Dottorato di Ricerca in Informatica
Università di Bologna, Padova, Venezia

Theory and Application of Extended Markovian Process Algebra

Marco Bernardo

22 Febbraio 1999

Coordinatore:
Prof. Özalp Babaoğlu

Tutore:
Prof. Roberto Gorrieri

Abstract

Many computing systems consist of a possibly huge number of components that not only work independently but also communicate with each other. The catastrophic consequences of failures, such as loss of human lives, environmental damages, and financial losses, in many of these critical systems compel computer scientists and engineers to develop techniques for ensuring that these systems are implemented correctly despite of their complexity. Although a number of theories and software tools have been developed to support the formal description and verification of functional properties of systems, only in recent years the formal modeling and assessment of performance characteristics have received attention.

This thesis addresses the problem of providing a suitable linguistic support which enables designers to formally describe and evaluate system performance in the early stages of system design, in order to avoid cost increases due to the late discovery of inefficiency. A reasonable solution should constitute a first step towards a methodology for the specification and analysis of computer, communication and software systems that achieves a reasonable balance among formality, expressivity, usability and efficiency.

As a solution to the problem above, in this thesis we propose an integrated approach to modeling and analyzing functional and performance characteristics of systems which relies on formal description techniques extended with a stochastic representation of time. Such an approach, inspired by Olderog's work on relating different views of concurrent systems, is based on both stochastically timed process algebras and stochastically timed Petri nets in order to profit from their complementary advantages: compositional modeling/manipulation capabilities and structural analysis techniques based on an explicit description of concurrency.

In order to develop the integrated approach in the Markovian case, in this thesis we propose a new stochastically timed process algebra called EMPA (Extended Markovian Process Algebra), inspired by Herzog's TIPP and Hillston's PEPA, which has a considerable expressive power, as it allows for the description of not only exponentially timed activities but also immediate activities, priority and probability related features, and nondeterminism, and is based on an intuitive master-slaves synchronization mechanism, which allows more complex synchronization mechanisms to be simulated.

In order to support the various phases and analyses of the integrated approach, EMPA is equipped with a suitable collection of semantics, mapping terms to integrated transition systems and their projections (functional transition systems and continuous/discrete time Markov chains) and generalized stochastic Petri nets by Ajmone Marsan-Balbo et al., respectively, as well as a notion of integrated equivalence, relating terms describing systems with the same functional and performance properties.

The major consequences of the restriction to exponentially distributed and zero durations are the possibility of defining the integrated semantics for EMPA in the interleaving style (thanks to the memoryless property of exponential distributions) and the capability of computing performance measures using Markov chains. The integrated equivalence, which is defined on the integrated semantics according to Larsen-Skou's probabilistic bisimulation because of its relationship with the aggregation technique for Markov chains known as ordinary lumping, is proved to be for a large class of terms the coarsest congruence contained in the intersection of a purely functional equivalence and a purely performance equivalence, thus allowing for compositional reasoning and emphasizing the necessity, beside the convenience, of defining such an equivalence on the integrated semantics instead of its projections.

In this thesis EMPA is then enriched with the capability of specifying performance measures at the algebraic level through rewards and the capability of describing data driven computations based on value passing among system components. Rewards are directly specified within terms and the semantics and the integrated equivalence are modified accordingly in order to make them performance measure sensitive. Value passing is handled by providing variable name independent semantic rules that produce compact versions of improved Hennessy-Lin's symbolic models. We show that by means of value passing based expressions it is possible to deal with activities having a generally distributed duration.

Since the generation of the semantic models underlying EMPA terms as well as the checking of the integrated equivalence and the simulation of EMPA terms can be fully mechanized, in this thesis we describe the implementation of the integrated approach in the Markovian case through a software tool called TwoTowers, which compiles EMPA descriptions of systems into their underlying semantic models and invokes other tools such as CWB-NC, MarCA, and GreatSPN to analyze these models. Exploiting already existing tools is advantageous both from the point of view of the implementors, as the construction of TwoTowers becomes easier and faster, and from the point of view of the users, since they are provided with a full range of automated techniques implemented in widely used tools.

This thesis is concluded by presenting several case studies concerning communication protocols (alternating bit protocol, CSMA/CD, token ring, ATM switch, adaptive mechanism for packetized audio) and distributed algorithms (randomized algorithm for dining philosophers, mutual exclusion algorithms) modeled with EMPA and analyzed with TwoTowers. The purpose of such case studies is to demonstrate the adequacy of the integrated approach, the expressiveness of EMPA, and the importance of developing automated tools such as TwoTowers to help designers in modeling and analyzing complex systems in a formal way.

Acknowledgements

I would like to thank Prof. Roberto Gorrieri for serving as my Ph.D. supervisor and Prof. Lorenzo Donatiello who initially served as my Ph.D. cosupervisor. I am very grateful to both of them for the valuable discussions, their suggestions, and the financial support and travel opportunities they gave me during the past four years. I would like to thank Prof. Ed Brinksma for his kind willingness to review my Ph.D. thesis.

During my graduate study, I have had the pleasure to collaborate with several people from the Department of Computer Science of the University of Bologna and North Carolina State University. First of all, I would like to thank Dr. Nadia Busi, for her suggestions about the presentation of the interleaving semantics for EMPA and the collaboration on the label oriented net semantics for EMPA, and Mario Bravetti, for his remarks and help on the congruence proof for $\sim_{\mathrm{EMB}}$ with respect to recursion and the collaboration on process algebras with generally distributed durations (which will be the topic covered in his Ph.D. thesis).

A period I really enjoyed has been my visit at the North Carolina State University, which has led to the implementation of TwoTowers. In particular, I would like to thank Prof. Rance Cleaveland for the stimulating discussions as well as the financial support, Dr. Steve Sims for his help when I was using the PAC-NC in order to produce a version of the CWB-NC tailored for EMPA, and Prof. Billy Stewart for introducing me to MarCA.

I would like to thank Prof. Marco Roccetti for using EMPA and TwoTowers to predict the performance of his novel adaptive packetized audio mechanism. Thanks also to Marcello Colucci, who developed an early prototype of TwoTowers, Claudio Premici, for the collaboration on case studies concerning medium access control protocols, Simone Mecozzi, for the collaboration on mutual exclusion algorithms, Alessandro Aldini, for the collaboration on ATM switches and packetized audio mechanisms, and Mauro Amico, for his technical support during the development of TwoTowers. Finally, thanks to Dr. Marina Ribaudo for introducing me to GreatSPN.

I am also grateful to Dr. Holger Hermanns and Pedro D'Argenio for their counterexample to a former congruence result for $\sim_{\mathrm{EMB}}$ with respect to the parallel composition operator.

To conlude, I would like to thank (for several different reasons I will not mention here) my family, my friends, the Ph.D. students of the Department of Computer Science of the University of Bologna and North Carolina State University, the researchers of the PAPM (Process Algebras and Performance Modeling) community, and the colleagues of the administration board of ADI (Associazione Dottorandi e Dottori di Ricerca Italiani – http://www.dottorato.it/). This thesis is dedicated to my father Giuseppe and my mother Francesca.

Thank you

how about getting off of these antibiotics
how about stopping eating when I'm full up
how about them transparent dangling carrots
how about that ever elusive kudo

thank you India
thank you terror
thank you disillusionment
thank you frailty
thank you consequence
thank you thank you silence

how about me not blaming you for everything
how about me enjoying the moment for once
how about how good it feels to finally forgive you
how about grieving it all one at a time

thank you India
thank you terror
thank you disillusionment
thank you frailty
thank you consequence
thank you thank you silence

the moment I let go of it was
the moment I got more than I could handle
the moment I jumped off of it was
the moment I touched down

how about no longer being masochistic
how about remembering your divinity
how about unabashedly bawling your eyes out
how about not equating death with stopping

thank you India
thank you providence
thank you disillusionment
thank you nothingness
thank you clarity
thank you thank you silence

Alanis Morissette

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many computing systems consist of a possibly huge number of components that not only work independently but also communicate with each other from time to time. Examples of such systems are communication protocols, operating systems, embedded control systems for automobiles, airplanes, and medical equipment, banking systems, automated production systems, control systems of nuclear and chemical plants, railway signaling systems, air traffic control systems, distributed systems and algorithms, computer architectures, and integrated circuits.

The catastrophic consequences such as loss of human lives, environmental damages, and financial losses, of failures in many of these critical systems (see, e.g., [123]) compelled computer scientists and engineers to develop techniques for ensuring that these systems are designed and implemented correctly despite of their complexity. The need of formal methods in developing complex systems is thus becoming well accepted. Formal methods seek to introduce mathematical rigor into each stage of the design process in order to build more reliable systems.

The need of formal methods is even more urgent when planning and implementing concurrent systems. In fact, they require a huge amount of detail to be taken into account (e.g., interconnection and synchronization structure, allocation and management of resources, real time constraints, performance requirements) and involve many people with different skills in the project (designers, implementors, debugging experts, performance and quality analysts). A uniform and formal description of the system under investigation reduces misunderstandings to a minimum when passing information from one task of the project to another.

Moreover, it is well known that the sooner errors are discovered, the less costly they are to fix. Consequently, it is imperative that a correct design is available before implementation begins. Formal methods were conceived to allow the correctness of a system design to be formally verified. Using these techniques the design can be described in a mathematically precise fashion, correctness criteria can be specified in a similarly precise way, and the design can be rigorously proved to meet or not the stated criteria.

Although a number of description techniques and related software tools have been developed to support the formal modeling and verification of functional properties of systems, only in recent years temporal characteristics have received attention. This has required extending formal description techniques by introducing the concept of time, represented either in a deterministic way or in a stochastic way. In the deterministic

case, the focus typically is on verifying the satisfaction of real time constraints, i.e. the fact that the execution of specific actions is guaranteed by a fixed deadline after some event has happened. As an example, if a train is approaching a railroad crossing, then bars must be guaranteed to be lowered on due time.

In the stochastic case, instead, systems are considered whose behavior cannot be deterministically predicted as it fluctuates according to some probability distribution. Due to economic reasons, such stochastically behaving systems are referred to as shared resource systems because there is a varying number of demands competing for the same resources. The consequences are mutual interference, delays due to contention, and varying service quality. Additionally, resource failures significantly influence the system behavior. In this case, the focus is on evaluating the performance of systems. As an example, if we consider again a railway system, we may be interested in minimizing the average train delay or studying the characteristics of the flow of passengers.

This thesis is intended as a contribution in the field of formal methods to the research aiming at providing a suitable formal support to performance evaluation during system design.

1.1 Performance Modeling and Analysis

The purpose of performance evaluation is to investigate and optimize the time varying behavior within and among individual components of shared resource systems. This is achieved by modeling and assessing the temporal behavior of systems, identifying characteristic performance measures, and developing design rules which guarantee an adequate quality of service. The desirability of taking account of the performance aspects of a system in the early stages of its design has been widely recognized [190, 66, 78, 31, 178]. Nevertheless, it often happens that a system is tested for efficiency only after it has been fully designed and tested for functionality. This results in two problems:

- The late discovery of poor performance causes the system to be designed again, so the cost of the project increases.
- The performance analysis is usually done on a model of a system extracted from the implementation while the functionality analysis is usually carried out on a model derived from a system design. As an example, functional verification could be conducted on a Petri net [157] or an algebraic term [133] describing the system, while performance could be evaluated on a Markov chain or a queueing network model [115] of the system. As a consequence, great care must be taken to ensure that these models are consistent with one another, i.e. do reflect (different aspects of) the same system.

Despite of the solid theoretical foundation and a rich practical experience, performance evaluation is still an art mastered by a small group of specialists. This is particularly true when systems are large or there are sophisticated interdependencies. The problem is that a linguistic support for the description of performance aspects is not provided. Performance is usually modeled by resorting to stochastic processes such as Markov

chains or graphical formalisms such as stochastically timed Petri nets [2], which yield low level models hard to understand as far as complex systems are concerned, or notations such as queueing networks, which are more abstract but do not allow all the important aspects of a system to be modeled in a formal way. From the designer viewpoint, it would be extremely advantageous to profit from a general purpose description technique resulting in readable performance models, possibly specified in a compositional way, which are unambiguous and can be formally analyzed in an efficient way.

1.2 An Integrated Approach

Due to the underlying well established theory and the related tool support, stochastically timed Petri nets (see [2] and the references therein) are probably the most successful formal description technique which accounts for functional as well as performance characteristics of concurrent systems. Once we get a stochastically timed Petri net as a model for the design of a given system, both its functional and performance characteristics can be analyzed on two different projected models (a classical Petri net and a stochastic process) obtained from the same integrated model (the stochastically timed Petri net), so we are guaranteed that the projected models are consistent with one another. Though quite helpful from the analysis standpoint because of their explicit description of concurrency, stochastically timed Petri nets are affected by some shortcomings that need to be addressed:

- Scarce readability of complex models, as the Petri net formalism is not a language.
- Lack of compositionality, i.e. the capability of constructing nets by composing smaller ones through built-in operators.
- Inability to perform an integrated analysis, i.e. an analysis carried out directly on the integrated model without building projected models.

Such drawbacks can be overcome by resorting to stochastically timed process algebras, a novel field originated from the seminal papers [144, 96, 97] whose growing interest is witnessed by the annual organization of an international workshop [147, 148, 149, 150, 151, 152] and the presence of several related Ph.D. theses [100, 73, 181, 168, 113, 166, 129, 85, 112]. First of all, stochastically timed process algebras naturally provide a compositional linguistic support, since they are algebraic languages composed of a small set of powerful operators whereby it is possible to systematically construct process terms from simpler ones, without incurring the graphical complexity of nets. Second, they comprise a family of actions which permit to express both the type and the duration of system activities. As a consequence, integrated semantic models (such as action labeled transition systems) underlying process terms which describe the design of a given system contain both functional and performance information, so from them projected models (such as action type labeled transition systems and Markov chains) can be derived which are guaranteed to be consistent

with each other and are used to investigate functional and performance characteristics, respectively, such as deadlock freeness and resource utilization. Also integrated semantic models are useful from the analysis standpoint as they allow to investigate mixed characteristics such as mean time to deadlock, which refer to both functionality and performance, and allow an integrated analysis to be conducted provided that suitable notions of equivalence are developed, which relate terms describing systems with the same functional and performance properties.

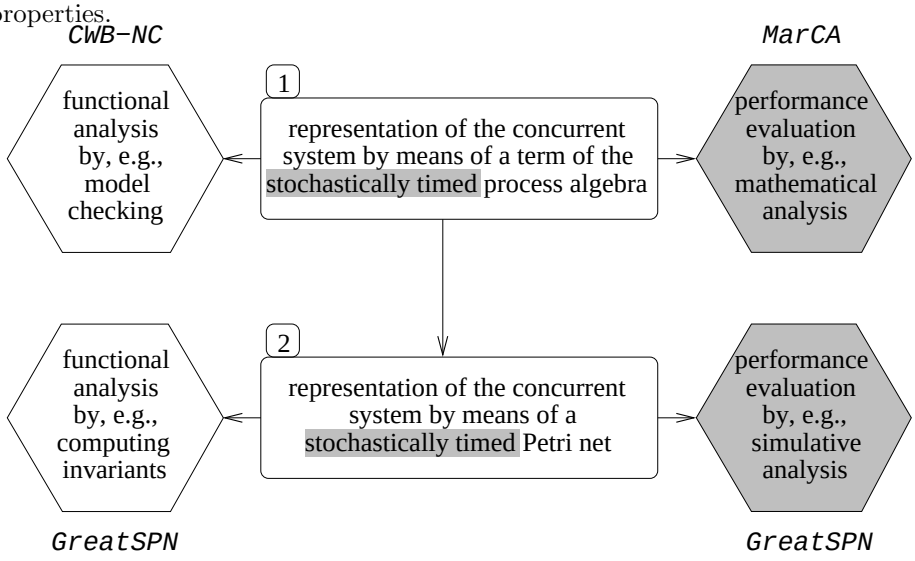


Figure 1.1: Integrated approach

The purpose of this thesis is to suitably combine stochastically timed process algebras and stochastically timed Petri nets so as to devise a formal approach [24, 27] to modeling and analyzing concurrent systems which should allow us to cope with the problems cited in the previous section. The approach we propose in this thesis results in three orthogonal integrations:

(i) The first integration relates the two different formalisms, hence two different views of concurrent systems according to [145], in order to profit from their complementary advantages. The abstract view is provided by process terms: they give an algebraic representation of system components and their interactions, whose semantic model is obtained by interleaving actions of concurrent components. The concrete view is provided instead by Petri nets: they give a machine like representation of systems with the explicit description of concurrency. This integration results in the two phases depicted in Fig. 1.1.

(ii) The second integration relates functional and performance aspects of concurrent systems, so that both of them are taken into account from the beginning of the design process thereby achieving our primary objective. This integration is depicted in Fig. 1.1 by means of the contrast between the nonshaded part and the shaded part.

(iii) The third integration consists of exploiting several existing tools (such as those [56, 180, 46] mentioned

in Fig. 1.1) tailored for specific purposes in order to analyze the various models. From a software development point of view, this means that the approach can be easily implemented by constructing a compiler for algebraic terms which yields their underlying semantic models and passes them to the already existing analysis tools.

Let us explain in more detail the two phases depicted in Fig. 1.1 in light of the three orthogonal integrations mentioned above.

1. The first phase requires the designer to specify the concurrent system as a term of the stochastically timed process algebra. Because of compositionality, the designer is allowed to develop the algebraic representation of the system in a modular way: every subsystem can be modeled separately, then these models are combined through the operators of the algebra. From the algebraic representation, an integrated interleaving semantic model is automatically derived in the form of a transition system labeled with both the type and the duration of the actions. The integrated interleaving semantic model can be analyzed as a whole by a notion of integrated equivalence or projected on a functional semantic model and a performance semantic model that can be analyzed by means of tools like CWB-NC [56] and MarCA [180], respectively.

   The functional analysis can be carried out by resorting to methods such as equivalence checking, preorder checking, and model checking [55]. Equivalence checking verifies whether a process term meets the specification of a given system, in the case when the specification is a process term as well, by proving that the term is equivalent to its specification. Preorder checking requires that the specification is still a process term treated as the minimal requirement to be met, owing to the fact that specifications can contain don't care points. Model checking requires specifications to be formalized as modal or temporal logic formulas to be satisfied, expressing assertions about safety, liveness, or fairness constraints.

   The performance analysis permits obtaining quantitative measures by resorting to either the study of a Markov chain [115] derived from the algebraic specification or the simulation [188] of the algebraic specification itself. The study of the Markov chain, which consists of solving suitable equations, can be conducted to obtain stationary measures (such as the average system throughput), in which case numerical solution methods such as Gaussian elimination [180] are applied, as well as transient measures (such as the probability of reaching a certain state within a given time), in which case numerical solution methods such as uniformization [180] are employed. The simulative analysis of the algebraic specification, which consists of statistically inferring performance measures from several independent executions of the specification, may be advantageous if the stochastically timed process algebra is given a semantics in the operational style [158] so the construction of the whole state space can be avoided. This makes it possible to generate only the states that are needed as the simulation proceeds, thereby allowing to cope with huge or infinite state spaces.

2. The second phase consists of automatically obtaining from the algebraic representation of the system an equivalent representation in the form of a stochastically timed Petri net. The net representation may be advantageous from the analysis standpoint in that usually more compact than the integrated interleaving semantic model resulting from the algebraic representation, since concurrency is kept explicit instead of being simulated by alternative computations obtained by interleaving actions of concurrent components. Additionally, the net representation turns out to be useful whenever a less abstract representation is required highlighting dependencies, conflicts, and synchronizations among system activities, and helpful in detecting some functional properties (e.g., partial deadlock) that can be easily checked only in a distributed setting. Moreover, the net representation is also useful whenever it allows for the application of efficient solution techniques to derive performance measures. The functional and performance analysis of the net representation can be undertaken using tools like GreatSPN [46].

   The functional analysis aims at detecting behavioral and structural properties of nets [139], i.e. both properties depending on the initial marking of the net and properties depending only upon the structure of the net. The technique of net invariants [164] is frequently used to conduct a structural analysis. Such a technique consists of computing the solutions of linear equation systems based on the incidence matrix of the net under consideration. These solutions single out places that do not change their token count during transition firings or indicate how often each transition has to fire in order to reproduce a given marking. By means of these solutions, properties such as boundedness, liveness, and deadlock can be studied without generating the underlying state space.

   The performance analysis aims at determining efficiency measures. This can be done by means of structural analysis techniques allowing for the efficient computation of performance bounds at the net level, or by resorting to either the numerical solution of a Markov chain (like in the first phase) derived from the net representation or the simulation of the net representation itself by playing the token game.

Since the two phases above are complementary, the choice between them should be made according to the adequacy of the related representation with respect to the analysis of the system under consideration and the availability of the corresponding tools. In any case, the designer is suggested to start with an algebraic representation of the system in order to take advantage of compositionality of algebras and avoid graphical complexity of nets.

1.3 Extended Markovian Process Algebra

In order to realize the integrated approach of Fig. 1.1, we have to choose a class of stochastically timed Petri nets and then a stochastically timed process algebra having possibly the same expressive power. The class of stochastically timed Petri nets we have chosen is that of generalized stochastic Petri nets (GSPNs) [3] because

they have been extensively studied and successfully applied. Since in the literature there is no stochastically timed process algebra having the same expressive power as GSPNs, in this thesis we develop a new one, called Extended Markovian Process Algebra (EMPA) [29], on the basis of MTIPP [75] and PEPA [100], which is enriched with expressive features typical of GSPNs. The name of our algebra stems from the fact that action durations are mainly expressed by means of exponentially distributed random variables (hence Markovian), but it is also possible to express prioritized probabilistic actions having duration zero as well as actions whose duration is unspecified (hence Extended).

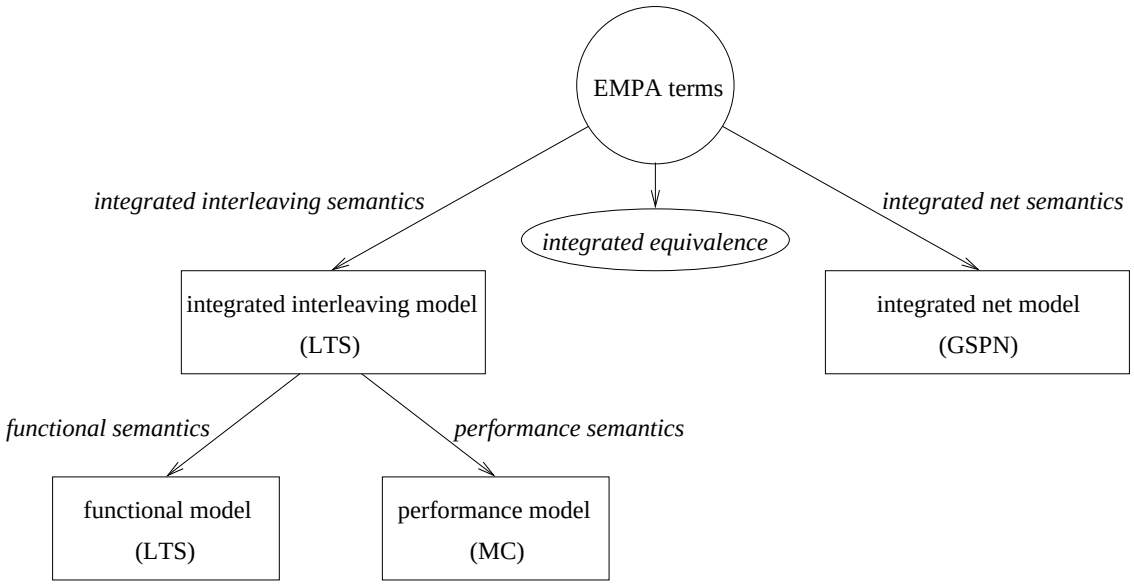


Figure 1.2: EMPA semantics and equivalence

In order to support the various phases and analyses of the integrated approach of Fig. 1.1, in this thesis EMPA is equipped with a suitable collection of operational semantics as well as a notion of integrated equivalence as depicted in Fig. 1.2. Each term has an integrated interleaving semantics represented by a labeled transition system (LTS for short), whose labels consist of both the type and the duration of the actions, and an integrated net semantics represented by a GSPN. From the integrated interleaving semantic model, two projected semantic models can be obtained: a functional model given by a LTS labeled only with the type of the actions and a performance model given by a Markov chain (MC for short). The integrated equivalence, which is based on ideas in [117, 92, 100, 41, 184, 133], relates terms describing systems with the same functional and performance properties and is defined in the bisimulation style [133] because of its relationship with the exact aggregation technique for MCs known as ordinary lumping [172].

The fact that EMPA can be given an integrated semantics through the interleaving approach in the same style as classical process algebras [133] and the fact that performance models underlying EMPA terms turn out to be MCs are, from the theoretical viewpoint, the major consequences of the restriction to exponentially distributed and zero durations. As far as the notion of integrated equivalence is concerned, the main result

we prove is that for a large class of terms it is the coarsest congruence contained in the intersection of two projected equivalences. This means that a qualitative analysis accounting for both functional and performance aspects, besides being convenient as it does not require the construction of projected models, can be conducted in a compositional way only on the integrated model.

Two extensions of EMPA are also proposed in this thesis. The former stems from addressing the problem of specifying performance measures and is obtained by including rewards [108] within actions. We prove that the theory developed for EMPA can be smoothly adapted to accommodate rewards, thus giving rise to a performance measure sensitive integrated equivalence. The latter extension is required to cope with systems where data plays a fundamental role and is obtained by adding the capability of modeling value passing among processes. Semantic rules producing compact versions of an improvement of the symbolic semantic models proposed in [81, 122, 121] are developed in order to exploit the inherent parametricity of value passing terms. Moreover, we show that value passing based expressions can be used to deal with generally distributed durations.

1.4 TwoTowers

Since the generation of the semantic models underlying EMPA terms and the checking of the integrated equivalence in Fig. 1.2 as well as the simulation of EMPA terms can be fully mechanized, in this thesis we also describe the implementation of a software tool, called TwoTowers [23], which supports the integrated approach in Fig. 1.1 in the Markovian case. [1]

Unlike other stochastically timed process algebra based tools, such as the PEPA Workbench [69] and the TIPP-tool [91], TwoTowers profits from well known software tools (CWB-NC [56], MarCA [180], and GreatSPN [46] as indicated in Fig. 1.1) to conduct the analysis of the semantic models it generates. This means that TwoTowers compiles EMPA descriptions of systems into their underlying semantic models, then these are passed to the related software tools in order to be analyzed.

This is advantageous both from the point of view of the implementors and from the point of view of the users. The development of TwoTowers is made easier and faster, because it is not necessary to write code for implementing the analysis routines, and users are provided with a full range of automated techniques implemented in widely used tools.

1.5 Structure of the Thesis

This thesis is organized as follows:

[1] The name of the tool stems from the two towers, Asinelli (97 m) and Garisenda (48 m), which are the symbol of the city of Bologna and date back to the 12th century.

- Chap. 2 contains some background about process algebras, Markov chains, and classical and stochastically timed Petri nets taken from [133, 106, 8, 33, 154, 158, 80, 82, 117, 115, 180, 172, 99, 108, 171, 79, 142, 157, 164, 139, 3, 137].

- Chap. 3 introduces EMPA by showing the syntax of its terms and formalizing the meaning of its operators. The concept of action is presented together with two classifications of actions based on their types and durations, respectively. Then the syntax of terms is defined and the meaning of each operator explained. Finally, after illustrating the execution policy adopted to choose among several simultaneously executable actions, the integrated interleaving semantics is given together with its functional and performance projections. The material presented in this chapter has been published in [27].

- Chap. 4 discusses the expressive power of EMPA by investigating its four kernels (nondeterministic, prioritized, probabilistic, and exponentially timed) and comparing them with process algebras appeared in the literature. The interplay of the probabilistic kernel with the exponentially timed kernel is recognized to allow phase type distributions to be modeled. Then, the expressiveness of the synchronization discipline of EMPA is compared with those of other stochastically timed process algebras and shown to be not so restrictive. Finally, several examples concerning the modeling of more and more complex queueing systems are given. The material presented in this chapter has been published in [10, 29].

- Chap. 5 introduces an integrated equivalence for EMPA. Such an equivalence is developed according to the bisimulation style, because of its relationship with the notion of ordinary lumping for MCs, and is shown to be for a large class of terms the coarsest congruence contained in the intersection of two projected equivalences. A sound and complete axiomatization of the integrated equivalence is given together with a checking algorithm. The material presented in this chapter has been published in [28, 29, 39].

- Chap. 6 defines two integrated net semantics for EMPA. The integrated location oriented net semantics is based on an extension of the rules for the integrated interleaving semantics such that the syntactical structure of terms is encoded within places. The integrated label oriented net semantics consists instead of a single rule and produces smaller nets because places do not fully account for the syntactical structure of terms. Both semantics are shown to be sound with respect to the integrated interleaving semantics. The material presented in this chapter has been published in [25, 27, 19].

- Chap. 7 extends EMPA with rewards in order to allow for the high level specification of performance measures such as throughput, utilization, and average response time. An algebraic method based on the idea of including yield and bonus rewards within actions is presented and the semantics and equivalence developed for EMPA are smoothly adapted accordingly in order to make them performance measure sensitive. The material presented in this chapter has been published in [14, 16].

- Chap. 8 extends EMPA with value passing in order to deal with systems where data plays a fundamental role. Conceptually, this is achieved in three steps. First, the syntax is extended in the usual way by adding input and output actions, conditional operators, and constant parameters. Second, the symbolic transition graphs with lookahead assignment are proposed as semantic models, which are an improvement of the symbolic models proposed in [81, 122, 121], and symbolic semantic rules are defined which map terms onto symbolic models. Such symbolic semantic rules do not make any assumption on variable names, are correct w.r.t. both the usual concrete semantic rules and the assignment evaluation order, and produce compact symbolic models. Third, a suitable integrated equivalence is defined on the symbolic models. Furthermore, it is shown that by means of value passing based expressions it is possible to deal with systems where generally distributed durations come into play. The material presented in this chapter has been published in [12, 17].

- Chap. 9 describes the architecture and the implementation of TwoTowers. The material presented in this chapter has been published in [23].

- Chap. 10 illustrates several case studies concerning communication systems (alternating bit protocol [26, 27, 12, 17], CSMA/CD [13, 16], token ring [18], ATM switch [15, 16], adaptive mechanism for packetized audio [169, 30]) and distributed algorithms (randomized algorithm for dining philosophers [23], mutual exclusion algorithms) modeled with EMPA and analyzed with TwoTowers. The purpose of such case studies is to demonstrate the adequacy of the integrated approach, the expressiveness of EMPA, and the importance of developing automated tools such as TwoTowers to help designers in modeling and analyzing complex systems in a formal way.

- Chap. 11 concludes the thesis by making some comparisons with related work and outlining future research. In particular, the most challenging open problem is addressed: what happens if we drop the Markovian restriction thereby allowing for actions with generally distributed duration?

Proofs of results are shown in the Appendix A.

Chapter 2

Process Algebras, Markov Chains, and Petri Nets

In this chapter we provide the background which is necessary to understand the remainder of the thesis. This chapter is organized as follows. In Sect. 2.1 we present process algebras. In Sect. 2.2 we introduce Markov chains. In Sect. 2.3 we recall Petri nets.

2.1 Process Algebras

In this section we recall the theory of process algebras which will serve as a basis for the development of EMPA in Chap. 3 and 5. In Sect. 2.1.1 we present the operators of an example process algebra. In Sect. 2.1.2 we introduce rooted labeled transition systems as they are used as semantic models for algebraic terms. In Sect. 2.1.3 we provide the operational semantics for the example process algebra. Finally, in Sect. 2.1.4 we recall the notion of bisimulation equivalence together with its equational and logical characterizations.

2.1.1 Algebraic Operators

Process algebras are algebraic languages which support the compositional description of concurrent systems and the formal verification of their properties. The basic elements of any process algebra are its actions, which represent activities carried out by the systems being modeled, and its operators (among which a parallel composition operator), which are used to compose algebraic descriptions. In this section we introduce an example process algebra based on a combination of the operators of well known process algebras such as CCS [133], CSP [106], ACP [8], and LOTOS [33]. Such a combination will be used in Sect. 3.2 to define the syntax for EMPA.

Let Act be a set of *actions* ranged over by $a, b, \ldots$ which contains a distinguished action τ denoting *unobservable* activities. Furthermore, let $Const$ be a set of *constants* ranged over by $A, B, \ldots$ and $ARFun = \{\varphi : Act \longrightarrow Act \mid \varphi^{-1}(\tau) = \{\tau\}\}$ be a set of *action relabeling functions* ranged over by $\varphi, \varphi', \ldots$.

Definition 2.1 *The set $\mathcal{L}$ of process terms is generated by the following syntax*

$$E ::= \underline{0} \mid a.E \mid E/L \mid E[\varphi] \mid E + E \mid E \parallel_S E \mid A$$

where $L, S \subseteq Act - \{\tau\}$. *The set* $\mathcal{L}$ *will be ranged over by* $E, F, \ldots$. ■

The *null term* "$\underline{0}$" is the term that cannot execute any action.

The *action prefix operator* "$a._-$" denotes the sequential composition of an action and a term. Term $a.E$ can execute action a and then behaves as term E.

The *abstraction operator* "$\_/L$" make actions unobservable. Term E/L behaves as term E except that each executed action a is hidden, i.e. turned into τ, whenever $a \in L$. This operator provides a means to encapsulate or ignore information.

The *relabeling operator* "$\_[\varphi]$" changes actions. Term $E[\varphi]$ behaves as term E except that each executed action a becomes $\varphi(a)$. This operator provides a means to obtain more compact algebraic descriptions, since it enables the reuse of terms in situations demanding the same functionality up to action names.

The *alternative composition operator* "$\_ + \_$" expresses a nondeterministic choice between two terms. Term $E_1 + E_2$ behaves as either term E_1 or term E_2 depending on whether an action of E_1 or an action of E_2 is executed.

The *parallel composition operator* "$\_ \parallel_S \_$" expresses the concurrent execution of two terms according to the following synchronization discipline: two actions can synchronize if and only if they have the same observable type in S, which becomes the resulting type. Term $E_1 \parallel_S E_2$ asynchronously executes actions of E_1 or E_2 not belonging to S and synchronously executes actions of E_1 and E_2 belonging to S if the requirement above is met.

In order to avoid ambiguities, we assume the binary operators to be left associative and we introduce the following operator precedence relation: abstraction = relabeling > action prefix > alternative composition > parallel composition. Parentheses can be used to alter associativity and precedence. Moreover, given a term E, we denote by $sort(E)$ the set of actions occurring in the action prefix operators of E.

Finally, let partial function $Def : Const \rightharpoonup \mathcal{L}$ be a set of *constant defining equations* of the form $A \triangleq E$. As an example of application of operators and defining equations, we consider a two place buffer, which can be modeled as the parallel composition of two communicating places as follows:

$$\begin{aligned} Buffer &\triangleq Place_1 \parallel_{\{move\}} Place_2 \\ Place_1 &\triangleq accept.move.Place_1 \\ Place_2 &\triangleq move.release.Place_2 \end{aligned}$$

The first place accepts items from the outside and moves them to the second place, while the second place receives items from the first place (through the synchronization on *move*) and gives them to the outside.

In order to guarantee the correctness of recursive definitions, we restrict ourselves to terms that are closed and guarded w.r.t. Def, i.e. those terms such that each constant occurring in them has a defining equation and appears in the context of an action prefix operator. This rules out undefined constants, meaningless

definitions such as $A \triangleq A$, and infinitely branching terms such as $A \triangleq a.\underline{0} \parallel_\emptyset A$ whose executable actions cannot be computed in finite time. Let us denote by "$\_ \equiv \_$" the syntactical equality between terms and by "$\_\ st\ \_$" the relation subterm-of.

Definition 2.2 *The term $E\langle A := E'\rangle$ obtained from $E \in \mathcal{L}$ by replacing each occurrence of A with E', where $A \triangleq E' \in Def$, is defined by induction on the syntactical structure of E as follows:*

$$\begin{aligned}
\underline{0}\langle A := E'\rangle &\equiv \underline{0} \\
(a.E)\langle A := E'\rangle &\equiv a.E\langle A := E'\rangle \\
E/L\langle A := E'\rangle &\equiv E\langle A := E'\rangle/L \\
E[\varphi]\langle A := E'\rangle &\equiv E\langle A := E'\rangle[\varphi] \\
(E_1 + E_2)\langle A := E'\rangle &\equiv E_1\langle A := E'\rangle + E_2\langle A := E'\rangle \\
(E_1 \parallel_S E_2)\langle A := E'\rangle &\equiv E_1\langle A := E'\rangle \parallel_S E_2\langle A := E'\rangle \\
B\langle A := E'\rangle &\equiv \begin{cases} E' & \text{if } B \equiv A \\ B & \text{if } B \not\equiv A \end{cases}
\end{aligned}$$

■

Definition 2.3 *The set of terms obtained from $E \in \mathcal{L}$ by repeatedly replacing constants by the right hand side terms of their defining equations in Def is defined by*

$$Subst_{Def}(E) = \bigcup_{n \in \mathbb{N}} Subst^n_{Def}(E)$$

where

$$Subst^n_{Def}(E) = \begin{cases} \{E\} & \text{if } n = 0 \\ \{F \in \mathcal{L} \mid F \equiv G\langle A := E'\rangle \wedge G \in Subst^{n-1}_{Def}(E) \wedge A\ st\ G \wedge A \triangleq E' \in Def\} & \text{if } n > 0 \end{cases}$$

■

Definition 2.4 *The set of constants occurring in $E \in \mathcal{L}$ w.r.t. Def is defined by*

$$Const_{Def}(E) = \{A \in Const \mid \exists F \in Subst_{Def}(E).\ A\ st\ F\}$$

■

Definition 2.5 *A term $E \in \mathcal{L}$ is closed and guarded w.r.t. Def if and only if for all $A \in Const_{Def}(E)$*

- *A is equipped in Def with defining equation $A \triangleq E'$, and*
- *there exists $F \in Subst_{Def}(E')$ such that, whenever an instance of a constant B satisfies $B\ st\ F$, then the same instance satisfies $B\ st\ a.G\ st\ F$.*

We denote by $\mathcal{G}$ the set of terms in $\mathcal{L}$ that are closed and guarded w.r.t. Def. ■

2.1.2 Rooted Labeled Transition Systems

In this section we present the definition of labeled transition system together with some related notions [154]. These mathematical models, which are essentially state transition graphs, are commonly adopted when defining the semantics for a process algebra in the operational style [158], and will be used in Sect. 3.4 to define the integrated interleaving semantics for EMPA.

Definition 2.6 *A rooted labeled transition system (LTS) is a tuple*

$$(S, U, \longrightarrow, s_0)$$

where:

- S *is a set whose elements are called states.*
- U *is a set whose elements are called labels.*
- $\longrightarrow \subseteq S \times U \times S$ *is called transition relation.*
- $s_0 \in S$ *is called the initial state.* ■

In the graphical representation of a LTS, states are drawn as black dots and transitions are drawn as arrows between pairs of states with the appropriate labels; the initial state is pointed to by an unlabeled arrow.

Below we recall two notions of equivalence defined for LTSs. The former, isomorphism, relates two LTSs if they are structurally equal. This is formalized by requiring the existence of a label preserving relation which is bijective, i.e. a bijection between the two state spaces such that any pair of corresponding states have identically labeled transitions toward any pair of corresponding states. The latter equivalence, bisimilarity, is coarser than isomorphism as it relates also LTSs which are not structurally equal provided that they are able to simulate each other. This is formalized by requiring the existence of a label preserving relation between the two state spaces which is not necessarily bijective.

Definition 2.7 *Let* $Z_k = (S_k, U, \longrightarrow_k, s_{0k})$, $k \in \{1, 2\}$, *be two LTSs.*

- Z_1 *is isomorphic to* Z_2 *if and only if there exists a bijection* $\beta : S_1 \longrightarrow S_2$ *such that:*
  - $\beta(s_{01}) = s_{02}$;
  - *for all* $s, s' \in S_1$ *and* $u \in U$

$$s \xrightarrow{u}_1 s' \iff \beta(s) \xrightarrow{u}_2 \beta(s')$$

- Z_1 *is bisimilar to* Z_2 *if and only if there exists a relation* $\mathcal{B} \subseteq S_1 \times S_2$ *such that:*
  - $(s_{01}, s_{02}) \in \mathcal{B}$;

– *for all* $(s_1, s_2) \in \mathcal{B}$ *and* $u \in U$

* *whenever* $s_1 \xrightarrow{u}_1 s_1'$, *then* $s_2 \xrightarrow{u}_2 s_2'$ *and* $(s_1', s_2') \in \mathcal{B}$;
* *whenever* $s_2 \xrightarrow{u}_2 s_2'$, *then* $s_1 \xrightarrow{u}_1 s_1'$ *and* $(s_1', s_2') \in \mathcal{B}$. ■

2.1.3 Operational Semantics

The semantics of our example process algebra is defined in this section following the structured operational approach [158]. This means that inference rules are given for each operator which define an abstract interpreter for the language. More precisely, the semantics is defined through the transition relation $\longrightarrow$ which is the least subset of $\mathcal{G} \times Act \times \mathcal{G}$ satisfying the inference rules in Table 2.1. Such rules, which formalize the meaning of each operator informally given in Sect. 2.1.1, yield LTSs where states are in correspondence with terms and transitions are labeled with actions. Given a term E, the outgoing transitions of the state corresponding to E are generated by proceeding by induction on the syntactical structure of E applying at each step the appropriate semantic rule until an action prefix operator is encountered or no rule can be used. This can be done in finite time because of the restriction to closed and guarded terms.

As an example, when computing the transitions for the state associated with *Buffer*, we apply first of all one of the inference rules for the parallel composition operator, say the first one. Then $Place_1$ is examined and the axiom for the action prefix operator is applied thus determining *accept* as executed action and $move.Place_1$ as term associated with the derivative state. The inference is concluded by returning to *Buffer* and determining *accept* as executed action (since $accept \notin \{move\}$) and $move.Place_1 \parallel_{\{move\}} Place_2$ as term associated with the derivative state. This is the only outgoing transition for the considered state since initially applying one of the other inference rules for the parallel composition operator results in a violation of their side conditions.

We recall that the abstraction operator, the relabeling operator, and the parallel composition operator are called static operators because they appear also in the derivative term of the consequence of the related semantic rules. By contrary, the action prefix operator and the alternative composition operator are classified as dynamic operators. We say that a term is sequential if every occurrence of static operators is within the scope of an action prefix operator. It is worth noting that the LTS underlying $E \in \mathcal{G}$ is finite if all of the subterms of terms in $Subst_{Def}(E)$ whose outermost operator is static contains no recursive constants.

Definition 2.8 *The operational interleaving semantics of* $E \in \mathcal{G}$ *is the LTS*

$$\mathcal{I}[\![E]\!] = (S_E, Act, \longrightarrow_E, E)$$

where:

- S_E *is the least subset of* $\mathcal{G}$ *such that:*

  – $E \in S_E$;

$$a.E \xrightarrow{a} E$$

$$\frac{E \xrightarrow{a} E'}{E/L \xrightarrow{a} E'/L} \text{ if } a \notin L \qquad \frac{E \xrightarrow{a} E'}{E/L \xrightarrow{\tau} E'/L} \text{ if } a \in L$$

$$\frac{E \xrightarrow{a} E'}{E[\varphi] \xrightarrow{\varphi(a)} E'[\varphi]}$$

$$\frac{E_1 \xrightarrow{a} E'}{E_1 + E_2 \xrightarrow{a} E'} \qquad \frac{E_2 \xrightarrow{a} E'}{E_1 + E_2 \xrightarrow{a} E'}$$

$$\frac{E_1 \xrightarrow{a} E_1'}{E_1 \parallel_S E_2 \xrightarrow{a} E_1' \parallel_S E_2} \text{ if } a \notin S \qquad \frac{E_2 \xrightarrow{a} E_2'}{E_1 \parallel_S E_2 \xrightarrow{a} E_1 \parallel_S E_2'} \text{ if } a \notin S$$

$$\frac{E_1 \xrightarrow{a} E_1' \quad E_2 \xrightarrow{a} E_2'}{E_1 \parallel_S E_2 \xrightarrow{a} E_1' \parallel_S E_2'} \text{ if } a \in S$$

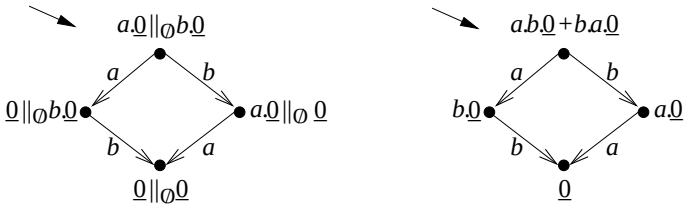
$$\frac{E \xrightarrow{a} E'}{A \xrightarrow{a} E'} \text{ if } A \triangleq E$$

Table 2.1: Operational semantics for the example process algebra

– *if* $E_1 \in S_E$ *and* $E_1 \xrightarrow{a} E_2$, *then* $E_2 \in S_E$.

- $\longrightarrow_E$ *is the restriction of* $\longrightarrow$ *to* $S_E \times Act \times S_E$. ■

We talk about interleaving semantics [133] because every parallel computation is represented in the LTS by means of alternative sequential computations obtained by interleaving actions executed by concurrent processes. As an example, note that $\mathcal{I}[\![a.\underline{0} \parallel_\emptyset b.\underline{0}]\!]$ is isomorphic to $\mathcal{I}[\![a.b.\underline{0} + b.a.\underline{0}]\!]$:



2.1.4 Bisimulation Equivalence

In this section we define a notion of equivalence for the example process algebra and we provide its equational and logical characterizations. The purpose of such an equivalence is to relate those terms representing systems which, though structurally different, behave the same from the point of view of an external observer. Following [133], the equivalence is given in the bisimulation style.

Definition 2.9 *A relation $\mathcal{B} \subseteq \mathcal{G} \times \mathcal{G}$ is a strong bisimulation if and only if, whenever $(E_1, E_2) \in \mathcal{B}$, then for all $a \in Act$:*

- *whenever $E_1 \stackrel{a}{\longrightarrow} E_1'$, then $E_2 \stackrel{a}{\longrightarrow} E_2'$ and $(E_1', E_2') \in \mathcal{B}$;*
- *whenever $E_2 \stackrel{a}{\longrightarrow} E_2'$, then $E_1 \stackrel{a}{\longrightarrow} E_1'$ and $(E_1', E_2') \in \mathcal{B}$.* ■

The union of all the strong bisimulations can be shown to be an equivalence relation which coincides with the largest strong bisimulation. Such a relation, denoted $\sim$, is defined to be the *strong bisimulation equivalence* and can be shown to be a congruence w.r.t. all the operators as well as recursive definitions. As an example, if $E_1 \sim E_2$ then $E_1 \parallel_S F \sim E_2 \parallel_S F$ for all F and S. This allows for the compositional manipulation of algebraic terms: given a term, replacing one of its subterms with a bisimilar subterm does not alter the whole meaning because the newly generated term is still bisimilar to the original one.

The effect of $\sim$ can be better seen through its equational characterization. Given that $\sim$ is a congruence, we present in Table 2.2 a set $\mathcal{A}$ of axioms for the set of nonrecursive terms in $\mathcal{G}$ on which a deductive system $Ded(\mathcal{A})$ can be built which satisfies reflexivity ($E = E$), symmetry ($E_1 = E_2 \Longrightarrow E_2 = E_1$), transitivity ($E_1 = E_2 \wedge E_2 = E_3 \Longrightarrow E_1 = E_3$), and substitutivity ($E_{1,1} = E_{2,1} \wedge \ldots \wedge E_{1,n} = E_{2,n} \Longrightarrow op(E_{1,1}, \ldots, E_{1,n}) = op(E_{2,1}, \ldots, E_{2,n})$ for each n-ary operator op of the algebra) [80]. It can be shown that the axiomatization above, which basically allows terms to be rewritten in such a way that only action prefix and alternative composition operators occur, is sound and complete w.r.t. $\sim$, which means that in $\mathcal{A}$ it can be proved $E_1 = E_2$ if and only if $E_1 \sim E_2$. We observe that axiom $\mathcal{A}_{11}$, also known as the expansion law, is the equational characterization of the notion of interleaving.

It is worth noting that *equivalence checking* is one of the major verification techniques in the process algebra setting. Given a term representing a system, another simpler term is considered which is taken to be a specification of the system and the purpose is to verify whether the given term is equivalent to the specification. As an example, the two place buffer may be specified as follows by considering the possible sequences of actions

$$\begin{aligned} BufferSpec &\triangleq accept.BufferSpec_1 \\ BufferSpec_1 &\triangleq move.(accept.BufferSpec_2 + release.BufferSpec) \\ BufferSpec_2 &\triangleq release.BufferSpec_1 \end{aligned}$$

and it can be shown that $Buffer \sim BufferSpec$, which means that the parallel composition of two interacting places described by *Buffer* complies with the specification. In the case of a similar technique known as

$$
\begin{array}{lrcl}
(\mathcal{A}_1) & (E_1 + E_2) + E_3 & = & E_1 + (E_2 + E_3) \\
(\mathcal{A}_2) & E_1 + E_2 & = & E_2 + E_1 \\
(\mathcal{A}_3) & E + \underline{0} & = & E \\
(\mathcal{A}_4) & E + E & = & E \\
 & & & \\
(\mathcal{A}_5) & \underline{0}/L & = & \underline{0} \\
(\mathcal{A}_6) & (a.E)/L & = & \begin{cases} a.E/L & \text{if } a \notin L \\ \tau.E/L & \text{if } a \in L \end{cases} \\
(\mathcal{A}_7) & (E_1 + E_2)/L & = & E_1/L + E_2/L \\
 & & & \\
(\mathcal{A}_8) & \underline{0}[\varphi] & = & \underline{0} \\
(\mathcal{A}_9) & (a.E)[\varphi] & = & \varphi(a).E[\varphi] \\
(\mathcal{A}_{10}) & (E_1 + E_2)[\varphi] & = & E_1[\varphi] + E_2[\varphi] \\
 & & & \\
(\mathcal{A}_{11}) & \sum_{i \in I_1} a_i.E_i \parallel_S \sum_{i \in I_2} a_i.E_i & = & \sum_{j \in J_1} a_j.(E_j \parallel_S \sum_{i \in I_2} a_i.E_i) + \\
 & & & \sum_{j \in J_2} a_j.(\sum_{i \in I_1} a_i.E_i \parallel_S E_j) + \\
 & & & \sum_{k \in K_1} \sum_{h \in H_k} a_k.(E_k \parallel_S E_h) + \\
 & & & \sum_{k \in K_2} \sum_{h \in H_k} a_k.(E_h \parallel_S E_k) \\
 & & & \\
 & \text{where } I_1 \cap I_2 & = & \emptyset \\
 & J_1 & = & \{i \in I_1 \mid a_i \notin S\} \\
 & J_2 & = & \{i \in I_2 \mid a_i \notin S\} \\
 & K_1 & = & \{k \in I_1 \mid a_k \in S \wedge \exists h \in I_2.\, a_h = a_k\} \\
 & K_2 & = & \{k \in I_2 \mid a_k \in S \wedge \exists h \in I_1.\, a_h = a_k\} \\
 & H_k & = & \begin{cases} \{h \in I_2 \mid a_h = a_k\} & \text{if } k \in K_1 \\ \{h \in I_1 \mid a_h = a_k\} & \text{if } k \in K_2 \end{cases}
\end{array}
$$

Table 2.2: Axioms for strong bisimulation equivalence

preorder checking, the specification is still a term but it is treated as the minimal requirement to be met, owing to the fact that it can contain don't care points. Finally, we cite a different verification technique known as *model checking*, which is based on formalizing safety and liveness properties by means of modal or temporal logic formulas and verifying whether a given term satisfies such formulas. As an example, the syntax of Hennessy-Milner logic [82] formulas is given by

$$\Phi ::= tt \mid ff \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid [a]\Phi \mid \langle a\rangle\Phi$$

and their semantics is given by the following satisfaction relation

$$\begin{aligned}
E &\models tt \\
E &\not\models ff \\
E \models \neg\Phi &\iff E \not\models \Phi \\
E \models \Phi_1 \wedge \Phi_2 &\iff E \models \Phi_1 \wedge E \models \Phi_2 \\
E \models \Phi_1 \vee \Phi_2 &\iff E \models \Phi_1 \vee E \models \Phi_2 \\
E \models \langle a\rangle\Phi &\iff \exists E'.\, E \xrightarrow{a} E' \wedge E' \models \Phi \\
E \models [a]\Phi &\iff \forall E'.\, E \xrightarrow{a} E' \Longrightarrow E' \models \Phi
\end{aligned}$$

As far as modal operators are concerned, $E \models \langle a\rangle\Phi$ means that it is possible for E to execute a and evolve into a state where Φ is satisfied, while $E \models [a]\Phi$ means that if E executes a then the derivative term necessarily satisfies Φ. The reason why we have recalled this particular modal logic is that it is the logical characterization of $\sim$, i.e. it can be proved that $E_1 \sim E_2$ if and only if E_1 and E_2 satisfy the same subset of Hennessy-Milner logic formulas.

We conclude by observing that bisimulation equivalence treats τ actions in the same way as observable actions. Since τ actions cannot be seen by an external observer, a less strong notion of bisimulation equivalence called *weak bisimulation equivalence* ($\approx$) can be set up which abstracts from τ actions and still is a congruence w.r.t. the parallel composition operator [133]. Instead of giving the definition of such an equivalence, we only provide the axioms to add to those in Table 2.2 in order to obtain its equational characterization:

$$\begin{aligned}
\tau.E &= E \\
a.\tau.E &= a.E \\
E + \tau.E &= \tau.E \\
a.(E_1 + \tau.E_2) + a.E_2 &= a.(E_1 + \tau.E_2)
\end{aligned}$$

Since any congruence w.r.t. the parallel composition operator is useful for minimization purposes, it turns out that $\approx$ is more convenient than $\sim$. As an example, moving items from the first place to the second place in the two place buffer can be regarded as an internal action, so the buffer may be specified as follows by considering the possible sequences of observable actions

$$\begin{aligned}
BufferSpec' &\triangleq accept.BufferSpec'_1 \\
BufferSpec'_1 &\triangleq accept.BufferSpec'_2 + release.BufferSpec' \\
BufferSpec'_2 &\triangleq release.BufferSpec'_1
\end{aligned}$$

and it can be shown that $Buffer/\{move\} \approx BufferSpec'$ with $BufferSpec'$ having fewer states than $BufferSpec$.

2.2 Markov Chains

In this section we recall the theory of Markov chains which will be used in Chap. 3 to define the performance semantics for EMPA. In Sect. 2.2.1 we propose the mathematical model given by probabilistically rooted labeled transition systems which provide a graph theoretic representation of Markov chains. In Sect. 2.2.2 and 2.2.3 we present continuous time and discrete time Markov chains, respectively. In Sect. 2.2.4 we introduce the notion of ordinary lumping. In Sect. 2.2.5 we present semi Markov chains. In Sect. 2.2.6 we explain how to specify performance measures via reward structures. Finally, in Sect. 2.2.7 we recall queueing systems.

