the absorbing Markov chain defined by $W^P_{i,s}$ (and $\overline{K} = \{1, \ldots, n\} \setminus K$). Then, we can employ Theorem 4.27 to describe the probability distribution $\underline{\nu}$ on the states of K once a state in K has been entered. The value $\nu(l)$ for $l \in K$ is the probability of the global GMP G to be in the global state l immediately after component i has entered its local synchronising state s.

As we can see, the probability distribution $\underline{\nu}$ can be expressed in terms of global steady-state probabilities of the EMC of a CTMC. Only in special cases, it is possible to derive $\underline{\nu}$ without direct reference to the global probabilities. In Chapter 5 we will see an example.

In the following, we describe the different scenarios that are imaginable for the waiting period of a component.

1: Only Synchronising Outgoing Transitions

We still assume the synchronising state s of component i, as above. The simplest case occurs when s has only synchronising outgoing transitions, say, t_{out}. In that case, component i can only proceed when the other components that are required to synchronise with i are ready to do so. The sojourn time of component i in s is then completely determined by the GMP $W_{i,s}^P$ and the starting distribution $\underline{p}$.

Example 4.28

We consider the process $R \stackrel{\text{def}}{=} A\|_{\{a,b\}}B \in \mathcal{L}_{\mathcal{CC}}$, where

$$A \stackrel{\text{def}}{=} a.[\lambda].b.[\mu].A$$

and

$$B \stackrel{\text{def}}{=} a.[\gamma].[\delta].b.B.$$

$[\![R]\!]$ is irreducible, and we consider the state $P \stackrel{\text{def}}{=} b.[\mu].A \in \mathit{Reach}(A)$. When component A reaches P, then component B can be either in state

$$[\gamma].[\delta].b.B \qquad \text{or} \qquad [\delta].b.B \qquad \text{or} \qquad b.B.$$

Then the absorbing Markov chain describing the waiting time for the synchronising state P has the generator matrix

$$\mathbf{Q} = \begin{pmatrix} -\gamma & \gamma & 0 \\ 0 & -\delta & \delta \\ 0 & 0 & 0 \end{pmatrix}.$$

However, we do not know in which state B is, when component A starts to wait, and thus have *no* starting distribution for Q.

2: Timeouts: Mixture between Synchronising and Timed Outgoing Transitions

In the second case we want to consider that Q has more than one outgoing transition. A mixture of synchronising and timed transitions is allowed. To keep things simple, we consider only the case of one timed and one synchronising transition.

Again we assume synchronising state s of component i, now with one outgoing synchronising transition t_{out} and one outgoing *timed* transition, t_{timed}. As in (4.16), we can define the GMP $W^P_{i,s}$, and again we denote with $\underline{p}$ the starting distribution.

The sojourn time of s, however, is now not determined by $W^P_{i,s}$ and $\underline{p}$ alone: the transition t_{timed} influences this quantity as well. There is always a positive probability that t_{timed} fires and state s is left before the awaited synchronisation occurs. If this happens, component i is generally no longer willing and able to synchronise over $lbl(t_{out})$, at least not until the next synchronising state is entered. Hence, we can see the timed transition as a kind of *timeout* which preempts the waiting period in state s.

We can describe the behaviour of t_{timed}, *i.e.,* the time until it fires, by a random variable X, which is exponentially distributed with rate $\mathcal{R}(t_{timed})$. On the other hand, we can assume a random variable Y which is distributed according to the phase-type distribution described by $W^P_{i,s}$ and $\underline{p}$. Since t_{timed} is a local transition, it is completely uninfluenced and independent from other components. That means that X is stochastically independent from Y. The sojourn time of s is then the random variable $Z = \min\{X, Y\}$.

Example 4.29

We reconsider Example 4.28, but with a different process A' for location 1:

$$A' \stackrel{\text{def}}{=} a.[\lambda].\big(\, b.[\mu].A' + [\zeta].[\theta].b.A' \,\big).$$

We consider now the synchronising state $P' \stackrel{\text{def}}{=} \underline{b.[\mu].A' + [\zeta].[\theta].b.A'} \in \mathit{Reach}(A')$. This state has a "timeout" transition with rate ζ. The absorbing Markov chain describing the waiting time is the same as in Example 4.28 ($\mathbf{Q}$), but now the sojourn time of P' is also influenced by the transition with rate ζ. It is the minimum of the waiting time described by $\mathbf{Q}$ and the timeout with rate ζ. This time is also phase-type distributed and its absorbing Markov chain has generator

$$\mathbf{Q}' = \begin{pmatrix} -(\gamma+\zeta) & \gamma & 0 & \zeta \\ 0 & -(\delta+\zeta) & \delta & \zeta \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

$\mathbf{Q}'$ has an additional absorbing state which corresponds to the state $[\theta].b.A' \in \mathit{Reach}(A')$. It reflects the fact that the timeout transition disabled the synchronisation.

As in Example 4.28, we are not able to determine the mean value of the sojourn time of P', since again the starting distribution of $\mathbf{Q}$ is lacking.

4.4.3 Summary

In this section, we have characterised waiting-times of components of a GMP. Waiting times are parameters associated with synchronising states that, even if only the mean values could be derived, would allow us to derive local steady-state probabilities for components by an ordinary steady-state analysis.

We have shown that waiting times are phase-type distributed, where the associated absorbing Markov chain is a part of the global state-space of the considered GMP. We have shown that the starting probabilities of a waiting time distribution can be expressed by means of the global steady-state probabilities of the EMC underlying the considered GMP.

We have distinguished two kinds of synchronising states, where the first is a special case of the second. In the first case, a synchronising state has only one outgoing transition. Then, the waiting time is simply the time that has to pass until the other components become ready to synchronise. In the second case, the synchronising state has also outgoing timed, local transitions, which can end the waiting time of the considered component without a synchronisation happening. This time can also be characterised as the minimum of a phase-type distribution (the actual waiting time), and an exponential distribution, which is the time until the waiting time *times out* and decides to proceed along locally.

4.5 Conclusions

We have defined formally the concept of *component* for $\mathcal{YAWN}$ processes, and *local measures* on them. The most important local measures for a component are *local steady-state probabilities* and *local throughputs*. We have shown that it is possible to describe the stochastic dependency between synchronising components most naturally in terms of their local throughput equations, defined by the local GMP and its branching probabilities, and a set of *global throughput* equations that are implicitly defined by the syntactic structure of the process description. We finally have defined waiting times and their distribution that occur naturally when components synchronise. We have introduced all these concepts in terms of the global state space of the considered $\mathcal{YAWN}$ process and its global steady-state distribution.

We have identified three important quantities which would, if known, determine the local steady-state probabilities. When all branching probabilities in all components would be known, then the local and global throughput equations were completely specified. However,

we have found that this system of linear equations is not of full rank: the "*final equation*" is missing. In ordinary CTMC analysis, the global balance equations are nothing else but throughput equations, and they also are not of full rank. However, for CTMCs, there is a final equation: it is the normalisation condition, the demand that all probabilities for all states have to sum up to one. For components, we can not give such a normalisation. The reason is that for the determination of local steady-state probabilities not only the throughput, but the sojourn time for all states must be known. Only then it is possible to relate the throughput with the probability: the steady-state probability to be in a certain state is the product of the throughput of this state and its sojourn time. For synchronising states we do, however, *not* know the sojourn times. The sojourn times of the synchronising states of a component are the (mean) waiting times as defined in before. So, if we knew the (mean) waiting times for synchronising states, we could define a normalisation equation for the local steady-state probabilities (which is nothing else but the requirement that the local probabilities have to sum up to one), and then had a system of local throughput equations with full rank.

In this chapter we did not propose a technique to really obtain the required quantities. Instead, we have defined conditions and properties that can be used to test the quality of potential methodologies to derive local measures. Such a methodology can either aim to derive a reasonable value for a reference throughput, *i.e.,* to derive the "final equation" (be it in an exact or approximate fashion), or to derive waiting times for the individual components. In both cases, the measures obtained must fulfil the system of throughput equations.

In the next chapter, we will present a class of SPA processes that is special in the sense that throughputs as well as waiting times can be derived efficiently.

In Chapter 6, we will introduce a formalism that is well suited to describe the structure of waiting times more thoroughly.

Chapter 5

Phase-Type Distributions, Semi-Markov Chains, and $\mathcal{YAWN}$

In this chapter, we will present a numerical technique that allows to derive local steady-state measures very efficiently for a special class of $\mathcal{YAWN}$ processes. The state-space explosion problem for this class of processes is solved.

5.1 Introduction

In Section 3.2, we have introduced a method to derive a continuous-time Markov chain from $\mathcal{YAWN}$ processes. All steady-state performance measures can be expressed by means of the steady-state probability distribution of this CTMC. The solution of the CTMC is that of a system of linear equations, where each equation corresponds to a state of the GMP defined by the $\mathcal{YAWN}$ process. Since the state space of a $\mathcal{YAWN}$ processes generally grows exponentially with the number of concurrent components, the CTMC suffers from the state space explosion problem.

In this chapter we will present a different approach to derive performance measures from $\mathcal{YAWN}$ processes. We call this approach the **AWCI**-Technique. The name comes from three properties, **A**, **WC**, and **I**, that characterise the processes that are suitable to be considered for a solution by the **AWCI**-Technique. Consequently, we call these processes **AWCI**-processes. The algorithm is only applicable to a restricted class of $\mathcal{YAWN}$ processes and allows the derivation of performance measures which are local to the components (*cf.* Section 4.2). In fact, the algorithm computes only local steady-state probabilities for the components, but as in the general case, these probabilities can be used to derive many other local results.

As we shall see, the **AWCI**-Technique requires exponential time (in the worst case) and linear space in the number of components to compute the local measures. Hence, for this class of processes the problem of state space explosion does not exist.

The $\mathcal{YAWN}$ processes that are suitable to be solved by the **AWCI**-Technique are shaped such that their stochastic behaviour can be described in terms of a semi-Markov process (SMC, *cf.* Appendix B.3.3). This SMC can be solved efficiently. The key measures that can be derived from the SMC are *throughputs*, which can be interpreted within the $\mathcal{YAWN}$ components according to the results of Section 4.3.

To derive the throughputs from the SMC, it is necessary, as we will see, to compute the mean value of the maximum of a set of phase-type distributed random variables. This is a hard problem, as long as the standard approach to derive mean values from phase-type distributed random variables is chosen (which is described in Appendix B.4). However, in this chapter we present a method to overcome this problem. The mean value of the maximum of a set of n phase-type distributed random variables can be computed in polynomial time in n.

As a by-product of these computations, also the mean waiting times of synchronising states can be obtained. Hence, the local probabilities of the components can be completely obtained.

Preliminary versions of parts of this chapter have been published in [15, 17]. The material of Section 5.5 and Section 5.6 is completely new and yet unpublished. The idea for the application example in Section 5.7 goes back to Boudewijn Haverkort.

Outline of this chapter. In Section 5.2, we introduce the class of $\mathcal{YAWN}$ processes that are suitable to be solved by the **AWCI**-Technique. In Section 5.3 we describe how the $\mathcal{YAWN}$ processes can be reformulated in terms of semi-Markov processes. In Section 5.4 we derive the important measures from the SMC and reinterpret them in terms of the components of the original $\mathcal{YAWN}$ process. In Section 5.5 we present a new technique to compute the mean value of the maximum of a set of phase-type distributed random variables. In Section 5.6 we show that also waiting times can be obtained for the considered processes. Section 5.7 concludes the chapter with a small application example.

5.2 The Class of Processes

The applicability of our approach is restricted to a certain class of $\mathcal{YAWN}$ processes, which we will describe in this section.

Generally, the $\mathcal{YAWN}$ processes considered here are of the form

$$R \stackrel{\text{def}}{=} P_1 \|_S P_2 \|_S \cdots \|_S P_n \tag{5.1}$$

for $n \in \mathbb{N}$, $S \subseteq Com$, and $\mathcal{P} = \{P_1, \ldots, P_n\} \subseteq \mathcal{L}_{\mathcal{YAWN}}$[1]. Sometimes we abbreviate and write $R \stackrel{\text{def}}{=} !_S\mathcal{P}$. The set $\mathcal{P}$ as well as the individual $P \in \mathcal{P}$ have to fulfil three requirements.

[1] Actually, we must assume $\mathcal{P}$ to be a multi-set, since different components can be syntactically equal. However, for sake of simplicity we assume that we can distinguish all components from each other, such that we can assume that $\mathcal{P}$ is an ordinary set.

5.2.1 Three Requirements

The processes that are suitable to be considered to be solved by the **AWCI**-Technique are characterised by three requirements **A**, **WC**, and **ID**, which we describe in the following.

A: Alternation. For each $P \in \mathcal{P}$, the GMP $[\![P]\!]$ has to show an *alternating behaviour* between local, invisible or timed transitions and synchronising transitions. We demand especially that there are no choices between synchronising actions and timed or non-synchronising actions, respectively. More formally, we say that *set* $\mathcal{P}$ *has property* **A** *with respect to synchronisation set* S if for all $P \in \mathcal{P}$ and for all $P' \in \mathit{Reach}(P)$ with $P' \xrightarrow{a}$ for some $a \in \mathit{Act}$ the condition $P' \xrightarrow{\mathbf{t}}\!\!\!\!\!/\;\;$ holds. This condition ensures from the start that once a component $P \in \mathcal{P}$ wants to synchronise with another component (or better, with *all* the others, by the structure of R), it has to wait until this synchronisation really happens. Hence, a component either executes local actions or waits for the other components to synchronise with it. To keep things more simple in this chapter, we demand without loss of generality that all $P \in \mathcal{P}$ only have transitions with labels from $S \cup \{\mathbf{t}\}$. To express this requirement in terms of Section 4.2.4, we require that the branching probabilities of all components are determined locally.

Alternation is the reason why times between the execution of an synchronisation and the start of the next waiting period are determined only locally and can be described uniquely by a phase-type distribution: if t is a synchronising transition of P_i, then the starting state of the next local phase is $\mathit{dst}(t)$ with probability 1, and the absorbing states are all those synchronising states which can be reached by $\mathit{dst}(t)$ without visiting another synchronising state in between. The states that lie between starting and absorbing states are all stable and the absorbing Markov chain describing the phase-type distribution is formed out of all those states and the respective transitions connecting them.

Example 5.1

We assume the processes

$$Q \stackrel{\text{def}}{=} a.[\lambda].(Q_1 + Q_2)$$

and

$$Q_1 \stackrel{\text{def}}{=} [\mu_1].[\mu_2].b.Q'$$
$$Q_2 \stackrel{\text{def}}{=} [\gamma_1].[\gamma_2].c.Q''$$

to be a derivative of components i, P_i (where Q', Q'' are some process constants which are defined somehow but are not of interest here). In Figure 5.1 (a), the transition system of Q is depicted, where the states are denoted $s_1, \ldots, s_9$ for easier reference. If we assume that $\{a, b, c\} \subseteq S$, states s_1, s_5 and s_8 are synchronising states. Since

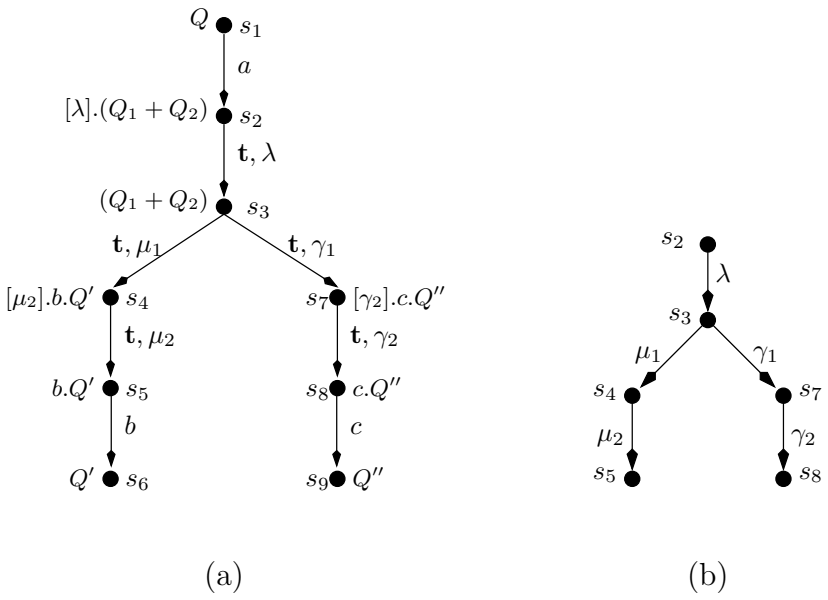


Figure 5.1: Transition system of Q and corresponding Markov chain

s_5 and s_8 are the synchronising states that can be reached next from s_2, the time, say D, between entering s_2 and entering state s_5 or s_8, respectively, is determined by all the transitions that lie between these states. Since $\mathcal{YAWN}$ is a Markovian SPA, the transitions denote exponential phases and hence, D is phase-type distributed. The corresponding absorbing Markov chain is depicted in Figure 5.1 (b) (states are denoted as in Figure 5.1 (a)).

WC: Weak (functional) congruence of components. From Equation (5.1) we see that all components synchronise over a single synchronisation set S. We define $\overline{S} = Com_{\mathbf{t}} \setminus S$. Then, for all $P, Q \in \mathcal{P}$ we require that

$$P \setminus \overline{S} \simeq Q \setminus \overline{S},$$

that is, the components have to be weakly congruent with respect to their functional behaviour only, and with respect to the actions contained in the synchronisation set S. Only those actions are left to be visible. Even the temporal information is irrelevant, since $\mathbf{t} \in \overline{S}$ (*cf.* Section 3.1.3). In the following, set $\mathcal{P}$ is said to have property **WC** with respect to a synchronisation set S if all pairs of components in $\mathcal{P}$ fulfill the above condition.

ID: Irreducibility. Apart from the two requirements **A** and **WC**, irreducibility of the components $P \in \mathcal{P}$ (*cf.* Definition 3.21) as well as the irreducibility and s-determinism of R is required. If these conditions are fulfilled, set $\mathcal{P}$ is said to have property **ID**.

5.2.2 Properties of $!_S\mathcal{P}$

In this section, we consider a set $\mathcal{P}$ of components with the properties **A, WC** and **ID** with respect to a synchronisation set S. We will derive some properties for $R \stackrel{\text{def}}{=} !_S\mathcal{P}$ which will prove to be useful in subsequent sections. It is sufficient to assume $\mathcal{P}$ having only two elements, P and Q, since the results will turn out to be valid also for larger component sets.

The first result shows that R is deadlock-free.

Lemma 5.2 Let $P' \in \mathit{Reach}(P)$ and $Q' \in \mathit{Reach}(Q)$ such that $P'\|_S Q' \in \mathit{Reach}(R)$. Whenever $P' \stackrel{a}{\twoheadrightarrow} P''$ for $a \in S$, then with probability one, Q' evolves into a state $Q'' \in \mathit{Reach}(Q)$ such that $Q' \stackrel{\mathbf{t}}{\twoheadrightarrow} Q''$ and $Q'' \stackrel{a}{\longrightarrow}$.

Proof: We assume that the probability that there is a transition sequence $Q' \stackrel{\mathbf{t}}{\twoheadrightarrow} Q''$ to a state Q'' with $Q'' \stackrel{a}{\longrightarrow}$ is smaller than one. In that case there must be a local, implicit probabilistic choice in Q', which leads to a synchronising state Q''' with $Q''' \stackrel{a}{\not\longrightarrow}$. Hence, there is a positive probability that $P'\|Q'$ evolves to $P''\|_S Q'''$, but this is an absorbing state, since P'' can not synchronise with Q'''. This is a contradiction to the irreducibility requirement of R. →•

The next lemma shows that there is a strong relation between components P and Q and the process $P\|_S Q$, if $\{P, Q\}$ have the properties **A, WC** and **ID** with respect to a synchronisation set $S \subseteq \mathit{Com}$. Again, we define $\overline{S} = \mathit{Com}_{\mathbf{t}} \setminus S$.

Proposition 5.3 Let $\{P, Q\}$ have the properties **A, WC** and **ID** with respect to a synchronisation set $S \subseteq \mathit{Com}$. Then

$$P \setminus \overline{S} \quad \simeq \quad (P\|_S Q) \setminus \overline{S}$$

The proof of Proposition 5.3 needs the following lemma.

Lemma 5.4 Let $P, Q \in \mathcal{L}_{\mathcal{YAWN}}$. Then $(P\|_S Q) \setminus \overline{S} \simeq (P \setminus \overline{S})\|_S (Q \setminus \overline{S})$

Proof: We prove that $(P\|_S Q) \setminus \overline{S}$ is even strongly bisimilar to $\big(P \setminus \overline{S}\big) \|_S \big(Q \setminus \overline{S}\big)$. We define the relation

$$B = \left\{ \left(\underline{(P'\|_S Q') \setminus \overline{S}},\ \underline{(P' \setminus \overline{S})\|_S (Q' \setminus \overline{S})} \right) \middle|\ \underline{(P'\|_S Q') \setminus \overline{S}} \in \mathit{Reach}(\underline{P\|_S Q) \setminus \overline{S}} \right\}$$

(the individual terms are underlined for better readability). Then we define the set $\mathcal{B}$ to be the symmetric and transient closure of B. We show that $\mathcal{B}$ is a strong Markovian bisimulation. We choose an arbitrary element from $\mathcal{B}$, say,

$$\left(\underline{(P' \|_S Q') \setminus \overline{S}}, \ \underline{(P' \setminus \overline{S}) \|_S (Q' \setminus \overline{S})} \right).$$

The expansion law (*cf.* Lemma 3.10) states that

$$P' \sim \sum_{i \in I} [\lambda_i].P_i + \sum_{j \in J} a_j.P_j$$

and

$$Q' \sim \sum_{k \in K} [\mu_k].Q_k + \sum_{l \in L} b_l.Q_l,$$

where I, J, K, L are appropriately chosen, pairwise disjoint index sets. Then (Lemma 3.10)

$$\begin{aligned}(P' \|_S Q') \ \sim \ & \sum_{i \in I} [\lambda_i].(P_i \|_S Q') \ + \ \sum_{\substack{j \in J \\ a_j \notin S}} a_j.(P_j \|_S Q') \\ & + \ \sum_{\substack{j \in J \\ a_j \in S \\ b_l = a_j}} a_j.(P_j \|_S Q_l) \\ & + \ \sum_{\substack{l \in L \\ b_l \notin S}} b_l.(P' \|_S Q_l) \ + \ \sum_{k \in K} [\mu_k].(P' \|_S Q_k).\end{aligned}$$

Consequently,

$$\begin{aligned}(P' \|_S Q') \setminus \overline{S} \ \sim \ & \sum_{i \in I} \mathbf{i}.\left((P_i \|_S Q') \setminus \overline{S}\right) \ + \ \sum_{\substack{j \in J \\ a_j \notin S}} \mathbf{i}.\left((P_j \|_S Q') \setminus \overline{S}\right) \\ & + \ \sum_{\substack{j \in J \\ a_j \in S \\ b_l = a_j}} a_j.\left((P_j \|_S Q_l) \setminus \overline{S}\right) \\ & + \ \sum_{\substack{l \in L \\ b_l \notin S}} \mathbf{i}.\left((P' \|_S Q_l) \setminus \overline{S}\right) \ + \ \sum_{k \in K} \mathbf{i}.\left((P' \|_S Q_k) \setminus \overline{S}\right).\end{aligned}$$

On the other hand,

$$P' \setminus \overline{S} \ \sim \ \sum_{i \in I} \mathbf{i}.\left(P_i \setminus \overline{S}\right) \ + \ \sum_{\substack{j \in J \\ a_j \notin S}} \mathbf{i}.\left(P_j \setminus \overline{S}\right) \ + \ \sum_{\substack{j \in J \\ a_j \in S}} a_j.\left(P_j \setminus \overline{S}\right)$$

and

$$Q' \setminus \overline{S} \ \sim \ \sum_{k \in K} \mathbf{i}.\left(Q_k \setminus \overline{S}\right) \ + \ \sum_{\substack{l \in L \\ b_l \notin S}} \mathbf{i}.\left(Q_l \setminus \overline{S}\right) \ + \ \sum_{\substack{l \in L \\ b_l \in S}} b_l.\left(Q_l \setminus \overline{S}\right).$$

Then we have

$$\begin{aligned}(P' \setminus \overline{S}) \parallel_S (Q' \setminus \overline{S}) \;\sim\; & \sum_{i \in I} \mathbf{i}.\,(P_i \setminus \overline{S}) \parallel_S (Q' \setminus \overline{S}) \;+\; \sum_{\substack{j \in J \\ a_j \notin S}} \mathbf{i}.\,(P_j \setminus \overline{S}) \parallel_S (Q' \setminus \overline{S}) \\ & + \sum_{\substack{j \in J \\ a_j \in S \\ b_l = a_j}} a_j.\,(P_i \setminus \overline{S}) \parallel_S (Q_l \setminus \overline{S}) \\ & + \sum_{\substack{l \in L \\ b_l \notin S}} \mathbf{i}.\,(P' \setminus \overline{S}) \parallel_S (Q_l \setminus \overline{S}) \;+\; \sum_{k \in K} \mathbf{i}.\,(P' \setminus \overline{S}) \parallel_S (Q_k \setminus \overline{S}).\end{aligned}$$

It is now easy to see that $\mathcal{B}$ is indeed a strong bisimulation. Hence, $(P \| _S Q) \setminus \overline{S} \sim (P \setminus \overline{S}) \| _S (Q \setminus \overline{S})$, which immediately leads to Lemma 5.4. →•

We now prove Proposition 5.3.

Proof (of Proposition 5.3): We consider the minimal representants of $P \setminus \overline{S}$ and $Q \setminus \overline{S}$ with respect to $\simeq$, denoted in the following by $P_{\min}$ and $Q_{\min}$, respectively (*cf.* Definition 3.15). Due to the alternating behaviour of P and Q, there are no choices between internal and synchronising actions in neither $P \setminus \overline{S}$ nor $Q \setminus \overline{S}$. As a consequence, $P_{\min}$ and $Q_{\min}$, which are isomorphic anyway, do not contain a single transition labelled with an invisible action. Consequently, in $P_{\min} \| _S Q_{\min}$, $P_{\min}$ and $Q_{\min}$ act in full synchrony, and hence

$$P_{\min} \;\simeq\; P_{\min} \| _S Q_{\min} \;\simeq\; Q_{\min}.$$

Due to the congruence property of $\simeq$ with respect to $\| _S$ and because $P_{\min} \simeq P \setminus \overline{S} \simeq Q \setminus \overline{S}$, we also have

$$P \setminus \overline{S} \;\simeq\; P \setminus \overline{S} \| _S Q \setminus \overline{S} \;\simeq\; Q \setminus \overline{S}.$$

By means of Lemma 5.4 we derive

$$P \setminus \overline{S} \;\simeq\; (P \| _S Q) \setminus \overline{S},$$

as desired. →•

The proof of Proposition 5.3 shows that if $R = P \| _S Q$, where $\{P, Q\}$ has the properties **A, WC** and **ID**, then $\{R, P, Q\}$ has these properties as well. This result can easily be generalised.

Corollary 5.5 Let $\mathcal{P} = \{P_1, \ldots, P_n\} \subseteq \mathcal{L}_{\mathcal{YAWN}}$ such that $\mathcal{P}$ has the properties **A, WC** and **ID** with respect to the synchronisation set S. If $R = P_1 \| _S \cdots \| _S P_n$, then $\{R\} \cup \mathcal{P}$ has these properties as well.

The next result solves a problem with checking property **ID**. Although it is possible to determine efficiently whether components are irreducible by means of state-space exploration, the conditions that R is s-deterministic and irreducible is not so easy to check: generally, the global state space of R must also be generated for this. The both following Lemma 5.6 and 5.7 together relieve us of this problem.

The first lemma shows that the irreducibility of R can indeed be ensured by some structural properties of the components.

Lemma 5.6 Let $\mathcal{P}$ be a set of components with properties **A** and **WC** with respect to a synchronisation set S, and let all components in $\mathcal{P}$ be irreducible. For all $P \in \mathcal{P}$ we assume that, if $P' \in Reach(P)$ is a synchronising state, and if $Q_1, Q_2, \ldots \in Reach(P)$ are the synchronising states that are reachable from P' with paths that only visit non-synchronising states, then $Q \setminus \overline{S} \simeq Q' \setminus \overline{S}$ for all $Q, Q' \in \{Q_1, Q_2, \ldots\}$.

Then $R = !_S\mathcal{P}$ is irreducible.

PROOF: Again, we assume only two components P and Q and $R = P\|_S Q$ and define $P_{\min}$ and $Q_{\min}$ to be the minimal representants of $P \setminus \overline{S}$ and $Q \setminus \overline{S}$, respectively. Since $P\setminus\overline{S} \simeq Q\setminus\overline{S}$, also $P_{\min} \simeq Q_{\min}$ and $P_{\min}$ and $Q_{\min}$ are isomorphic. For P, the synchronising successor states of a synchronising state $P' \in Reach(P)$ are all weakly congruent and are hence all represented by one state in $P_{\min}$. A consequence is that $P_{\min}$ and $Q_{\min}$ are choice-free: each choice in $P \setminus \overline{S}$ is eliminated in $P_{\min}$: if not, the choice would be between states that are not weakly congruent, which would contradict the premises. Since the irreducibility of P is also preserved in $P \setminus \overline{S}$ and also in $P_{\min}$, the transition system of $P_{\min}$ is strongly connected and forms a cycle.

The considerations are also true for Q, and hence $P_{\min}\|_S Q_{\min}$ is an irreducible process acting in full synchrony. Consequently, also $P \setminus \overline{S}\|_S Q \setminus \overline{S}$ is irreducible, and so is $P\|_S Q$. $\rightarrow\bullet$

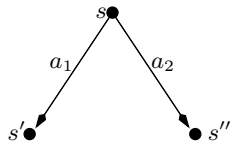
The next lemma shows that we can demand some properties of the components such that R is s-deterministic.

Lemma 5.7 Let $\mathcal{P}$ be a set of components with properties **A** and **WC** with respect to a synchronisation set S, and let all components in $\mathcal{P}$ be irreducible as well. If the components do only contain transitions with labels $a \in S$, or timed transitions, and are s-deterministic, then $R = !_S\mathcal{P}$ is s-deterministic.

PROOF: Again, it is sufficient to show the lemma only for a set of two components. The general result follows from the previous lemma in this section.

We prove the lemma by contradiction. We assume that $\mathcal{P} = \{P, Q\}$ fulfils all the requirements of the lemma, but $P\|_S Q$ is nondeterministic. Hence, there must be a state s in

$[\![P \|_S Q]\!]$ with (at least) two outgoing transitions of the form



(5.2)

If transition (s, a_1, s') corresponds to P, then (s, a_1, s'') must correspond to Q (or vice versa). Otherwise, one of the components would itself be nondeterministic, which contradicts the premises. Hence there is without loss of generality a transition with label a_1 in P and one with label a_2 in Q. Both a_1 and a_2 can not be element of S, since then the situation in (5.2) could not arise. But $a_1, a_2 \notin S$ again is a contradiction to the premises of the lemma. →●

A consequence of Lemma 5.7 is that, if a component makes only synchronising and timed transitions, then $!_S\mathcal{P}$ is s-deterministic. In the following we will hence replace property **ID** with property **I**, which together with **A** and **WC** implies property **ID** and which can be checked locally on the components:

> A set $\mathcal{P}$ of components is said to have property **I** with respect to a synchronisation set S, iff the premises of Lemma 5.6 and Lemma 5.7 are fulfilled for all $P \in \mathcal{P}$, and $\mathcal{P}$, respectively.

Although property **I** might look as a restriction to the class of components to which the **AWCI**-Technique is applicable, it is not. If there is a component P which has transitions with labels $a \in H$ with $H \cap S = \emptyset$, we can hide them safely from the view of the outside world, *i.e.,* consider $P \setminus (H \setminus \{\mathbf{i}\})$, since those transitions are obviously not supposed to synchronise with other components and do neither contribute to the quantitative nor to the functional behaviour of the component. Additionally, those transitions that now have label $\mathbf{i}$ can be safely eliminated, *i.e.,* there is a process $P' \simeq P \setminus (H \setminus \{\mathbf{i}\})$ that does not contain these transitions. P' is then suitable for the **AWCI**-Technique.

5.3 From GMP to SMC

The alternating behaviour of the components $P_1, \ldots, P_n$ allows us to characterise their stochastic behaviour in an uncommon way: rather than describing it by an ordinary CTMC, we can describe it by means of a semi-Markov process (*cf.* Appendix B.3.3)[2].

[2]Of course, a CTMC is a special form of a semi-Markov process. However, the semi-Markov processes we aim at here are generally not CTMCs.

5.3.1 SMCs from Components

In this section we derive an SMC description of components. To ease the definition we first introduce

Recipe 5.8 Let $P \in \mathcal{L}_{\mathcal{YAWN}}$ be alternating and $[\![P]\!] = (S, A, T, \mathcal{R})$. Let $s \in \mathcal{S}_{syn}(P)$. The absorbing CTMC $\{X_t\}$ *following* s is defined as follows:

- The state space $\mathcal{S}$ of $\{X_t\}$ are all those states $s' \in S, s' \neq s$ such that s' occurs in a path σ starting with s, ending with an synchronising state $s'' \in \mathcal{S}_{syn}(P)$, and no synchronising state occurs *within* σ.
- We assume that $m = \#\mathcal{S}$ and number the states in $\mathcal{S}$ from 1 to m. From now on we identify the states with their number and assume that the state being the (unique) successor of s has number 1.
- The generator matrix $\mathbf{Q}_s = (q_{ij})_{m,m}$ describing the CTMC $\{X_t\}$ is then defined as follows:

$$q_{ij} = \begin{cases} \mathcal{R}(t) & \text{if } t \in T \cap (\mathcal{S} \times A \times \mathcal{S}),\ src(t) = i \text{ and } dst(t) = j, \\ 0 & \text{otherwise.} \end{cases}$$

  The diagonal elements of $\mathbf{Q}_s$ are then defined as the negative sum of the non-diagonal elements of their respective row.

- The starting distribution of $\mathbf{Q}_s$ is defined to be $\underline{\pi}_0 = \underline{e}_1^m$ (*cf.* Appendix A.1).

The reformulation of a $\mathcal{YAWN}$ process in terms of a semi-Markov chain is now described in the following definition and the subsequent paragraph.

Definition 5.9 Let $P \in \mathcal{L}_{\mathcal{YAWN}}$ be alternating and irreducible. Then $\mathrm{SMC}(P) = (\Sigma, \mathbf{E}, J)$ is defined to be the semi-Markov chain corresponding to P, where

- $\Sigma = \mathcal{S}_{syn}(P)$ is the state space of the SMC;
- $\mathbf{E} = (e_{ij})_{n,n}$ is the embedded DTMC, where $n = \#\Sigma$;
- $J : \Sigma \longrightarrow \mathcal{F}^+$ is a mapping from states to positive distribution functions. If $s \in \Sigma$, then $J(s)$ is the phase-type distribution described by the absorbing CTMC with generator $\mathbf{Q}_s$, constructed by Recipe 5.8. Though not a distribution function, we will often identify $J(s)$ with the generator matrix $\mathbf{Q}_s$.

With Definition 5.9, only the state space and the sojourn time distributions of $\mathrm{SMC}(P)$ are actually defined. We now proceed with the definition of $\mathbf{E}$, the embedded Markov chain. We assume a certain numbering of the states in Σ and agree that a state with number i corresponds to the ith row of $\mathbf{E}$.

Although the mutual reachability of the SMC states is known, it remains to assign probabilities to the transitions to make it an actual EMC of the SMC. Reachability between synchronising states can also be expressed by means of the distributions assigned to the states. We assume $s_i, s_{j_1}, , \ldots, s_{j_k} \in \Sigma$ for $\{i, j_1, \ldots, j_k\} \subseteq \{1, \ldots, n\}$ such that $s_{j_1}, \ldots, s_{j_k}$ are the k absorbing states of $J(s_i)$. We assume that $s_i, s_{j_1}, \ldots, s_{j_k}$ each correspond to the respective rows i, $j_1, \ldots, j_k$ of $\mathbf{E}$, and so we are now going to define the probabilities $e_{i,j_1}, \ldots, e_{i,j_k}$. These values can be derived from $J(s_i)$. If $J(s_i) = \mathbf{Q}$ is the generator matrix of the absorbing CTMC describing the phase-type distribution (with, say, dimension $m \times m$) then this matrix generally can be brought in the form

$$\mathbf{Q} = (q_{ij})_{m,m} = \left(\begin{array}{c|c} \mathbf{T} & \mathbf{T}_a \\ \hline \mathbf{0} & \mathbf{0} \end{array}\right),$$

where $\mathbf{T}$ is a sub-matrix which describes the transitions between the non-absorbing states, $\mathbf{T}_a$ the transitions from non-absorbing to absorbing states, and the rows with only 0's on them the absorbing states, each row one. Again, we assume that there is a one-to-one correspondence between the rows of the matrix and the states of an transition system of the CTMC. Furthermore, we assume a one-to-one correspondence between the "absorbing" rows of $\mathbf{Q}$ and the states $s_{j_1}, \ldots, s_{j_k}$. For our purposes we can assume that we always start in the state that corresponds to the first row of the matrix.