2.2.1 Probabilistically Rooted Labeled Transition Systems

In this section we propose an extension of LTSs where the initial state is replaced by a probability mass function which determines for each state the probability that it is the initial one. Moreover, a holding time function is added in order to distinguish among vanishing states, tangible states, and absorbing states. Vanishing states have zero sojourn time and the labels of their outgoing transitions are the execution probabilities of the transitions themselves. Tangible states have a positive sojourn time and the labels of their outgoing transitions are the execution rates of the transitions themselves. Absorbing states have an infinite sojourn time because they have no outgoing transitions. By means of this interpretation, we are able to represent discrete time Markov chains (where tangible states are absent), continuous time Markov chains (where vanishing states are absent), and semi Markov chains (where both vanishing and tangible states coexist). These mathematical models will be used in Sect. 3.4 to define the performance semantics for EMPA.

Definition 2.10 *A probabilistically rooted labeled transition system (p-LTS) is a tuple*

$$(S, \mathbb{R}_+, \longrightarrow, P, H)$$

where:

- S *and* $\longrightarrow$ *are defined as for a LTS.*
- $P : S \longrightarrow \mathbb{R}_{[0,1]}$*, called initial state probability function, is such that* $\sum_{s \in S} P(s) = 1$*.*
- $H : S \longrightarrow \{v, t, a\}$*, called holding time function, is such that* [1]

$$\begin{array}{lcll} H(s) & = & t & \text{if } \sum\{\!|\, \lambda \in \mathbb{R}_+ \mid \exists s' \in S.\, s \xrightarrow{\lambda} s' \,|\!\} \neq 1 \\ H(s) & \in & \{t, v\} & \text{if } \sum\{\!|\, \lambda \in \mathbb{R}_+ \mid \exists s' \in S.\, s \xrightarrow{\lambda} s' \,|\!\} = 1 \\ H(s) & = & a & \text{if } \nexists s' \in S.\, \nexists \lambda \in \mathbb{R}_+.\, s \xrightarrow{\lambda} s' \end{array}$$

■

[1] $\{\!|$ and $|\!\}$ denote multiset delimiters.

In the graphical representation of a p-LTS, states and transitions are drawn as in a LTS and each state is labeled with its initial state probability (unless it is zero) as well as its holding time (unless it is clear from the context).

The notions of equivalence for p-LTSs (p-isomorphism and p-bisimilarity) carry over from the corresponding notions for LTSs. In particular, p-bisimilarity is developed according to [117], so it considers two p-LTSs to be equivalent if any pair of corresponding states have the same aggregated label to reach the same equivalence class of states. We recall that, given an equivalence relation $\mathcal{R}$ over set S, the set of equivalence classes of S w.r.t. $\mathcal{R}$ is denoted $S/\mathcal{R}$.

Definition 2.11 *Let $Z_k = (S_k, \mathbb{R}_+, \longrightarrow_k, P_k, H_k)$, $k \in \{1,2\}$, be two p-LTSs.*

- *Z_1 is p-isomorphic to Z_2 if and only if there exists a bijection $\beta : S_1 \longrightarrow S_2$ such that:*
  - *for all $s \in S_1$*
$$\begin{aligned} P_1(s) &= P_2(\beta(s)) \\ H_1(s) &= H_2(\beta(s)) \end{aligned}$$
  - *for all $s, s' \in S_1$ and $\lambda \in \mathbb{R}_+$*
$$s \xrightarrow{\lambda}_1 s' \iff \beta(s) \xrightarrow{\lambda}_2 \beta(s')$$
- *Z_1 is p-bisimilar to Z_2 if and only if there exists a relation $\mathcal{B} \subseteq S_1 \times S_2$ with reflexive, symmetric and transitive closure $\mathcal{B}' \subseteq (S_1 \cup S_2) \times (S_1 \cup S_2)$ such that:*
  - *for all $C \in (S_1 \cup S_2)/\mathcal{B}'$*
$$\sum_{s \in C \cap S_1} P_1(s) = \sum_{s \in C \cap S_2} P_2(s)$$
  - *whenever $(s_1, s_2) \in \mathcal{B}$, then*
$$H_1(s_1) = H_2(s_2)$$
  - *whenever $(s_1, s_2) \in \mathcal{B}$, then for all $C \in (S_1 \cup S_2)/\mathcal{B}'$*
$$\sum \{\!|\, \lambda \in \mathbb{R}_+ \mid \exists s_1' \in C \cap S_1.\, s_1 \xrightarrow{\lambda}_1 s_1' \,|\!\} = \sum \{\!|\, \lambda \in \mathbb{R}_+ \mid \exists s_2' \in C \cap S_2.\, s_2 \xrightarrow{\lambda}_2 s_2' \,|\!\}$$ ■

2.2.2 Continuous Time Markov Chains

A Markov chain is a stochastic process with discrete state space which is characterized by the memoryless property: the probability of being in a certain state at a given instant depends only on the current state instead of the whole history of visited states. Below we consider the continuous time case.

Definition 2.12 *A continuous time Markov chain (CTMC) [115] is a continuous time stochastic process* $X = \{X(t) \mid t \in \mathbb{R}_{[0,\infty[}\}$ *with discrete state space* S_X *such that, for each* $n \in \mathbb{N}_+$, $i_0, \ldots, i_{n-1}, i_n \in S_X$, $t_0, \ldots, t_{n-1}, t_n \in \mathbb{R}_{[0,\infty[}$ *where* $t_0 < \ldots < t_{n-1} < t_n$, *it turns out*

$$\Pr\{X(t_n) = i_n \mid X(t_{n-1}) = i_{n-1} \wedge \ldots \wedge X(t_0) = i_0\} = \Pr\{X(t_n) = i_n \mid X(t_{n-1}) = i_{n-1}\}$$ ■

Definition 2.13 *Let* X *be a CTMC.*

- *The transition matrix of* X *from time* $t \in \mathbb{R}_{[0,\infty[}$ *to time* $t' > t$ *is matrix* $\mathbf{P}_X(t,t')$ *defined by*

$$\mathbf{P}_X(t,t') = [\Pr\{X(t') = j \mid X(t) = i\}]_{i,j \in S_X}$$

- *The infinitesimal generator of* X *at time* $t \in \mathbb{R}_{[0,\infty[}$ *is matrix* $\mathbf{Q}_X(t)$ *defined by*

$$\mathbf{Q}_X(t) = [q_{i,j}(t)]_{i,j \in S_X} = \lim_{\Delta t \to 0} \frac{\mathbf{P}_X(t, t+\Delta t) - \mathbf{I}}{\Delta t}$$

  where $\mathbf{I}$ *is the identity matrix.*

- X *is a homogeneous CTMC (HCTMC) if and only if its infinitesimal generator is independent of the time.*

- *The state probability distribution function of* X *at time* $t \in \mathbb{R}_{[0,\infty[}$ *is vector* $\boldsymbol{\pi}_X(t)$ *defined by*

$$\boldsymbol{\pi}_X(t) = [\Pr\{X(t) = i\}]_{i \in S_X}$$

- *The stationary probability distribution function of* X *is vector* $\boldsymbol{\pi}_X$ *defined by*

$$\boldsymbol{\pi}_X = \lim_{t \to \infty} \boldsymbol{\pi}_X(t)$$ ■

In the continuous time case, the exponential distribution ($F(t) = 1 - e^{-\lambda \cdot t}$ for all $t \in \mathbb{R}_+$, with $\lambda \in \mathbb{R}_+$ called the rate) is the only distribution enjoying the memoryless property. Therefore, given a HCTMC X, the rates of its transitions can be interpreted as the rates of the exponential distributions describing the durations of the transitions themselves, and it can be proved that the sojourn time of state i is exponentially distributed with rate $\sum_{j \in S_X - \{i\}} q_{i,j}$ and the execution probability of the transition from state i to state j (given that the current state is i) is $q_{i,j} / \sum_{k \in S_X - \{i\}} q_{i,k}$. Moreover, if the state space is finite, it can be shown that the state probability distribution function of X at time $t \in \mathbb{R}_{[0,\infty[}$ is given by

$$\boldsymbol{\pi}_X(t) = \boldsymbol{\pi}_X(0) \cdot e^{\mathbf{Q}_X \cdot t}$$

while the stationary probability distribution function of X can be determined (if the transitions of X result in a strongly connected graph) by solving the global balance equation

$$\begin{aligned} \boldsymbol{\pi}_X \cdot \mathbf{Q}_X &= \mathbf{0} \\ \sum_{i \in S_X} \pi_i &= 1 \end{aligned}$$

Suitable numerical techniques are available for both equation systems [180].

The HCTMC X can equivalently be represented by means of the p-LTS

$$(S_X, \mathbb{R}_+, \{(i, q_{i,j}, j) \in S_X \times \mathbb{R}_+ \times S_X\}, \boldsymbol{\pi}_X(0), H_X)$$

where

$$H_X(i) = \begin{cases} t & \text{if } \exists j \in S_X . \, q_{i,j} > 0 \\ a & \text{otherwise} \end{cases}$$

2.2.3 Discrete Time Markov Chains

In the discrete time case, passing from one state to another is taken to be one time step.

Definition 2.14 *A discrete time Markov chain (DTMC) [115] is a discrete time stochastic process* $X = \{X_n \mid n \in \mathbb{N}\}$ *with discrete state space* S_X *such that, for each* $n \in \mathbb{N}_+$, $i_0, \ldots, i_{n-1}, i_n \in S_X$, *it turns out*

$$\Pr\{X_n = i_n \mid X_{n-1} = i_{n-1} \wedge \ldots \wedge X_0 = i_0\} = \Pr\{X_n = i_n \mid X_{n-1} = i_{n-1}\}$$ ■

Definition 2.15 *Let* X *be a DTMC.*

- *The transition matrix of* X *at step* $n \in \mathbb{N}$ *is matrix* $\mathbf{P}_X(n)$ *defined by*

$$\mathbf{P}_X(n) = [\Pr\{X_{n+1} = j \mid X_n = i\}]_{i,j \in S_X}$$

- X *is a homogeneous DTMC (HDTMC) if and only if its transition matrix is independent of the time.*

- *The state probability distribution function of* X *at step* $n \in \mathbb{N}$ *is vector* $\boldsymbol{\pi}_X(n)$ *defined by*

$$\boldsymbol{\pi}_X(n) = [\Pr\{X_n = i\}]_{i \in S_X}$$

- *The stationary probability distribution function of* X *is vector* $\boldsymbol{\pi}_X$ *defined by*

$$\boldsymbol{\pi}_X = \lim_{n \to \infty} \boldsymbol{\pi}_X(n)$$ ■

In the discrete time case, the geometric distribution ($P(n) = p^n \cdot (1-p)$ for all $n \in \mathbb{N}$, with $p \in \mathbb{R}_{]0,1[}$ called the parameter) is the only distribution enjoying the memoryless property. Therefore, given a HDTMC X, it can be proved that the sojourn time of state i is geometrically distributed with parameter $p_{i,i}$ and the execution probability of the transition from state i to state j (given that the current state is i) is $p_{i,j}$ itself. Moreover, if the state space is finite, it can be shown that the state probability distribution function of X at step $n \in \mathbb{N}_+$ is given by

$$\boldsymbol{\pi}_X(n) = \boldsymbol{\pi}_X(0) \cdot \mathbf{P}_X^n$$

while the stationary probability distribution function of X can be determined (if the transitions of X result in a strongly connected graph) by solving the global balance equation

$$\begin{aligned} \boldsymbol{\pi}_X \cdot \mathbf{P}_X &= \boldsymbol{\pi}_X \\ \sum_{i \in S_X} \pi_i &= 1 \end{aligned}$$

Suitable numerical techniques are available for both equation systems [180].

The HDTMC X can equivalently be represented by means of the p-LTS

$$(S_X, \mathbb{R}_{]0,1]}, \{(i, p_{i,j}, j) \in S_X \times \mathbb{R}_{]0,1]} \times S_X\}, \boldsymbol{\pi}_X(0), H_X)$$

where

$$H_X(i) = \begin{cases} v & \text{if } \exists j \in S_X . p_{i,j} > 0 \\ a & \text{otherwise} \end{cases}$$

2.2.4 Ordinary Lumping

The ordinary lumping [172] is an aggregation method that allows an exact analysis of a MC to be carried out on a smaller stochastic process which still is a MC. Exact analysis refers to the fact that, whenever the stationary probability distribution function of the original MC exists, the stationary probability of each macrostate of the lumped MC is the sum of the stationary probabilities of the original states it contains. Though quite helpful, this aggregation should be avoided when it may cause information loss, e.g. as a consequence of merging together states having different meanings w.r.t. a given performance measure.

Definition 2.16 *Let $(S_k, \mathbb{R}_+, \longrightarrow_k, P_k, H_k)$ be the p-LTS representing MC X_k, $k \in \{1,2\}$. X_2 is an ordinary lumping of X_1 if and only if S_2 is a partition of S_1 such that:*

- *for all $C \in S_2$*

$$\sum_{s \in C} P_1(s) = P_2(C)$$

- *for all* $C \in S_2$ *and* $s \in C$

$$H_1(s) = H_2(C)$$

- *for all* $C, C' \in S_2$ *and* $s, s' \in C$

$$\sum \{\!|\, \lambda \in \mathbb{R}_+ \mid \exists s'' \in C'.\, s \xrightarrow{\lambda}_1 s'' \,|\!\} = \sum \{\!|\, \lambda \in \mathbb{R}_+ \mid \exists s'' \in C'.\, s' \xrightarrow{\lambda}_1 s'' \,|\!\}$$

- *for all* $C, C' \in S_2$

$$C \xrightarrow{\lambda}_2 C' \iff \lambda = \sum \{\!|\, \mu \in \mathbb{R}_+ \mid \exists s \in C.\, \exists s' \in C'.\, s \xrightarrow{\mu}_1 s' \,|\!\}$$ ■

It is easily seen that, if a MC X' is an ordinary lumping of a MC X, then the p-LTSs underlying X and X' are p-bisimilar via the relation that associates each state of X with the state of X' that contains it.

2.2.5 Semi Markov Chains

A semi Markov chain [99] is a generalization of a DTMC where the times between transitions are allowed to be random variables which depend on the current, and possibly the next, state. A CTMC can be regarded as a special case of such a stochastic process.

Definition 2.17 *A semi Markov chain (SMC) is a stochastic process* $Z = \{Z(t) \mid t \in \mathbb{R}_{[0,\infty[}\}$ *with discrete state space* S_Z *defined by*

$$Z(t) = Y_{N(t)}$$

where, given a matrix of transition distributions $\mathbf{Q}_Z(t) = [q_{i,j}(t)]_{i,j \in S_Z}$ *defined over* $\mathbb{R}_{[0,\infty[}$ *and a vector of initial state probabilities* $\mathbf{a}_Z = [a_k]_{k \in S_Z}$*, we have that:*

- $(X, Y) = \{(X_n, Y_n) \mid n \in \mathbb{N}\}$ *is a two dimensional stochastic process such that:*
  - $\Pr\{Y_0 = k\} = a_k$ *for all* $k \in S_Z$*;*
  - $\Pr\{Y_{n+1} = j \wedge X_n \leq t \mid Y_n = i \wedge X_{n-1} = t_{n-1} \wedge \ldots \wedge X_0 = t_0 \wedge Y_0 = i_0\} = \Pr\{Y_{n+1} = j \wedge X_n \leq t \mid Y_n = i\} = q_{i,j}(t)$ *for all* $i, j \in S_Z$ *and* $t \in \mathbb{R}_{[0,\infty[}$*;*

  here Y_n *is the state of* Z *at the epoch of its* n*-th transition while* X_n *is the elapsed time between the* n*-th and* $(n+1)$*-th transitions;*
- $N = \{N(t) \mid t \in \mathbb{R}_{[0,\infty[}\}$ *is a stochastic process defined by*

$$N(t) = \sup\{n \in \mathbb{N} \mid S_n \leq t\}$$

where $S = \{S_n \mid n \in \mathbb{N}\}$ *is a stochastic process such that:*

– $S_0 = 0$;

– $S_n = \sum_{k=0}^{n-1} X_k$ *for all* $n \in \mathbb{N}_+$;

here S_n *is the epoch of the* n*-th transition while* $N(t)$ *is the number of transitions that occur by time* t*.*

Y *is called the embedded MC of* Z *and its transition matrix is given by* $\mathbf{P}_Y = [q_{i,j}(\infty)]_{i,j \in S_Z}$*.* ■

A SMC thus evolves as follows. Transitions occur from state to state according to a DTMC. When the process is in state i, the next state visited is j with probability $p_{i,j}$. When the successive states are i and j, the distribution function of the time to make the transition is $F_{i,j}(t) = q_{i,j}(t)/p_{i,j} = \Pr\{X_n \leq t \mid Y_n = i \wedge Y_{n+1} = j\}$. We denote by $\mathbf{h}(t) = [h_i(t)]_{i \in S_Z}$ the vector of the holding time distribution functions, where $h_i(t) = \sum_{j \in S_Z} q_{i,j}(t) = \Pr\{X_n \leq t \mid Y_n = i\}$.

Definition 2.18 *Let* Z *be a SMC.*

- *The transition matrix of* Z *from time* $t \in \mathbb{R}_{[0,\infty[}$ *to time* $t' > t$ *is matrix* $\mathbf{P}_Z(t,t')$ *defined by*

$$\mathbf{P}_Z(t,t') = [\Pr\{Z(t') = j \mid Z(t) = i\}]_{i,j \in S_Z}$$

- Z *is a homogeneous SMC (HSMC) if and only if its transition matrix is independent of the time.*

- *The state probability distribution function of* Z *at time* $t \in \mathbb{R}_{[0,\infty[}$ *is vector* $\boldsymbol{\pi}_Z(t)$ *defined by*

$$\boldsymbol{\pi}_Z(t) = [\Pr\{Z(t) = i\}]_{i \in S_Z}$$

- *The stationary probability distribution function of* Z *is vector* $\boldsymbol{\pi}_Z$ *defined by*

$$\boldsymbol{\pi}_Z = \lim_{t \to \infty} \boldsymbol{\pi}_Z(t)$$

■

Whenever the stationary probability distribution function of a HSMC Z exists, it can be determined by first solving

$$\begin{aligned} \boldsymbol{\pi}_Y \cdot \mathbf{P}_Y &= \boldsymbol{\pi}_Y \\ \sum_{i \in S_Z} \pi_{Y,i} &= 1 \end{aligned}$$

and then letting

$$\pi_{Z,i} = \frac{\pi_{Y,i} \cdot \bar{h}_i}{\sum_{k \in S_Z} \pi_{Y,k} \cdot \bar{h}_k}$$

where $\bar{h}_i$ is the average sojourn time in state i.

In this thesis we consider HSMCs where the times between transitions are allowed to be only exponentially distributed or zero and depend only on the current state. Moreover, all the outgoing transitions of a given state must be of the same type. A HSMC Z of this kind can be represented by means of the p-LTS

$$(S_Z, \mathbb{R}_+, \{(i, r_{i,j}, j) \in S_Z \times \mathbb{R}_+ \times S_Z\}, \boldsymbol{\pi}_Z(0), H_Z)$$

where $r_{i,j}$ is the rate or the probability of the transition from state i to state j depending on whether it is exponentially timed or immediate, while

$$H_Z(i) = \begin{cases} v & \text{if } \bar{h}_i = 0 \\ t & \text{if } \bar{h}_i \in \mathbb{R}_+ \\ a & \text{if } \bar{h}_i = \infty \end{cases}$$

2.2.6 Reward Structures

Reward structures permit to specify and derive measures for system models represented by SMCs. They will be used in Chap. 7 to describe performance indices within EMPA terms.

Definition 2.19 *A reward structure [108] for a SMC is composed of:*

- *A yield function $y_{i,j}(t)$ expressing the rate at which reward is accumulated at state i t time units after i was entered when the successor state is j.*
- *A bonus function $b_{i,j}(t)$ expressing the reward awarded upon exit from state i and subsequent entry into state j given that the holding time in state i was t time units.* ■

Since the generality of this structure is difficult to fully exploit due to the complexity of the resulting solution, the analysis is usually simplified by considering yield functions that do not depend on the time nor the successor state, as well as bonus functions that do not depend on the holding time of the previously occupied state: $y_{i,j}(t) = y_i$ and $b_{i,j}(t) = b_{i,j}$.

Several performance measures can be calculated by exploiting rewards. According to the classifications proposed in [171, 79], we have *instant-of-time measures*, expressing the gain received at a particular time instant, and *interval-of-time (or cumulative) measures*, expressing the overall gain received over some time interval. Both kinds of measures can refer to stationary or transient state. In the following, we shall concentrate on instant-of-time performance measures.

In the stationary case, instant-of-time performance measures quantify the long run gain received per unit of time. Given yield rewards y_i and bonus rewards $b_{i,j}$ for a certain SMC, the corresponding stationary performance measure is computed as:

$$\sum_i y_i \cdot \pi_i + \sum_i \sum_j b_{i,j} \cdot \phi_{i,j} \tag{2.1}$$

where π_i is the stationary probability of state i and $\phi_{i,j}$ is the stationary frequency with which the transition from state i to state j is traversed.

- In the case of a homogeneous semi Markov reward chain (HSMRC) with transition matrix of the embedded DTMC $\mathbf{P}$ (whose stationary probabilities are denoted π') and with average holding times $\mathbf{h}$ restricted to be either exponentially distributed or constant zero, the stationary frequency $\phi_{i,j}$ is given by the stationary frequency with which state i is entered (i.e. the ratio of its stationary probability to its average holding time) multiplied by the probability with which the transition from state i to state j is traversed given that the current state is i:

$$\phi_{i,j} = \frac{\pi'_i}{\sum_k \pi'_k \cdot h_k} \cdot p_{i,j}$$

- In the case of a homogeneous continuous time Markov reward chain (HCTMRC) with infinitesimal generator $\mathbf{Q}$, the stationary frequency $\phi_{i,j}$ is given by the stationary probability of state i multiplied by the rate of the transition from state i to state j (as the probability of traversing such a transition given that the current state is i is the ratio of its rate to the reciprocal of the average holding time of i):

$$\phi_{i,j} = \pi_i \cdot q_{i,j}$$

- In the case of a homogeneous discrete time Markov reward chain (HDTMRC) with transition matrix $\mathbf{P}$, the stationary frequency $\phi_{i,j}$ is given by the stationary probability of state i (as a transition from state i is executed one time unit after entering i) multiplied by the probability with which the transition from state i to state j is traversed given that the current state is i:

$$\phi_{i,j} = \pi_i \cdot p_{i,j}$$

In the transient state case, instant-of-time performance measures quantify the gain received at a given time instant. Given yield rewards y_i and bonus rewards $b_{i,j}$ for a certain SMC, the corresponding transient state performance measure is computed as:

$$\sum_i y_i \cdot \pi_i(t) + \sum_i \sum_j b_{i,j} \cdot \phi_{i,j}(t) \tag{2.2}$$

where $\pi_i(t)$ is the probability of being in state i at time t and $\phi_{i,j}(t)$ is the the transient frequency with which the transition from state i to state j is traversed at time t, which is computed in the same way as $\phi_{i,j}$ with $\pi_i(t)$ in place of π_i.

As a graph theoretic representation of HSMRCs, HCTMRCs, and HDTMRCs, we propose an extension of p-LTSs where bonus rewards are added to transition labels and a yield reward function over states is introduced. These mathematical models will be used in Sect. 7.2 to define the performance semantics for EMPA extended with rewards.

Definition 2.20 *A probabilistically rooted labeled transition system with rewards (pr-LTS) is a tuple*

$$(S, \mathbb{R}_+ \times \mathbb{R}, \longrightarrow, P, H, Y)$$

where:

- S, $\longrightarrow$, P, *and* H *are defined as for a p-LTS.*
- $Y : S \longrightarrow \mathbb{R}$ *is called yield reward function.* ■

Definition 2.21 *Let* $Z_k = (S_k, \mathbb{R}_+ \times \mathbb{R}, \longrightarrow_k, P_k, H_k, Y_k)$, $k \in \{1, 2\}$, *be two pr-LTSs.*

- Z_1 *is pr-isomorphic to* Z_2 *if and only if there exists a bijection* $\beta : S_1 \longrightarrow S_2$ *such that:*
  - *for all* $s \in S_1$

$$\begin{aligned} P_1(s) &= P_2(\beta(s)) \\ H_1(s) &= H_2(\beta(s)) \\ Y_1(s) &= Y_2(\beta(s)) \end{aligned}$$

  - *for all* $s, s' \in S_1$, $\lambda \in \mathbb{R}_+$, *and* $r \in \mathbb{R}$

$$s \xrightarrow{\lambda,r}_1 s' \iff \beta(s) \xrightarrow{\lambda,r}_2 \beta(s')$$

- Z_1 *is pr-bisimilar to* Z_2 *if and only if there exists a relation* $\mathcal{B} \subseteq S_1 \times S_2$ *with reflexive, symmetric and transitive closure* $\mathcal{B}' \subseteq (S_1 \cup S_2) \times (S_1 \cup S_2)$ *such that:*
  - *for all* $C \in (S_1 \cup S_2)/\mathcal{B}'$

$$\sum_{s \in C \cap S_1} P_1(s) = \sum_{s \in C \cap S_2} P_2(s)$$

  - *whenever* $(s_1, s_2) \in \mathcal{B}$, *then*

$$\begin{aligned} H_1(s_1) &= H_2(s_2) \\ Y_1(s_1) &= Y_2(s_2) \end{aligned}$$

  - *whenever* $(s_1, s_2) \in \mathcal{B}$, *then for all* $r \in \mathbb{R}$ *and* $C \in (S_1 \cup S_2)/\mathcal{B}'$

$$\sum \{\!|\, \lambda \in \mathbb{R}_+ \mid \exists s_1' \in C \cap S_1.\, s_1 \xrightarrow{\lambda,r}_1 s_1' \,|\!\} = \sum \{\!|\, \lambda \in \mathbb{R}_+ \mid \exists s_2' \in C \cap S_2.\, s_2 \xrightarrow{\lambda,r}_2 s_2' \,|\!\}$$ ■

The relationship between ordinary lumping with rewards [142] and pr-bisimilarity is the same as the relationship between ordinary lumping and p-bisimilarity reported at the end of Sect. 2.2.4.

2.2.7 Queueing Systems

A *queueing system (QS)* [115] is an abstract model largely used for evaluating the performance of computer and communication systems through the computation of measures such as system throughput, resource utilization, and user response time. A QS is a service center, composed of a waiting queue and a given number of servers, which provides a certain service to a population of customers according to a given discipline. As an example, we can think of a computer system as a QS, where the central unit can be viewed as the server and the various devices can be viewed as the customers.

In this thesis, we shall be mainly concerned with QSs $M/M/n/q/m$ with arrival rate λ and service rate μ, which are defined as follows (see Fig. 2.1):

1. The customer arrival process is Markovian with rate λ.
2. The customer service process is Markovian with rate μ.
3. There are n independent servers.
4. There is a FIFO queue with $q - n$ seats. When missing, parameter q denotes an unbounded queue.
5. There are m independent customers. When missing, parameter m denotes an unbounded population of customers.

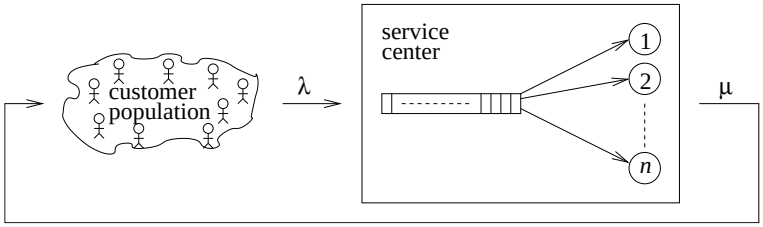


Figure 2.1: Structure of a QS

As an example, consider a QS $M/M/1$ with arrival rate λ and service rate μ. The stochastic process underlying this QS is a HCTMC. If state s_i represents the situation in which i customers are in the system, then state s_0 has a transition toward s_1 labeled with λ while state s_i, $i \in \mathbb{N}_+$, has a transition toward s_{i+1} labeled with λ and a transition toward s_{i-1} labeled with μ. It can be shown that this HCTMC admits stationary probability distribution function if and only if $\lambda < \mu$, in which case for all $i \in \mathbb{N}$

$$\pi_i = (1 - \lambda/\mu)(\lambda/\mu)^i$$

From this distribution we can compute the mean number of customers in the system as

$$\sum_{i=0}^{\infty} i \cdot \pi_i = \frac{(\lambda/\mu)}{(1 - \lambda/\mu)}$$

while the average waiting time in the system is derived by Little's law as the value above divided by λ

$$\frac{(1/\mu)}{(1-\lambda/\mu)}$$

Moreover, the utilization of the server is given by

$$\sum_{i=1}^{\infty} \pi_i = 1 - \pi_0 = \lambda/\mu$$

The presence of this background section is justified by the fact that QSs will be used as running examples throughout the thesis. More precisely, several QSs of the family M/M will be considered in Sect. 4.7 to emphasize the expressiveness of EMPA; two different EMPA descriptions of the same QS will be compared in Sect. 5.3 using the integrated equivalence; two different integrated net semantics of the same QS obtained by applying two different approaches will be presented in Sect. 6.3 in order to compare the two approaches; the EMPA descriptions of two QSs of the family M/M will be examined in Sect. 7.1 and 7.3 to illustrate the extension of EMPA with rewards; and a value passing EMPA description of a QS with generally distributed service time will be provided in Sect. 8.3.

2.3 Petri Nets

In this section we recall the noninterleaving mathematical model given by Petri nets which will be used in Chap. 6 to define the integrated net semantics for EMPA. In Sect. 2.3.1 we present classical Petri nets, while in Sect. 2.3.2 we introduce generalized stochastic Petri nets. Then in Sect. 2.3.3 we propose an extension of generalized stochastic Petri nets that will be used in Chap. 6: passive generalized stochastic Petri nets.

2.3.1 Classical Petri Nets

Unlike process algebras, which are algebraic languages supporting compositional modeling, Petri nets [157] are mathematical models which graphically describe concurrent systems. Unlike LTSs and their variants such as p-LTSs and pr-LTSs, which are interleaving models, Petri nets are truly concurrent models which explicitly highlight dependencies, conflicts, and synchronizations among system activities. The purpose of this section is to recall from [164, 139] some basic notions and analysis techniques concerning classical Petri nets, i.e. Petri nets abstracting from priority, probability, and timing aspects.

Since the formalism of multisets is required to introduce Petri nets, let us preliminarily define the following notation. A *multiset* M over a set S is a function $M : S \longrightarrow \mathbb{N}$, where $M(s)$ is said to be the multiplicity of element s and $dom(M) = \{s \in S \mid M(s) > 0\}$. We use "$\{\!|$" and "$|\!\}$" as brackets for multisets and we let $\mathcal{M}u_{\rm fin}(S)$ ($\mathcal{P}_{\rm fin}(S)$) be the collection of finite multisets over (subsets of) set S. We denote by "$\_ = \_$", "$\_ \subseteq \_$", "$\_ \oplus \_$", and "$\_ \ominus \_$" the multiset equality, inclusion, union, and difference, respectively. Finally, we denote by $\pi_i(M)$ the multiset obtained by projecting the tuples in multiset M on their i-th component.

A Petri net is a bipartite graph whose classes of nodes (places and transitions) basically represent system components and system activities, respectively. The state of the system is described by marking places with tokens and the computation can be described as the flow of tokens through the net caused by transition firings. Unlike LTSs and their variants, the state of a Petri net is thus given a representation distributed among places.

Definition 2.22 *A Petri net is a tuple*

$$(P, U, T, M_0)$$

where:

- P *is a set whose elements are called places.*
- U *is a set whose elements are called labels.*
- $T \subseteq \mathcal{M}u_{\text{fin}}(P) \times U \times \mathcal{M}u_{\text{fin}}(P)$ *whose elements are called transitions.*
- $M_0 \in \mathcal{M}u_{\text{fin}}(P)$ *is called the initial marking.*

We call marking every element of $\mathcal{M}u_{\text{fin}}(P)$. ■

In the graphical representation of a Petri net, places are drawn as circles while transitions are drawn as boxes with the appropriate labels. If the current marking of the net is M, we draw $M(p)$ black dots called *tokens* in place p. Each transition t can be written as ${}^\bullet t \xrightarrow{u_t} t^\bullet$ where ${}^\bullet t$ is the weighted *preset* of t (places where tokens are consumed), u_t is the *label* of t, and $t^\bullet$ is the weighted *postset* of t (places where tokens are produced). Given a transition t, we draw an arrow headed arc from each place in ${}^\bullet t$ to t as well as from t to each place in $t^\bullet$, where each arc is labeled with the multiplicity of the related place (one is the default value for arc labels).

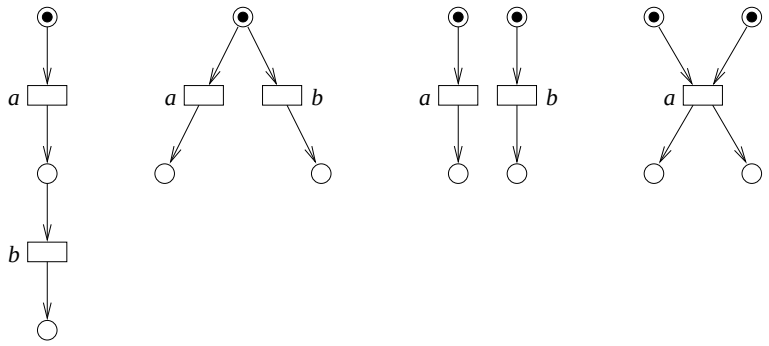
Definition 2.23 *Let* $N = (P, U, T, M_0)$ *be a Petri net.*

- *Transition* $t \in T$ *is enabled at marking* $M \in \mathcal{M}u_{\text{fin}}(P)$ *if and only if* ${}^\bullet t \subseteq M$. *We denote by* $ET(M)$ *the set of transitions enabled at marking* M.
- *The firing of transition* $t \in ET(M)$, *where* $M \in \mathcal{M}u_{\text{fin}}(P)$, *produces marking* $M' = (M \ominus {}^\bullet t) \oplus t^\bullet$. *This is written* $M\,[u_t\rangle\,M'$.
- *The reachability set* $RS(M)$ *of marking* $M \in \mathcal{M}u_{\text{fin}}(P)$ *is the least subset of* $\mathcal{M}u_{\text{fin}}(P)$ *such that:*
  - $M \in RS(M)$;
  - *if* $M_1 \in RS(M)$ *and* $M_1\,[u_t\rangle\,M_2$, *then* $M_2 \in RS(M)$.

- *The reachability graph (or interleaving marking graph) of N is the LTS*

$$\mathcal{RG}[\![N]\!] = (RS(M_0), U, [\rangle, M_0)$$ ■

As an example, given two activities a and b, we depict below four Petri nets representing a causal dependency between a and b, a conflict between a and b, a full independence between a and b, and a synchronization on a, respectively:



Definition 2.24 *Let $N_k = (P_k, U, T_k, M_{0k})$, $k \in \{1, 2\}$, be two Petri nets. N_1 is isomorphic to N_2 if and only if there exist two bijections $\beta_P : P_1 \longrightarrow P_2$ and $\beta_T : T_1 \longrightarrow T_2$ such that:*

- $\beta_P(M_{01}) = M_{02}$;
- *for all $t \in T_1$*

$$^\bullet\beta_T(t) = \beta_P(^\bullet t) \wedge u_{\beta_T(t)} = u_t \wedge \beta_T(t)^\bullet = \beta_P(t^\bullet)$$ ■

A major strength of Petri nets is their support for analysis of many properties and problems associated with concurrent systems. Two types of properties can be studied: those which depend on the initial marking (referred to as *marking dependent or behavioral*) and those which are independent of the initial marking (referred to as *structural*). Among such properties we cite:

- Reachability: a marking M is reachable from M_0 if there exists a sequence of transition firings transforming M_0 into M.
- Boundedness: a Petri net is bounded if the number of tokens in each place does not exceed a finite number for any reachable marking. As an example, if some places of a Petri net model of a given system are used to represent buffers, by verifying that the net is bounded it is guaranteed that overflow will not occur.
- Liveness: a Petri net is live if from every reachable marking it is possible to ultimately fire any transition of the net by progressing through some further firing sequence. The concept of liveness is closely related to the complete absence of deadlocks in concurrent systems.

- Fairness: a Petri net is fair if, for every firing sequence from a reachable marking, either it is finite or every transition in the net appears infinitely often in it.

Analysis methods for Petri nets may be classified into three groups. The *reachability tree method* involves essentially the enumeration of all reachable markings, so it can in principle be applied to any Petri net but it is limited to small Petri nets because of the interleaving state space explosion. On the other hand, *matrix equations* and *reduction techniques* are powerful but in many cases applicable only to special Petri nets. As an example of matrix equation based method, we briefly recall the technique of net invariants.

Definition 2.25 *Let $N = (P, U, T, M_0)$ be a Petri net such that $|P| = m$ and $|T| = n$.*

- *The incidence matrix of N is matrix $\mathbf{A} = [a_{i,j}] \in \mathbb{Z}^{n \times m}$ where*

$$a_{i,j} = i^{\bullet}(j) - {}^{\bullet}i(j)$$

- *An P-invariant is an integer solution of $\mathbf{A} \cdot \mathbf{y} = \mathbf{0}$.*
- *A T-invariant is an integer solution of $\mathbf{A}^{\mathrm{T}} \cdot \mathbf{x} = \mathbf{0}$.* ■

P-invariants single out places that do not change their token count during transition firings whereas T-invariants indicate how often each transition has to fire in order to reproduce a given marking. By means of such invariants it is possible to analyze the properties listed above without generating the reachability graph of the net. As an example, it can be shown that a Petri net is bounded if for each of its places there exists an P-invariant associating a positive value with that place, while a necessary condition in order for a Petri net to be live and bounded is that for each of its transitions there exists a T-invariant associating a positive value with that transition.

2.3.2 Generalized Stochastic Petri Nets

Generalized stochastic Petri nets [3] are an extension of classical Petri nets which is suited for performance evaluation purposes. They comprise inhibitor arcs as well as exponentially timed transitions with marking dependent rates and immediate transitions equipped with priorities and marking dependent weights. Rates of exponentially timed transitions are positive real numbers uniquely identifying the corresponding exponentially distributed durations, while priorities and weights are numbers used to represent prioritized and probabilistic choices, respectively. Note that in the following definition labels are marking dependent.

Definition 2.26 *A generalized stochastic Petri net (GSPN) is a tuple*

$$(P, U, T, M_0, L, W)$$

where:

- P *and* M_0 *are defined as for a Petri net.*
- $U = \hat{U}^{\mathcal{M}u_{\text{fin}}(P)}$.
- $T \subseteq \mathcal{M}u_{\text{fin}}(P) \times \mathcal{P}_{\text{fin}}(P) \times U \times \mathcal{M}u_{\text{fin}}(P)$.
- $L : T \longrightarrow \mathbb{N}$, *called priority function, is such that:*
  - $L(t) = 0$ *if* t *is exponentially timed;*
  - $L(t) \in \mathbb{N}_+$ *is the priority level of* t *if* t *is immediate.*
- $W : T \longrightarrow \mathbb{R}_+^{\mathcal{M}u_{\text{fin}}(P)}$, *called weight function, is such that:*
  - $W(t) \in \mathbb{R}_+^{\mathcal{M}u_{\text{fin}}(P)}$ *is the rate of the exponential distribution associated with* t *if* $L(t) = 0$;
  - $W(t) \in \mathbb{R}_+^{\mathcal{M}u_{\text{fin}}(P)}$ *is the weight of* t *if* $L(t) \in \mathbb{N}_+$. ■

In the graphical representation of a GSPN, transitions are drawn as either boxes (if exponentially timed) or bars (if immediate) with the appropriate labels. Each transition t can be written as a function of the current marking M_{curr} which returns $({}^\bullet t, {}^\circ t) \stackrel{u_t(M_{\text{curr}})}{\longrightarrow} t^\bullet$ where ${}^\circ t$ is the *inhibitor set* of t (places where tokens must be absent). Given a transition t, we draw a circle headed arc from each place in ${}^\circ t$ to t.

Definition 2.27 *Let* $N = (P, U, T, M_0, L, W)$ *be a GSPN.*

- *Transition* $t \in T$ *is enabled at marking* $M \in \mathcal{M}u_{\text{fin}}(P)$ *if and only if* ${}^\bullet t \subseteq M$ *and* $dom(M) \cap {}^\circ t = \emptyset$.
- *Transition* $t \in ET(M)$, *where* $M \in \mathcal{M}u_{\text{fin}}(P)$, *can fire if and only if* $L(t) = \max\{L(t') \mid t' \in ET(M)\}$.
- *The reachability graph (or interleaving marking graph) of* N *is the LTS*

$$\mathcal{RG}[\![N]\!] = (RS(M_0), \hat{U}, [\rangle, M_0)$$ ■

From $\mathcal{RG}[\![N]\!]$ we can extract the functional semantics of the net in the form of a LTS $\mathcal{F}[\![N]\!]$, by considering only functional information contained in the labels, and the performance semantics of the net in the form of a Markov chain $\mathcal{M}[\![N]\!]$, which is embedded in a more general stochastic process where both states enabling exponentially timed transitions and states enabling immediate transitions coexist.

It is worth recalling that for GSPNs there are structural analysis techniques not only for verifying functional properties but also for deriving performance measures. More precisely, it is possible to compute the maximum and the average numbers of tokens in the places (which provide a measure of the level of parallelism of the modeled system) as well as upper bounds on the throughput of transitions without generating the underlying state space. This is accomplished at the net level by solving suitable linear programming problems [45].

2.3.3 Passive Generalized Stochastic Petri Nets

We propose below an extension of GSPNs where passive (i.e., untimed) transitions as well as contextual arcs [137] are admitted. These nets will be used in Chap. 6 to define the integrated net semantics for EMPA.

Definition 2.28 *A passive generalized stochastic Petri net (PGSPN) is a tuple*

$$(P, U, T, M_0, L, W)$$

where:

- P, U, *and* M_0 *are defined as for a GSPN.*
- $T \subseteq \mathcal{M}u_{\rm fin}(P) \times \mathcal{P}_{\rm fin}(P) \times \mathcal{P}_{\rm fin}(P) \times U \times \mathcal{M}u_{\rm fin}(P)$.
- $L : T \longrightarrow \{-1\} \cup \mathbb{N}$ *is such that:*
  - $L(t) = -1$ *if* t *is passive;*
  - $L(t) = 0$ *if* t *is exponentially timed;*
  - $L(t) \in \mathbb{N}_+$ *is the priority level of* t *if* t *is immediate.*
- $W : T \longrightarrow \{*\} \cup \mathbb{R}_+^{\mathcal{M}u_{\rm fin}(P)}$ *is such that:*
  - $W(t) = *$ *if* $L(t) = -1$*;*
  - $W(t) \in \mathbb{R}_+^{\mathcal{M}u_{\rm fin}(P)}$ *is the rate of the exponential distribution associated with* t *if* $L(t) = 0$*;*
  - $W(t) \in \mathbb{R}_+^{\mathcal{M}u_{\rm fin}(P)}$ *is the weight of* t *if* $L(t) \in \mathbb{N}_+$. ■

The graphical representation of a PGSPN is similar to that of a GSPN with in addition the convention that passive transitions are drawn as black boxes. Each transition t can be written as a function of the current marking $M_{\rm curr}$ which returns $({}^\bullet t, {}^\circ t, \hat{t}) \stackrel{u_t(M_{\rm curr})}{\longrightarrow} t^\bullet$ where $\hat{t}$ is the *contextual set* of t (places where tokens must be present). Given a transition t, we draw an arc from each place in $\hat{t}$ to t.

Definition 2.29 *Let* $N = (P, U, T, M_0, L, W)$ *be a PGSPN.*

- *Transition* $t \in T$ *is enabled at marking* $M \in \mathcal{M}u_{\rm fin}(P)$ *if and only if* ${}^\bullet t \subseteq M$, $dom(M) \cap {}^\circ t = \emptyset$, *and* $\hat{t} \subseteq dom(M)$.
- *Transition* $t \in ET(M)$, *where* $M \in \mathcal{M}u_{\rm fin}(P)$, *can fire if and only if* $L(t) = -1$ *or* $L(t) = \max\{L(t') \mid t' \in ET(M)\}$. ■

The performance semantics can be extracted only for those PGSPNs whose possible passive transitions cannot fire.

Chapter 3

Syntax and Interleaving Semantics for EMPA

The experience of the past twenty years with process algebras has shown that several expressive features are necessary to be able to model real world systems. Here we focus on such features as time, priority, and probability to give the intuition about why and how they are added to a classical process algebra like the one in Sect. 2.1 to obtain EMPA.

First of all, since we are interested in evaluating the performance of shared resource systems, starting from a classical process algebra we need to add the capability of expressing the duration of system activities by means of random variables. If in particular we restrict the random variables to be exponentially distributed, then a HCTMC can be hopefully associated with every process algebra description of a system, thus making it possible to apply well known techniques to compute its performance measures. This can be achieved by viewing every process algebra action as being a pair $<a, \lambda>$ composed of the type a of the action (e.g., message transmission) and the rate $\lambda \in \mathbb{R}_+$ of the exponentially distributed random variable characterizing the duration of the action.

Second, we need a mechanism to abstract from those system activities representing just logical events or whose duration is some orders of magnitude smaller than the duration of the activities which are relevant from the performance viewpoint. Such a mechanism can be realized by introducing actions with duration zero, hence with rate ∞.

Third, it is desirable to be able to express the fact that some system activities take precedence over other system activities. In other words, we need to add the possibility of expressing prioritized choices to describe the system control in a more adequate and faithful way. This can be obtained by attaching to every action with duration zero a natural number l determining its priority level, hence the rate ∞_l.

Finally, it is worth being able to represent the fact that some competing system activities can be executed with different frequencies. This means that we need to introduce the capability of modeling probabilistic choices to better describe the system control. This can be implemented by assigning to every action with duration zero a positive real number w determining its weight, hence the rate $\infty_{l,w}$. This further capability

allows us to represent shared resource systems whose behavior is governed by a HDTMC, if we view the execution of every action with rate $\infty_{l,w}$ as taking one time unit.

In this chapter we introduce EMPA by showing the syntax of its terms and by formalizing the meaning of its operators [27]. This chapter is organized as follows. In Sect. 3.1 we introduce the concept of integrated action together with two classifications of integrated actions based on their types and durations, respectively. In Sect. 3.2 we define the syntax of terms and we informally explain the meaning of each operator. In Sect. 3.3 we illustrate the execution policy we adopt to choose among several simultaneously executable actions. Finally, in Sect. 3.4 we present the integrated interleaving semantics together with its functional and performance projections.

3.1 Integrated Actions: Types and Rates

As anticipated in the previous section, every *action* $<a, \tilde{\lambda}>$ of EMPA is naturally characterized through both its *type* a and its *duration* $\tilde{\lambda}$. Its second component, called *rate*, indicates the speed at which the action occurs and is used as a concise way to denote the random variable specifying the duration of the action. Like in classical process algebras, types divide actions into *observable or external* and *unobservable or internal*. As usual, we denote by τ the only internal action type we use. Moreover, rates divide actions into active and passive as follows:

- *Active actions* are actions whose rate is specified. An active action can be either exponentially timed or immediate:
  - *Exponentially timed actions* are actions whose rate is a positive real number. Such a number is interpreted as the parameter of the exponentially distributed random variable specifying the duration of the action. We recall that an exponentially distributed random variable X_λ with parameter $\lambda \in \mathbb{R}_+$ has probability distribution function $F_{X_\lambda}(t) \equiv \Pr\{X_\lambda \leq t\} = 1 - e^{-\lambda \cdot t}$ for all $t \in \mathbb{R}_+$, expected value $1/\lambda$, and variance $1/\lambda^2$, thus it is uniquely identified by its parameter.
  - *Immediate actions* are actions whose rate, denoted by $\infty_{l,w}$, is infinite. Such actions have duration zero and each of them is given a *priority level* $l \in \mathbb{N}_+$ and a *weight* $w \in \mathbb{R}_+$.
- *Passive actions* are actions whose rate, denoted by $*$, is undefined. The duration of a passive action is fixed only by synchronizing it with an active action of the same type.

Using exponentially timed actions only results in continuous time models, while using immediate actions only results in discrete time models. When both of them are used, exponentially timed actions model activities that are relevant from the performance point of view, while immediate actions model prioritized probabilistic events as well as activities that are either irrelevant from the performance point of view or unboundedly faster than the others. Passive actions model nondeterministic events as well as activities waiting for the

synchronization with timed activities of the same type. While exponentially timed actions of EMPA are exactly the same as exponentially timed actions of MTIPP [92] and PEPA [100], immediate actions and passive actions are different from those adopted in other stochastically timed process algebras. In particular, immediate actions of EMPA, which have the same structure as immediate transitions of GSPNs [3], differ from the immediate actions of [94] since these have neither associated priorities nor weights. Moreover, passive actions of EMPA, which resemble actions of classical process algebras, differ from both the passive actions of [92], since these have an associated duration, and the passive actions of [100], because these have an associated weight. It is worth noting that the coexistence of different kinds of actions provides EMPA with a considerable expressive power, as we shall see in Chap. 4.

We denote the set of actions by $Act = AType \times ARate$ where $AType$ is the set of types including τ and $ARate = \mathbb{R}_+ \cup Inf \cup \{*\}$, with $Inf = \{\infty_{l,w} \mid l \in \mathbb{N}_+ \wedge w \in \mathbb{R}_+\}$, is the set of rates. We use $a, b, \ldots$ as metavariables for $AType$, $\tilde{\lambda}, \tilde{\mu}, \ldots$ for $ARate$, and $\lambda, \mu, \ldots$ for $\mathbb{R}_+$. Finally, we denote by $APLev = \{-1\} \cup \mathbb{N}$ the set of *action priority levels*, where value -1 will be used for passive actions, and we assume that $* < \lambda < \infty_{l,w}$ for all $\lambda \in \mathbb{R}_+$ and $\infty_{l,w} \in Inf$.

3.2 Syntax of Terms and Informal Semantics of Operators

Let $Const$ be a set of *constants* ranged over by $A, B, \ldots$ and let $ATRFun = \{\varphi : AType \longrightarrow AType \mid \varphi^{-1}(\tau) = \{\tau\}\}$ be a set of *action type relabeling functions* ranged over by $\varphi, \varphi', \ldots$.

Definition 3.1 *The set $\mathcal{L}$ of process terms of EMPA is generated by the following syntax*

$$E ::= \underline{0} \mid <a, \tilde{\lambda}>.E \mid E/L \mid E[\varphi] \mid E + E \mid E \parallel_S E \mid A$$

where $L, S \subseteq AType - \{\tau\}$. The set $\mathcal{L}$ will be ranged over by $E, F, \ldots$. ■

In the rest of the section we informally explain the semantics of the operators: the formal semantics will be presented in Sect. 3.4.

The *null term* "$\underline{0}$" is the term that cannot execute any action.

The *action prefix operator* "$<a, \tilde{\lambda}>.\_$" denotes the sequential composition of an action and a term. Term $<a, \tilde{\lambda}>.E$ can execute an action with type a and rate $\tilde{\lambda}$ and then behaves as term E.

The *functional abstraction operator* "$\_/L$" abstracts from the type of the actions. Term E/L behaves as term E except that the type a of each executed action is turned into τ whenever $a \in L$.

The *functional relabeling operator* "$\_[\varphi]$" changes the type of the actions. Term $E[\varphi]$ behaves as term E except that the type a of each executed action becomes $\varphi(a)$.

The *alternative composition operator* "$\_ + \_$" expresses a choice between two terms. Term $E_1 + E_2$ behaves as either term E_1 or term E_2 depending on whether an action of E_1 or an action of E_2 is executed. As we shall see in Sect. 3.3, the way in which the choice is resolved depends on the kind of the actions involved in the choice itself.

The *parallel composition operator* "$\_ \|_S \_$" expresses the concurrent execution of two terms. Term $E_1 \|_S E_2$ asynchronously executes actions of E_1 or E_2 not belonging to S and synchronously executes actions of E_1 and E_2 belonging to S according to the two following synchronization disciplines. The synchronization discipline on action types establishes that two actions can synchronize if and only if they have the same observable type in S, which becomes the resulting type. The synchronization discipline on action rates, instead, establishes that an action with rate $\tilde{\lambda}$ can synchronize with an action with rate $\tilde{\mu}$ if and only if $\min(\tilde{\lambda}, \tilde{\mu}) = *$, and the resulting rate is given by $\max(\tilde{\lambda}, \tilde{\mu})$ up to normalization required by the bounded capacity assumption [101], as explained in Sect. 3.3. Note that, in case of n-way synchronization, at most one action can be active, whereas all the other $n - 1$ actions must be passive. The reasons for the adoption of such a synchronization discipline are its simplicity and its modularity: one single action determines the duration and the probability of the synchronization, while the durations and the probabilities of the other involved actions can be left unspecified. Our synchronization discipline, which is reminiscent of the composition of I/O automata [125], is different from those adopted in MTIPP [92] and PEPA [100] and in Sect. 4.6 it is shown to be not so restrictive as it might seem. We shall also see that a solution to the problem of defining the duration of the synchronization of two exponentially timed actions, naturally interpreted as the maximum of their durations, has been given in MLOTOS [93] by keeping action execution separated from time passing through an action type prefix operator and a time passing prefix operator. This solution nevertheless requires a different action structure which is no longer integrated.

In order to avoid ambiguities, we assume the binary operators to be left associative and we introduce the following operator precedence relation: functional abstraction = functional relabeling > action prefix > alternative composition > parallel composition. Parentheses can be used to alter associativity and precedence.

Finally, let partial function $Def : Const \rightarrow\!\!\!\!\circ\, \mathcal{L}$ be a set of *constant defining equations* of the form $A \triangleq E$. In order to guarantee the correctness of recursive definitions, as usual we restrict ourselves to the set $\mathcal{G}$ of terms that are closed and guarded w.r.t. Def. From now on, Def will not be explicitly mentioned as it will be clear from the context.

3.3 Execution Policy

Because of the presence of binary operators such as the alternative composition and the parallel composition, the situation in which several actions are simultaneously executable can arise. Thus we need a mechanism for choosing the action to be executed among the enabled ones. This mechanism, usually called *execution policy* [3], will be reflected by the semantic model underlying EMPA terms. Such a semantic model is a LTS where states are in correspondence with terms and transitions are labeled with actions: the execution policy determines the sojourn time of every state and the execution probability of every transition.

3.3.1 Sequence

Consider term $<a, \lambda>.\underline{0}$. This term represents a system that can execute an action having type a whose duration is exponentially distributed with rate λ. The underlying LTS has two states that correspond to terms $<a, \lambda>.\underline{0}$ and $\underline{0}$, respectively. At first sight, in this case we should have infinitely many transitions labeled with a from the first state to the second state, i.e. one transition for every possible duration of the action. Fortunately, this infinitely branching structure can be symbolically replaced by means of one single transition labeled with a, λ. The rate of the exponential distribution describing the duration of the action at hand contains all the information we need from the point of view of the performance model underlying the considered term, because this is a HCTMC as we shall see in Sect. 3.4.