To extract the branching probabilities from $\mathbf{Q}$, we detour. First, we turn $\mathbf{Q}$ in a uniformised DTMC $\mathbf{P}$, that is defined as

$$\mathbf{P} = (p_{ij})_{m,m} = \frac{1}{q}\mathbf{Q} + \mathbf{I} = \left(\begin{array}{c|c} \mathbf{U} & \mathbf{U}_a \\ \hline \mathbf{0} & \begin{array}{cc} 1 & \\ & \ddots \\ & 1 \end{array} \end{array}\right),$$

where $q = \max\{|q_{11}|, \ldots, |q_{mm}|\}$ and $\mathbf{I}$ the $m \times m$ unit matrix (*cf.* Appendix B.3.2). The entries p_{ij} of $\mathbf{P}$ describe the one-step probabilities of a discrete Markov process, *i.e.,* the probabilities to leave state i (once it is entered) and change to state j after a *mean* time period of $1/q$. We are interested to compute the probabilities to reach the respective absorbing states after an arbitrary number of steps. It is well known that the l-step

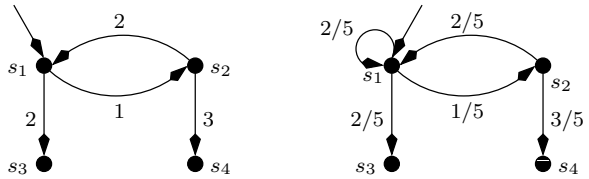


Figure 5.2: Absorbing CTMC (left) and associated DTMC (right) for Example 5.10

transition probabilities (for $l \in \mathbb{N}$) of DTMCs can be expressed by $\mathbf{P}^l$. We can derive

$$\mathbf{P}^l = \left(\begin{array}{c|c} \mathbf{U}^l & \sum_{i=0}^{l-1} \mathbf{U}^l \mathbf{U}_a \\ \hline \mathbf{0} & \begin{array}{cc} 1 & \\ & \ddots \\ & 1 \end{array} \end{array}\right).$$

The entries $u_{ij}^{(l)}$ of the matrix $\mathbf{U}^l$ denote the probabilities to end up in state j after starting in state i and jumping exactly l times. This holds also for the entries of the matrix $\sum_{i=0}^{l-1} \mathbf{U}^i \mathbf{U}_a$, but these probabilities can also be interpreted differently: they denote the probabilities to reach an absorbing state in at most l steps. Since we want to know the probabilities to reach an absorbing state in an arbitrary number of steps, we are interested in the limes $\lim_{l\to\infty} \mathbf{P}^l$, especially in the up-right sub-matrix, which then becomes the infinite sum $\sum_{i=0}^{\infty} \mathbf{U}^i \mathbf{U}_a$. The sum does converge[3] and we can write it down in closed-form:

$$\begin{aligned} \sum_{i=0}^{\infty} \mathbf{U}^i \mathbf{U}_a &= (\mathbf{I} - \mathbf{U})^{-1} \mathbf{U}_a \\ &= (\mathbf{I} - \mathbf{U})^{-1} \frac{\mathbf{T}_a}{q} \\ &= (q\mathbf{I} - q\mathbf{U})^{-1} \mathbf{T}_a \\ &= (q\mathbf{I} - (\mathbf{T} + q\mathbf{I}))^{-1} \mathbf{T}_a \\ &= -\mathbf{T}^{-1} \mathbf{T}_a. \end{aligned}$$

Since we always deal with absorbing Markov chains with a fixed starting state, we can express the probabilities to eventually reach the respective absorbing states by the row vector

$$\underline{\beta} = (\beta_1, \ldots, \beta_k) = -(1, 0, \ldots, 0)\mathbf{T}^{-1}\mathbf{T}_a.$$

[3] This is a general property of a matrix like U. See [55].

where β_l corresponds to state s_{j_l} of SMC(P). Note that a similar derivation is used to obtain branching probabilities in the course of elimination of vanishing markings in GSPN analysis [103, 102], *cf.* Section 3.2.2.[4]

Example 5.10

We consider the absorbing CTMC with two absorbing states, depicted in Figure 5.2. The generator matrix is

$$\mathbf{Q} = \begin{pmatrix} -3 & 1 & 2 & \cdot \\ 2 & -5 & \cdot & 3 \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \quad \text{and} \quad \mathbf{P} = \begin{pmatrix} \frac{2}{5} & \frac{1}{5} & \frac{2}{5} & \cdot \\ \frac{2}{5} & \cdot & \cdot & \frac{3}{5} \\ \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & 1 \end{pmatrix}$$

Starting state is s_1. One would expect that the probability $p(s_3)$ to eventually reach state s_3 satisfies the recurrence

$$p(s_3) = \frac{2}{5} \cdot p(s_3) + \frac{2}{25} \cdot p(s_3) + \frac{2}{5}.$$

Hence,

$$p(s_3) = \frac{2}{5} \cdot \frac{1}{1 - \frac{2}{5} - \frac{2}{25}} = \frac{10}{13}.$$

The probability for s_4 is consequently $p(s_4) = 1 - p(s_3) = \frac{3}{13}$.

If we employ the equations derived in this section, we have to compute $-\mathbf{T}^{-1}\mathbf{T}_a$. A quick `Maple` session reveals that

$$\mathbf{T}^{-1} = \begin{pmatrix} -\frac{5}{13} & \frac{-1}{13} \\ \frac{-2}{13} & \frac{-3}{13} \end{pmatrix},$$

and hence

$$-\mathbf{T}^{-1}\mathbf{T}_a = \begin{pmatrix} \frac{10}{13} & \frac{3}{13} \\ \frac{4}{13} & \frac{9}{13} \end{pmatrix}.$$

Since we assume s_1 to be the starting state, we are only interested in the first line and hence, $\beta = (\frac{10}{13}, \frac{3}{13})$, as expected.

[4] One might ask whether it is not better to derive this result via the embedded Markov chain. We demonstrate now that this is just a matter of personal taste. Let $\underline{q}$ be the vector of diagonal entries of $\mathbf{T}$. Define $\mathbf{D} = -\text{diag}(\underline{q})$. We redefine the matrices $\mathbf{U}$ and $\mathbf{U}_a$ as $\mathbf{U} = \mathbf{I} + \mathbf{D}^{-1}\mathbf{T}$ and $\mathbf{U}_a = \mathbf{D}^{-1}\mathbf{T}_a$. Again we are interested in the series $\sum_{i=0}^{\infty} \mathbf{U}^i\mathbf{U}_a$. Then $\sum_{i=0}^{\infty} \mathbf{U}^i\mathbf{U}_a = (\mathbf{I} - \mathbf{U})^{-1}\mathbf{U}_a = (\mathbf{I} - \mathbf{I} - \mathbf{D}^{-1}\mathbf{T})^{-1}\mathbf{D}^{-1}\mathbf{T}_a = -\mathbf{T}^{-1}\mathbf{T}_a$, as before.

Coming back to the definition of the EMC $\mathbf{E}$, we define the ith row of $\mathbf{E}$ now as follows: for $l = 1, \ldots, n$:

$$e_{i,l} = \begin{cases} \beta_r & \text{if } l = j_r \text{ for a } r \in \{1, \ldots, k\} \\ 0 & \text{else.} \end{cases}$$

The experienced reader will notice at once that the probability vectors $\underline{\beta}$ derived above are in fact the steady-state probabilities for the absorbing states of the absorbing Markov chain under the assumption that it is started in state 1 with probability 1. The steady-state probabilities can be stated in a different way, namely by $\underline{\pi}_\infty = \lim_{t\to\infty} \underline{\pi}(t)$, where

$$\underline{\pi}(t) = \underline{\pi}_0 \mathrm{e}^{\mathbf{Q}t},$$

and $\underline{\pi}_0$ is a vector of starting probabilities (as mentioned above, $\underline{\pi}_0 = (1, 0, \ldots, 0)$ in our case). The vector $\underline{\beta}$ is then a sub-vector of $\underline{\pi}_\infty$. Though this fact is not of practical use in this section, we will need it later.

5.3.2 Combining Two Processes

We have seen that, if $\{P, Q\} \subseteq \mathcal{L}_{\mathcal{YAWN}}$ which fulfils requirements **A, WC** and **I** for a certain synchronisation set S, then $R = P\|_S Q$ does fulfil these requirements as well (Corollary 5.5). Especially properties **A** and **I** are of interest to us, because they allow us to derive an SMC for R in the same fashion as described in the previous section.

In this section we want to explore the properties of $\mathrm{SMC}(R) = (\Sigma_R, \mathbf{E}_R, J_R)$. We are especially interested in the structure of $\mathbf{E}_R$ and of the delay distributions. We suspect that the branching probabilities of the combined processes can somehow be derived by simple multiplication of the branching probabilities of the components due to the fact that the probabilistic choices in the components are completely independent from each other.

Since $P \setminus \overline{S} \simeq (P\|_S Q) \setminus \overline{S}$, we can define a relation $\doteq$ between the synchronising states of P and Q by means of $\mathit{Reach}(P\|_S Q)$: if $P'\|_S Q' \in \mathit{Reach}(P\|_S Q)$ is a synchronising state, then P' and Q' are synchronising as well, and we define $P' \doteq Q'$. If we consider now each synchronising state $P'\|_S Q' \in \mathit{Reach}(R)$ as a state of an SMC, then we can assign a phase-type distribution that reflects the time until the next synchronisation happens, *i.e.,* until (one of) the next synchronising states is reached. Intuitively, the next synchronisation happens, when both components are ready to participate. If the (random) time to get ready is denoted X_P for P and X_Q for Q, then the time for $P\|_S Q$ is $X_P \boxplus X_Q$. X_P and X_Q are random variables, and if F_P and F_Q denote their distribution functions, then the distribution of $X_P \boxplus X_Q$ is $F_P \cdot F_Q$ (*cf.* Appendix A.3 for notation). The maximum of two phase-type distributed random variable is phase-type distributed and can be represented by the Kronecker sum of the absorbing Markov chains representing the individual distributions (*cf.* Section A.2).

In the following, we determine the branching probabilities between the synchronising states of $P\|_S Q$. Let hence $P'\|_S Q' \in \mathit{Reach}(P\|_S Q)$ be a synchronising state, $\mathbf{Q}_1$ be the absorbing

Markov chain describing the distribution of P' and $\mathbf{Q}_2$ that of Q'. Then, the distribution of $P' \|_S Q'$ is described by the absorbing CTMC with generator $\mathbf{Q}_1 \oplus \mathbf{Q}_2$. Let $\underline{\pi}_0^{(i)}$ be the starting probabilities of $\mathbf{Q}_i$, for $i = 1, 2$. Then $\underline{\pi}_0 = \underline{\pi}_0^{(1)} \otimes \underline{\pi}_0^{(2)}$ is the starting probability vector of $\mathbf{Q}_1$ and $\mathbf{Q}_2$.

Lemma 5.11 Let $\mathbf{Q}_i$, $i = 1, 2$ be the generator matrices of two absorbing Markov chains with starting distributions $\underline{\pi}_0^{(1)}$ and $\underline{\pi}_0^{(2)}$, respectively. Let $\underline{\pi}_0 = \underline{\pi}_0^{(1)} \otimes \underline{\pi}_0^{(2)}$ and $\underline{\pi}^{(i)}(t) = \underline{\pi}_0^{(i)} \mathrm{e}^{\mathbf{Q}_i t}$ for $i = 1, 2$. Then

$$\underline{\pi}(t) = \underline{\pi}_0 \mathrm{e}^{(\mathbf{Q}_1 \oplus \mathbf{Q}_2)t} = \underline{\pi}^{(1)}(t) \otimes \underline{\pi}^{(2)}(t).$$

Proof:

$$\begin{aligned}
\underline{\pi}(t) &= \underline{\pi}_0 \mathrm{e}^{(\mathbf{Q}_1 \oplus \mathbf{Q}_2)t} \\
&= \underline{\pi}_0 \mathrm{e}^{(\mathbf{Q}_1 \otimes \mathbf{I}_2)t + (\mathbf{I}_1 \otimes \mathbf{Q}_2)t} \\
&= \underline{\pi}_0 \mathrm{e}^{(\mathbf{Q}_1 t \otimes \mathbf{I}_2) + (\mathbf{I}_1 \otimes \mathbf{Q}_2 t)} \\
&= \underline{\pi}_0 \mathrm{e}^{\mathbf{Q}_1 t \otimes \mathbf{I}_2} \mathrm{e}^{\mathbf{I}_1 \otimes \mathbf{Q}_2 t}
\end{aligned}$$

By means of repeated application of the equation $(\mathbf{A} \otimes \mathbf{B})^n = (\mathbf{A}^n \otimes \mathbf{B}^n)$ (*cf.* Appendix A.2), the exponential $\mathrm{e}^{\mathbf{Q}_1 t \otimes \mathbf{I}_2}$ can be expressed as

$$\begin{aligned}
\mathrm{e}^{\mathbf{Q}_1 t \otimes \mathbf{I}_2} &= \sum_{n=0}^{\infty} \frac{(\mathbf{Q}_1 t \otimes \mathbf{I}_2)^n}{n!} \\
&= \sum_{n=0}^{\infty} \frac{(\mathbf{Q}_1 t)^n \otimes \mathbf{I}_2}{n!} \\
&= \sum_{n=0}^{\infty} \frac{(\mathbf{Q}_1 t)^n}{n!} \otimes \mathbf{I}_2 \\
&= \mathrm{e}^{\mathbf{Q}_1 t} \otimes \mathbf{I}_2,
\end{aligned}$$

and $\mathrm{e}^{\mathbf{I}_1 \otimes \mathbf{Q}_2 t}$ accordingly. Then

$$\begin{aligned}
\underline{\pi}(t) &= \underline{\pi}_0 \left(\mathrm{e}^{\mathbf{Q}_1 t} \otimes \mathbf{I}_2\right) \left(\mathbf{I}_1 \otimes \mathrm{e}^{\mathbf{Q}_2 t}\right) \\
&= \underline{\pi}_0 \left(\mathrm{e}^{\mathbf{Q}_1 t} \otimes \mathrm{e}^{\mathbf{Q}_2 t}\right) \\
&= \left(\underline{\pi}_0^{(1)} \otimes \underline{\pi}_0^{(2)}\right) \left(\mathrm{e}^{\mathbf{Q}_1 t} \otimes \mathrm{e}^{\mathbf{Q}_2 t}\right) \\
&= \underline{\pi}_0^{(1)} \mathrm{e}^{\mathbf{Q}_1 t} \otimes \underline{\pi}_0^{(2)} \mathrm{e}^{\mathbf{Q}_2 t} \\
&= \underline{\pi}^{(1)}(t) \otimes \underline{\pi}^{(2)}(t)
\end{aligned}$$

→●

A consequence of this is that

$$\underline{\pi}_\infty = \underline{\pi}_\infty^{(1)} \otimes \underline{\pi}_\infty^{(2)}.$$

Hence, the branching probabilities of $\mathbf{Q}_1 \oplus \mathbf{Q}_2$ can be obtained from those of $\mathbf{Q}_1$ and $\mathbf{Q}_2$ by simple multiplication, which implies that they are stochastically independent from each other.

Kronecker Product of Local EMCs

Why can we not combine the EMCs $\mathbf{E}_P$ and $\mathbf{E}_Q$ directly, *i.e.,* compute $\mathbf{E}_P \otimes \mathbf{E}_Q$? The reason is that, if $\mathbf{E}_P$ is of dimension n and $\mathbf{E}_Q$ of m, then $\mathbf{E}_P \otimes \mathbf{E}_Q$ is of dimension mn. But $\mathbf{E}_R$ is usually much smaller, sometimes even just as small as $\mathbf{E}_P$ or $\mathbf{E}_Q$. The reason is that in $\mathbf{E}_P \otimes \mathbf{E}_Q$ all rows of $\mathbf{E}_P$, *i.e.,* all states of $\mathrm{SMC}(P)$, are combined with all rows of $\mathbf{E}_Q$, *i.e.,* all states of $\mathrm{SMC}(Q)$. This is not appropriate, since not all of these combinations correspond to synchronising states of R. In fact, $\mathbf{E}_P \otimes \mathbf{E}_Q$ describes a DTMC which consists of several unconnected parts. Only that part that contains the state which corresponds to $P \|_S Q$ is of interest to us. Instead of constructing $\mathbf{E}_P \otimes \mathbf{E}_Q$, and extracting the part that interests us, we construct it directly: by means of the relation $\doteq$[5].

We now define the operation $\diamond_S$ on SMCs, for which $\mathrm{SMC}(P \|_S Q) = \mathrm{SMC}(P) \diamond_S \mathrm{SMC}(Q)$, where S, as usual, denotes the synchronisation set. If clear from the context, we omit the subscript and write simply $\diamond$.

Definition 5.12 (Diamond Product $\diamond_S$) Let $\{P, Q\} \subseteq \mathcal{L}_{\mathcal{YAWN}}$ have properties **A**, **WC** and **I** with respect to a synchronisation set S. Let $R = P \|_S Q$ and $\doteq \subseteq Reach(P) \times Reach(Q)$ be defined as above. Let $\mathrm{SMC}(P) = (\Sigma_P, \mathbf{E}_P, J_P)$ and $\mathrm{SMC}(Q) = (\Sigma_Q, \mathbf{E}_Q, J_Q)$.

Then we define the Diamond Product $\mathrm{SMC}(P) \diamond_S \mathrm{SMC}(Q)$ to be the SMC $(\Sigma, \mathbf{E}, J)$, where:

- Σ is the set of synchronising states of R.
- The matrix $\mathbf{E}$ is of dimension $n \times n$, where $n = \# \doteq$ and the entries $e_{(ij),(kl)}$ are defined as follows:
  - if i corresponds to synchronising state $P' \in Reach(P)$, j to $P'' \in Reach(P)$,
  - k to $Q' \in Reach(Q)$ and l to $Q'' \in Reach(Q)$, and
  - if $P' \doteq Q'$ and $P'' \doteq Q''$,

  then
  $$e_{(ij),(kl)} = e^{(P)}_{ij} \cdot e^{(Q)}_{kl}.$$
- The distribution functions for all $P' \|_S Q' \in \Sigma$ are described as
  $$J_R(P' \|_S Q') = J_P(P') \oplus J_Q(Q').$$

Lemma 5.13 Let P, Q, R, S be defined as in Definition 5.12. Then

$$\mathrm{SMC}(R) = \mathrm{SMC}(P) \diamond_S \mathrm{SMC}(Q)$$

[5]The deeper reason why $\mathbf{E}_P \otimes \mathbf{E}_Q$ disintegrates in several independent sub-matrices is that both $\mathbf{E}_P$ and $\mathbf{E}_Q$ are *periodic* and hence irreducible, but not ergodic. As a consequence, not all combinations of states from $\mathbf{E}_P$ and $\mathbf{E}_Q$ can be reached from one designated state. Stated differently, if $\mathbf{E}_P$ and $\mathbf{E}_Q$ were aperiodic (or at least one of them), then $\mathbf{E}_P \otimes \mathbf{E}_Q$ would be aperiodic as well and each combination of states could be reached.

Proof: Follows from the considerations of this section. →•

To keep notation simple, we will sometimes write $\mathbf{E}_P \diamond_S \mathbf{E}_Q$ for $\mathbf{E}$.

5.3.3 Combining more than Two Processes

If we have a set $\mathcal{P} = \{P_1, \ldots, P_n\} \subseteq \mathcal{L}_{\mathcal{YAWN}}$ which has the properties **A, WC** and **I** with respect to synchronisation set S, we can derive $\mathrm{SMC}(R)$ for $R = P_1 \|_S \cdots \|_S P_n$ by repeated application of $\diamond$:

$$\mathrm{SMC}(R) = (\cdots(\mathrm{SMC}(P_1) \diamond_S \mathrm{SMC}(P_2)) \diamond_S \cdots \diamond_S \mathrm{SMC}(P_{n-1})) \diamond_S \mathrm{SMC}(P_n).$$

5.4 SMC Results to SPA Results

In this section we are going to explain why a reformulation of SPA models in terms of SMC is useful. What we want to do is to solve the SMC. We will show that we obtain results that can be interpreted directly within the original model.

5.4.1 Relevant Information: Steady-State Probabilities

The structure of the SPA model and the derived SMC is very similar. In fact, the SMC can be seen as a reformulation of the original model. The entities that relate both models are synchronising states.

We assume again a set $\{P_1, \ldots, P_n\} \subseteq \mathcal{L}_{\mathcal{YAWN}}$ with the properties **A, WC** and **I** with respect to synchronisation set S, and let $R = P_1 \|_S \cdots \|_S P_n$. We define $\mathrm{SMC}(P_i) = (\Sigma_i, \mathbf{E}_i, J_i)$ and $\mathrm{SMC}(R) = (\Sigma_R, \mathbf{E}_R, J_R)$.

We want to solve $\mathrm{SMC}(R)$. To do so, we have to do three things:

1. We have to solve the EMC $\mathbf{E}_R$ and derive a steady-state probability vector $\underline{p}$;
2. We have to compute the mean values $\mu_s^{(R)}$ for all $s \in \Sigma_R$;
3. We have to actually compute the steady-state solution of $\mathrm{SMC}(R)$.

Steady-State Solutions of the EMC

We know that $\mathbf{E}_R$ is a sub-matrix of $\bigotimes_{i=1}^{n} \mathbf{E}_i$. It is common knowledge that, if $\underline{\pi}^{(i)}$ are the steady-state solutions of $\mathbf{E}_i$ for $i = 1, \ldots, n$, then the vector $\underline{\pi} = \bigotimes_{i=1}^{n} \underline{\pi}^{(i)}$ is a solution of the equation

$$\underline{\pi} \bigotimes_{i=1}^{n} \mathbf{E}_i = \underline{\pi}.$$

This can be shown by repeated application of Equation (A.1).

As stated in Section 5.3.2, $\bigotimes_{i=1}^{n} \mathbf{E}_i$ disintegrates into several independent sub-matrices, from which only one is interesting for us. Therefore, we have to choose all those entries from $\underline{\pi}$ that correspond to a state of $\mathbf{E}_R$. Since the $\underline{\pi}\ \underline{1} = 1$, the chosen entries generally do not sum up to one and must be renormalised.

Mean Sojourn Times of the SMC States

For all states $s \in \Sigma_R$, we have to compute the mean value of a random variable distributed according to $J_R(s)$. By definition, $J_R(s)$ is a Kronecker sum of absorbing Markov chains representing phase-type distributions. Even though a Kronecker representation of a matrix is very memory efficient, the computation times still grow exponentially with the number of summands.

For the time being, we postpone considerations about the actual computations until Section 5.5 and assume, that the proper value for $\underline{\mu}$ is available.

Steady-State Solutions of SMC(R)

Combining the steady-state solution $\underline{p}^{(R)}$ for $\mathbf{E}_R$ with the mean values $\underline{\mu}$, we are able to compute the steady-state probabilities for SMC(R) by means of Equation B.6, *i.e.*,

$$\underline{\sigma}^{(R)} = \frac{1}{\underline{p}^{(R)}\ \underline{\mu}^T}\ \underline{p}^{(R)}\ \mathrm{diag}(\underline{\mu}).$$

5.4.2 Throughputs for the $\mathcal{YAWN}$ Model

As for CTMCs, we can derive throughputs for states and transitions of SMC(R). Again, the throughput of a state is the mean number of visits to this state in unit time, and that of the transitions the mean number of instances per unit time this transition is "used". The vector $\underline{\tau}^{(R)} = \sigma^{(R)} \cdot \underline{\mu}^T$ is hence the vector of the state throughputs of R. Accordingly, $\mathbf{T}^{(R)} = \mathrm{diag}(\underline{\tau})\mathbf{E}_R$ is the vector of transition throughputs of SMC(R).

In this section, we will use the results of Chapter 4 to relate the throughputs derived for SMC(R) to the performance measures of R.

Visit Counts for the Components

We know from Section 4.2.4 that visit counts are scaling parameters that allow us to express the throughputs of states of a GMP relative to one reference throughput. To obtain visit counts, the branching probabilities of the GMP must be known for all transitions.

The good news is that, whenever a set $\mathcal{P}$ of $\mathcal{YAWN}$ processes has the properties **A**, **WC**, and **I** with respect to a synchronisation set S, then the branching probabilities for each $P \in \mathcal{P}$ are well-defined. This is an immediate consequence of the fact that a state of $[\![P]\!]$ is either stable, or synchronising without timeout. The fact that $!_S\mathcal{P} \setminus \overline{S} \simeq P \setminus \overline{S}$ for all $P \in \mathcal{P}$ ensures that no state of P becomes unreachable due to synchronisation constraints. Therefore, $[\![P]\!]$ not only describes the potential but also the actual behaviour of the component described by P. It is therefore possible to define for each P a system of linear equations of the form described in Section 4.2.4 to derive the visit counts for each component.

Once the visit counts are obtained, we must derive for each component i the scaling parameters $\overline{ab}_i$ for $a, b \in S$ to describe the relationship between a-throughputs and b-throughputs (*cf.* Section 4.3.1). Note that, due to the special structure of the process R, the a-throughputs for all $a \in S$ are equal for all components. Therefore, for $a, b \in S$,

$$\overline{ab}_i = \overline{ab}_j$$

for all $i, j \in \{1, \ldots, \#loc(R)\}$. We omit the indices in the following.

Throughputs

The states of SMC(R) correspond by definition to the synchronising states of R. We interpret the state throughputs that we have derived for SMC(R) as the throughputs for the synchronising states for R.

Each synchronising state of R is synchronising over a unique action $a \in S$. Since we now know the throughputs for each of these states, we can compute the values $\theta^R(a)$ for each $a \in S$. Since

$$\theta^R(a) = \theta^P(a)$$

for each $a \in S$ and $P \in \mathcal{P}$, we can interpret the a-throughputs of R directly within its components $P \in \mathcal{P}$.

The throughput relations within a component are described by the visit counts. It is therefore necessary to derive the reference throughput from one of the a-throughputs. For component i, we know that

$$\theta_i^P(a) = \sum_{\substack{t \in T_i \\ lbl(t) = a}} v_i(t)\nu_i$$

(*cf.* Section 4.3.1). Consequently,

$$\nu_i = \frac{\theta_i^P(a)}{\sum\limits_{\substack{t \in T_i \\ lbl(t)=a}} v_i(t)}.$$

Then, for all states $s \in S_i$, we have

$$\tau(s) = v_i(s)\nu_i.$$

Steady-state probabilities

Steady-state probabilities can now be obtained for all stable states of all components. For the synchronising states this is not possible, since we do not know their sojourn times yet. However, as already mentioned before, we can derive the sojourn times of synchronising states with a technique that we describe in Section 5.6. Then we can also derive steady-state probabilities for synchronising states.

5.4.3 Global Probabilities from Local Probabilities?

We consider now the question whether it is possible to compute global steady-state measures from the local probabilities. In the following we demonstrate why this is not possible.

We know throughputs for the starting states of absorbing Markov chains. As demonstrated in Appendix B.4, we can derive steady-state probabilities for the states of this Markov chain. Following Equation (B.8), we have the following system of linear equations:

$$\underline{\pi}\mathbf{T} = (-\tau, 0, \ldots, 0),$$

where $\mathbf{T} = \langle J_R(P') \rangle$ and τ is the throughput. Since $J_R(P')$ is the Kronecker sum of all generator matrices $J_i(P_i')$ for $i = 1, \ldots, n$, one might wonder whether the solution of the above system of linear equations, $\underline{\pi}$, can not be obtained from the local probabilities $\underline{\pi}_i$, for example by means of $\otimes$. Unfortunately, this is not possible, as the next examples demonstrates:

Example 5.14

We consider two absorbing CTMCs with generators $\mathbf{Q}_1$ and $\mathbf{Q}_2$, where

$$\mathbf{Q}_1 = \begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad \mathbf{Q}_2 = \begin{pmatrix} -2 & 2 & 0 \\ 0 & -2 & 2 \\ 0 & 0 & 0 \end{pmatrix}$$

We choose to compute the probability vectors π_1 and π_2 for the throughput $\tau = 1/4$, *i.e.,* $\underline{\pi}_1$ is the solution of the system of linear equations $\underline{\pi}_1 \langle \mathbf{Q}_1 \rangle = -\tau(1,0)$ and $\underline{\pi}_2$ that of $\underline{\pi}_2 \langle \mathbf{Q}_2 \rangle = -\tau(1,0)$. Then

$$\pi_1 = \left(\frac{1}{4}, \frac{1}{4}\right) \qquad \text{and} \qquad \pi_2 = \left(\frac{1}{8}, \frac{1}{8}\right).$$

The Kronecker product of both vectors is

$$\pi_1 \otimes \pi_2 = \frac{1}{32}(1,1,1,1).$$

On the other hand,

$$\underline{\pi} = \left(\frac{1}{12}, \frac{1}{18}, \frac{1}{9}, \frac{1}{36}, \frac{1}{27}, \frac{5}{27}, \frac{1}{72}, \frac{7}{216}\right)$$

is the solution of the system of linear equations $\underline{\pi} \langle \mathbf{Q}_1 \oplus \mathbf{Q}_2 \rangle = -\tau(1,0,0,0,0,0,0,0)$. Apart from the fact that both vectors have different length, there is no relation between the values of both vectors.

The example shows that the state probabilities of two components are not independent from each other and that we have therefore *no product-form*.

5.5 The Mean Value of the Maximum

As described in Appendix B.3.3, the solution of an SMC requires the computation of the steady-state solution of the EMC and the determination of the mean values of the sojourn time distributions. In the previous section the sojourn time distributions of SMC($P_1 \|_S P_2 \|_S \cdots \|_S P_n$) have been defined as Kronecker sums of absorbing Markov chains which describe the sojourn time distributions of the individual components. As already pointed out in Chapter 1, the Kronecker representation of (generator) matrices is very memory efficient, but does not really avoid the state-space explosion problem. Computations on matrices thus represented grow still exponentially with the number of Kronecker factors.

In this section, we will introduce an algorithm that computes the mean value of the maximum of phase-type distributed random variables in an efficient way: the size of the state space grows only linearly and the computation time only grows polynomially in the number of components and the size of the considered absorbing Markov chains, rather than exponentially.

5.5.1 A Bit of Random Variable Arithmetic

To begin with, we first consider some properties of the maximum of a set of positive random variables. If $X_1, X_2, \ldots, X_n$ are positive random variables with CDF F_i and pdf f_i for $i = 1, \ldots, n$, then it is well known that the CDF of $X_1 \boxplus \cdots \boxplus X_n$ is $\prod_{i=1}^{n} F_i$. If X_i is phase-type distributed according to absorbing Markov chain $\mathbf{Q}_i$ and starting distribution $\underline{\pi}_0^i$, then $\mathbf{Q} = \bigoplus_{i=1}^{n} \mathbf{Q}_i$ with the starting distribution $\underline{\pi}_0 = \bigotimes_{i=1}^{n} \underline{\pi}_0^{(i)}$ describes the absorbing Markov chain for the distribution of $X_1 \boxplus \cdots \boxplus X_n$.

We want to compute

$$E[X_1 \boxplus \cdots \boxplus X_n]. \tag{5.3}$$

We now pick an arbitrary random variable out of the set $\{X_1, \ldots, X_n\}$, X_1, say. Then we can rewrite (5.3) as follows [6]:

$$\begin{aligned} E[X_1 \boxplus \cdots \boxplus X_n] &= E\left[\frac{(X_1 \boxplus \cdots \boxplus X_n)\, X_1}{X_1}\right] \\ &= E\left[\left(\frac{X_1}{X_1} \boxplus \frac{X_2}{X_1} \boxplus \cdots \boxplus \frac{X_n}{X_1}\right) X_1\right] \\ &= E\left[\left(0 \boxplus \frac{X_2}{X_1} \boxplus \cdots \boxplus \frac{X_n}{X_1}\right) X_1\right] \\ &= E\left[\left(\left(0 \boxplus \frac{X_2}{X_1}\right) \boxplus \left(0 \boxplus \frac{X_3}{X_1}\right) \boxplus \cdots \boxplus \left(0 \boxplus \frac{X_n}{X_1}\right)\right) X_1\right] \end{aligned} \tag{5.4}$$

We define

$$Y_i^{(2)} = 0 \boxplus \frac{X_i}{X_1}$$

for $i = 2, \ldots, n$ and can rewrite

$$E[X_1 \boxplus \cdots \boxplus X_n] = E[X_1] + E\left[\left(Y_2^{(2)} \boxplus Y_3^{(2)} \boxplus \cdots \boxplus Y_n^{(2)}\right)\right]. \tag{5.5}$$

The random variables $Y_i^{(2)}$, for $i = 2, \ldots, n$, are non-negative since they are all bounded by 0. Hence, we can repeat the derivation from (5.4) to (5.5), now the $Y_j^{(2)}$, $j = 2, \ldots, n$ taking the roles of the X_i, $i = 1, \ldots, n$. Again we choose a variable, say, $Y_2^{(2)}$. We once again derive an equation, similar to (5.5), to obtain:

$$E[X_1 \boxplus \cdots \boxplus X_n] = E[X_1] + E[Y_2^{(2)}] + E\left[\left(Y_3^{(3)} \boxplus Y_4^{(3)} \boxplus \cdots \boxplus Y_n^{(3)}\right)\right] \tag{5.6}$$

where

$$Y_i^{(3)} = 0 \boxplus \frac{Y_i^{(2)}}{Y_2^{(2)}}$$

[6] Note that all expressions within the $E[\cdot]$ operators are in $(\boxplus, \odot)$ notation, *cf.* Appendix A.3.

for $i = 3, \ldots, n$. Applying this procedure $n-1$ times, we eventually end up with the equation

$$E[X_1 \boxplus \cdots \boxplus X_n] \quad = \quad E[X_1] + E\left[Y_2^{(2)}\right] + E\left[Y_3^{(3)}\right] + \cdots + E\left[Y_n^{(n)}\right]. \tag{5.7}$$

If we set $Y_i^{(1)} = X_i$ for $i = 1, \ldots, n$, then

$$E[X_1 \boxplus \cdots \boxplus X_n] = \sum_{i=1}^{n} E\left[Y_i^{(i)}\right].$$

We have hence expressed the mean value of the maximum of a set of positive random variables by the sum of the mean values of some other, related, random variables.

The derivation of Equation (5.7) has been performed in its most general form, but we are interested in the case where the random variables X_i, for $i = 1, \ldots, n$ are phase-type distributed. In this case, the interesting result is that the random variable $Y_i^{(i)}$ are phase-type distributed as well, and that the absorbing Markov chains describing the distributions of $Y_i^{(i)}$ are described by the original generator matrices $\mathbf{Q}_i$! The only difference between the distribution of X_i and $Y_i^{(i)}$ lie in the respective starting distributions!

To demonstrate this, we first do an example, before we continue with the formal derivation of the starting distribution of $Y_i^{(i)}$.

Example 5.15

As an example we consider three frogs on a lily pond[7]. The pond is covered with n lily pads, and each frog starts from pad i with probability $\pi_0(i)$. Each frog hops from pad to pad, where on each pad i it thinks for a negative exponentially distributed time about which pad to hop next. Frogs are not good at thinking, therefore, the next pad is actually chosen at random. The thinking time and the hopping probabilities depend only on the pad the frog sits on.

There is one pad that is actually a frog trap, probably installed by a representative of the french cuisine. Therefore, once a frog lands on the trap pad, it gets absorbed by a frog absorber.

We assume that the individual times of the frogs to reach the trap are random variables X_1, X_2 and X_3. All three random variables are in fact identical distributed and of phase-type: the lily pond with trap represents an absorbing Markov chain. The states of the absorbing Markov chain $\mathbf{Q}$ correspond to the lily pads. The probability vector $\underline{\pi}_0$ represents the starting distribution[8].

We assume that the frog hunt starts at time 0. We consider now the frogs 2 and 3 at the time instant X_1, *i.e.,* the time frog 1 is trapped. Now, we want to know how long

[7] This example is an adaption of an example given in [81].

[8] The reason why we consider all random variables as identically distributed is only to keep the example simple. We could also assume different lily ponds.

the frog trapper has still to wait until frogs 2 and 3 get trapped. A good guess would be $X_i - X_1$, for $i = 2, 3$. Nevertheless, we must be careful, because these differences may be negative — if frog 1 was the last to reach the trap, even all of them would be negative. It is hence necessary to define the residual life times of the other frogs as $Y_i^{(2)} = 0 \boxplus \frac{X_i}{X_1}$, for $i = 2, 3$. The remaining time until all the main ingredients for dinner are available is then $Y_2^{(2)} \boxplus Y_3^{(2)}$.

What are the distributions of $Y_2^{(2)}$ and $Y_3^{(2)}$? Since we assume that the lily pond has not changed and frogs 2 and 3 are still there, they have somehow changed the position since time 0. Nevertheless, the random variables $Y_2^{(2)}$ and $Y_3^{(2)}$ are still phase-type distributed with absorbing Markov chain $\mathbf{Q}$. Only the probabilities on which pad the remaining frogs sit when frog 1 is catched differ from the starting probabilities.

We now can ask, how long frog 3 still has to live if frog 1 *and* frog 2 have been trapped (without knowing the order of arrival). The time from the start is $X_1 \odot Y_2^{(2)} = X_1 \boxplus X_2$. Hence, the remaining life time of frog 3 is

$$Y_3^{(3)} = 0 \boxplus \frac{X_3}{X_1 \boxplus X_2} = 0 \boxplus \frac{X_3}{X_1 \odot Y_2^{(2)}}.$$

Again, the distribution of $Y_3^{(3)}$ is represented by $\mathbf{Q}$, the lily pond, but again the probability for frog 3 to be found on pad i has shifted somehow.

To conclude, the mean time until all three frogs are catched can be given by

$$E[X_1 \boxplus X_2 \boxplus X_3] = E[X_1] + E\left[Y_2^{(2)}\right] + E\left[Y_3^{(3)}\right]$$

In the next section we show how we can derive the starting probability distributions of the random variables $Y_2^{(2)}$ and $Y_3^{(3)}$.

5.5.2 Computing Residual Times

In this section we present a method to compute the probability distributions of an absorbing Markov chain under the assumption that a random, phase-type distributed amount of time has passed. We call this *the RT-Technique*.