3.3.2 Choice

Consider term $<a_1, \lambda_1>.\underline{0} + <a_2, \lambda_2>.\underline{0}$. To choose between the two exponentially timed actions we adopt the *race policy*: the action sampling the least duration succeeds. This means that all the alternative actions can be thought of as being in parallel execution and the completion of one of them interrupts and discards the others. Thus the meaning of the alternative composition operator in the Markovian setting deviates from the original one, as a sequential process is assumed to execute several actions in parallel to choose among them. This relationship between sequentiality and parallelism is justified in the Markovian setting by the property that several exponentially timed actions running in parallel are equivalent, from the point of view of an external observer, to a single exponentially timed action whose rate is the sum of the rates of the original actions [115]. The adoption of the race policy implies that (i) the random variable describing the sojourn time in the state corresponding to the term above is the minimum of the exponentially distributed random variables describing the durations of the two actions, and (ii) the execution probability of the two actions is determined as well by the exponentially distributed random variables describing their durations. For the term above, where two exponentially distributed random variables X_{λ_1} and X_{λ_2} come into play, we have that the sojourn time of the corresponding state is exponentially distributed with rate $\lambda_1 + \lambda_2$ because

$$\begin{aligned}
\Pr\{\min(X_{\lambda_1}, X_{\lambda_2}) \leq t\} &= \Pr\{X_{\lambda_1} \leq t \vee X_{\lambda_2} \leq t\} \\
&= 1 - \Pr\{X_{\lambda_1} > t \wedge X_{\lambda_2} > t\} \\
&= 1 - (1 - \Pr\{X_{\lambda_1} \leq t\}) \cdot (1 - \Pr\{X_{\lambda_2} \leq t\}) \\
&= 1 - (1 - (1 - e^{-\lambda_1 \cdot t})) \cdot (1 - (1 - e^{-\lambda_2 \cdot t})) \\
&= 1 - e^{-(\lambda_1 + \lambda_2) \cdot t}
\end{aligned}$$

and the execution probabilities of the two actions are $\lambda_1/(\lambda_1+\lambda_2)$ and $\lambda_2/(\lambda_1+\lambda_2)$, respectively, because e.g.

$$\begin{aligned}
\Pr\{X_{\lambda_1} < X_{\lambda_2}\} &= \int_0^\infty \Pr\{X_{\lambda_2} > t\} d(\Pr\{X_{\lambda_1} \leq t\}) \\
&= \int_0^\infty e^{-\lambda_2 \cdot t} \cdot \lambda_1 \cdot e^{-\lambda_1 \cdot t} dt \\
&= \lambda_1/(\lambda_1 + \lambda_2)
\end{aligned}$$

Since these two probabilities are nonzero, the underlying LTS has two states that correspond to terms

$<a_1, \lambda_1>.\underline{0} + <a_2, \lambda_2>.\underline{0}$ and $\underline{0}$, respectively, as well as two transitions from the first state to the second state labeled with a_1, λ_1 and a_2, λ_2, respectively. Note that if $a_1 = a_2$ and $\lambda_1 = \lambda_2$ then the two transitions collapse into a single one, but this transition has to be given rate $2 \cdot \lambda_1$ instead of λ_1 because otherwise the sojourn time of the first state would not be consistent with the sojourn time expressed by the term at hand.

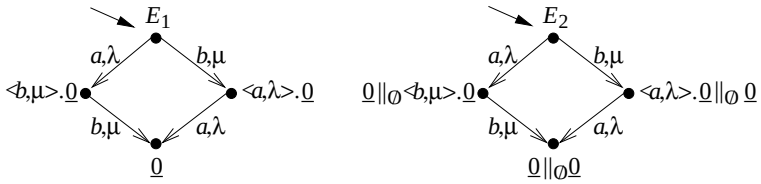
Consider now term $<a_1, \infty_{l_1,w_1}>.\underline{0} + <a_2, \infty_{l_2,w_2}>.\underline{0}$. Since both actions have the same duration hence the race policy does not apply, we choose the action to execute according to the *preselection policy*: only the actions having the highest priority level are executable and each of them is given an execution probability proportional to its own weight. The underlying LTS has two states that correspond to terms $<a_1, \infty_{l_1,w_1}>.\underline{0} + <a_2, \infty_{l_2,w_2}>.\underline{0}$ and $\underline{0}$, respectively, and the sojourn time in the first state is zero. If $l_1 > l_2$ ($l_2 > l_1$), then there is only one transition from the first state to the second state which is labeled with a_1, ∞_{l_1,w_1} (a_2, ∞_{l_2,w_2}). If $l_1 = l_2$, then there are two transitions from the first state to the second state which are labeled with a_1, ∞_{l_1,w_1} and a_2, ∞_{l_2,w_2}, respectively: the execution probability of the first transition is $w_1/(w_1 + w_2)$ while the execution probability of the second transition is $w_2/(w_1 + w_2)$. Note that if $a_1 = a_2$, $l_1 = l_2$, and $w_1 = w_2$ then the two transitions collapse into a single one, but this transition has to be given weight $2 \cdot w_1$ instead of w_1. In fact, if another alternative action with different type but the same priority level were present in the term, assigning weight w_1 instead of $2 \cdot w_1$ would alter the execution probability of an action with type a_1.

We assume that immediate actions take precedence over exponentially timed actions. If we consider e.g. term $<a_1, \lambda>.\underline{0} + <a_2, \infty_{l,w}>.\underline{0}$, then the underlying LTS has only one transition labeled with $a_2, \infty_{l,w}$.

Finally, consider term $<a_1, *>.\underline{0} + <a_2, *>.\underline{0}$. Since the duration of passive actions is undefined and these actions are assigned neither priority levels nor weights, they can be undergone to neither the race policy nor the preselection policy. This means that passive actions can be viewed as actions of classical process algebras, hence the term above expresses a purely *nondeterministic choice*.

3.3.3 Parallelism

Consider terms $E_1 \equiv <a, \lambda>.<b, \mu>.\underline{0} + <b, \mu>.<a, \lambda>.\underline{0}$ and $E_2 \equiv <a, \lambda>.\underline{0} \parallel_{\emptyset} <b, \mu>.\underline{0}$. Term E_1 represents a system that can execute either $<a, \lambda>$ followed by $<b, \mu>$ or $<b, \mu>$ followed by $<a, \lambda>$, while term E_2 represents a system that can execute $<a, \lambda>$ in parallel with $<b, \mu>$. As far as E_2 is concerned, it is natural to adopt the race policy to determine which action terminates first. The underlying LTSs are the following:



The isomorphism between the two LTSs is correct from the functional point of view, by definition of inter-

leaving, and also from the performance point of view. In fact, due to the *memoryless property* of exponential distributions [115], given an exponentially distributed random variable X it holds that the remaining delay after some time t' has elapsed is a random variable having the same distribution as X:

$$\Pr\{X \leq t + t' \mid X > t'\} = \Pr\{X \leq t\}$$

At the LTS level, this means that an exponentially timed action can be thought of as being started in the state where it is terminated. Therefore, if we assume that E_2 completes a before b, then the residual time to the completion of b is still exponentially distributed with rate μ, so the rate labeling the transition from state $\underline{0} \parallel_\emptyset <b, \mu>.\underline{0}$ to state $\underline{0} \parallel_\emptyset \underline{0}$ is μ itself instead of μ somehow conditional on the elapsed time.

Furthermore, it is worth noting that in the LTS for E_2 there is no transition from state E_2 to state $\underline{0} \parallel_\emptyset \underline{0}$ recording the possible simultaneous completion of a and b. The reason is that the probability of such a simultaneous completion is zero because the durations of a and b are described by means of continuous probability distribution functions.

If in E_1 and E_2 above a and b are both immediate, the preselection policy is adopted to decide which action terminates first. This implies that, from a modeling viewpoint, while durations can be assigned locally to actions of different system components because of the natural adoption of the race policy, priorities and weights must be assigned by looking at the global structure of a system because immediate actions (which thus have the same fixed duration) of different system components may interfere with each other. In particular, note that if the priority level of a is greater than the priority level of b, only the leftmost branch is actually present in the LTSs above. If a and b are both passive, instead, the action which terminates first is chosen nondeterministically.

As far as synchronization is concerned, we adopt the *bounded capacity assumption* [101]: the rate at which an action is carried out cannot be increased by synchronizing it with other actions. If we consider term $<a, \lambda>.\underline{0} \parallel_{\{a\}} <a, *>.\underline{0}$, the state associated with this term has only one outgoing transition with rate λ. If we consider instead term $F_1 \equiv <a, \lambda>.\underline{0} \parallel_{\{a\}} (<a, *>.\underline{0} + <a, *>.\underline{0})$ or $F_2 \equiv <a, \lambda>.\underline{0} \parallel_{\{a\}} (<a, *>.\underline{0} \parallel_\emptyset <a, *>.\underline{0})$, then two synchronizations are possible. Since both synchronizations involve the same active action and this has rate λ, the aggregated rate of the two synchronizations must be λ because of the race policy. To achieve this, the most neutral solution from the performance standpoint is to give the same execution probability to each transition resulting from the synchronization of the same active action with several alternative or independent passive actions is given the same execution probability. In our example, both synchronizations are thus assigned rate $\lambda/2$ through a rate normalization based on the equiprobability assumption. A similar procedure applies if we had $<a, \infty_{l,w}>$ in place of $<a, \lambda>$ in order to normalize weights in such a way that execution probabilities of immediate actions with different types are not altered.

We conclude by noting that turning nondeterminism into equiprobability in case of several synchronizations involving the same active action may seem arbitrary. However, what is important to preserve in such a case is the rate of the involved active action, which is the rate actually seen by an external

observer. Assigning a probability distribution like the uniform one to synchronization transitions and normalizing their rates accordingly is just a neutral way to preserve the observable rate in a performance model (MC) which does not allow nondeterminism. It is also worth noting that, should the probability of the synchronization transitions be known, this could be directly modeled. As an example, F_1 could be rewritten as $<a,\lambda>.\underline{0} \parallel_{\{a\}} (<\tau,\infty_{l,w_1}>.<a,*>.\underline{0} + <\tau,\infty_{l,w_2}>.<a,*>.\underline{0})$ and F_2 could be rewritten as $<a,\lambda>.\underline{0} \parallel_{\{a\}} ((<s_1,\infty_{l,w_1}>.<a,*>.\underline{0} + <s_2,*>.\underline{0}) \parallel_{\{s_1,s_2\}} (<s_2,\infty_{l,w_2}>.<a,*>.\underline{0} + <s_1,*>.\underline{0}))$ with $<s_1,\infty_{l,w_1}>$ ($<s_2,\infty_{l,w_2}>$) enabling the subsequent $<a,*>$ and disabling via synchronization on s_1 (s_2) the $<a,*>$ on the other side of the parallel composition.

3.4 Integrated and Projected Interleaving Semantics

In order to implement the first phase of the integrated approach of Fig. 1.1, we provide each EMPA term with a formally defined integrated interleaving semantics based on LTSs whose labels consist of both the type and the rate of the actions. From such a semantics two projected interleaving semantic models can be derived which describe system functionality and system performance, respectively.

The main problem to tackle when defining the semantics for EMPA is that the actions executable by a given term may have different priority levels but only those having the highest priority level are actually executable. Let us call the *potential move* of a given term a pair composed of an action executable by that term when ignoring priority levels and the derivative term obtained by executing that action; let us denote by $PMove = Act \times \mathcal{G}$ the set of all the potential moves. To solve the problem above, we compute inductively the multiset of the potential moves of a given term regardless of priority levels and then we select those having the highest priority level. Computing the multiset of all the potential moves is further motivated in our framework by the fact that not only the actual executability but also the execution probability of an action depend upon all the actions that are executable at the same time when it is executable. Only if we know all the potential moves of a given term with the related multiplicities, we can correctly determine the transitions of the corresponding state and their rates avoiding transition collapses and respecting the bounded capacity assumption. This is clarified by the following example.

Example 3.1 Consider term

$$E \equiv <a,\infty_{3,1}>.E_1 + <e,\infty_{2,1}>.A$$

where

$$\begin{aligned} E_1 &\equiv <b,\lambda>.(\underline{0} \parallel_{\emptyset} \underline{0}) + <c,\infty_{1,1}>.E_2 \\ E_2 &\equiv <h,\xi>.E_3 + <h,\xi>.E_3 \\ E_3 &\equiv <d,\mu>.\underline{0} \parallel_{\{d\}} (<d,*>.\underline{0} \parallel_{\emptyset} <d,*>.\underline{0}) \\ A &\triangleq <f,\gamma>.A \end{aligned}$$

Suppose we apply to E standard semantic rules for classical process algebras (such as those in Table 2.1),

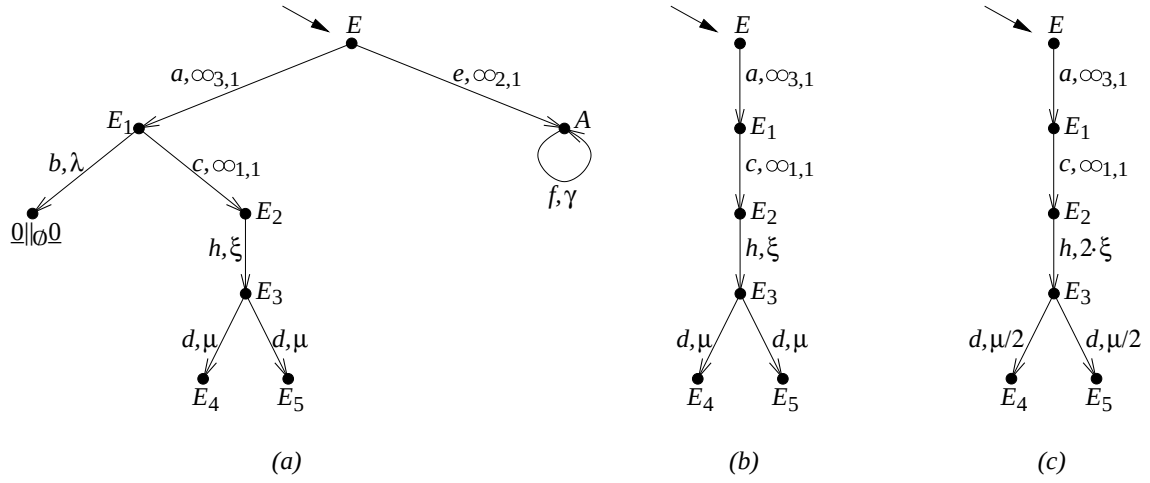


Figure 3.1: Integrated interleaving semantic models for Ex. 3.1

thereby disregarding priority levels, probabilities, and durations. Then we obtain the LTS in Fig. 3.1(a) with

$$\begin{aligned} E_4 &\equiv \underline{0} \parallel_{\{d\}} (\underline{0} \parallel_{\emptyset} <d, *>.\underline{0}) \\ E_5 &\equiv \underline{0} \parallel_{\{d\}} (<d, *>.\underline{0} \parallel_{\emptyset} \underline{0}) \end{aligned}$$

where transitions are in exact correspondence with the potential moves.

Now assume that priority levels are taken into account. Then lower priority transitions must be pruned, thus resulting in the LTS in Fig. 3.1(b). The new LTS is obtained by means of an auxiliary function we shall call *Select*.

Finally, consider the rate of the transition from E_2 to E_3 and the rates of the two transitions from E_3 to E_4 and E_5. In the correct semantic model for E, such rates have to be like in Fig. 3.1(c). As far as the transition from E_2 to E_3 is concerned, its rate is $2 \cdot \xi$ instead of ξ because in E_2 two exponentially timed actions with rate ξ occur and the race policy has been adopted. The problem is that both exponentially timed actions have the same type and rate and results in the same derivative term, so with classical semantic rules only one transition is produced. The same problem arises in the case of immediate actions. To overcome this, instead of producing e.g. two transitions with two different auxiliary labels [92], one transition having multiplicity two (which requires the adoption of a variant of LTS as a semantic model) [100], or directly one transition with the correct rate by means of auxiliary semantic rules [74], we keep track of the multiplicity of potential moves and then we construct transitions using an auxiliary function we shall call *Melt* that merges together those potential moves having the same action type, the same priority level, and the same derivative term. The rate of transitions derived by merging potential moves is computed by means of another auxiliary function we shall call *Aggr*.

As far as the transitions from E_3 to E_4 and E_5 are concerned, their rate is $\mu/2$ instead of μ because of the adoption of the bounded capacity assumption: the value $\mu/2$ stems from the assumption that independent passive actions have the same probability to participate in a synchronization. The same considerations would

hold if in E_3 we had an immediate action instead of an exponentially timed action or alternative passive actions instead of independent passive actions. In all of these cases a normalization of rates is required and this is carried out by means of an auxiliary function we shall call *Norm*.

The reader is invited to look again at this example after examining the formal definition of the semantics, in order to verify that the LTS of Fig. 3.1(c) is exactly the result of the application to E of the rules in Table 3.1 equipped with the auxiliary functions mentioned above. ■

A comment on different viewpoints (system view and component view) for priority levels is now in order. We take the view that the semantic model underlying an EMPA term should describe the actual behavior of the system represented by that term. In order to faithfully describe the actual behavior, lower priority transitions must be pruned. Our view is convenient from the modeling standpoint, as in practice it is always the case that a (possibly large) system is considered as a whole, so the priority levels imposed by the designer can be safely reflected by the semantic model underlying the term representing the system. A full matching of the intended meaning of a term and its underlying semantic model is important because it increases the confidence of the designer w.r.t. the correctness of the term developed to represent the system. We are nevertheless aware that our view may not be the most adequate when analyzing a system compositionally. Here the problem is that the components of a system interact with each other, so lower priority behaviors of a component (which do not belong to the semantic model of the component in isolation) may be exposed when that component interacts with the rest of the system (i.e. may belong to the semantic model of the whole system). As a consequence, in principle with our view it is not always possible e.g. to compositionally minimize the semantic model of the term representing a system. Fortunately, the choice of our view is supported by the fact that, as we shall see in Chap. 4 and 5, the ability to compositionally analyze a term can be retrieved by means of some technicalities which can be made transparent to the designer.

The formal definition of the integrated interleaving semantics for EMPA is shown in Table 3.1, which is composed of five parts grouped in two sections: the upper section contains the semantic rules, while the lower section contains the auxiliary fucntions used in the semantic rules. The integrated interleaving semantics for EMPA is based on the transition relation $\longrightarrow$, which is the least subset of $\mathcal{G} \times Act \times \mathcal{G}$ satisfying the inference rule in the first part of Table 3.1. This rule selects the potential moves that have the highest priority level (or are passive) and then merges together those having the same action type, the same priority level, and the same derivative term. The first operation is carried out through functions $Select : \mathcal{M}u_{\rm fin}(PMove) \longrightarrow \mathcal{M}u_{\rm fin}(PMove)$ and $PL : Act \longrightarrow APLev$, which are defined in the third part of Table 3.1. The second operation is carried out through function $Melt : \mathcal{M}u_{\rm fin}(PMove) \longrightarrow \mathcal{P}_{\rm fin}(PMove)$ and partial function $Aggr : ARate \times ARate \nrightarrow ARate$, which are defined in the fourth part of Table 3.1. We recall that function *Melt*, whose introduction is motivated by the need of preventing transitions from collapsing, avoids burdening transitions with auxiliary labels as well as keeping track of the fact that some transitions may have multiplicity greater than one. We regard *Aggr* as an associative and commutative operation, thus we take the liberty to apply it to multisets of rates.

$$\frac{(<a,\tilde{\lambda}>,E') \in Melt(Select(PM(E)))}{E \xrightarrow{a,\tilde{\lambda}} E'}$$

$$\begin{array}{rcl}
PM(\underline{0}) & = & \emptyset \\
PM(<a,\tilde{\lambda}>.E) & = & \{\!|\, (<a,\tilde{\lambda}>,E) \,|\!\} \\
PM(E/L) & = & \{\!|\, (<a,\tilde{\lambda}>,E'/L) \mid (<a,\tilde{\lambda}>,E') \in PM(E) \wedge a \notin L \,|\!\} \oplus \\
 & & \{\!|\, (<\tau,\tilde{\lambda}>,E'/L) \mid \exists a \in L.\, (<a,\tilde{\lambda}>,E') \in PM(E) \,|\!\} \\
PM(E[\varphi]) & = & \{\!|\, (<\varphi(a),\tilde{\lambda}>,E'[\varphi]) \mid (<a,\tilde{\lambda}>,E') \in PM(E) \,|\!\} \\
PM(E_1 + E_2) & = & PM(E_1) \oplus PM(E_2) \\
PM(E_1 \parallel_S E_2) & = & \{\!|\, (<a,\tilde{\lambda}>,E_1' \parallel_S E_2) \mid a \notin S \wedge (<a,\tilde{\lambda}>,E_1') \in PM(E_1) \,|\!\} \oplus \\
 & & \{\!|\, (<a,\tilde{\lambda}>,E_1 \parallel_S E_2') \mid a \notin S \wedge (<a,\tilde{\lambda}>,E_2') \in PM(E_2) \,|\!\} \oplus \\
 & & \{\!|\, (<a,\tilde{\gamma}>,E_1' \parallel_S E_2') \mid a \in S \wedge \exists \tilde{\lambda}_1,\tilde{\lambda}_2. \\
 & & \qquad (<a,\tilde{\lambda}_1>,E_1') \in PM(E_1) \wedge \\
 & & \qquad (<a,\tilde{\lambda}_2>,E_2') \in PM(E_2) \wedge \\
 & & \qquad \tilde{\gamma} = Norm(a,\tilde{\lambda}_1,\tilde{\lambda}_2,PM(E_1),PM(E_2)) \,|\!\} \\
PM(A) & = & PM(E) \qquad \text{if } A \triangleq E
\end{array}$$

$$Select(PM) = \{\!|\, (<a,\tilde{\lambda}>,E) \in PM \mid \forall (<b,\tilde{\mu}>,E') \in PM.\, PL(<a,\tilde{\lambda}>) \geq PL(<b,\tilde{\mu}>) \vee PL(<a,\tilde{\lambda}>) = -1 \,|\!\}$$

$$PL(<a,*>) = -1 \qquad PL(<a,\lambda>) = 0 \qquad PL(<a,\infty_{l,w}>) = l$$

$$Melt(PM) = \{(<a,\tilde{\lambda}>,E) \mid \exists \tilde{\mu} \in ARate.\, (<a,\tilde{\mu}>,E) \in PM \wedge \tilde{\lambda} = Aggr\{\!|\, \tilde{\gamma} \mid (<a,\tilde{\gamma}>,E) \in PM \wedge PL(<a,\tilde{\gamma}>) = PL(<a,\tilde{\mu}>) \,|\!\}\}$$

$$* \; Aggr \; * = * \qquad \lambda_1 \; Aggr \; \lambda_2 = \lambda_1 + \lambda_2 \qquad \infty_{l,w_1} \; Aggr \; \infty_{l,w_2} = \infty_{l,w_1+w_2}$$

$$Norm(a,\tilde{\lambda}_1,\tilde{\lambda}_2,PM_1,PM_2) = \begin{cases} Split(\tilde{\lambda}_1, 1/(\pi_1(PM_2))(<a,*>)) & \text{if } \tilde{\lambda}_2 = * \\ Split(\tilde{\lambda}_2, 1/(\pi_1(PM_1))(<a,*>)) & \text{if } \tilde{\lambda}_1 = * \end{cases}$$

$$Split(*,p) = * \qquad Split(\lambda,p) = \lambda \cdot p \qquad Split(\infty_{l,w},p) = \infty_{l,w \cdot p}$$

Table 3.1: Inductive rules for EMPA integrated interleaving semantics

The multiset $PM(E) \in \mathcal{M}u_{\rm fin}(PMove)$ of potential moves of $E \in \mathcal{G}$ is defined by structural induction in the second part of Table 3.1 according to the intuitive meaning of the operators explained in Sect. 3.2. It is worth noting that, unlike the definition of the semantics for classical process algebras (see Table 2.1), we compute all the potential moves of a term at once instead of computing one potential move at a time, since this is the most convenient way to correctly determine the transitions (*Select*) and their rates (*Melt* and *Norm*). In order to enforce the bounded capacity assumption, in the rule for the parallel composition operator a normalization is required which suitably computes the rates of potential moves resulting from the synchronization of the same active action with several independent or alternative passive actions. The normalization operates in such a way that applying *Aggr* to the rates of the synchronizations involving the active action gives as a result the rate of the active action itself, and that each synchronization is assigned the same execution probability. This normalization is carried out through partial function $Norm : AType \times ARate \times ARate \times \mathcal{M}u_{\rm fin}(PMove) \times \mathcal{M}u_{\rm fin}(PMove) \not\longrightarrow ARate$ and function $Split : ARate \times \mathbb{R}_{]0,1]} \longrightarrow ARate$, which are defined in the fifth part of Table 3.1. [1] Note that $Norm(a, \tilde{\lambda}_1, \tilde{\lambda}_2, PM_1, PM_2)$ is defined if and only if $\min(\tilde{\lambda}, \tilde{\mu}) = *$, which is the condition on action rates we have required in Sect. 3.2 in order for a synchronization to be permitted.

Example 3.2 Consider term

$$E \equiv E_1 \parallel_{\{a\}} (E_2 \parallel_\emptyset E_3)$$

where

$$\begin{aligned} E_1 &\equiv <a, \lambda>.\underline{0} \\ E_2 &\equiv <a, *>.\underline{0} + <a, *>.\underline{0} \\ E_3 &\equiv <a, *>.\underline{0} \end{aligned}$$

Then E_1 has one potential move $(<a, \lambda>, \underline{0})$, E_2 has one potential move $(<a, *>, \underline{0})$ with multiplicity two, and E_3 has one potential move $(<a, *>, \underline{0})$. As a consequence, $E_2 \parallel_\emptyset E_3$ has both potential move $(<a, *>, \underline{0} \parallel_\emptyset E_3)$ with multiplicity two and potential move $(<a, *>, E_2 \parallel_\emptyset \underline{0})$. Therefore, when computing the potential moves for E, function *Norm* produces both $(<a, \lambda/3>, \underline{0} \parallel_{\{a\}} (\underline{0} \parallel_\emptyset E_3))$ with multiplicity two and $(<a, \lambda/3>, \underline{0} \parallel_{\{a\}} (E_2 \parallel_\emptyset \underline{0}))$, and subsequently function *Melt* produces both $(<a, 2 \cdot \lambda/3>, \underline{0} \parallel_{\{a\}} (\underline{0} \parallel_\emptyset E_3))$ and $(<a, \lambda/3>, \underline{0} \parallel_{\{a\}} (E_2 \parallel_\emptyset \underline{0}))$, as expected. ■

Definition 3.2 *The integrated interleaving semantics of $E \in \mathcal{G}$ is the LTS*

$$\mathcal{I}[\![E]\!] = (S_{E,\mathcal{I}}, Act, \longrightarrow_{E,\mathcal{I}}, E)$$

where:

- *$S_{E,\mathcal{I}}$ is the least subset of $\mathcal{G}$ such that:*

[1] We recall that e.g. $(\pi_1(PM_2))(<a, *>)$ in the fifth part of Table 3.1 denotes the multiplicity of tuples of PM_2 whose first component is $<a, *>$.

  - $E \in S_{E,\mathcal{I}}$;
  - *if* $E_1 \in S_{E,\mathcal{I}}$ *and* $E_1 \stackrel{a,\tilde{\lambda}}{\longrightarrow} E_2$, *then* $E_2 \in S_{E,\mathcal{I}}$.

- $\longrightarrow_{E,\mathcal{I}}$ *is the restriction of* $\longrightarrow$ *to* $S_{E,\mathcal{I}} \times Act \times S_{E,\mathcal{I}}$. ■

Definition 3.3 *$E \in \mathcal{G}$ is performance closed if and only if $\mathcal{I}[\![E]\!]$ does not contain passive transitions. We denote by $\mathcal{E}$ the set of performance closed terms of $\mathcal{G}$.* ■

Because of function *Select*, a tangible state has only outgoing exponentially timed transitions and, possibly, passive transitions. Likewise, a vanishing state has only outgoing immediate transitions of the same priority level and, possibly, passive transitions. If the term at hand is performance closed, which means that it is completely specified from the performance standpoint, then neither tangible states nor vanishing states have outgoing passive transitions.

Given a term $E \in \mathcal{G}$, from its integrated interleaving semantics $\mathcal{I}[\![E]\!]$, which fully represents the behavior of E because transitions are decorated by both action types and action rates, we can derive two projected semantic models by simply dropping action rates or action types, respectively.

Definition 3.4 *The functional semantics of $E \in \mathcal{G}$ is the LTS*

$$\mathcal{F}[\![E]\!] = (S_{E,\mathcal{F}}, AType, \longrightarrow_{E,\mathcal{F}}, E)$$

where:

- $S_{E,\mathcal{F}} = S_{E,\mathcal{I}}$.
- $\longrightarrow_{E,\mathcal{F}}$ *is the restriction of* $\longrightarrow_{E,\mathcal{I}}$ *to* $S_{E,\mathcal{F}} \times AType \times S_{E,\mathcal{F}}$. ■

The performance semantics of a performance closed term is a HCTMC, a HDTMC, or a HSMC depending on whether the underlying integrated interleaving semantic model has only exponentially timed transitions, only immediate transitions, or both. In the case of a HCTMC, all the transitions from one state to the same state are merged into a single transition whose rate is the sum of the original rates according to the race policy. In the case of a HDTMC, the probability of the transition resulting from merging all the transitions from one state to the same state is computed as the ratio of its weight (which is the sum of the original weights) to the sum of the weights of all the transitions leaving the same state as the transition at hand according to the preselection policy. The performance semantics, hereafter called Markovian semantics, can thus be represented as a p-LTS.

Definition 3.5 *The Markovian semantics of $E \in \mathcal{E}$ is the p-LTS*

$$\mathcal{M}[\![E]\!] = (S_{E,\mathcal{M}}, \mathbb{R}_+, \longrightarrow_{E,\mathcal{M}}, P_{E,\mathcal{M}}, H_{E,\mathcal{M}})$$

where:

- $S_{E,\mathcal{M}} = S_{E,\mathcal{I}}$.
- $\longrightarrow_{E,\mathcal{M}}$ *is the least subset of* $S_{E,\mathcal{M}} \times \mathbb{R}_+ \times S_{E,\mathcal{M}}$ *such that:*
  - $F \xrightarrow{\lambda}_{E,\mathcal{M}} F'$ *whenever* $\lambda = \sum\{\!|\, \mu \mid \exists a.\, F \xrightarrow{a,\mu}_{E,\mathcal{I}} F' \,|\!\}$;
  - $F \xrightarrow{p}_{E,\mathcal{M}} F'$ *whenever* $p = \sum\{\!|\, w \mid \exists a, l.\, F \xrightarrow{a,\infty_{l,w}}_{E,\mathcal{I}} F' \,|\!\} / \sum\{\!|\, w \mid \exists a, l, F''.\, F \xrightarrow{a,\infty_{l,w}}_{E,\mathcal{I}} F'' \,|\!\}$.
- $P_{E,\mathcal{M}} : S_{E,\mathcal{M}} \longrightarrow \mathbb{R}_{[0,1]}$, $P_{E,\mathcal{M}}(F) = \begin{cases} 1 & \text{if } F \equiv E \\ 0 & \text{if } F \not\equiv E \end{cases}$
- $H_{E,\mathcal{M}} : S_{E,\mathcal{M}} \longrightarrow \{v, t, a\}$, $H_{E,\mathcal{M}}(F) = \begin{cases} v & \text{if } \exists a, l, w, F'.\, F \xrightarrow{a,\infty_{l,w}}_{E,\mathcal{I}} F' \\ t & \text{if } \exists a, \lambda, F'.\, F \xrightarrow{a,\lambda}_{E,\mathcal{I}} F' \\ a & \text{if } \nexists a, \tilde{\lambda}, F'.\, F \xrightarrow{a,\tilde{\lambda}}_{E,\mathcal{I}} F' \end{cases}$

  where v *stands for vanishing,* t *for tangible, and* a *for absorbing.* ■

Example 3.3 Consider again term E of Ex. 3.1. We show $\mathcal{F}[\![E]\!]$ and $\mathcal{M}[\![E]\!]$ in Fig. 3.2(a) and (b), respectively. ■

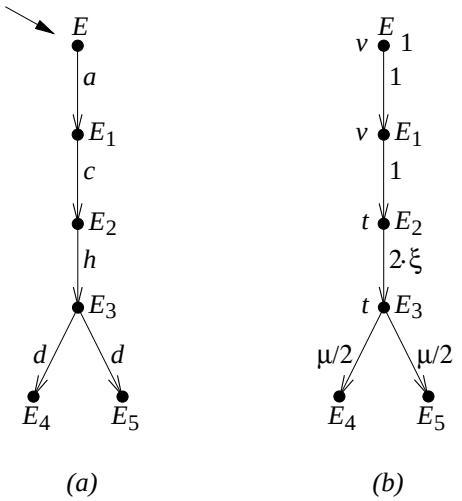


Figure 3.2: Projected interleaving semantic models for Ex. 3.1

Chapter 4

On the Expressive Power of EMPA

Due to the coexistence of exponentially timed actions, prioritized weighted immediate actions, and passive actions, EMPA can be viewed as being made out of four kernels (see Fig. 4.1): a *nondeterministic kernel*, a *prioritized kernel*, a *probabilistic kernel*, and an *exponentially timed kernel*. The purpose of this chapter is to dwell upon EMPA in order to assess its expressive power by investigating its four kernels [10, 29]. As expected, we shall see that the expressiveness of EMPA is considerable. In particular, some of the systems reported in the examples of this chapter and Chap. 10 cannot be modeled with other Markovian process algebras. This modeling related argument somehow justifies the complexity of the definition of the semantics for EMPA given in Chap. 3.

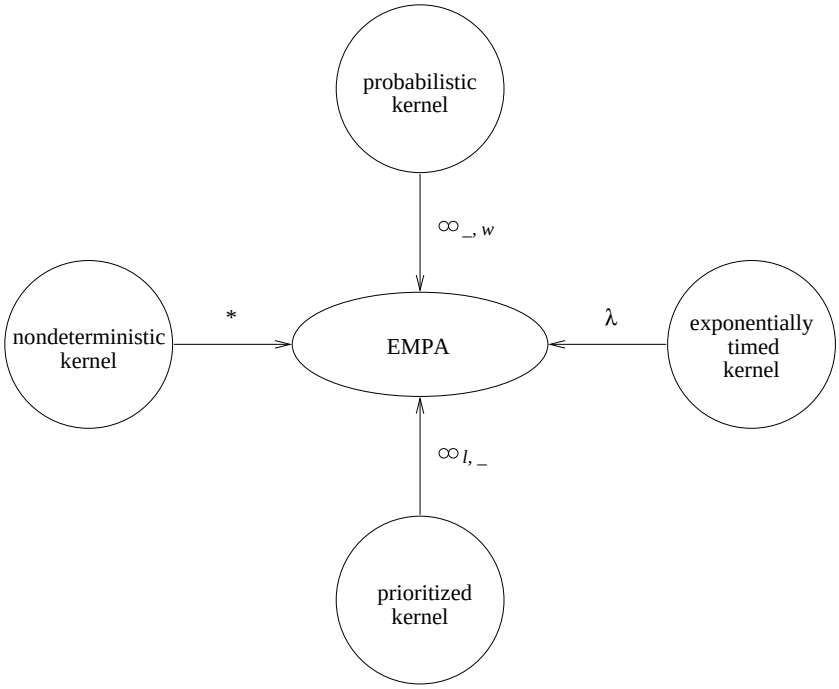


Figure 4.1: EMPA kernels

This chapter is organized as follows. In Sect. 4.1 to 4.4 we examine each of the four kernels separately by presenting how the functional semantics for EMPA (in the case of the nondeterministic kernel and the prioritized kernel) or the integrated interleaving semantics for EMPA and its projections (in the case of the probabilistic kernel and the exponentially timed kernel) specializes to the kernel at hand, and by making some comparisons with related process algebras appeared in the literature. In Sect. 4.5 we consider the interplay between the probabilistic kernel and the exponentially timed kernel from the point of view of the performance semantics and we recognize that it allows phase type distributions to be modeled. In Sect. 4.6 we emphasize the role played by passive actions and we report some remarks on the expressiveness of the synchronization discipline on action rates adopted in EMPA. Finally, in Sect. 4.7 we show how to model more and more complex QSs with EMPA in order to stress its expressive power.

4.1 Nondeterministic Kernel

The *nondeterministic kernel* $\mathrm{EMPA}_{\mathrm{nd}}$ is the sublanguage of EMPA obtained by considering only the passive actions. Since passive actions have no durations, priority levels, and weights associated with them, $\mathrm{EMPA}_{\mathrm{nd}}$ is a classical process algebra and allows for pure nondeterminism. We define below the syntax and the functional semantics of $\mathrm{EMPA}_{\mathrm{nd}}$ terms.

Definition 4.1 *The set $\mathcal{L}_{\mathrm{nd}}$ of process terms of EMPA$_{\mathrm{nd}}$ is generated by the following syntax*

$$E ::= \underline{0} \mid <a, *>.E \mid E/L \mid E[\varphi] \mid E + E \mid E \parallel_S E \mid A$$

where $L, S \subseteq AType - \{\tau\}$. We denote by $\mathcal{G}_{\mathrm{nd}}$ the set of closed and guarded terms of $\mathcal{L}_{\mathrm{nd}}$. ■

Definition 4.2 *The functional semantics of $E \in \mathcal{G}_{\mathrm{nd}}$ is the LTS*

$$\mathcal{F}_{\mathrm{nd}}[\![E]\!] = (S_{E,\mathcal{F}_{\mathrm{nd}}}, AType, \longrightarrow_{E,\mathcal{F}_{\mathrm{nd}}}, E)$$

where:

- *$S_{E,\mathcal{F}_{\mathrm{nd}}}$ is the least subset of $\mathcal{G}_{\mathrm{nd}}$ such that:*
  - *$E \in S_{E,\mathcal{F}_{\mathrm{nd}}}$;*
  - *if $E_1 \in S_{E,\mathcal{F}_{\mathrm{nd}}}$ and $E_1 \stackrel{a}{\longrightarrow}_{\mathcal{F}_{\mathrm{nd}}} E_2$, then $E_2 \in S_{E,\mathcal{F}_{\mathrm{nd}}}$.*
- *$\longrightarrow_{E,\mathcal{F}_{\mathrm{nd}}}$ is the restriction of $\longrightarrow_{\mathcal{F}_{\mathrm{nd}}}$ (defined in Table 4.1) to $S_{E,\mathcal{F}_{\mathrm{nd}}} \times AType \times S_{E,\mathcal{F}_{\mathrm{nd}}}$.* ■

Proposition 4.1 *For all $E \in \mathcal{G}_{\mathrm{nd}}$, $\mathcal{F}_{\mathrm{nd}}[\![E]\!]$ is isomorphic to $\mathcal{F}[\![E]\!]$.* ■

We conclude by observing that the operators of $\mathrm{EMPA}_{\mathrm{nd}}$ are the same as the operators of the example process algebra of Sect. 2.1 and the semantic rules are the same as those in Table 2.1. Therefore, $\mathrm{EMPA}_{\mathrm{nd}}$ is a mix of classical process algebras such as CCS [133], CSP [106], ACP [8], and LOTOS [33], hence all the results and analysis techniques developed for classical process algebras can be applied to $\mathrm{EMPA}_{\mathrm{nd}}$.

$$<a,*>.E \xrightarrow{a}_{\mathcal{F}_{\rm nd}} E$$

$$\frac{E \xrightarrow{a}_{\mathcal{F}_{\rm nd}} E'}{E/L \xrightarrow{a}_{\mathcal{F}_{\rm nd}} E'/L} \text{ if } a \notin L \qquad \frac{E \xrightarrow{a}_{\mathcal{F}_{\rm nd}} E'}{E/L \xrightarrow{\tau}_{\mathcal{F}_{\rm nd}} E'/L} \text{ if } a \in L$$

$$\frac{E \xrightarrow{a}_{\mathcal{F}_{\rm nd}} E'}{E[\varphi] \xrightarrow{\varphi(a)}_{\mathcal{F}_{\rm nd}} E'[\varphi]}$$

$$\frac{E_1 \xrightarrow{a}_{\mathcal{F}_{\rm nd}} E'}{E_1 + E_2 \xrightarrow{a}_{\mathcal{F}_{\rm nd}} E'} \qquad \frac{E_2 \xrightarrow{a}_{\mathcal{F}_{\rm nd}} E'}{E_1 + E_2 \xrightarrow{a}_{\mathcal{F}_{\rm nd}} E'}$$

$$\frac{E_1 \xrightarrow{a}_{\mathcal{F}_{\rm nd}} E_1'}{E_1 \parallel_S E_2 \xrightarrow{a}_{\mathcal{F}_{\rm nd}} E_1' \parallel_S E_2} \text{ if } a \notin S \qquad \frac{E_2 \xrightarrow{a}_{\mathcal{F}_{\rm nd}} E_2'}{E_1 \parallel_S E_2 \xrightarrow{a}_{\mathcal{F}_{\rm nd}} E_1 \parallel_S E_2'} \text{ if } a \notin S$$

$$\frac{E_1 \xrightarrow{a}_{\mathcal{F}_{\rm nd}} E_1' \quad E_2 \xrightarrow{a}_{\mathcal{F}_{\rm nd}} E_2'}{E_1 \parallel_S E_2 \xrightarrow{a}_{\mathcal{F}_{\rm nd}} E_1' \parallel_S E_2'} \text{ if } a \in S$$

$$\frac{E \xrightarrow{a}_{\mathcal{F}_{\rm nd}} E'}{A \xrightarrow{a}_{\mathcal{F}_{\rm nd}} E'} \text{ if } A \triangleq E$$

Table 4.1: Inductive rules for $\text{EMPA}_{\rm nd}$ functional semantics

4.2 Prioritized Kernel

The *prioritized kernel* $\text{EMPA}_{{\rm pt},w}$ is the sublanguage of EMPA obtained by considering only the passive actions and the immediate actions having weight w. Since each immediate action is given a priority level, $\text{EMPA}_{{\rm pt},w}$ is a prioritized process algebra: in this framework, the priority level of a passive action is considered to be unspecified and the weight and the duration of an immediate action are ignored. We define below the syntax and the functional semantics of $\text{EMPA}_{{\rm pt},w}$ terms.

Definition 4.3 *The set $\mathcal{L}_{{\rm pt},w}$ of process terms of $EMPA_{{\rm pt},w}$ is generated by the following syntax*

$$E ::= \underline{0} \mid <a,\tilde{\lambda}>.E \mid E/L \mid E[\varphi] \mid E + E \mid E \parallel_S E \mid A$$

where $\tilde{\lambda} \in \{\infty_{l,w} \mid l \in \mathbb{N}_+\} \cup \{*\}$ and $L, S \subseteq AType - \{\tau\}$. We denote by $\mathcal{G}_{{\rm pt},w}$ the set of closed and guarded terms of $\mathcal{L}_{{\rm pt},w}$. ■

Definition 4.4 *The functional semantics of $E \in \mathcal{G}_{\mathrm{pt},w}$ is the LTS*

$$\mathcal{F}_{\mathrm{pt},w}[\![E]\!] = (S_{E,\mathcal{F}_{\mathrm{pt},w}}, \mathit{AType}, \longrightarrow_{E,\mathcal{F}_{\mathrm{pt},w}}, E)$$

where:

- *$S_{E,\mathcal{F}_{\mathrm{pt},w}}$ is the least subset of $\mathcal{G}_{\mathrm{pt},w}$ such that:*
  - *$E \in S_{E,\mathcal{F}_{\mathrm{pt},w}}$;*
  - *if $E_1 \in S_{E,\mathcal{F}_{\mathrm{pt},w}}$ and $E_1 \stackrel{a}{\longrightarrow}_{\mathcal{F}_{\mathrm{pt},w}} E_2$, then $E_2 \in S_{E,\mathcal{F}_{\mathrm{pt},w}}$.*
- *$\longrightarrow_{E,\mathcal{F}_{\mathrm{pt},w}}$ is the restriction of $\longrightarrow_{\mathcal{F}_{\mathrm{pt},w}}$ (defined in Table 4.2) to $S_{E,\mathcal{F}_{\mathrm{pt},w}} \times \mathit{AType} \times S_{E,\mathcal{F}_{\mathrm{pt},w}}$.* ■

Functions not defined in Table 4.2 are as in Table 3.1.

$$\frac{(<a,\tilde{\lambda}>,E') \in \mathit{Select}(PM(E))}{E \stackrel{a}{\longrightarrow}_{\mathcal{F}_{\mathrm{pt},w}} E'}$$

$$\begin{array}{rcl}
PM(\underline{0}) & = & \emptyset \\
PM(<a,\tilde{\lambda}>.E) & = & \{(<a,\tilde{\lambda}>,E)\} \\
PM(E/L) & = & \{(<a,\tilde{\lambda}>,E'/L) \mid (<a,\tilde{\lambda}>,E') \in PM(E) \wedge a \notin L\} \cup \\
 & & \{(<\tau,\tilde{\lambda}>,E'/L) \mid (<a,\tilde{\lambda}>,E') \in PM(E) \wedge a \in L\} \\
PM(E[\varphi]) & = & \{(<\varphi(a),\tilde{\lambda}>,E'[\varphi]) \mid (<a,\tilde{\lambda}>,E') \in PM(E)\} \\
PM(E_1 + E_2) & = & PM(E_1) \cup PM(E_2) \\
PM(E_1 \parallel_S E_2) & = & \{(<a,\tilde{\lambda}>,E_1' \parallel_S E_2) \mid a \notin S \wedge (<a,\tilde{\lambda}>,E_1') \in PM(E_1)\} \cup \\
 & & \{(<a,\tilde{\lambda}>,E_1 \parallel_S E_2') \mid a \notin S \wedge (<a,\tilde{\lambda}>,E_2') \in PM(E_2)\} \cup \\
 & & \{(<a,\tilde{\gamma}>,E_1' \parallel_S E_2') \mid a \in S \wedge \\
 & & \qquad (<a,\tilde{\lambda}_1>,E_1') \in PM(E_1) \wedge \\
 & & \qquad (<a,\tilde{\lambda}_2>,E_2') \in PM(E_2) \wedge \\
 & & \qquad ((\tilde{\lambda}_1 = * \wedge \tilde{\gamma} = \tilde{\lambda}_2) \vee (\tilde{\lambda}_2 = * \wedge \tilde{\gamma} = \tilde{\lambda}_1))\} \\
PM(A) & = & PM(E) \qquad \text{if } A \triangleq E
\end{array}$$

Table 4.2: Inductive rules for $\mathrm{EMPA}_{\mathrm{pt},w}$ functional semantics

Proposition 4.2 *For all $E \in \mathcal{G}_{\mathrm{pt},w}$, $\mathcal{F}_{\mathrm{pt},w}[\![E]\!]$ is isomorphic to $\mathcal{F}[\![E]\!]$.* ■

Since every active action is given an explicit priority level, from the syntactical point of view $\text{EMPA}_{\text{pt},w}$ is similar to the proposal of [51] where for each action type both a prioritized version and an unprioritized version are provided. The difference is that in [51] unprioritized actions are preempted only by internal prioritized actions in order to get a congruence, while the semantic treatment of priorities in $\text{EMPA}_{\text{pt},w}$ is quite different because the actual priority level of an action is independent of its visibility. As we shall see in Chap. 5, in order to get a congruence in the presence of actions having different priority levels we shall extend EMPA with a priority operator in the style of [6] such that priority levels are enforced only within its scope (every EMPA term can be thought of as having a single occurrence of the priority operator as outermost operator). Our idea of leaving priority levels as they are specified within actions is convenient from the modeling point of view, as the system designer is not required to introduce artificial prioritized τ loops in the algebraic descriptions like in [51] nor to burden the algebraic descriptions with occurrences of the priority operator like in [6]. We also would like to remind that the treatment of priorities in EMPA is the same as that of GSPNs. This will ease mapping EMPA terms into GSPNs in Chap. 6.

Furthermore, we observe that $\text{EMPA}_{\text{pt},w}$ is different from the proposal of [44], where a prioritized choice operator is explicitly defined, from the proposal of [179], where priority is expressed as extremal probability and the computation proceeds in locksteps, from CCSR [68], where priority is used to arbitrate between simultaneous resource requests and lockstep parallelism is considered, and from CCS^{prio} [53], where actions are allowed to preempt others only at the same site so as to capture a notion of localized precedence.

Finally, if we consider the features a prioritized process algebra should possess according to [179], we have that the priority relation of $\text{EMPA}_{\text{pt},w}$ is globally dynamic, i.e. it may be the case that an action with type a has priority over an action with type b in one state and the converse in some other state. On the other hand, the priority relation of $\text{EMPA}_{\text{pt},w}$ cannot define arbitrary partial orders because the leveled priority structure causes incomparable actions to have the same priority. As an example, it is not possible in $\text{EMPA}_{\text{pt},w}$ to express the fact that, in a given state, action type a takes precedence over action types b and c while action type d takes precedence only over action type c.

4.3 Probabilistic Kernel

The *probabilistic kernel* $\text{EMPA}_{\text{pb},l}$ is the sublanguage of EMPA obtained by considering only the passive actions and the immediate actions having priority level l. Since each immediate action is given a weight, $\text{EMPA}_{\text{pb},l}$ is a probabilistic process algebra: in this framework, the weight of a passive action is considered to be unspecified and the priority level and the duration of an immediate action are ignored. We define below the syntax, the integrated semantics, the functional semantics, and the performance semantics of $\text{EMPA}_{\text{pb},l}$ terms.

Definition 4.5 *The set $\mathcal{L}_{\text{pb},l}$ of process terms of $\text{EMPA}_{\text{pb},l}$ is generated by the following syntax*

$$E ::= \underline{0} \mid <a, \tilde{\lambda}>.E \mid E/L \mid E[\varphi] \mid E + E \mid E \parallel_S E \mid A$$

where $\tilde{\lambda} \in \{\infty_{l,w} \mid w \in \mathbb{R}_+\} \cup \{*\}$ *and* $L, S \subseteq AType - \{\tau\}$. *We denote by* $\mathcal{G}_{\mathrm{pb},l}$ *the set of closed and guarded terms of* $\mathcal{L}_{\mathrm{pb},l}$. ■

Definition 4.6 *The integrated semantics of* $E \in \mathcal{G}_{\mathrm{pb},l}$ *is the LTS*

$$\mathcal{I}_{\mathrm{pb},l}[\![E]\!] = (S_{E,\mathcal{I}_{\mathrm{pb},l}}, Act, \longrightarrow_{E,\mathcal{I}_{\mathrm{pb},l}}, E)$$

where:

- $S_{E,\mathcal{I}_{\mathrm{pb},l}}$ *is the least subset of* $\mathcal{G}_{\mathrm{pb},l}$ *such that:*
  - $E \in S_{E,\mathcal{I}_{\mathrm{pb},l}}$;
  - *if* $E_1 \in S_{E,\mathcal{I}_{\mathrm{pb},l}}$ *and* $E_1 \stackrel{a,\tilde{\lambda}}{\longrightarrow}_{\mathcal{I}_{\mathrm{pb},l}} E_2$, *then* $E_2 \in S_{E,\mathcal{I}_{\mathrm{pb},l}}$.
- $\longrightarrow_{E,\mathcal{I}_{\mathrm{pb},l}}$ *is the restriction of* $\longrightarrow_{\mathcal{I}_{\mathrm{pb},l}}$ *(defined in Table 4.3) to* $S_{E,\mathcal{I}_{\mathrm{pb},l}} \times Act \times S_{E,\mathcal{I}_{\mathrm{pb},l}}$. ■

Functions not defined in Table 4.3 are as in Table 3.1. We denote by $\mathcal{E}_{\mathrm{pb},l}$ the set of performance closed terms of $\mathcal{G}_{\mathrm{pb},l}$.

$$\frac{(<a, \tilde{\lambda}>, E') \in Melt(PM(E))}{E \stackrel{a,\tilde{\lambda}}{\longrightarrow}_{\mathcal{I}_{\mathrm{pb},l}} E'}$$

Table 4.3: Rule for $\mathrm{EMPA}_{\mathrm{pb},l}$ integrated semantics

Definition 4.7 *The functional semantics of* $E \in \mathcal{G}_{\mathrm{pb},l}$ *is the LTS*

$$\mathcal{F}_{\mathrm{pb},l}[\![E]\!] = (S_{E,\mathcal{F}_{\mathrm{pb},l}}, AType, \longrightarrow_{E,\mathcal{F}_{\mathrm{pb},l}}, E)$$

where:

- $S_{E,\mathcal{F}_{\mathrm{pb},l}} = S_{E,\mathcal{I}_{\mathrm{pb},l}}$.
- $\longrightarrow_{E,\mathcal{F}_{\mathrm{pb},l}}$ *is the restriction of* $\longrightarrow_{E,\mathcal{I}_{\mathrm{pb},l}}$ *to* $S_{E,\mathcal{F}_{\mathrm{pb},l}} \times AType \times S_{E,\mathcal{F}_{\mathrm{pb},l}}$. ■

Definition 4.8 *The Markovian semantics of* $E \in \mathcal{E}_{\mathrm{pb},l}$ *is the p-LTS*

$$\mathcal{M}_{\mathrm{pb},l}[\![E]\!] = (S_{E,\mathcal{M}_{\mathrm{pb},l}}, \mathbb{R}_{]0,1]}, \longrightarrow_{E,\mathcal{M}_{\mathrm{pb},l}}, P_{E,\mathcal{M}_{\mathrm{pb},l}}, H_{E,\mathcal{M}_{\mathrm{pb},l}})$$

where:

- $S_{E,\mathcal{M}_{\mathrm{pb},l}} = S_{E,\mathcal{I}_{\mathrm{pb},l}}$.

- $\longrightarrow_{E,\mathcal{M}_{\mathrm{pb},l}}$ *is the least subset of* $S_{E,\mathcal{M}_{\mathrm{pb},l}} \times \mathbb{R}_{]0,1]} \times S_{E,\mathcal{M}_{\mathrm{pb},l}}$ *such that* $F \xrightarrow{p}_{E,\mathcal{M}_{\mathrm{pb},l}} F'$ *whenever*

$$p = \sum\{\!|\, w \mid \exists a, l.\, F \xrightarrow{a,\infty_{l,w}}_{E,\mathcal{I}_{\mathrm{pb},l}} F' \,|\!\}\,/\,\sum\{\!|\, w \mid \exists a, l, F''.\, F \xrightarrow{a,\infty_{l,w}}_{E,\mathcal{I}_{\mathrm{pb},l}} F'' \,|\!\}$$

- $P_{E,\mathcal{M}_{\mathrm{pb},l}} : S_{E,\mathcal{M}_{\mathrm{pb},l}} \longrightarrow \mathbb{R}_{[0,1]}$, $P_{E,\mathcal{M}_{\mathrm{pb},l}}(F) = \begin{cases} 1 & \text{if } F \equiv E \\ 0 & \text{if } F \not\equiv E \end{cases}$
- $H_{E,\mathcal{M}_{\mathrm{pb},l}} : S_{E,\mathcal{M}_{\mathrm{pb},l}} \longrightarrow \{v, a\}$, $H_{E,\mathcal{M}_{\mathrm{pb},l}}(F) = \begin{cases} v & \text{if } \exists a, l, w, F'.\, F \xrightarrow{a,\infty_{l,w}}_{E,\mathcal{I}_{\mathrm{pb},l}} F' \\ a & \text{if } \nexists a, l, w, F'.\, F \xrightarrow{a,\infty_{l,w}}_{E,\mathcal{I}_{\mathrm{pb},l}} F' \end{cases}$ ■

Note that the performance model underlying a term of $\mathcal{E}_{\mathrm{pb},l}$ is a HDTMC, if we view the execution of every immediate action as taking one time unit.