We assume two phase-type distributed random variables A, B with representations $(\underline{\alpha}, \mathbf{A})$ for A and $(\underline{\beta}, \mathbf{B})$ for B. The associated generator matrices are denoted by $\dot{\mathbf{A}} = (a_{ij})_{k_a,k_a}$ and $\dot{\mathbf{B}} = (b_{ij})_{k_b,k_b}$, their starting distributions by $\dot{\underline{\alpha}}$ and $\dot{\underline{\beta}}$, respectively. The column vectors $\underline{\mathbf{A}}_0$ and $\underline{\mathbf{B}}_0$ are defined as

$$\underline{\mathbf{A}}_0 = -\mathbf{A}\underline{1} \qquad \text{and} \qquad \underline{\mathbf{B}}_0 = -\mathbf{B}\underline{1}.$$

The distribution function of A is given by

$$F_A(t) = 1 - \underline{\alpha} e^{\mathbf{A}t} \underline{1}.$$

The distribution function of B, F_B, is defined accordingly.

We want to compute $df(Z)$ (*cf.* Appendix B.2 for notation), where $Z = \frac{A \boxplus B}{A} = 0 \boxplus \frac{B}{A}$. Z is the *residual time* of B given A. How can we compute $df(Z)$? We can imagine that the probability mass which has been distributed over the states of the CTMC with generator $\dot{\mathbf{B}}$ according to $\dot{\beta}$ at time 0 has shifted somehow towards the absorbing state in the time period $[0, A)$. Hence, the residual time of B with given A is a phase-type distribution with generator matrix $\dot{\mathbf{B}}$ and a new starting distribution. This starting distribution is equal to the distribution $\underline{\dot{\beta}}(A) = \underline{\dot{\beta}} e^{\dot{\mathbf{B}}A}$. Note that $\underline{\dot{\beta}}(A)$ is a random variable itself, depending on A. To take all different outcomes of A into account, we have to decondition $\underline{\dot{\beta}}(A)$, *i.e.,* we have to compute $E\left[\underline{\dot{\beta}}(A)\right]$ in the following way:

$$\underline{\dot{\beta}}' = \int_0^\infty \underline{\dot{\beta}} e^{\dot{\mathbf{B}}t} dF_A(t). \tag{5.8}$$

However, $\underline{\dot{\beta}}'$ must be computed effectively, and doing this by means of Equation (5.8) is usually much too awkward. To compute $\underline{\dot{\beta}}'$ more elegantly, we use the same trick that is also used in Jensens method: *uniformisation* (*cf.* Section B.3.2). We do not use the generator matrix $\dot{\mathbf{B}}$ directly, but rather an associated stochastic matrix $\dot{\mathbf{P}}_B$, which is defined as

$$\dot{\mathbf{P}}_B = \frac{\dot{\mathbf{B}}}{b} + \mathbf{I},$$

where $b = \max\{|b_{ii}|\}$. Consequently, $\dot{\mathbf{B}} = b(\dot{\mathbf{P}}_B - \mathbf{I})$. Substituting this in Equation (5.8) yields

$$\begin{aligned}
\underline{\dot{\beta}}' &= \int_0^\infty \underline{\dot{\beta}} e^{b(\dot{\mathbf{P}}_B - \mathbf{I})t} dF_A(t) && (5.9)\\
&= \underline{\dot{\beta}} \int_0^\infty e^{-bt} e^{\dot{\mathbf{P}}_B bt} dF_A(t) && (5.10)\\
&= \underline{\dot{\beta}} \int_0^\infty e^{-bt} \sum_{n=0}^\infty \frac{(bt)^n}{n!} \dot{\mathbf{P}}_B^n dF_A(t) && (5.11)\\
&= \underline{\dot{\beta}} \sum_{n=0}^\infty \frac{b^n}{n!} \dot{\mathbf{P}}_B^n \int_0^\infty e^{-bt} t^n dF_A(t) && (5.12)\\
&= \underline{\dot{\beta}} \sum_{n=0}^\infty \frac{b^n}{n!} \dot{\mathbf{P}}_B^n E[e^{-bA} A^n] && (5.13)
\end{aligned}$$

One might wonder whether $E[\mathrm{e}^{-bA}A^n]$ is not as difficult to compute as $\dot{\beta}'$. But in fact, $E[\mathrm{e}^{-bA}A^n]$ can be computed quite easily if we employ uniformisation again. In the following derivation, we need the phase-type density of A, which, according to [115], is given as

$$f_A(t) = \underline{\alpha}\mathrm{e}^{\mathbf{A}t}\underline{\mathbf{A}}_0$$

Similar as before, we can define a matrix $\mathbf{P}_A$, such that

$$\mathbf{P}_A = \frac{\mathbf{A}}{a} + \mathbf{I},$$

where $a = \max\{|a_{ii}|\}$. This time, $\mathbf{P}_A$ is sub-stochastic since A is not a complete generator matrix. Therefore, the entries of $\mathbf{P}_A$ have values between 0 and 1, but the rows do not necessarily sum up to 1. However, $\mathbf{A} = a(\mathbf{P}_A - \mathbf{I})$, and we can derive

$$\begin{aligned}
E[\mathrm{e}^{-bA}A^n] &= \int_0^\infty \mathrm{e}^{-bt}t^n f_A(t)dt && (5.14)\\
&= \int_0^\infty \mathrm{e}^{-bt}t^n \underline{\alpha}\mathrm{e}^{\mathbf{A}t}\underline{\mathbf{A}}_0 dt && (5.15)\\
&= \underline{\alpha}\int_0^\infty \mathrm{e}^{-bt}t^n \mathrm{e}^{a(\mathbf{P}_A-\mathbf{I})t}\underline{\mathbf{A}}_0 dt && (5.16)\\
&= \underline{\alpha}\int_0^\infty \mathrm{e}^{-(b+a)t}t^n \mathrm{e}^{\mathbf{P}_A at}\underline{\mathbf{A}}_0 dt && (5.17)\\
&= \underline{\alpha}\int_0^\infty t^n \mathrm{e}^{-(b+a)t}\sum_{m=0}^{\infty}\frac{(at)^m}{m!}\mathbf{P}_A^m dt\underline{\mathbf{A}}_0 && (5.18)\\
&= \underline{\alpha}\sum_{m=0}^{\infty}\frac{a^m}{m!}\mathbf{P}_A^m\underline{\mathbf{A}}_0\int_0^\infty t^{n+m}\mathrm{e}^{-(b+a)t}dt && (5.19)
\end{aligned}$$

We then recognise that the integral in Equation (5.19) resembles the $(n+m)$th moment of an exponential distribution with rate $a+b$, *i.e.*, $\frac{(n+m)!}{(a+b)^{n+m}}$, so that we can rewrite Equation (5.19) as follows:

$$E[\mathrm{e}^{-bA}A^n] = \underline{\alpha}\sum_{m=0}^{\infty}\frac{a^m}{m!}\mathbf{P}_A^m\underline{\mathbf{A}}_0\frac{(n+m)!}{(a+b)^{m+n+1}} \quad (5.20)$$

Equation (5.20) is an infinite sum, but it can be brought in closed form. First, we reformulate Equation (5.20) as follows:

$$\begin{aligned}
&\underline{\alpha}\sum_{m=0}^{\infty}\frac{a^m}{m!}\mathbf{P}_A^m\underline{\mathbf{A}}_0\frac{(n+m)!}{(a+b)^{m+n+1}}\\
=\ &\underline{\alpha}\frac{1}{(a+b)^{n+1}}\left(\sum_{m=0}^{\infty}\left(\frac{a}{a+b}\mathbf{P}_A\right)^m\prod_{i=1}^{n}(m+i)\right)\underline{\mathbf{A}}_0
\end{aligned}$$

As can be shown, the sum now has the shape of the nth derivative of the geometric series. Since $\frac{a}{a+b}\mathbf{P}_A$ is sub-stochastic, the sum does converge and we can write

$$\begin{aligned}
& \underline{\alpha}\frac{n!}{(a+b)^{n+1}}\left(\sum_{m=0}^{\infty}\left(\frac{a}{a+b}\mathbf{P}_A\right)^m\prod_{i=1}^{n}(m+i)\right)\underline{\mathbf{A}}_0 \\
= \; & \underline{\alpha}\frac{n!}{(a+b)^{n+1}}\left(\mathbf{I}-\frac{a}{a+b}\mathbf{P}_A\right)^{-(n+1)}\underline{\mathbf{A}}_0 \\
= \; & n!\cdot\underline{\alpha}\left((a+b)\mathbf{I}-\mathbf{P}_A a\right)^{-(n+1)}\underline{\mathbf{A}}_0 \\
= \; & n!\cdot\underline{\alpha}\left(b\mathbf{I}-\mathbf{A}\right)^{-(n+1)}\underline{\mathbf{A}}_0 \qquad (5.21)
\end{aligned}$$

Now we have to combine Equation (5.13) and Equation (5.21) to compute the new starting distribution $\dot{\beta}'$:

$$\begin{aligned}
\dot{\beta}' &= \underline{\dot{\beta}}\sum_{n=0}^{\infty}\frac{b^n}{n!}\dot{\mathbf{P}}_B^n E[\mathrm{e}^{-bA}A^n] \\
&= \underline{\dot{\beta}}\sum_{n=0}^{\infty}\frac{b^n}{n!}\dot{\mathbf{P}}_B^n\underline{\alpha}\sum_{m=0}^{\infty}\frac{a^m}{m!}\mathbf{P}_A^m\underline{\mathbf{A}}_0\frac{(n+m)!}{(a+b)^{m+n+1}} \qquad (5.22) \\
&= \underline{\dot{\beta}}\sum_{n=0}^{\infty}\dot{\mathbf{P}}_B^n b^n\underline{\alpha}\left(b\mathbf{I}-\mathbf{A}\right)^{-(n+1)}\underline{\mathbf{A}}_0 \\
&= \underline{\dot{\beta}}\sum_{n=0}^{\infty}\dot{\mathbf{P}}_B^n b^n\underline{\alpha}\left(b\left(\mathbf{I}-\frac{1}{b}\mathbf{A}\right)\right)^{-(n+1)}\underline{\mathbf{A}}_0 \\
&= \frac{1}{b}\underline{\dot{\beta}}\sum_{n=0}^{\infty}\dot{\mathbf{P}}_B^n\underline{\alpha}\left(\mathbf{I}-\frac{1}{b}\mathbf{A}\right)^{-(n+1)}\underline{\mathbf{A}}_0 \qquad (5.23)
\end{aligned}$$

From Equation (5.22) as well as (5.23) algorithms can be developed which efficiently compute $\dot{\beta}'$. Nevertheless, careful considerations have to be made to guarantee convergence and numerical stability. This will be discussed in Section 5.5.3.

5.5.3 Numerical and Complexity Issues

Equations (5.22) and (5.23) propose different methods to compute $\underline{\dot{\beta}'}$. Which to choose depends on the numerical quality of the solution. Both ways may have their advantages or disadvantages. In this section we will shed light on this. We assume that the outer sum in (5.22) will be computed from $n=0$ to $n=N$ and the inner sum, for (5.22), from $m=0$ to $m=M$.

Equation (5.22): Double Summation

Using Equation (5.22), all employed variables have to be prevented from overflow. We define the following functions L and R recursively as follows:

$$L(n) = \begin{cases} \underline{\dot{\beta}}, & n = 0, \\ \dfrac{b}{n} L(n-1)\dot{\mathbf{P}}_B, & \text{otherwise}, \end{cases}$$

and

$$R(n,m) = \begin{cases} \dfrac{\underline{\alpha}}{(a+b)^{n+1}}, & m = 0, \\ \dfrac{a}{a+b}\dfrac{m+n}{m} R(n,m-1)\mathbf{P}_A, & \text{otherwise}. \end{cases}$$

Then (5.22) can be rewritten as:

$$\dot{\beta}' = \sum_{n=0}^{\infty} L(n) \sum_{m=0}^{\infty} R(n,m)\underline{\mathbf{A}}_0.$$

The computation of $L(n)$ as well as $R(n,m)$ is carried out on positive numbers only. This has great advantages from a numerical point of view, since cancellation errors, a frequent source of numerical instability introduced by subtraction, can not occur. A disadvantage is, of course, that only finite summations can be computed, and hence a truncation error is introduced. We assume in the following that the finite sum

$$\dot{\beta}' = \sum_{n=0}^{N} L(n) \sum_{m=0}^{M} R(n,m)\underline{\mathbf{A}}_0 \tag{5.24}$$

for $N, M \in \mathbb{N}$ is computed.

The complexity of this algorithm can be assessed as follows: $L(n)$ must be computed for $n = 0, \ldots, N$, *i.e.*, $N+1$ times. Since $L(n)$ is computed recursively from $L(n-1)$, only one vector-matrix multiplication is required. Hence, the computation of all $L(n)$ for $n = 0, \ldots, N$ requires $\mathcal{O}(Nk^2)$ operations. The computation of $R(n,m)$ is also done recursively and hence requires only one matrix-vector multiplication for each $(n,m) \in \{0, \ldots, N\} \times \{0, \ldots, M\}$. Hence, the number of operations is $\mathcal{O}(NMk^2)$ and consequently that of the whole computation of $\underline{\dot{\beta}'}$ of order $\mathcal{O}(NMk^2)$.

Equation (5.23): Explicit Inversion

The computation of $\underline{\dot{\beta}'}$ by means of Equation (5.23) involves an explicit matrix inversion. If $\mathbf{M}$ is a regular $k \times k$ matrix then $\mathbf{M}^{-1}$ is usually computed by solving k systems of linear

equations: since $\mathbf{M}\mathbf{M}^{-1} = \mathbf{I}_k$, one has to solve $\mathbf{M}\underline{m}_i = \underline{e}_i$ for $i = 1, \ldots, k$ to obtain the k column vectors $\underline{m}_i$ of $\mathbf{M}^{-1}$. In our case,

$$\mathbf{M} = \left(\mathbf{I} - \frac{\mathbf{A}}{b}\right). \tag{5.25}$$

Once the inverse of $\mathbf{M}$ is obtained, the inner sum of Equation (5.22) is replaced by a single matrix-vector multiplication. All computations of factorials and exponentials of scalars have reduced to a minimum. As done in the previous paragraph, we can define two recursive functions which facilitate the computation of $\underline{\dot{\beta}}'$:

$$E(n) = \begin{cases} \frac{1}{b}\underline{\dot{\beta}}, & n = 0, \\ E(n-1)\dot{\mathbf{P}}_B, & otherwise, \end{cases}$$

and

$$F(n) = \begin{cases} \underline{\alpha}\mathbf{M}^{-1}, & n = 0, \\ F(n-1)\mathbf{M}^{-1}, & otherwise. \end{cases}$$

Then

$$\underline{\dot{\beta}}' = \sum_{n=0}^{\infty} E(n)(F(n)\underline{\mathbf{A}}_0).$$

Note that the vector $\underline{\mathbf{A}}_0$ has to be multiplied to $F(n)$ for every iteration. $F(n)$ can not be combined with $E(n)$, since both matrices have generally different dimensions.

This approach requires an explicit matrix inversion. The complexity of the solution of a system of linear equations is $\mathcal{O}(k^3)$, hence, a matrix inversion requires $\mathcal{O}(k^4)$ steps. On the other hand, the explicit computation of $\dot{\beta}'$ requires only two matrix-vector-multiplications per iteration step for the computation of $E(n)$ and $F(n)$. Therefore, the computation of $\dot{\beta}'$ is of order $\mathcal{O}(Nk^2)$. Nevertheless, the overall complexity remains $\mathcal{O}(k^4)$.

Equation (5.23): Successive Solution of Systems of Linear Equations

Another way to compute $\underline{\dot{\beta}}'$ by means of Equation (5.23) is the following: for each iteration step n, the vector $\underline{\alpha}$ is multiplied with $\mathbf{M}^{-(n+1)}$ ($\mathbf{M}$ is defined as in (5.25)). For $n = 0$, we have to compute $\underline{x}^{(0)} = \underline{\alpha}\mathbf{M}^{-1}$, which is the solution of the system of linear equations

$$\underline{x}^{(0)}\mathbf{M} = \underline{\alpha}$$

Then, for $n \geq 1$ we must compute $\underline{x}^{(n)} = \underline{\alpha}\mathbf{M}^{-(n+1)} = \underline{x}^{(n-1)}\mathbf{M}^{-1}$, which is the solution of the system of linear equations

$$\underline{x}^{(n)}\mathbf{M} = \underline{x}^{(n-1)}.$$

Hence, each step in the summation of (5.23) requires the solution of one system of linear equations, and

$$\underline{\dot{\beta}}' = \sum_{n=0}^{\infty} E(n)(\underline{x}^{(n+1)}\underline{\mathbf{A}}_0).$$

The complexity of this approach is then of order $\mathcal{O}(Nk^3 + Nk^2) = \mathcal{O}(Nk^3)$.

Conclusions

All three approaches we have described have the same complexity. The three independent parameter that influence the behaviour of any algorithm based on the approaches so far are k, M, N. If we define $K = max\{k, M, N\}$, then the worst case complexity for all three approaches is of order $\mathcal{O}(K^4)$. Of course, this is a rough measure. The question which approach is the most efficient must be determined by comparison of real implementations. The complexity is no criterion to choose an approach.

From a numerical point of view, however, we can say a priori that probably an algorithm based on Equation (5.22) is most attractive, since computations are carried out on positive numbers only. This suggests stability since cancellation errors can not occur.

5.5.4 Comparative Complexity Considerations

In this section we compare the worst-case complexity of our approach to compute $E[X_1 \boxplus \cdots \boxplus X_n]$ with the worst-case complexity of the common approach.

The latter approach, which we will refer to as the *explicit approach*, is based on the expression $\underline{\alpha}\mathbf{T}^{-1}\underline{1}$, where $(\alpha, \mathbf{T})$ is the representation of the phase-type distribution of $X_1 \boxplus \cdots \boxplus X_n$.

We assume n different absorbing CTMCs with representations $(\underline{\alpha}_i, \mathbf{A}_i)$, which describe the describe the distributions of X_i, $i = 1, \ldots, n$. We assume that the generators have dimensions $k_i \times k_i$ and define $k = \max\{k_i\}$.

Explicit Approach. We can assume a constant e which is the mean number of non-zero entries in the matrices $\dot{\mathbf{A}}_i$ *per row*. Hence, matrix $\dot{\mathbf{A}}_i$ has about $e \cdot k_i$ non-zero entries. We assume that the storage requirements of the matrices $\dot{\mathbf{A}}_i$ can be neglected.

Let $\dot{\mathbf{A}} = \bigoplus_{i=1}^{n} \dot{\mathbf{A}}_i$ and $\underline{\dot{\alpha}} = \bigotimes_{i=1}^{n} \underline{\dot{\alpha}}_i$. Then $\dot{\mathbf{A}}$ has about $e \cdot n$ non-zero entries per row and hence the number of non-zeros of $\dot{\mathbf{A}}$ is of order $\mathcal{O}(k^n)$.

The mean value $m = E[X_1 \boxplus \cdots \boxplus X_n]$ of a random variable distributed according to $\dot{\mathbf{A}}$ can be computed as $m = -\underline{\alpha}\mathbf{A}^{-1}\underline{1}$. First a solution of $\underline{x}\mathbf{A} = \underline{\alpha}$ should be computed. Then $m = \underline{x}\,\underline{1}$. The computation of the mean value then reduces to the solution of a system of linear equations. If this is done numerically, for example by means of a Jacobi iteration, then one iteration step costs a number of operations that grows with the number

of non-zero elements of $\dot{\mathbf{A}}$, *i.e.,* with $\mathcal{O}(k^n)$. Usually the number of iterations is negligible compared to the number of states (in [7], Bell and Haverkort report a number of about 4000 iterations for a (sparse) system of of 750 million equations). Hence, although the solution process might require only $\mathcal{O}(nk)$ memory (by exploiting the Kronecker structure), the time it requires grows exponentially with the number of components.

The RT-Technique. The RT-Technique takes several steps:

1. Computing a mean value for all $\mathbf{A}_i$, *i.e.,* n times. If we assume complexity $\mathcal{O}(k_i^3)$ for case i, the overall effort is, in the worst case, $\mathcal{O}(nk^3)$.
2. Computing starting vectors according to either (5.22) or (5.23). This has to be done $\frac{n(n-1)}{2}$ times, *i.e.,* $\mathcal{O}(n^2)$.
3. From the considerations of Section 5.5.3 we see that the computation of the vector $\underline{\dot{\beta}'}$ requires about $\mathcal{O}(K^4)$ steps, for $K = \max\{N, M, k\}$. Which one to choose is then a numerical, not a complexity issue.

The overall complexity of our approach is consequently of order $\mathcal{O}(nK^3+n^2K^4) = \mathcal{O}(n^2K^4)$.

Conclusions. We see that the number of operations to compute the mean value m by means of the RT-Technique is polynomial, whereas the effort to compute *even one* iteration step by means of the explicit approach is exponential in the number of the random variables. The memory consumption is to neglect for both approaches since it is never necessary to construct the global state space.

Note that the complexity bound $\mathcal{O}(n^2K^4)$ describes the worst case. For example, we have always assumed that a matrix-vector multiplication requires $\mathcal{O}(k^2)$ steps. This is certainly true for dense matrices, but if the used generator matrices happen to be sparse, the effort for a matrix-vector multiplication depends linearly on the number of non-zero entries of the matrices, which is of order $\mathcal{O}(k)$. It is not uncommon that generator matrices are in fact sparse, so frequently the number of operations needed by the RT-Technique can be bounded more tightly by a function of order $\mathcal{O}(n^2K^3)$, at least, if no explicit matrix inversions must be computed.

5.5.5 Summary

The RT-Technique allows to derive the mean value of the maximum of phase-type distributed random variables. The basic approach is to express the mean value of the maximum of the random variable $X_1, \ldots, X_n$, *i.e.,* $E[X_1 \boxplus \cdots \boxplus X_n]$, as a sum of mean values of random variables $Y_1, \ldots, Y_n$, *i.e.,*

$$E[X_1 \boxplus \cdots \boxplus X_n] = E[Y_1] + \cdots + E[Y_n].$$

We have seen that the random variables Y_i can be represented by the same absorbing Markov chain as X_i for $i = 1, \ldots, n$. Only the starting distribution corresponding to Y_i is different to that of X_i.

The RT-Technique provides the efficient means to derive the appropriate starting distributions for the random variables Y_i for $i = 1, \ldots, n$. The actual mean value of the random variables Y_i can then be computed by standard measures. The method is also numerically stable if an algorithm is used that is based on Uniformisation: cancellation errors are then avoided.

The complexity to derive the starting distributions of the random variables Y_i is polynomial in the number of components and the size of considered Markov chains. We can conclude that the RT-Technique is the main reason for a considerable efficiency improvement of the **AWCI**-Technique, as introduced in the previous sections of this chapter.

5.6 Waiting Times

In this section we show that we even can compute waiting times for **AWCI**-processes.

5.6.1 Introduction

In this section, we show which means are required to compute waiting times for **AWCI**-processes. We do this along the lines of a running example.

Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of components (indices indicating the locations) with the requirements **A**, **WC**, and **I** with respect to a synchronisation set S. Let $R \stackrel{\text{def}}{=} !_S\mathcal{P}$. Let $\text{SMC}(P_i) = (\Sigma_i, \mathbf{E}_i, J_i)$ and $\text{SMC}(R) = (\Sigma, \mathbf{E}, J)$. We consider a synchronising state $G = L_1 \|_S L_2 \|_S \cdots \|_S L_n \in \mathcal{S}_{syn}(!_S\mathcal{P})$. The time that passes between a visit of G and one of the next reachable synchronising states is described by $J(G) = \bigoplus_{i=1}^{n} J(L_i)$. Let X_i for $i = 1, \ldots, n$ be random variables distributed according to $J(L_i)$. Then, the mean value for the distribution $J(G)$ is $E[X_1 \boxplus \cdots \boxplus X_n]$. If we assume that component i becomes ready to synchronise, then the time until the synchronisation happens can be expressed by

$$\frac{X_1 \boxplus \cdots \boxplus X_n}{X_i}.$$

We define the mean value of this random variable as $M_i(G) = E[X_1 \boxplus \cdots \boxplus X_n] - E[X_i]$. $M_i(G)$ is the mean waiting time of component i and can be computed efficiently by the RT-Technique proposed in Section 5.5.

We now observe that $M_i(G)$ is a quantity that relates to location i, but it is *not* a quantity that relates to the local sub-process L_i of G. The reason is that the global synchronising state G denotes the *beginning* of a time period from which all components work on their own. $M_i(G)$ then is the mean time that all locations except i still work on their own,

whereas location i has already reached *a local synchronising state that succeeds* L_i and waits *there*. That means that $M_i(G)$ is a quantity that relates to the successors of L_i, not to L_i itself. Moreover, $M_i(G)$ is only *one* measure, even though L_i can have more than one successor state. That means that in $M_i(G)$ the information for all successor states of L_i about their respective mean waiting times is summarised.

What we learn from this is that, if we want to obtain waiting times for a local synchronising state L_i, we must consider synchronising predecessor states, not successor states of L_i. Moreover, the analysis method must be fine enough to obtain measures for individual states, not only measures that summarise information of several states.

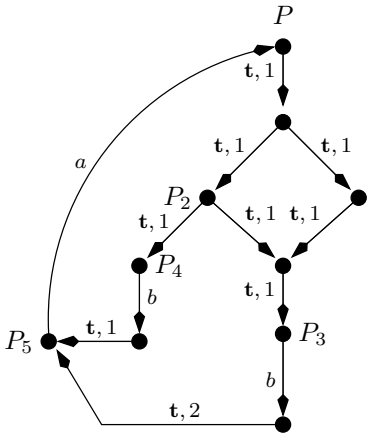
Obviously, it is necessary to have a clear notion of the reachability between synchronising states of the considered process. Fortunately, this information is already explicitly expressed in the EMCs $\mathbf{E}$ and $\mathbf{E}_i$ for $i = 1, 2, \ldots, n$ of the considered **AWCI**-process. In the following we say that a synchronising state L_i of component i can reach a synchronising state L_i' (in the same component), if the probability $e^{(i)}(L_i, L_i') > 0$. The same should hold for global synchronising states and $\mathbf{E}$.

We illustrate this by means of the next example.

Example 5.16

We assume a process P, which is defined as follows:

$$
\begin{aligned}
P &\stackrel{\text{def}}{=} [1].([1].P_2 + [1].[1].[1].P_3),\\
P_2 &\stackrel{\text{def}}{=} [1].P_4 + [1].[1].P_3,\\
P_3 &\stackrel{\text{def}}{=} b.[2].P_5,\\
P_4 &\stackrel{\text{def}}{=} b.[1].P_5,\\
P_5 &\stackrel{\text{def}}{=} a.P.
\end{aligned}
$$



The transition system of P is depicted in the right diagram. The process $R \stackrel{\text{def}}{=} P\|_{\{a,b\}}P$ has two locations and is an **AWCI**-process. The set of global synchronising states of $[\![R]\!]$ is

$$\mathcal{S}_{syn}(R) = \left\{\underline{P_3\|_{\{a,b\}}P_3},\ \underline{P_3\|_{\{a,b\}}P_4},\ \underline{P_4\|_{\{a,b\}}P_3},\ \underline{P_4\|_{\{a,b\}}P_4},\ \underline{P_5\|_{\{a,b\}}P_5}\right\}.$$

The elements of $\mathcal{S}_{syn}(R)$ are underlined for better readability.

The embedded Markov chain $\mathbf{E}$ of SMC(R) is depicted in Figure 5.3. As we easily can see by the definition of P_3, P_4, and P_5, there is only one synchronising state representing a synchronisation over action a ($\underline{P_5\|_{\{a,b\}}P_5}$), and four states representing a synchronisation over action b (all the others).

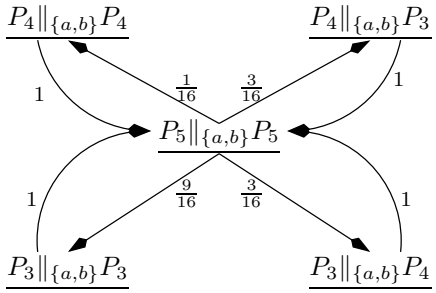
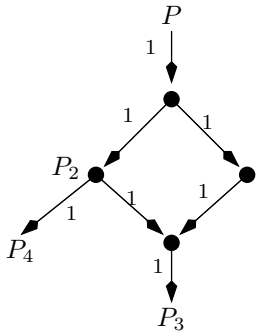


Figure 5.3: EMC of R (Example 5.16)

The absorbing Markov chain that describes the distribution of the time to reach a synchronising successor state of state P is depicted in Figure 5.4. We now want to



$$\mathbf{Q} = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 5.4: Absorbing Markov chain $\mathbf{Q}$ for P

compute the mean waiting time for component 1 of the state $G \stackrel{\text{def}}{=} P_5 \|_{\{a,b\}} P_5$, *i.e.*, the quantity $M_1(G)$. If X_1, X_2 are two random variables that both are distributed according to $\mathbf{Q}$, then $M_1(G) = E[X_1 \boxplus X_2] - E[X_1]$. Numerically, we derive $M_1(G) = \frac{15}{16}$.

The quantity $M_1(G)$ is the mean waiting time of location 1 under the assumption that it waits either in local state P_3 or local state P_4. This information, however, is not sufficient. We want to know the mean waiting time under the assumption that we know in which state component 1 does wait. We denote these quantities as $M^G_{P_5,P_3}$ and $M^G_{P_5,P_4}$, respectively. We define the probabilities for component 1 to wait either in state P_3 or P_4 as π_3 and π_4, respectively. Then, $\pi_4 = \frac{1}{4}$, since there is only one path leading to P_4 from P_5, and the probability for this path to be chosen is $\frac{1}{4}$. Consequently, $\pi_3 = 1 - \pi_4 = \frac{3}{4}$. A first guess for $M^G_{P_5,P_3}$ and $M^G_{P_5,P_4}$ would be to weight the overall mean waiting time with the probabilities π_3 and π_4, *i.e.*, to set $M^G_{P_5,P_3} = \pi_3 M_1(G) = \frac{45}{64}$ and $M^G_{P_5,P_3} = \pi_4 M_1(G) = \frac{15}{64}$.

These results are incorrect, unfortunately. Instead, we must define two new absorbing Markov chains, one describing the case that P_3 is reached from P and the other

describing the case that P_4 is reached. These two Markov chains are depicted in Figure 5.5. The left one (denoted $\mathbf{Q}_3$) describes the phase-type distribution of the

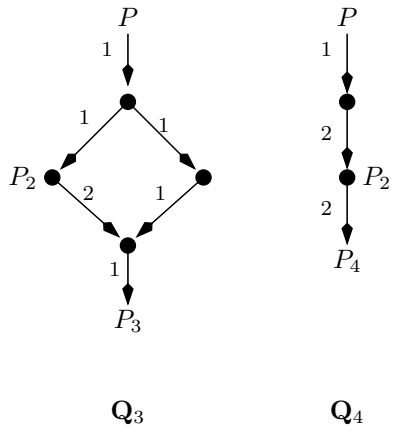


Figure 5.5: The CTMCs for P_3 and P_4 (Example 5.16)

time that is needed to reach state P_3 from P *under the assumption that we know* that P_3 (and not P_4) is reached. The right one (denoted $\mathbf{Q}_4$) describes the same for P_4. Note that for some transitions the assigned rates have been adapted such that the mean sojourn times of the respective source states are the same as in the original Markov chain, $\mathbf{Q}$. If Y_3 is a random variable distributed according to $\mathbf{Q}_3$ and Y_4 is a random variable distributed according to $\mathbf{Q}_4$, then the mean waiting time for state P_3 can be expressed as $E[Y_3 \boxplus X_2] - E[Y_3]$, and that for P_4 as $E[Y_4 \boxplus X_2] - E[Y_4]$. This means, that

$$M^G_{P_5,P_3} = (E[Y_3 \boxplus X_2] - E[Y_3])$$

and

$$M^G_{P_5,P_4} = (E[Y_4 \boxplus X_2] - E[Y_4]).$$

Computing this numerically, we obtain

$$M^G_{P_5,P_3} = \frac{85}{108}$$

and

$$M^G_{P_5,P_4} = \frac{25}{18}.$$

If we take the sum of these both quantities, weighted with π_3 and π_4, we obtain

$$\pi_3 M^G_{P_5,P_3} + \pi_4 M^G_{P_5,P_4} = \frac{15}{16} = M_1(L).$$

What we have learned from Example 5.16 is that for each local synchronising state L_i of a

component i, for all its preceding local synchronising state L_i' an absorbing Markov chain has to be constructed from $J_i(L_i')$.

This conclusion, however, raises another problem. It is not necessarily the case that a local synchronising state L_i has a unique predecessor. That implies that a global synchronising state G with sub-process $\Downarrow_i(G) = L_i$ also has no unique predecessor. The waiting times for component i depend then on the predecessor state from which the considered state G has been entered. Therefore, we have to take all different scenarios that can happen into account. We illustrate this with the next example.

Example 5.17

We reconsider the processes P and R from Example 5.16. The local synchronising state P_5 of $[\![P]\!]$ has two synchronising predecessors: P_3 and P_4. As indicated by the EMC of R in Figure 5.3, there are four different possibilities to reach the synchronising state $R' \stackrel{\text{def}}{=} P_5 \|_{\{a,b\}} P_5$ (namely $\underline{P_3 \|_{\{a,b\}} P_3}$, $\underline{P_3 \|_{\{a,b\}} P_4}$, $\underline{P_4 \|_{\{a,b\}} P_3}$, and $\underline{P_4 \|_{\{a,b\}} P_4}$). Each of these four states represent a different situation for component 1 to reach the local synchronising state P_5, and consequently, they all have to be taken into account.

To formulate this differently, if component 1 enters state P_5 and begins to wait for component 2 to synchronise, there are four possible different synchronisations that could have preceded the one that has to come and that component 1 is waiting for.

For all of these four situations we can derive waiting times for component 1. If we consider the case that $P_3 \|_{\{a,b\}} P_3$ is representing the preceding synchronisation, both components have to wait an exponentially distributed time period with rate 1 until they reach state P_5. The mean time until the next synchronisation is therefore $E[X_1 \boxplus X_1']$, where X_1 and X_1' are independent exponentially distributed random variables with rate 1. The waiting time of component 1 for this considered case is therefore $W_{3,3} = E[X_1 \boxplus X_1'] - E[X_1] = \frac{1}{2}$.

If we consider the case that $P_4 \|_{\{a,b\}} P_4$ is representing the preceding synchronisation, the mean time until the next synchronisation can be expressed by $E[X_2 \boxplus X_2']$, where X_2, X_2' are independent, exponentially distributed random variables with rate 2. The waiting time for this case is $W_{4,4} = E[X_2 \boxplus X_2'] - E[X_2] = \frac{1}{4}$.

The waiting times for the other cases can be derived accordingly and have the values $W_{3,4} = \frac{1}{6}$ and $W_{4,3} = \frac{2}{3}$.

For each synchronising predecessor G' of a global synchronising state G the mean waiting time $W^L_{G',G}$ of a component i with $\Downarrow_i(G) = L$ can be derived. However, we want to have a value for the mean waiting time that is independent of the synchronisation that happened last. This can be done by summing the values $W^L_{G',G}$ for each predecessor G', each weighted with the probability that G' was indeed the predecessor of G.

Example 5.18

We continue Example 5.17. Since the states of the EMC $\mathbf{E}$ correspond directly to the synchronising states of $[\![R]\!]$, we can see immediately in Figure 5.3 that the synchronising states preceding state $P_5 \|_{\{a,b\}} P_5$ are

$$\underline{P_3 \|_{\{a,b\}} P_3},\ \underline{P_3 \|_{\{a,b\}} P_4},\ \underline{P_4 \|_{\{a,b\}} P_3},\ \text{and}\ \underline{P_4 \|_{\{a,b\}} P_4}.$$

We must obtain for each of the preceding states the probability that, under the assumption that we just have entered $P_5 \|_{\{a,b\}} P_5$, we came from this specific preceding state.

The question which synchronising state can reach another one in one step can be answered by means of the EMC of the considered process. We show now that we can also derive the desired probabilities from this EMC.

We state the problem in a more abstract way. We assume an irreducible DTMC with probability matrix $\mathbf{D} = (d_{ij})_{n,n}$ and steady-state probability vector $\underline{p}$. If we assume steady-state, and if we assume that we are in a certain state $j \in \{1, \ldots, n\}$: what is the probability that state j was entered from state $i \neq j$?

Lemma 5.19 Let $\mathbf{D} = (d_{ij})_{n,n}$ a stochastic matrix describing an irreducible DTMC $\{X_k, k = 0, 1, 2, 3, \ldots\}$, and let $\underline{p}$ be its steady-state probability vector. Let i, j for $i \neq j$ be two states. If we assume that we are in state i, then the steady-state probability that the previous state was j is given by

$$b_{ij} = \frac{p_j \cdot d_{ji}}{p_i}$$

Proof: We are looking for the probability

$$b_{ij} = \Pr(\textit{previous state was } j \mid \textit{current state is } i)$$

We can express this conditional probability as

$$b_{ij} \quad = \quad \frac{\Pr(\textit{previous state was } j \textit{ and current state is } i)}{\Pr(\textit{current state is } i)}$$

From the denominator we know that in steady-state

$$\Pr(\textit{current state is } i) \quad = \quad p_i.$$

The numerator, on the other hand, can be expressed as

$$\Pr(\textit{previous state was } j \textit{ and current state is } i) \quad = \quad p_j d_{ji}.$$

Combining the two equations yields

$$b_{ij} = \frac{p_j \cdot d_{ji}}{p_i}.$$

→●

Note that we can (and, of course, will) use this lemma also for semi-Markov chains: to derive the probabilities b_{ij} for an SMC, it is enough to consider only the EMC (which is a DTMC) and its steady-state distribution, rather than the steady-state distribution of the whole SMC. The reason for this is that it is not important how *long* a state is occupied; only the *frequency* with which a state is entered, compared with the other states, is important. This information is provided by the steady-state probability distribution of the EMC, which, as we know, is a normalised vector of visit counts.

Example 5.20

We continue Example 5.18. We number the states of $\mathbf{E}$ such that

$$\begin{aligned} P_5 \|_{\{a,b\}} P_5 &\;\hat{=}\; 1 \\ P_3 \|_{\{a,b\}} P_3 &\;\hat{=}\; 2 \\ P_3 \|_{\{a,b\}} P_4 &\;\hat{=}\; 3 \\ P_4 \|_{\{a,b\}} P_3 &\;\hat{=}\; 4 \\ P_4 \|_{\{a,b\}} P_4 &\;\hat{=}\; 5 \end{aligned}$$

The probability matrix has the form

$$\mathbf{E} = \begin{pmatrix} 0 & \frac{9}{16} & \frac{3}{16} & \frac{3}{16} & \frac{1}{16} \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Then the steady-state probabilities of $\mathbf{E}$ are equal to $\underline{\pi} = \left(\frac{1}{2}, \frac{9}{32}, \frac{3}{32}, \frac{3}{32}, \frac{1}{32}\right)$.

For state 1, the probabilities $b_{1,j}$ for $j = 2, 3, 4, 5$, computed according to Lemma 5.19, yield

$$\begin{aligned} b_{1,2} &= \frac{9}{16}, \quad b_{1,3} = \frac{3}{16}, \\ b_{1,4} &= \frac{3}{16}, \quad b_{1,5} = \frac{1}{16}. \end{aligned}$$

We are now ready to compute the mean waiting time of a local synchronising state, independent from which synchronising predecessor state it has been entered.

5.6.2 An Algorithm for the Waiting Times

In this section, we summarise the considerations of the previous section and define an algorithm to compute the waiting times for **AWCI**-processes.

We assume a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of components which fulfils the requirements **A**, **WC**, and **I** with respect to a synchronisation set S. Let $R = !_S\mathcal{P}$, $\mathrm{SMC}(P_i) = (\Sigma_i, \mathbf{E}_i = (e_{kl}^{(i)})_{m_i,m_i}, J_i)$, where $m_i = \#\Sigma_i$, and $\mathrm{SMC}(R) = (\Sigma, \mathbf{E} = (e_{kl})_{m,m}, J)$, where $m = \#\Sigma$. The sets Σ and Σ_i, for $i = 1, \ldots, n$, contain by definition the synchronising states of $[\![R]\!]$ and $[\![P_i]\!]$, respectively. If $s, s' \in \Sigma_i$ correspond to the kth and lth row of $\mathbf{E}_i$, then we define $e^{(i)}(s, s') = e_{kl}^{(i)}$. Let $\underline{p}^{(i)}$ be the steady-state probability vector of $\mathbf{E}_i$ and $\underline{p}$ that of $\mathbf{E}$. Similarly as before, for $s \in \Sigma_i$, we define $p^{(i)}(s) = p_k^{(i)}$, if the kth row of $\mathbf{E}_i$ corresponds to s.

For the EMC $\mathbf{E}$, we define $\mathbf{B}$ to be the matrix $(b_{k,l})_{m,m}$ of probabilities which are defined as

$$b_{kl} = \frac{p_l e_{lk}}{p_k}$$

for $k, l \in \{1, \ldots, n_i\}$. Note that the probabilities b_{kl} are those that we have derived in Lemma 5.19.

If we assume the above definitions and notations to be global variables of some program, then Algorithm 5.1 on page 129 computes the waiting time for a given local synchronising state of a given location. In the following, we comment on this algorithm.

Input and Output: The algorithm computes the waiting time of the local synchronising state L of component i.

Line 1. $\mathit{Glob}(L)$ is the set of all *global* synchronising states $G \in \mathcal{S}_{syn}(P)$ for which $\Downarrow_i(G) = L$ holds.

Line 2. Outer loop: computations take place for all global synchronising states $G \in \mathit{Glob}(L)$. Each iteration of this loop computes a value W_G^L (see Line 11).

Line 3. The set $\mathit{Pred}(G)$ contains all global synchronising states P which, according to the EMC $\mathbf{E}$, can reach G in one step.

Line 4. Inner loop: computation take place for all states $P \in \mathit{Pred}(G)$. Each iteration of this loop computes a value $W_{P,G}^L$.

Lines 5–7. Here random variables $X_1, \ldots, X_n$, except X_i, are defined, where X_k is distributed according to $J_k(\Downarrow_i(P))$ for $k \neq i$.

Line 8 X_i is distributed according to $J_i(\Downarrow_i(P, L))$, the conditional distribution that assumes that an absorbing state of $J_i(\Downarrow_i(P))$ is reached that corresponds to L.

Line 9. Here, $W^L_{P,G}$ is computed, the mean value $E[\max_{j=1,\ldots,n}\{X_j\}] - E[X_i]$. This can be seen as a procedure call to the RT-Technique of Section 5.5.

Line 10. End of inner loop.

Line 11. Outer loop again. Here, the values $W^L_{P,G}$, computed in the inner loop, are combined in a weighted sum, where each value $W^L_{P,G}$ is weighted with the probability $b_{P,G}$. The result is, as mentioned above, W^L_G. This is the mean value of the waiting time of L under the assumption that the waiting time ends by a synchronisation that is represented by state G.

Line 12. End of outer loop.

Line 13. W_L is the final result computed by this algorithm. It is a sum that combines all values W^L_G for all $G \in \mathit{Glob}(L)$, weighted with the EMC steady-state probability of state G.

5.6.3 Importing the Waiting Times

Let us assume now that $G_i = (S_i, A_i, T_i, \mathcal{R}_i, P_i) = [\![P_i]\!]$ for $i \in \{1, \ldots, n\}$ and that we have computed all waiting times W^L for $L \in \mathcal{S}_{syn}(P_i)$. Then we define the GMP $G'_i = (S'_i, A'_i, T'_i, \mathcal{R}_i, P'_i)$ as follows:

1. $S'_i = S_i$, $A'_i = A_i$, $T'_i = T_i$, and $P'_i = P_i$;
2. $\mathcal{R}'_i$ is defined as follows: for $t \in T'$,

$$\mathcal{R}'_i(t) = \begin{cases} \dfrac{1}{W^L}, & \text{if } L \in \mathcal{S}_{syn}(P_i) \text{ and } \mathit{src}(t) = L; \\ \mathcal{R}_i(t), & \text{otherwise.} \end{cases}$$

We assign therefore a rate to each synchronising transition with source state L that determines that the sojourn time of L is W^L. Note that the GMP G' is not properly timed anymore (*cf.* Section 3.1.2), since now also synchronising transitions have a rate assigned. However, there is no need to be worried about this since all we want to do with P' is to derive a steady-state distribution for it. Since all transition of G' now have a rate associated, a generator matrix can be derived most easily and the system of the global balance equations can be solved with standard methods. The result is the local steady-state distribution for component i.

· Input: a location i and a synchronising state L of location i.

· Output: mean waiting time W^L of L.

1. $\mathit{Glob}(L) := \{G \mid G \in \mathcal{S}_{syn}(P) \text{ and } \Downarrow_i(G) = L\}$
2. **forall** $G \in \mathit{Glob}(L)$ **do**
3. $\quad \mathit{Pred}(G) := \{P \mid P \text{ is a predecessor of } G \text{ in } \mathbf{E}\}$
4. $\quad$ **forall** $P \in \mathit{Glob}(L)$ **do**
5. $\quad\quad$ **forall** $k \in \{1, \ldots, n\}, k \neq i$ **do**
6. $\quad\quad\quad X_k \sim J_k\big(\Downarrow_k(P)\big)$
7. $\quad\quad$ **end**
8. $\quad\quad X_i \sim J_i\big(\Downarrow_i(P), L\big)$
9. $\quad\quad W^L_{P,G} := E[\max_{j \neq i}\{X_j\}] - E[X_i]$
10. $\quad$ **end**
11. $\quad W^L_G := \sum_{P \in \mathit{Pred}(G)} b_{P,G} W^L_{P,G}$
12. **end**
13. $W^L = \sum_{G \in \mathit{Glob}(L)} p(G) W^L_G$

Algorithm 5.1: The **WT** algorithm

5.7 Application Example

In this section, we present an application example. We will not give numerical results for this example, but rather illustrate that the class of processes that is suitable to be analysed by the **AWCI**-Technique, is large enough to contain meaningful examples.

5.7.1 The Model

The application we have chosen originates from the area of parallel processing. The system addressed is a master/slave configuration, in which a master prepares the slaves to compute tasks in parallel and to return the results to the master eventually.

The set of visible actions is defined as

$$Com := \{\mathsf{split}, \mathsf{join}, \mathsf{reqhelp}, \mathsf{ready}\}.$$

S and R are auxiliary sets and defined as $S := \{\mathsf{split}, \mathsf{join}\}$ and $R := Com \setminus S$. The other action identifiers have to be seen as variables for rates. Then, the master/slave system *System* is defined as

$$\begin{aligned}
System &\stackrel{\text{def}}{=} Master \parallel_S (SlaveSystem_1 \parallel_S \cdots \parallel_S SlaveSystem_n) \\
Master &\stackrel{\text{def}}{=} \mathsf{split}.[\mathsf{garbcol}].\mathsf{join}.[\mathsf{newjobs}].Master \\
SlaveSystem &\stackrel{\text{def}}{=} (Slave \parallel_R Helper \parallel_R Helper\ ) \setminus R \\
Slave &\stackrel{\text{def}}{=} \mathsf{split}.[\mathsf{working}].\mathsf{reqhelp}.SlaveWait \\
SlaveWait &\stackrel{\text{def}}{=} \mathsf{ready}.[\mathsf{working}].\mathsf{join}.Slave \\
Helper &\stackrel{\text{def}}{=} \mathsf{reqhelp}.\big(\ [\mathsf{goodmood}].[\mathsf{fasthelp}].\mathsf{ready}.Helper \\
&\qquad\qquad + [\mathsf{badmood}].[\mathsf{slowhelp}].\mathsf{ready}.Helper\big).
\end{aligned}$$

The master *Master* submits jobs to the slave systems $SlaveSystem_i$ $(i = 1, \ldots, n)$ and indicates with action split that the slaves should begin to work.[9] It then passes the waiting time with garbage collecting. After that, it waits on action join for the slaves. When all slaves have indicated that they have finished, the master generates new jobs for the slaves (newjobs). A slave system $SlaveSystem_i$ consists of three components, *Slave* and two instances of *Helper*. *Slave* waits for the master to submit a job on action split. It then starts working on the job (working) and eventually requests help from the helpers (action reqhelp). The helpers eventually return control to *Slave* (action ready) and the slave proceeds to work. Finally, the slave indicates the master that it is ready (action join) and starts from the beginning. The helpers accept requests from the slave *Slave*. The decision whether a helper works quickly (fasthelp) or slowly (slowhelp) depends on the

[9] For the sake of simplicity, we assume that the slave components $SlaveSystem_i$ are all identical to the process *SlaveSystem* defined above. Of course, this is not generally necessary.

stochastic choice between badmood and goodmood. After a helper has returned control to the slave, it starts from the beginning.

This example shows that the components which are used for the subsequent solution approach must not necessarily be sequential. Thanks to the expansion law, they can be the result of the parallel composition of smaller components, although that compositional structure can not be exploited by the **AWCI**-Technique. The modeller, however, has more modelling freedom than the three properties **A, WC**, and **I** might impose.

In the following, we will consider the processes $SlaveSystem_i$ (for $i = 1, 2$) and *Master* as the components for our decomposition approach. The flow of control for this system is shown in the sequence chart in Figure 5.6. The vertical arrow compounds are the threads the system comprises. A solid arrow denotes that the thread is active, a dotted line denotes that the thread is waiting. Dashed horizontal lines denote synchronisation instances (labelled with the respective synchronising actions).

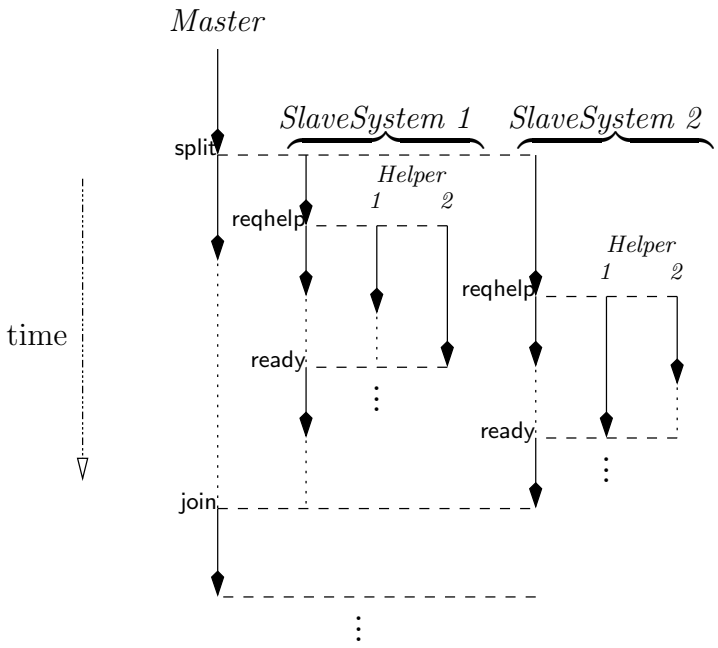


Figure 5.6: Sequence chart of Master/Slave system

5.7.2 Checking for A, WC and I

Before we can do anything, we must make sure that the set

$$\mathcal{C} = \{SlaveSystem_1, SlaveSystem_2, Master\}$$

has the properties **A, WC** and **I**. To do so, we have to check the $\mathcal{YAWN}$ processes $\mathit{SlaveSystem} \setminus \overline{S}$ and $\mathit{Master} \setminus \overline{S}$.

We start with property **I**. To ensure the irreducibility and s-determinism of the components, we normally have to perform a reachability analysis of all of them. In case of this example, however, a sharp look at the definitions of *Master* and *SlaveSystem* is sufficient to assure us of the irreducibility of these components.

It is obvious that *SlaveSystem* contains transitions with label **i**: the transitions that were formerly labelled with reqhelp and ready and which have been hidden now. However, the transitions can be eliminated, since they do not lead to nondeterminism and are also not in choice with other transitions. We conclude that condition **I** is met by *Master* as well as by *SlaveSystem*.

Since no choices between timed and untimed actions do occur, the processes are alternating. Therefore, property **A** is also fulfilled with respect to $\mathcal{C}$.

Now we must check whether property **WC** holds for $\mathcal{C}$. The *Master* is very simple, hence we can agree immediately that

$$\mathit{Master} \setminus \overline{S} \quad \simeq \quad \mathit{recX} : \mathsf{split.join}.X.$$

Now we must show that also $\mathit{SlaveSystem} \setminus \overline{S} \simeq \quad \mathit{recX} : \mathsf{split.join}.X$. Looking sharply at the definition of *SlaveSystem* for a while reveals that the only visible actions are split and join(as already recognised above) and that there is only one synchronising state for each action. Since *SlaveSystem* is irreducible, we can conclude that both states are mutually reachable and hence indeed

$$\mathit{SlaveSystem} \setminus \overline{S} \quad \simeq \quad \mathit{recX} : \mathsf{split.join}.X.$$

As a consequence, $\mathit{SlaveSystem} \setminus \overline{S} \simeq \mathit{Master} \setminus \overline{S}$, so condition **WC** is fulfilled by set $\mathcal{C}$.

5.7.3 Derivation of the Local SMCs

In this section we derive $\mathrm{SMC}(\mathit{Master}) = (\Sigma_M, \mathbf{E}_M, J_M)$ and $\mathrm{SMC}(\mathit{SlaveSystem}) = (\Sigma_S, \mathbf{E}_S, J_S)$. The state space Σ_M of $\mathrm{SMC}(\mathit{Master})$ is

$$\Sigma_M = \left\{ \underbrace{\mathit{Master}}_{m_1}, \underbrace{\mathsf{join}.[\mathsf{newjobs}].\mathit{Master}}_{m_2} \right\},$$

where the states are renamed m_1 and m_2 for easier reference.

The state space of $\mathrm{SMC}(\mathit{SlaveSystem})$ is

$$\Sigma_S = \left\{ \underbrace{(\mathit{Slave} \parallel_R \mathit{Helper} \parallel_R \mathit{Helper}) \setminus R}_{s_1}, \underbrace{(\mathsf{join}.\mathit{Slave} \parallel_R \mathit{Helper} \parallel_R \mathit{Helper}) \setminus R}_{s_2} \right\},$$

where the states are renamed s_1 and s_2.

As can easily be seen, the EMCs of *Master* as well as for *SlaveSystem* have only two states. The EMCs have the probability matrix

$$\mathbf{E}_M = \mathbf{E}_S = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \tag{5.26}$$

For *Master*,

$$J_M(m_1) = \begin{pmatrix} -\mathsf{garbcol} & \mathsf{garbcol} \\ 0 & 0 \end{pmatrix} \qquad J_M(m_2) = \begin{pmatrix} -\mathsf{newjobs} & \mathsf{newjobs} \\ 0 & 0 \end{pmatrix}$$

The distributions for *SlaveSystem* are slightly more complicated. $J_S(s_1)$ is an 18×18 matrix. If we abbreviate rate $\mathsf{working}$ with w, $\mathsf{goodmood}$ with g, $\mathsf{badmood}$ with b, $\mathsf{fasthelp}$ with f and $\mathsf{slowhelp}$ with s, then we define in Figure 5.7 the auxiliary matrix $\mathbf{S}$ (where

$$\mathbf{S} = \begin{pmatrix}
w & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & g & b & \cdot & g & \cdot & \cdot & \cdot & b & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & f & \cdot & g & \cdot & \cdot & \cdot & b & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & s & \cdot & \cdot & g & \cdot & \cdot & \cdot & b & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & g & b & \cdot & g & \cdot & \cdot & \cdot & b & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & f & \cdot & \cdot & \cdot & \cdot & \cdot & f & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & s & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & f & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & g & b & \cdot & \cdot & \cdot & \cdot & \cdot & f & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & f & \cdot & \cdot & \cdot & \cdot & \cdot & f & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & s & \cdot & \cdot & s & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & g & b & \cdot & s & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & f & \cdot & s & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & s & \cdot & \cdot & s & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & g & b & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & f & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & s & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & w & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{pmatrix}$$

Figure 5.7:

all "dotted" entries are meant to be equal 0) and define $J_S(s_1) = \mathbf{S} - \mathrm{diag}(\mathbf{S}\underline{1})$. $J_S(s_1)$ describes the time that the *SlaveSystem* needs to perform the work that was ordered by the master, with involvement of the two *Helper*.

For the last distribution we find that $J_S(s_2) = (0)$. This is the time for the *SlaveSystem* to become ready again to accept a new job from the *Master*.

The overall system *System* has about 650 states[10]. Adding another slave system would increase the number of states to about 11650 states, adding two even to about 210000. This shows that even small specifications lead to and impressive growth of the numbers of states.

[10]These numbers have been derived by a prototype tool based on the software package SPNP.

5.7.4 Solving the SMCs

The next step in our example is to solve this SMPs we have defined above, or better, the $\mathrm{SMC}(\mathit{System}) = (\Sigma_A, \mathbf{E}_A, J_A) = \mathrm{SMC}(\mathit{Master}) \diamond \mathrm{SMC}(\mathit{SlaveSystem}) \diamond \mathrm{SMC}(\mathit{SlaveSystem})$. The state space is then given as

$$\Sigma_A = \left\{ \underbrace{m_1 \parallel_S s_1 \parallel_S s_1}_{a_1}, \underbrace{m_2 \parallel_S s_2 \parallel_S s_2}_{a_2} \right\},$$

the EMC as

$$\mathbf{E}_A = \mathbf{E}_M \diamond \mathbf{E}_S \diamond \mathbf{E}_S = \begin{pmatrix} & 1 \\ 1 & \end{pmatrix}.$$

Finally,

$$J_A(a_1) = J_M(m_1) \oplus J_S(s_1) \oplus J_S(s_1)$$

and

$$J_A(a_2) = J_M(m_2) \oplus J_S(s_2) \oplus J_S(s_2).$$

Solving the EMCs

The first step to solve $\mathrm{SMC}(\mathit{System})$ is to find a solution for the EMC $\mathbf{E}_A$. Although $\mathbf{E}_A$ has a trivial solution, as well as $\mathbf{E}_M$ and $\mathbf{E}_S$, this is the right moment to demonstrate that $\mathbf{E}_M \otimes \mathbf{E}_S \otimes \mathbf{E}_S$ does not yield the desired EMC for $\mathrm{SMC}(\mathit{System})$:

$$\begin{aligned} \mathbf{E}_M \otimes \mathbf{E}_S \otimes \mathbf{E}_S &= \begin{pmatrix} \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & \cdot \\ 1 & \cdot & \cdot & \cdot \end{pmatrix} \otimes \mathbf{E}_S \\ &= \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \end{aligned}$$

This matrix describes four independent two-state DTMCs, from which only one is of interest for us, since it represents the part with the starting state. Hence, the EMC has not grown at all and its solution is again trivial: $\underline{\pi}^{(\mathit{System})} = (\frac{1}{2}, \frac{1}{2})$.

Mean Values for SMC(*System*)

What we need to compute next are the mean values for the distributions $J_A(a)$ for $a \in \Sigma_A$. We know that

$$\begin{aligned} J_A(a_2) &= J_M(m_2) \oplus J_S(s_2) \oplus J_S(s_2) \\ &= J_M(m_2) \oplus (0) \\ &= \begin{pmatrix} -\mathsf{newjobs} & \mathsf{newjobs} \\ 0 & 0 \end{pmatrix}. \end{aligned}$$

Consequently, the mean value of $J_A(m_2 \|_S s_2 \|_S s_2)$ is $\frac{1}{\mathsf{newjobs}}$.

More difficult is the computation of the mean value of $J_A(a_1) = J_M(m_1) \oplus J_S(s_1) \oplus J_S(s_1)$, since this CTMC has already $2 \cdot 18^2 = 648$ states. Although this is still a relatively small matrix, there is no point to present a symbolic representation of the result here.

Steady-state Solution of SMC(*System*) and Throughputs

The solution of SMC(*System*) is now trivial. By means of Equation B.6 we derive that the steady-state solution of SMC(*System*) is

$$\sigma(a_1) = \frac{J_A(a_1)}{J_A(a_1) + J_A(a_2)}$$

and

$$\sigma(a_2) = \frac{J_A(a_2)}{J_A(a_1) + J_A(a_2)}.$$

Consequently, the throughput of both a_1 and a_2 is

$$\tau(a_1) = \tau(a_2) = \frac{1}{J_A(a_1) + J_A(a_2)}.$$

Reinterpreting the Throughputs

The throughputs $\tau(a_1)$ and $\tau(a_2)$ can now be interpreted in the original local GMPs $[\![Master]\!]$ and $[\![SlaveSystem]\!]$. The throughputs $\tau(a_1)$ is the throughput of the synchronising state $Master \|_S SlaveSystem_1 \|_S SlaveSystem_2 \in \mathcal{S}_{syn}(System)$ and therefore, also of the local states $Master$, $SlaveSystem_1$ and $SlaveSystem_2$. Since we know the branching probabilities for all states of all components, we can employ the results of Section 4.3.1 to obtain the throughputs for all states and the steady-state probabilities for all stable states.

5.7.5 Waiting Times

For the synchronising states, however, we can not derive the steady-state probabilities, since we have to know the mean sojourn time of these states.

Therefore, to obtain the values for these states, we have to employ the results of Section 5.6 to obtain the sojourn times of synchronising states. We have to look at the EMC $\mathbf{E}_A$. This is a simple two-state DTMC, where each state represents the synchronisation of the three components. State a_1 represents a synchronisation over action split, state a_2 over action join.

We have already obtained all information necessary to compute the waiting time and can therefore use Algorithm 5.1 immediately. We show this for state m_1 of the *Master*.

According to the algorithm we first have to find all global synchronising states that have m_1 as a sub-process. There is only one, a_1. For this state we have to find all predecessors. Again, there is only one, a_2. If $X_1 \sim J_M(m_2)$, $X_2 \sim J_S(s_2)$, $X_3 \sim J_S(s_2)$, then we want to compute (in the notation of Algorithm 5.1):

$$W^{m_1}_{m_2,m_1} = E[X_1 \boxplus X_2 \boxplus X_3] - E[X_1].$$

For this specific example, $W^{m_1}_{m_2,m_1}$ is the only value that has to be computed, and therefore, $W^{m_1} = W^{m_1}_{m_2,m_1}$. More specifically, since $X_2 = X_3 = 0$ with probability one,

$$\begin{aligned} W^{m_1} &= E[X_1 \boxplus X_2 \boxplus X_3] - E[X_1] \\ &= E[X_1] - E[X_1] \\ &= 0. \end{aligned}$$

The waiting time for local state m_1 is hence 0.

The same algorithm could now be applied to state m_2 of *Master*, but actually this is not necessary. The reason is that we already know the steady state probabilities of all stable states of *Master* (computed by means of the throughputs $\tau(a_1)$ and $\tau(a_2)$), and of the synchronising state m_1 (which is zero). Since there is now only one state left for which the probability is not known, we can compute it easily from the probabilities that are already known (by simple subtraction of the known probabilities from 1).

The waiting times for state s_1 of component $SlaveSystem_1$ (and also for $SlaveSystem_2$) can be expressed by

$$W^{s_1} = E[X_1 \boxplus X_2 \boxplus X_3] - E[X_2] = E[X_1]$$

5.8 Conclusions

The state-space explosion problem for **AWCI**-processes has been solved. We have introduced a technique, the **AWCI**-Technique, that allows to derive local steady-state measures

for **AWCI**-processes. **AWCI**-processes describe a stochastic semi-Markov process, where the synchronisations are regeneration points. The **AWCI**-Technique is based on a reformulation of the original model, given as a set of local GMP, in terms of a semi-Markov process in a way that is much more compact than a global transition system as derived by the standard SOS rules. We have identified conditions that are sufficient to ensure that the considered system can indeed be solved by the **AWCI**-Technique. Computations are never carried out on the global state space, but always on parts of local state spaces.

As has been shown, a very important problem that arises in the course of the evaluation with the **AWCI**-Technique is the computation of mean values of the maximum of phase-type distributed random variables. Following a naive approach, the complexity of this computation of these values would grow exponentially in the number of considered random variables. In this chapter, it has been shown that the worst case complexity of this problem is in fact only polynomial in the number of considered random variables and polynomial in the size of the absorbing Markov chains that represent the distributions of these random variables.

The **AWCI**-Technique aims at the computation of throughputs of synchronising states. These throughputs can then be interpreted in the entire component by means of the local throughput equations (*cf.* Chapter 4). Throughputs are not sufficient to describe steady-state probabilities for synchronising states. However, we have seen that for **AWCI**-processes waiting times for local synchronising states can be obtained efficiently.

By means of a simple example we have seen that, even though the class of processes is restricted, meaningful examples that meet the requirements of the **AWCI**-Technique are widely available.

Part III

Event Structures and Performance Measures

Chapter 6

Unfoldings and Stochastic Measures

The most common way to give meaning to process algebra specifications is to derive a transition system by means of a set of SOS rules, thus yielding a structure that describes the possible actions and global state changes of the system. This semantics is usually of the interleaving type, *i.e.,* a process $a.\mathsf{stop} \| b.\mathsf{stop}$ is identified with the process $a.b.\mathsf{stop} + b.a.\mathsf{stop}$.

We have seen that for Markovian stochastic process algebras, transition systems are a very advantageous structure to serve as a semantics, since performance models, *i.e.,* CTMCs can be derived most easily.

However, interleaving is only one possibility to express parallelism. There are other formalisms which have an explicit notion of parallelism, *e.g.,* Mazurkiewicz-traces [104], Petri nets, and event structures of different kinds [116, 50, 18, 94, 84].

In this chapter, we will focus on a special variant of *partial-order formalisms*, or *event structures*. The event structures we will consider here are derived from $\mathcal{YAWN}$ processes by means of an *unfolding semantics*. This type of semantics is introduced for process algebras by Langerak and Brinksma [96, 97] and is based on an approach known for Petri nets [105, 106]. We want to find an answer to the question what these formalisms can offer to performance evaluation. This, first, includes the question whether performance models can be expressed in terms of a partial-order model anyway, and, second, whether it is possible to derive measures from them.

The first question has been already addressed in the literature. In [84], Katoen has defined a partial-order semantics for a stochastic process algebra in terms of *bundle event structures*. The SPA was of a most general type, *i.e.,* the actions where endowed with general, positive distribution functions. In [38], Cloth has presented a semantics for a similar SPA, based on unfoldings, as invented by Langerak and Brinksma [96, 97]. A similar approach has been chosen in [127]. All approaches have proven that the representation of most general stochastic systems is straightforward in terms of event structures.

However, the question, whether it is possible to exploit the expressiveness of event structures to derive stochastic measures for them is not yet fully answered. In [86], an approach is presented to use stochastic bundle event structures as a basis to derive a discrete-event simulation model. There are so far no approaches known to exploit stochastic partial-order models for the performance evaluation, thereby using numerical techniques, except for special cases [127, 39]. This is where the considerations of this chapter start.

Outline of this Chapter. In Section 6.1, we describe the generation of event structures in an intuitive manner for *Petri nets*. Second, in Section 6.2, we introduce the actual derivation of event structures from $\mathcal{YAWN}$ specifications. In Section 6.3, we will define *event occurrence times*, random variables that describe the time until an event of an event structure happens, given that it happens at all. In Section 6.4, we define event occurrence probabilities, the probabilities for events to happen at all. In Section 6.5 we use event occurrence times and event occurrence probabilities to express waiting times for local synchronising states of a $\mathcal{YAWN}$ process in terms of its unfolding. In Section 6.6, we reconsider an approach proposed in [16] to compute performance measures from an event structure. We conclude the chapter with Section 6.7.

6.1 The Idea of Unfolding

Unfoldings have first been defined for Petri nets [105, 106]. Unfoldings of Petri nets provide the easiest way to gain admission to the world of event structures.

We demonstrate the idea of unfolding by means of a simple example. We consider the Petri net N in Figure 6.1. N has four places, $s_1, \ldots, s_4$, and four transitions, $t_1, \ldots, t_4$. This net

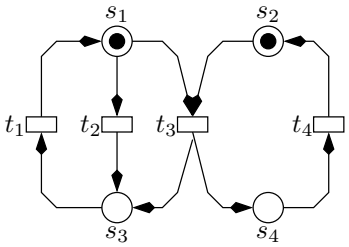


Figure 6.1: Simple Petri net N

has one synchronising transition t_3 and a conflict situation between t_2 and t_3: t_2 is enabled if and only if t_3 is enabled, but only one of then can then fire. The net is cyclic. Now, an unfolding U of N is also a Petri net, that describes in a certain way the same behaviour as N. The big difference is that an unfolding is always *a*cyclic.

An unfolding can be derived from a Petri net by means of the token game. It is created by the successive addition of places and transitions, starting from the empty net. For each

place in the initial marking of N (identifying markings with sets of places[1]) we introduce a new place in the unfolding (*cf.* Figure 6.2, (a)). Now, we fire all transitions from the current marking in N. For each fired transition and each of the places where a token was newly placed, we introduce respective new transitions and new places in the unfolding. In Figure 6.2, (b), we see the unfolding after t_2 and t_3 have been fired. Note that s_3 has been duplicated. We repeat this for all so-far-reached markings in N. In Figure 6.2, (c) and (d), we see some further unfoldings of N. A formal algorithm for the derivation of an unfolding can be found in [53]. Finite unfoldings of cyclic nets like N can be ordered

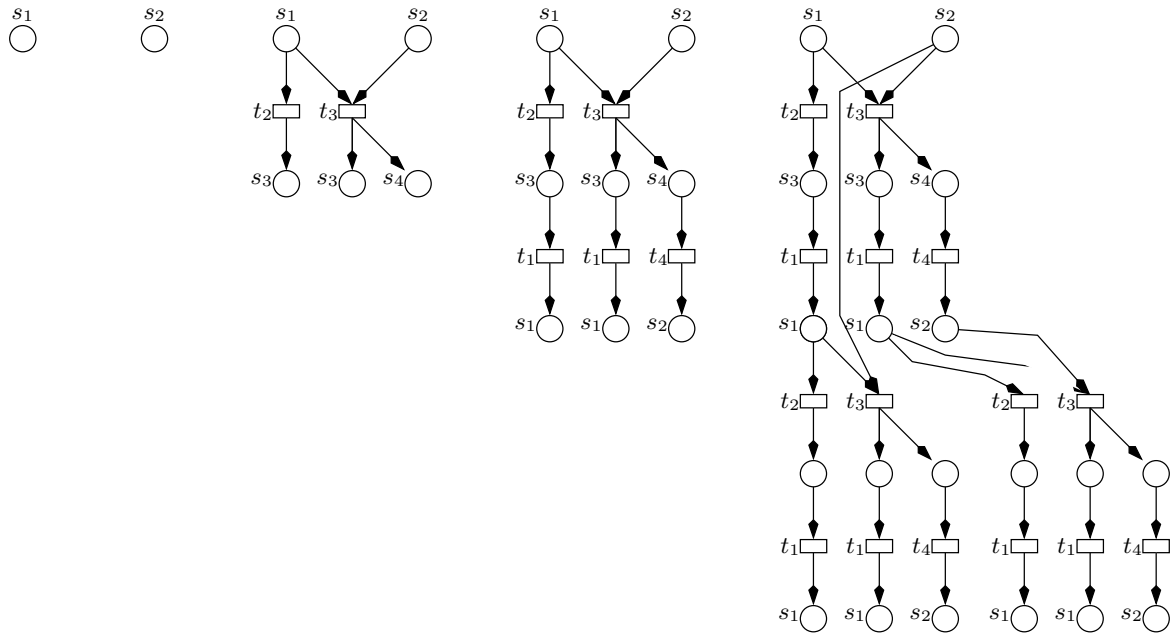


Figure 6.2: Unfoldings for N

in a prefix-relation. The finite prefixes approximate then an infinite unfolding, which is uniquely determined.

Unfoldings can be seen as Petri nets, however, to stress the fact that they are unfoldings, places are usually referred to as *conditions*, transitions as *events*. Events and conditions are also called *nodes*. We can define several relations on nodes which are implied by the structure of the unfolding:

- the causal relation $\leq$. An event e_2 depends on e_1 ($e_1 \leq e_2$), if, when e_2 was fired (or "occurred") in a system run, then e_1 must also have occurred in this run.

[1] In case that we consider non-safe nets, we have to consider multi-sets here.

- the *conflict* relation: two events e_1 and e_2 are in conflict with each other when neither of them can occur in a system run, in which the respective other has already occurred. To be more specific, two events are in conflict in an unfolding, if there is a condition on which both events depend. As conditions are actually places, and events are transitions, we see that in the described case, only one transition (event) can consume the token in the shared place (condition).

The causal and conflict relations can easily be lifted to the set of all nodes.

An unfolding defines an event structure of a special kind, a *prime* event structure [116]. A prime event structure of an unfolding is the set of events of the unfolding, together with the causal relation and the conflict relation, restricted to the event sets.

6.2 Unfolding $\mathcal{YAWN}$ Processes

In this section, we describe an unfolding approach for $\mathcal{YAWN}$ that is based on the approach of Langerak and Brinksma [96, 97]. There, the unfolding was defined for an non-stochastic process algebra. However, as has been shown in [38, 127] , enhancing the approach to stochastic process algebras is straightforward and, more importantly, *orthogonal* to the original approach. Therefore, first we can describe the unfolding approach without explicit reference to the stochastic part of $\mathcal{YAWN}$.

The explanations and definitions in this section are simplified, since many technical details are not relevant for us. For a thorough and formally correct treatment of the topic, we refer the reader to [96, 97, 126].

The *unfolding* of a process algebra specification has many similarities to the unfolding of a Petri net, as we have described it in the previous section. This, however, is not obvious. In Petri nets, we have the concepts of *place* and *transition*, both of which correspond directly to the *conditions* and *events* of the unfolding[2]. In fact, an event in an unfolding represents the *firing of a transition* in the run of the considered Petri net. The firing of a transition is only possible, when it is enabled, *i.e.,* if the enabling condition of the transition is fulfilled. For save Petri nets, this means that there is a token in all places connected to the transition by an incoming arc. The unfolding of a Petri net is then derived by playing the token game.

In process algebras, there is no concept of place or transition in the sense of Petri nets. Instead of firing rules, there are SOS rules that govern the behaviour described by the considered processes. Transitions between states are derived by means of SOS rules. Also there is no token game, which would allow to derive an unfolding. Moreover, the unfolding of an Petri net is again a Petri net, which seems to be very natural. For process algebras, no such natural target formalism does exist.

The approach proposed in [96, 97] comprises two steps:

[2]This correspondence has been formally defined in [50].

1. Invention of concepts that allow to mimic the purpose of places and the firing of transitions in Petri nets. We describe these concepts in Section 6.2.1

2. Definition of a formalism which allows the representation unfoldings of process algebra specifications. We describe this formalism in Section 6.2.2;

Before we come to these topics, we first must define the language for which we want to introduce the concepts. For the sake of simplicity, we will describe all the concepts used in this section only for a sub-language of $\mathcal{YAWN}$ that does not support the hiding operators and that does only use process constants as a means to describe recursion.