Proposition 4.3 *For all* $E \in \mathcal{G}_{\mathrm{pb},l}$, $\mathcal{I}_{\mathrm{pb},l}[\![E]\!]$ *is isomorphic to* $\mathcal{I}[\![E]\!]$. ■

Corollary 4.1 *For all* $E \in \mathcal{G}_{\mathrm{pb},l}$, $\mathcal{F}_{\mathrm{pb},l}[\![E]\!]$ *is isomorphic to* $\mathcal{F}[\![E]\!]$. ■

Corollary 4.2 *For all* $E \in \mathcal{E}_{\mathrm{pb},l}$, $\mathcal{M}_{\mathrm{pb},l}[\![E]\!]$ *is p-isomorphic to* $\mathcal{M}[\![E]\!]$. ■

Unlike probabilistic process algebras appeared in the literature, $\mathrm{EMPA}_{\mathrm{pb},l}$ does not rely on an explicit probabilistic alternative composition operator since the probabilistic information, i.e. weights, is encoded within actions. The idea of using weights instead of probabilities, taken from GSPNs and the probabilistic process algebra WSCCS [184], is due to operational convenience. Besides, in $\mathrm{EMPA}_{\mathrm{pb},l}$ probabilistic and nondeterministic aspects coexist due to the presence of passive actions, thereby causing $\mathrm{EMPA}_{\mathrm{pb},l}$ to be viewed as a possible linguistic counterpart of formal models for randomized distributed computations such as those defined in [173]. By performing more accurate comparisons, we see that $\mathrm{EMPA}_{\mathrm{pb},l}$ differs from PCCS [111], WSCCS [184], and CPP [118] due to the two reasons above (i.e., absence of an explicit probabilistic alternative composition operator and/or presence of pure nondeterminism) as well as the fact that in these calculi the computation proceeds in locksteps. If we consider probabilistic calculi where the computation does not proceed in locksteps, we realize that $\mathrm{EMPA}_{\mathrm{pb},l}$ differs from prACP [7] and PCSP [174] because of the two usual reasons. Furthermore, in prACP an explicit probabilistic parallel composition operator is introduced while the corresponding operator of $\mathrm{EMPA}_{\mathrm{pb},l}$ is implicitly probabilistic thanks to weights. Finally, $\mathrm{EMPA}_{\mathrm{pb},l}$ differs from the probabilistic calculus proposed in [162], where probabilities can be assigned only to internal actions, and from LOTOS-P [132], where an explicit probabilistic alternative composition operator is introduced.

We conclude by observing that, according to the classification of models of probabilistic processes proposed in [71], performance closed terms of $\mathrm{EMPA}_{\mathrm{pb},l}$ represent *generative* models because the relative frequency of performing actions having different types is explicitly described. More accurately, $\mathcal{E}_{\mathrm{pb},l}$ is even finer as it can produce *stratified* models where the intended relative frequencies of actions are preserved in a levelwise fashion in the presence of a restriction.

Example 4.1 Consider an operating system having three processes to be multiprogrammed: the garbage collector gc, user process up_1, and user process up_2. Suppose that each process is given 1/3 of the CPU cycles and that this frequency must be preserved for the garbage collector whenever one of the user processes is denied further access to the machine due to a restriction context. The corresponding $\text{EMPA}_{\text{pb},l}$ term is

$$Sched \triangleq <gc, \infty_{l,1}>.Sched + <\tau, \infty_{l,2}>.(<up_1, \infty_{l,1}>.Sched + <up_2, \infty_{l,1}>.Sched)$$

If we consider $Sched \parallel_{\{up_2\}} \underline{0}$, then the execution probability of the action having type gc is still 1/3, as required. ■

It is however possible to describe also *reactive* models in $\text{EMPA}_{\text{pb},l}$, i.e. models where the choice among actions having different types is nondeterministic. This is achieved by means of the interplay of weighted immediate actions and passive actions, i.e. by means of terms that are not performance closed.

Example 4.2 Suppose that two people want to flip a coin in order to make a decision. Only one of them actually flips the coin and the outcome of the coin toss is a head with probability 1/2, a tail with probability 1/2. This scenario can be modeled as follows:

$$\begin{aligned} GetCoin &\triangleq <person_1, *>.FlipCoin + <person_2, *>.FlipCoin \\ FlipCoin &\triangleq <flip, \infty_{l,1/2}>.Head + <flip, \infty_{l,1/2}>.Tail \end{aligned}$$

The underlying model is reactive because the relative frequency with which the two people get the coin is not specified, i.e. it is left to the environment, while the probability of the outcome of the coin toss is governed by the system. ■

4.4 Exponentially Timed Kernel

The *exponentially timed kernel* EMPA_{et} is the sublanguage of EMPA obtained by considering only the passive actions and the exponentially timed actions. Since each exponentially timed action is given a duration through a probability distribution function, EMPA_{et} is a stochastically timed process algebra. We define below the syntax, the integrated semantics, the functional semantics, and the performance semantics of EMPA_{et} terms.

Definition 4.9 *The set $\mathcal{L}_{\text{et}}$ of process terms of $EMPA_{\text{et}}$ is generated by the following syntax*

$$E ::= \underline{0} \mid <a, \tilde{\lambda}>.E \mid E/L \mid E[\varphi] \mid E + E \mid E \parallel_S E \mid A$$

where $\tilde{\lambda} \in \mathbb{R}_+ \cup \{*\}$ and $L, S \subseteq AType - \{\tau\}$. We denote by $\mathcal{G}_{\text{et}}$ the set of closed and guarded terms of $\mathcal{L}_{\text{et}}$. ■

Definition 4.10 *The integrated semantics of $E \in \mathcal{G}_{\text{et}}$ is the LTS*

$$\mathcal{I}_{\text{et}}[\![E]\!] = (S_{E,\mathcal{I}_{\text{et}}}, Act, \longrightarrow_{E,\mathcal{I}_{\text{et}}}, E)$$

where:

- $S_{E,\mathcal{I}_{\rm et}}$ *is the least subset of* $\mathcal{G}_{\rm et}$ *such that:*
  - $E \in S_{E,\mathcal{I}_{\rm et}}$;
  - *if* $E_1 \in S_{E,\mathcal{I}_{\rm et}}$ *and* $E_1 \stackrel{a,\tilde{\lambda}}{\longrightarrow}_{\mathcal{I}_{\rm et}} E_2$, *then* $E_2 \in S_{E,\mathcal{I}_{\rm et}}$.
- $\longrightarrow_{E,\mathcal{I}_{\rm et}}$ *is the restriction of* $\longrightarrow_{\mathcal{I}_{\rm et}}$ *(defined in Table 4.4) to* $S_{E,\mathcal{I}_{\rm et}} \times Act \times S_{E,\mathcal{I}_{\rm et}}$. ■

Functions not defined in Table 4.4 are as in Table 3.1. We denote by $\mathcal{E}_{\rm et}$ the set of performance closed terms of $\mathcal{G}_{\rm et}$.

$$\frac{(<a,\tilde{\lambda}>,E') \in Melt(PM(E))}{E \stackrel{a,\tilde{\lambda}}{\longrightarrow}_{\mathcal{I}_{\rm et}} E'}$$

Table 4.4: Rule for $\text{EMPA}_{\rm et}$ integrated semantics

Definition 4.11 *The functional semantics of* $E \in \mathcal{G}_{\rm et}$ *is the LTS*

$$\mathcal{F}_{\rm et}[\![E]\!] = (S_{E,\mathcal{F}_{\rm et}}, AType, \longrightarrow_{E,\mathcal{F}_{\rm et}}, E)$$

where:

- $S_{E,\mathcal{F}_{\rm et}} = S_{E,\mathcal{I}_{\rm et}}$.
- $\longrightarrow_{E,\mathcal{F}_{\rm et}}$ *is the restriction of* $\longrightarrow_{E,\mathcal{I}_{\rm et}}$ *to* $S_{E,\mathcal{F}_{\rm et}} \times AType \times S_{E,\mathcal{F}_{\rm et}}$. ■

Definition 4.12 *The Markovian semantics of* $E \in \mathcal{E}_{\rm et}$ *is the p-LTS*

$$\mathcal{M}_{\rm et}[\![E]\!] = (S_{E,\mathcal{M}_{\rm et}}, \mathbb{R}_+, \longrightarrow_{E,\mathcal{M}_{\rm et}}, P_{E,\mathcal{M}_{\rm et}}, H_{E,\mathcal{M}_{\rm et}})$$

where:

- $S_{E,\mathcal{M}_{\rm et}} = S_{E,\mathcal{I}_{\rm et}}$.
- $\longrightarrow_{E,\mathcal{M}_{\rm et}}$ *is the least subset of* $S_{E,\mathcal{M}_{\rm et}} \times \mathbb{R}_+ \times S_{E,\mathcal{M}_{\rm et}}$ *such that* $F \stackrel{\lambda}{\longrightarrow}_{E,\mathcal{M}_{\rm et}} F'$ *whenever*
$$\lambda = \sum \{\!|\, \mu \mid \exists a.\, F \stackrel{a,\mu}{\longrightarrow}_{E,\mathcal{I}_{\rm et}} F' \,|\!\}$$
- $P_{E,\mathcal{M}_{\rm et}} : S_{E,\mathcal{M}_{\rm et}} \longrightarrow \mathbb{R}_{[0,1]}$, $P_{E,\mathcal{M}_{\rm et}}(F) = \begin{cases} 1 & \text{if } F \equiv E \\ 0 & \text{if } F \not\equiv E \end{cases}$

- $H_{E,\mathcal{M}_{\rm et}} : S_{E,\mathcal{M}_{\rm et}} \longrightarrow \{t, a\}$, $H_{E,\mathcal{M}_{\rm et}}(F) = \begin{cases} t & \text{if } \exists a, \lambda, F'.\, F \xrightarrow{a,\lambda}_{E,\mathcal{I}_{\rm et}} F' \\ a & \text{if } \nexists a, \lambda, F'.\, F \xrightarrow{a,\lambda}_{E,\mathcal{I}_{\rm et}} F' \end{cases}$ ■

Note that the performance model underlying a term of $\mathcal{E}_{\rm et}$ is a HCTMC.

Proposition 4.4 *For all $E \in \mathcal{G}_{\rm et}$, $\mathcal{I}_{\rm et}[\![E]\!]$ is isomorphic to $\mathcal{I}[\![E]\!]$.* ■

Corollary 4.3 *For all $E \in \mathcal{G}_{\rm et}$, $\mathcal{F}_{\rm et}[\![E]\!]$ is isomorphic to $\mathcal{F}[\![E]\!]$.* ■

Corollary 4.4 *For all $E \in \mathcal{E}_{\rm et}$, $\mathcal{M}_{\rm et}[\![E]\!]$ is p-isomorphic to $\mathcal{M}[\![E]\!]$.* ■

We would like to point out that $\text{EMPA}_{\rm et}$ closely resembles the stochastically timed process algebras MTIPP [92] and PEPA [100]. As we shall see in Sect. 4.6, the main difference among them is the synchronization discipline on action rates.

4.5 Joining Probabilistic and Exponentially Timed Kernels

When we consider the whole EMPA, we have to cope with the coexistence of immediate and exponentially timed actions. From the performance standpoint, we have that the sojourn time of tangible states is exponentially distributed, while the sojourn time of vanishing states is zero. The performance semantics of a term in $\mathcal{E}$ for which both kinds of transitions coexist can be given as a HCTMC built through an algorithm [27] that eliminates immediate transitions together with the related vanishing states. Due to its generality, such an algorithm can be regarded as an alternative to the technique of the embedded MC, which has been used e.g. to define the MC underlying a GSPN [3].

The first step of the algorithm consists of

1. dropping action types,
2. removing selfloops composed of an immediate transition (hereafter called immediate selfloops for short),
3. changing the rate of each immediate transition into the corresponding execution probability, and
4. determining the initial state probability function.

Formally, from $\mathcal{I}[\![E]\!]$ we obtain the p-LTS $\mathcal{P}_1[\![E]\!] = (S_{E,1}, \mathbb{R}_+, \longrightarrow_{E,1}, P_{E,1}, H_{E,1})$ where: [1]

- $S_{E,1} = S_{E,\mathcal{I}}$.
- Let $PM_1(s) = \mathit{Melt}(\{\!|\, (\tilde{\lambda}, s') \mid s \xrightarrow{a,\tilde{\lambda}}_{E,\mathcal{I}} s' \,|\!\})$ for each $s \in S_{E,1}$. Then $\longrightarrow_{E,1}$ is the least subset of $S_{E,1} \times \mathbb{R}_+ \times S_{E,1}$ such that:

[1] With abuse of notation, we apply function *Melt* to multisets of pairs whose first components are rates instead of actions.

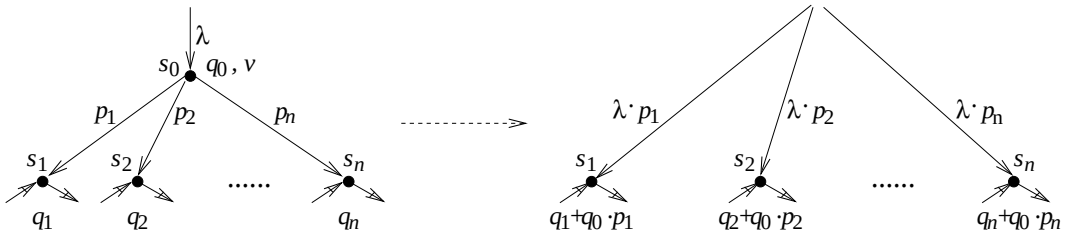


Figure 4.2: Graph reduction rule for the Markovian semantics

- If $(\lambda, s') \in PM_1(s)$, then $s \xrightarrow{\lambda}_{E,1} s'$.
- If there are exactly $m \geq 1$ potential moves (∞_{l,w_j}, s_j), $1 \leq j \leq m$, in $PM_1(s)$ such that $s_j \not\equiv s$, then there are m transitions $s \xrightarrow{w_j/w}_{E,1} s_j$, $1 \leq j \leq m$, where $w = \sum_{j=1}^{m} w_j$.

- $P_{E,1} : S_{E,1} \longrightarrow \mathbb{R}_{[0,1]}$, $P_{E,1}(s) = \begin{cases} 1 & \text{if } s \equiv E \\ 0 & \text{if } s \not\equiv E \end{cases}$

- $H_{E,1} : S_{E,1} \longrightarrow \{v, t, a\}$, $H_{E,1}(s) = \begin{cases} v & \text{if } \exists a, l, w, s'.\, s \xrightarrow{a,\infty_{l,w}}_{E,\mathcal{I}} s' \\ t & \text{if } \exists a, \lambda, s'.\, s \xrightarrow{a,\lambda}_{E,\mathcal{I}} s' \\ a & \text{if } \nexists a, \tilde{\lambda}, s'.\, s \xrightarrow{a,\tilde{\lambda}}_{E,\mathcal{I}} s' \end{cases}$

The k-th step, $k \geq 2$, consists of applying the graph reduction rule in Fig. 4.2 to a given vanishing state $s_0 \in S_{E,k-1}$. With this step we thus consider a fork of immediate transitions that is treated by

1. eliminating the related vanishing state as well as the immediate transitions themselves,
2. splitting the transitions entering the state upstream the fork,
3. removing immediate selfloops created by splitting immediate transitions leaving one of the states downstream the fork and entering the state upstream the fork, and
4. distributing the initial state probability associated with the state upstream the fork among the states downstream the fork.

Formally, if we assume that the vanishing state considered at the k-th step is the one in Fig. 4.2, we build the p-LTS $\mathcal{P}_k[\![E]\!] = (S_{E,k}, \mathbb{R}_+, \longrightarrow_{E,k}, P_{E,k}, H_{E,k})$ where:

- $S_{E,k} = S_{E,k-1} - \{s_0\}$.
- Let $PM_k(s) = Melt(\{\!|\, (\lambda, s') \mid s \xrightarrow{\lambda}_{E,k-1} s' \wedge s' \not\equiv s_0 \,|\!\} \oplus \{\!|\, (\lambda \cdot p_i, s_i) \mid s \xrightarrow{\lambda}_{E,k-1} s_0 \wedge 1 \leq i \leq n \,|\!\})$ for each $s \in S_{E,k}$. Then $\longrightarrow_{E,k}$ is the least subset of $S_{E,k} \times \mathbb{R}_+ \times S_{E,k}$ such that:

– If $H_{E,k-1}(s) = t$, or $H_{E,k-1}(s) = v$ but $s \notin \{s_i \mid 1 \leq i \leq n\}$, and $(\lambda, s') \in PM_k(s)$, then $s \xrightarrow{\lambda}_{E,k} s'$.

– If $H_{E,k-1}(s) = v$, $s \equiv s_i$, and there are exactly $m \geq 1$ potential moves (p_j, s_j), $1 \leq j \leq m$, in $PM_k(s)$ such that $s_j \not\equiv s$, then there are m transitions $s \xrightarrow{p_j/p}_{E,k} s_j$, $1 \leq j \leq m$, where $p = \sum_{j=1}^{m} p_j$.

- $P_{E,k} : S_{E,k} \longrightarrow \mathbb{R}_{[0,1]}$, $P_{E,k}(s) = \begin{cases} P_{E,k-1}(s) & \text{if } s \notin \{s_i \mid 1 \leq i \leq n\} \\ P_{E,k-1}(s) + P_{E,k-1}(s_0) \cdot p_i & \text{if } \exists i \in \{1, \ldots, n\}.\, s \equiv s_i \end{cases}$
- $H_{E,k} : S_{E,k} \longrightarrow \{v, t, a\}$, $H_{E,k}(s) = H_{E,k-1}(s)$.

Definition 4.13 *Let $E \in \mathcal{E}$ be such that $\mathcal{I}[\![E]\!]$ contains both exponentially timed and immediate transitions. The Markovian semantics of E is the p-LTS*

$$\mathcal{M}[\![E]\!] = (S_{E,\mathcal{M}}, \mathbb{R}_+, \longrightarrow_{E,\mathcal{M}}, P_{E,\mathcal{M}}, H_{E,\mathcal{M}})$$

obtained by applying the algorithm above. ■

Theorem 4.1 *Let $E \in \mathcal{E}$ be such that $\mathcal{I}[\![E]\!]$ contains both exponentially timed and immediate transitions.*

(*i*) *For every $k \in \mathbb{N}_+$ and $s \in S_{E,k}$ such that $H_{E,k}(s) = v$, $\sum\{\!|\, p \mid s \xrightarrow{p}_{E,k} s' \,|\!\} = 1$.*

(*ii*) *For every $k \in \mathbb{N}_+$, $\sum_{s \in S_{E,k}} P_{E,k}(s) = 1$.*

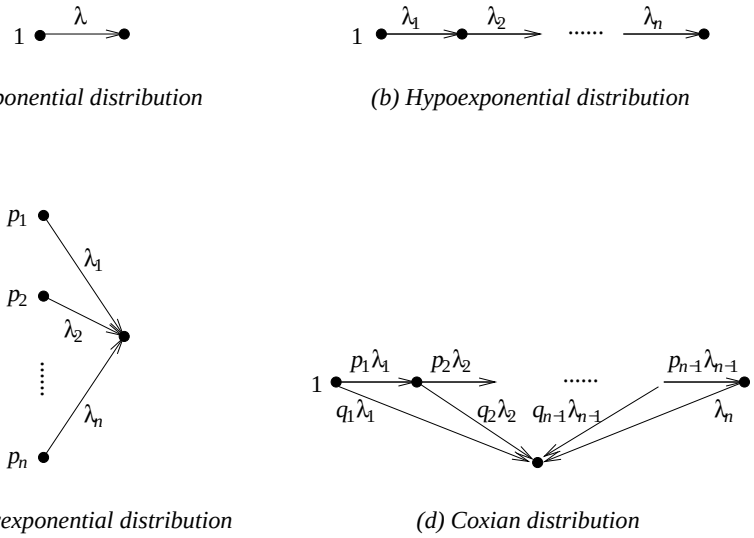
(*iii*) *The elimination of immediate selfloops is correct from the performance viewpoint.*

(*iv*) *$\mathcal{M}[\![E]\!]$ is unique.*

(*v*) *If $\mathcal{I}[\![E]\!]$ has finitely many states, then the algorithm terminates within $O(|\{s \in S_{E,\mathcal{I}} \mid s \text{ vanishing}\}|)$ steps.* ■

We conclude by observing that the coexistence in EMPA of the probabilistic and the exponentially timed kernels allows phase type distributions to be modeled. This makes the limitation to exponential distributions less restrictive, as it becomes possible to describe or approximate many probability distributions frequently occurring in practice.

A *phase type distribution* [141] is a continuous distribution function describing the time to absorption in a finite state HCTMC having exactly one absorbing state. Well known examples of phase type distributions are the exponential distribution, the hypoexponential distribution, the hyperexponential distribution, and the Coxian distribution, which are characterized in terms of time to absorption in a finite state HCTMC with an absorbing state as depicted in Fig. 4.3. Since an absorbing state can be modeled by term $\underline{0}$, the distributions above can be easily represented by means of series parallel combinations of exponentially timed actions as follows:



(a) Exponential distribution *(b) Hypoexponential distribution*

(c) Hyperexponential distribution *(d) Coxian distribution*

Figure 4.3: Phase type distributions

- An exponential distribution with rate $\lambda \in \mathbb{R}_+$ can be modeled by means of term

$$Exp_\lambda \stackrel{\Delta}{=} <a, \lambda>.\underline{0}$$

whose Markovian semantics is p-isomorphic to the HCTMC in Fig. 4.3(a).

- An n-stage hypoexponential distribution with rates $\lambda_i \in \mathbb{R}_+$, $1 \leq i \leq n$, can be modeled by means of the set of inductively defined terms

$$\begin{array}{rcl} Hypoexp_{m,\lambda_1,\ldots,\lambda_m} & \stackrel{\Delta}{=} & <a, \lambda_1>.Hypoexp_{m-1,\lambda_2,\ldots,\lambda_m}, \;\; 2 \leq m \leq n, \\ Hypoexp_{1,\lambda} & \stackrel{\Delta}{=} & Exp_\lambda \end{array}$$

whose Markovian semantics is p-isomorphic to the HCTMC in Fig. 4.3(b).

- An n-stage hyperexponential distribution with rates $\lambda_i \in \mathbb{R}_+$, $1 \leq i \leq n$, and branching probabilities $p_i \in \mathbb{R}_{]0,1]}$, $1 \leq i \leq n$, where $\sum_{i=1}^{n} p_i = 1$, can be modeled by means of the set of inductively defined terms

$$\begin{array}{rcl} Hyperexp_{m,\lambda_1,\ldots,\lambda_m,p_1,\ldots,p_m} & \stackrel{\Delta}{=} & Hyperexp_{m-1,\lambda_1,\ldots,\lambda_{m-1},p_1,\ldots,p_{m-1}} + <a, \infty_{1,p_m}>.Exp_{\lambda_m}, \;\; 2 \leq m \leq n, \\ Hyperexp_{1,\lambda,p} & \stackrel{\Delta}{=} & <a, \infty_{1,p}>.Exp_\lambda \end{array}$$

whose Markovian semantics is p-isomorphic to the HCTMC in Fig. 4.3(c).

- An n-stage Coxian distribution with rates $\lambda_i \in \mathbb{R}_+$, $1 \leq i \leq n$, and branching probabilities $p_i, q_i \in \mathbb{R}_{]0,1]}$ where $p_i + q_i = 1$, $1 \leq i \leq n-1$, can be modeled by means of the set of inductively defined terms

$$\begin{array}{rcl} Cox_{m,\lambda_1,\ldots,\lambda_m,p_1,\ldots,p_{m-1},q_1,\ldots,q_{m-1}} & \stackrel{\Delta}{=} & <a, \lambda_1>.(<a, \infty_{1,q_1}>.\underline{0} + <a, \infty_{1,p_1}>. \\ & & \qquad Cox_{m-1,\lambda_2,\ldots,\lambda_m,p_2,\ldots,p_{m-1},q_2,\ldots,q_{m-1}}), \;\; 2 \leq m \leq n, \\ Cox_{1,\lambda} & \stackrel{\Delta}{=} & Exp_\lambda \end{array}$$

whose Markovian semantics is p-isomorphic to the HCTMC in Fig. 4.3(d).

4.6 The Gluing Role of Passive Actions: Synchronization

The four kernels of EMPA share a common feature: the presence of passive actions. This is due to the synchronization discipline on action rates adopted in EMPA, which causes passive actions to act as a glue for the various kernels.

The main consequence of our synchronization discipline on action rates, which is reminiscent of the composition of I/O automata [125], is that only *master-slaves communications* are directly expressible. The rate of the action resulting from a synchronization is determined by the rate of the only possible active action involved in the synchronization itself. This choice has been made due to its simplicity, since it avoids the need to define the rate of the action deriving from the synchronization of two active actions. Also, this choice has been made due to its modularity. When modeling an n-way synchronization, only the designer of the active component must know the rate of the synchronization, while the other $n-1$ designers can get rid of it by leaving it unspecified through passive actions. Furthermore, possible subsequent changes of the rate affect only one component.

As observed in Sect. 3.4, to compute correctly the rate of a synchronization in case of master-slaves communication according to the bounded capacity assumption, a normalization is required that takes into account the number of alternative or independent passive actions that can be synchronized with the active action at hand.

Example 4.3 Consider a bank having two automated teller machines each providing service at rate μ and suppose that the customer interarrival time is exponentially distributed with rate λ. This scenario can be represented as follows:

$$\begin{aligned} Bank &\triangleq Customers \parallel_{\{a\}} (ATM \parallel_{\emptyset} ATM) \\ Customers &\triangleq <arrive, \lambda>.Customers \\ ATM &\triangleq <arrive, *>.<serve, \mu>.ATM \end{aligned}$$

The normalization operates in such a way that in $\mathcal{I}[\![Bank]\!]$ the two transitions leaving the initial state have rate $\lambda/2$, so the rate λ of the involved active component *Customers* is preserved. Such a normalization is completely transparent to the designer in EMPA and stochastically timed process algebras like PEPA [100] since it is embodied in the semantic rules. On the contrary, in the case of stochastically timed process algebras like MTIPP [92], where passive actions are simulated through exponentially timed actions with rate 1 and the rate of the synchronization of two actions is given by the product of their rates, no normalization is carried out because the bounded capacity assumption is not made. As a consequence, it is responsibility of the designer to define the rates of actions with type *arrive* in both terms *ATM* so that their sum is 1, otherwise the expected underlying HCTMC would not be obtained. The problem of the context dependent

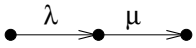
meaning of the rates of MTIPP actions has been partially alleviated in IMTIPP [94] and PMTIPP [165] by means of the introduction of unprioritized unweighted immediate actions and a probabilistic choice operator, respectively, and completely solved in MLOTOS [93] by keeping action execution separated from time passing through an action type prefix operator and a time passing prefix operator. ■

Despite of the fact that master-slaves communications frequently occur in computing systems, it would be useful to be able to describe other kinds of communication, as recognized in [101]. Some of them can be described in other stochastically timed process algebras: for example, *flexible master-slaves communications* in MTIPP [92], where the service requirement is expressed by means of an action whose rate describes a scaling factor instead of a passive action, and *patient communications* in PEPA [100], where the two terms involved in a synchronization work together at the rate of the slowest one. Unlike patient communications, flexible master-slaves communications can be modeled indirectly in EMPA as we shall see in Sect. 4.7.2. Other kinds of communications are listed in [101]: *polite communications*, *impolite communications*, and *timed synchronizations*. Below we show that all of them can be described indirectly with EMPA obtaining the expected underlying HCTMC. This means that the limitation to master-slaves synchronizations, introduced for the sake of simplicity and modularity, is not so restrictive from the modeling viewpoint as it might seem.

Example 4.4 Consider two people speaking at the phone in a polite way: only one of them speaks at a given instant. This scenario can be modeled as follows:

$$\begin{aligned} PPCall &\triangleq Caller \parallel_{\{talk\}} Callee \\ Caller &\triangleq <talk, \lambda>.<talk, *>.\underline{0} \\ Callee &\triangleq <talk, *>.<talk, \mu>.\underline{0} \end{aligned}$$

The underlying HCTMC comprises three states:

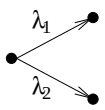


The sojourn times of the first two states are exponentially distributed with rate λ and μ, respectively, and this is consistent with the fact that only one person at a time speaks. ■

Example 4.5 Consider two people speaking at the phone in an impolite way: both of them speak simultaneously and the call finishes as soon as one of them stops speaking (first-to-finish synchronization). This scenario can be modeled as follows:

$$\begin{aligned} IPCall &\triangleq Person_1 \parallel_{\{hang_1, hang_2\}} Person_2 \\ Person_1 &\triangleq <talk, \lambda_1>.<hang_1, \infty_{1,1}>.\underline{0} + <hang_2, *>.\underline{0} \\ Person_2 &\triangleq <talk, \lambda_2>.<hang_2, \infty_{1,1}>.\underline{0} + <hang_1, *>.\underline{0} \end{aligned}$$

The underlying HCTMC comprises three states:

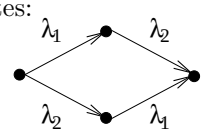


The sojourn time of the first state is exponentially distributed with rate $\lambda_1 + \lambda_2$ and this is consistent with the fact that the two people speak simultaneously and the phone call finishes as soon as one of them stops speaking. ■

Example 4.6 Consider two polite people eating a meal together: neither will start eating until both have been served their meal, then both proceed to eat at their own speed, and neither will leave the table until both have finished (last-to-finish synchronization). This scenario can be modeled as follows:

$$\begin{aligned} Meal &\triangleq (Person_1 \parallel_{\{start,leave\}} Person_2) \\ Person_1 &\triangleq <start, \infty_{1,1}>.<eat, \lambda_1>.<leave, \infty_{1,1}>.\underline{0} \\ Person_2 &\triangleq <start, *>.<eat, \lambda_2>.<leave, *>.\underline{0} \end{aligned}$$

The underlying HCTMC comprises four states:



The sojourn time of the first state is exponentially distributed with rate $\lambda_1 + \lambda_2$ (meaning that both people are eating), while the sojourn times of the second and the third state are exponentially distributed with rate λ_1 and λ_2 respectively (meaning that only one person is eating while the other person has finished and is waiting). This HCTMC describes the phase type distribution that results from the product of two independent exponential distributions, or equivalently the phase type distributed random variable computed as the maximum of two independent exponentially distributed random variables. Defining the duration of a synchronization as the maximum of the durations of the involved actions is the interpretation adopted in MLOTOS [93], where an action type prefix operator and a time passing prefix operator are employed instead of a single action prefix operator. [2] In the scenario above, $Person_i$ would be modeled as $(\lambda_i).eat.\underline{0}$, with eat taking no time, and the synchronization set would be $\{eat\}$. ■

4.7 Describing Queueing Systems

The purpose of this section is to stress that an algebraic formalism like EMPA provides the designer with a compositional linguistic support which is usually lacking in the performance evaluation field, thereby easing the modeling process. As an example, we shall consider a full overview of well known system models such as QSs with memoryless arrival and service processes (some of which have already been described e.g. in [75, 100]), in order to exercise all the expressive capabilities of EMPA.

In Sect. 4.7.1 we model a QS $M/M/1/q$ and we show that its underlying HCTMC coincides with the Markovian semantics of the algebraic description in order to emphasize the correctness of the semantics itself. Afterwards, we complicate the model by allowing for a service rate which depends on the workload

[2] This causes actions to be no longer integrated, hence the modeling is slightly different, and the state space to potentially double in general.

of the system (Sect. 4.7.2), by introducing customers requiring different service times (Sect. 4.7.3) or having different priorities (Sect. 4.7.4), by considering the service request of each customer as being composed of several subrequests to be processed in parallel (Sect. 4.7.5), and by considering a network of QSs instead of a single one where the routing of customers is probabilistic (Sect. 4.7.6). In each of the cases above we shall succeed to get the desired EMPA model from the algebraic model of the QS $M/M/1/q$ thanks to compositionality and the powerful interplay of the kernels.

Other examples based on QSs will be encountered later on. More precisely, in Sect 5.3 we shall relate two different descriptions of the same QS by means of the integrated equivalence. In Sect. 6.3 two approaches to the definition of the integrated net semantics will be compared on a given QS. In Chap. 7 QSs will be used to explain the algebraic treatment of rewards. Finally, in Sect. 8.3 a value passing description of a QS with generally distributed service time will be developed.

4.7.1 Queueing Systems M/M/1/q

In this section we concentrate on QSs $M/M/1/q$, i.e. QSs which have one server and a FIFO queue with $q-1$ seats and serve a population of unboundedly many customers. How can we model a QS $M/M/1/q$ with arrival rate λ and service rate μ? Let a be the action type "a customer arrives at the queue of the service center", d be the action type "a customer is delivered by the queue to the server", and s be the action type "a customer is served by the server". Then the QS under consideration can be modeled with EMPA as follows:

- $QS_{M/M/1/q} \triangleq Arrivals \parallel_{\{a\}} (Queue_0 \parallel_{\{d\}} Server)$
  - $Arrivals \triangleq <a, \lambda>.Arrivals$
  - $Queue_0 \triangleq <a, *>.Queue_1$
    $Queue_h \triangleq <a, *>.Queue_{h+1} + <d, *>.Queue_{h-1}, \quad 0 < h < q-1$
    $Queue_{q-1} \triangleq <d, *>.Queue_{q-2}$
  - $Server \triangleq <d, \infty_{1,1}>.<s, \mu>.Server$

It is worth noting that we have described the whole system as the composition of the arrival process with the composition of the queue and the server (using action types a and d as interfaces among components), then we have separately modeled the arrival process, the queue, and the server. Since the queue is independent of both the arrival rate and the service rate, passive actions have been employed in component *Queue*. As a consequence, if we want to modify the description by changing the arrival rate or the service rate, only component *Arrivals* or *Server* needs to be modified while component *Queue* is not affected. Additionally, the delivery of a customer to the server can be neglected from the performance point of view: this is achieved by means of the immediate action with type d in component *Server*.

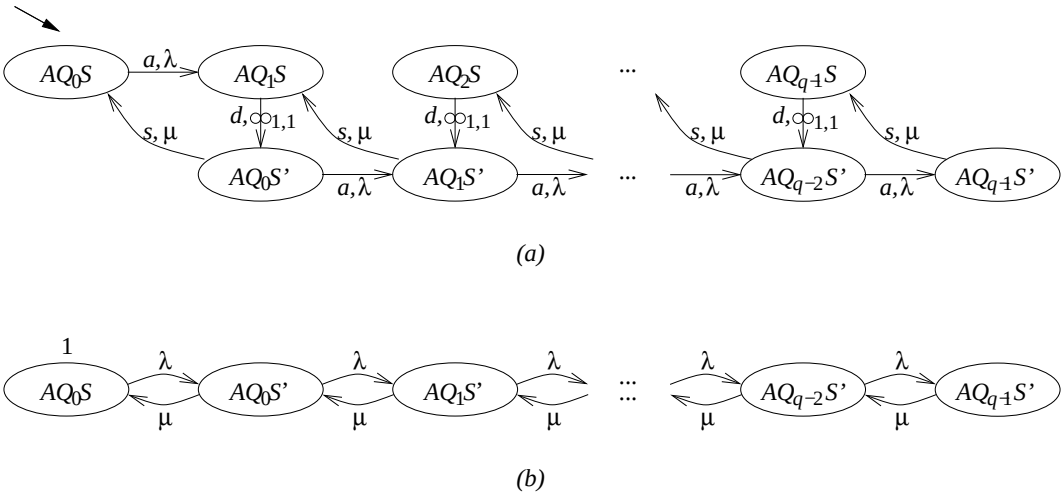


Figure 4.4: Semantic models of $QS_{M/M/1/q}$

We conclude by showing $\mathcal{M}[\![QS_{M/M/1/q}]\!]$ in Fig. 4.4*(b)*, which is obtained from $\mathcal{I}[\![QS_{M/M/1/q}]\!]$ in Fig. 4.4*(a)*, where AQ_hS stands for $Arrivals \parallel_{\{a\}} (Queue_h \parallel_{\{d\}} Server)$, AQ_hS' stands for $Arrivals \parallel_{\{a\}} (Queue_h \parallel_{\{d\}} <s, \mu>.Server)$, and $0 \leq h \leq q-1$. We observe that $\mathcal{M}[\![QS_{M/M/1/q}]\!]$ is p-isomorphic to the HCTMC underlying a queueing system $M/M/1/q$ [115].

4.7.2 Queueing Systems M/M/1/q with Scalable Service Rate

Assume that a QS $M/M/1/q$ with arrival rate λ provides service at a speed depending on the number of customers in the queue. Let us denote by μ the basic service rate and by $sf : \mathbb{N}_+ \longrightarrow \mathbb{R}_+$ the function describing the scaling factor. This QS can be modeled as follows:

- $SSRQS_{M/M/1/q} \triangleq Arrivals \parallel_{\{a\}} (Queue_0 \parallel_{\{d_h \mid 1 \leq h \leq q-1\}} Server)$
  - $Arrivals \triangleq <a, \lambda>.Arrivals$
  - $Queue_0 \triangleq <a, *>.Queue_1$
    $Queue_h \triangleq <a, *>.Queue_{h+1} + <d_h, *>.Queue_{h-1}, \;\; 0 < h < q-1$
    $Queue_{q-1} \triangleq <d_{q-1}, *>.Queue_{q-2}$
  - $Server \triangleq <d_1, \infty_{1,1}>.Server_1 + \ldots + <d_{q-1}, \infty_{1,1}>.Server_{q-1}$
    - $Server_h \triangleq <s, sf(h) \cdot \mu>.Server, \;\; 1 \leq h \leq q-1$

It is worth noting that the structure of $SSRQS_{M/M/1/q}$ is the same as that of $QS_{M/M/1/q}$. Only component *Server* has been significantly modified in order to model the fact that service is provided at a rate depending on the queue length.

4.7.3 Queueing Systems M/M/1/q with Different Service Rates

Assume that a QS $M/M/1/q$ must serve two different types of customers. Both types are characterized by the same arrival rate λ, but red customers require a service rate μ_r whereas black customers require a service rate μ_b. Let η denote a finite string over the alphabet $\{r, s\}$. This QS can be modeled as follows:

- $DSRQS_{M/M/1/q} \triangleq Arrivals \parallel_{\{a_r,a_b\}} (Queue_\epsilon \parallel_{\{d_r,d_b\}} Server)$
  - $Arrivals \triangleq <a_r, \lambda>.Arrivals + <a_b, \lambda>.Arrivals$
  - $Queue_\epsilon \triangleq <a_r, *>.Queue_r + <a_b, *>.Queue_b$
    $Queue_{r\eta} \triangleq <a_r, *>.Queue_{r\eta r} + <a_b, *>.Queue_{r\eta b} + <d_r, *>.Queue_\eta, \;\; 0 \leq |\eta| < q-2$
    $Queue_{b\eta} \triangleq <a_r, *>.Queue_{b\eta r} + <a_b, *>.Queue_{b\eta b} + <d_b, *>.Queue_\eta, \;\; 0 \leq |\eta| < q-2$
    $Queue_{r\eta} \triangleq <d_r, *>.Queue_\eta, \;\; |\eta| = q-2$
    $Queue_{b\eta} \triangleq <d_b, *>.Queue_\eta, \;\; |\eta| = q-2$
  - $Server \triangleq <d_r, \infty_{1,1}>.<s_r, \mu_r>.Server + <d_b, \infty_{1,1}>.<s_b, \mu_b>.Server$

Again, note that the structure of $DSRQS_{M/M/1/q}$ is the same as that of $QS_{M/M/1/q}$. Only the components have been locally modified in order to be able to treat the two types of customers.

4.7.4 Queueing Systems M/M/1/q with Different Priorities

Assume that a QS $M/M/1/q$ must serve two different types of customers characterized by the same arrival rate λ and the same service rate μ. Red customers are assigned a priority level r greater than the priority level b assigned to black customers. There are two cases.

In the first case, we assume that the priority mechanism only affects the queueing discipline, i.e. we assume that possible preemption on the customer being served cannot be exercised. This QS can be modeled as follows:

- $PQS_{M/M/1/q} \triangleq Arrivals \parallel_{\{a_r,a_b\}} (Queue_{0,0} \parallel_{\{d_r,d_b\}} Server)$
  - $Arrivals \triangleq <a_r, \lambda>.Arrivals + <a_b, \lambda>.Arrivals$
  - $Queue_{0,0} \triangleq <a_r, *>.Queue_{1,0} + <a_b, *>.Queue_{0,1}$
    $Queue_{i,0} \triangleq <a_r, *>.Queue_{i+1,0} + <a_b, *>.Queue_{i,1} + <d_r, *>.Queue_{i-1,0}, \;\; 0 < i < q-1$
    $Queue_{0,j} \triangleq <a_r, *>.Queue_{1,j} + <a_b, *>.Queue_{0,j+1} + <d_b, *>.Queue_{0,j-1}, \;\; 0 < j < q-1$
    $Queue_{i,j} \triangleq <a_r, *>.Queue_{i+1,j} + <a_b, *>.Queue_{i,j+1} +$
    $<d_r, *>.Queue_{i-1,j} + <d_b, *>.Queue_{i,j-1}, \;\; 0 < i \wedge 0 < j \wedge i + j < q-1$
    $Queue_{q-1,0} \triangleq <d_r, *>.Queue_{q-2,0}$
    $Queue_{0,q-1} \triangleq <d_b, *>.Queue_{0,q-2}$
    $Queue_{i,j} \triangleq <d_r, *>.Queue_{i-1,j} + <d_b, *>.Queue_{i,j-1}, \;\; 0 < i \wedge 0 < j \wedge i + j = q-1$

– $Server \triangleq <d_r, \infty_{r,1}>.<s, \mu>.Server + <d_b, \infty_{b,1}>.<s, \mu>.Server$

Note that the precedence of red customers over black ones has been enforced by means of the two immediate actions with different priority levels in *Server*.

In the second case, we assume that preemption on a black customer being served can be exercised by red customers. This QS can be modeled as follows (*Arrivals* and $Queue_{i,j}$ are omitted since they stay the same):

- $PPQS_{M/M/1/q} \triangleq Arrivals \parallel_{\{a_r,a_b\}} (Queue_{0,0} \parallel_{\{d_r,d_b\}} Server)$
  - $Server \triangleq <d_r, \infty_{r,1}>.Server_r + <d_b, \infty_{b,1}>.Server_b$
    - $Server_r \triangleq <s, \mu>.Server$
    - $Server_b \triangleq <s, \mu>.Server + <d_r, \infty_{r,1}>.<s, \mu>.Server_b$

Note that, due to the memoryless property of the exponential distribution, there is no difference between the preemptive restart policy (i.e., the preempted customer restarts from the beginning) and the preemptive resume policy (i.e., the preempted customer resumes from the point at which it has been interrupted).

4.7.5 Queueing Systems with Forks and Joins

In this section we want to model a QS composed of n QSs $M/M/1/q$ with the same service rate μ operating in parallel. The service request r of each customer arrived at the QS is split into n subrequests sr_i, $1 \leq i \leq n$, each of which is sent to the corresponding QS $M/M/1/q$. After being served, the n subrequests sr'_i, $1 \leq i \leq n$, are rejoined resulting in r' and the whole request is considered fulfilled. This QS with fork and join can be modeled as follows:

- $FJQS \triangleq In \parallel_{\{r\}} (Fork \parallel_{\{sr_i | 1 \leq i \leq n\}} Center \parallel_{\{sr'_i | 1 \leq i \leq n\}} Join) \parallel_{\{r'\}} Out$
  - $In \triangleq <r, \lambda>.In$
  - $Fork \triangleq F[\varphi_1] \parallel_{\{r\}} F[\varphi_2] \parallel_{\{r\}} \ldots \parallel_{\{r\}} F[\varphi_n]$
    - $F \triangleq <r, *>.<sr, \infty_{1,1}>.F$
    - $\varphi_i = \{(sr, sr_i)\} \cup \{(a, a) \mid a \in AType - \{sr\}\}, \;\; 1 \leq i \leq n$
  - $Center \triangleq C[\varphi'_1] \parallel_\emptyset C[\varphi'_2] \parallel_\emptyset \ldots \parallel_\emptyset C[\varphi'_n]$
    - $C \triangleq Queue_0 \parallel_{\{d\}} Server$
      - $Queue_0 \triangleq <sr, *>.Queue_1$
        $Queue_h \triangleq <sr, *>.Queue_{h+1} + <d, *>.Queue_{h-1}, \;\; 0 < h < q-1$
        $Queue_{q-1} \triangleq <d, *>.Queue_{q-2}$
      - $Server \triangleq <d, \infty_{1,1}>.<s, \mu>.<sr', \infty_{1,1}>.Server$

* $\varphi'_i = \{(sr, sr_i), (sr', sr'_i)\} \cup \{(a, a) \mid a \in AType - \{sr, sr'\}\}, \quad 1 \leq i \leq n$

– $Join \triangleq J_0[\varphi''_1] \parallel_{\{r'\}} J_0[\varphi''_2] \parallel_{\{r'\}} \ldots \parallel_{\{r'\}} J_0[\varphi''_n]$

* $J_0 \triangleq <sr', *>.J_1$

  $J_h \triangleq <sr', *>.J_{h+1} + <r', *>.J_{h-1}, \quad h > 0$

* $\varphi''_i = \{(sr', sr'_i)\} \cup \{(a, a) \mid a \in AType - \{sr'\}\}, \quad 1 \leq i \leq n$

– $Out \triangleq <r', \infty_{1,1}>.Out$

Note that the availability of the functional relabeling operator has allowed us to obtain more compact algebraic representations of components having the same structure but differing for some action types only. Moreover, note the n-way synchronization over r among the fork components and the n-way synchronization over r' among the join components.

4.7.6 Queueing Networks

A queueing network (QN) is composed of a set of QSs linked to each other, where every QS can receive customers from the outside (external sources), from the other QSs in the network, and from itself (feedback paths). The case of open QNs, where interactions with the outside are allowed, is particularly interesting because this kind of QN can be used to describe store and forward packet switched communication networks.

Let us focus our attention on an open QN composed of n QSs $M/M/1/q$ with service rates $\mu_1, \mu_2, \ldots, \mu_n$, respectively. Assume that there are n external sources of customers with rates $\lambda_1, \lambda_2, \ldots, \lambda_n$, respectively. Let us denote by $r_{i,j}$ and $p_{i,j}$ the routing action type and the routing probability, respectively, from QS i to QS j or the outside ($j = n + 1$). This QN can be modeled as follows:

- $QN \triangleq QS_1 \parallel_{R_2} QS_2 \parallel_{R_3} \ldots \parallel_{R_n} QS_n$

  – $QS_i \triangleq Arrivals_i \parallel_{\{a_i\}} (Queue_{i,0} \parallel_{\{d_i, r_{i,i}\}} Server_i), \quad 1 \leq i \leq n$

  * $Arrivals_i \triangleq <a_i, \lambda_i>.Arrivals_i$

  * $Queue_{i,0} \triangleq <a_i, *>.Queue_{i,1} + <r_{1,i}, *>.Queue_{i,1} + \ldots + <r_{n,i}, *>.Queue_{i,1}$

    $Queue_{i,h} \triangleq <a_i, *>.Queue_{i,h+1} + <r_{1,i}, *>.Queue_{i,h+1} + \ldots + <r_{n,i}, *>.Queue_{i,h+1} + <d_i, *>.Queue_{i,h-1}, \quad 0 < h < q - 1$

    $Queue_{i,q-1} \triangleq <d_i, *>.Queue_{i,q-2}$

  * $Server_i \triangleq <d_i, \infty_{1,1}>.<s_i, \mu_i>.Router_i$

    · $Router_i \triangleq <r_{i,1}, \infty_{1,p_{i,1}}>.Server_i + \ldots + <r_{i,n+1}, \infty_{1,p_{i,n+1}}>.Server_i$

  – $R_j = \{r_{i,j}, r_{j,i} \mid 1 \leq i < j\}, \quad 2 \leq j \leq n$

Observe that the description of the QN has been obtained by simply composing the descriptions of the single QSs. Furthermore, routing probabilities have been easily specified by means of the weights of immediate actions.

Chapter 5

Integrated Equivalence for EMPA

In order to finalize the realization of the first phase of the integrated approach of Fig. 1.1, we need to set up an integrated equivalence for EMPA. This equivalence should relate terms describing concurrent systems that are indistinguishable from the point of view of an external observer, i.e. having the same functional and performance properties. The purpose of this chapter is to develop such an integrated equivalence as well as to make sure that it is a congruence in order to allow for compositional reasoning and manipulation [28, 29, 39]. As we shall see, it is possible to come up with a compact and elegant integrated equivalence despite of the considerable expressive power of EMPA. It is worth noting that the integrated equivalence allows for a *qualitative analysis*, i.e. it allows us to investigate whether two terms represent two concurrent systems possessing the same functional and performance characteristics regardless of their actual values. In order to carry out a *quantitative analysis*, i.e. to know whether a functional property holds or to compute a performance measure, we have to study the projected semantic models of (the simplest) one of the two terms.

This chapter is organized as follows. In Sect. 5.1 we introduce a notion of equivalence denoted $\sim_{\rm FP}$ which is defined on the projected semantic models and we show that it is not appropriate because it is not a congruence. In Sect. 5.2 we present a notion of equivalence denoted $\sim_{\rm EMB}$ which is defined on the integrated semantic model by refining the idea of probabilistic bisimulation [117] according to the various kernels of EMPA. In Sect. 5.3 we prove that $\sim_{\rm EMB}$ is a congruence for a large class of terms which allow for a restricted form of nondeterminism. In Sect. 5.4 we demonstrate that $\sim_{\rm EMB}$ is the coarsest congruence contained in $\sim_{\rm FP}$ for a large class of performance closed terms, thereby motivating why it is necessary to define the integrated equivalence on the integrated semantic model. In Sect. 5.5 we give a sound and complete axiomatization of $\sim_{\rm EMB}$ for the set of nonrecursive terms which allow for a restricted form of nondeterminism. Finally, in Sect. 5.6 we develop a $\sim_{\rm EMB}$ checking algorithm, which can also be used to minimize the integrated interleaving semantic model of a term, and we show the relationship between $\sim_{\rm EMB}$ and the notion of ordinary lumping for MCs, thereby motivating why it is correct from the performance viewpoint to define the integrated equivalence in the bisimulation style.

5.1 A Deceptively Integrated Equivalence: $\sim_{\mathrm{FP}}$

Projected semantic models of EMPA terms come equipped with two notions of equivalence: bisimilarity and p-bisimilarity, respectively. Since the purpose of the integrated equivalence is to relate terms which behave the same from the functional and performance standpoint, a natural candidate is the intersection of the two projected bisimulation equivalences mentioned above.

Definition 5.1 *Let $E_1, E_2 \in \mathcal{G}$. We say that E_1 is functionally equivalent to E_2, written $E_1 \sim_{\mathrm{F}} E_2$, if and only if $\mathcal{F}[\![E_1]\!]$ is bisimilar to $\mathcal{F}[\![E_2]\!]$.* ■

Definition 5.2 *Let $E_1, E_2 \in \mathcal{E}$. We say that E_1 is performance equivalent to E_2, written $E_1 \sim_{\mathrm{P}} E_2$, if and only if $\mathcal{M}[\![E_1]\!]$ is p-bisimilar to $\mathcal{M}[\![E_2]\!]$.* ■

Unfortunately, the examples below show that $\sim_{\mathrm{FP}} = \sim_{\mathrm{F}} \cap \sim_{\mathrm{P}}$, defined over $\mathcal{E}$, is not useful as it is not a congruence w.r.t. the alternative and the parallel composition operators.

Example 5.1 Consider terms

$$\begin{aligned} E_1 &\equiv <a, \lambda>.\underline{0} + <b, \mu>.\underline{0} \\ E_2 &\equiv <a, \mu>.\underline{0} + <b, \lambda>.\underline{0} \end{aligned}$$

where $a \neq b$ and $\lambda \neq \mu$. It turns out that $E_1 \sim_{\mathrm{FP}} E_2$ but $E_1 \parallel_{\{b\}} \underline{0} \not\sim_{\mathrm{P}} E_2 \parallel_{\{b\}} \underline{0}$ because the left hand side term can execute only one action with rate λ while the right hand side term can execute only one action with rate μ. Note that the action with type a of E_1 has execution probability $\lambda/(\lambda+\mu)$, while the action with type a of E_2 has execution probability $\mu/(\lambda+\mu)$, but this is not detected by $\sim_{\mathrm{FP}}$. ■

Example 5.2 Consider terms

$$\begin{aligned} E_1 &\equiv <a, \infty_{l_1,w}>.\underline{0} \\ E_2 &\equiv <a, \infty_{l_2,w}>.\underline{0} \end{aligned}$$

with $l_1 < l_2$. It turns out that $E_1 \sim_{\mathrm{FP}} E_2$ but $E_1 + <b, \infty_{l_1,w'}>.\underline{0} \not\sim_{\mathrm{F}} E_2 + <b, \infty_{l_1,w'}>.\underline{0}$ because the left hand side term can execute an action with type b while the right hand side term cannot. ■

Example 5.3 Consider terms

$$\begin{aligned} E_1 &\equiv <a, \infty_{l,w_1}>.\underline{0} \\ E_2 &\equiv <a, \infty_{l,w_2}>.\underline{0} \end{aligned}$$

with $w_1 \neq w_2$. It turns out that $E_1 \sim_{\mathrm{FP}} E_2$ but $E_1 + <b, \infty_{l,w}>.<b, \lambda>.\underline{0} \not\sim_{\mathrm{P}} E_2 + <b, \infty_{l,w}>.<b, \lambda>.\underline{0}$ because the left hand side term reaches the class composed of state $<b, \lambda>.\underline{0}$ with probability $w/(w_1+w)$ while the right hand side term reaches the same class with probability $w/(w_2+w)$. ■

The examples above show that $\sim_{\rm FP}$ is unable to keep track of the link between the functional part and the performance part of the actions. This means that to achieve the congruence property, it is necessary to define the integrated equivalence on the integrated interleaving semantic model, as will be stressed by Thm. 5.4. Incidentally, this is even convenient with respect to $\sim_{\rm FP}$, since it avoids the need to build the two projected semantic models and checking them for bisimilarity and p-bisimilarity, respectively.

5.2 A Really Integrated Equivalence: $\sim_{\rm EMB}$

In order to define a really integrated equivalence in the bisimulation style for our extended Markovian setting ($\sim_{\rm EMB}$), we have to consider the various kernels of EMPA:

- The exponentially timed kernel and the probabilistic kernel should be treated according to the notion of *probabilistic bisimulation* [117]: two equivalent terms have the same aggregated probability to reach the same equivalence class of terms by executing actions of the same type and priority level.

  - In the case of the exponentially timed kernel, the notion of probabilistic bisimulation must be refined by additionally requiring that two equivalent terms have identically distributed sojourn times. For example, if we consider terms $E_1 \equiv <a,\lambda>.F + <a,\mu>.G$ and $E_2 \equiv <a,2\cdot\lambda>.F + <a,2\cdot\mu>.G$, then actions $<a,\lambda>$ and $<a,2\cdot\lambda>$ have the same execution probability $\lambda/(\lambda+\mu)$ and actions $<a,\mu>$ and $<a,2\cdot\mu>$ have the same execution probability $\mu/(\lambda+\mu)$, but the average sojourn time of E_1 is twice the average sojourn time of E_2, hence E_1 cannot be considered equivalent to E_2. Due to the race policy, both average sojourn times and execution probabilities are obtained from rates. Therefore, requiring that two equivalent terms have identically distributed sojourn times and the same aggregated probability to reach the same equivalence class by executing exponentially timed actions of the same type amounts to requiring that two equivalent terms have the same aggregated rate to reach the same equivalence class by executing exponentially timed actions of the same type. As an example, it must hold that

    $$<a,\lambda_1>.E + <a,\lambda_2>.E \sim_{\rm EMB} <a,\lambda_1+\lambda_2>.E$$

    This coincides with the notion of *Markovian bisimulation* [92, 100, 41].

  - In the case of the probabilistic kernel, the notion of probabilistic bisimulation must be restated in terms of weights. As a consequence, two equivalent terms are required to have the same aggregated weight to reach the same equivalence class by executing immediate actions of the same type and priority level. As an example, it must hold that

    $$<a,\infty_{l,w_1}>.E + <a,\infty_{l,w_2}>.E \sim_{\rm EMB} <a,\infty_{l,w_1+w_2}>.E$$

    This coincides with the notion of *direct bisimulation* [184].

- The nondeterministic kernel should be treated by following the classical notion of bisimulation [133]. Thus, two equivalent terms are required to have the same passive actions to reach the same equivalence class, regardless of the actual number of these passive actions. As an example, it must hold that

$$<a,*>.E + <a,*>.E \sim_{\mathrm{EMB}} <a,*>.E$$

- Considerations on the prioritized kernel are deferred to the next section.

All the conditions above that should be met in order for two terms to be considered equivalent can be subsumed by means of the following function expressing the aggregated rate with which a term can reach a class of terms by executing actions of a given type and priority level.

Definition 5.3 *We define partial function* $Rate : \mathcal{G} \times AType \times APLev \times \mathcal{P}(\mathcal{G}) \not\to ARate$ *by* [1]

$$Rate(E,a,l,C) = Aggr\{\!|\, \tilde{\lambda} \mid \exists E' \in C.\, E \xrightarrow{a,\tilde{\lambda}} E' \wedge PL(<a,\tilde{\lambda}>) = l \,|\!\}$$ ■

Definition 5.4 *An equivalence relation* $\mathcal{B} \subseteq \mathcal{G} \times \mathcal{G}$ *is a strong extended Markovian bisimulation (strong EMB) if and only if, whenever* $(E_1, E_2) \in \mathcal{B}$*, then for all* $a \in AType$*,* $l \in APLev$*, and* $C \in \mathcal{G}/\mathcal{B}$

$$Rate(E_1,a,l,C) = Rate(E_2,a,l,C)$$

In this case we say that E_1 *and* E_2 *are strongly extended Markovian bisimilar (strongly EMB).* ■

As an example, the identity relation $Id_{\mathcal{G}}$ over $\mathcal{G}$ is a strong EMB and it is contained in any strong EMB due to reflexive property. We now prove that the largest strong EMB is the union of all the strong EMBs and we define the integrated equivalence as the largest strong EMB.

Lemma 5.1 *Let* $\{\mathcal{B}_i \mid i \in I\}$ *be a family of strong EMBs. Then* $\mathcal{B} = (\cup_{i \in I} \mathcal{B}_i)^+$ *is a strong EMB.* ■

Proposition 5.1 *Let* $\sim_{\mathrm{EMB}}$ *be the union of all the strong EMBs. Then* $\sim_{\mathrm{EMB}}$ *is the largest strong EMB.* ■

Definition 5.5 *We call* $\sim_{\mathrm{EMB}}$ *the strong extended Markovian bisimulation equivalence (strong EMBE). We say that* $E_1, E_2 \in \mathcal{G}$ *are strongly extended Markovian bisimulation equivalent (strongly EMBE) if and only if* $E_1 \sim_{\mathrm{EMB}} E_2$*.* ■

In other words, two terms $E_1, E_2 \in \mathcal{G}$ are strongly EMBE if and only if they are strongly EMB. It is worth noting that, despite of the presence of several different kernels, we have been able to come up with a compact and elegant integrated equivalence in the style of [117].

We conclude this section by exhibiting a necessary condition and a sufficient condition in order for two terms to be strongly EMBE. The necessary condition below is not concerned with equivalence classes, so it is easily checkable. The necessary condition guarantees that strongly EMBE terms carry out actions of the

[1] We let $Aggr\,\emptyset = \bot$, $\tilde{\lambda}\, Aggr \bot = \tilde{\lambda}$, $Split(\bot, p) = \bot$, and $\bot < \tilde{\lambda}$ for all $\tilde{\lambda}$.

same type and priority level at exactly the same aggregated rate: this will be used in Sect. 5.4 and recalled in Sect. 5.6. From this result it follows that, by applying *Aggr* with a ranging over *Atype*, states associated with strongly EMBE terms have identically distributed sojourn times, if tangible, or identical total weights, if vanishing.

Proposition 5.2 *Let $E_1, E_2 \in \mathcal{G}$. If $E_1 \sim_{\rm EMB} E_2$ then for all $a \in AType$ and $l \in APLev$*

$$Rate(E_1, a, l, \mathcal{G}) = Rate(E_2, a, l, \mathcal{G})$$

■

As far as the necessary condition is concerned, it is worth pointing out that, following [133], $\sim_{\rm EMB}$ could be alternatively given a stratified definition as the limit of a sequence of decreasing relations $\sim_{{\rm EMB},n}$ such that $\sim_{{\rm EMB},0} = \mathcal{G} \times \mathcal{G}$ and $E_1 \sim_{{\rm EMB},n+1} E_2$ if and only if for all $a \in AType$, $l \in APLev$, and $C \in \mathcal{G}/\sim_{{\rm EMB},n}$ it holds $Rate(E_1, a, l, C) = Rate(E_2, a, l, C)$. The relation exploited in the necessary condition is thus $\sim_{{\rm EMB},1}$.

The sufficient condition below is based on the notion of strong EMB up to $\sim_{\rm EMB}$, which is helpful to avoid redundancy in strong EMBs. For example, if $\mathcal{B}$ is a strong EMB and $(E_1 \parallel_S E_2, E_3) \in \mathcal{B}$, then also $(E_2 \parallel_S E_1, E_3) \in \mathcal{B}$ although it may be retrieved from the fact that $E_2 \parallel_S E_1 \sim_{\rm EMB} E_1 \parallel_S E_2$. The sufficient condition states that in order for two terms $E_1, E_2 \in \mathcal{G}$ to be strongly EMBE, it suffices to find out a strong EMB up to $\sim_{\rm EMB}$ containing the pair (E_1, E_2). The notion of strong EMB up to $\sim_{\rm EMB}$ will be used in Sect. 5.3.

Definition 5.6 *A relation $\mathcal{B} \subseteq \mathcal{G} \times \mathcal{G}$ is a strong EMB up to $\sim_{\rm EMB}$ if and only if, whenever $(E_1, E_2) \in \mathcal{B}$, then for all $a \in AType$, $l \in APLev$, and $C \in \mathcal{G}/(\mathcal{B} \cup \mathcal{B}^{-1} \cup \sim_{\rm EMB})^+$*

$$Rate(E_1, a, l, C) = Rate(E_2, a, l, C)$$

■

Proposition 5.3 *If $\mathcal{B} \subseteq \mathcal{G} \times \mathcal{G}$ is a strong EMB up to $\sim_{\rm EMB}$ and $(E_1, E_2) \in \mathcal{B}$, then $E_1 \sim_{\rm EMB} E_2$.* ■

5.3 Congruence Result for $\sim_{\rm EMB}$

In a process algebraic framework it is important to investigate whether an equivalence is a congruence, because if this is the case then the equivalence itself can be used to manipulate terms in a compositional way and to compositionally build a minimized semantic model for a given term. As far as $\sim_{\rm EMB}$ is concerned, priority levels and nondeterminism have to be carefully considered in order to get a congruence.

As far as the prioritized kernel is concerned, it might seem useful to be able to write equations like

$$\begin{array}{rcll} <a_1, \infty_{l_1,w_1}>.E_1 + <a_2, \infty_{l_2,w_2}>.E_2 & \sim_{\rm EMB} & <a_2, \infty_{l_2,w_2}>.E_2 & \text{if } l_2 > l_1 \\ <a_1, \lambda>.E_1 + <a_2, \infty_{l,w}>.E_2 & \sim_{\rm EMB} & <a_2, \infty_{l,w}>.E_2 & \end{array}$$

Unfortunately, the applicability of such equations depends on the context. For example, $(<a_1, \lambda>.E_1 + <a_2, \infty_{l,w}>.E_2) \parallel_{\{a_2\}} \underline{0}$ and $(<a_2, \infty_{l,w}>.E_2) \parallel_{\{a_2\}} \underline{0}$ are not equivalent because the former can execute one action while the latter cannot execute actions at all. To solve the problem, we follow the proposal of [6] by

introducing a *priority operator* "$\Theta(\_)$": priority levels are taken to be potential and they become effective only within the scope of the priority operator. We thus consider the language $\mathcal{L}_\Theta$ generated by the following syntax

$$E ::= \underline{0} \mid <a, \tilde{\lambda}>.E \mid E/L \mid E[\varphi] \mid \Theta(E) \mid E + E \mid E \parallel_S E \mid A$$

whose semantic rules are those in Table 3.1 except that the rule in the first part is replaced by

$$\frac{(<a, \tilde{\lambda}>, E') \in Melt(PM(E))}{E \xrightarrow{a,\tilde{\lambda}} E'}$$

and the following rule for the priority operator is introduced in the second part

$$PM(\Theta(E)) = Select(PM(E))$$

It is easily seen that EMPA coincides with the set of terms $\{\Theta(E) \mid E \in \mathcal{L}\}$.

Moreover, only a restricted form of nondeterminism has to be allowed: whenever two alternative or independent passive actions of the same type can be synchronized with the same active action, the derivative terms of the two passive actions must be equivalent.

Definition 5.7 *The set of terms which allow for a restricted form of nondeterminism is defined by*

$$\mathcal{G}_{\Theta,\mathrm{rnd}} = \{E \in \mathcal{G}_\Theta \mid \forall F \in S_{E,\mathcal{I}}.\, RND(F)\}$$

where predicate $RND : \mathcal{G}_\Theta \longrightarrow \{true, false\}$ *is defined by structural induction as follows:*

$$\begin{aligned}
RND(\underline{0}) & \\
RND(<a, \tilde{\lambda}>.E) & \\
RND(E/L) & \Longleftarrow RND(E) \\
RND(E[\varphi]) & \Longleftarrow RND(E) \\
RND(\Theta(E)) & \Longleftarrow RND(E) \\
RND(E_1 + E_2) & \Longleftarrow RND(E_1) \wedge RND(E_2) \\
RND(E_1 \parallel_S E_2) & \Longleftarrow RND(E_1) \wedge RND(E_2) \wedge \\
& \quad (\nexists a \in S.\, E_1 \xrightarrow{a,*} E_1' \wedge E_1 \xrightarrow{a,*} E_1'' \wedge \\
& \qquad E_2 \xrightarrow{a,\tilde{\lambda}} E_2' \wedge \tilde{\lambda} \neq * \wedge \\
& \qquad E_1' \not\sim_{\mathrm{EMB}} E_1'') \wedge \\
& \quad (\nexists a \in S.\, E_1 \xrightarrow{a,\tilde{\lambda}} E_1' \wedge \tilde{\lambda} \neq * \wedge \\
& \qquad E_2 \xrightarrow{a,*} E_2' \wedge E_2 \xrightarrow{a,*} E_2'' \wedge \\
& \qquad E_2' \not\sim_{\mathrm{EMB}} E_2'') \\
RND(A) & \Longleftarrow RND(E) \qquad \text{if } A \triangleq E
\end{aligned}$$

■

Such a restriction is motivated by the fact that $\sim_{\mathrm{EMB}}$ is not in general a congruence for the parallel composition operator. If we take e.g.

$$\begin{aligned} E_1 &\equiv <a,*>.\underline{0} + <a,*>.<b,\mu>.\underline{0} \\ E_2 &\equiv <a,*>.\underline{0} + <a,*>.\underline{0} + <a,*>.<b,\mu>.\underline{0} \\ F_1 &\equiv <a,*>.<a,*>.\underline{0} \parallel_\emptyset <a,*>.<b,\mu>.\underline{0} \\ F_2 &\equiv <a,*>.\underline{0} \parallel_\emptyset <a,*>.\underline{0} \parallel_\emptyset <a,*>.<b,\mu>.\underline{0} \\ G &\equiv <a,\lambda>.\underline{0} \end{aligned}$$

then $E_1 \sim_{\rm EMB} E_2$ and $F_1 \sim_{\rm EMB} F_2$ but $E_1 \parallel_{\{a\}} G \not\sim_{\rm EMB} E_2 \parallel_{\{a\}} G$ and $F_1 \parallel_{\{a\}} G \not\sim_{\rm EMB} F_2 \parallel_{\{a\}} G$. For example $Rate(E_1 \parallel_{\{a\}} G, a, 0, [<b,\mu>.\underline{0}]_{\sim_{\rm EMB}}) = \lambda/2 \neq \lambda/3 = Rate(E_2 \parallel_{\{a\}} G, a, 0, [<b,\mu>.\underline{0}]_{\sim_{\rm EMB}})$. The problem is that the way rate normalization works in the rule for the parallel composition operator in the case of alternative or independent passive potential moves of the same type is not compatible with the idempotence for passive actions of the same type captured by $\sim_{\rm EMB}$. Nevertheless, as we shall see the congruence property holds (hence compositional reasoning is allowed) for a very large class of terms so the expressiveness of EMPA can still be exploited. We also observe that the characterization of $\mathcal{G}_{\Theta,\rm rnd}$ is rather semantical in nature, as it is required that whenever two alternative or independent passive actions of the same type can be synchronized with the same active action, the derivative terms of the two passive actions are related by $\sim_{\rm EMB}$. However, a weaker, easier to check characterization may be set up by requiring that the derivative terms are related by a more syntactical equivalence which approximates $\sim_{\rm EMB}$. As an example, one could adopt structural congruence over terms modulus associativity and commutatitivity of the alternative and parallel composition operators. This would be sufficient to single out a reasonable class of terms, which comprises all the terms in the case studies of Chap. 10 and in the compositional verification of Ex. 5.4.

We now show that $\sim_{\rm EMB}$ is preserved by all the operators and recursive definitions as far as we restrict ourselves to $\mathcal{G}_{\Theta,\rm rnd}$.

Theorem 5.1 *Let $E_1, E_2 \in \mathcal{G}_{\Theta,\rm rnd}$. If $E_1 \sim_{\rm EMB} E_2$ then:*

(*i*) *For all $<a,\tilde{\lambda}> \in Act$, $<a,\tilde{\lambda}>.E_1 \sim_{\rm EMB} <a,\tilde{\lambda}>.E_2$.*

(*ii*) *For all $L \subseteq AType - \{\tau\}$, $E_1/L \sim_{\rm EMB} E_2/L$.*

(*iii*) *For all $\varphi \in ATRFun$, $E_1[\varphi] \sim_{\rm EMB} E_2[\varphi]$.*

(*iv*) $\Theta(E_1) \sim_{\rm EMB} \Theta(E_2)$.

(*v*) *For all $F \in \mathcal{G}_{\Theta,\rm rnd}$, $E_1 + F \sim_{\rm EMB} E_2 + F$ and $F + E_1 \sim_{\rm EMB} F + E_2$.*

(*vi*) *For all $F \in \mathcal{G}_{\Theta,\rm rnd}$ and $S \subseteq AType - \{\tau\}$ such that $E_1 \parallel_S F$, $E_2 \parallel_S F$, $F \parallel_S E_1$, $F \parallel_S E_2 \in \mathcal{G}_{\Theta,\rm rnd}$, $E_1 \parallel_S F \sim_{\rm EMB} E_2 \parallel_S F$ and $F \parallel_S E_1 \sim_{\rm EMB} F \parallel_S E_2$.* ■

In order to prove that $\sim_{\rm EMB}$ is preserved by recursive definitions as well, we extend its definition to terms that are guarded and closed up to constants devoid of defining equation. Note that such constants act as variables.

Definition 5.8 *A constant $A \in Const$ is free w.r.t. Def_Θ if and only if, for all $E \in \mathcal{L}_\Theta$, $A \triangleq E \notin Def_\Theta$.* ■

Definition 5.9 *A term $E \in \mathcal{L}_\Theta$ is partially closed and guarded (pcg) w.r.t. Def_Θ if and only if for each constant $A \in Const_{Def_\Theta}(E)$ either A is free w.r.t. Def_Θ or*

- *A is equipped in Def_Θ with defining equation $A \triangleq E'$, and*
- *there exists $F \in Subst_{Def_\Theta}(E')$ such that, whenever an instance of a constant B nonfree w.r.t. Def_Θ satisfies B st F, then the same instance satisfies B st $<a, \tilde{\lambda}>.G$ st F.* ■

Definition 5.10 *The term $E\langle\langle A := B\rangle\rangle$ obtained from $E \in \mathcal{L}_\Theta$ by replacing each occurrence of $A \in Const$ free w.r.t. Def_Θ with $B \in \mathcal{G}_\Theta$ is defined by induction on the syntactical structure of E as follows:*

$$
\begin{array}{rcl}
\underline{0}\langle\langle A := B\rangle\rangle & \equiv & \underline{0} \\
(<a, \tilde{\lambda}>.E)\langle\langle A := B\rangle\rangle & \equiv & <a, \tilde{\lambda}>.E\langle\langle A := B\rangle\rangle \\
E/L\langle\langle A := B\rangle\rangle & \equiv & E\langle\langle A := B\rangle\rangle/L \\
E[\varphi]\langle\langle A := B\rangle\rangle & \equiv & E\langle\langle A := B\rangle\rangle[\varphi] \\
\Theta(E)\langle\langle A := B\rangle\rangle & \equiv & \Theta(E\langle\langle A := B\rangle\rangle) \\
(E_1 + E_2)\langle\langle A := B\rangle\rangle & \equiv & E_1\langle\langle A := B\rangle\rangle + E_2\langle\langle A := B\rangle\rangle \\
(E_1 \parallel_S E_2)\langle\langle A := B\rangle\rangle & \equiv & E_1\langle\langle A := B\rangle\rangle \parallel_S E_2\langle\langle A := B\rangle\rangle \\
A'\langle\langle A := B\rangle\rangle & \equiv & \left\{ \begin{array}{ll} B & \text{if } A' \equiv A \\ A' & \text{if } A' \not\equiv A \wedge A' \text{ free w.r.t. } Def_\Theta \\ A'' & \text{if } A' \not\equiv A \wedge A' \triangleq E \in Def_\Theta \wedge A'' \triangleq E\langle\langle A := B\rangle\rangle \in Def_\Theta \end{array} \right.
\end{array}
$$
■

Definition 5.11 *Let $E_1, E_2 \in \mathcal{L}_\Theta$ be pcg and suppose that $Const_{Def_\Theta}(E_1) \cup Const_{Def_\Theta}(E_2)$ contains $\{A_i \in Const \mid i \in I\}$ as free constants. We say that E_1 and E_2 are strongly EMBE if and only if, for all sets $\{B_i \in \mathcal{G}_\Theta \mid i \in I\}$ such that $E_1\langle\langle A_i := B_i\rangle\rangle_{i\in I}, E_2\langle\langle A_i := B_i\rangle\rangle_{i\in I} \in \mathcal{G}_\Theta$, it turns out that*

$$E_1\langle\langle A_i := B_i\rangle\rangle_{i\in I} \sim_{\mathrm{EMB}} E_2\langle\langle A_i := B_i\rangle\rangle_{i\in I}$$
■

Theorem 5.2 *Let $E_1, E_2 \in \mathcal{L}_\Theta$ be pcg and suppose that $Const_{Def_\Theta}(E_1) \cup Const_{Def_\Theta}(E_2)$ contains only $A \in Const$ as a free constant. Let $A_1 \triangleq E_1\langle\langle A := A_1\rangle\rangle$, $A_2 \triangleq E_2\langle\langle A := A_2\rangle\rangle \in Def_\Theta$ with $A_1, A_2 \in \mathcal{G}_{\Theta,\mathrm{rnd}}$. If $E_1 \sim_{\mathrm{EMB}} E_2$, then $A_1 \sim_{\mathrm{EMB}} A_2$.* ■

The following example illustrates the benefits of the congruence property.

Example 5.4 Consider a QS $M/M/n/n$ with arrival rate λ and service rate μ. According to the terminology of [187], this QS can be given two different descriptions with EMPA: a *state oriented description*, where the focus is on the state of the set of servers (intended as the number of servers that are currently busy), and a *resource oriented description*, where the servers (i.e. the resource) are modeled separately. The state oriented description is given by

$$\begin{aligned}
QS^{so}_{M/M/n/n} &\triangleq Arrivals \parallel_{\{a\}} Servers_0 \\
Arrivals &\triangleq <a, \lambda>.Arrivals \\
Servers_0 &\triangleq <a, *>.Servers_1 \\
Servers_h &\triangleq <a, *>.Servers_{h+1} + <s, h \cdot \mu>.Servers_{h-1}, \quad 1 \leq h \leq n-1 \\
Servers_n &\triangleq <s, n \cdot \mu>.Servers_{n-1}
\end{aligned}$$

whereas the resource oriented description is given by

$$\begin{aligned}
QS^{ro}_{M/M/n/n} &\triangleq Arrivals \parallel_{\{a\}} Servers \\
Arrivals &\triangleq <a, \lambda>.Arrivals \\
Servers &\triangleq S \parallel_\emptyset S \parallel_\emptyset \ldots \parallel_\emptyset S \\
S &\triangleq <a, *>.<s, \mu>.S
\end{aligned}$$

Since in these representations immediate actions do not occur, we have that $\Theta(QS^{so}_{M/M/n/n}) \sim_{\rm EMB} QS^{so}_{M/M/n/n}$ and $\Theta(QS^{ro}_{M/M/n/n}) \sim_{\rm EMB} QS^{ro}_{M/M/n/n}$. Observed that $QS^{so}_{M/M/n/n}$, $QS^{ro}_{M/M/n/n} \in \mathcal{G}_{\Theta,\rm rnd}$, we can take advantage of the fact that $\sim_{\rm EMB}$ is a congruence: to prove $QS^{so}_{M/M/n/n} \sim_{\rm EMB} QS^{ro}_{M/M/n/n}$, it suffices to prove $Servers_0 \sim_{\rm EMB} Servers$. This is the case because of the strong EMB up to $\sim_{\rm EMB}$ given by the relation made out of the following pairs of terms:

$$\begin{aligned}
&Servers_0, && S \parallel_\emptyset S \parallel_\emptyset \ldots \parallel_\emptyset S \\
&Servers_1, && <s, \mu>.S \parallel_\emptyset S \parallel_\emptyset \ldots \parallel_\emptyset S \\
&Servers_2, && <s, \mu>.S \parallel_\emptyset <s, \mu>.S \parallel_\emptyset \ldots \parallel_\emptyset S \\
&\ldots, && \ldots \\
&Servers_n, && <s, \mu>.S \parallel_\emptyset <s, \mu>.S \parallel_\emptyset \ldots \parallel_\emptyset <s, \mu>.S
\end{aligned}$$

■

5.4 Coarsest Congruence Result for $\sim_{\rm EMB}$

In this section we investigate the relationship between $\sim_{\rm EMB}$ (restricted to $\mathcal{E}$ with modified semantics for priorities) and $\sim_{\rm FP}$ (defined over $\mathcal{E}$). The first result we prove is the inclusion of $\sim_{\rm EMB}$ in $\sim_{\rm FP}$.

Theorem 5.3 *Let $E_1, E_2 \in \mathcal{E}$. If $E_1 \sim_{\rm EMB} E_2$ then $E_1 \sim_{\rm FP} E_2$.* ■

The inclusion $\sim_{\rm EMB} \subseteq \sim_{\rm FP}$ over $\mathcal{E}$ is strict, as one can see by considering the examples below. Additionally, such examples show that $\sim_{\rm EMB}$ cannot abstract from priority levels nor weights of immediate actions; otherwise, the congruence property would no longer hold.

Example 5.5 Consider terms E_1 and E_2 of Ex. 5.1. Then $E_1 \sim_{\rm FP} E_2$ but $E_1 \not\sim_{\rm EMB} E_2$ because $Rate(E_1, a, 0, \mathcal{E}) = \lambda \neq \mu = Rate(E_2, a, 0, \mathcal{E})$ thereby violating the necessary condition in Prop. 5.2(ii). ■

Example 5.6 Consider terms E_1 and E_2 of Ex. 5.2. Then $E_1 \sim_{\rm FP} E_2$ but $E_1 \not\sim_{\rm EMB} E_2$ since $Rate(E_1, a, l_1, \mathcal{E}) = \infty_{l_1,w} \neq \bot = Rate(E_2, a, l_1, \mathcal{E})$ thereby violating the necessary condition in Prop. 5.2(ii). Let $\sim_{\rm EMB'}$ be the equivalence defined by relaxing Def. 5.4 to abstract from the priority levels of immediate actions. Then $E_1 \sim_{\rm EMB'} E_2$ but $\sim_{\rm EMB'}$ would not be a congruence. For example, $\Theta(E_1 + <b, \infty_{l_1,w'}>.\underline{0}) \not\sim_{\rm EMB'} \Theta(E_2 + <b, \infty_{l_1,w'}>.\underline{0})$ and $\Theta(E_1 \parallel_\emptyset <b, \infty_{l_1,w'}>.\underline{0}) \not\sim_{\rm EMB'} \Theta(E_2 \parallel_\emptyset <b, \infty_{l_1,w'}>.\underline{0})$ because the left hand side terms can execute an action with type b while the right hand side terms cannot. ■

Example 5.7 Consider terms E_1 and E_2 of Ex. 5.3. Then $E_1 \sim_{\rm FP} E_2$ but $E_1 \not\sim_{\rm EMB} E_2$ because $Rate(E_1, a, l, \mathcal{E}) = \infty_{l,w_1} \neq \infty_{l,w_2} = Rate(E_2, a, l, \mathcal{E})$ thereby violating the necessary condition in Prop. 5.2(ii). Let $\sim_{\rm EMB'}$ be the equivalence defined by relaxing Def. 5.4 to consider execution probabilities instead of weights for immediate actions (see the notion of *relative bisimulation* proposed in [184]). Then $E_1 \sim_{\rm EMB'} E_2$ but $\sim_{\rm EMB'}$ would not be a congruence. For example, $E_1 + <b, \infty_{l,w}>.\underline{0} \not\sim_{\rm EMB'} E_2 + <b, \infty_{l,w}>.\underline{0}$ and $E_1 \parallel_\emptyset <b, \infty_{l,w}>.\underline{0} \not\sim_{\rm EMB'} E_2 \parallel_\emptyset <b, \infty_{l,w}>.\underline{0}$ because the left hand side terms can execute actions having type b with probability $w/(w_1 + w)$ while the right hand side terms can execute actions having type b with probability $w/(w_2 + w)$. ■

The second result we prove is that $\sim_{\rm EMB}$ is the coarsest congruence contained in $\sim_{\rm FP}$ for the set $\mathcal{E}'$ of terms in $\mathcal{E}$ such that for each state of their integrated interleaving semantic model (i) potential moves of the same type have the same priority level and (ii) observable potential moves have priority levels not less than invisible potential moves. As already anticipated, this result emphasizes the necessity of defining the integrated equivalence on the integrated semantic model in order to get the congruence property.

Theorem 5.4 *Let $E_1, E_2 \in \mathcal{E}'$. Then $E_1 \sim_{\rm EMB} E_2$ if and only if, for all $F \in \mathcal{G}$ and $S \subseteq AType - \{\tau\}$ such that $E_1 + F$, $E_2 + F$, $E_1 \parallel_S F$, $E_2 \parallel_S F \in \mathcal{E}'$, it turns out that $E_1 + F \sim_{\rm FP} E_2 + F$ and $E_1 \parallel_S F \sim_{\rm FP} E_2 \parallel_S F$.* ■

The restriction to $\mathcal{E}'$ is motivated by the fact that if we take e.g.

$$\begin{aligned} E_1 &\equiv <a, \lambda>.\underline{0} + <a, \infty_{1,1}>.\underline{0} \\ E_2 &\equiv <a, \infty_{1,1}>.\underline{0} \end{aligned}$$

or

$$\begin{aligned} E_1 &\equiv <a, \lambda>.\underline{0} + <\tau, \infty_{1,1}>.\underline{0} \\ E_2 &\equiv <\tau, \infty_{1,1}>.\underline{0} \end{aligned}$$

then $E_1 \not\sim_{\rm EMB} E_2$ but E_1 and E_2 cannot be distinguished w.r.t. $\sim_{\rm FP}$ by means of a context based on the alternative composition operator or the parallel composition operator (remember that the two equivalences are defined over two languages having different semantics as far as priorities are concerned). It is worth pointing out that the coarsest congruence result proved above holds in particular for $\rm EMPA_{et}$, hence it holds for other Markovian process algebras such as MTIPP [92] and PEPA [100].

5.5 Axiomatization of $\sim_{\mathrm{EMB}}$

In this section we develop an equational characterization of $\sim_{\mathrm{EMB}}$ for the set $\mathcal{G}_{\Theta,\mathrm{rnd},\mathrm{nrec}}$ of nonrecursive terms of $\mathcal{G}_{\Theta,\mathrm{rnd}}$. Such a characterization is based on the set $\mathcal{A}$ of axioms in Table 5.1 and we denote by $Ded(\mathcal{A})$ the corresponding deductive system. As can be noted, the main difference w.r.t. the axiomatization of the strong bisimulation equivalence for classical process algebras (see Table 2.2) lies in axiom $\mathcal{A}_4$: the idempotence property of the alternative composition operator holds only in the case of passive actions. It is also worth mentioning that, as far as exponentially timed actions are concerned, axiom $\mathcal{A}_4$ is a consequence of the adoption of the race policy and of the property that the minimum of two independent exponentially distributed random variables is still an exponentially distributed random variable whose rate is the sum of the two original rates, while axiom $\mathcal{A}_{13}$ (i.e., the expansion law) is a consequence of the memoryless property of exponential distributions.

Now we prove that $Ded(\mathcal{A})$ is a sound and complete deductive system for $\sim_{\mathrm{EMB}}$ over $\mathcal{G}_{\Theta,\mathrm{rnd},\mathrm{nrec}}$. To accomplish this, we introduce as in [82] the definition of normal form based on the action prefix and the alternative composition operators, then we prove that every term in $\mathcal{G}_{\Theta,\mathrm{rnd},\mathrm{nrec}}$ can be transformed via $Ded(\mathcal{A})$ into an equivalent term in $\mathcal{G}_{\Theta,\mathrm{rnd},\mathrm{nrec}}$ which is in normal form.

Definition 5.12 *$F \in \mathcal{G}_{\Theta,\mathrm{rnd},\mathrm{nrec}}$ is in sum normal form (snf) if and only if $F \equiv \sum_{i\in I} <a_i, \tilde{\lambda}_i>.F_i$ where there is at most one summand for every action type, priority level, and derivative term, and every F_i is itself in snf. We assume $F \equiv \underline{0}$ whenever $I = \emptyset$.* ■

Definition 5.13 *We define function $size : \mathcal{G}_{\Theta,\mathrm{rnd},\mathrm{nrec}} \longrightarrow \mathbb{N}_+$ by structural induction as follows:*

$$\begin{aligned}
size(\underline{0}) &= 1\\
size(<a, \tilde{\lambda}>.E) &= 1 + size(E)\\
size(E/L) &= size(E)\\
size(E[\varphi]) &= size(E)\\
size(\Theta(E)) &= size(E)\\
size(E_1 + E_2) &= \max(size(E_1), size(E_2))\\
size(E_1 \parallel_S E_2) &= size(E_1) + size(E_2)
\end{aligned}$$

■

Lemma 5.2 *For all $E \in \mathcal{G}_{\Theta,\mathrm{rnd},\mathrm{nrec}}$, $size(E) \geq 1$.* ■

Lemma 5.3 *For all axioms $E_1 = E_2$ in Table 5.1, $size(E_1) \geq size(E_2)$.* ■

The two lemmas above are implicitly used in several places during the proof of the result below.

Lemma 5.4 *For all $E \in \mathcal{G}_{\Theta,\mathrm{rnd},\mathrm{nrec}}$ there exists $F \in \mathcal{G}_{\Theta,\mathrm{rnd},\mathrm{nrec}}$ in snf such that $\mathcal{A} \vdash E = F$.* ■

Theorem 5.5 *The deductive system $Ded(\mathcal{A})$ is sound and complete for $\sim_{\mathrm{EMB}}$ over $\mathcal{G}_{\Theta,\mathrm{rnd},\mathrm{nrec}}$.* ■

$$
\begin{array}{lrcll}
(\mathcal{A}_1) & (E_1 + E_2) + E_3 & = & E_1 + (E_2 + E_3) & \\
(\mathcal{A}_2) & E_1 + E_2 & = & E_2 + E_1 & \\
(\mathcal{A}_3) & E + \underline{0} & = & E & \\
(\mathcal{A}_4) & <a, \tilde{\lambda}_1>.E + <a, \tilde{\lambda}_2>.E & = & <a, \tilde{\lambda}_1 \; Aggr \; \tilde{\lambda}_2>.E & \text{if } PL(<a, \tilde{\lambda}_1>) = PL(<a, \tilde{\lambda}_2>) \\
 & & & & \\
(\mathcal{A}_5) & \underline{0}/L & = & \underline{0} & \\
(\mathcal{A}_6) & (<a, \tilde{\lambda}>.E)/L & = & \left\{ \begin{array}{ll} <a, \tilde{\lambda}>.E/L & \text{if } a \notin L \\ <\tau, \tilde{\lambda}>.E/L & \text{if } a \in L \end{array} \right. & \\
(\mathcal{A}_7) & (E_1 + E_2)/L & = & E_1/L + E_2/L & \\
 & & & & \\
(\mathcal{A}_8) & \underline{0}[\varphi] & = & \underline{0} & \\
(\mathcal{A}_9) & (<a, \tilde{\lambda}>.E)[\varphi] & = & <\varphi(a), \tilde{\lambda}>.E[\varphi] & \\
(\mathcal{A}_{10}) & (E_1 + E_2)[\varphi] & = & E_1[\varphi] + E_2[\varphi] & \\
 & & & & \\
(\mathcal{A}_{11}) & \Theta(\underline{0}) & = & \underline{0} & \\
(\mathcal{A}_{12}) & \Theta(\sum_{i \in I} <a_i, \tilde{\lambda}_i>.E_i) & = & \sum_{j \in J} <a_j, \tilde{\lambda}_j>.\Theta(E_j) & \\
 & \text{where } J & = & \{i \in I \mid \tilde{\lambda}_i = * \vee \forall h \in I.\, PL(<a_i, \tilde{\lambda}_i>) \geq PL(<a_h, \tilde{\lambda}_h>)\} & \\
 & & & & \\
(\mathcal{A}_{13}) & \sum_{i \in I_1} <a_i, \tilde{\lambda}_i>.E_i \parallel_S \sum_{i \in I_2} <a_i, \tilde{\lambda}_i>.E_i & = & \sum_{j \in J_1} <a_j, \tilde{\lambda}_j>.(E_j \parallel_S \sum_{i \in I_2} <a_i, \tilde{\lambda}_i>.E_i) + & \\
 & & & \sum_{j \in J_2} <a_j, \tilde{\lambda}_j>.(\sum_{i \in I_1} <a_i, \tilde{\lambda}_i>.E_i \parallel_S E_j) + & \\
 & & & \sum_{k \in K_1} \sum_{h \in H_k} <a_k, \mathit{Split}(\tilde{\lambda}_k, 1/n_k)>.(E_k \parallel_S E_h) + & \\
 & & & \sum_{k \in K_2} \sum_{h \in H_k} <a_k, \mathit{Split}(\tilde{\lambda}_k, 1/n_k)>.(E_h \parallel_S E_k) & \\
 & & & & \\
 & \text{where } I_1 \cap I_2 & = & \emptyset & \\
 & J_1 & = & \{i \in I_1 \mid a_i \notin S\} & \\
 & J_2 & = & \{i \in I_2 \mid a_i \notin S\} & \\
 & K_1 & = & \{k \in I_1 \mid a_k \in S \wedge \exists h \in I_2.\, a_h = a_k \wedge \tilde{\lambda}_h = *\} & \\
 & K_2 & = & \{k \in I_2 \mid a_k \in S \wedge \exists h \in I_1.\, a_h = a_k \wedge \tilde{\lambda}_h = *\} & \\
 & H_k & = & \left\{ \begin{array}{ll} \{h \in I_2 \mid a_h = a_k \wedge \tilde{\lambda}_h = *\} & \text{if } k \in K_1 \\ \{h \in I_1 \mid a_h = a_k \wedge \tilde{\lambda}_h = *\} & \text{if } k \in K_2 \end{array} \right. & \\
 & n_k & = & |H_k| &
\end{array}
$$

Table 5.1: Axioms for $\sim_{\mathrm{EMB}}$

5.6 A $\sim_{\mathrm{EMB}}$ Checking Algorithm

We conclude our study of $\sim_{\mathrm{EMB}}$ by presenting an algorithm that can be used to check whether two finite state terms are equivalent according to $\sim_{\mathrm{EMB}}$ or not and to minimize the integrated interleaving semantic model of a finite state term w.r.t. $\sim_{\mathrm{EMB}}$.

begin
  $Partition := S/\sim_{\mathrm{EMB},1}$;
  $Splitters := Partition$;
  repeat
    $OldPartition := Partition$;
    choose C' **in** $Splitters$;
    $Splitters := Splitters - \{C'\}$;
    for each $a \in AType$ **do**
      for each $l \in APLev$ **do**
        for each $C \in OldPartition$ **do begin**
          $\langle$let $Partition_C$ be the coarsest partition of C such that for every $E_1, E_2 \in C'' \in Partition_C$
            $Rate(E_1, a, l, C') = Rate(E_2, a, l, C')\rangle$;
          if $Partition_C \neq \{C\}$ **then begin**
            $Partition := Partition - \{C\} \cup Partition_C$;
            $Splitters := Splitters - \{C\} \cup Partition_C$
          end
        end
  until $Splitters = \emptyset$
end

Table 5.2: Algorithm to check for $\sim_{\mathrm{EMB}}$

The algorithm, whose structure is shown in Table 5.2, is an adaptation to our framework of the algorithm proposed in [153] to solve the relational coarsest partition problem. Given a LTS with state space S representing the union of the integrated interleaving semantic models of two finite state terms to be checked for $\sim_{\mathrm{EMB}}$ or the integrated interleaving semantic model of a finite state term to be minimized w.r.t. $\sim_{\mathrm{EMB}}$, the algorithm relies on the stratified definition of $\sim_{\mathrm{EMB}}$ by repeatedly refining the current partition of S. According to [153], this algorithm can be implemented in $O(m \log n)$ time and $O(m + n)$ space where n is the number of states and m is the number of transitions.

It is worth noting that a variant of the algorithm in Table 5.2 can be used to compute the coarsest ordinary lumping of the Markovian semantics of a given term, hence allowing for the determination of performance measures by solving a smaller MC which is equivalent to the original one. Below we give the definition of lumped Markovian semantics and we show the relation among $\sim_{\mathrm{EMB}}$ and ordinary lumping.

Definition 5.14 *The lumped Markovian semantics of $E \in \mathcal{E}$ is the p-LTS*

$$\mathcal{M}_l[\![E]\!] = (S_{E,\mathcal{M}_l}, \mathbb{R}_+, \longrightarrow_{E,\mathcal{M}_l}, P_{E,\mathcal{M}_l}, H_{E,\mathcal{M}_l})$$

where:

- $S_{E,\mathcal{M}_l} = Partition$ *where* $Partition$ *is the result of the algorithm in Table 5.3.*
- $\longrightarrow_{E,\mathcal{M}_l}$ *is the least subset of* $S_{E,\mathcal{M}_l} \times \mathbb{R}_+ \times S_{E,\mathcal{M}_l}$ *such that* $C \xrightarrow{\lambda}_{E,\mathcal{M}_l} C'$ *whenever*

  $$\lambda = \sum \{\!|\, \mu \mid \exists s' \in C'.\, s \xrightarrow{\mu}_{E,\mathcal{M}} s' \,|\!\}$$

  with s fixed in C.
- $P_{E,\mathcal{M}_l} : S_{E,\mathcal{M}_l} \longrightarrow \mathbb{R}_{[0,1]}$, $P_{E,\mathcal{M}_l}(C) = \sum_{s \in C} P_{E,\mathcal{M}}(s)$.
- $H_{E,\mathcal{M}_l} : S_{E,\mathcal{M}_l} \longrightarrow \{v, t, a\}$, $H_{E,\mathcal{M}_l}(C) = H_{E,\mathcal{M}}(s)$ *with s fixed in C.* ■

begin
  $Partition := S_{E,\mathcal{M}}/\sim_1$;
  $Splitters := Partition$;
  repeat
    $OldPartition := Partition$;
    choose C' **in** $Splitters$;
    $Splitters := Splitters - \{C'\}$;
    for each $C \in OldPartition$ **do begin**
      ⟨let $Partition_C$ be the coarsest partition of C such that for every $s_1, s_2 \in C'' \in Partition_C$
        $H(s_1) = H(s_2)$ and
        $\sum\{\!|\, \lambda \mid \exists s \in C'.\, s_1 \xrightarrow{\lambda}_{E,\mathcal{M}} s \,|\!\} = \sum\{\!|\, \lambda \mid \exists s \in C' s_2 \xrightarrow{\lambda}_{E,\mathcal{M}} s \,|\!\}$⟩;
      if $Partition_C \neq \{C\}$ **then begin**
        $Partition := Partition - \{C\} \cup Partition_C$;
        $Splitters := Splitters - \{C\} \cup Partition_C$
      end
    end
  until $Splitters = \emptyset$
end

Table 5.3: Algorithm to compute the coarsest ordinary lumping

Theorem 5.6 *Let $E_1, E_2 \in \mathcal{E}$. If $E_1 \sim_{\mathrm{EMB}} E_2$ then $\mathcal{M}_l[\![E_1]\!]$ is p-bisimilar to $\mathcal{M}_l[\![E_2]\!]$.* ■

Corollary 5.1 *Let $E_1, E_2 \in \mathcal{E}$. If $E_1 \sim_{\mathrm{EMB}} E_2$ then $\mathcal{M}_l[\![E_1]\!]$ is p-isomorphic to $\mathcal{M}_l[\![E_2]\!]$.* ■

The corollary above, which is a generalization of a similar result in [100], reveals the adequacy of $\sim_{\mathrm{EMB}}$ from the performance standpoint. If $E_1 \sim_{\mathrm{EMB}} E_2$ then their underlying lumped Markovian models have the same transient and stationary probability distributions, i.e. they describe two concurrent systems having the same performance characteristics.

Chapter 6

Net Semantics for EMPA

In order to implement the second phase of the integrated approach of Fig. 1.1, we must provide each EMPA term with an integrated net semantics accounting for both functional and performance aspects. As explained in Chap. 1, a good candidate for the integrated net model is the class of GSPNs, because they allow performance aspects to be taken into account since the beginning of the design process and are equipped with tool support.

In this chapter we provide two integrated net semantics: a *location oriented* one [25, 27] and a *label oriented* one [19]. In the location oriented approach [62, 145], the GSPN corresponding to a term contains a place for every position of the sequential subterms w.r.t. the static operators. As a consequence, the resulting GSPNs turn out to be safe, i.e. in every reachable marking each net place contains at most one token. All the information about the syntactical structure of terms is encoded within net places so that the relationships among sequential terms are smoothly preserved. This is exploited to define net transitions by means of inductive rules similar to those for the integrated interleaving semantics.

In the label oriented approach [43], instead, each net place corresponds to a sequential subterm independently of its position w.r.t. static operators. As a consequence, instances of the same sequential term occurring in different positions w.r.t. static operators can be represented by multiple tokens within the same net place. This is possible because the syntactical structure of terms w.r.t. static operators is no longer fully retained within net places associated with sequential subterms. Therefore, there is no longer a clear correspondence between the inductive rules for generating net transitions and those for the integrated interleaving semantics.

Both integrated net semantics are shown to be sound w.r.t. the integrated interleaving semantics, which means that the integrated net semantics and the integrated interleaving semantics of a given term represent the same system both from the functional and the performance point of view. To this aim, we adapt the proposal in [145] to our stochastically timed framework by resorting to the following two principles:

- *Functional retrievability principle*: the functional semantics of each term should be retrievable from its integrated net semantics. Such a principle can be formalized by requiring that, for each term,

the functional semantics of its integrated net semantics is isomorphic or bisimilar to its functional semantics.

- *Performance retrievability principle*: the performance semantics of each term should be retrievable from its integrated net semantics. Such a principle can be formalized by requiring that, for each term, the Markovian semantics of its integrated net semantics is p-isomorphic or p-bisimilar to its Markovian semantics.

Additionally, both net semantics are defined in such a way that they allow for the reinterpretation at the algebraic level of structural analysis results. This is achieved by keeping a correspondence between net places and sequential terms and between actions and net transition labels.

This chapter is organized as follows. In Sect. 6.1 we define the integrated location oriented net semantics for EMPA and we prove the retrievability result. In Sect. 6.2 we define the integrated label oriented net semantics for EMPA and we prove the retrievability result. Finally, in Sect. 6.3 we compare the two integrated net semantics.

6.1 Integrated Location Oriented Net Semantics

The integrated net semantics can be defined by resorting to a suitable extension of the semantic rules of Sect. 3.4. Following [62, 145], we associate with a given term a net such that:

1. Net places correspond to the sequential subterms of the given term and its derivatives. This is achieved through a suitable decomposition function, which e.g. assigns two different places $<a, \tilde{\lambda}>.E \parallel_S id$ and $id \parallel_S <b, \tilde{\mu}>.F$ to term $<a, \tilde{\lambda}>.E \parallel_S <b, \tilde{\mu}>.F$.

2. Net transitions are defined by induction on the syntactical structure of the sets of sequential terms by means of rules similar to those for the integrated interleaving semantics.

3. Net markings correspond roughly to the given term and its derivatives.

This approach is called *location oriented* because all the information about the positions of sequential subterms w.r.t. static operators is encoded within places so that the relationships among sequential subterms are preserved. In the example above, $\parallel_S id$ is recorded in the first place while $id \parallel_S$ is recorded in the second place.

The purpose of this section is to define the integrated location oriented net semantics for EMPA. The resulting semantic models turn out to be PGSPNs (with neither inhibitor nor contextual arcs) because of the presence of passive actions in EMPA terms.

This section is organized as follows. In Sect. 6.1.1 we introduce the syntax of net places and we map terms onto places. In Sect. 6.1.2 we inductively define net transitions. In Sect. 6.1.3 we present nets associated

with terms. Finally, in Sect. 6.1.4 we prove the correctness of the integrated location oriented net semantics w.r.t. the integrated interleaving semantics.

6.1.1 Net Places

The first step in the definition of the integrated location oriented net semantics consists of establishing a correspondence between net places and sequential terms.

Definition 6.1 *The set $\mathcal{V}_{\rm loc}$ of places is generated by the following syntax*

$$V ::= \underline{0} \mid <a, \tilde{\lambda}>.E \mid V/L \mid V[\varphi] \mid V + V \mid V \parallel_S id \mid id \parallel_S V \mid A$$

where $L, S \subseteq AType - \{\tau\}$. We use $V, V', \ldots$ as metavariables for $\mathcal{V}_{\rm loc}$ and $Q, Q', \ldots$ as metavariables for $\mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm loc})$. ■

The main difference w.r.t. the syntax of EMPA terms (Def. 3.1) is that the binary operator "$\_ \parallel_S \_$" has been replaced by the two unary operators "$\_ \parallel_S id$" and "$id \parallel_S \_$". This is the means whereby it is possible to express the decomposition of terms into sequential terms mapped onto places.

Definition 6.2 *The decomposition function $dec_{\rm loc} : \mathcal{G} \longrightarrow \mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm loc})$ is defined by induction on the syntactical structure of the terms in $\mathcal{G}$ as follows:*

$$\begin{aligned}
dec_{\rm loc}(\underline{0}) &= \{\!|\, \underline{0} \,|\!\} \\
dec_{\rm loc}(<a, \tilde{\lambda}>.E) &= \{\!|\, <a, \tilde{\lambda}>.E \,|\!\} \\
dec_{\rm loc}(E/L) &= \{\!|\, V/L \mid V \in dec_{\rm loc}(E) \,|\!\} \\
dec_{\rm loc}(E[\varphi]) &= \{\!|\, V[\varphi] \mid V \in dec_{\rm loc}(E) \,|\!\} \\
dec_{\rm loc}(E_1 + E_2) &= \{\!|\, V_1 + V_2 \mid V_1 \in dec_{\rm loc}(E_1) \wedge V_2 \in dec_{\rm loc}(E_2) \,|\!\} \\
dec_{\rm loc}(E_1 \parallel_S E_2) &= \{\!|\, V \parallel_S id \mid V \in dec_{\rm loc}(E_1) \,|\!\} \oplus \{\!|\, id \parallel_S V \mid V \in dec_{\rm loc}(E_2) \,|\!\} \\
dec_{\rm loc}(A) &= dec_{\rm loc}(E) \qquad \text{if } A \stackrel{\Delta}{=} E
\end{aligned}$$

$Q \in \mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm loc})$ is complete if and only if there exists $E \in \mathcal{G}$ such that $dec_{\rm loc}(E) = Q$. ■

The decomposition function is well defined because we consider only closed and guarded terms. It is injective as well, if we identify each constant with the right hand side term of its defining equation, and assigns place sets, rather than multisets, to terms. Note that the decomposition function embeds into places the static operators occurring within terms.

Example 6.1 Consider terms

$$\begin{aligned}
E_1 &\equiv <a, \lambda>.<b, \mu>.\underline{0} + <b, \mu>.<a, \lambda>.\underline{0} \\
E_2 &\equiv <a, \lambda>.\underline{0} \parallel_\emptyset <b, \mu>.\underline{0}
\end{aligned}$$

Then

$$dec_{\rm loc}(E_1) = \{<a,\lambda>.<b,\mu>.\underline{0} + <b,\mu>.<a,\lambda>.\underline{0}\}$$
$$dec_{\rm loc}(E_2) = \{<a,\lambda>.\underline{0} \parallel_\emptyset id, id \parallel_\emptyset <b,\mu>.\underline{0}\}$$

The different number of places in the decomposition of the two terms reflects their different degree of parallelism. ■

6.1.2 Net Transitions

The second step in the definition of the integrated location oriented net semantics consists of introducing an appropriate relation over net places whereby net transitions are constructed. Following the guideline of Sect. 3.4, we define the transition relation $\longrightarrow_{\rm loc}$ as the least subset of $\mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm loc}) \times (AType \times ARate^{\mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm loc})}) \times \mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm loc})$ generated by the inference rule reported in the first part of Table 6.1, which in turn is based on the multiset $PT_{\rm loc}(Q) \in \mathcal{M}u_{\rm fin}((AType \times ARate^{\mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm loc})}) \times \mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm loc}))$ of potential transitions having preset $Q \in \mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm loc})$ defined by structural induction in the second part of Table 6.1.

These rules are strictly related to those in Table 3.1 for the integrated interleaving semantics of EMPA terms. The major differences are listed below and are clarified by the corresponding upcoming examples:

1. There are three rules for the alternative composition operator, instead of one. In the first two rules only a part of the sequential terms needs to have an alternative: such a part is not complete whereas its alternative is. This guarantees that none of the sequential terms in the complete alternative has been previously involved in an execution, so the noncomplete alternative has not been discarded yet due to an action previously executed by a sequential term in the complete alternative (see Ex. 6.2). The first two rules are not necessary in case of guarded alternative compositions of the form $\sum_{i\in I} <a_i, \tilde{\lambda}_i>.E_i$.

2. There are three rules for the parallel composition operator, instead of one. This is a consequence of the distributed notion of state typical of Petri nets (see Ex. 6.3).

3. There are no rules for constants. The treatment of constants has been already embodied in function $dec_{\rm loc}$ (see Def. 6.2), which is used in the rule for the action prefix operator.

4. Function *Select* does not appear because it is unnecessary, due to the inclusion of the race policy and the preselection policy in the net firing rule, and difficult to implement, due to the distributed notion of state (see Ex. 6.4).