Definition 6.1 The syntax of such language is therefore to be the set $\mathcal{L}_{unf}$ of terms defined by the following grammar:

$$P \quad \longrightarrow \quad \mathsf{stop} \quad | \quad a.P \quad | \quad [\lambda].P \quad | \quad P + P \quad | \quad P \|_S P \quad | \quad A,$$

where $\lambda \in \mathbb{R}^+$, $S \subseteq Com$, and A is a process constant.

Note that we have allowed the timed prefix, $[\lambda].P$, but since we delay the treatment of time until Section 6.2.4, we assume that such an expression stands for $\mathbf{t}.P$.

6.2.1 Fragments and States

Several event structure semantics for process algebras have been proposed, *e.g.,* from Boudol et al. [18], Golz et al. [98], or Katoen [84]. These semantics are not based on an unfolding approach, but are of the denotational type. However, common to all of them is that an event of such an event structure denotes a *state change* of the considered process. This is not different for the approach of Langerak and Brinksma. The unfolding approach is based on a successive derivation of state changes and their respective dependencies and conflicts.

In Petri nets, places play a crucial role to define the behaviour of a net and serve as a template for conditions to be added to an unfolding. For the unfolding approach of Langerak, the concept of *fragment* is introduced[3]. Fragments have two purposes: first, they are supposed to play a similar role as places for Petri nets, and second, since there are no tokens to define the enabling of a state change, they are also used to determine when an event can happen. Fragments are defined as follows:

Definition 6.2 (Fragments) The set *Frag* of all *Fragments* is defined by the following grammar:

$$F \rightarrow \mathsf{stop} \mid a.P \mid [\lambda].P \mid (F) \parallel_A \mid \parallel_S (F)$$

where $P \in \mathcal{L}_{unf}$, and $S \subseteq Com$.

[3]Actually, the concept that we call *fragment* here, is named *component* in [96, 97]. We have renamed this to avoid confusion with what have called *component* in the previous chapters.

Fragments are prefixed process algebra expressions (*i.e.,* they are syntactic entities) which are enclosed in *synchronisation contexts*, denoted by the $(\cdot)\|_S$ operators. If possible, we will omit the contexts, or, if not possible, at least the parentheses. A fragment that is prefixed with a timed prefix (like $[\lambda].\cdot$ for $\lambda \in \mathbb{R}$) is said to be a *timed* fragment. A fragment that is prefixed with a synchronising prefix (like $a.\cdot$ for $a \in Com$) is said to be a *synchronising* fragment. In the following, we consider sets of fragments, and it will become necessary to assign a synchronisation context to all elements of a fragment set. If $F \subseteq Frag$, then we define $\|_A F = \{\|_A(f) \mid f \in F\}$ and $F\|_A = \{(f)\|_A \mid f \in F\}$. We enhance this on relations, *i.e.,* if $R \subseteq Frag \times Frag$, then $\|_A R = \{(\|_A(f_1), \|_A(f_2)) \mid (f_1, f_2) \in R\}$, and $R\|_A$ accordingly.

Fragments are now used to define a new notion of *state*, the so called *d-state*. Whereas in the usual meaning a state is just a process algebra term, a d-state is a set of fragments, together with a relation on this set, the *choice relation*. However, there is a tight relationship between the two concepts, as we will see below. Formally, a d-state is defined as follows:

Definition 6.3 A *d-state* is a tuple $(\mathcal{S}, \mathcal{C})$, where $\mathcal{S}$ is a set[4] of fragments and $\mathcal{C} \subseteq \mathcal{S} \times \mathcal{S}$ an irreflexive and symmetric relation between fragments, the *choice* relation.

Formally, process algebra terms are mapped on d-states by means of the decomposition function *dec*:

Definition 6.4 *dec* is defined inductively as follows (we assume that $dec(P) = (\mathcal{S}(P), \mathcal{C}(P))$ and $dec(Q) = (\mathcal{S}(Q), \mathcal{C}(Q))$):

$$\begin{aligned}
dec(\mathsf{stop}) &= (\{\mathsf{stop}\}, \emptyset) \\
dec(a.P) &= (\{a.P\}, \emptyset) \\
dec([\lambda].P) &= (\{[\lambda].P\}, \emptyset) \\
dec(P\|_A Q) &= (\mathcal{S}(P)\|_A \cup \|_A \mathcal{S}(Q), \mathcal{C}(P)\|_A \cup \|_A \mathcal{C}(Q)) \\
dec(P + Q) &= (\mathcal{S}(P) \cup \mathcal{S}(Q), \mathcal{C}(P) \cup \mathcal{C}(Q) \cup (\mathcal{S}(P) \times \mathcal{S}(Q))^s) \\
dec(A) &= dec(P), \text{if } A \stackrel{\text{def}}{=} P
\end{aligned}$$

where $(A \times B)^s$ denotes the symmetric closure of $(A \times B)$.

Example 6.5

A d-state can obviously seen as a reformulation of a process algebra term. To demonstrate this, we consider the process

$$P = (a.P_1 + b.P_2) \|_a (c.P_3 + a.P_4).$$

[4] Formally, we should consider multi-sets here, since there can be identical fragments which must be distinguished. However, we assume that we can somehow distinguish all fragments, so that the use of sets is safe. In fact, in [96, 97], the fragments are endowed with additional unique indices, which we have left out here for the sake of simplicity.

Then,

$$dec(P) \quad = \quad (\mathcal{S}, \mathcal{C})$$

where

$\mathcal{S} = \{(a.P_1)\|_a,\ (b.P_2)\|_a,\ \|_a(c.P_3),\ \|_a(a.P_4)\ \}$ and $\mathcal{C} = \{(a.P_1, b.P_2), (c.P_3, a.P_4)\}^s$.

The term P is decomposed in the four fragments contained in $\mathcal{S}$. Each fragment is enclosed in a synchronisation context, denoting that the respective fragment is supposed to synchronise, in this case, over the action a. The fact that there is a choice between different behaviours in term P, on both sides of the parallel operator, is now reflected by the choice relation $\mathcal{C}$. For the sub-process $a.P_1 + b.P_2$ we have now the elements $\{(a.P_1, b.P_2), (b.P_2, a.P_1)\}$ in the choice relation $\mathcal{C}$. The same holds for the sub-process $\{(c.P_3, a.P_4), (a.P_4, c.P_3)\}$.

We only consider complete d-states, i.e., d-states $(\mathcal{S}, \mathcal{C})$ for which a term P exists such that $dec(P) = (\mathcal{S}, \mathcal{C})$. *dec* is not injective, but all terms $P, P', P'', \ldots$ which decompose to the same d-state differ only with respect to the commutativity and associativity of the choice operator.

6.2.2 Condition Event Structures

Before we describe how d-states are used to derive an unfolding, we describe the formalism that is used to represent unfoldings. In case for Petri nets this was not necessary since the unfolding of a Petri net was again a Petri net. Here the unfolding is not an acyclic Petri net (although very similar), but a new structure: a *condition event structure*.

Definition 6.6 A *condition event structure* is a 4-tuple $(D, E, \bigstar, \prec)$, where

- D is a set of conditions
- E is a set of events
- $\bigstar \subseteq D \times D$ is a symmetric and irreflexive *choice* relation
- $\prec \subseteq (D \times E) \cup (E \times D)$ is the *flow relation*

There are some similarities between the acyclic Petri nets that we have seen in Section 6.1 and condition event structures. Events, conditions and the flow relation have the same purpose for both formalisms. New in condition event structures is the choice relation $\bigstar$. We describe its purpose by means of an example:

Example 6.7

We consider the condition event structure $(D, E, \bigstar, \prec)$, where

$$\begin{aligned} D &= \{d_1, d_2, d_3, d_4, d_5\} \\ E &= \{e_1, e_2, e_3\} \\ \bigstar &= \{(d_2, d_4)\}^s \end{aligned}$$

and the flow relation is defined as described by the arcs in Figure 6.3. If we imagine

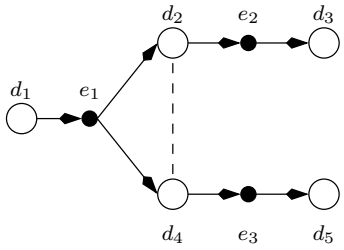


Figure 6.3: Condition Event Structure (Example 6.7)

a token in d_1 (just as we do for Petri nets), then the event e_1 is enabled by this token and can happen. The successor conditions of e_1 are d_2 and d_4, and they are in the choice relation, as is depicted by the dashed line in Figure 6.3: $d_2 \bigstar d_4$. The question is now, what happens after the occurrence of e_1. Where does the token go? Intuitively, we can assume that the token is distributed over *both* conditions d_2 and d_4, as long as the choice between has not been made. The purpose of the choice relation is to describe that *only one* of the events e_2 and e_3 can happen.

We introduce now some notation which allows us to deal more conveniently with condition event structures.

The transitive closure of $\prec$ on $D \cup E$ is denoted by $<$, the transitive and reflexive closure by $\leq$. A *marking* is a set of conditions. A node is either a condition or an event, and we define the set of all nodes N as $N = D \cup E$. The *initial marking* M_0 is defined by $M_0 = \{d \in D \mid \neg\exists d' \in D : d' \leq d \text{ and } d' \neq d\}$.

Although we do not define it explicitly in the following, we always assume an event $\perp \in E$ that is minimal with respect to the flow relation, *i.e.,* there is no node x such that $x \leq \perp$, and, for all $y \in N$, $\perp \leq y$ does hold. We call this event the *bottom event*.

The *preset* of a node x, denoted by ${}^\bullet x$, is defined as ${}^\bullet x = \{y \in N \mid y \prec x\}$. Accordingly, the *postset* $x^\bullet$ is defined as $x^\bullet = \{y \in N \mid x \prec y\}$. When the preset (postset) of a node x is a singleton, *i.e.,* ${}^\bullet x = \{y\}$ or $x^\bullet = \{y\}$, we frequently identify ${}^\bullet x$ and $x^\bullet$ with y, respectively.

We enhance the notion of pre- and postset on sets of nodes. When I is an arbitrary index set, then ${}^\bullet\{\ x_i \in N \mid i \in I\} = \bigcup_{i\in I} {}^\bullet x_i$. $\cdot^\bullet$ is defined accordingly. We abbreviate ${}^\bullet({}^\bullet x)$ and $(x^\bullet)^\bullet$ as ${}^{\bullet\bullet}x$ and $x^{\bullet\bullet}$, respectively.

Sometimes it is necessary to denote the set of conditions that are in a choice relation with a certain condition d. We denote this set as $\bigstar d = \{d' | d' \bigstar d\}$.

The relation $\leq$ is the causality relation, as for Petri net unfoldings. There is also a conflict relation.

Definition 6.8 The *conflict relation* on nodes $\sharp \subseteq N \times N$ is defined as follows: for $x_1, x_2 \in N$, $x_1 \sharp x_2$ iff there are two nodes y_1 and y_2, such that $y_1 \leq x_1$, $y_2 \leq x_2$, and

1. either $y_1 \bigstar y_2$,
2. or $y_1, y_2 \in E$ and ${}^\bullet y_1 \cap {}^\bullet y_2 \neq \emptyset$.

Note that, different to Petri net unfoldings, condition event structures have *two* sources of conflict. The first is the same as for Petri net unfoldings (Item 2 in Definition 6.8). The second one is is introduced by the choice relation that is defined on the set of conditions (Item 1).

We have said before that event structures have an explicit notion of concurrency. This is expressed by the following relation, which is the negation of $\leq$ and $\sharp$: the *independence relation* $\asymp$.

Definition 6.9 (Independence relation $\asymp$) Two nodes $x, x' \in N$ are said to be *independent* ($x \asymp x'$), iff neither $x \leq x'$, $x' \leq x$, nor $x \sharp x'$. If $X \subseteq N$, we say that $x \asymp X$ iff $\forall x' \in N : x \asymp x'$.

Independent events can occur independently from each other. Stated otherwise, when e, e' are events and $e \asymp e'$, then, when e happens then e' can also happen (if its occurrence is not inhibited by a third event), and vice versa.

6.2.3 From Processes to Event Structures

In this section, we describe how a condition event structure can be derived from a process algebra term by unfolding. To do so, we first introduce an extension to the definition of condition event structures, which links d-states and condition event structures.

Definition 6.10 (Fragment event structure) A condition event structure $(D, E, \bigstar, \prec)$, together with mappings $l_D : D \longrightarrow \mathit{Frag}$ and $l_E : E \longrightarrow \mathit{Act}_{\mathbf{t}}$, is said to be a *fragment event structure*.

Frequently, we will denote a fragment event structure by a tuple $(D, E, \bigstar, \prec, l_D, l_E)$. A fragment event structure is a condition event structure, where the conditions are labelled with fragments and the events with actions. The fragments are needed to define the unfolding algorithm for process algebra terms.

Now, we will define the unfolding procedure for a process $P \in \mathcal{L}_{unf}$. For Petri nets, the unfolding is generated by playing the token game and introducing new conditions and events, depending on which transition was fired and which places were changed. The basic operation there is the firing of transitions. We define now a mechanism that takes the same place in the context of condition event structures, as transition-firing in the context of Petri nets. We define a set of SOS rules that define transitions between *sets of fragments* and *d-states*. The transitions are labelled with actions. The derived transitions are used

1)	$\dfrac{}{\{a.P\} \xrightarrow{a} dec(P)}$
2)	$\dfrac{}{\{[\lambda].P\} \xrightarrow{\mathbf{t}} dec(P)}$
3)	$\dfrac{\mathcal{S} \xrightarrow{a} (\mathcal{S}', \mathcal{C}')}{\mathcal{S}\|\|_A \xrightarrow{a} (\mathcal{S}'\|\|_A, \mathcal{C}'\|\|_A)} \quad (a \notin S)$
4)	$\dfrac{\mathcal{S} \xrightarrow{a} (\mathcal{S}', \mathcal{C}')}{\mathcal{S}\|\|_A \xrightarrow{a} (\|\|_A\mathcal{S}', \|\|_A\mathcal{C}')} \quad (a \notin S)$
5)	$\dfrac{\mathcal{S}_1 \xrightarrow{a} (\mathcal{S}_1', \mathcal{C}_1') \qquad \mathcal{S}_2 \xrightarrow{a} (\mathcal{S}_2', \mathcal{C}_2')}{\mathcal{S}_1\|\|_A \cup \|\|_A\mathcal{S}_2 \xrightarrow{a} (\mathcal{S}_{1'}\|\|_A \cup \|\|_A\mathcal{S}_2', \mathcal{C}_1'\|\|_A \cup \|\|_A\mathcal{C}_2')} \quad (a \in S)$

Table 6.1: SOS defining a transition relation between d-states

to successively extend an existing unfolding. If $P \in \mathcal{L}_{unf}$ and $dec(P) = (\mathcal{S}, \mathcal{C})$, then the smallest non-trivial finite unfolding of P is a fragment event structure $(D, E, \bigstar, \prec, l_D, l_E)$, where D is a set of conditions with the same cardinality as $\mathcal{S}$, $l_D = \mathcal{S}$, $l_E = \emptyset$, $\prec = \emptyset$, and $d \bigstar d'$ iff $l_D(d)\ \mathcal{C}\ l_D(d')$.

Starting from this unfolding, we can define extensions of a fragment event structure as follows.

Definition 6.11 Let $\mathcal{E} = (D, E, \bigstar, \prec, l_D, l_E)$ be a fragment event structure. The set of possible extensions $PE(\mathcal{E})$ of $\mathcal{E}$ is the set of all pairs $(\mathcal{D}, \mathcal{S} \xrightarrow{a} (\mathcal{S}', \mathcal{C}'))$ such that

1. $\mathcal{D} \subseteq D$ is a set of pairwise independent conditions with $l_D(\mathcal{D}) = \mathcal{S}$;
2. the transition $\mathcal{S} \xrightarrow{a} (\mathcal{S}', \mathcal{C}')$ can be derived by the rules in Table 6.1;
3. E does not already contain an event e with $l_E(e) = a$ such that ${}^\bullet e = \mathcal{D}$.

The possible extensions are used to enhance the original fragment event structure by adding a new event e, $e \notin E$, with $l_E(e) = a$, and a set of new conditions $\mathcal{D}'$, one for each fragment in $\mathcal{S}'$, and labelled by l_D with the respective fragments. The choice relation $\mathcal{C}$ is used to enhance the choice relation $\bigstar$ on $\mathcal{D}'$. Additionally, we require that ${}^\bullet e = \mathcal{D}$ and $e^\bullet = \mathcal{D}'$. In [97], an algorithm is drafted which constructs a fragment event structure, starting from an initial unfolding by successive addition of possible extensions to the unfolding. Without describing it in more detail, we define the unfolding $Unf(P)$ of a $\mathcal{YAWN}$ process P to be the fragment event structure constructed by successive addition of all possible extensions of the initial marking $\mathcal{S}$, where $dec(P) = (\mathcal{S}, \mathcal{C})$.

6.2.4 Markovian Fragment Event Structures

So far, we have not considered the fact that the language $\mathcal{L}_{unf}$ describes processes with timed prefixes. However, the extension of fragment event structures into a stochastic, Markovian version can be done most easily. Actually, a fragment event structure does implicitly contain already all information to make it a stochastic fragment event structure, as we will see below. However, we will now define the necessary enhancements explicitly. First, we enhance the definition of fragment event structure to describe stochastic information.

Definition 6.12 (Markovian Stochastic Fragment Event Structure) A fragment event structure $(D, E, \bigstar, \prec, l_D, l_E)$, together with a mapping $\lambda_D : D \longrightarrow \mathbb{R}^+ \cup \{\infty\}$, is said to be a *Markovian stochastic fragment event structure*.

We write Markovian stochastic fragment event structures as tuple $(D, E, \bigstar, \prec, l_D, l_E, \lambda_D)$. The function λ_D assigns rates to conditions. We could define λ_D by means of a refined version of the unfolding algorithm, where each extension of an unfolding also extends the definition of λ_D. However, this is not necessary. The function l_D maps conditions to fragments, and these fragments contain the information that we need. Therefore, we define λ_D by means of l_D.

Definition 6.13 Let $Unf(P) = (D, E, \bigstar, \prec, l_D, l_E)$ be a fragment event structure derived from a process $P \in \mathcal{L}_{unf}$. Then, we define the function $\lambda_D : D \longrightarrow \mathbb{R}^+ \cup \{\infty\}$ as follows. For all $d \in D$,

$$\lambda_D(d) = \begin{cases} \lambda & \text{if } l_D(d) = [\lambda].P \\ \infty & \text{otherwise.} \end{cases}$$

Note that a fragment $l_D(d)$ for $d \in D$ is usually enclosed in a synchronisation context. For the sake of simplicity, we have ignored them, since they do not contain relevant information for the definition of λ_D.

6.2.5 Some Important Definitions

In this section, we define the most basic notions to handle condition event structures. We consider a condition event structure $\mathcal{E} = (D, E, \bigstar, \prec)$.

First, we introduce the notion of configuration. A configuration is a (finite) set of events $C \subseteq E$ that have happened in a system run and that is closed with respect to the flow equation, *i.e.,* if $e \in C$, the all events $e' \in E$ with $e' \leq e$ are also in C. A configuration describes how far the considered event structure has evolved. More formally, a configuration is defined as follows.

Definition 6.14 (Configuration) A set of events $C \subseteq E$ is said to be a *configuration*, iff it is conflict-free ($\forall e, e' \in C : (e, e') \notin \sharp$) and left-closed (if $e \in C$, $e' \in E$ and $e' \leq e$ then $e' \in C$).

Example 6.15

If we reconsider Example 6.7, then $\{e_1\}, \{e_1, e_2\}$ and $\{e_1, e_3\}$ are configurations. The set $\{e_2, e_3\}$ is not a configuration.

A special case of a configuration is the *local configuration*.

Definition 6.16 (Local configuration) A configuration C is said to be a *local configuration* iff $\max(C)$ (with respect to $\leq$) is a singleton. If e is an event, we write $[e]$ for the local configuration which has e as the maximum element.

The local configuration $[e]$ of an event e is uniquely determined: it contains all, and only those, events (including e) that are necessary to let e happen.

A *marking* is a set of conditions. We denote the set of minimal conditions of a condition event structure (with respect to $\leq$) as M_0, the *initial marking*.

A very important concept is now that of a *cut*: a cut can be seen as a reachable marking within a condition event structure.

Definition 6.17 (Cut) A *cut* is a marking M such that for each pair $d, d' \in M$, $d \neq d'$ holds: $d \asymp d'$ or $d \bigstar d'$ and that M is maximal with respect to set inclusion.

Example 6.18

For Example 6.7, $\{d_1\}$ is a cut (in fact, also the initial marking), as well as $\{d_2, d_4\}$, $\{d_3\}$, and $\{d_5\}$. The set $\{d_3, d_5\}$ is not a cut, since $d_3 \sharp d_5$.

A nice property is that there is a one-to-one correspondence between cuts and configurations.

Definition 6.19 Let C be a configuration and M a cut.

1. $Cut(C) = (M_0 \cup {}^\bullet C) \setminus ({}^\bullet C \cup \bigstar({}^\bullet C))$
2. $Conf(M) = \{e \in E \mid \exists d \in M : e \leq d\}$.

Lemma 6.20 Let C be a configuration and M a cut.

1. $Cut(C)$ is a cut;
2. $Conf(M)$ is a configuration;
3. $Conf(Cut(C)) = C$;
4. $Cut(Conf(C)) = M$.

Proof: See [97], Theorem 3.10. →•

6.3 Event Occurrence Times

In the previous section, we have described an unfolding semantics for $\mathcal{YAWN}$. The unfoldings are fragment event structures, enhanced with stochastic information. We have seen that an unfolding semantics is very suitable to give meaning to stochastic process algebra models, and in [38, 84, 127] it has been shown that it is straightforward to give more general than just Markovian stochastic process algebras a semantics in terms of stochastic event structures.

The representation of stochastic models is one issue. Another issue is to actually use the representation to derive performance measures. In the following sections, we will investigate whether stochastically enhanced partial-order models can be of use for the stochastic analysis of performance models.

We will not be able to provide an exhaustive answer to the above question. We begin with the definition of stochastic measures that describe the stochastic properties of partial-order models. With these measures we will be able to describe the occurrence times of events as random variables, the probability that an event occurs at all and the time that passes between the occurrence between a certain "starting" event and several stopping events.

These measures were used in [16]. There, an algorithm for the approximation of waiting times of local, synchronising states is proposed. However, some unproven assumptions were

made, so that the whole approach was not complete. In this chapter, however, we will be able to complete the approach of [16] and to assess it more thoroughly.

In this section, we will consider the *event occurrence time* of an event e of a Markovian fragment event structure $\mathcal{E} = (D, E, \bigstar, <, l_D, l_E, \lambda_D)$, such that $\mathcal{E} = Unf(P)$ for $P \in \mathcal{L}_{unf}$. An event $e \in E$ determines a finite system run, in which all events of the local configuration $[e]$ have happened. We are now interested in the time between system start, *i.e.,* the occurrence of event $\perp$, and the occurrence of e, assuming that e does not become permanently inhibited, *e.g.,* by the occurrence of an event e' that is in conflict with e.

We will assume the maximal-progress property. If t is the time where event e becomes enabled, and t' is the time until it occurs, then we always assume that $t = t'$.

6.3.1 Definition of Occurrence Times

The stochastic information of a Markovian fragment event structure is associated with the conditions. Now, for $d \in D$, we denote by Y_d a random variable which is either exponentially distributed with rate $\lambda_D(d)$, when $\lambda_D(d)$ is finite, or 0 with probability 1, when $\lambda_D(d) = \infty$. The time interval described by Y_d starts with the occurrence of the event ${}^\bullet d$. We say that a condition d is *active* from the moment the (unique) event $e \in {}^\bullet d$ occured *and* the time Y_d has passed. We say that Y_d is the *activation delay* of d. A condition d is *deactivated* from the instant that an event e' happens that is either in conflict with d ($d \sharp e'$), or which depends on d ($d \leq e'$). In the former case, d must not necessarily have been activated. Stated differently, d can be deactivated before it ever becomes activated. In the latter case, d has been activated, *i.e.,* the activation delay Y_d must have been expired. In that case, event e' is an element of $d^\bullet$.

It is now possible to express the time between system start and the occurrence of event e in terms of the random variables Y_d. We denote the occurrence time of event e as X_e, and define it as follows:

Definition 6.21 (Event occurrence time) Let $e \in E$. Then the *event occurrence time* of e is defined as

$$X_e = \begin{cases} 0 & \text{if } e = \perp; \\ \max\{Y_{d'} + X_{e'} \mid d' \in {}^\bullet e \text{ and } e' \in {}^\bullet d'\} & \text{otherwise.} \end{cases}$$

Note that, since ${}^\bullet d$ is a singleton for each condition $d \in D$, the random variable X_e is uniquely determined.

Example 6.22

We consider the processes P_1 and Q_1, which are structurally identical, except that we chose different rates for timed prefixes.

$$\begin{array}{lcl|lcl} P_1 & \stackrel{\text{def}}{=} & [\lambda_1].P_2 & Q_1 & \stackrel{\text{def}}{=} & [\mu_1].Q_2 \\ P_2 & \stackrel{\text{def}}{=} & a.\mathsf{stop} \ + \ [\lambda_2].\mathsf{stop} \ + [\lambda_3].\mathsf{stop} & Q_2 & \stackrel{\text{def}}{=} & a.\mathsf{stop} \ + \ [\mu_2].\mathsf{stop} \ + [\mu_3].\mathsf{stop} \end{array}$$

Let $R \stackrel{\text{def}}{=} P_1 \|_a Q_1$. The unfolding $Unf(R)$ of R is depicted in Figure 6.4 (the re-

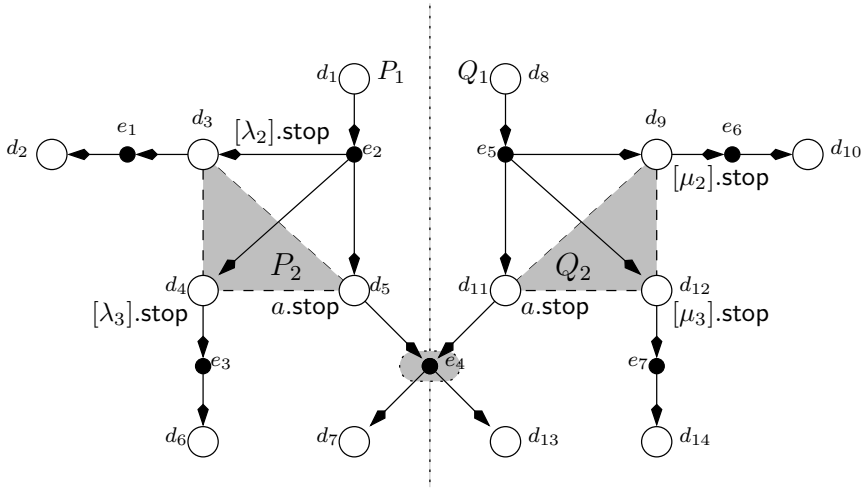


Figure 6.4: Unfolding of R (Example 6.22)

spective three conditions that make up the states P_2 and Q_2 are connected by the shaded areas). We have two sets of conditions here which are in a choice relation: $\{d_3, d_4, d_5\}$ for component P_1, and $\{d_9, d_{11}, d_{12}\}$ for component Q_1.

We give event occurrence times for different events. Event e_2 happens as soon as condition d_1 is activated. This happens at time Y_{d_1}, where Y_{d_1} is exponentially distributed with rate λ_1. Therefore, $X_{e_2} = Y_{d_1}$. Event e_3 happens as soon as condition d_4 is activated. This happens at time $X_{e_2} + Y_{d_4}$ (Y_{d_4} being exponentially distributed with rate λ_3), hence, $X_{e_3} = X_{e_2} + Y_{d_4}$.

Event e_4 is the only synchronisation event in the unfolding, and it happens when both d_5 and d_{11} are activated. Since $\lambda_D(d_5) = \lambda_D(d_{11}) = \infty$, $Y_{d_5} = Y_{d_{11}} = 0$ with probability 1, and the event enabling time for e_4 is therefore

$$X_{e_4} = \max\{X_{e_2} + Y_{d_5}, X_{e_5} + Y_{d_{11}}\} = \max\{X_{e_2}, X_{e_5}\}$$

6.3.2 PERT Networks

The occurrence time of an event e in an unfolding $Unf(P)$ is completely determined by the Markovian fragment event structure that is given by $Unf(P)$, restricted on the events of $[e]$. This fragment events structure is conflict-free, and the flow relation $\prec$ defines an directed acyclic graph with root $\bot$ and (only) leaf e.

These structures (which are conflict-free, finite condition event structures) have some similarities to a concept known for a very long time: PERT networks. PERT is an abbreviation for *Performance Evaluation and Review Technique*, and was already developed in the 50's of the last century [99]. Following [49], a PERT network describes a *project*, which comprises *activities* and *event*s[5]. An activity is an entity that consumes time, which is described by a non-negative random variable. An *event* in a PERT network is an well-defined *occurrence* in time, that does not consume time itself. Activities and *event*s are ordered by a precedence relation. Events denote the start or end of an activity, and an activity can only start, when its preceding activities have finished. A project can be represented by an directed, acyclic graph, where the vertices denote activities and the nodes *event*[6].

Obviously, there are some similarities between PERT networks and stochastic condition event structures: activities correspond to conditions, *event*s to events and the precedence relation to causality. It is easy to interpret a conflict-free, finite Markovian condition event structure as a PERT network, and vice versa.

PERT networks are used in Operations Research to assess the duration of projects that are pursued by the accomplishment of different (dependent) activities. The measures of interest of an PERT network are therefore its *completion time* distribution, and the mean completion time. Deriving accurate results for these quantities is not a trivial problem, since the length of activities can be arbitrarily distributed and activities may depend on each other. Many research activities aimed at the development of techniques to derive reliable results for completion time measures (in [1] a bibliography on the subject is given).

Several methods for the computation of the completion time of a PERT network exist, and some are implemented in the tool PEPP [43, 61]. In [127, 126], the tool FOREST is described. FOREST is capable to derive a finite unfolding of a process algebra specification (as defined in [96]), and exports a task-graph to PEPP. PEPP then computes approximations of completion (event occurrence) time distributions as well as mean completion times.

6.4 Probabilities for Events

Important information about an event e is the probability for it to happen. We call this the *occurrence probability* of e. In this section, we will define the occurrence probability for events. Informally stated, an event e occurs if and only if no other event has occurred before, which is in conflict with e. The probability that e happens is the probability that all relevant conflict situations are decided in favour of e. Therefore, we must turn our attention first on the resolution of conflicts in a fragment event structure.

[5] We write the PERT *event*in a *slanted face* to distinguish them from event structure *events*.

[6] Or vice-versa: it is also possible to regard nodes as activities and vertices as *event*: both representations are equivalent.

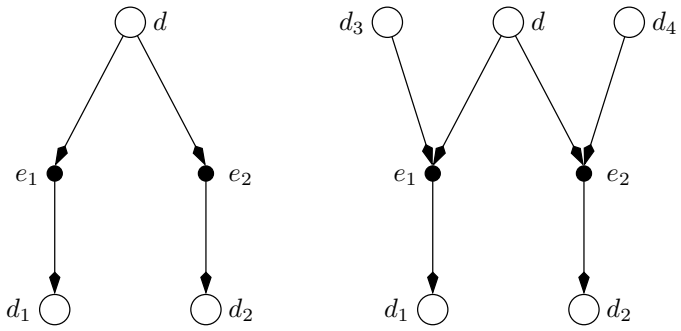
6.4.1 Sources of Conflict

The definition of the conflict relation $\sharp$ states that there are two sources of conflict in an unfolding (*cf.* Definition 6.8). The following theorem states that an unfolding $Unf(P)$ that is derived from an s-deterministic process $P \in \mathcal{L}_{unf}$ has only *one* source of conflict.

Theorem 6.23 In an unfolding $Unf(P)$ for $P \in \mathcal{YAWN}$ that is s-deterministic, the only source of conflict in $Unf(P)$ is the choice between conditions.

Proof: According to Definition 6.8, there are two sources of conflict between two nodes x_1 and x_2: either the conflict is introduced by the choice relation between preceding conditions (we call this a conflict of Type 1), or because there is a condition d, such that $d \leq x_1$ and $d \leq x_2$ (Type 2)[7]. We want to show that all conflicts are of Type 1. We do this by showing that a conflict that seems to be of Type 2 must have been introduced earlier by a conflict of Type 1.

Example 6.24



(a) (b)

Figure 6.5: Conflict inducing situations

A typical conflict situation of Type 2 is depicted in Figure 6.5 (a). However, Figure 6.5 (a) is incomplete. The depicted situation can never arise in an unfolding, since there is no single fragment that allows the derivation of two events. Therefore, if we have a situation that a condition d has more than one successor event, then for any of these events $e \in d^\bullet$, the preset ${}^\bullet e$ must contain more than one condition. In Figure 6.5 (b), we see an example of this situation. Since $\#{}^\bullet e_j > 1$ for $j = 1, 2$, both events are *synchronising*. A first conclusion of our considerations so far is therefore that conflicts of Type 2 can only occur when synchronisation is involved.

[7] Actually, there might be more conditions than only one on which x_1 and x_2 can both depend. The considerations for two conditions can, however, be easily extended to the more general case.

So, if we assume a condition d with $\#d^\bullet > 1$, then for events $e, e' \in d^\bullet$, the conditions $d' \in {}^\bullet e \cup {}^\bullet e'$ for $d' \neq d$ must be independent from d: otherwise, e and e' could not happen. We can also conclude that there must be conditions $c \in {}^\bullet e$ and $c' \in {}^\bullet e'$ such that c and c' are in conflict. To justify this, we assume that all conditions of ${}^\bullet e \cup {}^\bullet e'$ are pairwise independent. Then also e and e' would be independent. There are three different scenarios that we have to consider now:

1. All events in ${}^{\bullet\bullet}e$ occur before event ${}^\bullet d$, and event ${}^\bullet d$ happens before some of the events ${}^{\bullet\bullet}e'$. In this case, event e would happen.

2. The same situation, only with the role of e and e' swapped; then e' would happen.

3. All events in ${}^{\bullet\bullet}e \cup {}^{\bullet\bullet}e'$ occur before event ${}^\bullet d$. In this case, when ${}^\bullet d$ occurs, both e and e' still can happen. Which one will happen can not be described stochastically: this is a nondeterministic situation. This means that the unfolding is not s-deterministic, which contradicts the assumption.

Therefore, there must be a $c \in {}^\bullet e$ and $c' \in {}^\bullet e'$ such that $c \sharp c'$.

Example 6.25

For the situation depicted in Figure 6.5 (b), d_3 and d_4 must be in conflict.

Since we have now at least two conflicting conditions $c \in {}^\bullet e$ and $c' \in {}^\bullet e'$, we can repeat the argumentation of this proof: either the conflict can be traced back to a conflict introduced by choice; then we are done; or we find again a situation similar to that in Figure 6.5: then we can start the argumentation of this proof again. This perhaps can be repeated for a while, but since the flow relation is well-founded, this must eventually end, at least at the initial cut. The only way to introduce conflict in the initial cut is the choice relation. Hence, conflicts in a s-deterministic process is always introduced by choice, which proves the theorem. →●

Please note that this theorem does not state that a process is always s-deterministic when only choices introduce conflict. The process $a.P + a.Q$ is the simplest example for a nondeterministic process.

6.4.2 Occurrence Probabilities

In this section, we define the function $\Omega : E \to [0, 1]$, which assigns occurrence probabilities to events. Note that Ω is *not* a probability measure on E: the probabilities usually do not sum up to one. Would Ω be indeed a probability measure, then, since we must assume that

the bottom event $\perp$ occurs with probability one, all other events would have an occurrence probability of zero, which does not make sense.

Citing Feller [54], one should never speak of probabilities "except in relation to a given sample space". Therefore, we should indicate on which sample space the probabilities $\Omega(e)$ are actually based. We will not do this formally, since it would go beyond our needs in this chapter. However, informally, the sample space is *the set of all system runs* of an unfolding, *i.e.,* the set of all infinite configurations. This set is not denumerable (except when the unfolding is finite), which makes an explicit definition of a probability measure on its sample points impossible. However, interpreted on the sample space, $\Omega(e)$ for $e \in E$ is the probability for event e to occur in a system run. This probability can indeed be expressed by finite means and without explicit reference to the sample space.

Although Ω is not a probability measure on E, it defines a probability measure on *subsets* of events of E. We will come to this later.

In Section 6.2, we have described an unfolding semantics for Markovian process algebra in its most general form. However, in this section we limit the class of processes that we want to consider. We not only demand that a process P has to be an element of $\mathcal{L}_{unf}$, but also of $\mathcal{L}_{\mathcal{CC}}$ (*cf.* Definition 3.20). This restriction ensures that the choice relation between fragments (and in the unfolding, between conditions) is not only symmetric, but also transitive. If we would drop the structural conditions of $\mathcal{L}_{\mathcal{CC}}$, then this property can not be ensured. Choices can be seen as experiments, where the outcomes are the events that can resolve the choice. Hence, obtaining probabilities for the different "outcomes" of a "choice experiment" is crucial for the determination of the occurrence probabilities of events. When we now allow processes of arbitrary structure, then the choice relation can become arbitrarily complicated, and, consequently, also the derivation of probabilities for the different outcomes of a choice. If, on the other hand, we allow only processes $P \in \mathcal{L}_{unf} \cap \mathcal{L}_{\mathcal{CC}}$, then due to the transitivity of the choice relation, the definition of occurrence probabilities becomes much simpler. Since we have considered only processes $P \in \mathcal{L}_{\mathcal{CC}}$ in this dissertation anyway, we will not aim at utmost generality, but stick with this restriction.