5. Rate normalization is carried out by function $norm_{\rm loc} : (Act \times \mathcal{V}'_{\rm loc} \times \mathbb{N}_+) \longrightarrow AType \times ARate^{\mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm loc})}$ defined in the third part of Table 6.1, where $\mathcal{V}'_{\rm loc}$ is generated by the same syntax as $\mathcal{V}_{\rm loc}$ except that $V + V$ is replaced by $V + id$ and $id + V$ and $M_{\rm curr}$ is the current marking. In order to determine the correct rate of transitions deriving from the synchronization of the same active action with several independent or alternative passive actions of the same type, function $norm_{\rm loc}$ considers for each transition three parameters: the basic action, the basic place, and the passive contribution. The basic action is the

$$\frac{(norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V,f),Q') \in melt_{\mathrm{loc},2}(melt_{\mathrm{loc},1}(PT_{\mathrm{loc}}(Q)))}{Q \xrightarrow{norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V,f)}_{\mathrm{loc}} Q'}$$

$PT_{\mathrm{loc}}(\{\!|\, \underline{0} \,|\!\}) = \emptyset$

$PT_{\mathrm{loc}}(\{\!|\, <a,\tilde{\lambda}>.E \,|\!\}) = \{\!|\, (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,<a,\tilde{\lambda}>.E,1), dec_{\mathrm{loc}}(E)) \,|\!\}$

$PT_{\mathrm{loc}}(Q/L) = \{\!|\, (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V/L,f),Q'/L) \mid (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V,f),Q') \in PT_{\mathrm{loc}}(Q) \wedge a \notin L \,|\!\} \oplus$
$\{\!|\, (norm_{\mathrm{loc}}(<\tau,\tilde{\lambda}>,V/L,f),Q'/L) \mid \exists a \in L.\, (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V,f),Q') \in PT_{\mathrm{loc}}(Q) \,|\!\}$

$PT_{\mathrm{loc}}(Q[\varphi]) = \{\!|\, (norm_{\mathrm{loc}}(<\varphi(a),\tilde{\lambda}>,V[\varphi],f),Q'[\varphi]) \mid (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V,f),Q') \in PT_{\mathrm{loc}}(Q) \,|\!\}$

$PT_{\mathrm{loc}}((Q_1 + Q_2) \oplus Q_3) = \{\!|\, (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V+id,f),Q') \mid (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V,f),Q') \in PT_{\mathrm{loc}}(Q_1 \oplus Q_3) \,|\!\}$
if Q_1 not complete $\wedge\, Q_2$ complete $\wedge\, dom(Q_1) \cap dom(Q_3) = \emptyset$

$PT_{\mathrm{loc}}((Q_1 + Q_2) \oplus Q_3) = \{\!|\, (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,id+V,f),Q') \mid (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V,f),Q') \in PT_{\mathrm{loc}}(Q_2 \oplus Q_3) \,|\!\}$
if Q_1 complete $\wedge\, Q_2$ not complete $\wedge\, dom(Q_2) \cap dom(Q_3) = \emptyset$

$PT_{\mathrm{loc}}(Q_1 + Q_2) = \{\!|\, (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V+id,f),Q') \mid (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V,f),Q') \in PT_{\mathrm{loc}}(Q_1) \,|\!\} \oplus$
$\{\!|\, (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,id+V,f),Q') \mid (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V,f),Q') \in PT_{\mathrm{loc}}(Q_2) \,|\!\}$
if Q_1 complete $\wedge\, Q_2$ complete

$PT_{\mathrm{loc}}(Q \parallel_S id) = \{\!|\, (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V \parallel_S id,f),Q' \parallel_S id) \mid (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V,f),Q') \in PT_{\mathrm{loc}}(Q) \wedge a \notin S \,|\!\}$

$PT_{\mathrm{loc}}(id \parallel_S Q) = \{\!|\, (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,id \parallel_S V,f), id \parallel_S Q') \mid (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V,f),Q') \in PT_{\mathrm{loc}}(Q) \wedge a \notin S \,|\!\}$

$PT_{\mathrm{loc}}(Q_1 \parallel_S id \oplus id \parallel_S Q_2) = \{\!|\, (norm_{\mathrm{loc}}(<a,\max(\tilde{\lambda}_1,\tilde{\lambda}_2)>,V,f),Q_1' \parallel_S id \oplus id \parallel_S Q_2') \mid$
$a \in S \wedge \min(\tilde{\lambda}_1,\tilde{\lambda}_2) = * \wedge \exists V_1,V_2,f_1,f_2.$
$(norm_{\mathrm{loc}}(<a,\tilde{\lambda}_1>,V_1,f_1),Q_1') \in PT_{\mathrm{loc}}(Q_1) \wedge$
$(norm_{\mathrm{loc}}(<a,\tilde{\lambda}_2>,V_2,f_2),Q_2') \in PT_{\mathrm{loc}}(Q_2) \wedge$
$V \equiv \begin{cases} V_1 \parallel_S id & \text{if } \tilde{\lambda}_1 = \tilde{\lambda}_2 = * \\ V_1 \parallel_S id & \text{if } \tilde{\lambda}_1 \in \mathbb{R}_+ \cup Inf \\ id \parallel_S V_2 & \text{if } \tilde{\lambda}_2 \in \mathbb{R}_+ \cup Inf \end{cases} \wedge f = \begin{cases} f_1 \cdot f_2 & \text{if } \tilde{\lambda}_1 = \tilde{\lambda}_2 = * \\ f_2 & \text{if } \tilde{\lambda}_1 \in \mathbb{R}_+ \cup Inf \\ f_1 & \text{if } \tilde{\lambda}_2 \in \mathbb{R}_+ \cup Inf \end{cases} \,|\!\}$

$norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V,f) = <a, Split(\tilde{\lambda}, f/\sum\{\!|\, f' \mid Q_1 \xrightarrow{norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V,f')}_{\mathrm{loc}} Q_2 \wedge Q_1 \subseteq M_{\mathrm{curr}} \,|\!\})>$

$melt_{\mathrm{loc},1}(PT) = \{(norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V,f),Q) \mid \exists f' \in \mathbb{N}_+.\, (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V,f'),Q) \in PT \wedge$
$f = \sum\{\!|\, f'' \mid (norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V,f''),Q) \in PT \,|\!\}\}$

$melt_{\mathrm{loc},2}(PT) = \{(norm_{\mathrm{loc}}(<a,\tilde{\lambda}>,V,f),Q) \mid$
$\exists \tilde{\mu},V'.\, (norm_{\mathrm{loc}}(<a,\tilde{\mu}>,V',f),Q) \in PT \wedge$
$\tilde{\lambda} = Aggr\{\!|\, \tilde{\gamma} \mid \exists V''.\, (norm_{\mathrm{loc}}(<a,\tilde{\gamma}>,V'',f),Q) \in PT \wedge PL(<a,\tilde{\gamma}>) = PL(<a,\tilde{\mu}>) \,|\!\} \wedge$
$V = inner_{\_+id}(\{V'' \mid \exists \tilde{\gamma}.\, norm_{\mathrm{loc}}(<a,\tilde{\gamma}>,V'',f),Q) \in PT \wedge PL(<a,\tilde{\gamma}>) = PL(<a,\tilde{\mu}>)\})\}$

Table 6.1: Inductive rules for EMPA integrated location oriented net semantics

action whose rate is involved in the normalization. The basic place is the place contributing with the basic action to the transition (see Ex. 6.5). The passive contribution is the product of the number of alternative passive actions of places contributing to the transition with such actions (see Ex. 6.6). These three parameters are initialized by the rule for the action prefix operator and then modified by the third rule for the parallel composition operator: the second parameter is modified by every rule. The normalizing factor for a given transition is the ratio of its passive contribution to the sum of the passive contributions of the enabled (see Ex. 6.7) transitions having the same basic action and the same basic place as the transition at hand.

6. Potential transitions are merged by functions $melt_{\mathrm{loc},1} : \mathcal{M}u_{\mathrm{fin}}((AType \times ARate^{\mathcal{M}u_{\mathrm{fin}}(\mathcal{V}_{\mathrm{loc}})}) \times \mathcal{M}u_{\mathrm{fin}}(\mathcal{V}_{\mathrm{loc}})) \longrightarrow \mathcal{P}_{\mathrm{fin}}((AType \times ARate^{\mathcal{M}u_{\mathrm{fin}}(\mathcal{V}_{\mathrm{loc}})}) \times \mathcal{M}u_{\mathrm{fin}}(\mathcal{V}_{\mathrm{loc}}))$ and $melt_{\mathrm{loc},2} : \mathcal{P}_{\mathrm{fin}}((AType \times ARate^{\mathcal{M}u_{\mathrm{fin}}(\mathcal{V}_{\mathrm{loc}})}) \times \mathcal{M}u_{\mathrm{fin}}(\mathcal{V}_{\mathrm{loc}})) \longrightarrow \mathcal{P}_{\mathrm{fin}}((AType \times ARate^{\mathcal{M}u_{\mathrm{fin}}(\mathcal{V}_{\mathrm{loc}})}) \times \mathcal{M}u_{\mathrm{fin}}(\mathcal{V}_{\mathrm{loc}}))$ defined in the fourth part of Table 6.1. Function $melt_{\mathrm{loc},1}$ merges the potential transitions having the same basic action, the same basic place, and the same postset by summing their passive contributions (see Ex. 6.6). Function $melt_{\mathrm{loc},2}$ merges the potential transitions having the same basic action type, the same priority level, the same passive contribution, and the same postset by applying operation *Aggr* to their basic action rates. Since the basic places of these potential transitions can differ only due to "$\_ + id$" or "$id + \_$" operators, and since the basic place of the resulting potential transition must be uniquely defined in order for function $norm_{\mathrm{loc}}$ to work correctly, the choice is made through function $inner_{\_+id}$ by taking the basic place having the innermost "$\_ + id$" operator (see Ex. 6.8).

Example 6.2 Consider term

$$E \equiv (<a, \lambda>.\underline{0} \parallel_{\emptyset} <b, \mu>.\underline{0}) + <c, \gamma>.\underline{0}$$

whose decomposition is given by

$$dec_{\mathrm{loc}}(E) = \{\!|\, (<a, \lambda>.\underline{0} \parallel_{\emptyset} id) + <c, \gamma>.\underline{0}, (id \parallel_{\emptyset} <b, \mu>.\underline{0}) + <c, \gamma>.\underline{0} \,|\!\}$$

By applying the rules in Table 6.1, we get the following transitions

$$\begin{array}{rcl}
\{\!|\, (<a, \lambda>.\underline{0} \parallel_{\emptyset} id) + <c, \gamma>.\underline{0} \,|\!\} & \xrightarrow{norm_{\mathrm{loc}}(<a,\lambda>,(<a,\lambda>.\underline{0} \parallel_{\emptyset} id)+id,1)}_{\mathrm{loc}} & \{\!|\, \underline{0} \parallel_{\emptyset} id \,|\!\} \\
\{\!|\, (id \parallel_{\emptyset} <b, \mu>.\underline{0}) + <c, \gamma>.\underline{0} \,|\!\} & \xrightarrow{norm_{\mathrm{loc}}(<b,\mu>,(id \parallel_{\emptyset}<b,\mu>.\underline{0})+id,1)}_{\mathrm{loc}} & \{\!|\, id \parallel_{\emptyset} \underline{0} \,|\!\} \\
dec_{\mathrm{loc}}(E) & \xrightarrow{norm_{\mathrm{loc}}(<c,\gamma>,id+<c,\gamma>.\underline{0},1)}_{\mathrm{loc}} & \{\!|\, \underline{0} \,|\!\}
\end{array}$$

If $dec_{\mathrm{loc}}(E)$ is the current marking then all the transitions above are enabled and firing the first transition results in marking $\{\!|\, \underline{0} \parallel_{\emptyset} id, (id \parallel_{\emptyset} <b, \mu>.\underline{0}) + <c, \gamma>.\underline{0} \,|\!\}$ which cannot be the preset of any transition labeled with action type c, because the execution of either $<a, \lambda>$ or $<b, \mu>$ prevents $<c, \gamma>$ from being executed according to the intended meaning of E. This fact is detected by the rules in Table 6.1, i.e. they generate no transition labeled with action type c for the marking above, since the alternative $id \parallel_{\emptyset} <b, \mu>.\underline{0}$ of $<c, \gamma>.\underline{0}$ is not complete.

To understand the presence of Q_3 in the first two rules for the alternative composition operator, let us now slightly modify term E in the following way

$$E' \equiv (<a, \lambda>.<b, *>.\underline{0} \parallel_{\{b\}} <b, \mu>.\underline{0}) + <c, \gamma>.\underline{0}$$

where

$$dec_{\rm loc}(E') = \{\!| \, (<a, \lambda>.<b, *>.\underline{0} \parallel_{\{b\}} id) + <c, \gamma>.\underline{0}, (id \parallel_{\{b\}} <b, \mu>.\underline{0}) + <c, \gamma>.\underline{0} \, |\!\}$$

By applying the rules in Table 6.1, we get the two transitions

$$\begin{array}{lll} \{\!| \, (<a, \lambda>.<b, *>.\underline{0} \parallel_{\{b\}} id) + <c, \gamma>.\underline{0} \, |\!\} & \xrightarrow{norm_{\rm loc}(<a,\lambda>,(<a,\lambda>.<b,*>.\underline{0} \parallel_{\{b\}} id)+id,1)}_{\rm loc} & \{\!| \, <b, *>.\underline{0} \parallel_{\{b\}} id \, |\!\} \\ dec_{\rm loc}(E') & \xrightarrow{norm_{\rm loc}(<c,\gamma>,id+<c,\gamma>.\underline{0},1)}_{\rm loc} & \{\!| \, \underline{0} \, |\!\} \end{array}$$

If $dec_{\rm loc}(E')$ is the current marking then all the transitions above are enabled and firing the first transition results in marking $\{\!| \, <b, *>.\underline{0} \parallel_{\{b\}} id, (id \parallel_{\{b\}} <b, \mu>.\underline{0}) + <c, \gamma>.\underline{0} \, |\!\}$ which is the preset of the following transition obtained by initially applying the first rule for the alternative composition operator with $Q_3 = \{\!| \, <b, *>.\underline{0} \parallel_{\{b\}} id \, |\!\}$

$$\{\!| \, <b, *>.\underline{0} \parallel_{\{b\}} id, (id \parallel_{\{b\}} <b, \mu>.\underline{0}) + <c, \gamma>.\underline{0} \, |\!\} \xrightarrow{norm_{\rm loc}(<b,\mu>,(id \parallel_{\{b\}} <b,\mu>.\underline{0})+id,1)}_{\rm loc} \{\!| \, \underline{0} \parallel_{\{b\}} id, id \parallel_{\{b\}} \underline{0} \, |\!\}$$

If Q_3 were not taken into account in the first two rules for the alternative composition operator, then the transition above would not be constructed because the third rule for the parallel composition operator is not initially applicable to the given preset. ■

Example 6.3 Consider term

$$E \equiv <a, \tilde{\lambda}>.\underline{0} \parallel_{\emptyset} <b, \tilde{\mu}>.\underline{0}$$

whose decomposition is given by

$$dec_{\rm loc}(E) = \{\!| \, <a, \tilde{\lambda}>.\underline{0} \parallel_{\emptyset} id, id \parallel_{\emptyset} <b, \tilde{\mu}>.\underline{0} \, |\!\}$$

By applying the rules in Table 6.1, we get the two independent transitions

$$\begin{array}{lll} \{\!| \, <a, \tilde{\lambda}>.\underline{0} \parallel_{\emptyset} id \, |\!\} & \xrightarrow{norm_{\rm loc}(<a,\tilde{\lambda}>,<a,\tilde{\lambda}>.\underline{0} \parallel_{\emptyset} id,1)}_{\rm loc} & \{\!| \, \underline{0} \parallel_{\emptyset} id \, |\!\} \\ \{\!| \, id \parallel_{\emptyset} <b, \tilde{\mu}>.\underline{0} \, |\!\} & \xrightarrow{norm_{\rm loc}(<b,\tilde{\mu}>,id \parallel_{\emptyset} <b,\tilde{\mu}>.\underline{0},1)}_{\rm loc} & \{\!| \, id \parallel_{\emptyset} \underline{0} \, |\!\} \end{array}$$

as expected. If we replaced the three rules for the parallel composition operator with a single rule similar to that in Table 3.1, then we would get instead the two alternative transitions

$$\begin{array}{lll} dec_{\rm loc}(E) & \xrightarrow{norm_{\rm loc}(<a,\tilde{\lambda}>,<a,\tilde{\lambda}>.\underline{0} \parallel_{\emptyset} id,1)}_{\rm loc} & \{\!| \, \underline{0} \parallel_{\emptyset} id, id \parallel_{\emptyset} <b, \tilde{\mu}>.\underline{0} \, |\!\} \\ dec_{\rm loc}(E) & \xrightarrow{norm_{\rm loc}(<b,\tilde{\mu}>,id \parallel_{\emptyset} <b,\tilde{\mu}>.\underline{0},1)}_{\rm loc} & \{\!| \, <a, \tilde{\lambda}>.\underline{0} \parallel_{\emptyset} id, id \parallel_{\emptyset} \underline{0} \, |\!\} \end{array}$$

which are not consistent with the fact that the two subterms of E are independent. ■

Example 6.4 Consider term

$$E \equiv (<a, \lambda>.\underline{0} + <c, \infty_{1,1}>.\underline{0}) \parallel_{\{c\}} (<b, \mu>.\underline{0} + <c, *>.\underline{0})$$

whose decomposition comprises places $V_1 \parallel_{\{c\}} id$ and $id \parallel_{\{c\}} V_2$ where

$$\begin{aligned} V_1 &\equiv <a,\lambda>.\underline{0} + <c,\infty_{1,1}>.\underline{0} \\ V_2 &\equiv <b,\mu>.\underline{0} + <c,*>.\underline{0} \end{aligned}$$

By applying the rules in Table 6.1, we get the three transitions

$$\begin{array}{rcl} \{\!|\, V_1 \parallel_{\{c\}} id \,|\!\} & \xrightarrow{norm_{\rm loc}(<a,\lambda>,(<a,\lambda>.\underline{0}+id) \parallel_{\{c\}} id,1)}_{\rm loc} & \{\!|\, \underline{0} \parallel_{\{c\}} id \,|\!\} \\ \{\!|\, id \parallel_{\{c\}} V_2 \,|\!\} & \xrightarrow{norm_{\rm loc}(<b,\mu>,id \parallel_{\{c\}} (<b,\mu>.\underline{0}+id),1)}_{\rm loc} & \{\!|\, id \parallel_{\{c\}} \underline{0} \,|\!\} \\ dec_{\rm loc}(E) & \xrightarrow{norm_{\rm loc}(<c,\infty_{1,1}>,(id+<c,\infty_{1,1}>.\underline{0}) \parallel_{\{c\}} id,1)}_{\rm loc} & \{\!|\, \underline{0} \parallel_{\{c\}} id, id \parallel_{\{c\}} \underline{0} \,|\!\} \end{array}$$

If $dec_{\rm loc}(E)$ is the current marking then all the transitions above are enabled, but the third transition prevents both the first one and the second one from firing. This could not be caught by means of a function similar to *Select* because the three transitions have different presets. ■

Example 6.5 Consider term

$$E \equiv (<a,\lambda>.\underline{0} \parallel_{\{a\}} <a,*>.(\underline{0}+\underline{0})) + (<a,\lambda>.\underline{0} \parallel_{\{a\}} <a,*>.\underline{0})$$

whose decomposition comprises places $(V_1 \parallel_{\{a\}} id) + (V_1 \parallel_{\{a\}} id)$, $(V_1 \parallel_{\{a\}} id) + (id \parallel_{\{a\}} V_3)$, $(id \parallel_{\{a\}} V_2) + (V_1 \parallel_{\{a\}} id)$, and $(id \parallel_{\{a\}} V_2) + (id \parallel_{\{a\}} V_3)$ where

$$\begin{aligned} V_1 &\equiv <a,\lambda>.\underline{0} \\ V_2 &\equiv <a,*>.(\underline{0}+\underline{0}) \\ V_3 &\equiv <a,*>.\underline{0} \end{aligned}$$

By applying the rules in Table 6.1, we get the following two transitions

$$\begin{array}{rcl} dec_{\rm loc}(E) & \xrightarrow{norm_{\rm loc}(<a,\lambda>,(V_1 \parallel_{\{a\}} id)+id,1)}_{\rm loc} & \{\!|\, \underline{0} \parallel_{\{a\}} id, id \parallel_{\{a\}} (\underline{0}+\underline{0}) \,|\!\} \\ dec_{\rm loc}(E) & \xrightarrow{norm_{\rm loc}(<a,\lambda>,id+(V_1 \parallel_{\{a\}} id),1)}_{\rm loc} & \{\!|\, \underline{0} \parallel_{\{a\}} id, id \parallel_{\{a\}} \underline{0} \,|\!\} \end{array}$$

If $dec_{\rm loc}(E)$ is the current marking then both transitions are enabled and the normalizing factor is 1 for both of them, as expected. This example motivates the use of $\mathcal{V}'_{\rm loc}$ instead of $\mathcal{V}_{\rm loc}$ for expressing the basic place: if $\mathcal{V}_{\rm loc}$ were used, then the two transitions above would have the same basic place (beside the same basic action), so they would be given the wrong normalizing factor 1/2 by function $norm_{\rm loc}$. ■

Example 6.6 Consider term

$$E \equiv <a,\lambda>.\underline{0} \parallel_{\{a\}} ((<a,*>.\underline{0} + <a,*>.\underline{0}) \parallel_{\emptyset} <a,*>.\underline{0})$$

whose decomposition comprises places $V_1 \parallel_{\{a\}} id$, $id \parallel_{\{a\}} (V_2 \parallel_{\emptyset} id)$, and $id \parallel_{\{a\}} (id \parallel_{\emptyset} V_3)$ where

$$\begin{aligned} V_1 &\equiv <a,\lambda>.\underline{0} \\ V_2 &\equiv <a,*>.\underline{0} + <a,*>.\underline{0} \\ V_3 &\equiv <a,*>.\underline{0} \end{aligned}$$

By applying the rules in Table 6.1, we get the following two transitions

$$\{\!|\, V_1 \parallel_{\{a\}} id, id \parallel_{\{a\}} (V_2 \parallel_\emptyset id) \,|\!\} \xrightarrow{norm_{\rm loc}(<a,\lambda>,V_1 \parallel_{\{a\}} id,2)}_{\rm loc} \{\!|\, \underline{0} \parallel_{\{a\}} id, id \parallel_{\{a\}} (\underline{0} \parallel_\emptyset id) \,|\!\}$$
$$\{\!|\, V_1 \parallel_{\{a\}} id, id \parallel_{\{a\}} (id \parallel_\emptyset V_3) \,|\!\} \xrightarrow{norm_{\rm loc}(<a,\lambda>,V_1 \parallel_{\{a\}} id,1)}_{\rm loc} \{\!|\, \underline{0} \parallel_{\{a\}} id, id \parallel_{\{a\}} (id \parallel_\emptyset \underline{0}) \,|\!\}$$

where value 2 for the passive contribution of the first transition is determined by function $melt_{\rm loc,1}$. If $dec_{\rm loc}(E)$ is the current marking then both transitions are enabled and the normalizing factor is 2/3 for the first transition and 1/3 for the second transition, as expected. This example motivates the use of passive contributions: if the normalizing factor were computed as the inverse of the number of enabled transitions having the same basic action and the same basic place as the transition at hand, then we would obtain the wrong normalizing factor 1/2 for the two transitions above. ■

Example 6.7 Consider term

$$E \equiv <a,\lambda>.\underline{0} \parallel_{\{a\}} (<a,*>.<a,*>.\underline{0} \parallel_\emptyset <a,*>.\underline{0})$$

whose decomposition comprises places $V_1 \parallel_{\{a\}} id$, $id \parallel_{\{a\}} (V_2 \parallel_\emptyset id)$ and $id \parallel_{\{a\}} (id \parallel_\emptyset V_3)$ where

$$\begin{aligned} V_1 &\equiv <a,\lambda>.\underline{0} \\ V_2 &\equiv <a,*>.<a,*>.\underline{0} \\ V_3 &\equiv <a,*>.\underline{0} \end{aligned}$$

By applying the rules in Table 6.1, we get the following two transitions

$$\{\!|\, V_1 \parallel_{\{a\}} id, id \parallel_{\{a\}} (V_2 \parallel_\emptyset id) \,|\!\} \xrightarrow{norm_{\rm loc}(<a,\lambda>,V_1 \parallel_{\{a\}} id,1)}_{\rm loc} \{\!|\, \underline{0} \parallel_{\{a\}} id, id \parallel_{\{a\}} (V_3 \parallel_\emptyset id) \,|\!\}$$
$$\{\!|\, V_1 \parallel_{\{a\}} id, id \parallel_{\{a\}} (id \parallel_\emptyset V_3) \,|\!\} \xrightarrow{norm_{\rm loc}(<a,\lambda>,V_1 \parallel_{\{a\}} id,1)}_{\rm loc} \{\!|\, \underline{0} \parallel_{\{a\}} id, id \parallel_{\{a\}} (id \parallel_\emptyset \underline{0}) \,|\!\}$$

as well as the following one as a consequence of the first transition above

$$\{\!|\, V_1 \parallel_{\{a\}} id, id \parallel_{\{a\}} (V_3 \parallel_\emptyset id) \,|\!\} \xrightarrow{norm_{\rm loc}(<a,\lambda>,V_1 \parallel_{\{a\}} id,1)}_{\rm loc} \{\!|\, \underline{0} \parallel_{\{a\}} id, id \parallel_{\{a\}} (\underline{0} \parallel_\emptyset id) \,|\!\}$$

If $dec_{\rm loc}(E)$ is the current marking then only the first and the second transitions are enabled and their normalizing factor computed by $norm_{\rm loc}$ is 1/2 as expected. This example motivates the consideration of enabled transitions when computing the normalizing factor: if also the third transition above were taken into account though not enabled at $dec_{\rm loc}(E)$, then we would obtain the wrong normalizing factor 1/3 for the first two transitions. ■

Example 6.8 Consider term

$$E \equiv (<a,\lambda>.\underline{0} + <a,\lambda>.\underline{0}) \parallel_{\{a\}} (<a,*>.\underline{0} \parallel_\emptyset <a,*>.\underline{0})$$

whose decomposition comprises places $V_1 \parallel_{\{a\}} id$, $id \parallel_{\{a\}} (V_2 \parallel_\emptyset id)$, and $id \parallel_{\{a\}} (id \parallel_\emptyset V_2)$ where

$$\begin{aligned} V_1 &\equiv <a,\lambda>.\underline{0} + <a,\lambda>.\underline{0} \\ V_2 &\equiv <a,*>.\underline{0} \end{aligned}$$

By applying the rules in Table 6.1, we get the following two transitions

$$\{\!|\, V_1 \,\|_{\{a\}}\, id, id \,\|_{\{a\}} (V_2 \,\|_\emptyset\, id) \,|\!\}\ \xrightarrow{norm_{\rm loc}(<a,2\cdot\lambda>,(<a,\lambda>.\underline{0}+id)\,\|_{\{a\}}\, id),1)}_{\rm loc}\ \{\!|\, \underline{0} \,\|_{\{a\}}\, id, id \,\|_{\{a\}} (\underline{0} \,\|_\emptyset\, id) \,|\!\}$$

$$\{\!|\, V_1 \,\|_{\{a\}}\, id, id \,\|_{\{a\}} (id \,\|_\emptyset\, V_2) \,|\!\}\ \xrightarrow{norm_{\rm loc}(<a,2\cdot\lambda>,(<a,\lambda>.\underline{0}+id)\,\|_{\{a\}}\, id),1)}_{\rm loc}\ \{\!|\, \underline{0} \,\|_{\{a\}}\, id, id \,\|_{\{a\}} (id \,\|_\emptyset\, \underline{0}) \,|\!\}$$

each of which is obtained by applying function $melt_{\rm loc,2}$ to two potential transitions having as a basic place $(<a,\lambda>.\underline{0} + id) \,\|_{\{a\}}\, id$ and $(id + <a,\lambda>.\underline{0}) \,\|_{\{a\}}\, id$, respectively. If $dec_{\rm loc}(E)$ is the current marking then both transitions are enabled and the normalizing factor is $1/2$ for both transitions, as expected. This example motivates that fact that, if two potential transitions having different basic places are merged by function $melt_{\rm loc,2}$, the basic place of the resulting potential transition must be uniquely identified: if the two transitions above had as a basic place $(<a,\lambda>.\underline{0} + id) \,\|_{\{a\}}\, id$ and $(id + <a,\lambda>.\underline{0}) \,\|_{\{a\}}\, id$, respectively, then we would obtain the wrong normalizing factor 1 for them. ■

6.1.3 Nets Associated with Terms

The third step in the definition of the integrated location oriented net semantics consists of associating with each term an appropriate PGSPN (with neither inhibitor nor contextual arcs) by exploiting the previous two steps.

Definition 6.3 *The integrated location oriented net semantics of $E \in \mathcal{G}$ is the PGSPN*

$$\mathcal{N}_{\rm loc}[\![E]\!] = (P_{E,\mathcal{N}_{\rm loc}}, AType \times ARate^{\mathcal{M}u_{\rm fin}(P_{E,\mathcal{N}_{\rm loc}})}, \longrightarrow_{E,\mathcal{N}_{\rm loc}}, dec_{\rm loc}(E), L, W)$$

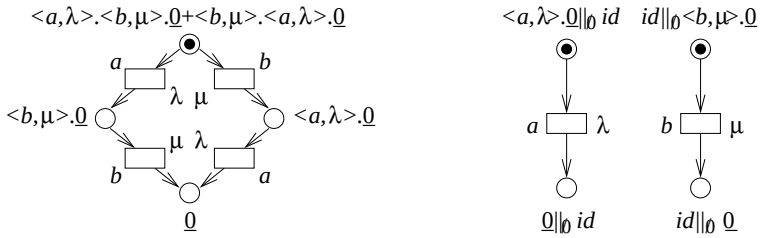
where:

- $P_{E,\mathcal{N}_{\rm loc}}$ *is the least subset of* $\mathcal{V}_{\rm loc}$ *such that:*
  - $dom(dec_{\rm loc}(E)) \subseteq P_{E,\mathcal{N}_{\rm loc}}$;
  - *if* $dom(Q_1) \subseteq P_{E,\mathcal{N}_{\rm loc}}$ *and* $Q_1 \xrightarrow{norm_{\rm loc}(<a,\tilde{\lambda}>,V,f)}_{\rm loc} Q_2$, *then* $dom(Q_2) \subseteq P_{E,\mathcal{N}_{\rm loc}}$.
- $\longrightarrow_{E,\mathcal{N}_{\rm loc}}$ *is the restriction of* $\longrightarrow_{\rm loc}$ *to* $\mathcal{M}u_{\rm fin}(P_{E,\mathcal{N}_{\rm loc}}) \times (AType \times ARate^{\mathcal{M}u_{\rm fin}(P_{E,\mathcal{N}_{\rm loc}})}) \times \mathcal{M}u_{\rm fin}(P_{E,\mathcal{N}_{\rm loc}})$.
- $L : \longrightarrow_{E,\mathcal{N}_{\rm loc}} \longrightarrow APLev$ *such that:*
  - $L(Q_1, norm_{\rm loc}(<a,*>,V,f), Q_2) = -1$;
  - $L(Q_1, norm_{\rm loc}(<a,\lambda>,V,f), Q_2) = 0$;
  - $L(Q_1, norm_{\rm loc}(<a,\infty_{l,w}>,V,f), Q_2) = l$.
- $W : \longrightarrow_{E,\mathcal{N}_{\rm loc}} \longrightarrow \{*\} \cup \mathbb{R}_+^{\mathcal{M}u_{\rm fin}(P_{E,\mathcal{N}_{\rm loc}})}$ *such that:*
  - $W(Q_1, norm_{\rm loc}(<a,*>,V,f), Q_2) = *$;
  - $W(Q_1, norm_{\rm loc}(<a,\lambda>,V,f), Q_2) = \lambda'$ *if* $norm_{\rm loc}(<a,\lambda>,V,f) = <a,\lambda'>$;

– $W(Q_1, norm_{\rm loc}(<a, \infty_{l,w}>, V, f), Q_2) = w'$ *if* $norm_{\rm loc}(<a, \infty_{l,w}>, V, f) = <a, \infty_{l,w'}>$. ■

We observe that $\mathcal{N}_{\rm loc}[\![E]\!]$ is a GSPN whenever $E \in \mathcal{E}$.

Example 6.9 Consider terms E_1 and E_2 introduced in Ex. 6.1. We show below $\mathcal{N}_{\rm loc}[\![E_1]\!]$ and $\mathcal{N}_{\rm loc}[\![E_2]\!]$:



Since the two models above for E_1 and E_2 are different, we are no longer in an interleaving framework. ■

We conclude with two properties of the nets obtained by applying the integrated location oriented net semantics which can be demonstrated with proofs similar to those provided in [145] Thm. 3.4.4 and 3.5.5, respectively.

Theorem 6.1 *Let* $E \in \mathcal{G}$.

(*i*) $\mathcal{N}_{\rm loc}[\![E]\!]$ *is safe, i.e. every marking reachable from the initial one is a set.*

(*ii*) *If there are no static operators within the scope of the definition of any recursive constant occurring in* E*, then* $\mathcal{N}_{\rm loc}[\![E]\!]$ *is finite.* ■

6.1.4 Retrievability Result

In this section we assess the soundness of the integrated location oriented net semantics w.r.t. the integrated interleaving semantics.

Theorem 6.2 *Let* $E \in \mathcal{G}$. *Then* $\mathcal{RG}[\![\mathcal{N}_{\rm loc}[\![E]\!]]\!]$ *is isomorphic to* $\mathcal{I}[\![E]\!]$. ■

Corollary 6.1 *Let* $E \in \mathcal{G}$. *Then* $\mathcal{F}[\![\mathcal{N}_{\rm loc}[\![E]\!]]\!]$ *is isomorphic to* $\mathcal{F}[\![E]\!]$. ■

Corollary 6.2 *Let* $E \in \mathcal{E}$. *Then* $\mathcal{M}[\![\mathcal{N}_{\rm loc}[\![E]\!]]\!]$ *is p-isomorphic to* $\mathcal{M}[\![E]\!]$. ■

6.2 Integrated Label Oriented Net Semantics

In this section we take a different approach, called label oriented [43], to the definition of the integrated net semantics for EMPA, aiming at the obtainment of more compact nets. Following such an approach, we no longer fully retain the syntactical structure of terms w.r.t. static operators within net places (so, e.g., the place associated with $<a, \tilde{\lambda}>.\underline{0} \parallel_\emptyset <a, \tilde{\lambda}>.\underline{0}$ can contain up to two tokens), but we keep track of it by means of places with outgoing inhibitor and contextual arcs. Moreover, we exploit inhibitor arcs to produce a linear representation of the alternative composition operator instead of the multiplicative one of the location oriented approach. Given $E_1 + E_2$, the idea is that each place representing a derivative of E_1 (E_2) is decorated with a conflict name k ($\bar{k}$) and that for every conflict name there is a place which, when holding at least one token, inhibits all the transitions having in their preset places decorated with the complementary conflict name.

The structure of this section parallels that of the previous section. Furthermore, this section is concluded by showing that most inhibitor arcs (actually, all the inhibitor arcs if guarded alternative compositions of the form $\sum_{i \in I} <a_i, \tilde{\lambda}_i>.E_i$ are employed) and all the contextual arcs can be safely removed thereby obtaining an optimized integrated label oriented net semantics.

6.2.1 Net Places

The first step in the definition of the integrated label oriented net semantics consists of introducing a set of places onto which terms will be mapped.

Definition 6.4 *The set $\mathcal{V}_{\rm lab}$ of places is defined by*

$$\mathcal{V}_{\rm lab} = GACom \cup Confl \cup Inh \cup Ren$$

where:

- *GACom is a set of guarded alternative compositions decorated with a conflict set defined by*

$$GACom = \{KE \mid K \subseteq Confl \wedge E \equiv \sum_{h=1}^{n} <a_h, \tilde{\lambda}_h>.E_h \in \mathcal{G} \wedge n \in \mathbb{N}_+\}$$

- *Confl is a set of conflicts defined by*

$$Confl = \chi \cup \bar{\chi}$$

  where χ is a set with countably many elements and $\bar{\bar{k}} = k$ for all $k \in \chi$.

- *Inh is a set of action type inhibitors defined by*

$$Inh = \{I_a \mid a \in AType\}$$

- *Ren is a set of action type renaming functions defined by*

$$Ren = \{[a_1 \mapsto a_2] \mid a_1, a_2 \in AType \text{ distinct}\} \cup \{[a_1, a_2 \mapsto a] \mid a_1, a_2, a \in AType \text{ distinct}\}$$

where:

  - *place* $[a_1 \mapsto a_2]$ *corresponds to partial function* $\rho : \mathcal{P}_{\rm fin}(\mathcal{V}_{\rm lab}) \multimap \mathcal{P}_{\rm fin}(\mathcal{V}_{\rm lab})$ *such that*

$$\rho(\{<a_1, \tilde{\lambda}>\} \cup V) = \{<a_2, \tilde{\lambda}>\} \cup V$$

  - *place* $[a_1, a_2 \mapsto a]$ *corresponds to partial function* $\rho : \mathcal{P}_{\rm fin}(\mathcal{V}_{\rm lab}) \multimap \mathcal{P}_{\rm fin}(\mathcal{V}_{\rm lab})$ *such that*

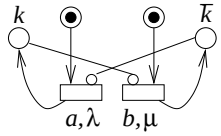
$$\rho(\{<a_1, \tilde{\lambda}>, <a_2, \tilde{\mu}>\} \cup V) = \{<a, \max(\tilde{\lambda}, \tilde{\mu})>\} \cup V$$

    whenever $\min(\tilde{\lambda}, \tilde{\mu}) = *$.

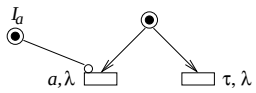
We use $V, V', \ldots$ *as metavariables for* $\mathcal{V}_{\rm lab}$ *and* $Q, Q', \ldots$ *as metavariables for* $\mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm lab})$. ■

The first class of places, *GACom*, is made out of guarded alternative compositions each juxtaposed to a conflict set K. When used in the syntax of places, the guarded alternative composition operator is assumed to be associative and commutative and admit $\underline{0}$ as neutral element. It is worth noting that no static operator occurs in places in *GACom*. Places in *GACom* will have neither outgoing inhibitor nor outgoing contextual arcs.

The second class of places, *Confl*, is composed of conflicts. Place k is used to prevent the execution of every transition leaving place $KE \in GACom$ such that $\bar{k} \in K$. This allows terms like $E_1 + E_2$ to be represented as the parallel composition of $\{k\}E_1$ and $\{\bar{k}\}E_2$ by means of inhibitor arcs. For example, the net we wish to associate with $<a, \lambda>.\underline{0} + <b, \mu>.\underline{0}$ has the following form: [1]



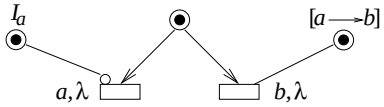
The third class of places, *Inh*, is composed of action type inhibitors. Place I_a is used to prevent (by means of inhibitor arcs) the execution of every transition having type a which is forbidden due to a functional abstraction, a functional relabeling, or a synchronization on a. For example, the net we wish to associate with $(<a, \lambda>.\underline{0})/\{a\}$ has the following form: [2]



[1] We shall see that for terms whose outermost operator is a guarded alternative composition a more compact net is produced.

[2] Also a place $[a \mapsto \tau]$ should appear in the contextual set of the transition labeled with τ, λ. It is not shown here for the sake of simplicity.

The fourth class of places, *Ren*, is made out of action type renaming functions. Places of type $[a_1 \mapsto a_2]$ are used to implement the functional abstraction operator as well as the functional relabeling operator, whereas places of type $[a_1, a_2 \mapsto a]$ are used to implement synchronizations. In both cases, contextual arcs are employed. For example, the net we wish to associate with $(<a, \lambda>.\underline{0})[b/a]$ has the following form:



More precisely, since information about occurrences of static operators cannot be retained when mapping terms onto places in *GACom* due to the form of such places, we use renamings to correctly relate action types occurring in action type sets of functional abstraction or parallel composition operators or in action type relabeling functions of functional relabeling operators to the corresponding scopes of such static operators represented by places in *GACom*. To achieve this, we assume the existence of a mechanism generating a new action type for every action type occurring in action type sets of functional abstraction or parallel composition operators or in the domain of action type relabeling functions of functional relabeling operators. Action type renaming functions, as well as action type inhibitors, will be concerned only with these newly generated action types, which are denoted by the original action types with subscript *new*.

Example 6.10 Consider the following terms

$$\begin{aligned} E_1 &\equiv (<a, \lambda>.\underline{0})/\{a\} + <a, \lambda>.\underline{0} \\ E_2 &\equiv (<a, \lambda>.\underline{0})[b/a] + <a, \lambda>.\underline{0} \\ E_3 &\equiv (<a, \lambda>.\underline{0} \parallel_{\{a\}} <a, *>.\underline{0}) + <a, \lambda>.\underline{0} \end{aligned}$$

If we want to give an integrated label oriented net semantics to the terms above, then the static operator occurring in each of them must affect only its scope by completely disregarding the right hand side operand of the alternative composition operator. To achieve this, from the point of view of the integrated label oriented net semantics we rename the action types inside the scope of the static operators as follows

$$\begin{aligned} E'_1 &\equiv (<a_{new,1}, \lambda>.\underline{0})/\{a\} + <a, \lambda>.\underline{0} \\ E'_2 &\equiv (<a_{new,2}, \lambda>.\underline{0})[b/a] + <a, \lambda>.\underline{0} \\ E'_3 &\equiv (<a_{new,l}, \lambda>.\underline{0} \parallel_{\{a\}} <a_{new,r}, *>.\underline{0}) + <a, \lambda>.\underline{0} \end{aligned}$$

and we introduce the following inhibitors and renamings

$$\begin{array}{ll} I_{a_{new,1}} & [a_{new,1} \mapsto \tau] \\ I_{a_{new,2}} & [a_{new,2} \mapsto b] \\ I_{a_{new,l}}, I_{a_{new,r}} & [a_{new,l}, a_{new,r} \mapsto a] \end{array}$$

■

Before presenting the decomposition function for the label oriented approach, let us introduce some notation. Given $D \subseteq AType$, we denote by $E\langle D := D_y\rangle$ the term obtained from E by replacing occurrences of elements in D with occurrences of the corresponding elements in $D_y = \{a_y \mid a \in D\}$. To facilitate such a replacement, we view every constant definition $A \triangleq E$ as $A(a_1, \ldots, a_n) \triangleq E$ where $\{a_1, \ldots, a_n\} = sort(E)$.

We also denote by $Dom(\varphi)$ the set $\{a \in AType \mid \varphi(a) \neq a\}$. Finally, for $K \subseteq Confl$ we let $K(Q) = \{\!|\, (K \cup K')E \mid K'E \in Q \cap GACom \,|\!\} \oplus (Q \cap (Confl \cup Inh \cup Ren))$.

Definition 6.5 *The decomposition function $dec_{\rm lab} : \mathcal{G} \longrightarrow \mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm lab})$ is defined by induction on the syntactical structure of terms in $\mathcal{G}$ as follows:*

$$
\begin{array}{rcl}
dec_{\rm lab}(\underline{0}) & = & \emptyset \\
dec_{\rm lab}(<a,\tilde{\lambda}>.E) & = & \{\!|\, \emptyset<a,\tilde{\lambda}>.E \,|\!\} \\
dec_{\rm lab}(E/L) & = & dec_{\rm lab}(E\langle L := L_{new}\rangle) \oplus \\
 & & \{\!|\, I_{a_{new}} \mid a \in L \,|\!\} \oplus \\
 & & \{\!|\, [a_{new} \mapsto \tau] \mid a \in L \,|\!\} \\
dec_{\rm lab}(E[\varphi]) & = & dec_{\rm lab}(E\langle Dom(\varphi) := Dom(\varphi)_{new}\rangle) \oplus \\
 & & \{\!|\, I_{a_{new}} \mid a \in Dom(\varphi) \,|\!\} \oplus \\
 & & \{\!|\, [a_{new} \mapsto \varphi(a)] \mid a \in Dom(\varphi) \,|\!\} \\
dec_{\rm lab}(E_1 + E_2) & = & \left\{ \begin{array}{ll} \{k_{new}\}(dec_{\rm lab}(E_1)) \oplus \{\bar{k}_{new}\}(dec_{\rm lab}(E_2)) & \textit{if } E_1 + E_2 \textit{ is not guarded} \\ \{\!|\, \emptyset E_1 + E_2 \,|\!\} & \textit{if } E_1 + E_2 \textit{ is guarded} \end{array} \right. \\
dec_{\rm lab}(E_1 \parallel_S E_2) & = & dec_{\rm lab}(E_1\langle S := S_{new,l}\rangle) \oplus \\
 & & dec_{\rm lab}(E_2\langle S := S_{new,r}\rangle) \oplus \\
 & & \{\!|\, I_{a_{new,l}} \mid a \in S \,|\!\} \oplus \\
 & & \{\!|\, I_{a_{new,r}} \mid a \in S \,|\!\} \oplus \\
 & & \{\!|\, [a_{new,l}, a_{new,r} \mapsto a] \mid a \in S \,|\!\} \\
dec_{\rm lab}(A) & = & dec_{\rm lab}(E) \qquad \textit{if } A \stackrel{\Delta}{=} E
\end{array}
$$

■

It is worth noting that no place is associated with $\underline{0}$ and that there are two clauses for the alternative composition operator: the former implements the scheme shown when discussing places in *Confl*, the latter is an optimization of the previous one exploiting the guardedness of alternative compositions in order to further reduce the size of the resulting nets. Finally, observe that $dec_{\rm lab}$ is not injective and does not necessarily associates place sets with terms.

Example 6.11 Consider term

$$<a,\tilde{\lambda}>.E + <b,\tilde{\mu}>.F$$

Creating by means of the second clause for the alternative composition operator only one place $\emptyset<a,\tilde{\lambda}>.E + <b,\tilde{\mu}>.F$ with two transitions and two arcs is more convenient than creating through the first clause four places $\{k\}<a,\tilde{\lambda}>.E$, $\{\bar{k}\}<b,\tilde{\mu}>.F$, k, and $\bar{k}$ with two transitions and six arcs. ■

Example 6.12 The following pairs of terms give rise to the same decomposition:

$$\begin{array}{rcl} \underline{0} & \text{and} & \underline{0} \parallel_\emptyset \underline{0} \\ E \parallel_\emptyset F & \text{and} & F \parallel_\emptyset E \\ <a, \tilde{\lambda}>.E + <b, \tilde{\mu}>.F & \text{and} & <b, \tilde{\mu}>.F + <a, \tilde{\lambda}>.E \\ <a, \tilde{\lambda}>.E + \underline{0} & \text{and} & <a, \tilde{\lambda}>.E \end{array}$$

Additionally, note that term

$$<a, \tilde{\lambda}>.\underline{0} \parallel_\emptyset <a, \tilde{\lambda}>.\underline{0}$$

is mapped onto a single place given by $\emptyset<a, \tilde{\lambda}>.\underline{0}$ with multiplicity two. In the location oriented approach, instead, the same term would be mapped onto the two distinct places $<a, \tilde{\lambda}>.\underline{0} \parallel_\emptyset id$ and $id \parallel_\emptyset <a, \tilde{\lambda}>.\underline{0}$. ■

6.2.2 Net Transitions

The second step in the definition of the integrated label oriented net semantics consists of suitably connecting places by means of transitions. The set of transitions $\longrightarrow_{\rm lab}$ is defined as the least subset of $\mathcal{P}_{\rm fin}(GACom) \times \mathcal{P}_{\rm fin}(Confl \cup Inh) \times \mathcal{P}_{\rm fin}(Ren) \times (AType \times ARate^{\mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm lab})}) \times \mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm lab})$ generated by the axiom reported in the first part of Table 6.2, where $n \in \mathbb{N}_+$, $n_i \in \mathbb{N}_+$ for all $i = 1, \ldots, n$, and $m \in \mathbb{N}$.

The preset of the transition is a finite nonempty set of places in $GACom$ each with a set of factorized summands, which contains more than one place if and only if the transition is due to a synchronization. Since the preset is a set, the synchronization of a place with itself (through suitable renamings) is avoided.

The inhibitor set of the transition is a finite nonempty set of places in $Confl \cup Inh$. There is a place for every conflict name which is complementary to a conflict name appearing in the conflict set of some place in the preset. There is also a place for the action type labeling the transition. These places prevent the transition from firing (i.e. they contain tokens) if either a transition with a place in its preset having a complementary conflict name has been previously fired, or the action type labeling the transition is subject to functional abstraction, functional relabeling, or synchronization.

The contextual set of the transition is a finite (possibly empty) set of places in Ren. These places allow the transition to be fired (i.e. they contain tokens) if there is a composition of renamings that, when applied to the set of factorized actions appearing in the preset, produces the action labeling the transition.

The postset of the transition is the multiset composed of both the places representing the decomposition of the derivative terms of the factorized summands in the preset and the places associated with the conflict names in the preset. As a consequence, if the transition fires then all the transitions with a place in their preset having a complementary conflict name are inhibited.

The axiom is subject to the following side conditions which are explained below:

1. For all $i, i' = 1, \ldots, n$, $a_i \neq a_{i'}$ and $K_i \cap \bar{K}_{i'} = \emptyset$ whenever $i \neq i'$.

2. For all $i = 1, \ldots, n$ and $h, h' = 1, \ldots, n_i$, it must hold $PL(<a_i, \tilde{\lambda}_{i,h}>) = PL(<a_i, \tilde{\lambda}_{i,h'}>)$ and $dec_{\rm lab}(E_{i,h}) = dec_{\rm lab}(E_{i,h'})$, whereas $F_i \equiv \sum_{h=n_i+1}^{m_i} <a_{i,h}, \tilde{\lambda}_{i,h}>.E_{i,h}$ is such that for all $h = n_i + 1, \ldots, m_i$ it must hold $a_{i,h} \neq a_i$, $PL(<a_{i,h}, \tilde{\lambda}_{i,h}>) \neq PL(<a_i, \tilde{\lambda}_{i,1}>)$, or $dec_{\rm lab}(E_{i,h}) \neq dec_{\rm lab}(E_{i,1})$.

3. $\{<a, \tilde{\lambda}>\} = \rho_1(\ldots(\rho_m(\{<a_i, Aggr\{\!|\, \tilde{\lambda}_{i,h} \mid 1 \leq h \leq n_i \,|\!\}> \mid 1 \leq i \leq n\}))\ldots)$.

$$(\bigcup_{i=1}^{n} \{K_i \sum_{h=1}^{n_i} <a_i, \tilde{\lambda}_{i,h}>.E_{i,h} + F_i\},\ \bigcup_{i=1}^{n} \bar{K}_i \cup \{I_a\},\ \bigcup_{j=1}^{m} \{\rho_j\}) \xrightarrow{norm_{\rm lab}(<a,\tilde{\lambda}>,E,f_1,f_2)}_{\rm lab} \bigoplus_{i=1}^{n} dec_{\rm lab}(E_{i,1}) \oplus \bigoplus_{i=1}^{n} K_i$$

$$norm_{\rm lab}(<a, \tilde{\lambda}>, E, f_1, f_2) = \begin{cases} <a, *> & \text{if } \tilde{\lambda} = * \\ <a, \tilde{\lambda} \cdot f_1 \cdot f_2/s> & \text{if } \tilde{\lambda} \in \mathbb{R}_+ \\ <a, \infty_{l,w \cdot f_1 \cdot f_2/s}> & \text{if } \tilde{\lambda} = \infty_{l,w} \end{cases}$$

where $s = \sum\{\!|\, f_2' \mid \exists Q_1, K, R, E', Q_2.\, (Q_1, K \cup \{I_a\}, R) \xrightarrow{norm_{\rm lab}(<a,\tilde{\lambda}>,E',f_1,f_2')}_{\rm lab} Q_2 \wedge dec_{\rm lab}(E') = dec_{\rm lab}(E) \,|\!\}$

Table 6.2: Axiom for EMPA integrated label oriented net semantics

Condition 1 states that all the places in the preset have different factorized action types as well as disjoint conflict sets. This avoids the generation of synchronization transitions whose preset contains places associated with terms which cannot be synchronized.

Example 6.13 Consider term

$$<a, \lambda>.E \parallel_{\{a\}} (<a, *>.F \parallel_{\emptyset} <a, *>.G)$$

with E, F, and G containing no occurrences of a. Its decomposition generates places $\emptyset<a_{new,l}, \lambda>.E$, $\emptyset<a_{new,r}, *>.F$, and $\emptyset<a_{new,r}, *>.G$ in the case $F \not\equiv G$. Without constraint $a_i \neq a_{i'}$, a transition whose preset is composed of all the three places above could be generated through suitable renamings. This is not correct as the second and the third places are independent of each other according to the intended semantics of the term, so at most one of them can be in the preset of a transition.

Additionally, consider term

$$(<a, \lambda>.E \parallel_{\emptyset} \underline{0}) + (<b, *>.F \parallel_{\emptyset} \underline{0})$$

whose decomposition generates places $\{k\}<a, \lambda>.E$ and $\{\bar{k}\}<b, *>.F$. Without constraint $K_i \cap \bar{K}_{i'} = \emptyset$, a synchronization between these two places would in principle be possible through a suitable renaming but this would not reflect the intended semantics of the term. ■

Condition 2 states that the summands factorized for each place in the preset of the transition are all and only those involving the same action type a_i, the same priority level, and derivative terms having

the same decomposition. We require terms to have the same decomposition instead of being syntactically identical because function $dec_{\rm lab}$ is not injective. Such a factorization, together with the rate aggregation of condition 3, prevents transitions from being given wrong rates.

Example 6.14 Consider term

$$<a,\lambda>.E + <a,\lambda>.F$$

Its decomposition is place $\emptyset<a,\lambda>.E + <a,\lambda>.F$. If $dec_{\rm lab}(E) = dec_{\rm lab}(F)$, due to the factorization we generate one transition labeled with $a, 2 \cdot \lambda$ as expected. ■

Condition 3 states that the basic action from which the action labeling the transition is derived (by means of function $norm_{\rm lab}$) is the result of the application of the composition of the renaming functions appearing in the contextual set to the set of actions factorized for each place in the preset. If the contextual set is empty, the composition is taken to be the identity function over Act. Note that, because of the definition of Ren, the basic action is undefined (hence a transition cannot be built) whenever more than one place in the preset contributes with an active action.

Example 6.15 Consider term

$$<a,\lambda>.\underline{0} \parallel_{\{a\}} (<a,*>.\underline{0} \parallel_{\{a\}} <a,*>.\underline{0})$$

Its decomposition comprises places $\emptyset<a_{new,l},\lambda>.\underline{0}$, $\emptyset<a'_{new,l},*>.\underline{0}$, and $\emptyset<a'_{new,r},*>.\underline{0}$ and renamings $[a_{new,l}, a_{new,r} \mapsto a]$ and $[a'_{new,l}, a'_{new,r} \mapsto a_{new,r}]$. The second renaming applied to $\{<a_{new,l},\lambda>, <a'_{new,l},*>, <a'_{new,r},*>\}$ results in $\{<a_{new,l},\lambda>, <a_{new,r},*>\}$, from which we obtain $\{<a,\lambda>\}$ through the first renaming. ■

Function $norm_{\rm lab} : Act \times (\mathcal{G} \cup \{\bot\}) \times \mathbb{N}^{\mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm lab})} \times \mathbb{N}^{\mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm lab})} \longrightarrow AType \times ARate^{\mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm lab})}$, defined in the second part of Table 6.2, computes the actual rate of transitions according to their basic rate and three factors – the active derivative term E, the active contribution f_1, and the passive contribution f_2 – which are defined below:

$$E \equiv \begin{cases} E_{i,1} & \text{if } \exists! i = 1,\ldots,n.\, \tilde{\lambda}_{i,1} \in \mathbb{R}_+ \cup Inf \\ \bot & \text{if } \forall i = 1,\ldots,n.\, \tilde{\lambda}_{i,1} = * \end{cases}$$

$$f_1 = \begin{cases} M_{\rm curr}(K_i \sum_{h=1}^{n_i} <a_i,\tilde{\lambda}_{i,h}>.E_{i,h} + F_i) & \text{if } \exists! i = 1,\ldots,n.\, \tilde{\lambda}_{i,1} \in \mathbb{R}_+ \cup Inf \\ 1 & \text{if } \forall i = 1,\ldots,n.\, \tilde{\lambda}_{i,1} = * \end{cases}$$

$$f_2 = \begin{cases} \prod_{i=1}^{n} \{\!|\, n_i \cdot M_{\rm curr}(K_i \sum_{h=1}^{n_i} <a_i,\tilde{\lambda}_{i,h}>.E_{i,h} + F_i) \mid \tilde{\lambda}_{i,1} = * \,|\!\} & \text{if } \exists i = 1,\ldots,n.\, \tilde{\lambda}_{i,1} = * \\ 1 & \text{if } n = 1 \wedge \tilde{\lambda}_{1,1} \in \mathbb{R}_+ \cup Inf \end{cases}$$

In case of several transitions coming from the synchronization of a place contributing with an active action with several places contributing with alternative passive actions of the same type, factor f_2 is suitably normalized (see the second part of Table 6.2). Such transitions are those having the same type, the same

active basic rate, and active derivative terms with the same decomposition. The normalizing factor s for them is computed by summing up their passive contributions.

The factor E is the derivative term appearing in the place (if any) of the preset which contributes to the transition with an active action. Since transitions coming from the synchronization of a place contributing with a nonpassive action with several places contributing with alternative passive actions of the same type have in general different presets, we need this piece of information beside the basic action in order to correctly carry out the normalization.