Example 6.26

(From [96]) We consider the process $P \stackrel{\text{def}}{=} c.\mathsf{stop} + (a.\mathsf{stop} \| b.\mathsf{stop})$. P is not element of $\mathcal{L}_{\mathcal{CC}}$. In Figure 6.6 we see the unfolding of P. The actions a and b are independent from each other, and hence are the conditions d_1 and d_2. Both a and b are, however, in choice with c, hence condition d_3 is in choice with d_1 and d_2.

We define $\Omega(e)$ in two steps. First, we must derive the probability for an event to happen, once all its preceding events have happened. We denote these probabilities as *immediate occurrence probabilities* (IOP). Second, we must combine the IOP to obtain the values for $\Omega(e)$.

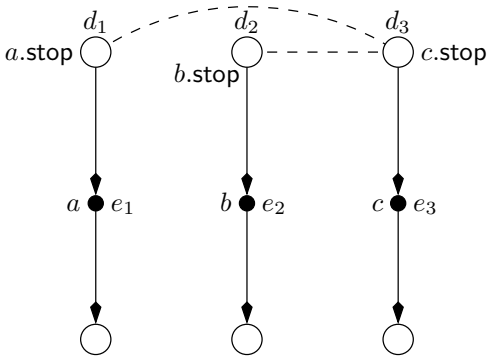


Figure 6.6: $Unf(P)$ (Example 6.26)

Immediate Occurrence Probabilities

We assume a process $P \in \mathcal{L}_{\mathcal{CC}} \cap \mathcal{L}_{unf}$, the unfolding $Unf(P) = (D, E, \prec, l_E, l_D, \lambda_D)$ and a fixed event $e \in E$ for which we want to derive the occurrence probability, $\Omega(e)$. Event e occurs immediately when all conditions $d \in {}^\bullet e$ have become active. So, the probability that e occurs is equal to the probability that all $d \in {}^\bullet e$ are active at one moment in time.

We first want to derive the IOP for e. Therefore, we assume that all events $g \in {}^{\bullet\bullet}e$ will happen. Under this assumption, the probability whether e will happen or not depends on whether the conditions in ${}^\bullet e$ will be active at the same time or not. This depends not only on ${}^\bullet e$, but on all conditions that are in a choice relation with conditions in ${}^\bullet e$.

A condition $d \in {}^\bullet e$ is prevented from becoming active or is deactivated when an event happens that is in conflict with d or which depends on d.

The latter case can only happen when the event that deactivates d is event e itself. If there is another event $e' \in d^\bullet$ which has happened, then it is in conflict with e, and, as we have seen in the proof of Theorem 6.23, e as well as e' must denote synchronisations over identical actions. We have assumed earlier that all events in ${}^{\bullet\bullet}e$ have happened already. If e' is not in conflict with one of the events $g \in {}^{\bullet\bullet}e$, then both e' and e can still happen. This is the situation we have depicted in Figure 6.5 (b), and which implies nondeterminism. Since we assumed the considered process to be s-deterministic, we can conclude that e' must be in conflict with one of the events $g \in {}^{\bullet\bullet}e$. To conclude, a condition $d \in {}^\bullet e$ can only be deactivated by the occurrence of an event that is in conflict with d.

Since we assume that $g \in {}^\bullet d$ has happened, the only nodes that can be in conflict with d are the conditions $d' \in \bigstar d$ and their descendants with respect to $\leq$.

Example 6.27

We reconsider Example 6.22 (Figure6.4). We want to derive the occurrence probability of event e_4, the only synchronisation event in the unfolding. We assume that e_2 and e_5 will happen. Event e_4 depends on the set of conditions ${}^\bullet e_4 = \{d_5, d_{11}\}$.

The conditions that can prevent e_4 from happening are $\bigstar d_5 = \{d_3, d_4\}$, and $\bigstar d_{11} = \{d_9, d_{12}\}$.

The condition d_3 is in choice with conditions d_4 and d_5. Hence, when event e_1 or e_3 happens, then d_5 would be deactivated. The situation is similar for conditions d_9 and d_{12}.

Event e only happens when all conditions $d \in {}^\bullet e$ become active simultaneously. To capture this formally, we define intervals I_d on the real line, which denote the time period for a condition d to be active. For event e to happen, the intersection of all I_d for $d \in {}^\bullet e$ must be non-empty. To define I_d, we first need to define a random variable Y_d^{off}, which denotes the time between the occurrence of the event ${}^\bullet d$ and the deactivation of d by conflicting events.

The time until a condition d gets deactivated is the minimum of the occurrence times of all events that are in conflict with d. In case that there are no such events, Y_d^{off} equals ∞.

Definition 6.28 (Y_d^{off}) Let d be a condition. Then the random variable Y_d^{off} is defined as

$$Y_d^{off} = \min_{\substack{d' \in \bigstar d \\ e' \in d'^\bullet}} \{X_{e'}, \infty\}.$$

Note that ∞ is in the set because $\bigstar d$ can be empty. In that case, d can never be deactivated, once it is activated.

Definition 6.29 (I_d) Let d be a condition with $g \in {}^\bullet d$. Then the left-closed interval I_d is defined as

$$I_d = \begin{cases} \left[X_g + Y_d, Y_d^{off}\right) & \text{if } X_g + Y_d \leq Y_d^{off}, \\ \emptyset & \text{otherwise.}^{8} \end{cases}$$

Example 6.30

We continue Example 6.22. For condition $d_5 \in {}^\bullet e_4$, we have random variable $Y_{d_5}^{off} = \min\{X_{e_2} + Y_{d_3}, X_{e_2} + Y_{d_4}\}$. The event preceding d_5 is e_2, so then

$$I_{d_5} = \left[X_{e_2} + Y_{d_5}, Y_{d_5}^{off}\right)$$

For the other side, we have $Y_{d_{11}}^{off} = \min\{X_{e_5} + Y_{d_9}, X_{e_5} + Y_{d_{12}}\}$, and $e_5 \in {}^\bullet d_{11}$. Then

$$I_{d_{11}} = \left[X_{e_5} + Y_{d_{11}}, Y_{d_{11}}^{off}\right)$$

---

[8] We distinguish the two cases here since the interval would not be defined for the case $X_g + Y_d > Y_s^{off}$, I_d.

We are now ready to define the IOP for event e. We denote this probability as $\Omega(e|^{\bullet\bullet}e)$.

Definition 6.31 (Immediate Occurrence Probability) The probability $\Omega(e|^{\bullet\bullet}e)$ is defined as

$$\Omega(e|^{\bullet\bullet}e) = \Pr\left(\left(\bigcap_{d\in{}^{\bullet}e} I_d\right) \neq \emptyset\right).$$

Since we assume maximal-progress, event e happens as soon as all conditions $d \in {}^{\bullet}e$ are active. Hence, event e happens at time $t = \inf\left(\bigcap_{d\in{}^{\bullet}e} I_d\right)$, given that $\bigcap_{d\in{}^{\bullet}e} I_d$ is not empty.

Note that we can express $\Omega(e|^{\bullet\bullet}e)$ also as

$$\Omega(e|^{\bullet\bullet}e) = \Pr\left(\max_{d\in{}^{\bullet}e}\{X_{{}^{\bullet}d} + Y_d\} \leq \min_{d\in{}^{\bullet}e}\left\{Y_d^{off}\right\}\right), \tag{6.1}$$

i.e., as the probability that the maximum of all left bounds of all intervals I_d are smaller than the minimum of all right bounds of these intervals. This formulation of $\Omega(e|^{\bullet\bullet}e)$ is easier to deal with than referring to the interval notation, hence, in the following we will use it exclusively.

The quantity $\Omega(e|^{\bullet\bullet}e)$ is the probability that all intervals I_d for $d \in {}^{\bullet}e$ have a non-empty intersection. Since I_d is the interval in which condition d is active, $\Omega(e|^{\bullet\bullet}e)$ is also the probability that all conditions are simultaneously active for some time, thus enabling event e. Due to the maximal-progress assumption, e actually happens in that case, hence $\Omega(e|^{\bullet\bullet}e)$ is also the probability that e occurs, under the assumption that all events $e' \in {}^{\bullet\bullet}e$ occur.

Example 6.32

For Example 6.22, $\Omega(e_4|^{\bullet\bullet}e_4) = \Pr((I_{d_5} \cap I_{d_{11}}) \neq \emptyset)$, or, written as in Equation 6.1,

$$\begin{aligned}\Omega(e_4|^{\bullet\bullet}e_4) &= \Pr\Big(\max\{X_{e_2} + Y_{d_5}, X_{e_5} + Y_{d_{11}}\} \\ &\qquad \leq \min\left\{Y_{d_5}^{off}, Y_{d_{11}}^{off}\right\}\Big) \\ &= \Pr\left(\max\{X_{e_2} + Y_{d_5}, X_{e_5} + Y_{d_{11}}\}\right. \qquad (6.2)\\ &\qquad \left.\leq \min\{X_{e_2} + Y_{d_4}, X_{e_2} + Y_{d_3}, X_{e_5} + Y_{d_9}, X_{e_5} + Y_{d_{12}}\}\right).\end{aligned}$$

Since d_5 and d_{11} correspond to fragments with synchronising action a, $Y_{d_5} = Y_{d_{11}} = 0$. Hence, Equation (6.2) simplifies to

$$\Omega(e_4|^{\bullet\bullet}e_4) = \Pr\left(\max\{X_{e_2}, X_{e_5}\} \leq \min\{X_{e_2} + Y_{d_4}, X_{e_2} + Y_{d_3}, X_{e_5} + Y_{d_9}, X_{e_5} + Y_{d_{12}}\}\right) \tag{6.3}$$

Deriving the occurrence probabilities $\Omega(e)$

From the IOPs, we can now derive $\Omega(e)$, the unconditional occurrence probability of e. Intuitively, the occurrence probability of an event e is the probability that all choices that are relevant for the occurrence of e are actually decided in favour of e. Formally, we define this probability as follows:

Definition 6.33 (Occurrence Probability) Let e be an event and $G_e = \bigcup_{g \in {}^{\bullet\bullet}e}[g]$. Then the occurrence probability of e, $\Omega(e)$, is defined as

$$\Omega(e) = \Omega(e|^{\bullet\bullet}e) \cdot \prod_{e' \in {}^{\bullet\bullet}e} \Omega(e') \tag{6.4}$$

$$= \Omega(e|^{\bullet\bullet}e) \cdot \prod_{e' \in G_e} \Omega(e'|^{\bullet\bullet}e') \tag{6.5}$$

$$= \prod_{e' \in [e]} \Omega(e'|^{\bullet\bullet}e') \tag{6.6}$$

Equation (6.4) is the probability that event e happens under the condition that all predecessor events have happened, *i.e.*, the IOP of e, multiplied with the probability that all preceding events $e' \in {}^{\bullet\bullet}e$ have happened. The latter probability can be expressed as the product of all IOPs of all events $e' \in G_e$ (Equation (6.5)). Therefore, the occurrence probability of event e is the product of all IOPs of the events $e' \in [e]$ (Equation (6.6)).

Example 6.34

We reconsider Example 6.22. We want to compute $\Omega(e_4)$. We therefore need the IOPs $\Omega(e'|^{\bullet\bullet}e')$ for $e' \in [e_4]$, e for all events $\{e_2, e_5, e_4\}$. For e_2 and e_5, this is easy: both events happen with probability 1.

From Equation (6.3) we know $\Omega(e_4|^{\bullet\bullet}e_4)$, and therefore, $\Omega(e_4) = \Omega(e_4|^{\bullet\bullet}e_4)$.

Example 6.35

We consider the processes

$$P_1 \stackrel{\text{def}}{=} [\mu].\mathsf{stop} + [\lambda].[\delta].P_2 \qquad P_2 \stackrel{\text{def}}{=} [\zeta].\mathsf{stop} + a.\mathsf{stop} \qquad Q_1 \stackrel{\text{def}}{=} [\alpha].\mathsf{stop} + [\beta].[\mu].a.\mathsf{stop}$$

and $R \stackrel{\text{def}}{=} P_1 \|_{\{a\}} Q_1$. In Figure 6.7 the unfolding $Unf(R)$ is depicted. Both P_1 and Q_1 can only do one synchronisation, if at all. This is represented by event e_8 in the

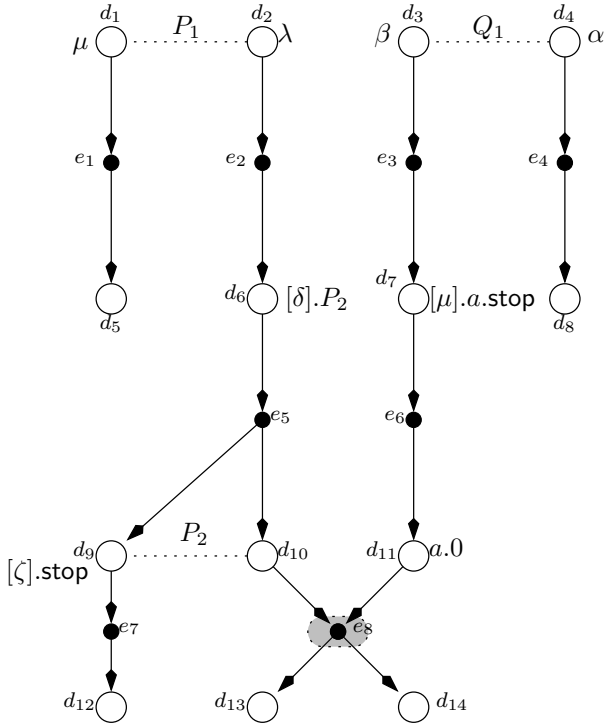


Figure 6.7: Unfolding of R (Example 6.35)

unfolding. What is the probability $\Omega(e_8)$? To compute this, we need the IOPs for all events $e \in [e_8] = \{e_2, e_3, e_5, e_6, e_8\}$.

We first compute $\Omega(e_2|^{\bullet\bullet}e_2)$. The only condition e_2 depends on immediately, is d_2, and d_2 is in choice with d_1. We must therefore determine the probability that d_2 is not deactivated by the occurrence of e_1. This probability is also the probability $\Omega(e_2|^{\bullet\bullet}e_2)$. In this case,

$$\Omega(e_2|^{\bullet\bullet}e_2) = \Pr\left(Y_{d_2} \leq Y_{d_2}^{\mathit{off}}\right),$$

where $Y_{d_2}^{\mathit{off}} = X_{e_1} = Y_{d_1}$. Since we consider exponential distributions,

$$\Omega(e_2|^{\bullet\bullet}e_2) = \frac{\lambda}{\lambda + \mu}.$$

By analogous considerations, we can derive

$$\Omega(e_3|^{\bullet\bullet}e_3) = \frac{\beta}{\alpha + \beta}.$$

Since both events only have $\bot$ in their preset, also $\Omega(e_2) = \Omega(e_2|^{\bullet\bullet}e_2)$ and $\Omega(e_3) = \Omega(e_3|^{\bullet\bullet}e_3)$.

Since e_5 and e_6 only depend on d_6 and d_7, respectively, they must happen, once e_2 and e_3 have happened. Therefore, $\Omega(e_5|^{\bullet\bullet}e_5) = \Omega(e_6|^{\bullet\bullet}e_6) = 1$. An immediately consequence is that $\Omega(e_5) = \Omega(e_2|^{\bullet\bullet}e_2) = \frac{\lambda}{\lambda+\mu}$, and $\Omega(e_6) = \Omega(e_3|^{\bullet\bullet}e_3) = \frac{\beta}{\alpha+\beta}$.

Deriving $\Omega(e_8|^{\bullet\bullet}e_8)$ is slightly more difficult. Here we have the case that, if e_5 and e_6 happen, the occurrence of e_8 can be inhibited by the occurrence of event e_7. Therefore, we have to find the probability that e_8 occurs before e_7, *i.e.*, $\Omega(e_8|^{\bullet\bullet}e_8)$. According to the definition,

$$\Omega(e_8|^{\bullet\bullet}e_8) = \Pr\left(\max_{d\in{}^\bullet e_8}\{X_{\bullet d}+Y_d\} \leq \min_{d\in{}^\bullet e_8}\left\{Y_d^{off}\right\}\right). \tag{6.7}$$

By elementary arithmetic with max and min operators, we can rewrite Equation (6.7) as

$$\begin{aligned}
\Omega(e_8|^{\bullet\bullet}e_8) &= \Pr\left(\max\{X_{e_5}+Y_{d_{10}}, X_{e_6}+Y_{d_{11}}\} \leq \min\{\infty, X_{e_5}+Y_{d_9}\}\right)\\
&= \Pr\left(\max\{X_{e_5}, X_{e_6}\} \leq X_{e_5}+Y_{d_9}\right)\\
&= \Pr(X_{e_6}\leq X_{e_5}) + \Pr(X_{e_6} > X_{e_5})\Pr(X_{e_6}\leq X_{e_5}+Y_{d_9})\\
&= \Pr(X_{e_6}\leq X_{e_5}) + \Pr(X_{e_5} < X_{e_6} \leq X_{e_5}+Y_{d_9}).
\end{aligned}$$

To conclude,

$$\begin{aligned}
\Omega(e_8) &= \Omega(e_8|^{\bullet\bullet}e_8)\cdot\Omega(e_5|^{\bullet\bullet}e_5)\cdot\Omega(e_6|^{\bullet\bullet}e_6)\cdot\Omega(e_2|^{\bullet\bullet}e_2)\cdot\Omega(e_3|^{\bullet\bullet}e_3)\\
&= \frac{\beta}{\alpha+\beta}\frac{\lambda}{\lambda+\mu}\left(\Pr(X_{e_6}\leq X_{e_5}) + \Pr(X_{e_5} < X_{e_6} \leq X_{e_5}+Y_{d_9})\right).
\end{aligned}$$

6.4.3 Special Cases

In the previous section, we have derived the occurrence probabilities $\Omega(e)$ for an event e in its most general form.

According to Equation (6.1),

$$\Omega(e|^{\bullet\bullet}e) = \Pr\left(\max_{d\in{}^\bullet e}\{X_{\bullet d}+Y_d\} \leq \min_{d\in{}^\bullet e}\left\{Y_d^{off}\right\}\right).$$

Written differently,

$$\Omega(e|^{\bullet\bullet}e) = \Pr(Z_e \leq 0),$$

where Z_e is a random variable defined on the real numbers, defined as

$$Z_e = \max_{d\in{}^\bullet e}\{X_{\bullet d}+Y_d\} - \min_{d\in{}^\bullet e}\left\{Y_d^{off}\right\}. \tag{6.8}$$

We then can write

$$\Omega(e|^{\bullet\bullet}e) = F_{Z_e}(0) = \int_{-\infty}^{0} f_{Z_e}(t)dt,$$

where F_{Z_e} is the distribution of Z_e and f_{Z_e} its density.

Computing F_{Z_e} is usually very difficult. It depends on the distributions of the occurrence times $X_{e'}$ of the events $e' \in {}^{\bullet\bullet}e$. These are usually not independent from each other, and, since they are phase-type distributions, their representations can become quite large. Furthermore, computing F_{Z_e} from the right-hand-side of Equation (6.8) is not an easy task, either. However, there are special cases, where Equation (6.8) becomes much easier to handle. These cases are *internal choice* and *synchronisation without timeouts*, as will be discussed below.

Internal Choice

We consider an event e that

1. is not a synchronisation, and
2. for $\{d\} = {}^{\bullet}e$, the set
$$\bigcup_{d' \in \bigstar d} d'^{\bullet}$$
contains only events that are not synchronisations.

These two conditions ensure that the time until deactivation of condition d, Y_d^{off}, is determined solely by the random variable $Y_{d'}$ for $d' \in \bigstar d$. Since we agreed on the restriction to only consider processes that are elements of $\mathcal{L}_{CC}$, the conditions $\{d\} \cup \bigstar d$ are all pairwise in a choice relation. The choice is resolved by the occurrence of a non-synchronisation event. Hence, this situation describes an *internal choice*.

For this special case, Equation (6.8) can be simplified. The set ${}^{\bullet\bullet}e$ is a singleton, say, $\{g\}$. Then we have:

$$\begin{aligned} Z_e &= \max_{d' \in {}^{\bullet}e} \{X_{{}^{\bullet}d'} + Y_{d'}\} - \min_{d' \in {}^{\bullet}e} \left\{ Y_{d'}^{off} \right\} \\ &= X_g + Y_d - Y_d^{off} \\ &= X_g + Y_d - \min_{\substack{d' \in \bigstar d \\ e' \in d'^{\bullet}}} \{X_{e'}\}. \end{aligned}$$

We can rewrite the minimum expression, and yield

$$\begin{aligned} Z_e &= X_g + Y_d - \min_{d' \in \bigstar d} \{X_g + Y_{d'}\} \\ &= Y_d - \min_{d' \in \bigstar d} \{Y_{d'}\} \end{aligned}$$

Hence,

$$\Omega(e|{}^{\bullet\bullet}e) = \Pr(Y_d - \min_{d' \in \bigstar d} \{Y_{d'}\} \leq 0).$$

This is the probability that one exponentially distributed random variable is smaller than another exponentially distributed random variable. This probability can be expressed solely by the rates of the respective distributions. This means that

$$\Omega(e|^{\bullet\bullet}e) = \frac{\lambda_D(d)}{\lambda_D(d) + \sum_{d' \in \bigstar d} \lambda_D(d')} \tag{6.9}$$

Example 6.36

We consider the process $P \stackrel{\text{def}}{=} [1].\mathsf{stop} + [2].\mathsf{stop} + [3].\mathsf{stop}$. The unfolding of this prefix is finite and depicted in Figure 6.8 All three events e_1, e_2, e_3 are not synchronisations,

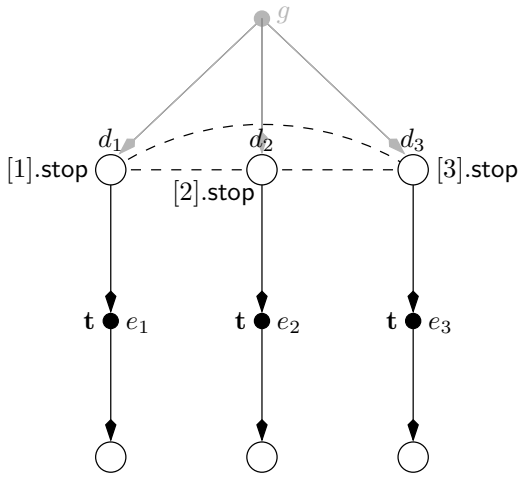


Figure 6.8: Unfolding for Example 6.36

and $\lambda_D(d_i) = i$ for $i = 1, 2, 3$. In this example, event $g = \bot$.

We then have

$$\Omega(e_1|^{\bullet\bullet}e_1) = \frac{\lambda_D(d_1)}{\lambda_D(d_1) + \lambda_D(d_2) + \lambda_D(d_3)} = \frac{1}{1+2+3} = \frac{1}{6}.$$

Accordingly, $\Omega(e_2|^{\bullet\bullet}e_2) = \frac{1}{3}$, and $\Omega(e_3|^{\bullet\bullet}e_3) = \frac{1}{2}$.

Synchronisation without Timeouts

The second special case we want to consider is when event e is synchronising, but for $d \in {}^\bullet e : \bigstar d = \emptyset$. In this case, once condition $d \in {}^\bullet e$ has been activated, it never gets deactivated, except by the occurrence of e. In terms of random variables, we have $Y_d^{off} = \infty$. Then, Equation (6.1) simplifies to

$$\Omega(e|^{\bullet\bullet}e) = \Pr\left(\max_{d \in {}^\bullet e}\{X_{{}^\bullet d} + Y_d\} \leq \infty\right) = 1.$$

According to Definition 6.33, we then have

$$\Omega(e) = \Omega(e|^{\bullet\bullet}e) \cdot \prod_{e' \in [g]} \Omega(e'|^{\bullet\bullet}e') = \Omega(g),$$

where $\{g\} = {}^{\bullet\bullet}e$.

The special cases we have treated here are not unfamiliar. Both have already appeared in Chapter 4. The restriction to internal choice and synchronisations without timeouts ensured that the branching probabilities within components can be determined locally (*cf.* Section 4.3.1).

6.4.4 Concluding Remarks

In this section, we have defined the occurrence probabilities for events. Basically, we first have computed probability distributions on sets of events whose preceding conditions are in a choice relation. These probabilities are valid for the case that the resolution of this choice is imminent. Then, we have defined the occurrence probability of an event e to be the probability that all choices that are relevant for the occurrence of e are actually decided in favour of e. This is therefore the product of all individual probabilities of all events e' in e.

This approach has a close resemblance to the one chosen by Katoen in [84]. There, for bundle event structures a stochastic extension is defined where probability distributions were assigned to sets of mutually conflicting events, so-called *clusters*. Events in clusters are required to denote local activity. Therefore, in terms of this section, only local probabilistic choices have been modelled. It is not mandatory to assign *all* events a probability. Thus the description of nondeterminism is possible. However, when we assume the special case where all events carry a probability, then for all configurations of such an event structure a probability can be derived, by simple multiplication of the probabilities of all the events in this configuration.

The similarities to our approach for the unfolding are obvious. We also derive probabilities for individual events (the IOPs). With Definition 6.33, for an event e we multiply the

IOPs of all events in the local configuration $[e]$ to obtain the occurrence probability for e. The difference is that the individual probabilities that we assigned to events are not given deliberately, but are rather already implicitly determined by the *stochastic temporal* behaviour of the whole process.

We should point out that the probabilities $\Omega(e)$, although computed by means of the local configuration $[e]$, is *not* the probability that the state $St([e])$ is ever entered. We demonstrate this with the following example.

Example 6.37

We consider the two events e_1 and e_2 in part (a) of Figure 6.9.

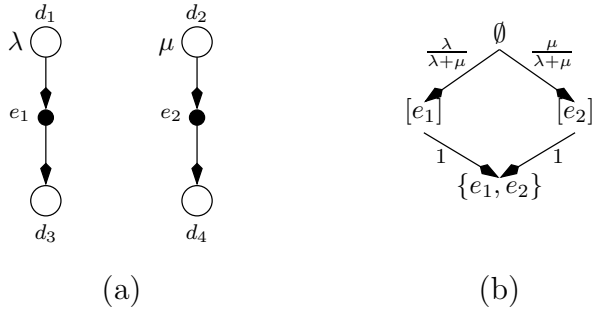


Figure 6.9: Illustration of Example 6.37

The transition system defined on the sets of configurations, together with the branching probabilities, is shown in part (b). The probability that the local configuration $[e_1]$ is reached is $\frac{\lambda}{\lambda+\mu}$, but the probability that e_1 will happen is 1.

The occurrence probability of event e, $\Omega(e)$ is the probability that a state of the associated transition system is entered via a transition with label $l_E(e)$. So $\Omega(e)$ is the probability that the considered system evolves along a path through the transition system that has a transition in it that is labelled with the indexed action $l_E(e)$.

6.5 Waiting Times

In Section 6.3, we have defined for an event e of an unfolding its event occurrence time X_e. We can use these occurrence times to characterise an important performance measure that we have identified in Section 4.4, namely waiting times.

We proceed in three steps: first, we define the waiting time for a component under the assumption that one specific synchronisation event occurs. We call this the *individual*

waiting time (IWT). Second, we define the waiting time for a component under the assumption that the considered component *starts* waiting with the occurrence of a specific event e. We call this a *waiting period*. Third, we define the (mean) waiting time for a certain local state of a component. We will see that these three quantities are related: we need individual waiting times to define waiting periods, and we need waiting periods to define waiting times.

6.5.1 Individual Waiting Times

We consider a process P and its unfolding $Unf(P) = (D, E, \bigstar, \prec, l_D, l_E, \lambda_D)$. Remember that events e with $\#^\bullet e > 1$ are synchronisation events. Let us assume that e is a synchronisation event. For $d \in {}^\bullet e$, with $loc(d) = i$, the instance at which $e' \in {}^\bullet d$ happens denotes the start of a waiting period for location i. Since $Y_d = 0$ with probability 1 for $d \in {}^\bullet e$ (since d denotes a synchronising fragment), the *individual waiting time* of component i for event e, $IW_{e,i}$, is defined as

$$IW_{e,i} = \max_{g \in {}^{\bullet\bullet}e} \{X_g\} - X_{e'}.$$

Since $e' \in {}^{\bullet\bullet}e$, we obtain

$$IW_{e,i} = \max_{g \in {}^{\bullet\bullet}e} \{X_g - X_{e'}, 0\}$$

Individual waiting times are hence positive or zero.

As the above derivation shows, individual waiting times can always be described as

$$\max\{X_1, \ldots, X_n\} - X_j,$$

where $\{X_i \mid i = 1, \ldots, n\}$ is a set of (possibly dependent) positive random variables. Each X_i for $i \in \{1, \ldots, n\}$ describes the time at which location i becomes ready to synchronise. Hence, the random variable $\max\{X_1, \ldots, X_n\}$ is the enabling time of the considered synchronisation, and $\max\{X_1, \ldots, X_n\} - X_j$ is the time that has to pass between the instance component j becomes ready to synchronise, and the actual synchronisation.

6.5.2 Waiting Periods

The waiting periods we have defined in Section 6.5.1 are *individual* waiting times: they are only about *one* instant, one specific situation of a synchronisation. The reason for this is that we have chosen a synchronisation event as the reference for the IWT. A synchronisation event denotes the end of a waiting period, but usually there is more than only one event that ends the same waiting period. We will see this in the next example. However, a waiting period of a location i is started by *exactly one* event, say, $e \in E$. Let $F = \{e_j | j \in J\} \subseteq E$ (J being an index set) be the set of events that end the waiting period started by e. The local configurations $[e_j]$ of the events $e_j \in F$ for $j \in J$ all have the event e in common, *i.e.*, $e \in \bigcap_{j \in J}^{\infty} [e_j]$. To find out all about the waiting period started by e, we must take all events $e_j \in F$ for $j \in J$ into account.

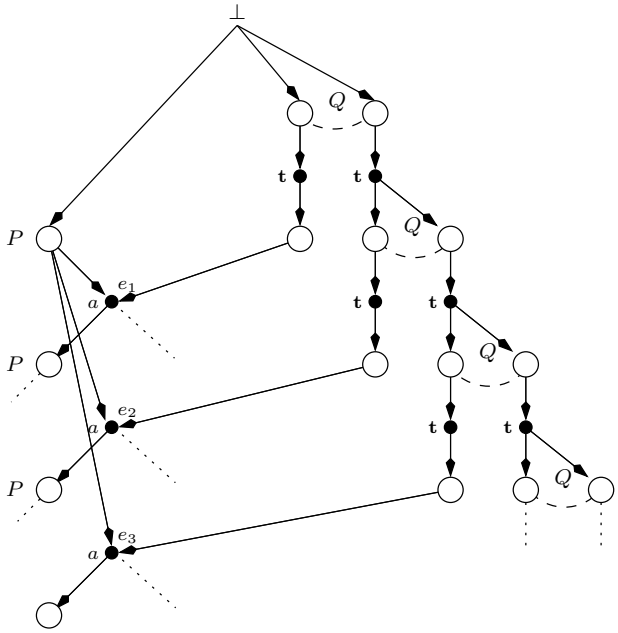


Figure 6.10: Unfolding of $P\|_a Q$ (Example 6.38)

Example 6.38

We assume two processes P and Q as follows:

$$\begin{aligned} P &\stackrel{\text{def}}{=} a.P \\ Q &\stackrel{\text{def}}{=} [].a.Q + [].Q \end{aligned}$$

An (incomplete) sketch of the unfolding of the process $P\|_a Q$ is depicted in Figure 6.10. The start event, $\bot$, denotes also the start of a waiting period: component 1 (P) waits for a synchronisation with component 2 (Q) over a. As we can see in Figure 6.10, this waiting period is ended by the events $e_1, e_2, e_3, \ldots$. Actually, there are infinite many events that end the waiting period started with $\bot$.

The events $e_j \in F$ can neither be independent nor causally related. Hence, they must be in conflict. Let $p(e_j)$, $j \in J$ be the probability that event e_j happens, under the condition, that e has already happened. Since all events $e_j \in F$ causally depend on e, the occurrence of one of them implies already the occurrence of e, and therefore, the probability $\Pr(e_j \text{ and } e \text{ happen})$ equals $\Pr(e_j \text{ happens}) = \Omega(e_j)$. Then, from the definition of conditional probabilities, we can derive

$$p(e_j) = \Pr(e_j \text{ happens} \mid e \text{ happens}) = \frac{\Pr(e_j \text{ and } e \text{ happen})}{\Pr(e \text{ happens})} = \frac{\Omega(e_j)}{\Omega(e)}.$$

For each of the events e_j, $j \in J$, we can compute the IWT $IW_{e_j,i}$, as defined in Section 6.5.2. Then, the waiting time of component i under the assumption that event e has started the waiting period, can be derived as

$$WP_e = \sum_{j \in J} p(e_j) IW_{e_j,i} = \frac{1}{\Omega(e)} \sum_{j \in J} \Omega(e_j) IW_{e_j,i}$$

6.5.3 Mean Waiting Times

The waiting periods WP_e, as derived in the previous section, are in terms of events only. Our aim is to express the mean waiting times introduced in Section 4.4, but these refer to local, synchronising states. In this section, we establish a relation between (mean) waiting times and waiting periods.

We assume now a synchronising state s of a component i. Generally, the events e that denote the start of a waiting period also relate to local synchronising states: they denote the time instance where such a state is entered. Since we assume irreducibility, for state s there are infinitely many events in the unfolding which denote that this state is entered. We denote this set of events as M_s. For each $e \in M_s$, a waiting period WP_e can be described. It is obvious that these quantities must be combined somehow to yield the (mean) waiting time W_s of local state s. The questions are now:

1. Which events $e \in M_s$ should be chosen such that the quantities WP_e can be combined?
2. How should the WP_e be combined?

We give answers to these questions in the following two paragraphs.

Choosing the Right Events

Let $e, e' \in M_s$. Events e and e' can not be independent from each other: a component can not enter one of its states twice without a dependency between these two events.

Hence, e and e' are either in conflict or causally related. We define now a relation $\lhd_s \subseteq (M_s \cup \{\bot\}) \times (M_s \cup \{\bot\})$ as follows: for $e, e' \in M_s \cup \{\bot\}$,

$$e \lhd_s e' \quad \Longleftrightarrow \quad e < e' \text{ and there is no other event } e'' \in M_s \text{ such that } e < e'' < e'.$$

We denote by $\trianglelefteq_s$ the transitive closure of $\lhd_s$.

Lemma 6.39 Let $P \in \mathcal{L}_{unf} \cap \mathcal{L}_{\mathcal{CC}}$ be an s-deterministic, irreducible process. Let s be a local, synchronising state of a component i of P, and let M_s be defined as before. Then, the relation $\lhd_s$ defines a tree with root $\bot$.

Proof: Event $\perp$ is the root, since every element of the unfolding $Unf(P)$ depends on $\perp$ by definition. $\lhd_s$ is also acyclic, since $<$ is acyclic. Remains to show that each $e \in M_s$ has exactly one predecessor with respect to $\lhd_s$. Assume that this is not the case, *i.e.,* there is an event $e \in M_s$ with two events e', e'' such that $e' \lhd_s e$ and $e'' \lhd_s e$. For e to happen, both e' and e'' must happen before. That means that both events must be independent, which we have ruled out before. Therefore, an event $e \in M_s$ can only have one predecessor. $\rightarrow\bullet$

As a consequence, if we have two events $e, e' \in M_s$, and neither $e \trianglelefteq_s e'$ nor $e' \trianglelefteq_s e$, then e and e' must be in conflict. This has interesting consequences. Let $M_s^l = \{e \in M_s \mid e \text{ is a node of depth } l \text{ in } M_s\}$. Remember now that $\Omega(e)$ denotes the occurrence probability of event e (*cf.* Section 6.4). Ω is not a probability measure on the whole set of events E, but Ω, restricted to the sets M_s^l, *is* a probability measure.

Lemma 6.40 Let $P \in \mathcal{L}_{unf} \cap \mathcal{L}_{\mathcal{CC}}$ be an s-deterministic, irreducible process. Let s be a local, synchronising state of a component i of P, and let, for $l \in \mathbb{N}$, M_s^l be defined as before. Then $\Omega\big|_{M_s^l}$ is a probability measure on M_s^l.

Proof: By induction on the depth of the tree:

$\mathbf{l = 0}$: $M_s^0 = \{\perp\}$, and $\Omega(\perp) = 1$ by definition.

$\mathbf{l \rightarrow l + 1}$: We assume that the lemma has been shown for M_s^l. The events $e \in M_s^{l+1}$ depend on that of M_s^l. We have to prove the axioms for a probability measure.

1. By definition of Ω, $0 \leq \Omega(e) \leq 1$ for all events $e \in M_s^{l+1}$.
2. For $e \in M_s^l$, $\sum_{\{e' : e \lhd_s e'\}} \Omega(e') \leq \Omega(e)$, since for $e \lhd_s e'$, the probability that e' happens can not be larger than the probability that e happens, *i.e.,* $\Omega(e') \leq \Omega(e)$. Since the events $e' \in M_s^{l+1}$ are mutually exclusive, their probabilities do not "overlap", and therefore $\sum_{\{e' : e \lhd_s e'\}} \Omega(e') \leq \Omega(e)$ holds. Irreducibility ensures that, once a certain state has been entered, the probability that it will be entered again is equal to one. Hence, by deconditioning, we can conclude that equality holds, *i.e.,*
$$\sum_{\{e' : e \lhd_s e'\}} \Omega(e') = \Omega(e).$$
From this it follows immediately that
$$\begin{aligned} \sum_{e \in M_s^l} \sum_{\{e' : e \lhd_s e'\}} \Omega(e') &= \sum_{e' \in M_s^{l+1}} \Omega(e') \\ &= 1 \end{aligned}$$
by means of the induction hypotheses.

$\rightarrow\bullet$

Combining Waiting Periods

We have ordered the events starting a waiting period for a local synchronising state s ($e \in M_s$) in a tree, and we have shown that the function Ω, restricted on the events with equal depth l in this tree (M_s^l for $l \in \mathbb{N}$), is a probability distribution.

Each event $e \in M_s^l$ denotes the l-th instance of a waiting period of synchronising state s since system start. The events $e \in M_s^l$ all have different histories, and, most likely, different values for $E[WP_e]$, $e \in M_s^l$. Then, most naturally, the l-th waiting time of state s can be expressed as

$$WT_s^l = \sum_{e \in M_s^l} \Omega(e)\, WP_e$$

WT_s^l is a random variable, a weighted sum of waiting periods, WP_e for $e \in M_s^l$. The mean waiting time can then be expressed as $E[WT_s^l] = \sum_{e \in M_s^l} \Omega(e) E[WP_e]$.

The waiting times that we have defined in Section 4.4 were *steady-state* waiting times: we have assumed that the considered system has run for a very long (*i.e.,* infinitely long) time period. The quantities WT_s^l are quantities that only relate to *finite* system runs: at least the number of steps until one of the events in M_s^l happens is finite. Hence, the actual waiting time has to be defined as

$$WT_s = \lim_{l \to \infty} \sum_{e \in M_s^l} \Omega(e)\, WP_e, \tag{6.10}$$

and

$$E[WT_s] = \lim_{l \to \infty} \sum_{e \in M_s^l} \Omega(e) E[WP_e].$$

6.6 Event Structures for Waiting Times

In [16], an algorithm has been proposed to compute the mean waiting times for synchronising local states of a component. The approach is based on bundle event structures with stochastic extension. The ingredients used there are the same as we have defined in this chapter: the algorithm relies on the computation of mean event occurrence times, event occurrence probabilities, and individual waiting times. However, the problem was not addressed how to exactly combine the individual waiting times

In this chapter, we have gathered enough knowledge about the properties of stochastic event structures to fill in the spaces that were left blank in [16].

We first give a renewed version of the algorithm given in [16] (Algorithm 6.1) before we comment on it.

· Input: a process $P \in \mathcal{L}_{unf}$,
a location i
a synchronising state s of location i
a number $l \in \mathbb{N}$.

· Output: approximated mean waiting time $\widehat{E[WT_s]}$ of s.

1. Compute the unfolding $\mathcal{E} = (D, E, \prec, l_D, l_E, \lambda_D) = Unf(P)$ of P;
2. Derive $M_s^l \subseteq E$;
3. $W := 0$
4. **forall** $e \in M_s^l$ **do**
5. Compute WP_e;
6. Compute $\Omega(e)$;
7. $W := W + \Omega(e) \cdot WP_e$
8. **end**
9. $\widehat{E[WT_s]} := W$

Algorithm 6.1: Deriving W_s from $Unf(P)$.

6.6.1 The Algorithm

In the following, we describe Algorithm 6.1:

Input and Output: The algorithm computes the waiting time of the local synchronising state s of component i from the unfolding $Unf(P)$ of $P \in \mathcal{L}_{unf}$. The value l determines the depth of the tree of events that is defined by $\lhd_s$.

Line 1. Compute the unfolding of P.

Line 2. The set M_s^l is the set of events which are of depth l in the tree defined by the relation $\lhd_s$.

Line 3. W is an auxiliary real variable, set to zero at the beginning of the subsequent loop.

Line 4. Loop: computations take place for all events $e \in M_s^l$.

Line 5. Compute the mean waiting period that is started by event e.

Line 6 Compute the probability that e occurs, *i.e.,* that the waiting period WP_e really starts.

Line 7. Update the weighted sum of the waiting periods that have occurred.

Line 8. End of loop.

Line 9. Output $\widehat{E[WT_s]}$.

We have defined the mean waiting time for a local synchronising state s to be $E[WT_s] = \lim_{l\to\infty} \sum_{e\in M_s^l} \Omega(e)\, WP_e$. In finite time only an approximation of the actual value can be computed, and so Algorithm 6.1 computes only an approximation of the waiting time of an synchronising state. The accuracy grows with the input parameter l. The larger l is, the deeper in the tree the computations take place and the nearer we are to the steady-state.

These considerations are nevertheless naive, since it assumes that the waiting periods and occurrence probabilities are already available. This is not the case, and therefore we have to look at this.

Waiting Periods

The algorithm combines different values for waiting periods. However, as we have seen, a waiting period WP_e is characterised by one starting event e, and possibly infinitely many stopping events. In case of infinitely many, it would only be possible to take finitely many of them into account. Therefore, a value for WP_e can only be an approximation.

Occurrence Probabilities

Even before we can compute an approximation of a mean waiting time, we must compute the probability with which the starting events of a waiting period do occur. This requires in general the knowledge of the complete distribution of the occurrence time of all events that are in choice with the event in question, as we know from Definition 6.31 and 6.33. Moreover, we must compute the distribution of the minimum of these occurrence times. This is usually not an easy task: the different occurrence times usually depend on each other, since they can share part of their history.

Unfoldings and Complete Prefixes

An unfolding of a recursive process is usually an infinite structure. To carry out actual computations on it, it must be represented finitely. In [96, 97], Langerak and Brinksma have defined an algorithm to compute a finite prefix of an unfolding that contains exactly

the information the unfolding contains (such a prefix is said to be complete). In [95], an alternative representation of the unfolding is described, which is based on the complete prefix approach. Finite representation of an unfolding is therefore no problem.

It would of course be desirable to do all computations only on the finite prefix. However, the question arises, if a complete finite prefix contains enough information to completely describe also the stochastic properties of the unfolding in question, such as occurrence probabilities and occurrence times.

As we have seen in Section 6.4, the probability $\Omega(e)$ of an event e to occur depends on the complete history of this event (*i.e.,* $[e]$) and on the history of the conditions that are in choice with ${}^\bullet e$. In a finite prefix, this history is only for those events completely available that are part of this prefix. In case that information is needed about events that are in the unfolding, but not in the current prefix, this is not the case. So a complete prefix might be not sufficient to describe occurrence probabilities or event enabling times.

We have found in Section 6.4.3 two special cases, however, where the choice between different conditions is independent of the history (local choice and synchronisation without timeouts). If we only consider processes which have these kinds of choice, the occurrence probabilities of events can possibly described by the information that is available in a complete prefix alone. Further research is, however, required to validate this assumption. For the time being, it has been shown only that the prefix is sufficient to express waiting times for the case that in the prefix occurs *no choice at all* [39].

6.6.2 Final Comments

From what we have derived in the previous section, we must conclude that the idea to compute mean waiting times from unfoldings is at least *questionable*. The algorithm does only make sense when the space complexity (*i.e.,* the memory consumption of a potential algorithm) is lower than an ordinary steady-state analysis of the considered SPA model.

Since we have defined waiting times as a limit (*cf.* Definition 6.10), the accuracy of the results that we can obtain by Algorithm 6.1 depends on the maximal depth of the tree defined by $\lhd_s$ that we allow the algorithm to consider for the computation. Moreover, each waiting period has possibly infinitely many stopping events, so that the accuracy of a waiting time depends also on the number of stopping events we take into account. A (non-trivial) tree has the unpleasant property that its breadth grows exponentially with the depth, hence, the higher the accuracy that we wish to obtain for waiting times, the longer the computations take, and the faster the memory limit is reached. Obviously, the upper bound for the complexity of such an approach can not be described by a polynomial.

6.7 Conclusions

In this chapter we have defined the following stochastic measures for unfoldings of $\mathcal{YAWN}$ processes:

- for each event e the time that passes from system start until event e happens, given that it happens at all;
- the probability that an event e actually happens in a system run;
- three different quantities, that are related to waiting times:

  individual waiting times, which denotes the time that one component has to wait until *one specific* synchronisation event happens, that ends the waiting;

  waiting periods, which denote the time that one component has to wait until *one of the possible* synchronisation events happens that ends the waiting;

  mean waiting times, which are a weighted sum of mean waiting periods that relate to the same local state in which the considered component waits.

We have defined a conceptual algorithm to compute mean waiting times for a local synchronising state of a component. The discussion of this algorithm has shown that an approach based on unfoldings to compute actual measures is not feasible.

Chapter 7

Conclusions

In this dissertation, we have considered Markovian stochastic process algebras (SPA). SPA are formalisms with well-defined semantics for the compositional specification of functional and performability models. The steady-state performability analysis of Markovian SPA specifications requires the solution of a system of linear equations. This is usually a complex task, since the number of equations grows exponentially with the size of the considered specification (state-space explosion).

The largeness problem is shared by many other specification formalisms for performance modelling, and there are many approaches known for these formalisms to avoid it. Some of these approaches have been adapted to SPA formalisms, either by translation, or by the identification of subclasses of SPA models that have advantageous properties for the more efficient solution technique. However, so far, there has been no genuine approache for SPAs that actually takes the compositional nature of model specifications as a starting point to overcome the largeness problem.

In this dissertation, we have taken first steps in the direction of a solution technique that exploits the structure of SPA specifications.

Outline of this Chapter. In Section 7.1, we summarise the original contributions of this dissertation. In Section 7.2, we come to some final conclusions. In Section 7.3, we give directions for future research.

7.1 Summary

7.1.1 The Stochastic Process Algebra $\mathcal{YAWN}$

In Chapter 3, we have introduced the stochastic process algebra $\mathcal{YAWN}$. $\mathcal{YAWN}$ is inspired by the work of Hermanns on IMC [66]. However, we have chosen a different method to

define the semantics of $\mathcal{YAWN}$. Instead of using labelled multi-transition systems to express the meaning of a $\mathcal{YAWN}$ specification, we have used ordinary labelled transition systems, endowed with an extra function that assigned rates to transitions. This approach facilitated the technical derivations in Chapter 4.

7.1.2 Global and Local Measures

In Chapter 1, we have argued that a solution approach that exploits the compositionality of SPA models, *i.e.,* that considers components in isolation, can only provide a subset of the performance measures that could be obtained from the ordinary CTMC steady-state probabilities. The measures that can be obtained must be *local* to the components, since treating components in isolation destroys information about dependencies between measures of different components. Hence, there is a price to pay: advanced efficiency causes reduced expressiveness.

In Chapter 4, we have formally defined what local measures are. We have expressed local steady-state probabilities and local throughputs in terms of the overall, *global* steady-state probabilities and throughputs. We have done this by means of projections, functions, that map states of the global state-space on states of components, *i.e.,* local states. Then, we have identified three important classes of quantities that must be known for a component-wise solution of SPA models: *local throughputs*, *branching probabilities*, and *waiting times*. Where the first two quantities are sufficient to allow the derivation of local steady-state probabilities for *non-synchronising states*, *i.e.,* states which only have outgoing transitions with associated rates, the knowledge of the third quantity would allow the derivation of local steady-state probabilities for *all* states.

7.1.3 SPA and Semi-Markov Chains

In Chapter 5, we have applied the knowledge that we have gained in Chapter 4. We have introduced a class of SPA processes, which have the so called **AWCI**-property. This class of processes, the so-called **AWCI**-processes, can be solved very efficiently. The main idea of the solution approach is to derive a semi-Markov chain (SMC) from the considered **AWCI**-process. The **AWCI**-property ensures that this is always possible. The SMC can be solved efficiently, and the measures of interest are the throughputs of the SMC states. These throughputs can be interpreted in the original **AWCI**-process, and can be used to derive steady-state probabilities for local, non-synchronising states. Moreover, for **AWCI**-processes it is also possible to efficiently derive mean waiting times for local, synchronising states, so that the derivation of steady-state probabilities for these states is also possible.

The core of the technique is the solution of the SMC. This requires, as we have shown, the efficient computation of the maximum of phase-type distributed random variables. With a naive approach, the computation of these quantities would require exponential

time and possibly exponential memory in the number of random variables. However, we have provided an algorithm that computes these quantities in only polynomial time and space requirements in the number of involved variables. As a consequence, the state-space explosion problem has been solved for this class of processes. The overall effort to compute measures is, however, still exponential in time in the worst case, although still much more efficient than ordinary steady-state analysis of the global Markov chain.

7.1.4 Event Structures and $\mathcal{YAWN}$

In Chapter 6, we have considered an event structure semantics for SPA. In Chapter 4, we have distinguished between local and global behaviour. This distinction is, however, not appropriately expressed in the standard semantics of $\mathcal{YAWN}$ *i.e.,* labelled transition systems (LTS). LTS have no explicit notion of locality. Event structures are, however, *true-concurrency* formalisms. They have an explicit notion of concurrence, and thus, for locality. In Chapter 6, we have investigated whether event structures offer advantages for the compositional solution of SPA models. We first have defined a semantics for a sub-language of $\mathcal{YAWN}$ that is based on *unfolding* [96]. Then we have investigated the stochastic properties of the event structures. We have defined two basic stochastic measures, first, the occurrence time of an event, counted from system start, and second, the probability that an event happens at all. We have shown that we can express the waiting times of synchronising states in terms of occurrence times and occurrence probabilities.

Then, we have reconsidered an algorithm that was first proposed in [16]. This algorithm computes approximations for the mean waiting times for synchronising states of a $\mathcal{YAWN}$ process. We have shown that the algorithm is not feasible.

7.2 Conclusions

In this dissertation, we have taken a first step towards the direction of a genuine approach for the compositional solution of stochastic process algebras models. Our approach to this problem has been very general. We first have derived the measures that could be obtained by such a solution technique. Then we have investigated why such a solution technique is not trivial: there are three classes of quantities that have to be known for a complete componentwise solution: *branching probabilities*, *throughputs*, and *waiting times* of synchronising states. If we consider a component C that we wish to solve, then these three types of quantities describe the influence of other components on C.

- The *branching probabilities* of synchronising states are unknown when there is a choice between local, timed transitions, and synchronising transitions. The question which one will be executed first (and disable the other) is decided by a race: the

synchronisation wins when all components that have to participate in the synchronisation become ready faster than the local transition. Stated the other way round, if the local transition is executed before the last synchronisation partner is are ready to synchronise, the synchronisation is disabled (the component has a *timeout*). Obviously, the question whether other components are faster than a local transition can only be answered if we take up the global view.

So, it comes as no surprise that we have required in Chapter 5 that the **AWCI**-processes must not have timeouts (property **A**), as described above. This restriction ensures that we do not have to derive branching probabilities for synchronising states. If the **AWCI**-technique can be enhanced such that timouts can be permitted can only be subject to speculation for now.

- *Throughputs* describe the pace at which the considered system evolves. Local throughputs depend linearly on the throughputs of other components, as we have seen in Chapter 4. Therefore, throughputs are global parameters.

- *Waiting times* describe the time that a component willing to synchronise has to wait until all other participants in the synchronisation are ready to do so. Hence, also waiting times are global parameters.

The three quantities are not independent from each other:

- In case that we know throughputs and waiting times, we can derive local steady-state probabilities all local states of a component.

- In case that we know branching probabilities and waiting times for a component, we can directly derive local throughputs and local steady-state probabilities for a component by ordinary CTMC analysis.

- In case that we know throughputs and branching probabilities, we can at least derive local steady-state probabilities for local, *non-synchronising* states.

As we can see, for a complete derivation of local steady-state probabilities, waiting times have always to be known. They can not be described by throughputs and branching probabilities. As a consequence, the derivation of waiting times should be given special attention. In Chapter 5, we have seen that it is possible to derive waiting times for **AWCI**-processes. The reason for this to work is that the starting distribution of the phase-type distributions that describe the waiting times can be obtained. However, in Chapter 6, we have tried to find a method to derive waiting times for more general processes, but we have failed. The complexity makes the chosen approach unfeasible. It is the big challenge for the componentwise solution of SPA models to find methods that compute or, at least, approximate waiting times of components.

7.3 Research Directions

As we have pointed out before, the derivation of waiting times and the determination of branching probabilities are a very challenging problem. However, the question, whether these quantities can be derived or approximated for the most general class of $\mathcal{YAWN}$ processes can not be answered yet. The considerations of Chapter 6 suggest that it is very likely that the task to compute waiting times is too complex to be actually carried out. It is well possible that the ordinary steady-state analysis of an SPA model is in general the most efficient technique to derive waiting times. In that case, bothering with them does not make sense, of course.

The technique to solve **AWCI**-processes, however, is a good starting point for further research. There are many open questions that have not been addressed in this dissertation. We have only considered steady-state analysis, but the question, whether the **AWCI**-processes can facilitate transient analysis is completely open. Another question is whether it is possible to allow more general than just exponential distributions. This would require a more advanced SPA than $\mathcal{YAWN}$ is, especially one that can express general distributions. For those SPA it is still possible to define the **AWCI** restrictions. In principle, the **AWCI**-technique should also work for these processes, even though the numerical difficulties would probably be much higher.

This dissertation can only be seen as a first step towards a component-based solution technique for SPA models. The future shows us if there will be a next step.

Part IV

Appendices

Appendix A

Miscellaneous

A.1 Notation

Many problems in this thesis can be conveniently expressed in matrix and vector notation. Vectors are underlined, *i.e.,* $\underline{v}$ denotes a vector. If not stated otherwise, we always assume row vectors. However, we allow three exceptions: first, for $n \in \mathbb{N}$,

$$\underline{1}_n = (\underbrace{1, \ldots, 1}_{n \text{ times}})^T$$

is the column vector of n ones. Second,

$$\underline{0}_n = \underline{1}_n - \underline{1}_n$$

is the column vector of n zeros. If the length of the vectors $\underline{0}_n$ and $\underline{1}_n$ becomes clear from the context, the subscript n is omitted. Third, the column vector $\underline{e}_i^n$ is defined as

$$\underline{e}_i^n = (0, \ldots, 0, \underbrace{1}_{i\text{-th position}}, 0, \ldots, 0)^T.$$

Again, if possible, the length n is omitted.

Matrices are always denoted with bold upper case letters, $\mathbf{A}, \mathbf{B}, \ldots$. For a matrix

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{1,m} \end{pmatrix}$$

we write $\mathbf{A} = (a_{ij})_{n,m}$.

If $\underline{v} = (v_1, v_2, \ldots, v_n)$ is a vector, then $\mathrm{diag}(\underline{v})$ is defined to be the diagonal matrix with vector $\underline{v}$ as diagonal.

A.2 Kronecker Products and Sums

Most material in this section is from [132].

The Kronecker product $\otimes$ of two matrices $\mathbf{D} = (d_{ij})_{n,m}$ and $\mathbf{E}$ (of arbitrary dimension) is defined as

$$\mathbf{D} \otimes \mathbf{E} := \begin{pmatrix} d_{1,1}\mathbf{E} & d_{1,2}\mathbf{E} & \cdots & d_{1,m}\mathbf{E} \\ d_{2,1}\mathbf{E} & d_{2,2}\mathbf{E} & \cdots & d_{2,m}\mathbf{E} \\ \vdots & & \ddots & \vdots \\ d_{n,1}\mathbf{E} & d_{n,2}\mathbf{E} & \cdots & d_{n,m}\mathbf{E} \end{pmatrix}$$

The Kronecker product is compatible with the normal scalar multiplication, *i.e.,* if we identify real numbers with 1×1 matrices, then $(r) \otimes M = M \otimes (r) = rM$, where $r \in \mathbb{R}$ and $M \in \mathbb{R}^{n \times m}$. The most important law is the following:

Lemma A.1 Let $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ be matrices, where the dimensions are such that the matrix products $\mathbf{AC}$ and $\mathbf{BD}$ are defined. Then

$$(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC} \otimes \mathbf{BD}) \tag{A.1}$$

An immediate consequence of this is that

$$r(\mathbf{A} \otimes \mathbf{B}) = (r\mathbf{A}) \otimes \mathbf{B} = \mathbf{A} \otimes (r\mathbf{B})$$

for $r \in \mathbb{R}$ and arbitrary matrices $\mathbf{A}$ and $\mathbf{B}$. Another consequence of Lemma A.1 is that, if $\mathbf{A}$ and $\mathbf{B}$ are square matrices, then

$$(\mathbf{A} \otimes \mathbf{B})^n = (\mathbf{A}^n \otimes \mathbf{B}^n). \tag{A.2}$$

Another operation on matrices, which is very important in the field of CTMCs, is the *Kronecker sum*:

Definition A.2 Let $\mathbf{A}$ be an $n \times n$ matrix and $\mathbf{B}$ and $m \times m$ matrix. Then the *Kronecker sum* of $\mathbf{A}$ and $\mathbf{B}$ is a $(nm) \times (nm)$ matrix defined by

$$\mathbf{A} \oplus \mathbf{B} := (\mathbf{A} \otimes \mathbf{I}_B) + (\mathbf{I}_A \otimes \mathbf{B}), \tag{A.3}$$

where $\mathbf{I}_A$ and $\mathbf{I}_B$ are the $n \times n$ and $m \times m$ unit matrices, respectively.

A.3 Max-Plus Algebra

The maximum operation on scalars, together with the usual addition, form an algebra. For $X, Y \in S$, where S is some linearly ordered set, we define $X \boxplus Y = \max\{X, Y\}$. $\boxplus$ is a monoid operation, *i.e.,* $\boxplus$ is associative, and there is a neutral element. On the real numbers, $-\infty$ is defined to be the neutral element.

Very confusing at first sight, we denote the usual addition of numbers as *product*, $\odot$.

A semifield is a set S endowed with two operations $+$ and $\cdot$ where $+$ is a monoid on S with neutral element ε and $\cdot$ defines a group on $S \setminus \{\varepsilon\}$ with neutral element e. $\cdot$ must be distributive with respect to $+$ and $\varepsilon \cdot e = e \cdot \varepsilon = \varepsilon$. A semifield is

1. idempotent, if $+$ is idempotent, *i.e.,* $a + a = a$, $\forall a \in S$;
2. commutative, if $\cdot$ is commutative.

$\mathbb{R} \cup \{-\infty\}$ with the operations $\boxplus$ and $\odot$ is an idempotent, commutative semifield with neutral element $-\infty$ for $\boxplus$ and 0 for $\odot$. When possible, we omit $\odot$, *i.e.,* we write ab instead of $a \odot b$. The division $a \odot b^{-1}$ (which actually is the substraction $a - b$ on numbers) is written as fraction $\frac{a}{b}$ (note that the fraction line is thicker than the usual one).

Appendix B

Stochastic Preliminaries

In this appendix, we describe the stochastic preliminaries of this dissertation. The material of this section (except Theorem B.5) is based on various textbooks about probability theory (*e.g.,* Feller [54]), stochastic processes (*e.g.,* Kulkarni [93], Howard [80], Çinlar [33]), and performance evaluation (*e.g.,* Haverkort [62], Harrison et al. [60]).

B.1 Sample Spaces and Probability Measures

Basic structure of probability theory is the sample space, Ω, together with a probability measure $\Pr : 2^{\Omega} \to [0, 1]$. Subsets of Elements of Ω are are said to be *events*. A probability measure on a sample space has to obey the following axioms:

1. for any event $A \subseteq \Omega$, $0 \leq \Pr(A) \leq 1$;
2. $\Pr(\Omega) = 1$;
3. for any sequence $A_1, A_2, A_3, \ldots$ of disjoint events,

$$\Pr\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \Pr(A_i)$$

B.2 Random Variables and Distribution Functions

Most often, we are not dealing with probability spaces directly, but with *functions on probability spaces*.

Definition B.1 A function $X : \Omega \to E$ from a probability space in the set E is said to be a *random variable*.

A random variable X is completely characterised by its distribution function, which is defined as $F_X(t) = \Pr\{\omega : X(\omega) \leq t\}$. We abbreviate the expression $\Pr\{\omega : X(\omega) \leq t\}$ as $\Pr(X \leq t)$. If X is a random variable, then we denote by $df(X)$ the distribution function of X.

It is common to not even refer to the probability space of a random variable, but to simply assume "*a random variable X with a distribution function F.*" Whenever we introduce a random variable this way, we always can be sure that there is indeed a probability space Ω and a probability measure $\Pr : 2^{\Omega} \to [0, 1]$ such that X is a random variable on Ω and $F(t) = \Pr(X \leq t)$.

A random variable X is said to be *positive* if $\Pr(X \leq 0) = 0$. A random variable is said to be *non-negative* if $\Pr(X < 0) = 0$. Accordingly, we say that the distribution function of a random variable X is positive (non-negative), if X is positive (non-negative).

B.3 Stochastic Processes

A stochastic process is a collection of random variables $\{X(i), i \in T\}$ on a sample space Ω, indexed by the parameter i taking values in the parameter set T. The random variables take values in a set S, the *state space of the stochastic process*. If T is linearly ordered and enumerable then $\{X_i\}$ is called a *discrete-time* stochastic process. If T is linearly ordered and continuous, then $\{X_i\}$ is a continuous-time stochastic process. In this dissertation, we will deal with *deterministic-time* and *continuous-time Markov chains* (DTMCs and CTMCs, respectively), special classes of stochastic processes.

B.3.1 DTMC

Definition B.2 Let Ω be a sample space and P a probability measure on it. Let S be a countable state space. A stochastic process $X = \{X_m, m \in \mathbb{N}\}$ is said to be a *discrete-time Markov chain* (DTMC) provided that

$$P\{X_{m+1} = j \mid X_0, X_1, \ldots, X_m\} = P\{X_{m+1} = j \mid X_m\}$$

for all $j \in S$ and $m \in \mathbb{N}$.

In this thesis we only have to deal with *time-homogeneous* DTMCs, *i.e.,* DTMCs for which $P\{X_{m+1} = j \mid X_m = i\}$ is independent of m.

A finite state DTMC with, say, n states can be described by a stochastic matrix $\mathbf{P} = (p_{ij})_{n,n}$, *i.e.,* a non-negative matrix which satisfies $\mathbf{P}\underline{1} = \underline{1}$. $\mathbf{P}$ is said to be the *transition probability matrix* of the DTMC. A DTMC can be seen as a set of states and a token that jumps from state to state. The entry p_{ij} describes the probability that the token jumps to state j, once it has landed on i. The i-th row of P contains all possibilities to leave state

i. The trivial case is included, *i.e.,* if $p_{ii} > 0$, then this is the probability to leave i and return to it immediately. It is sometimes convenient to identify states of the DTMC with rows of the corresponding probability matrix. DTMCs can be depicted by directed graphs, where the nodes denote the states and the arcs the possible transitions between states. A transition between two states i, j is possible, if $p_{ij} > 0$. Or, the other way round, if $p_{ij} = 0$ then there is no arc from i to j. Arcs are labelled with the respective probabilities. In Figure B.1 an example of a stochastic matrix and the corresponding graph is shown.

$$\begin{pmatrix} 0.1 & 0.3 & 0.6 \\ 0.2 & 0 & 0.8 \\ 1 & 0 & 0 \end{pmatrix}$$

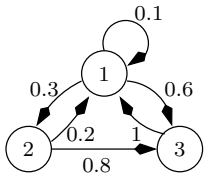


Figure B.1: Stochastic matrix and graph of a 3-state DTMC

Classification of states. The state space S of a DTMC can be classified according to certain reachability properties. A state j is reachable from another state i ($i \to j$), if there is a path leading from i to j in the corresponding graph. States i and j are said to be communicating ($i \leftrightarrow j$), if $i \to j$ and $j \to i$. $\leftrightarrow$ is an equivalence relation on the state space and induces a partition on S with the set of equivalence classes, $S/_{\leftrightarrow}$. A DTMC is said to be *irreducible*, if $S/_{\leftrightarrow}$ has only one element, *i.e.,* if every state $i \in S$ can communicate with every other state $j \in S$.

A state i is said to be *transient*, if there is a positive probability that, once i has been reached, it can never be reached again. All other states are said to be recurrent[1]. If a state i is transient, all states communicating with j are transient as well. Consequently, the same holds for the property of a state to be recurrent.

The probability to choose a certain finite path in a DTMC is the product of the probabilities of all transitions that are "used" on this path. p^l_{ij} is then defined to be the probability to reach state j from i with a path of length l. There may be several paths with length l which lead from i to j, hence, p^l_{ij} is the sum of of the probabilities of all those paths. A state i is said to be *periodic* with period $a > 1$, if $p^l_{ii} = 0$, unless $l = \nu a$ for $\nu \in Nset$ and a is the largest integer with this property. State i is said to be *aperiodic*, if no such a exists.

Stochastic measures. A basic stochastic measure that can be obtained from DTMCs is the probability to be in certain state after a certain amount of time. Since time is discrete for DTMCs, it can be measured in the number of state changes (including the trivial ones).

[1] In the literature there is usually the distinction between null-recurrence and positive recurrence. Since we only consider finite state DTMCs, the distinction is not important for us: in finite-state DTMCs, all recurrent states are positive-recurrent.

The matrix $\mathbf{P}$ describes the 1-step probabilities to change from one state to another. It is not hard to verify that $\mathbf{P}^2$ describes the 2-step probabilities and, generally, $\mathbf{P}^l$, the l-step transition probabilities. If $\underline{\alpha} = (\alpha_1, \ldots, \alpha_n)$ is a probabilistic vector of dimension n (the *starting distribution*), then the vector $\underline{\pi}(m) = \underline{\alpha}\mathbf{P}^m$ is a probability vector with entries $\pi_i(m), i = 1, \ldots, n$, which denote the probability to be in state i after m steps, when starting in state j with probability α_j for $j = 1, \ldots, n$. The limit $\underline{\pi}(\infty) = \underline{\alpha} \lim_{m\to\infty} \mathbf{P}^m$ does exist, when $\mathbf{P}$ is irreducible and aperiodic, and $\pi_i(\infty)$ denotes the probability to be in state i after an infinite number of steps. $\underline{\pi}(\infty)$ is usually referred to as the *steady-state probability distribution* and denoted $\underline{\pi}$.

If the DTMC described by $\mathbf{P}$ is irreducible and aperiodic, it is said to be *ergodic*. It can be shown that the steady-state distribution is the solution of the system of linear equations

$$\underline{\pi}\mathbf{P} = \underline{\pi} \tag{B.1}$$

and $\underline{\pi}\,\underline{1} = 1$. Note that the starting distribution is irrelevant for steady-state measures.

Steady-State Probabilities of Periodic DTMCs

Periodic DTMCs have special properties that make it more difficult to define steady-state probabilities. The limit $\lim_{k\to\infty} P^k$ does not exist, and therefore also $\pi(\infty)$, as defined in the previous section, is not defined. However, we can define a steady-state probability vector for periodic DTMC by means of Equation (B.1).

Definition B.3 Let $\mathbf{P}$ be the probability matrix of an irreducible, periodic Markov chain. Then the steady-state distribution vector of P is defined to be the (unique) probability vector $\underline{\pi}$ that solves the equation

$$\underline{\pi}\mathbf{P} = \underline{\pi}$$

We can express the steady-state probability vector of a periodic DTMC differently. We will do so in Theorem B.5. First, in the following lemma, we prove that, although $\lim_{k\to\infty} \mathbf{P}^k$ does not exist, there exists a limes for a power of $\mathbf{P}$.

Lemma B.4 Let $\mathbf{P} = (p_{ij})_{n,n}$ be the probability matrix of an irreducible, periodic DTMC with period p. Then there is a integer ν such that $\mathbf{P}^{\nu p}$ is a stochastic matrix which describes an aperiodic, irreducible DTMC.

PROOF: $\mathbf{P}^{\nu p}$ is a stochastic matrix. By definition of periodicity, there must be a state i and an integer ν such that $p_{ii}^{(\nu p)} > 0$ (where $p_{ii}^{(\nu p)}$ is the diagonal element of $\mathbf{P}^{\nu p}$ at position (i, i)). Therefore, $\mathbf{P}^{\nu p}$ describes an ergodic DTMC. →•

As a consequence, the limes $\lim_{k\to\infty} \mathbf{P}^{k\nu p}$ does exist.

Theorem B.5 Let $\mathbf{P}$ be the probability matrix of an irreducible, periodic DTMC with period p and integer ν, as defined in Lemma B.4. Let $\underline{\pi}_0$ be an arbitrary starting distribution for $\mathbf{P}$. Let $\mathbf{Q} = \lim_{k\to\infty} \mathbf{P}^{k\nu p}$. We define the vector $\underline{\pi}$ as

$$\underline{\pi} = \underline{\pi}_0 \frac{1}{\nu p} \mathbf{Q} \sum_{l=0}^{\nu p-1} \mathbf{P}^l \tag{B.2}$$

Then $\underline{\pi}$ is a solution of Equation (B.1).

Proof:

$$\begin{aligned}
\underline{\pi}\mathbf{P} &= \left(\underline{\pi}_0 \frac{1}{\nu p} \mathbf{Q} \sum_{l=0}^{\nu p-1} \mathbf{P}^l\right) \mathbf{P} \\
&= \underline{\pi}_0 \frac{1}{\nu p} \mathbf{Q} \sum_{l=1}^{\nu p} \mathbf{P}^l \\
&= \underline{\pi}_0 \frac{1}{\nu p} \left(\underbrace{\mathbf{Q}\mathbf{P}^{\nu p}}_{=\mathbf{Q}\mathbf{I}} + \mathbf{Q} \sum_{l=1}^{\nu p-1} \mathbf{P}^l\right) \\
&= \underline{\pi}_0 \frac{1}{\nu p} \mathbf{Q} \sum_{l=0}^{\nu p-1} \mathbf{P}^l \\
&= \underline{\pi}.
\end{aligned}$$

We see that the steady-state probabilities are independent from the starting distributions, as is also the case for ergodic DTMCs. →•

We have not shown that $\underline{\pi}$, as defined in Theorem B.5, is a probability vector. We can however be sure that $\underline{\pi}$ has only non-negative entries. In case that $\underline{\pi}$ is not stochastic requires then only a renormalisation to obtain a proper probability vector.

We can give an intuitive explanation of Equation B.2. We can rewrite it as

$$\underline{\pi} = \underline{\pi}_0 \left(\frac{1}{\nu p}\mathbf{Q} + \frac{1}{\nu p}\mathbf{Q}\mathbf{P} + \frac{1}{\nu p}\mathbf{Q}\mathbf{P}^2 + \frac{1}{\nu p}\mathbf{Q}\mathbf{P}^3 + \cdots + \frac{1}{\nu p}\mathbf{Q}\mathbf{P}^{\nu p-1}\right)$$

This is the sum of all terms $\mathbf{Q}\mathbf{P}^l$ for $l = 0, \ldots, \nu p - 1$, weighted with the probability $\frac{1}{\nu p}$. The entries q_{ij} of the matrix $\mathbf{Q}\mathbf{P}^l$ are the probabilities to change from state i to state j in an infinite number of steps (which, however, must be divisible by νp)[2], *and then in l additional steps.* Since we must assume that in steady-state each value of $l \in \{0, \ldots, \nu p-1\}$ has the same probability, the νp terms must be weighted with a uniform distribution.

[2]This remark does not make sense, but helps the intuition.

B.3.2 CTMC

CTMCs have, as the name suggests, much in common with DTMCs. As for DTMCs, a CTMC has a discrete state space (which, from now on, we assume to be finite), a graphical representation and a matrix representation. The major difference is that the token that is jumping from state to state remains for a certain time period in each state that can be described by negative-exponentially distributed random variable.

A negative-exponential distribution (we omit the "negative" from now on) is completely specified by by the equation $F(t) = 1 - \mathrm{e}^{\lambda t}$ and the positive, real parameter λ, the *rate*.

An n-state CTMC can be described by the so-called generator matrix $\mathbf{Q} = (q_{ij})_{n,n}$, whose non-diagonal entries are non-negative real numbers, and which fullfills the equation $\mathbf{Q}\underline{1} = \underline{0}$. Obviously, the diagonal elements $q_i = q_{ii}$ for $1 \leq i \leq n$ are the negative sums of the non-diagonal entries of row i. The generator matrix can be interpreted as follows: if the token enters state i, it remains there for a time period that is exponentially distributed with rate $|q_i|$ and jumps then to state j with probability $q_{ij}/|q_i|$. There is no way to return to state i immediately, *i.e.,* without entering one or more other states first.

Classification of states. The classification of states for CTMCs is nearly identical to that for DTMC. The difference is that there is no concept of periodicity of states, hence a CTMC is ergodic, if it is irreducible and vice versa. States that have no outgoing transitions are called *absorbing*. A CTMC with absorbing states is called absorbing.

Steady-State measures. For irreducible CTMCs a steady-state distribution can be computed by means of the system of linear equations

$$\underline{\pi}\mathbf{Q} = 0 \tag{B.3}$$

and $\underline{\pi}\,\underline{1} = 1$. As for the DTMC case, for steady-state measures the starting distribution is not important.

Relations to DTMCs We can immediately verify that DTMCs can be seen as CTMCs: a reformulation of Equation (B.1) is $\underline{\pi}(\mathbf{P}-\mathbf{I}) = 0$, and $\mathbf{P}-\mathbf{I}$ has all properties of a generator matrix. On the other hand, it is not difficult to turn a generator matrix $\mathbf{Q}$ in a transition matrix of a DTMC with the additional condition that both Markov chains have the same steady-state distribution. To do so, one has to choose a $q > 0$ such that $\mathbf{P} = \frac{\mathbf{Q}}{q} + \mathbf{I}$ is a stochastic matrix. This is fulfilled if $q > \max_{i=1,\ldots,n} |q_{ii}|$. Then it is easy to see that, if $\underline{\pi}\mathbf{Q} = 0$ for a non-zero vector $\underline{\pi}$, then $\underline{\pi}\mathbf{P} = \underline{\pi}$. The transformation from $\mathbf{Q}$ to $\mathbf{P}$ is usually referred to as *uniformisation*.