Example 6.16 Consider term

$$(<a,\lambda>.E + <a,\lambda>.F) \parallel_{\{a\}} <a,*>.\underline{0}$$

where E and F contain no occurrences of a and $dec_{\rm lab}(E) \neq dec_{\rm lab}(F)$. Its decomposition consists of places $\emptyset<a_{new,l},\lambda>.E + <a_{new,l},\lambda>.F$ and $\emptyset<a_{new,r},*>.\underline{0}$. There are two transitions both labeled with a,λ whose postsets are $dec_{\rm lab}(E)$ and $dec_{\rm lab}(F)$, respectively. The rates of these two transitions must be separately normalized since they result from the participation of two distinct active actions. This is achieved by labeling the first transition with E and the second transition with F. ■

The marking dependent factor f_1 is the number of tokens in the place (if any) of the preset which contributes to the transition with an active action.

Example 6.17 Consider term

$$<a,\lambda>.E \parallel_{\emptyset} <a,\lambda>.E$$

Its decomposition gives rise to place $\emptyset<a,\lambda>.E$ with multiplicity two, i.e. marked with two tokens. This place has one outgoing transition whose rate must be $2\cdot\lambda$ at the first firing and λ at the second firing. The needed piece of information is recorded by factor f_1. ■

The marking dependent factor f_2 is the product, over the set of places (if any) of the preset which contribute to the transition with a passive action, of the number of factorized summands multiplied by the number of tokens contained in the related places.

Example 6.18 Consider term

$$<a,\lambda>.\underline{0} \parallel_{\{a\}} ((<a,*>.\underline{0} \parallel_{\emptyset} <a,*>.\underline{0} \parallel_{\emptyset} <a,*>.\underline{0}) + (<a,*>.\underline{0} + <a,*>.\underline{0}))$$

Its decomposition produces three places in *GACom* given by $V_1 = \emptyset<a_{new,l},\lambda>.\underline{0}$ marked with one token, $V_2 = \{k\}<a_{new,r},*>.\underline{0}$ marked with three tokens, and $V_3 = \{\bar{k}\}<a_{new,r},*>.\underline{0} + <a_{new,r},*>.\underline{0}$ marked with one token. Place V_1 has two outgoing transitions $t_{1,2}$ and $t_{1,3}$ having type a which result from the synchronization with V_2 and V_3, respectively. Transition $t_{1,2}$ has basic rate λ, factor $f_1 = M_{\rm curr}(V_1)$, and factor $f_2 = 1\cdot M_{\rm curr}(V_2)$; when fired, its rate is $\lambda\cdot 1\cdot(1\cdot 3) = 3\cdot\lambda$. Transition $t_{1,3}$ has basic rate λ, factor $f_1 = M_{\rm curr}(V_1)$, and factor $f_2 = 2\cdot M_{\rm curr}(V_3)$; when fired, its rate is $\lambda\cdot 1\cdot(2\cdot 1) = 2\cdot\lambda$. The actual rate for

the firing of $t_{1,2}$ is $3 \cdot \lambda/5$ and the actual rate for the firing of $t_{1,3}$ is $2 \cdot \lambda/5$. The normalizing factor $5 = 3 + 2$ is computed by $norm_{\rm lab}$. ■

6.2.3 Nets Associated with Terms

The third step in the definition of the integrated label oriented net semantics consists of associating with each term an appropriate PGSPN by exploiting the previous two steps.

Definition 6.6 *The integrated label oriented net semantics of $E \in \mathcal{G}$ is the PGSPN*

$$\mathcal{N}_{\rm lab}[\![E]\!] = (P_{E,\mathcal{N}_{\rm lab}}, AType \times ARate^{\mathcal{M}u_{\rm fin}(P_{E,\mathcal{N}_{\rm lab}})}, \longrightarrow_{E,\mathcal{N}_{\rm lab}}, dec_{\rm lab}(E), L, W)$$

where:

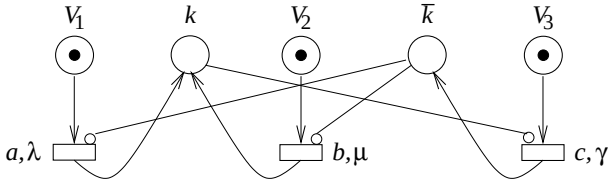
- *$P_{E,\mathcal{N}_{\rm lab}}$ is the least subset of $\mathcal{V}_{\rm lab}$ such that:*
  - *$dom(dec_{\rm lab}(E)) \subseteq P_{E,\mathcal{N}_{\rm lab}}$;*
  - *if $dom(Q_1 \oplus R) \subseteq P_{E,\mathcal{N}_{\rm lab}}$ and $(Q_1, K \cup \{I_a\}, R) \xrightarrow{norm_{\rm lab}(<a,\tilde{\lambda}>,E,f_1,f_2)}_{\rm lab} Q_2$, then $dom(Q_2 \oplus K \oplus \{I_a\}) \subseteq P_{E,\mathcal{N}_{\rm lab}}$.*
- *$\longrightarrow_{E,\mathcal{N}_{\rm lab}}$ is the restriction of $\longrightarrow_{\rm lab}$ to $\mathcal{P}_{\rm fin}(P_{E,\mathcal{N}_{\rm lab}} \cap GACom) \times \mathcal{P}_{\rm fin}(P_{E,\mathcal{N}_{\rm lab}} \cap (Confl \cup Inh)) \times \mathcal{P}_{\rm fin}(P_{E,\mathcal{N}_{\rm lab}} \cap Ren) \times (AType \times ARate^{\mathcal{M}u_{\rm fin}(P_{E,\mathcal{N}_{\rm lab}})}) \times \mathcal{M}u_{\rm fin}(P_{E,\mathcal{N}_{\rm lab}})$.*
- *$L : \longrightarrow_{E,\mathcal{N}_{\rm lab}} \longrightarrow APLev$ such that:*
  - *$L(Q_1, K \cup \{I_a\}, R, norm_{\rm lab}(<a,*>, E, f_1, f_2), Q_2) = -1$;*
  - *$L(Q_1, K \cup \{I_a\}, R, norm_{\rm lab}(<a,\lambda>, E, f_1, f_2), Q_2) = 0$;*
  - *$L(Q_1, K \cup \{I_a\}, R, norm_{\rm lab}(<a,\infty_{l,w}>, E, f_1, f_2), Q_2) = l$.*
- *$W : \longrightarrow_{E,\mathcal{N}_{\rm lab}} \longrightarrow \{*\} \cup \mathbb{R}_+^{\mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm lab})}$ such that:*
  - *$W(Q_1, K \cup \{I_a\}, R, norm_{\rm lab}(<a,*>, E, f_1, f_2), Q_2) = *$;*
  - *$W(Q_1, K \cup \{I_a\}, R, norm_{\rm lab}(<a,\lambda>, E, f_1, f_2), Q_2) = \lambda'$ if $norm_{\rm lab}(<a,\lambda>, E, f_1, f_2) = <a,\lambda'>$;*
  - *$W(Q_1, K \cup \{I_a\}, R, norm_{\rm lab}(<a,\infty_{l,w}>, E, f_1, f_2), Q_2) = w'$ if $norm_{\rm lab}(<a,\infty_{l,w}>, E, f_1, f_2) = <a,\infty_{l,w'}>$.* ■

We observe that $\mathcal{N}_{\rm lab}[\![E]\!]$ is not necessarily a GSPN whenever $E \in \mathcal{E}$ (see Ex. 6.22).

We report below some examples showing that the PGSPNs representing the integrated label oriented net semantics of EMPA terms are not necessarily safe and give a linear representation of alternative compositions.

Example 6.19 Consider terms E_1 and E_2 introduced in Ex. 6.1. Then $\mathcal{N}_{\rm lab}[\![E_1]\!]$ and $\mathcal{N}_{\rm lab}[\![E_2]\!]$ have the same structure as $\mathcal{N}_{\rm loc}[\![E_1]\!]$ and $\mathcal{N}_{\rm loc}[\![E_2]\!]$, respectively (see Ex. 6.9), up to $\underline{0}$ related places and their incoming arcs. ■

Example 6.20 Consider term E of Ex. 6.2. Then $\mathcal{N}_{\rm lab}[\![E]\!]$ is the following net:

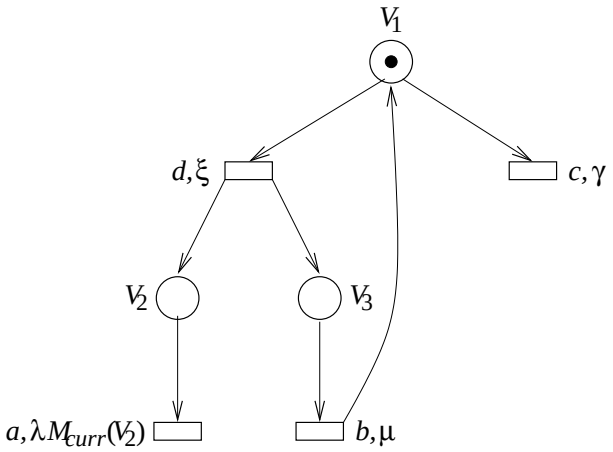


where $V_1 = \{k\}<a,\lambda>.\underline{0}$, $V_2 = \{k\}<b,\mu>.\underline{0}$, and $V_3 = \{\bar{k}\}<c,\gamma>.\underline{0}$. Since E is not a guarded alternative composition, the second clause of $dec_{\rm lab}$ for the alternative composition operator cannot be applied. ■

Example 6.21 Consider term

$$A \triangleq <d,\xi>.(<a,\lambda>.\underline{0} \parallel_\emptyset <b,\mu>.A) + <c,\gamma>.\underline{0}$$

Then $\mathcal{N}_{\rm lab}[\![A]\!]$ is the following net:

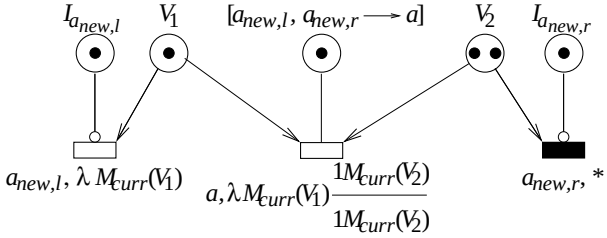


where $V_1 = \emptyset<d,\xi>.(<a,\lambda>.\underline{0} \parallel_\emptyset <b,\mu>.A) + <c,\gamma>.\underline{0}$, $V_2 = \emptyset<a,\lambda>.\underline{0}$, and $V_3 = \emptyset<b,\mu>.A$. Since A is a guarded alternative composition, the second clause of $dec_{\rm lab}$ for the alternative composition operator has been applied. It is worth noting that both $\mathcal{I}[\![A]\!]$ and $\mathcal{N}_{\rm loc}[\![A]\!]$ are infinite whereas $\mathcal{N}_{\rm lab}[\![A]\!]$ is finite. ■

Example 6.22 Consider term

$$E \equiv <a,\lambda>.\underline{0} \parallel_{\{a\}} (<a,*>.\underline{0} \parallel_\emptyset <a,*>.\underline{0})$$

Then $\mathcal{N}_{\rm lab}[\![E]\!]$ is the following net:



where $V_1 = \emptyset{<}a_{new,l}, \lambda{>}.\underline{0}$ and $V_2 = \emptyset{<}a_{new,r}, *{>}.\underline{0}$. Place V_2 contains more than one token, hence the net is not safe. The net is not a GSPN although $E \in \mathcal{E}$. There is only one enabled transition; it can fire only once with rate λ.

If we consider term

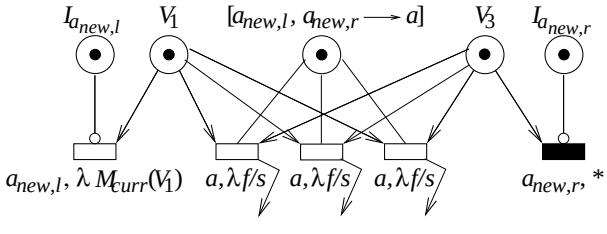
$$E' \equiv (<a, \lambda>.\underline{0} \parallel_{\emptyset} <a, \lambda>.\underline{0}) \parallel_{\{a\}} (<a, *>.\underline{0} \parallel_{\emptyset} <a, *>.\underline{0})$$

then we get the same net as above, but with two tokens in place V_1. Again there is only one enabled transition; it can fire the first time with rate $2 \cdot \lambda$ and the second time with rate λ.

Finally, consider term

$$E'' \equiv <a, \lambda>.\underline{0} \parallel_{\{a\}} (<a, *>.F_1 + <a, *>.F_2 + <a, *>.F_3)$$

where $dec_{\rm lab}(F_1) \neq dec_{\rm lab}(F_2)$, $dec_{\rm lab}(F_1) \neq dec_{\rm lab}(F_3)$, and $dec_{\rm lab}(F_2) \neq dec_{\rm lab}(F_3)$. Then $\mathcal{N}_{\rm lab}[\![E'']\!]$ is the following net:



where $V_3 = \emptyset{<}a_{new,r}, *{>}.F_1 + {<}a_{new,r}, *{>}.F_2 + {<}a_{new,r}, *{>}.F_3$ and $f = M_{\rm curr}(V_1) \cdot 1 \cdot M_{\rm curr}(V_3)$ and $s = 1 \cdot M_{\rm curr}(V_3) + 1 \cdot M_{\rm curr}(V_3) + 1 \cdot M_{\rm curr}(V_3)$. There are three mutually exclusive enabled transitions: each of them can fire only once with rate $\lambda/3$. ■

We conclude with a property of the nets obtained by applying the integrated label oriented net semantics which includes the case shown in Ex. 6.21.

Theorem 6.3 *Let $E \in \mathcal{G}$. If the three following conditions hold:*

- *For each functional abstraction operator $\_/L$ occurring within a recursive constant definition, $sort(E) \cap L = \emptyset$.*
- *For each functional relabeling operator $\_[\varphi]$ occurring within a recursive constant definition, $sort(E) \cap Dom(\varphi) = \emptyset$.*

- *For each parallel composition operator* $\_ \|_S \_$ *occurring within a recursive constant definition,* $sort(E) \cap S = \emptyset$.

then $\mathcal{N}_{\rm lab}[\![E]\!]$ *is finite.* ■

6.2.4 Retrievability Result

In this section we prove the correctness of the integrated label oriented net semantics. More precisely, we demonstrate that $\mathcal{RG}[\![\mathcal{N}_{\rm lab}[\![E]\!]]\!]$ is strongly EMB to $\mathcal{I}[\![E]\!]$ for all $E \in \mathcal{G}$. We observe that the isomorphism result does not hold in general. If we consider term E of Ex. 6.22, we have that its integrated interleaving semantics has three states and two transitions each labeled with $a, \lambda/2$ while the reachability graph of its integrated label oriented net semantics has only two states and one transition labeled with a, λ.

Theorem 6.4 *Let* $\mathcal{B} = \{(E,Q) \in \mathcal{G} \times \mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm lab}) \mid \exists K \in \mathcal{P}_{\rm fin}(Confl).\, Q = dec_{\rm lab}(E) \oplus K\}$ *and let* $\mathcal{B}'$ *be the reflexive, symmetric and transitive closure of* $\mathcal{B}$. *Whenever* $(E,Q) \in \mathcal{B}$, *then* $Rate(E,a,l,C) = Rate(Q,a,l,C)$ *for all* $a \in AType$, $l \in APLev$, *and* $C \in (\mathcal{G} \cup \mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm lab}))/\mathcal{B}'$. ■

Corollary 6.3 *Let* $E \in \mathcal{G}$. *Then* $\mathcal{F}[\![\mathcal{N}_{\rm lab}[\![E]\!]]\!]$ *is bisimilar to* $\mathcal{F}[\![E]\!]$. ■

Corollary 6.4 *Let* $E \in \mathcal{E}$. *Then* $\mathcal{M}[\![\mathcal{N}_{\rm lab}[\![E]\!]]\!]$ *is p-bisimilar to* $\mathcal{M}[\![E]\!]$. ■

Theorem 6.5 *Let* $E \in \mathcal{G}$ *a be finite state term and let* $state(Z)$ *be the set of states of LTS* Z. *Then* $|state(\mathcal{RG}[\![\mathcal{N}_{\rm lab}[\![E]\!]]\!])| \leq |state(\mathcal{I}[\![E]\!])|$. *In particular,* $|state(\mathcal{RG}[\![\mathcal{N}_{\rm lab}[\![E]\!]]\!])| < |state(\mathcal{I}[\![E]\!])|$ *whenever* $E \equiv E_1 \|_S E_2$ *with* $E_1 \equiv E_2$ *and* $(sort(E_1) \cup sort(E_2)) \cap S = \emptyset$. ■

6.2.5 Net Optimization

In this section we present an optimized version of the integrated label oriented net semantics which stems from the fact that all the places containing information about the syntactical structure of terms w.r.t. static operators can be ruled out (together with the related transitions in case of inhibitor place). This is a consequence of the fact that such places are generated only when applying function $dec_{\rm lab}$ to the initial term or the derivative sequential terms of a transition: the axiom of Table 6.2 never causes the movement of a token in one of these places in isolation.

All the inhibitor places associated with action type inhibitors can be removed because, as soon as a token is deposited in an inhibitor place, all the transitions having this place in their inhibitor set cannot fire any more as tokens are never removed from inhibitor places. Likewise, all the contextual places associated with action type renaming functions can be ruled out as well. Here, the reason is that, as soon as a token is deposited in a contextual place, all the transitions having this place in their contextual set can always fire as tokens are never removed from contextual places.

Definition 6.7 *The optimized integrated label oriented net semantics of $E \in \mathcal{G}$ is the PGSPN*

$$\mathcal{N}_{\mathrm{lab,o}}[\![E]\!] = (P_{E,\mathcal{N}_{\mathrm{lab,o}}}, AType \times ARate^{\mathcal{M}u_{\mathrm{fin}}(P_{E,\mathcal{N}_{\mathrm{lab,o}}})}, \longrightarrow_{E,\mathcal{N}_{\mathrm{lab,o}}}, dec_{\mathrm{lab}}(E) \ominus (Inh \cup Ren), L, W)$$

where:

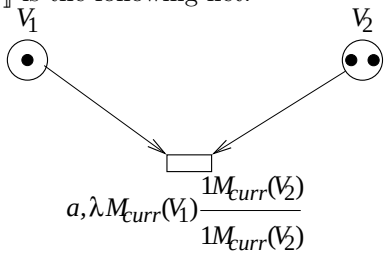
- $P_{E,\mathcal{N}_{\mathrm{lab,o}}} = P_{E,\mathcal{N}_{\mathrm{lab}}} - (Inh \cup Ren)$.
- $\longrightarrow_{E,\mathcal{N}_{\mathrm{lab,o}}}$ *is the least subset of* $\mathcal{P}_{\mathrm{fin}}(P_{E,\mathcal{N}_{\mathrm{lab,o}}} \cap GACom) \times \mathcal{P}_{\mathrm{fin}}(P_{E,\mathcal{N}_{\mathrm{lab,o}}} \cap Conft) \times (AType \times ARate^{\mathcal{M}u_{\mathrm{fin}}(P_{E,\mathcal{N}_{\mathrm{lab,o}}})}) \times \mathcal{M}u_{\mathrm{fin}}(P_{E,\mathcal{N}_{\mathrm{lab,o}}})$ *such that* $(Q_1, K) \xrightarrow{norm_{\mathrm{lab}}(<a,\tilde{\lambda}>,E,f_1,f_2)}_{E,\mathcal{N}_{\mathrm{lab,o}}} Q_2 \ominus (Inh \cup Ren)$ *whenever* $(Q_1, K \cup \{I_a\}, R) \xrightarrow{norm_{\mathrm{lab}}(<a,\tilde{\lambda}>,E,f_1,f_2)}_{E,\mathcal{N}_{\mathrm{lab}}} Q_2$.
- $L : \longrightarrow_{E,\mathcal{N}_{\mathrm{lab,o}}} \longrightarrow APLev$ *and* $W : \longrightarrow_{E,\mathcal{N}_{\mathrm{lab,o}}} \longrightarrow \{*\} \cup \mathbb{R}_+^{\mathcal{M}u_{\mathrm{fin}}(P_{E,\mathcal{N}_{\mathrm{lab,o}}})}$ *are defined as for* $\mathcal{N}_{\mathrm{lab}}[\![E]\!]$. ■

We observe that $\mathcal{N}_{\mathrm{lab,o}}[\![E]\!]$ is a GSPN whenever $E \in \mathcal{E}$.

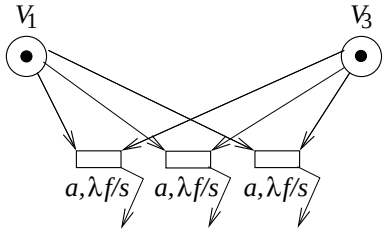
Theorem 6.6 *Let $E \in \mathcal{G}$. Then $\mathcal{RG}[\![\mathcal{N}_{\mathrm{lab,o}}[\![E]\!]]\!]$ is isomorphic to $\mathcal{RG}[\![\mathcal{N}_{\mathrm{lab}}[\![E]\!]]\!]$.* ■

Corollary 6.5 *If all the alternative compositions occurring in $E \in \mathcal{G}$ are guarded, then $\mathcal{N}_{\mathrm{lab,o}}[\![E]\!]$ has neither inhibitor nor contextual arcs.* ■

Example 6.23 Consider the nets shown in Ex. 6.22 and let us see what improvements we gain with the optimization defined above. $\mathcal{N}_{\mathrm{lab,o}}[\![E]\!]$ is the following net:



while $\mathcal{N}_{\mathrm{lab,o}}[\![E'']\!]$ is the following net:



■

6.3 Comparing Integrated Net Semantics

We conclude this chapter by comparing the two integrated net semantics for EMPA w.r.t. the complexity of the related net representations and the complexity of their definitions. Finally, we consider an illustrative example.

6.3.1 Complexity in Net Representation

The label oriented net semantics is remarkably advantageous since it produces significantly smaller nets for many EMPA terms as it can be deduced from Thm. 6.3 and 6.5. The three main reasons are the following.

Unsafeness The decomposition function of the label oriented approach assigns the same place to syntactically identical terms which are composed in parallel whenever the synchronization set does not contain action types occurring in such terms. This is not the case for the location oriented approach because each place keeps track of the syntactical structure of the term, and notably the presence of parallel composition operators. As a consequence, in the former case nets are not necessarily safe, whereas in the latter case they are. It is instructive, in this respect, to consider term A of Ex. 6.21: $\mathcal{N}_{\rm lab}[\![A]\!]$ is a finite unsafe net, while $\mathcal{N}_{\rm loc}[\![A]\!]$ is an infinite safe net.

The non injectivity of $dec_{\rm lab}$ induces further identifications; e.g., terms which are congruent up to associativity, commutativity, and $\underline{0}$ as neutral element for "$\_ + \_$" and "$\_ \|_\emptyset \_$" are mapped to the very same net. In the location oriented approach, instead, each term is assigned a different net; however, if two terms are congruent up to the axioms above, then they give rise to isomorphic nets.

Linearly distributed choice The decomposition function of the label oriented approach handles unguarded alternative compositions as if they were parallel compositions without synchronization set by preserving the mutual exclusion through conflicts. It is easy to see that, if we need p_1 places for representing E_1 and p_2 places for representing E_2, then $E_1 + E_2$ is represented with $p_1 + p_2 + 2$ places. On the contrary, the location oriented approach handles alternative compositions by means of products: $E_1 + E_2$ is represented with $p_1 \cdot p_2$ places.

Of course, the price to pay for producing smaller nets is the presence of inhibitor places associated with conflicts, while the location oriented approach needs no inhibitor arcs.

Null terms optimization The label oriented net semantics does not produce useless places for terms such as $\underline{0}$, $\underline{0} \|_S \underline{0}$, and so on.

6.3.2 Complexity in Definition

Not only the complexity of net representations is different in the two approaches, but also the complexity of the description of the two semantics is different. In fact, the label oriented approach has only one axiom for

generating all the transitions, while the location oriented approach produces transitions by as many inductive rules as the integrated interleaving semantics for EMPA has.

Both semantics satisfy the retrievability principles. However, in the case of the location oriented approach there is a strong retrievability result, as for each term the reachability graph of its integrated location oriented net semantics is isomorphic to its integrated interleaving semantics. For the label oriented approach, instead, we have a weaker retrievability result, as for each term the reachability graph of its integrated label oriented net semantics is strongly EMB to its integrated interleaving semantics.

6.3.3 Integrated Net Semantics of Queueing System $M/M/2/3/4$

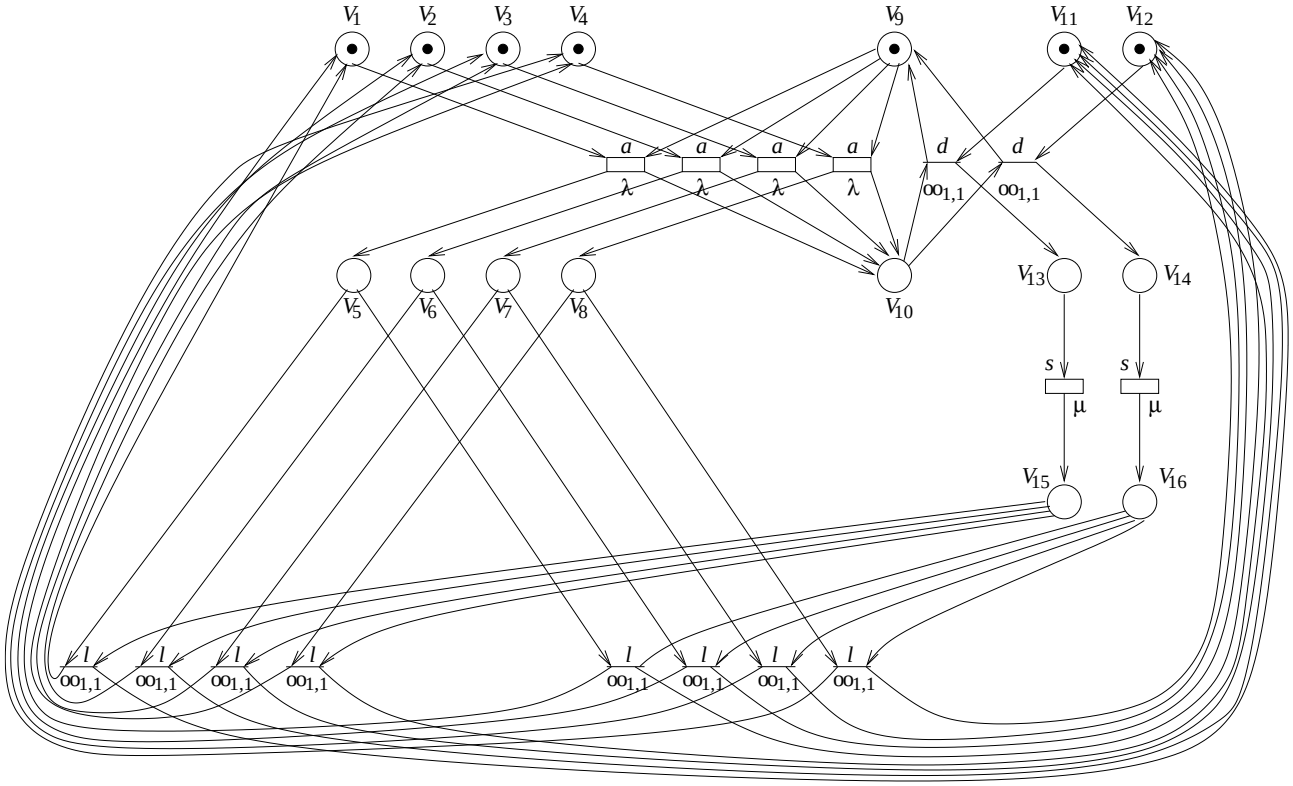
We conclude by showing the two integrated net semantics of a QS $M/M/2/3/4$ with arrival rate λ and service rate μ. This QS can be modeled as follows:

$$\begin{aligned}
QS_{M/M/2/3/4} &\triangleq Customers_4 \parallel_{\{a,l\}} (Queue \parallel_{\{d\}} Servers_2) \\
Customers_4 &\triangleq C \parallel_\emptyset C \parallel_\emptyset C \parallel_\emptyset C \\
C &\triangleq <a, \lambda>.<l, *>.C \\
Queue &\triangleq <a, *>.<d, *>.Queue \\
Servers_2 &\triangleq S \parallel_\emptyset S \\
S &\triangleq <d, \infty_{1,1}>.<s, \mu>.<l, \infty_{1,1}>.S
\end{aligned}$$

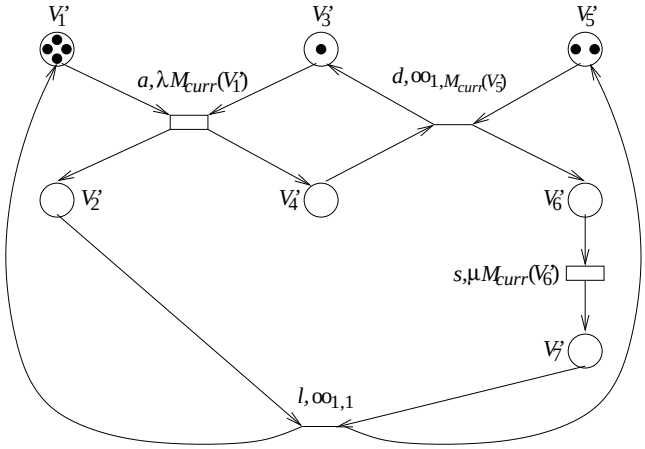
where a stands for arrive, l stands for leave, d stands for deliver, and s stands for serve. Note that, in order to keep the drawing of the two resulting nets reasonably simple as far as arcs are concerned, the behavior of C has been defined by considering only the arrival at and the departure from the system. A more precise definition would have included actions $<d, *>$ and $<s, *>$ in C and action types d and s in the first synchronization set. Since the customers are identical, the simplification does not alter the functional and performance properties of the whole system.

$\mathcal{N}_{\rm loc}[\![QS_{M/M/2/3/4}]\!]$ is reported in Fig. 6.1(a), while $\mathcal{N}_{\rm lab,o}[\![QS_{M/M/2/3/4}]\!]$ is reported in Fig. 6.1(b). Note that the marking dependence of rates has been indicated only for those transitions whose rate value varies according to the number of tokens in the preset. The names of places have been shortened as follows:

- $V_1 = (((C \parallel_\emptyset id) \parallel_\emptyset id) \parallel_\emptyset id) \parallel_{\{a,l\}} id$,
  $V_2 = (((id \parallel_\emptyset C) \parallel_\emptyset id) \parallel_\emptyset id) \parallel_{\{a,l\}} id$,
  $V_3 = ((id \parallel_\emptyset C) \parallel_\emptyset id) \parallel_{\{a,l\}} id$,
  $V_4 = (id \parallel_\emptyset C) \parallel_{\{a,l\}} id$,
  $V_5 = (((<l, *>.C \parallel_\emptyset id) \parallel_\emptyset id) \parallel_\emptyset id) \parallel_{\{a,l\}} id$,
  $V_6 = (((id \parallel_\emptyset <l, *>.C) \parallel_\emptyset id) \parallel_\emptyset id) \parallel_{\{a,l\}} id$,
  $V_7 = ((id \parallel_\emptyset <l, *>.C) \parallel_\emptyset id) \parallel_{\{a,l\}} id$,
  $V_8 = (id \parallel_\emptyset <l, *>.C) \parallel_{\{a,l\}} id$,
  $V_9 = id \parallel_{\{a,l\}} (Queue \parallel_{\{d\}} id)$,



(a) Location oriented net semantics



(a) Label oriented net semantics

Figure 6.1: Integrated net semantics of $QS_{M/M/2/3/4}$

$V_{10} = id \parallel_{\{a,l\}} (<d,*>.Queue \parallel_{\{d\}} id)$,
$V_{11} = id \parallel_{\{a,l\}} (id \parallel_{\{d\}} (S \parallel_{\emptyset} id))$,
$V_{12} = id \parallel_{\{a,l\}} (id \parallel_{\{d\}} (id \parallel_{\emptyset} S))$,
$V_{13} = id \parallel_{\{a,l\}} (id \parallel_{\{d\}} (<s,\mu>.<l,\infty_{1,1}>.S \parallel_{\emptyset} id))$,
$V_{14} = id \parallel_{\{a,l\}} (id \parallel_{\{d\}} (id \parallel_{\emptyset} <s,\mu>.<l,\infty_{1,1}>.S))$,
$V_{15} = id \parallel_{\{a,l\}} (id \parallel_{\{d\}} (<l,\infty_{1,1}>.S \parallel_{\emptyset} id))$,
$V_{16} = id \parallel_{\{a,l\}} (id \parallel_{\{d\}} (id \parallel_{\emptyset} <l,\infty_{1,1}>.S))$;

- $V'_1 = \emptyset C', V'_2 = \emptyset <l,*>.C'$ with $C' \equiv C\langle\{a,l\} := \{a_{new,l}, l_{new,l}\}\rangle$,
  $V'_3 = \emptyset Q', V'_4 = \emptyset <d,*>.Q'$ with $Q' \equiv Q\langle\{a,d\} := \{a_{new,r}, d_{new,l}\}\rangle$,
  $V'_5 = \emptyset S', V'_6 = \emptyset <s,\mu>.<l,\infty_{1,1}>.S', V'_7 = \emptyset <l,\infty_{1,1}>.S'$ with $S' \equiv S\langle\{d,l\} := \{d_{new,r}, l_{new,r}\}\rangle$.

As expected, the integrated label oriented net semantics is smaller than the integrated location oriented net semantics. The latter has 16 places, 16 transitions, and 60 arcs, whereas the former net has only 7 places, 4 transitions, and 14 arcs.

Chapter 7

Extending EMPA with Rewards

The problem we address in this chapter is how to specify and compute performance measures such as throughput, utilization, and average response time for systems modeled with EMPA terms. As seen in Chap. 2, a commonly used technique to derive performance measures for MCs is based on rewards [108]. Given this technique, we realize that the actual problem we are faced with is how to specify rewards in our algebraic setting. This means that we have to find out a formal way to define rewards at a suitable level of abstraction without having to manually scan the state space of the MC to assign rewards to states and transitions.

The method we propose in this chapter consists of specifying rewards directly within the actions forming the algebraic terms. As we shall see, such an algebra based method is relatively expressive and easy to use, as several performance measures can be specified without the knowledge of any extra formalism, and has a low computational cost, because rewards specified within the actions of a given term are assigned to states and transitions during the construction of the MC underlying that term. Most importantly, it allows for reasoning about algebraic descriptions with a performance measure sensitive equivalence [14, 16]. For the sake of simplicity, the algebra based method will be developed by considering the specification of a single reward based performance measure, but it can be easily extended to deal with several measures simultaneously.

This chapter is organized as follows. In Sect. 7.1 we investigate the problem of specifying rewards in the case of stochastically timed process algebras. In Sect. 7.2 we extend the theory developed for EMPA in order to take rewards into account according to the algebra based method. In Sect. 7.3 we present an application of the resulting stochastically timed process algebra which is concerned with the performability modeling of a QS where failures and repairs can occur. Finally, in Sect. 7.4 we compare the algebra based method with an alternative, logic based method.

7.1 Specifying Performance Measures with Rewards

When using a formal description technique to represent the performance aspects of a system, the stochastic process associated with the underlying performance model is not directly provided by the system designer but automatically derived from the more abstract formal description of the system in order to ease the task of the designer. As a consequence, rewards should not be defined at the level of the stochastic process but at the level of the formal description, and then automatically inherited by the stochastic process.

This is exactly what happens for well known and tool supported extensions of the Petri net formalism such as reward generalized stochastic Petri nets [47] and stochastic activity networks with rewards [163]. In both cases, yield rewards (also called rate rewards) are naturally associated with net markings, while bonus rewards (also called impulse rewards) are naturally associated with net transitions/activities.

Developing a method to assign rewards to algebraic terms in order to support the high level specification of performance measures causes a relevant difference between stochastically timed Petri nets and stochastically timed process algebras to emerge. While for nets it is natural to associate yield rewards with net markings and bonus rewards with net transitions, in the case of algebras it is natural to associate bonus rewards with actions but it is not clear how yield rewards should be specified because the concept of state is somehow implicit.

The method we propose in this chapter for stochastically timed process algebras consists of specifying both yield and bonus rewards within actions: $<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>$, where both yield reward $\tilde{r}_1$ and bonus reward $\tilde{r}_2$ are unspecified only in case of passive actions. As far as yield rewards are concerned, we assume that the yield reward earned by a state is the sum of the yield rewards of the actions it can execute (additivity assumption). Since rewards are specified in the process algebraic description, we call the proposed method algebra based. We now assess its adequacy w.r.t. the following criteria: expressive power, ease of use, computational cost, and equational characterization.

As far as the first two criteria (expressive power and ease of use) are concerned, we observe that the algebra based method achieves a reasonable balance in that it allows many of the more frequent performance measures to be specified in a relatively easy way, which in particular does not require the knowledge of any extra formalism to describe reward structures. As an example, we show how to specify for a QS $M/M/n/n$ with arrival rate λ and service rate μ several stationary performance measures frequently occurring in practice such as those identified in [48]: rate type (e.g. throughput of a service center), counting type (e.g. mean number of customers waiting in a service center), delay type (e.g. mean response time experienced by customers in a service center), and percentage type (e.g. utilization of a service center). As seen in Ex. 5.4, the state oriented description of the QS at hand is given by:

$$
\begin{aligned}
QS^{so}_{M/M/n/n} &\triangleq Arrivals \parallel_{\{a\}} Servers_0 \\
Arrivals &\triangleq <a, \lambda>.Arrivals \\
Servers_0 &\triangleq <a, *>.Servers_1 \\
Servers_h &\triangleq <a, *>.Servers_{h+1} + <s, h \cdot \mu>.Servers_{h-1}, \quad 1 \leq h \leq n-1 \\
Servers_n &\triangleq <s, n \cdot \mu>.Servers_{n-1}
\end{aligned}
$$

whereas the resource oriented description is given by:

$$
\begin{aligned}
QS^{ro}_{M/M/n/n} &\triangleq Arrivals \parallel_{\{a\}} Servers \\
Arrivals &\triangleq <a, \lambda>.Arrivals \\
Servers &\triangleq S \parallel_{\emptyset} S \parallel_{\emptyset} \ldots \parallel_{\emptyset} S \\
S &\triangleq <a, *>.<s, \mu>.S
\end{aligned}
$$

where a stands for arrival of a customer and s stands for service of a customer. These two different descriptions represent the same system as in Ex. 5.4 we have shown that $QS^{so}_{M/M/n/n} \sim_{\text{EMB}} QS^{ro}_{M/M/n/n}$.

Let us compute for the QS above the mean number of customers in the system, which is the sum of the numbers of customers over all the states with each number weighted by the stationary probability of the corresponding state. According to formula (2.1), every state of the HCTMC underlying each of the two terms above must then be given a yield reward equal to the number of customers in that state. Such a number is the number of s actions executable in that state. Therefore, in the case of $QS^{so}_{M/M/n/n}$ we must replace every action of the form $<s, h \cdot \mu>$ with $<s, h \cdot \mu, h, 0>$, while in the case of $QS^{ro}_{M/M/n/n}$ every action of the form $<s, \mu>$ must be replaced with $<s, \mu, 1, 0>$ by virtue of the additivity assumption for yield rewards. All the other actions must be given zero (or unspecified if passive) rewards.

If we want to compute the throughput of the QS, defined as the mean number of customers served per time unit, we have to take into account the rate of actions having type s. In fact, the throughput is given by the service rate multiplied by the stationary probability of being in a state where service can be provided. As a consequence, in the case of $QS^{so}_{M/M/n/n}$ we must replace every action of the form $<s, h \cdot \mu>$ with $<s, h \cdot \mu, h \cdot \mu, 0>$ or equivalently $<s, h \cdot \mu, 0, 1>$, while in the case of $QS^{ro}_{M/M/n/n}$ we must replace every action of the form $<s, \mu>$ with $<s, \mu, \mu, 0>$ or equivalently $<s, \mu, 0, 1>$.

If we want to compute instead the mean response time of the QS, defined as the mean time spent by one customer in the service center, we can exploit Little's law [115] which states that the mean response time experienced by a customer is equal to the mean number of customers in the service center divided by the customer arrival rate. Therefore, in the case of $QS^{so}_{M/M/n/n}$ we must replace every action of the form $<s, h \cdot \mu>$ with $<s, h \cdot \mu, h/\lambda, 0>$, while in the case of $QS^{ro}_{M/M/n/n}$ we must replace every action of the form $<s, \mu>$ with $<s, \mu, 1/\lambda, 0>$.

Finally, if we want to compute the utilization of the QS, defined as the fraction of time during which servers are busy, we have to single out those states having an outgoing transition labeled with s, because the utilization is the sum of the stationary probabilities of such states. Thus, in the case of $QS^{so}_{M/M/n/n}$ we must replace every action of the form $<s, h \cdot \mu>$ with $<s, h \cdot \mu, 1, 0>$. However, in the case of $QS^{ro}_{M/M/n/n}$ the

algebra based method fails to determine the utilization due to the additivity assumption: the yield reward to associate with actions of the form $<s, \mu>$ would be the reciprocal of the number of transitions labeled with s leaving the same state. Since one of the two main objectives of the algebra based method is its ease of use, we prefer to keep the specification of rewards as simple as possible, i.e. just by means of numbers. Thus we avoid the introduction of arithmetical expressions involving particular functions such as the one determining the number of transitions of a given type leaving the same state. Incidentally, the inability to compute the utilization in the case of the resource oriented description should not come as a surprise, since this description is more suited to the determination of performance indices concerning a single server instead of the whole set of servers. As it turns out, it is quite easy to measure the utilization of a given server specified in $QS^{ro}_{M/M/n/n}$, whereas this is not possible for $QS^{so}_{M/M/n/n}$. This means that the style [187] used to describe a given system through an algebraic term is strongly related to the possibility of specifying certain performance measures through the algebra based method.

For the considered QS, the algebra based method also allows transient measures to be expressed according to formula (2.2). As an example, yield rewards can be used to measure the mean number of customers in the system at a given instant or the probability that a certain server is in use at a given instant, whereas bonus rewards can be employed to assess the frequency with which customers arrive or are served at a given instant.

The third criterion (computational cost) requires associating rewards with states and transitions to be not exceedingly expensive: in particular, a full scan of the state space should be avoided. As we shall see in Sect. 7.2.4, the algebra based method satisfies this requirement because rewards can be computed and assigned to states and transitions at semantic model construction time.

Finally, the fourth criterion (equational characterization) requires the method to allow process terms to be compositionally manipulated without altering their performance measures. This is an important feature. As an example, if one uses a measure insensitive equivalence to reduce the state space before evaluating the performance, there is the risk to merge together states which are different w.r.t. the measures of interest, thus resulting in wrong performance figures. In Sect. 7.2.5 we shall see that the algebra based method permits the definition of performance measure sensitive congruences over process terms. This is the other main objective of the algebra based method and constitutes its major advantage.

7.2 Algebraic Treatment of Rewards

In this section we incorporate rewards into EMPA according to the algebra based method outlined in the previous section: the resulting stochastically timed process algebra is called EMPA_r. For the sake of simplicity, the theory is developed for a single pair of rewards associated with each action. However, it may be defined for an arbitrary number of pairs of rewards associated with each action, which corresponds to considering several performance measures at once.

The structure of this section parallels the structure of Chap. 3, 5, and 6 in order to emphasize the extensions required for each part of the integrated approach of Fig. 1.1. This section ends with some compatibility results showing when EMPA_r reduces to EMPA.

7.2.1 Integrated Actions: Types, Rates, and Rewards

The change of the structure of EMPA actions is the only syntactic consequence of the adoption of the algebra based method, because the method itself consists of describing rewards directly within actions. An action of EMPA_r is a quadruple $<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>$ where $\tilde{r}_1$ is the yield reward and $\tilde{r}_2$ is the bonus reward. Since the performance characteristics of passive actions are completely unspecified, passive actions are assigned unspecified rewards denoted by $*$: the rewards of a passive action become fixed only upon synchronization with an active action of the same type. We thus denote by

$$Act_\text{r} = \{<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2> \in AType \times ARate \times AYReward \times ABReward \mid \tilde{\lambda} = * \iff \tilde{r}_1 = \tilde{r}_2 = *\}$$

the set of actions, where $AYReward = \mathbb{R} \cup \{*\}$ is the set of yield rewards and $ABReward = \mathbb{R} \cup \{*\}$ is the set of bonus rewards, ranged over by $\tilde{r}, \tilde{r}', \ldots$.

7.2.2 Syntax of Terms and Informal Semantics of Operators

The syntax of EMPA_r differs from that of EMPA only because of the presence of rewards within actions.

Definition 7.1 *The set $\mathcal{L}_\text{r}$ of process terms of EMPA_r is generated by the following syntax*

$$E ::= \underline{0} \mid <a, \tilde{\lambda}, \tilde{r}, \tilde{r}>.E \mid E/L \mid E[\varphi] \mid E + E \mid E \parallel_S E \mid A$$

where $L, S \subseteq AType - \{\tau\}$. We denote by $\mathcal{G}_\text{r}$ the set of closed and guarded terms of $\mathcal{L}_\text{r}$. ■

The meaning of the operators is essentially unchanged. As far as the action prefix operator is concerned, the execution of action $<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>$ makes the corresponding state earn additional yield reward $\tilde{r}_1$ and the related transition gain bonus reward $\tilde{r}_2$. As far as the parallel composition operator is concerned, in case of synchronization the rewards of the resulting action are given by the rewards of the only active action involved in the synchronization (or $*$ if only passive actions are involved) possibly normalized. More precisely, because of the additivity assumption, yield rewards are subject to the same normalization as rates. Bonus rewards, instead, are not subject to normalization at all to avoid an underestimation of the performance measures. The reason is that, as seen in Chap. 2, in the calculation of the performance measures the bonus reward of a transition is multiplied by a factor which is proportional to the rate of the transition itself, hence normalizing the rates is all we have to do.

7.2.3 Execution Policy

The execution policy and the semantic model for EMPA_r are the same as those for EMPA. The only difference is that the labels of the LTS underlying EMPA_r carry rewards. Because of the additivity assumption, the yield reward earned during the sojourn in a state is the sum of the yield rewards of the actions it can execute, while the bonus reward gained by traversing a transition is the bonus reward of the action labeling that transition.

7.2.4 Integrated and Projected Interleaving Semantics

The integrated interleaving semantics for EMPA_r is generated by the rules shown in Table 7.1. Domains and codomains of auxiliary functions are similar to those of the auxiliary functions in Table 3.1 with Act replaced by Act_r, except for auxiliary functions *Aggr*, *Norm*, and *Split* where each occurrence of $ARate$ must be replaced by $ARate \cup AYReward$ ranged over by $\tilde{t}, \tilde{t}', \ldots$.

Definition 7.2 *The integrated interleaving semantics of $E \in \mathcal{G}_\text{r}$ is the LTS*

$$\mathcal{I}_\text{r}[\![E]\!] = (S_{E,\mathcal{I}_\text{r}}, Act_\text{r}, \longrightarrow_{E,\mathcal{I}_\text{r}}, E)$$

where:

- *$S_{E,\mathcal{I}_\text{r}}$ is the least subset of $\mathcal{G}_\text{r}$ such that:*
  - *$E \in S_{E,\mathcal{I}_\text{r}}$;*
  - *if $E_1 \in S_{E,\mathcal{I}_\text{r}}$ and $E_1 \stackrel{a,\tilde{\lambda},\tilde{r}_1,\tilde{r}_2}{\longrightarrow}_\text{r} E_2$, then $E_2 \in S_{E,\mathcal{I}_\text{r}}$.*
- *$\longrightarrow_{E,\mathcal{I}_\text{r}}$ is the restriction of $\longrightarrow_\text{r}$ to $S_{E,\mathcal{I}_\text{r}} \times Act_\text{r} \times S_{E,\mathcal{I}_\text{r}}$.* ■

Definition 7.3 *$E \in \mathcal{G}_\text{r}$ is performance closed if and only if $\mathcal{I}_\text{r}[\![E]\!]$ does not contain passive transitions. We denote by $\mathcal{E}_\text{r}$ the set of performance closed terms of $\mathcal{G}_\text{r}$.* ■

Definition 7.4 *The functional semantics of $E \in \mathcal{G}_\text{r}$ is the LTS*

$$\mathcal{F}_\text{r}[\![E]\!] = (S_{E,\mathcal{F}_\text{r}}, AType, \longrightarrow_{E,\mathcal{F}_\text{r}}, E)$$

where:

- *$S_{E,\mathcal{F}_\text{r}} = S_{E,\mathcal{I}_\text{r}}$.*
- *$\longrightarrow_{E,\mathcal{F}_\text{r}}$ is the restriction of $\longrightarrow_{E,\mathcal{I}_\text{r}}$ to $S_{E,\mathcal{F}_\text{r}} \times AType \times S_{E,\mathcal{F}_\text{r}}$.* ■

The performance semantics of a performance closed term is a HCTMRC, a HDTMRC, or a HSMRC depending on whether the underlying integrated interleaving semantic model has only exponentially timed transitions, only immediate transitions, or both. The performance semantics, hereafter called Markovian semantics, can thus be represented as a pr-LTS.

$$\frac{(<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>, E') \in Melt_{\mathrm{r}}(Select_{\mathrm{r}}(PM_{\mathrm{r}}(E)))}{E \xrightarrow{a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2}_{\mathrm{r}} E'}$$

$$\begin{aligned}
PM_{\mathrm{r}}(\underline{0}) &= \emptyset \\
PM_{\mathrm{r}}(<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>.E) &= \{\!| \, (<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>, E) \, |\!\} \\
PM_{\mathrm{r}}(E/L) &= \{\!| \, (<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>, E'/L) \mid (<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>, E') \in PM_{\mathrm{r}}(E) \wedge a \notin L \, |\!\} \oplus \\
&\quad \{\!| \, (<\tau, \tilde{\lambda}_1, \tilde{r}_1, \tilde{r}_2>, E'/L) \mid \exists a \in L. \, (<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>, E') \in PM_{\mathrm{r}}(E) \, |\!\} \\
PM_{\mathrm{r}}(E[\varphi]) &= \{\!| \, (<\varphi(a), \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>, E'[\varphi]) \mid (<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>, E') \in PM_{\mathrm{r}}(E) \, |\!\} \\
PM_{\mathrm{r}}(E_1 + E_2) &= PM_{\mathrm{r}}(E_1) \oplus PM_{\mathrm{r}}(E_2) \\
PM_{\mathrm{r}}(E_1 \parallel_S E_2) &= \{\!| \, (<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>, E_1' \parallel_S E_2) \mid a \notin S \wedge (<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>, E_1') \in PM_{\mathrm{r}}(E_1) \, |\!\} \oplus \\
&\quad \{\!| \, (<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>, E_1 \parallel_S E_2') \mid a \notin S \wedge (<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>, E_2') \in PM_{\mathrm{r}}(E_2) \, |\!\} \oplus \\
&\quad \{\!| \, (<a, \tilde{\gamma}, \tilde{r}_1, \tilde{r}_2>, E_1' \parallel_S E_2') \mid a \in S \wedge \exists \tilde{\lambda}_1, \tilde{\lambda}_2, \tilde{r}_{1,1}, \tilde{r}_{1,2}, \tilde{r}_{2,1}, \tilde{r}_{2,2}. \\
&\qquad\qquad (<a, \tilde{\lambda}_1, \tilde{r}_{1,1}, \tilde{r}_{1,2}>, E_1') \in PM_{\mathrm{r}}(E_1) \wedge \\
&\qquad\qquad (<a, \tilde{\lambda}_2, \tilde{r}_{2,1}, \tilde{r}_{2,2}>, E_2') \in PM_{\mathrm{r}}(E_2) \wedge \\
&\qquad\qquad \tilde{\gamma} = Norm(a, \tilde{\lambda}_1, \tilde{\lambda}_2, PM_{\mathrm{r}}(E_1), PM_{\mathrm{r}}(E_2)) \wedge \\
&\qquad\qquad \tilde{r}_1 = Norm(a, \tilde{r}_{1,1}, \tilde{r}_{2,1}, PM_{\mathrm{r}}(E_1), PM_{\mathrm{r}}(E_2)) \wedge \\
&\qquad\qquad \tilde{r}_2 = \begin{cases} \tilde{r}_{1,2} & \text{if } \tilde{r}_{2,2} = * \\ \tilde{r}_{2,2} & \text{if } \tilde{r}_{1,2} = * \end{cases} \, |\!\} \\
PM_{\mathrm{r}}(A) &= PM_{\mathrm{r}}(E) \qquad \text{if } A \triangleq E
\end{aligned}$$

$$\begin{aligned}
Select_{\mathrm{r}}(PM) = \{\!| \, (<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>, E) \in PM \mid & \\
& \forall (<a', \tilde{\lambda}', \tilde{r}_1', \tilde{r}_2'>, E') \in PM. \, PL_{\mathrm{r}}(<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>) \geq PL_{\mathrm{r}}(<a', \tilde{\lambda}', \tilde{r}_1', \tilde{r}_2'>) \vee \\
& PL_{\mathrm{r}}(<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>) = -1 \, |\!\}
\end{aligned}$$

$$PL_{\mathrm{r}}(<a, *, *, *>) = -1 \qquad PL_{\mathrm{r}}(<a, \lambda, r_1, r_2>) = 0 \qquad PL_{\mathrm{r}}(<a, \infty_{l,w}, r_1, r_2>) = l$$

$$\begin{aligned}
Melt_{\mathrm{r}}(PM) = \{(<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>, E) \mid & \\
& \exists \tilde{\lambda}', \tilde{r}_1'. \, (<a, \tilde{\lambda}', \tilde{r}_1', \tilde{r}_2>, E) \in PM \wedge \\
& \tilde{\lambda} = Aggr\{\!| \, \tilde{\lambda}'' \mid \exists \tilde{r}_1''. \, (<a, \tilde{\lambda}'', \tilde{r}_1'', \tilde{r}_2>, E) \in PM \wedge PL_{\mathrm{r}}(<a, \tilde{\lambda}'', \tilde{r}_1'', \tilde{r}_2>) = PL_{\mathrm{r}}(<a, \tilde{\lambda}', \tilde{r}_1', \tilde{r}_2>) \, |\!\} \wedge \\
& \tilde{r}_1 = Aggr\{\!| \, \tilde{r}_1'' \mid \exists \tilde{\lambda}''. \, (<a, \tilde{\lambda}'', \tilde{r}_1'', \tilde{r}_2>, E) \in PM \wedge PL_{\mathrm{r}}(<a, \tilde{\lambda}'', \tilde{r}_1'', \tilde{r}_2>) = PL_{\mathrm{r}}(<a, \tilde{\lambda}', \tilde{r}_1', \tilde{r}_2>) \, |\!\}\}
\end{aligned}$$

$$* \; Aggr \; * = * \qquad t_1 \; Aggr \; t_2 = t_1 + t_2 \qquad \infty_{l,w_1} \; Aggr \; \infty_{l,w_2} = \infty_{l,w_1+w_2}$$

$$Norm(a, \tilde{t}_1, \tilde{t}_2, PM_1, PM_2) = \begin{cases} Split(\tilde{t}_1, 1/(\pi_1(PM_2))(<a, *, *, *>)) & \text{if } \tilde{t}_2 = * \\ Split(\tilde{t}_2, 1/(\pi_1(PM_1))(<a, *, *, *>)) & \text{if } \tilde{t}_1 = * \end{cases}$$

$$Split(*, p) = * \qquad Split(t, p) = t \cdot p \qquad Split(\infty_{l,w}, p) = \infty_{l, w \cdot p}$$

Table 7.1: Inductive rules for $\mathrm{EMPA_r}$ integrated interleaving semantics

Definition 7.5 *The Markovian semantics of $E \in \mathcal{E}_{\rm r}$ is the pr-LTS*

$$\mathcal{M}_{\rm r}[\![E]\!] = (S_{E,\mathcal{M}_{\rm r}}, \mathbb{R}_+ \times \mathbb{R}, \longrightarrow_{E,\mathcal{M}_{\rm r}}, P_{E,\mathcal{M}_{\rm r}}, H_{E,\mathcal{M}_{\rm r}}, Y_{E,\mathcal{M}_{\rm r}})$$

where:

- $S_{E,\mathcal{M}_{\rm r}} = S_{E,\mathcal{I}_{\rm r}}$.
- $\longrightarrow_{E,\mathcal{M}_{\rm r}}$ *is the least subset of* $S_{E,\mathcal{M}_{\rm r}} \times (\mathbb{R}_+ \times \mathbb{R}) \times S_{E,\mathcal{M}_{\rm r}}$ *such that:*
  - $F \stackrel{\lambda,r}{\longrightarrow}_{E,\mathcal{M}_{\rm r}} F'$ *whenever* $\lambda = \sum\{\!|\, \mu \mid \exists a, r_1.\, F \stackrel{a,\mu,r_1,r}{\longrightarrow}_{E,\mathcal{I}_{\rm r}} F' \,|\!\}$;
  - $F \stackrel{p,r}{\longrightarrow}_{E,\mathcal{M}_{\rm r}} F'$ *whenever* $p = \sum\{\!|\, w \mid \exists a, l, r_1.\, F \stackrel{a,\infty_{l,w},r_1,r}{\longrightarrow}_{E,\mathcal{I}_{\rm r}} F' \,|\!\} / \sum\{\!|\, w \mid \exists a, l, r_1, r_2, F''.\, F \stackrel{a,\infty_{l,w},r_1,r_2}{\longrightarrow}_{E,\mathcal{I}_{\rm r}} F'' \,|\!\}$.
- $P_{E,\mathcal{M}_{\rm r}} : S_{E,\mathcal{M}_{\rm r}} \longrightarrow \mathbb{R}_{[0,1]}$, $P_{E,\mathcal{M}_{\rm r}}(F) = \begin{cases} 1 & \text{if } F \equiv E \\ 0 & \text{if } F \not\equiv E \end{cases}$
- $H_{E,\mathcal{M}_{\rm r}} : S_{E,\mathcal{M}_{\rm r}} \longrightarrow \{v, t, a\}$, $H_{E,\mathcal{M}_{\rm r}}(F) = \begin{cases} v & \text{if } \exists a, l, w, r_1, r_2, F'.\, F \stackrel{a,\infty_{l,w},r_1,r_2}{\longrightarrow}_{E,\mathcal{I}_{\rm r}} F' \\ t & \text{if } \exists a, \lambda, r_1, r_2, F'.\, F \stackrel{a,\lambda,r_1,r_2}{\longrightarrow}_{E,\mathcal{I}_{\rm r}} F' \\ a & \text{if } \nexists a, \tilde\lambda, r_1, r_2, F'.\, F \stackrel{a,\tilde\lambda,\tilde r_1,\tilde r_2}{\longrightarrow}_{E,\mathcal{I}_{\rm r}} F' \end{cases}$
- $Y_{E,\mathcal{M}_{\rm r}} : S_{E,\mathcal{M}_{\rm r}} \longrightarrow \mathbb{R}$, $Y_{E,\mathcal{M}_{\rm r}}(F) = \sum\{\!|\, r_1 \mid \exists a, \tilde\lambda, r_2, F'.\, F \stackrel{a,\tilde\lambda,r_1,r_2}{\longrightarrow}_{E,\mathcal{I}_{\rm r}} F' \,|\!\}$. ■

7.2.5 Integrated Equivalence

The integrated equivalence for $\text{EMPA}_{\rm r}$ is developed along the same line as $\sim_{\rm EMB}$. For the time being, let us consider yield rewards only. Suppose we extend the definition of $\sim_{\rm EMB}$ to $\mathcal{G}_{\rm r}$ in the expected way, i.e. by ignoring the presence of rewards within actions. Then, for the sake of simplicity, one may be tempted to define the integrated equivalence for $\text{EMPA}_{\rm r}$ by relating $E_1, E_2 \in \mathcal{G}_{\rm r}$ whenever $E_1 \sim_{\rm EMB} E_2$ and they have the same yield reward, intended as the sum of the yield rewards of the actions they can execute. However, as the examples below demonstrate, in this way one would fail both to capture an equivalence correctly accounting for the performance measure at hand and to obtain a congruence.