Another important DTMC with respect to to a certain CTMC is its *embedded Markov chain* (EMC). Let $\underline{q} = (q_{11}, q_{22}, \ldots, q_{nn})$ be the vector of the diagonal entries of $\mathbf{Q}$ and

$\mathbf{\Delta} = \text{diag}(\underline{q})$. Then the EMC of the CTMC described by $\mathbf{Q}$ is the DTMC described by the probability matrix

$$\mathbf{E} = -\mathbf{\Delta}^{-1}\mathbf{Q} + \mathbf{I}.$$

The rows of $\mathbf{E} = (e_{ij})_{n,n}$ correspond directly to the states of the originating CTMC, and the values e_{ij} for $i, j \in \{1, \ldots, n\}$ are the probability that CTMC $\mathbf{Q}$ changes from state i to state j, *under the assumption that state i is actually left*. Therefore, $e_{ii} = 0$ for all $i = 1, \ldots, n$.

Throughputs. An important concept for CTMCs with non-trivial steady-state solution is that of *throughput*. Let $\mathbf{Q} = (q_{ij})_{n,n}$ be a generator matrix with steady-state probability vector $\underline{\pi}$. The throughput of a state i is the mean number of visits to state i per unit time. The throughput of a transition t is the mean number of instances per unit time that t is "used" to change state. The throughput of state i is defined by $-\pi_i q_{ii}$. The throughput of a transition t with source state i depends on the throughput of i: if p is the probability that state i is left via transition t, then the throughput of t is $-p\pi_i q_{ii}$.

Let $\mathbf{\Delta} = \text{diag}\,(q_{11}, \ldots, q_{nn})$. The vector of all state throughputs is $\underline{\tau} = -\underline{\pi}\mathbf{\Delta}$. The derivation of the throughputs of each transition is slightly more involved. Let $\mathbf{P} = (p_{ij})_{n,n} = \mathbf{I} - \mathbf{\Delta}^{-1}\mathbf{Q}$ be the EMC of $\mathbf{Q}$. Entries p_{ij} of $\mathbf{P}$ describe the jump probability from state i to state j given that state i is really left. Then the throughputs of the individual transitions are described by a matrix $\mathbf{T}$ which is defined as $\mathbf{T} = (t_{ij})_{n,n} = \text{diag}(\underline{\tau})\mathbf{P}$. The multiplication of $\text{diag}(\underline{\tau})$ with $\mathbf{P}$ "distributes" the state throughputs on the outgoing transitions of the respective state.

Throughputs are sometimes referred to as *probability flux* through the respective states or transitions, comparing probability with a *liquid* that circulates through the Markov chain (as long as it is irreducible). State probabilities are then to be seen as a certain reservoir of probability liquid that accumulates in a state due to the limited capacity of the outgoing transitions to drain the probability from the state. This capacity is expressed by the rates of the transitions. The system of linear equations, $\underline{\pi}\mathbf{Q} = 0$, given above, simply states that the probability flux going into a state is equal to that going out.

This is only a metaphorical view on CTMCs, but it will prove useful in Section B.4.

Transient measures. As for DTMCs, it is sometimes interesting to have non-steady-state measures. The question is, what the probabilities are to be in a certain state after a certain time period t? The answer is a probability vector $\underline{\pi}(t)$, which is defined as

$$\underline{\pi}(t) = \underline{\pi}(0)\mathrm{e}^{\mathbf{Q}t}, \tag{B.4}$$

where $\underline{\pi}(0) = \underline{\alpha}$, the starting distribution. The matrix exponential is defined via the Taylor-McLaurin expansion of the exponential function:

$$\mathrm{e}^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}.$$

From a numerical point of view, Equation (B.4) is not very useful to compute $\pi(t)$ [113]. Instead, Jensen has developed a method, which is numerically very stable and which allows the computation of the number of summations that are required to yield a given precision in advance. The method is based on uniformisation, *i.e.,* one has to choose a $q \geq \max_{i=1,\ldots,n} |q_{ii}|$ and compute $\mathbf{P} = \frac{\mathbf{Q}}{q} + \mathbf{I}$. Then $\mathbf{Q} = q(\mathbf{P} - \mathbf{I})$ and (B.4) can be transformed as follows:

$$\begin{aligned} \underline{\pi}(t) &= \underline{\pi}(0)\mathrm{e}^{\mathbf{Q}t} \\ &= \underline{\pi}(0)\mathrm{e}^{q(\mathbf{P}-\mathbf{I})t} \\ &= \underline{\pi}(0)\mathrm{e}^{-qt}\mathrm{e}^{q\mathbf{P}t} \\ &= \underline{\pi}(0)\sum_{n=0}^{\infty} e^{-qt}\frac{(qt)^n}{n!}\mathbf{P}^n \end{aligned}$$

This summation drafts an algorithm to compute $\underline{\pi}(t)$. The numerical stability is due to the fact that only positive values between 0 and 1 are added, which reduces the danger of cancellation errors.

Numerical techniques. Continuous-time Markov chains that describe performance or dependability models are usually very large in terms of number of rows of the generator matrix. On the other hand, these matrices are very sparse: a matrix might have dimensions in the order of $10^9 \times 10^9$, but it is not uncommon that each row has less then 100 non-zero entries.

As seen above, generator matrices define a system of linear equations that has to be solved in order to derive the steady-state probability vector. Such large systems of linear equations can not be solved by direct methods, as Gaussian Elimination, anymore. Instead, numerical methods must be employed. The three techniques that are used most often are Jacobi-Iteration, Gauss-Seidel-Iteration, and the Successive Over-Relaxation (SOR) method. In [133] a comprehensive overview for numerical solution of Markov chains can be found.

Kronecker Sums of Generator Matrices If $\mathbf{A}$ and $\mathbf{B}$ are generator matrices of CTMCs, $\mathbf{Q} = \mathbf{A} \oplus \mathbf{B}$ describes a CTMC in which both $\mathbf{A}$ and $\mathbf{B}$ act concurrently and independent from each other. Especially, if $\underline{\pi}_{\mathbf{A}}(t)$ and $\underline{\pi}_{\mathbf{B}}(t)$ are the transient probability vectors for $\mathbf{A}$ and $\mathbf{B}$, respectively (with given starting distributions $\underline{\pi}_{\mathbf{A}}(0)$ and $\underline{\pi}_{\mathbf{B}}(0)$), then the transient probability vector $\underline{\pi}_{\mathbf{Q}}(t)$ at time t, is equal to $\underline{\pi}_{\mathbf{A}}(t) \otimes \underline{\pi}_{\mathbf{B}}(t)$ (with starting distribution $\underline{\pi}_{\mathbf{Q}} = \underline{\pi}_{\mathbf{A}}(0) \otimes \underline{\pi}_{\mathbf{B}}(0)$). This can be shown by repeated application of Lemma A.1.

B.3.3 Semi-Markov Chains

Definition

Semi-Markov chains (SMC) can be seen as generalisations of discrete- as well as continuous-time Markov chains. A semi-Markov chain can be described by a tuple $(\Sigma, \underline{p}_0, \mathbf{E}, J)$, where Σ is the state space of the SMC, $\mathbf{E}$ a stochastic matrix of dimension $\#\Sigma$, describing a DTMC, the *embedded Markov chain* (EMC), $\underline{p}_0$ is a starting probability distribution of $\mathbf{E}$ and $J : \Sigma \longrightarrow \mathcal{F}^+$ is a function that assigns non-negative distribution functions (the elements of $\mathcal{F}^+$) to states. $J(s)$ describes the state sojourn time of state $s \in \Sigma$. An SMC is a CTMC, if for all $s \in \Sigma$, $J(s)$ is a negative-exponential distribution (with arbitrary rate $r_s \in \mathbb{R}^+$). An SMC is a DTMC, if the sojourn time distributions for all states are ignored.

Steady-State Solution

The steady-state solution of an SMC is a probability vector $\underline{\pi} = (\pi_1, \ldots, \pi_n)$ that can be computed quite easily. Let $\underline{p} = (p_1, \ldots, p_n)$ the steady-state solution of EMC $\mathbf{E}$. We assume that the solution exists, or, stated otherwise, a solution for an SMC only exists if a solution for its EMC exists. For $F = J(i)$, let μ_i be the solution to the integral $\int_0^\infty 1 - F(x)dx$, $i = 1, \ldots, n$. Then μ_i is the mean value of a random variable that is distributed according to $J(i)$. Then the steady-state probabilities π_i of the SMC are defined as follows:

$$\pi_i = \frac{p_i \mu_i}{\sum_{j=1}^{n} p_j \mu_j}, \tag{B.5}$$

or, in matrix notation,

$$\underline{\pi} = \frac{1}{\underline{p}\ \underline{\mu}^T}\ \underline{p}\ \mathrm{diag}(\underline{\mu}). \tag{B.6}$$

For a proof we refer to [80, Chapters 10 and 11].

B.4 Phase-Type Distributions

An important class of non-negative distribution functions is the class of so-called *phase-type* distributions. Phase-type distributions are strongly related to absorbing CTMCs. In fact, if X is a phase-type distributed random variable, then there is an absorbing CTMC with a generator matrix $\mathbf{Q} \in \mathbb{R}^{m \times m}$ with absorbing state m and with starting distribution $\underline{\pi}(0)$ such that

$$\Pr(X \leq t) = \underline{\pi}(t)\underline{e}_m,$$

where $\underline{\pi}(t) = \underline{\pi}(0)\mathrm{e}^{\mathbf{Q}t}$. This is also valid the other way around, *i.e.,* if $\mathbf{Q}$ describes an absorbing CTMC, then $F(t) = \underline{\pi}(t)\underline{e}_m$ is a phase-type distribution.

Generator matrices which represent phase-type distributions are generally of the form

$$\mathbf{Q} = \begin{pmatrix} \mathbf{T} & \mathbf{T}_0 \\ \mathbf{0} & 0 \end{pmatrix}, \tag{B.7}$$

where $\mathbf{T}$ is a $(m-1) \times (m-1)$ sub-matrix and $\mathbf{T}_0$ is a $(m-1) \times 1$ column vector. Note that $\mathbf{T}_0 = -\mathbf{T}\underline{1}$.

It is common to describe a phase-type distribution by its *representation*, *i.e.,* the tuple $(\underline{\alpha}, \mathbf{T})$, where $\alpha_i = \pi_i(0)$ for $i = 1, \ldots, m-1$.

If we have a phase-type distribution defined by a representation $(\underline{\alpha}, \mathbf{T})$, it is sometimes necessary to refer to the complete generator matrix. We denote this with a dot, *i.e.,* $\dot{\mathbf{T}}$, where

$$\dot{\mathbf{T}} = \begin{pmatrix} \mathbf{T} & -\mathbf{T}\underline{1} \\ \mathbf{0} & 0 \end{pmatrix}$$

Accordingly, $\underline{\dot{\alpha}}$ is the starting distribution of $\dot{\mathbf{T}}$, *i.e.,* $\underline{\dot{\alpha}} = (\alpha_1, \alpha_2, \ldots, \alpha_{m-1}, 1 - \underline{\alpha}\,\underline{1})$. On the other hand, if $\mathbf{Q}$ is the generator matrix of an absorbing Markov chain of the form given in Equation B.7, then we define $\langle \mathbf{Q} \rangle = \mathbf{T}$.

Moments of phase-type distributions. If X is a phase-type distributed random variable with representation $(\underline{\alpha}, \mathbf{T})$, then the moments of the random variable are given by

$$E[X^n] = (-1)^n \cdot n! \cdot \underline{\alpha}\mathbf{T}^{-n}\underline{1}$$

The derivation $E[X^n]$ does not really require an explicit matrix inversion. Instead, one can solve the system of linear equations

$$\underline{x}^{(1)}\mathbf{T} = \underline{\alpha}.$$

Then $E[X] = -\underline{x}^{(1)}\underline{1}$. For the nth moment ($n > 1$) one has to compute the vectors $\underline{x}^{(i)}$ successively for $i = 2, \ldots, n$, where $\underline{x}^i$ is the solution of the system of linear equations

$$\underline{x}^{(i)}\mathbf{T} = \underline{x}^{(i-1)}$$

Then $E[X^n] = (-1)^n \cdot n! \cdot \underline{x}^{(n)}\underline{1}$.

Throughputs and state probabilities. We will now look at phase-type distributions from an unusual angle. We assume an absorbing CTMC with generator matrix $\mathbf{Q} = (q_{ij})_{n,n}$, one distinguished starting state and one absorbing state. Without loss of generality, we assume the first row of $\mathbf{Q}$ to correspond to the starting state and the last to the absorbing state. We now assume that with rate τ, probability "liquid" is flowing into the starting state (see Section B.3.2) and distributes over the CTMC according to transition rates given by the generator matrix. How much probability mass π_i is accumulating in each state $1 \leq i \leq n$?

Clearly, this requires a proper definition of flow balance equations. For the first state, we have

$$\sum_{j=2}^{n} \pi_1 q_{1j} = \tau + \sum_{j=2}^{n} \pi_j q_{j1}$$

For states $2 \leq i \leq n-1$, we simply have

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} \pi_i q_{ij} = \sum_{\substack{j=1 \\ j \neq i}}^{n} \pi_j q_{ji}$$

For the absorbing state, n, we have the equation

$$\tau = \sum_{j=1}^{n-1} \pi_j q_{jn}$$

With a little rewriting of the equations and the insight that the last equation is actually not necessary, we can conclude that we are looking for the solution $\underline{\pi}$ of the following system of linear equations:

$$\underline{\pi}\mathbf{T} = (-\tau, 0, \ldots, 0), \tag{B.8}$$

where $\mathbf{T}$ is defined as in B.7. Normally, $\underline{\pi}$ will be sub-stochastic, *i.e.,* $\underline{\pi}\underline{1} < 1$. To ensure that $\underline{\pi}\underline{1} \leq 1$ holds, we have to require that $1/\tau \geq -\underline{e}_1^T \mathbf{T}^{-1}\underline{1}$: since

$$\pi\mathbf{T} = -\tau(1, 0, \ldots, 0)$$

is equivalent to

$$\pi = -\tau(1, 0, \ldots, 0)\mathbf{T}^{-1},$$

we can derive

$$\begin{aligned} & \underline{\pi}\underline{1} && \leq && 1 \\ \iff & -\tau(1, 0, \ldots, 0)\mathbf{T}^{-1}\underline{1} && \leq && 1 \\ \iff & -(1, 0, \ldots, 0)\mathbf{T}^{-1}\underline{1} && \leq && \frac{1}{\tau}, \end{aligned}$$

so that the requirement above for $\underline{\pi}$ is met.

Bibliography

[1] V. Adlakha and V. Kulkarni. A classified bibliography of research on stochastic PERT networks: 1966-1987. Technical report, Dept. of Operations Research, University of North Carolina, Chapel Hill, 1987.

[2] S.C. Agrawal, J.P. Buzen, and A.W. Shum. Response Time Preservation: a general technique for developing approximate algorithms for queueing networks. *Performance Evaluation Review*, 12(3):63–77, August 1984.

[3] S. C. Allmaier, M. Kowarschik, and G. Horton. State space construction and steady-state solution of GSPNs on a shared-memory-multiprocessor. In *Proceedings of the Seventh International Workshop on Petri Nets and Performance Models*, pages 112–121. IEEE Computer Society Press, Oct 1997.

[4] H.H. Ammar and S.M. Rezaul Islam. Time scale decomposition of a class of generalised stochastic Petri nets. *IEEE Transactions on Software Engineering*, 15(6):809–820, June 1989.

[5] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[6] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22(2):248–260, 1975.

[7] Alexander Bell and Boudewijn R. Haverkort. Serial and parallel out-of-core solution of linear systems arising from GSPNs. In Adrian Tentner, editor, *High Performance Computing (HPC '01) — Grand Challenges in Computer Simulation*, pages 242–247, San Diego, CA, USA, April 2001. The Society for Modeling and Simulation International.

[8] J.A. Bergstra and J.W. Klop. An introduction to process algebra. In J.C.M. Baeten, editor, *Applications of Process Algebra*, Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[9] Marco Bernardo. An algebra-based method to associate rewards with EMPA terms. In R. Gorrieri P. Degano and A. Marchetti, editors, *Automata, Languages and Programming*, volume 1256 of *Lecture Notes in Computer Science*, pages 358–368. Springer Verlag, 1997.

[10] Marco Bernardo and Mario Bravetti. Formal specification of performance measures for process algebra models of concurrent systems. Technical Report UBLCS–1998–08, University of Bologna (Italy), 1998.

[11] Marco Bernardo and Mario Bravetti. Reward based congruences: Can we aggregate more? In de Alfaro and Gilmore [45], pages 136–151.

[12] Marco Bernardo, Lorenzo Donatiello, and Roberto Gorrieri. Modeling and analyzing concurrent systems with MPA. In Herzog and Rettelbach [71].

[13] Marco Bernardo and Roberto Gorrieri. Extended markovian process algebra. In U. Montanari and V. Sassone, editors, *7th Int. Conf. on Concurrency Theory (CONCUR 1996)*, volume 1119 of *Lecture Notes in Computer Science*, pages 315–330. Springer Verlag, August 1996.

[14] A. Blackmore and S. Tripathi. Automated time scale decomposition of SPNs. In *Proceedings of the Fifth International Workshop on Petri Nets and Performance Models*. IEEE Computer Society Press, 1993.

[15] Henrik Bohnenkamp and Boudewijn Haverkort. Semi-numerical solution of stochastic process algebra models. In Priami [122], pages 71–84.

[16] Henrik Bohnenkamp and Boudewijn Haverkort. Stochastic event structures for the decomposition of stochastic process algebra models. In Hillston and Silva [76], pages 25–39.

[17] Henrik C. Bohnenkamp and Boudewijn R. Haverkort. Semi-numerical solution of stochastic process algebra models. In Katoen [85], pages 228–243.

[18] Gérard Boudol and Ilaria Castellani. Permutation of transitions: An event structure semantics for CCS and SCCS. In de Bakker et al. [46].

[19] M. Bravetti and R. Gorrieri. Interactive generalized semi-Markov processes. In Hillston and Silva [76].

[20] Ed Brinksma and Holger Hermanns. Process algebra and markov chains. In Ed Brinksma, Holger Hermanns, and Joost-Pieter Katoen, editors, *Lectures on Formal Methods and Performance Analysis*, volume 2090 of *Lecture Notes in Computer Science*, pages 183–231. Springer Verlag, 2001.

[21] Hendrik Brinksma. The specification language LOTOS. In *Proceedings NGI/SION Informatica Symposium*, pages 374–387. NGI-SIC, 1985.

[22] Hendrik Brinksma. *On the design of extended LOTOS*. PhD thesis, University of Twente, The Netherlands, 1988.

[23] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[24] R. E. Bryant. Symbolic boolean manipolation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[25] Peter Buchholz. Markovian process algebra: Composition and equivalence. In Herzog and Rettelbach [71].

[26] Javier Campos, José Manuel Colom, Hauke Jungnitz, and Manuel Silva. Approximate throughput computation of stochastic marked graphs. *IEEE Transactions on Software Engineering*, 20(7):526–535, July 1994.

[27] K.M. Chandy, U. Herzog, and L. Woo. Parametric analysis of queueing networks. *IBM Journal on Research and Development*, 19(1):36–42, January 1975.

[28] G. Ciardo, J. Gluckman, and D. Nicol. Distributed state space generation of discrete-state stochastic models. Technical report, College of William and Mary, June 1996. To appear in *Informs Journal of Computing*.

[29] G. Ciardo, G. Luettgen, and R. Siminiceanu. Efficient symbolic state-space construction for asynchronous systems. In M. Nielson and D. Simpson, editors, *Application and Theory of Petri Nets (ICATPN 2000, Proceedings)*, volume 1825 of *Lecture Notes in Computer Science*, pages 103–122. Springer-Verlag, June 2000.

[30] G. Ciardo, G. Luettgen, and R. Siminiceanu. Saturation: An efficient iteration strategy for symbolic state-space generation. In Margaria and Yi [100], pages 328–342.

[31] G. Ciardo and K. S. Trivedi. A decomposition approach for stochastic reward net models. *Performance Evaluation*, 18:37–59, 1993.

[32] Gianfranco Ciardo. What a structural world. In Reinhard German and Boudewijn Haverkort, editors, *Petri Nets and Performance Models (PNPM 2001)*, pages 3–16. IEEE Computer Society, September 2001.

[33] Erhan Çinlar. *Introduction to Stochastic Processes*. Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1975.

[34] Graham Clark. Formalising the specification of rewards with PEPA. In Ribaudo [125].

[35] Graham Clark. *Techniques for the Construction and Analysis of Algebraic Performace Models*. PhD thesis, University of Ediburgh, Scotland, 2000.

[36] Graham Clark, Stephen Gilmore, and Jane Hillston. Specifying performance measures with PEPA. In Katoen [85].

[37] E. M. Clarke, M. Fujita, P.C. McGeer, K. McMillan, and J.C.Y. Yang. Multi-terminal binary decision diagrams; an efficient structure for matrix representation. Early version of [56], February 1993.

[38] Lucia Cloth. Complete finite prefix for stochastic process algebra. Diploma thesis, Rheinisch-Westfälische Technische Hochschule, Department of Computer Science, Aachen, Germany, 2000.

[39] Lucia Cloth, Henrik Bohnenkamp, and Boudewijn Haverkort. Using max-plus algebra for the evaluation of stochastic process algebra prefixes. In de Alfaro and Gilmore [45].

[40] P. J. Courtois. *Decomposability: Queueing and Computer System Applications*. Academic Press, 1977.

[41] Pedro R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. PhD thesis, Department of Computer Science, University of Twente, November 1999.

[42] Pedro R. D'Argenio, Holger Hermanns, Joost-Pieter Katoen, , and Ric Klaren. MoDeST - a modelling and description language for stochastic timed systems. In de Alfaro and Gilmore [45], pages 87–104.

[43] P. Dauphin, R. Hofmann, R. Klar, B. Mohr, A. Quick, M. Siegle, and F. Sötz. ZM4/SIMPLE: a general approach to performance-measurement and evaluation of distributed systems. In T.L. Casavant and M. Singhal, editors, *Advances in Distributed Computing: Concepts and Design*. IEEE Computer Society Press, 1992.

[44] Luca de Alfaro. Stochastic transition systems. In Davide Sangiorgi and Robert de Simone, editors, *CONCUR '98: Concurrency Theory (Proceedings)*, volume 1466 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[45] Luca de Alfaro and Stephen Gilmore, editors. *Process Algebra and Probabilistic Methods (PAPM-ProbmiV 2001)*, volume 2165 of *Lecture Notes in Computer Science*. Springer Verlag, September 2001.

[46] J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors. *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*. Springer-Verlag, June 1988.

[47] Susanna Donatelli. Superposed stochastic automata: a class of stochastic Petri nets with parallel solution and distributed state space. *Performance Evaluation*, 18:21–26, 1993.

[48] Amani El-Rays, Marta Kwiatkowska, and Gethin Norman. Solving infinite stochastic process algebra models through matrix-geometric methods. In Hillston and Silva [76].

[49] Salah E. Elmaghraby. *Activity Networks*. John Wiley & Sons, 1977.

[50] Joost Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28:575–591, 1991.

[51] Agner K. Erlang. The theory of probabilities and telephone conversations. *Nyt Tidsskrift for Matematik B*, 20, 1909.

[52] Agner K. Erlang. Solution of some problems in the theory of probabilities of significance in automatic telephone exchanges. *Elektrotkeknikeren*, 13, 1917.

[53] J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. In *Proc. TACAS '96*, volume 1055 of *Lecture Notes in Computer Science*, pages 87–106. Springer-Verlag, 1997.

[54] William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. John Wiley & Sons, third edition, 1968.

[55] F.-J. Fritz, B. Huppert, and W. Willems. *Stochastische Matrizen*. Springer-Verlag, Berlin, Heidelberg, New York, 1979.

[56] M. Fujita, P.C. McGeer, and J.C.-Y. Yang. Multi-terminal binary decision diagrams; an efficient structure for matrix representation. *Formal Methods in System Design*, 10(2/3):149–169, April 1997.

[57] W. J. Gordon and G.J. Newell. Closed queueing system with exponential servers. *Operations Research*, 15:254–265, 1967.

[58] Norbert Götz. *Stochastische Prozeßalgebren — Integration von funktionalem Entwurf und Leistungsbewertung Verteilter Systeme*. PhD thesis, Univerität Erlangen-Nürnberg, Germany, 1994.

[59] Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Markovian analysis of large finite state machines. *IEEE Transactions on Computer-Aided Design,*, 15(12):1479–1493, 1996.

[60] Peter G. Harrison and Naresh M. Patel. *Performance Modelling of Communication Networks and Computer Architectures*. Addison-Wesley, 1993.

[61] F. Hartleb. Stochastic graph models for performace evaluation of parallel programms and the evaluation tool PEPP. In N. Götz, U. Herzog, and M. Rettelbach, editors, *Proceedings of the QMIPS Workshop on Formalisms, Principles and State-of-the-art*, volume 26(14) of *Arbeitsberichte des IMMD*, pages 207–224. University Erlangen-Nürnberg, 1993.

[62] B. R. Haverkort. *Performance of Computer Communication Systems: A Model-Based Approach*. John Wiley & Sons, 1998.

[63] Boudewijn Haverkort, Alexander Bell, and Henrik Bohnenkamp. On the efficient sequential and distributed evaluation of very large stochastic Petri nets. In Peter Buchholz and Manuel Silva, editors, *Proceedings of the Eighth International Workshop on Petri Nets and Performance Models*, pages 12–21. IEEE Computer Society Press, Oct 1999.

[64] Boudewijn R. Haverkort and Kishor S. Trivedi. Specification techniques for Markov reward models. *Discrete Event Systems: Theory and Applications*, 3:219–247, 1993.

[65] Matthew Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.

[66] Holger Hermanns. *Interactive Markov Chains*. PhD thesis, Universität Erlangen-Nürnberg, Germany, 1998.

[67] Holger Hermanns and Michael Rettelbach. Syntax, Semantics, Equivalences, and Axioms for MTIPP. In Herzog and Rettelbach [71].

[68] Holger Hermanns and Michael Rettelbach. Towards a superset of LOTOS for performance prediction. In Ribaudo [125], pages 77–94.

[69] Ulrich Herzog. EXL: Syntax, semantics and examples. Technical Report IMMD7-16/90, University of Erlangen-Nürnberg, Nov 1990.

[70] Ulrich Herzog. Formal description, time and performance analysis – a framework. In T. Härder, H. Wedekind, and G. Zimmermann, editors, *Entwurf und Betrieb verteilter Systeme*, volume 264 of *Informatik-Fachberichte*. Springer-Verlag, 1990.

[71] Ulrich Herzog and Michael Rettelbach, editors. *Proceedings of the 2nd workshop on process algebras and performance modelling*, volume 27 of *Arbeitsberichte des IMMD*. FAU Erlangen-Nürnberg, 1994.

[72] J. Hillston and V. Mertsiotakis. A simple time scale decomposition technique for stochastic process algebras. In Jane Hillston and Stephen Gilmore, editors, *Proceedings of The Third Workshop on Process Algebra and Performance Modelling*, volume 38(7) of *The Computer Journal*, pages 566–577. Oxford University Press, 1995.

[73] Jane Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, University of Edinburgh, 1994.

[74] Jane Hillston. Exploiting structure in solution: decomposing composed models. In Priami [122].

[75] Jane Hillston and Leïla Kloul. An efficient kronecker representation for PEPA models. In de Alfaro and Gilmore [45], pages 120–135.

[76] Jane Hillston and Manuel Silva, editors. *Proceedings of the Seventh International Workshop on Process Algebras and Performance Modelling (PAPM '99)*. Prensas Universitarias de Zaragoza, 1999.

[77] Jane Hillston and Nigel Thomas. Product form solution for a class of PEPA models. In *Proceedings of IEEE International Computer and Dependability Symposium*, Durham, NC, September 1998. IEEE Computer Society Press.

[78] C.A.R. Hoare. Communicating sequential processes. *Communications of ACM*, 21:666–677, 1978.

[79] C.A.R. Hoare. *Communicating Sequential Processes*. Series in Computer Science. Prentice-Hall International, 1985.

[80] Ronald A. Howard. *Dynamic Probabilistic Systems*, volume 2: Semimarkov and Decision Processes. John Wiley & Sons, 1971.

[81] Ronald A. Howard. *Dynamic Probabilistic Systems*, volume 1: Markov Models. John Wiley & Sons, 1971.

[82] ISO. LOTOS: A formal description technique based on the temporal ordering of observational behaviour, 1989. IS 8807.

[83] J.R. Jackson. Networks of waiting lines. *Operations Research*, 5:518–521, 1957.

[84] Joost-Pieter Katoen. *Quantitative and Qualitative Extensions of Event Structures*. PhD thesis, University of Twente, 1996.

[85] Joost-Pieter Katoen, editor. *Proceedings of the 5th International AMAST Workshop, ARTS '99*, volume 1601 of *Lecture Notes in Computer Science*. Springer-Verlag, May 1999.

[86] Joost-Pieter Katoen, Ed Brinksma, Diego Latella, and Rom Langerak. Stochastic simulation of event structures. In Ribaudo [125].

[87] John G. Kemeny and J. Laurie Snell. *Finite Markov Chains*. Van Nostrand, 1960.

[88] John G. Kemeny, J. Laurie Snell, and Anthony W. Knapp. *Denumerable Markov Chains*. Springer-Verlag, 2nd edition, 1976.

[89] W. Knottenbelt and P. Harrison. Distributed disk-based solution techniques for large Markov models. In Brigitte Plateau, William J. Stewart, and Manuel Silva, editors, *Proceedings of the 3rd International Workshop on Numerical Solutions of Markov Chains*, pages 58–75. Prensas Universitarias de Zaragoza, 1999.

[90] W. Knottenbelt, M. Mestern, P. Harrison, and P. Kritzinger. Probability, parallelism and the state space exploration problem. In Ramon Puigjaner, Nunzio N. Savino, and Bartomeu Serra, editors, *Computer Performance Evaluation: Modelling Techniques and Tools*, volume 1496 of *Lecture Notes in Computer Science*, pages 165–179. Springer-Verlag, 1998.

[91] William J. Knottenbelt. *Parallel Performance Analysis of Large Markov Models*. PhD thesis, University of London, February 2000.

[92] D. D. Kouvatsos and N. P. Xenios. Maximum entropy analysis of general queueing networks with blocking. In H. G. Perros and T. Atiok, editors, *Workshop on queueing networks with blocking*. North-Holland, Amsterdam, 1989.

[93] Vidyadhar G. Kulkarni. *Modeling and Analysis of Stochastic Systems*. Chapman & Hall, London, Glasgow, Weinheim, 1995.

[94] Rom Langerak. *Transformations and Semantics for LOTOS*. PhD thesis, Department of Computer Science, University of Twente, 1992.

[95] Rom Langerak. Deriving a graph rewriting system from a complete finite prefix of an unfolding. *Electronic Notes in Theoretical Computer Science*, 27, 1999. `URL: http://www.elsevier.nl/locate/entcs/volume27.html`.

[96] Rom Langerak and Ed Brinksma. A complete finite prefix for process algebra. In *Proceedings of the International Conference on Computer-Aided Verification*, Lecture Notes in Computer Science. Springer Verlag, July 1999.

[97] Rom Langerak and Ed Brinksma. A complete finite prefix for process algebra. Technical report, University of Twente, 1999. Full Version of [96].

[98] Rita Loogen and Ursula Golz. Modelling nondeterministic concurrent processes with event structures. *Fundamenta Informaticae*, XIV:39–74, 1991.

[99] D. G. Malcolm, J. H. Roseboom, C. E. Clark, and E. Fazar. Applications of a technique for research and development program evaluation. *Operations Research*, 7(5):646–669, 1959.

[100] T. Margaria and W. Yi, editors. *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001, Proceedings)*, volume 2031 of *Lecture Notes in Computer Science*. Springer-Verlag, April 2001.

[101] Andrei A. Markov. Investigation of an important case of dependent trials. *Izvestia Acad. Nauk SPB, VI, ser. 1*, 61, 1907. (Russian, cited after [88]).

[102] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1995.

[103] M. Ajmone Marsan, G. Conte, and G. Balbo. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(2):93–122, 1984.

[104] Antoni Mazurkiewicz. Basic notions of trace theory. In de Bakker et al. [46].

[105] K. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Proc. CAV '92. Fourth Workshop on Computer Aided Verification*, volume 663 of *Lecture Notes in Computer Science*, pages 164–174. Springer Verlag, 1992.

[106] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[107] Vasilis Mertsiotakis and Manuel Silva. A throughput approximation algorithm for decision free processes. In Ribaudo [125], pages 161–178.

[108] Vassilios Mertsiotakis. *Approximate Analysis Methods for Stochastic Process Algebras*. PhD thesis, Universität Erlangen-Nürnberg, Germany, 1998.

[109] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[110] Robin Milner. *Communication and Concurrency*. Series in Computer Science. Prentice Hall International, 1989.

[111] Isi Mitrani. *Simulation techniques for discrete event systems*. Cambridge University Press, 1982.

[112] Isi Mitrani, Alexander Ost, and Michael Rettelbach. TIPP and the spectral expansion method. In F. Baccelli, A. Jean-Marie, and I. Mitrani, editors, *Quantitative Methods in Parallel Systems*, Esprit Basic Research Series. Springer-Verlag, October 1995.

[113] Cleve Moler and Charles van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Review*, 20(4):801–836, Oct 1978.

[114] M.K. Molloy. Performance analysis using stochastic petri nets. *IEEE Transaction on Computers*, C-31:913–917, September 1982.

[115] Marcel F. Neuts. *Matrix-Geometric Solutions in Stochastic Models – An Algorithmic Approach*. Series in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, 1981.

[116] Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. Petri nets, event structures, and domains, part I. *Theoretical Computer Science*, 13:85–108, 1981.

[117] N. Nounou and Y. Yemini. Algebraic specification-based performance analysis of communication protocols. In *Protocol specification, Testing and Verification*, pages 541–560. Elsevier, 1985.

[118] D.M.R. Park. *Concurrency and Automata on Infinite Sequences*, volume 104 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[119] Brigitte Plateau and Karim Atif. Stochastic automata networks for modeling parallel systems. *IEEE Transactions on Software Engineering*, 17(10):1093–1008, October 1991.

[120] Gordon Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Computer Science Department, Aarhus University, September 1981.

[121] Corrado Priami. *Enhanced Operational Semantics for Concurrency*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 1996.

[122] Corrado Priami, editor. *Proceedings of the sixth workshop on process algebras and performance modelling*. Universita Degli Studi di Verona, 1998.

[123] Martin L. Puterman. *Markov Decision Processes*. John Wiley & Sons, 1994.

[124] Michael Rettelbach. *Stochastische Prozessalgebren mit zeitlosen Aktivitäten und probabilistischen Verzweigungen*. PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, April 1996.

[125] Marina Ribaudo, editor. *Proceedings of the fourth workshop on process algebras and performance modelling*. Edizione C.L.U.T. Torino, 1996.

[126] Theo C. Ruys. *Towards Effective Model Checking*. PhD thesis, University of Twente, 2001.

[127] Theo C. Ruys, Rom Langerak, Joost-Pieter-Katoen, Diego Latella, and Mieke Massink. First passage time analysis of stochastic process algebra using partial orders. In Margaria and Yi [100], pages 220–235.

[128] Gerald S. Shedler. *Regenerative Stochastic Simulation*. Academic Press, 1993.

[129] Markus Siegle. Technique and tool for symbolic representation and manipulation of stochastic transition systems. Technical Report TR IMMD 7, 2/98, Universität Erlangen-Nürnberg, March 1998.

[130] Markus Siegle. Compositional representation and reduction of stochastic labelled transition systems based on decision node BDDs. In D. Baum, N. Mueller, and R. Roedler, editors, *Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen (MMB'99)*, pages 173–185. VDE Verlag, September 1999.

[131] H.A. Simon and A. Ando. Aggregation of variables in dynamic systems. *Econometrica*, 29:111–138, 1961.

[132] Willi-Hans Steeb. *Kronecker Product of Matrices and Applications*. BI-Wissenschaftsverlag, 1991.

[133] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

[134] Nigel Thomas and Stephen Gilmore. Applying quasi-separability to Markovian process algebra. In Priami [122].

[135] Aad P.A. van Moorsel. *Performability Evaluation Concepts and Techniques*. PhD thesis, University of Twente, The Netherlands, 1993.

[136] Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state systems. In *26th Annual Symposium on Foundations of Computer Science (FOCS '85)*, pages 327–338. IEEE Computer Society Press, October 1985.

[137] Glynn Winskel. *The Formal Semantics of Programming Languages*. The MIT Press, 1993.

Index

Curriculum Vitae

Name:	Henrik Bohnenkamp
Birthdate/Location:	17. April 1967 in Minden/Westfalen
Nationality:	German
Marital status:	unmarried, no children

Education

1973 – 1975	Elementary school Barkhausen/Porta Westfalica
1975 – 1977	Elementary school Meißen/Minden
1977 – 1986	Ratsgymnasium Minden, finished with Abitur
1988 – 1995	Student in Computer Science, Friedrich-Alexander Universität Erlangen-Nürnberg. Diploma thesis supervised by Prof. Ulrich Herzog, Michael Rettelbach, Markus Siegle: "*Kompositionelle Semantiken stochastischer Prozeßalgebren zur Erzeugungung reduzierter Transitionssysteme*"
1995 – 2001	Phd-Student, Laboratory for Performance Evaluation and Distributed Systems, Rheinisch-Westfälische Technische Hochschule Aachen

Professional Activities

1991 – 1995	Student assistant at the Lehrstuhl 4 of FAU Erlangen-Nürnberg
1995 – 2000	Teaching Assistant, Laboratory for Performance Evaluation and Distributed Systems, Rheinisch-Westfälische Technische Hochschule Aachen
Since 2001	Post-Doc researcher, Universiteit Twente, The Netherlands

Other Activities

1986 – 1988	Alternative civilian service ("Zivildienst")