Example 7.1 Consider terms

$$\begin{aligned} A &\stackrel{\triangle}{=} <a, \lambda, r>.<b, \mu, r_1>.A \\ B &\stackrel{\triangle}{=} <a, \lambda, r>.<b, \mu, r_2>.B \end{aligned}$$

where $r_1 \neq r_2$. Then $A \sim_{\rm EMB} B$ and A and B have the same yield reward r, but if we solve the two underlying HCTMRCs we obtain two different values of the performance measure we are interested in: $r \cdot \mu/(\lambda+\mu) + r_1 \cdot \lambda/(\lambda+\mu)$ for the system modeled by A, $r \cdot \mu/(\lambda+\mu) + r_2 \cdot \lambda/(\lambda+\mu)$ for the system modeled by B. ■

Example 7.2 Consider terms

$$\begin{aligned} E_1 &\equiv <a, \lambda, r_1>.\underline{0} + <b, \mu, r_2>.\underline{0} \\ E_2 &\equiv <a, \lambda, r_2>.\underline{0} + <b, \mu, r_1>.\underline{0} \end{aligned}$$

where $r_1 \neq r_2$. Then $E_1 \sim_{\mathrm{EMB}} E_2$ and E_1 and E_2 have the same yield reward $r_1 + r_2$, but e.g. $E_1 \parallel_{\{b\}} \underline{0}$ has yield reward r_1 while $E_2 \parallel_{\{b\}} \underline{0}$ has yield reward r_2, so the congruence property would not be achieved. ■

The first example shows that, to define a sensible equivalence, the two conditions to meet, i.e. equivalence according to $\sim_{\mathrm{EMB}}$ and identical yield rewards, should be more intimately connected, e.g. by defining a new equivalence $\sim'_{\mathrm{EMB}}$ whose quotient set is obtained from the quotient set of $\sim_{\mathrm{EMB}}$ by further relating those terms having the same yield reward (which would result in $A \not\sim'_{\mathrm{EMB}} B$ as far as the first example is concerned). The second example shows that, to achieve the congruence property, we need to establish an even stronger connection between the two conditions than that provided by $\sim'_{\mathrm{EMB}}$. We cannot treat the reward separately from the rest of the action: equality for rewards must be recursively checked in the bisimilarity clause.

Below we demonstrate that it is really easy to extend the definition of $\sim_{\mathrm{EMB}}$ in such a way that both objectives are achieved. We preliminarily observe that, according to the formulas of Sect. 2.2.6 we have that yield rewards must be handled in the same way as rates, because of the additivity assumption. Bonus rewards of actions of the same type and priority level, instead, cannot be summed up, as this would result in an overestimation of the specified performance measures. The reason is that, in the calculation of the performance measures, the bonus reward of a transition is multiplied by a factor which is proportional to the rate of the transition itself, hence summing rates up is all we have to do. As an example, it must hold that

$$<a, \lambda_1, r_{1,1}, r>.E + <a, \lambda_2, r_{2,1}, r>.E \sim <a, \lambda_1 + \lambda_2, r_{1,1} + r_{2,1}, r>.E$$

Definition 7.6 *We define partial function* $RY : \mathcal{G}_{\mathrm{r}} \times AType \times APLev \times ABReward \times \mathcal{P}(\mathcal{G}_{\mathrm{r}}) \not\to ARate \times AYReward$ *by*

$$\begin{aligned} RY(E, a, l, \tilde{r}, C) \;=\; & (Aggr\{\!|\, \tilde{\lambda} \mid \exists \tilde{r}_1.\, \exists E' \in C.\, E \xrightarrow{a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}}_{\mathrm{r}} E' \wedge PL_{\mathrm{r}}(<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}>) = l \,|\!\}, \\ & Aggr\{\!|\, \tilde{r}_1 \mid \exists \tilde{\lambda}.\, \exists E' \in C.\, E \xrightarrow{a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}}_{\mathrm{r}} E' \wedge PL_{\mathrm{r}}(<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}>) = l \,|\!\}) \end{aligned}$$ ■

Definition 7.7 *An equivalence relation* $\mathcal{B} \subseteq \mathcal{G}_{\mathrm{r}} \times \mathcal{G}_{\mathrm{r}}$ *is a strong extended Markovian reward bisimulation (strong EMRB) if and only if, whenever* $(E_1, E_2) \in \mathcal{B}$*, then for all* $a \in AType$*,* $l \in APLev$*,* $\tilde{r} \in ABReward$*, and* $C \in \mathcal{G}_{\mathrm{r}}/\mathcal{B}$

$$RY(E_1, a, l, \tilde{r}, C) = RY(E_2, a, l, \tilde{r}, C)$$

In this case we say that E_1 *and* E_2 *are strongly extended Markovian reward bisimilar (strongly EMRB).* ■

The proofs of the following results are smooth adaptations of those of the corresponding results in Chap. 5.

Proposition 7.1 *Let $\sim_{\rm EMRB}$ be the union of all the strong EMRBs. Then $\sim_{\rm EMRB}$ is the largest strong EMRB.* ■

Definition 7.8 *We call $\sim_{\rm EMRB}$ the strong extended Markovian reward bisimulation equivalence (strong EMRBE). We say that $E_1, E_2 \in \mathcal{G}_{\rm r}$ are strongly extended Markovian reward bisimulation equivalent (strongly EMRBE) if and only if $E_1 \sim_{\rm EMRB} E_2$.* ■

Necessary and sufficient conditions similar to Prop. 5.2 and 5.3, where *Rate* is replaced by RY, hold for $\sim_{\rm EMRB}$.

Theorem 7.1 *$\sim_{\rm EMRB}$ is a congruence over $\mathcal{G}_{\rm r,\Theta,rnd}$.* ■

Theorem 7.2 *Let $E_1, E_2 \in \mathcal{G}_{\rm r}$. If $E_1 \sim_{\rm EMRB} E_2$ then $\mathcal{F}_{\rm r}[\![E_1]\!]$ is bisimilar to $\mathcal{F}_{\rm r}[\![E_2]\!]$.* ■

Theorem 7.3 *Let $E_1, E_2 \in \mathcal{E}_{\rm r}$. If $E_1 \sim_{\rm EMRB} E_2$ then $\mathcal{M}_{\rm r}[\![E_1]\!]$ is pr-bisimilar to $\mathcal{M}_{\rm r}[\![E_2]\!]$.* ■

A coarsest congruence result similar to Thm. 5.4 holds for $\sim_{\rm EMRB}$.

Theorem 7.4 *Let $\mathcal{A}_{\rm r}$ be the set of axioms in Table 7.2. The deductive system $Ded(\mathcal{A}_{\rm r})$ is sound and complete for $\sim_{\rm EMRB}$ over $\mathcal{G}_{\rm r,\Theta,rnd,nrec}$.* ■

Finally, an algorithm similar to that of Table 5.2, where *Rate* is replaced by RY, can be used to check for $\sim_{\rm EMRB}$ and a result similar to Cor. 5.1, where p-isomorphism is replaced by pr-isomorphism, holds for $\sim_{\rm EMRB}$.

We conclude by observing that, according to the formulas of Sect. 2.2.6, $\sim_{\rm EMRB}$ could be defined in such a way that a more aggregating version of axiom $\mathcal{A}_{\rm r,4}$ holds or only rewards of the same kind are used. More precisely, one may want to obtain the following equalities

$$\begin{array}{rcl}
<a,\lambda_1,r_{1,1},r_{1,2}>.E + <a,\lambda_2,r_{2,1},r_{2,2}>.E & \sim & <a,\lambda_1+\lambda_2,r_{1,1}+r_{2,1},\frac{\lambda_1}{\lambda_1+\lambda_2}\cdot r_{1,2}+\frac{\lambda_2}{\lambda_1+\lambda_2}\cdot r_{2,2}>.E \\
<a,\infty_{l,w_1},r_{1,1},r_{1,2}>.E + <a,\infty_{l,w_2},r_{2,1},r_{2,2}>.E & \sim & <a,\infty_{l,w_1+w_2},r_{1,1}+r_{2,1},\frac{w_1}{w_1+w_2}\cdot r_{1,2}+\frac{w_2}{w_1+w_2}\cdot r_{2,2}>.E \\
<a,\lambda_1,r_{1,1},r_{1,2}>.E + <a,\lambda_2,r_{2,1},r_{2,2}>.E & \sim & <a,\lambda_1+\lambda_2,r_{1,1}+r_{2,1}+\lambda_1\cdot r_{1,2}+\lambda_2\cdot r_{2,2},0>.E \\
<a,\infty_{l,w_1},r_{1,1},r_{1,2}>.E + <a,\infty_{l,w_2},r_{2,1},r_{2,2}>.E & \sim & <a,\infty_{l,w_1+w_2},r_{1,1}+r_{2,1}+\frac{w_1}{w_1+w_2}\cdot r_{1,2}+\frac{w_2}{w_1+w_2}\cdot r_{2,2},0>.E
\end{array}$$

Unfortunately, setting up the definition of $\sim_{\rm EMRB}$ in such a way that all the rules above are obeyed leads to a violation of the congruence property in the discrete time case.

7.2.6 Integrated Net Semantics

The integrated net semantics $\mathcal{N}_{\rm loc,r}$ and $\mathcal{N}_{\rm lab,r}$ for EMPA$_{\rm r}$ are defined in the same way as the integrated net semantics for EMPA, with yield rewards treated in the same way as rates w.r.t. marking dependency, aggregations, and normalizations. Since for GSPNs with rewards we have that yield rewards are associated with net markings and bonus rewards are associated with net transitions, given a term of EMPA$_{\rm r}$ the

$$(\mathcal{A}_{\mathrm{r},1}) \quad (E_1 + E_2) + E_3 = E_1 + (E_2 + E_3)$$

$$(\mathcal{A}_{\mathrm{r},2}) \quad E_1 + E_2 = E_2 + E_1$$

$$(\mathcal{A}_{\mathrm{r},3}) \quad E + \underline{0} = E$$

$$(\mathcal{A}_{\mathrm{r},4}) \quad <a, \tilde{\lambda}_1, \tilde{r}_{1,1}, \tilde{r}>.E + <a, \tilde{\lambda}_2, \tilde{r}_{2,1}, \tilde{r}>.E = <a, \tilde{\lambda}_1 \; Aggr \; \tilde{\lambda}_2, \tilde{r}_{1,1} \; Aggr \; \tilde{r}_{2,1}, \tilde{r}>.E$$
$$\text{if } PL_{\mathrm{r}}(<a, \tilde{\lambda}_1, \tilde{r}_{1,1}, \tilde{r}>) = PL_{\mathrm{r}}(<a, \tilde{\lambda}_2, \tilde{r}_{2,1}, \tilde{r}>)$$

$$(\mathcal{A}_{\mathrm{r},5}) \quad \underline{0}/L = \underline{0}$$

$$(\mathcal{A}_{\mathrm{r},6}) \quad (<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>.E)/L = \begin{cases} <a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>.E/L & \text{if } a \notin L \\ <\tau, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>.E/L & \text{if } a \in L \end{cases}$$

$$(\mathcal{A}_{\mathrm{r},7}) \quad (E_1 + E_2)/L = E_1/L + E_2/L$$

$$(\mathcal{A}_{\mathrm{r},8}) \quad \underline{0}[\varphi] = \underline{0}$$

$$(\mathcal{A}_{\mathrm{r},9}) \quad (<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>.E)[\varphi] = <\varphi(a), \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>.E[\varphi]$$

$$(\mathcal{A}_{\mathrm{r},10}) \quad (E_1 + E_2)[\varphi] = E_1[\varphi] + E_2[\varphi]$$

$$(\mathcal{A}_{\mathrm{r},11}) \quad \Theta(\underline{0}) = \underline{0}$$

$$(\mathcal{A}_{\mathrm{r},12}) \quad \Theta(\sum_{i \in I} <a_i, \tilde{\lambda}_i, \tilde{r}_{i,1}, \tilde{r}_{i,2}>.E_i) = \sum_{j \in J} <a_j, \tilde{\lambda}_j, \tilde{r}_{j,1}, \tilde{r}_{j,2}>.\Theta(E_j)$$
$$\text{where } J = \{i \in I \mid \tilde{\lambda}_i = * \vee \forall h \in I.\, PL_{\mathrm{r}}(<a_i, \tilde{\lambda}_i, \tilde{r}_{i,1}, \tilde{r}_{i,2}>) \geq PL_{\mathrm{r}}(<a_h, \tilde{\lambda}_h, \tilde{r}_{h,1}, \tilde{r}_{h,2}>)\}$$

$$(\mathcal{A}_{\mathrm{r},13}) \quad \sum_{i \in I_1} <a_i, \tilde{\lambda}_i, \tilde{r}_{i,1}, \tilde{r}_{i,2}>.E_i \parallel_S \sum_{i \in I_2} <a_i, \tilde{\lambda}_i, \tilde{r}_{i,1}, \tilde{r}_{i,2}>.E_i =$$
$$\sum_{j \in J_1} <a_j, \tilde{\lambda}_j, \tilde{r}_{j,1}, \tilde{r}_{j,2}>.(E_j \parallel_S \sum_{i \in I_2} <a_i, \tilde{\lambda}_i, \tilde{r}_{i,1}, \tilde{r}_{i,2}>.E_i) +$$
$$\sum_{j \in J_2} <a_j, \tilde{\lambda}_j, \tilde{r}_{j,1}, \tilde{r}_{j,2}>.(\sum_{i \in I_1} <a_i, \tilde{\lambda}_i, \tilde{r}_{i,1}, \tilde{r}_{i,2}>.E_i \parallel_S E_j) +$$
$$\sum_{k \in K_1} \sum_{h \in H_k} <a_k, \mathit{Split}(\tilde{\lambda}_k, 1/n_k), \mathit{Split}(\tilde{r}_{k,1}, 1/n_k), \tilde{r}_{k,2}>.(E_k \parallel_S E_h) +$$
$$\sum_{k \in K_2} \sum_{h \in H_k} <a_k, \mathit{Split}(\tilde{\lambda}_k, 1/n_k), \mathit{Split}(\tilde{r}_{k,1}, 1/n_k), \tilde{r}_{k,2}>.(E_h \parallel_S E_k)$$
$$\text{where } I_1 \cap I_2 = \emptyset$$
$$J_1 = \{i \in I_1 \mid a_i \notin S\}$$
$$J_2 = \{i \in I_2 \mid a_i \notin S\}$$
$$K_1 = \{k \in I_1 \mid \exists h \in I_2.\, a_h = a_k \in S \wedge \tilde{\lambda}_h = *\}$$
$$K_2 = \{k \in I_2 \mid \exists h \in I_1.\, a_h = a_k \in S \wedge \tilde{\lambda}_h = *\}$$
$$H_k = \begin{cases} \{h \in I_2 \mid a_h = a_k \wedge \tilde{\lambda}_h = *\} & \text{if } k \in K_1 \\ \{h \in I_1 \mid a_h = a_k \wedge \tilde{\lambda}_h = *\} & \text{if } k \in K_2 \end{cases}$$
$$n_k = |H_k|$$

Table 7.2: Axioms for $\sim_{\mathrm{EMRB}}$

corresponding GSPN with rewards is defined in such a way that the bonus reward of a net transition is the bonus reward of the action labeling the net transition, while the yield reward of a net marking is the sum of the yield rewards of the actions labeling the net transitions enabled in the net marking. Retrievability results similar to those of Chap. 6 can be established also in the presence of rewards.

7.2.7 Compatibility Results

We conclude by showing some results which prove that EMPA_r is somehow a natural extension of EMPA. This means that the introduction of rewards does not change the structure of the semantic models and that there is a relationship between $\sim_{\text{EMRB}}$ and $\sim_{\text{EMB}}$.

Definition 7.9 *We define function* $plain : \mathcal{G}_\text{r} \longrightarrow \mathcal{G}$ *by structural induction as follows:*

$$\begin{aligned}
plain(\underline{0}) &= \underline{0} \\
plain(<a, \tilde{\lambda}, \tilde{r}_1, \tilde{r}_2>.E) &= <a, \tilde{\lambda}>.plain(E) \\
plain(E/L) &= plain(E)/L \\
plain(E[\varphi]) &= plain(E)[\varphi] \\
plain(E_1 + E_2) &= plain(E_1) + plain(E_2) \\
plain(E_1 \parallel_S E_2) &= plain(E_1) \parallel_S plain(E_2) \\
plain(A) &= A
\end{aligned}$$

■

Proposition 7.2 *Let* $E \in \mathcal{G}_\text{r}$. *Then* $\mathcal{I}_\text{r}[\![E]\!]$ *is isomorphic to* $\mathcal{I}[\![plain(E)]\!]$ *up to rewards and the merging of transitions of the same type and priority level differing for their bonus rewards.* ■

Proposition 7.3 *Let* $E \in \mathcal{G}_\text{r}$. *Then* $\mathcal{F}_\text{r}[\![E]\!]$ *is isomorphic to* $\mathcal{F}[\![plain(E)]\!]$. ■

Proposition 7.4 *Let* $E \in \mathcal{E}_\text{r}$. *Then* $\mathcal{M}_\text{r}[\![E]\!]$ *is p-isomorphic to* $\mathcal{M}[\![plain(E)]\!]$ *up to rewards and the merging of transitions differing for their bonus rewards.* ■

Proposition 7.5 *Let* $E \in \mathcal{G}_\text{r}$. *Then* $\mathcal{N}_{\text{loc,r}}[\![E]\!]$ *is isomorphic to* $\mathcal{N}_{\text{loc}}[\![plain(E)]\!]$ *up to rewards and the merging of transitions of the same type and priority level differing for their bonus rewards.* ■

Proposition 7.6 *Let* $E \in \mathcal{G}_\text{r}$. *Then* $\mathcal{N}_{\text{lab,r}}[\![E]\!]$ *is isomorphic to* $\mathcal{N}_{\text{lab}}[\![plain(E)]\!]$ *up to rewards and the merging of transitions of the same type and priority level differing for their bonus rewards.* ■

Proposition 7.7 *Let* $E_1, E_2 \in \mathcal{G}_\text{r}$. *If* $E_1 \sim_{\text{EMRB}} E_2$ *then* $plain(E_1) \sim_{\text{EMB}} plain(E_2)$. ■

The following example shows that the vice versa of the last result is not true in general. We would like to point out that this is not inconsistent with $\sim_{\text{EMB}}$. The purpose of $\sim_{\text{EMB}}$ is to relate terms describing systems having the same functional and performance properties. If $plain(E_1) \sim_{\text{EMB}} plain(E_2)$ but $E_1 \not\sim_{\text{EMRB}} E_2$, this simply means that we are evaluating two different performance indices for E_1 and E_2.

Example 7.3 Consider terms

$$A \triangleq <a, \lambda, 1, 0>.<b, \mu, 0, 0>.A$$
$$B \triangleq <a, \lambda, 0, 0>.<b, \mu, 1, 0>.B$$

Then $plain(A) \sim_{\mathrm{EMB}} plain(B)$ but $A \not\sim_{\mathrm{EMRB}} B$. If we view a and b as the transmission over two different channels, then by means of A we can compute the utilization of the former channel, whereas by means of B we can compute the utilization of the latter channel. ■

Corollary 7.1 *Let $E_1, E_2 \in \mathcal{G}_{\mathrm{r}}$ be such that each of their active actions has yield and bonus rewards equal to zero. Then $E_1 \sim_{\mathrm{EMRB}} E_2$ if and only if $plain(E_1) \sim_{\mathrm{EMB}} plain(E_2)$.* ■

7.3 Performability Modeling of a Queueing System

In this section we report an example of application of $\sim_{\mathrm{EMRB}}$. The performance of computing and communicating systems is often degradable because internal or external faults can reduce the quality of the delivered service even though that service remains proper according to its specification. It is therefore important to measure their ability to perform, or *performability*, at different accomplishment levels specifying the extent to which a system is faulty, i.e. which resources are faulty and, among them, which ones have failed, which ones are being recovered, and which ones contain latent faults [131]. From the modeling point of view, we would like to be able to describe both performance and dependability within a single model. On the other hand, this results in problems from the analysis standpoint such as largeness, caused by the presence of several resources working in parallel possibly at different operational levels, and stiffness, originated from the large difference of performance related event rates and rare failure related event rates implying numerical instability. As recognized in [186], this leads to a natural hierarchy of models: a higher level dependability model and a set of as many lower level performance models as there are states in the higher level model. This stems from the fact that the rate of occurrence of failure and repair events is smaller than the rate of occurrence of performance related events, hence the system achieves a quasi steady state w.r.t. the performance related events between successive occurrences of failure or repair events [59]. This means that the system can be characterized by weighing these quasi steady state performance measures by the probabilities of the corresponding states of the higher level model.

We show that $\sim_{\mathrm{EMRB}}$ can be used while building the hierarchy of models of [186] to correctly manipulate the lower level models, so that they are translated into equivalent models whose solution is well known. Let us consider a QS $M/M/n/n+q$ whose servers can fail and be repaired, where the arrival rate is λ, the service rates are μ_i $(1 \leq i \leq n)$, the failure rates are ϕ_i $(1 \leq i \leq n)$, and the repair rates are ρ_i $(1 \leq i \leq n)$, with ϕ_i and ρ_i much smaller than λ and μ_i:

$$
\begin{aligned}
FRQS_{M/M/n/n+q} &\triangleq Arrivals \parallel_{\{a\}} (Queue_0 \parallel_D Servers) \\
D &= \{d_i \mid 1 \leq i \leq n\} \\
Arrivals &\triangleq <a, \lambda, 0, 0>.Arrivals \\
Queue_0 &\triangleq <a, *, *, *>.Queue_1 \\
Queue_h &\triangleq <a, *, *, *>.Queue_{h+1} + \sum_{i=1}^{n} <d_i, *, *, *>.Queue_{h-1}, \quad 1 \leq h \leq q-1 \\
Queue_q &\triangleq \sum_{i=1}^{n} <d_i, *, *, *>.Queue_{q-1} \\
Servers &\triangleq S_1 \parallel_\emptyset S_2 \parallel_\emptyset \ldots \parallel_\emptyset S_n \\
S_i &\triangleq <d_i, \infty_{1,1}, 0, 0>.S'_i, \quad 1 \leq i \leq n \\
S'_i &\triangleq <s_i, \mu_i, 0, 1>.S_i + \\
&\quad\ <f_i, \phi_i, 0, 0>.<r_i, \rho_i, 0, 0>.S'_i, \quad 1 \leq i \leq n
\end{aligned}
$$

where a stands for arrival, d_i stands for delivery, s_i stands for service, f_i stands for failure, and r_i stands for repair. Note that actions s_i have been given bonus reward 1, since we are interested in computing the throughput of the system, i.e. the number of customers served per unit of time.

Now, since the monolithic model above causes largeness and stiffness problems during its analysis, we build the hierarchy of models proposed in [186] to facilitate the analysis. First we recognize that the higher level dependability model, i.e. the failure-repair model, can be represented as follows:

$$
\begin{aligned}
FR &\triangleq FR_1 \parallel_\emptyset FR_2 \parallel_\emptyset \ldots \parallel_\emptyset FR_n \\
FR_i &\triangleq <f_i, \phi_i, 0, 0>.<r_i, \rho_i, 0, 0>.FR_i, \quad 1 \leq i \leq n
\end{aligned}
$$

and can be efficiently studied since it trivially admits a product form solution, i.e. the stationary probability of a given state of $\mathcal{M}_{\mathrm{r}}[\![FR]\!]$ is the product of the stationary probabilities of the related states of $\mathcal{M}_{\mathrm{r}}[\![FR_i]\!]$ for $1 \leq i \leq n$. Each state of FR determines the set I of operational servers and the set J of failed servers, with $I \cup J = \{1, \ldots, n\}$ and $I \cap J = \emptyset$, so the corresponding lower level performance model is given by:

$$
\begin{aligned}
FRQS_{M/M/n/n+q,I,J} &\triangleq FRQS_{M/M/n/n+q} \parallel_{D_J \cup F_I} \underline{0} \\
D_J &= \{d_j \mid j \in J\} \\
F_I &= \{f_i \mid i \in I\}
\end{aligned}
$$

The effect of the synchronization with $\underline{0}$ is that only operational servers can receive customers (D_J) and these servers cannot fail (F_I). It is easily seen that $FRQS_{M/M/n/n+q,I,J}$ is equivalent via $\sim_{\mathrm{EMRB}}$ to $QS_{M/M/|I|/|I|+q,I}$ described below:

$$
\begin{aligned}
QS_{M/M/|I|/|I|+q,I} &\triangleq Arrivals \parallel_{\{a\}} (Queue_{I,0} \parallel_{D_I} Servers_I) \\
D_I &= \{d_i \mid i \in I\} \\
Arrivals &\triangleq <a, \lambda, 0, 0>.Arrivals \\
Queue_{I,0} &\triangleq <a, *, *, *>.Queue_{I,1} \\
Queue_{I,h} &\triangleq <a, *, *, *>.Queue_{I,h+1} + \sum_{i \in I} <d_i, *, *, *>.Queue_{I,h-1}, \quad 1 \leq h \leq q-1 \\
Queue_{I,q} &\triangleq \sum_{i \in I} <d_i, *, *, *>.Queue_{I,q-1} \\
Servers_I &\triangleq S_{i_1} \parallel_\emptyset S_{i_2} \parallel_\emptyset \ldots \parallel_\emptyset S_{i_{|I|}}, \quad \{i_1, i_2, \ldots, i_{|I|}\} = I \\
S_i &\triangleq <d_i, \infty_{1,1}, 0, 0>.<s_i, \mu_i, 0, 1>.S_i, \quad i \in I
\end{aligned}
$$

Since the servers of $QS_{M/M/|I|/|I|+q,I}$ are subject neither to failures nor to repairs, the manipulations above preserve the properties of the system under study and give rise to a model whose solution is well known in the literature [115]. The overall throughput is finally obtained as the weighted sum of the throughputs of every lower level model, where the product form stationary probabilities of the higher level model are used as weights.

7.4 Comparison with a Logic Based Method

Prior to the algebra based method, a different method was proposed in [48]. Such a method was inspired by the preliminary work in [83], where it is proposed to use a temporal logic formula to partition the semantic model of an algebraic description in such a way that each part exhibits or not a particular behavior formalized through the logic formula itself. The idea is to define a reward structure as a function of such a partition, which associates a unique (yield) reward to all the states of the same class.

In [48] this logic based method is further elaborated on. The process of specifying performance measures is split into two stages. The first stage consists of defining a reward specification, which is a pair composed of a Hennessy-Milner logic formula (see Sect. 2.1.4) and an expression: every state satisfying the modal logic formula is assigned as a yield reward the value of the expression, which may consist of the usual arithmetic operators applied to real numbers, action rates, and special variables storing previously or currently assigned rewards. The second stage, instead, consists of defining a reward attachment that determines at which process derivatives a particular reward specification is evaluated. The logic based method seems to be quite adequate because temporal logic formulas make assertions about changing state, so they result in a suitable link between process terms, which essentially describe system behavior through actions thus leaving the concept of state implicit, and yield rewards, which are associated with states. Such a method addresses only yield rewards and stationary measures.

If we compare the algebra based method and the logic based method w.r.t. the four criteria of Sect. 7.1, we see that in general the algebra based method is less powerful than the logic based method, as rewards are simply expressed as real numbers in the former method and particular behaviors formalizable through

logic formulas cannot be captured (see e.g. the specification of the utilization for $QS^{ro}_{M/M/n/n}$ in Sect. 7.1), but easier to learn and use, as it does not require the knowledge of any extra formalism to specify rewards (consider e.g. the logic formula necessary to specify the mean number of customers for $QS^{ro}_{M/M/n/n}$ in Sect. 7.1). The logic based method is more time consuming than the algebra based method, as it would require in principle an additional scan of the state space in order to check states against the modal logic formulas in order to attach yield rewards: fortunately, model checking on the fly should be possible. Finally, an equational characterization is possible in the case of the algebra based method but not in the case of the logic based method, hence with the latter method a compositional, performance measure preserving term manipulation cannot be conducted. In fact, with the former method, if two terms are equivalent (according to $\sim_{\mathrm{EMRB}}$) then they have the same functional and performance properties and can be compositionally manipulated without altering the performance measure expressed by the included rewards. With the latter method, instead, it is not possible to define an integrated equivalence like $\sim_{\mathrm{EMRB}}$. What it could be done is defining an integrated equivalence like $\sim'_{\mathrm{EMB}}$ outlined in Sect. 7.2.5, but we would end up with an equivalence which is not a congruence. Because of the results about the relationship between (functional) bisimulation equivalence and Hennessy-Milner logic formula satisfiability [82], the logic based method only guarantees that if two terms are related by $\sim_{\mathrm{EMB}}$ then equivalent states get the same yield reward, hence the performance index under study has the same value for the two terms. The converse does not hold: if two terms satisfy a given set of Hennessy-Milner logic formulas, then the two terms may be (functionally) bisimulation equivalent but not necessarily equivalent according to $\sim_{\mathrm{EMB}}$, which means that the performance index for the two terms may be different.

Example 7.4 Consider the EMPA terms

$$\begin{aligned} A &\triangleq <a,\lambda>.<b,\mu>.A \\ B &\triangleq <a,\mu>.<b,\lambda>.B \end{aligned}$$

where $\lambda \neq \mu$, and the logic based reward specification

$$\langle a\rangle tt \Longrightarrow r$$

which associates reward $r \in \mathbb{R}$ with each state where an action of type a can be executed. Both A and B satisfy the modal logic formula above, and indeed $\mathcal{F}[\![A]\!]$ is bisimulation equivalent to $\mathcal{F}[\![B]\!]$, but $A \not\sim_{\mathrm{EMB}} B$ and the value of the performance index at hand is $r \cdot \mu/(\lambda+\mu)$ for A and $r \cdot \lambda/(\lambda+\mu)$ for B. ■

Recently, in [49] the lack of equational characterization for the logic based method has been remedied. This has been accomplished by using a Markovian modal logic inspired by the probabilistic modal logic of [117], instead of the Hennessy-Milner logic, and by showing that two terms satisfy the same Markovian modal logic formulas if and only if they are Markovian bisimulation equivalent. As a consequence, Markovian bisimulation equivalent states get the same reward according to the approach of [49]. In this respect, the algebra based method turns out to be more flexible, as it allows different rewards to be associated with Markovian bisimulation equivalent states, hence the need for the previously presented extension of

the Markovian bisimulation equivalence that takes rewards into account. Additionally, in [49] the ease of use of the logic based method has been enhanced by proposing a high level language for enquiring about the stationary performance characteristics possessed by a process term. Such a language, whose formal underpinning is constituted by the Markovian modal logic (which thus becomes transparent to the user), is based on the combination of the standard mathematical notation (arithmetical, relational and logical operators as well as probability), a notation based on the Markovian bisimulation equivalence which is useful to focus queries directly on states, and a notation expressing the potential to perform an action of a given type.

Chapter 8

Extending EMPA with Value Passing

When using EMPA, there is a large class of systems that cannot be dealt with. Such a class is composed of those systems where data plays a fundamental role, so it is not possible to abstract from data in their modeling process. Think e.g. of a system that receives messages and undertakes different activities depending on the contents of the received messages. The problem with EMPA is that, unlike other Markovian process algebras such as MLOTOS [93], data is not considered at all. The purpose of this chapter is to suitably extend EMPA with value passing in order to be able to model and analyze such systems. Conceptually, this is achieved in three steps. First, we extend the syntax in the usual way by adding input and output actions, conditional operators, and constant parameters. Second, we propose as semantic models the symbolic transition graphs with lookahead assignment, which are an improvement of the symbolic models proposed in [81, 122, 121], and (unlike [93]) we define symbolic semantic rules mapping terms onto the symbolic models. Such symbolic semantic rules do not make any assumption on variable names, are correct w.r.t. both the usual concrete semantic rules and the assignment evaluation order, and produce compact symbolic models. Third, we exhibit a suitable integrated equivalence for the symbolic models [12, 17].

From the performance modeling standpoint, a relevant application of value passing based expressions is the capability of dealing with systems where generally distributed durations come into play. To support this, discrete time algebraic models must be used, i.e. only immediate and passive actions can occur. In such models, generally distributed durations are represented through value passing based expressions, the occurrence times of timed events (sampled according to the previously mentioned expressions) are stored into appropriate variables, and suitable clock variables are employed in order to detect when it is time to execute certain timed events.

This chapter is organized as follows. In Sect. 8.1 we propose a symbolic approach to value passing based on lookahead. In Sect. 8.2 we extend the theory developed for EMPA in order to deal with value passing according to the symbolic approach based on lookahead. In Sect. 8.3 we show how to model generally distributed durations by means of value passing based expressions.

8.1 An Improved Symbolic Approach to Value Passing

In order to introduce value passing, following [133] we modify the syntax in the usual way by first introducing actions which can read or write values. As a consequence, beside *unstructured actions* of the form $<a, \tilde{\lambda}>$, we have *input actions* of the form $<a?(\underline{x}), \tilde{\lambda}>$, where $\underline{x}$ is a vector of variables, as well as *output actions* of the form $<a!(\underline{e}), \tilde{\lambda}>$, where $\underline{e}$ is a vector of expressions. Second, a new operator is necessary to reflect the influence of data on the execution. This is a *conditional operator* of the form "if _ then _ else _" where the first operand is a boolean expression whereas the second and the third operands are terms. Third, we must keep track of the data we are interested in. This is achieved by means of *parametrized constant definitions* of the form $A(\underline{x}) \stackrel{\Delta}{=} E$ where $\underline{x}$ is a vector of variables. It is worth noting that by means of the syntactic ingredients above, it is possible to give a finite algebraic representation of systems that would require infinitely many constant defining equations in EMPA. For example, an unbounded buffer, where interarrival times are exponentially distributed with rate λ and interdeparture times are exponentially distributed with rate μ, can be described with the following single constant defining equation:

$$\begin{aligned} \mathit{Buffer}(i) \stackrel{\Delta}{=}\ & \text{if } (i = 0) \text{ then} \\ & \quad <\mathit{arrive}, \lambda>.\mathit{Buffer}(i+1) \\ & \text{else} \\ & \quad <\mathit{arrive}, \lambda>.\mathit{Buffer}(i+1) + <\mathit{leave}, \mu>.\mathit{Buffer}(i-1) \end{aligned}$$

Now the question becomes how to define the semantics for such a value passing, stochastically timed calculus. The proposal in [133] of expanding a term of the form $<a?(x), \tilde{\lambda}>.E$ according to all the possible values v that variable x can take on, which results in term $\sum_v <a_v, \tilde{\lambda}>.E\{v/x\}$ with $E\{v/x\}$ denoting the term obtained from E by replacing each free occurrence of x with v, should be avoided for two reasons. The first (practical) one is that such an expansion causes infinitely many transitions leaving the same state to arise whenever the value domain is infinite, thereby making automatic analysis impossible. The second (theoretical) one is that such an expansion may wrongly increase the speed at which activities are carried out. If we consider e.g. term $<a?(x), \lambda>.E$ where $x \in \{1, 2\}$, then the expansion results in term $<a_1, \lambda>.E\{1/x\} + <a_2, \lambda>.E\{2/x\}$, but the two terms are not equivalent from the performance point of view because the mean sojourn time for the former term is $1/\lambda$ whereas for the latter is $1/(2 \cdot \lambda)$. This second drawback could be overcome by normalizing the rates in the expanded term, but this would only work for finite domain variables.

To solve the problems above, the integrated interleaving semantic model may be given by a LTS in the form of a symbolic transition graph [81], as it enables a finite representation of a large class of process descriptions, so that conventional analysis techniques apply.

Let us preliminarily introduce some syntactic categories that we shall use in the sequel. Let *Val* be a set of *values* ranged over by v, *Var* be a set of *variables* ranged over by x, y, z, *Exp* be a set of *expressions* over $\mathit{Val} \cup \mathit{Var}$ ranged over by e, and *BExp* be a set of boolean expressions in *Exp* ranged over by β. Let also

$Eval = \{\rho : Var \longrightarrow Val\}$ be a set of *evaluations* and $Sub = (Var \times Exp)^*$ be a set of *substitutions* ranged over by σ, which will be sometimes denoted by $\underline{x} := \underline{e}$ to specify *assignments*. We shall write $\rho[\underline{x} \mapsto \underline{v}]$ to denote the evaluation which differs from ρ only in that it maps $\underline{x}$ to $\underline{v}$, $\underline{e}\sigma$ to denote the expression obtained from $\underline{e}$ by applying σ to the variables occurring in it, and $\sigma_1 \triangleright \sigma_2$ to denote the concatenation of σ_1 and σ_2. Moreover, let $Act_{\rm vp}$ be a set of unstructured, input and output actions whose type is ranged over by α. We define the sets of free and bound variables occurring in an action type as follows: $fv(a!(\underline{e})) = fv(\underline{e})$, $bv(a?(\underline{x})) = \underline{x}$, and $fv(\alpha) = bv(\alpha) = \emptyset$ in all the other cases.

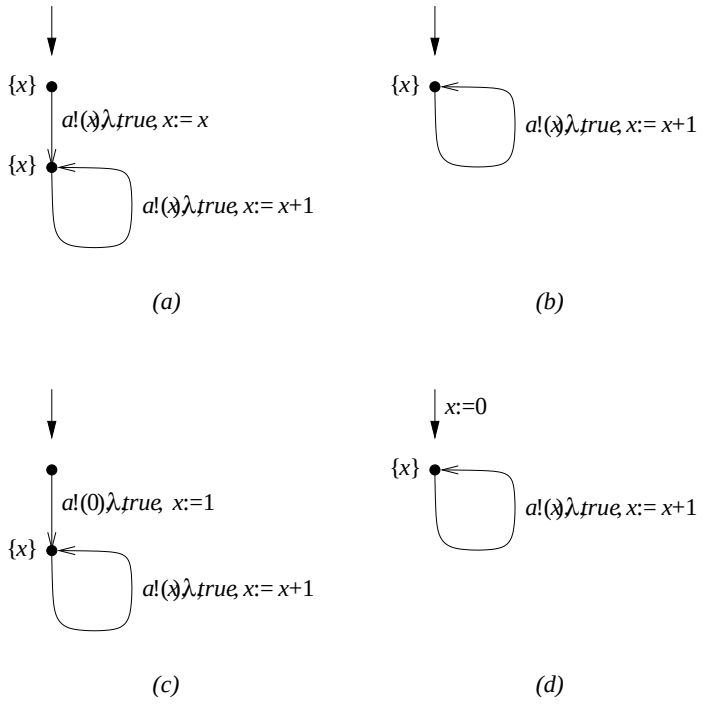


Figure 8.1: Symbolic models for $A(x) \stackrel{\Delta}{=} <a!(x), \lambda>.A(x+1)$ and $A(0)$

A *symbolic transition graph* [81] (STG for short) is a tuple

$$(S, \longrightarrow, s_0; Var, Act_{\rm vp} \times BExp)$$

where S is a set of states, with $s_0 \in S$ designated as the initial one, each labeled with the set $fv(s) \subseteq Var$ of its free variables, while $\longrightarrow \subseteq S \times (Act_{\rm vp} \times BExp) \times S$ is the transition relation. The meaning of transition $s_1 \stackrel{\alpha,\beta}{\longrightarrow} s_2$, where $fv(\alpha) \cup fv(\beta) \subseteq fv(s_1)$ and $fv(s_2) \subseteq fv(s_1) \cup bv(\alpha)$, is that s_2 can be reached from s_1 by performing α whenever β is satisfied at s_1. As can be seen, the model is called symbolic because the values of the variables occurring in input and output actions do not get evaluated. This keeps the model finite for many terms in the case of infinite value domains.

As pointed out in [122], still many intuitively simple processes cannot be represented as finite STGs. As an example, consider

$$A(x) \stackrel{\Delta}{=} <a!(x), \lambda>.A(x+1)$$

The STG for $A(x)$ has countably many states representing $A(x+n)$, $n \in \mathbb{N}$, with transitions $A(x+n) \xrightarrow{a!(x+n),\lambda,true} A(x+n+1)$ for all $n \in \mathbb{N}$. As a solution to this problem, in [122] it is proposed to associate an assignment with each transition. A *symbolic transition graph with assignment* (STGA for short) is a tuple

$$(S, \longrightarrow, s_0; Var, Act_{\text{vp}} \times BExp \times Sub)$$

where $\longrightarrow \subseteq S \times (Act_{\text{vp}} \times BExp \times Sub) \times S$. The meaning of transition $s_1 \xrightarrow{\alpha,\beta,\sigma} s_2$ with $\sigma \equiv \underline{x} := \underline{e}$, where $fv(\alpha) \subseteq \underline{x}$ and $fv(\beta) \cup fv(\underline{e}) \subseteq fv(s_1)$ and $fv(s_2) \subseteq \underline{x} \cup bv(\alpha)$, is the same as that of $s_1 \xrightarrow{\alpha,\beta} s_2$ where additionally the free variables $\underline{x}$ at s_2 are assigned the values of $\underline{e}$ evaluated at s_1 before executing α. Since $fv(\alpha) \subseteq \underline{x}$, σ is used to update the variables both in α and in s_2. As an example, the STGA for $A(x)$ above is shown in Fig. 8.1(a).

As observed in [121], a symbolic model for $A(x)$ like that in Fig. 8.1(a) is somewhat strange, as one would expect a more intuitive and compact symbolic model like that in Fig. 8.1(b). To achieve this, in [121] a variant of STGA (which we shall denote by STGA') is proposed, in which the meaning of transition $s_1 \xrightarrow{\alpha,\beta,\sigma} s_2$ with $\sigma \equiv \underline{x} := \underline{e}$, where $fv(\alpha) \cup fv(\beta) \cup fv(\underline{e}) \subseteq fv(s_1)$ and $fv(s_2) \subseteq fv(s_1) \cup \underline{x} \cup bv(\alpha)$, is the same as for STGA with the difference that σ is evaluated after executing α, hence it no longer applies to the variables in α but only to the variables in s_2.

In [121] it is also observed that some simple processes still have a strange representation with STGA'. As an example, see the STGA' for $A(0)$ in Fig. 8.1(c). To overcome this drawback, in this chapter we propose symbolic transition graphs with lookahead assignment (STGLA for short). A STGLA is a tuple

$$(S, \longrightarrow, s_0, \sigma_0; Var, Act_{\text{vp}} \times BExp \times Sub)$$

where $(S, \longrightarrow, s_0; Var, Act_{\text{vp}} \times BExp \times Sub)$ is a STGA' and $\sigma_0 \equiv \underline{x}_0 := \underline{v}_0 \in Sub$ is the initial assignment with $\underline{v}_0$ vector of values of Val and $\underline{x}_0 \subseteq fv(\sigma_0)$. The STGLA for $A(0)$ is depicted in Fig. 8.1(d).

This symbolic model, which we introduced in a preliminary work [12] independently of [121], combines the idea of evaluating assignments after executing actions with the idea of having an initial assignment for the initial state. From the point of view of an algebraic description, this can be seen as an attempt to keep symbolic the initial term of the description as well as every derivative term before generating its outgoing transitions, hence the terminology lookahead assignment. By keeping a term symbolic we just mean that the variables occurring in that term are never instantiated.

8.2 Semantic Treatment of Lookahead Based Value Passing

In this section we define the semantics for a value passing extension of EMPA we call EMPA_{vp} using STGLA as semantic models. Such symbolic semantic rules do not make any assumption on variable names, are correct

w.r.t. both the usual concrete semantic rules and the assignment evaluation order, and produce compact STGLA. The structure of this section parallels the structure of Chap. 3 and 5 in order to emphasize the extensions required for each part of EMPA. This section ends with some compatibility results showing when EMPA_{vp} reduces to EMPA.

8.2.1 Integrated Actions

The major modification here concerns action types. Let $AName = AName_{\text{U}} \cup AName_{\text{IO}}$ be a set of *action names* (ranged over by a), where $AName_{\text{U}}$ is a set of *unstructured action names* (including τ) and $AName_{\text{IO}}$ is a set of *input/output action names*, such that $AName_{\text{U}} \cap AName_{\text{IO}} = \emptyset$. Then the set of action types is $AType_{\text{vp}} = AType_{\text{U}} \cup AType_{\text{I}} \cup AType_{\text{O}}$ (ranged over by α) where $AType_{\text{U}} = AName_{\text{U}}$ is a set of *unstructured action types*, $AType_{\text{I}} = \{a?(\underline{x}) \mid a \in AName_{\text{IO}} \wedge \underline{x} \in \underline{Var}\}$ is a set of *input action types*, and $AType_{\text{O}} = \{a!(\underline{e}) \mid a \in AName_{\text{IO}} \wedge \underline{e} \in \underline{Exp}\}$ is a set of *output action types*. We also define function $name : AType_{\text{vp}} \longrightarrow AName$ by letting $name(a) = a$ for $a \in AName_{\text{U}}$ and $name(a?(\underline{x})) = name(a!(\underline{e})) = a$ for $a \in AName_{\text{IO}}$. Finally, we let

$$Act_{\text{vp}} = \{<\alpha, \tilde{\lambda}> \in AType_{\text{vp}} \times ARate \mid \alpha \in AType_{\text{I}} \Longrightarrow \tilde{\lambda} = *\}$$

to enforce the fact that input actions must be passive. This complies with the master-slaves synchronization discipline, in the sense that it allows several input actions to receive the same value carried by a single output action.

8.2.2 Syntax of Terms and Informal Semantics of Operators

Let $ANRFun = \{\varphi : AName \longrightarrow AName \mid \varphi(\tau) = \tau \wedge \varphi(AName_{\text{U}} - \{\tau\}) \subseteq AName_{\text{U}} - \{\tau\} \wedge \varphi(AName_{\text{IO}}) \subseteq AName_{\text{IO}}\}$ be a set of *action name relabeling functions*. For the sake of convenience, we lift $ANRFun$ to the set $ATRFun_{\text{vp}}$ of action type relabeling functions by preserving action type parameters.

Definition 8.1 *The set $\mathcal{L}_{\text{vp}}$ of process terms of $EMPA_{\text{vp}}$ is generated by the following syntax*

$$E ::= \sum_{i \in I} <\alpha_i, \tilde{\lambda}_i>.E_i \mid \text{if } (\beta, p) \text{ then } E \text{ else } E \mid E/L \mid E[\varphi] \mid E \parallel_S E \mid A(\underline{e})$$

where I is a finite set of indices, $p \in \mathbb{R}_{]0,1[}$, and $L, S \subseteq AName - \{\tau\}$. We denote by $\mathcal{G}_{\text{vp}}$ the set of closed and guarded terms of $\mathcal{L}_{\text{vp}}$. ■

Some comments about the operators are now in order. First of all, we observe that, unlike EMPA, we have a *guarded alternative composition operator* instead of an alternative composition operator separated from an action prefix operator. This causes the summands of the alternative compositions to be guarded, which simplifies the treatment of the lookahead assignments without exceedingly reducing the modeling power, as will be explained in Sect. 8.2.4. Term $\sum_{i \in I} <\alpha_i, \tilde{\lambda}_i>.E_i$ is denoted by $\underline{0}$ whenever $I = \emptyset$.

Likewise, the *conditional operator*, which is new w.r.t. EMPA, is interpreted in a nonstandard way: *if* (β, p) *then* E_1 *else* E_2 is viewed as the internal, immediate guarded alternative composition $<\tau, \infty_{\omega,p}>.E_1 + <\tau, \infty_{\omega,1-p}>.E_2$ governed by β. The new priority level ω is introduced in order to cause the resolution of conditional choices to take precedence over any other action (passive ones included). We thus define $APLev_{\rm vp} = APLev \cup \{\omega\}$, $Inf_{\rm vp} = Inf \cup \{\infty_{\omega,w} \mid w \in \mathbb{R}_+\}$, $ARate_{\rm vp} = \mathbb{R}_+ \cup Inf_{\rm vp} \cup \{*\}$, and $Act'_{\rm vp} = \{<\alpha, \tilde{\lambda}> \in AType_{\rm vp} \times ARate_{\rm vp} \mid \alpha \in AType_{\rm I} \Longrightarrow \tilde{\lambda} = *\}$. Note that condition (β, p) is composed of a functional part given by boolean expression β as well as a performance part given by probability p. The former part and its negation are used as boolean guards for the two transitions leaving the state associated with the term above. The latter part expresses the probability that β is satisfied and is introduced to allow for the construction of the performance model of every term containing occurrences of the conditional operator, which is in its classical form a source of pure nondeterminism. Our interpretation of the conditional operator, besides being convenient from the performance viewpoint as p can be naturally attached to immediate actions, makes explicit possible conditional deadlocks due to synchronization constraints as e.g. in terms like

$$(\text{if } (x = 1, 0.5) \text{ then } <a, \lambda>.E_1 \text{ else } <b, \mu>.E_2) \parallel_{\{a,b\}} (\text{if } (y = 2, 0.5) \text{ then } <a, *>.F_1 \text{ else } <b, *>.F_2)$$

and simplifies the treatment of lookahead assignments as we shall see in Sect. 8.2.4.

As far as the parallel composition operator is concerned, in an n-way synchronization either all the participating actions are unstructured or at most one of the participating actions is an output action and all the others are (passive) input actions. The synchronization among input/output actions is managed through assignments. More precisely, if $<a!(\underline{e}), \tilde{\lambda}>$ is synchronized with $<a?(\underline{x}), *>$, then assignment $\underline{x} := \underline{e}$ is generated and the resulting action is $<a!(\underline{e}), \tilde{\lambda}>$ up to rate normalization. In the case of $<a?(\underline{x}_1), *>$ synchronized with $<a?(\underline{x}_2), *>$, instead, assignment $\underline{x}_2 := \underline{x}_1$ is generated and the resulting action is $<a?(\underline{x}_1), *>$.

In order to avoid ambiguities, we assume the binary operators to be left associative and we introduce the following operator precedence relation: abstraction = relabeling > action prefix > alternative composition > parallel composition > conditional.

Finally, we assume that for each constant defining equation of the form $A(\underline{x}) \stackrel{\Delta}{=} E$ the set $fv(E)$ of variables occurring in expressions within E is contained in $\underline{x}$.

8.2.3 Execution Policy

The execution policy for $\text{EMPA}_{\rm vp}$ is the same as the execution policy for EMPA, with in addition the fact that the computation is driven by boolean guards and lookahead assignments. This is due to the fact that the semantic model for $\text{EMPA}_{\rm vp}$ is a compact STGLA.

Definition 8.2 *A symbolic transition graph with lookahead assignment (STGLA) is a tuple*

$$(S, \longrightarrow, s_0, \sigma_0; Var, Act'_{\rm vp} \times BExp \times Sub)$$

where:

- *S is a set whose elements are called states, each of which is labeled with the set $fv(s) \subseteq Var$ of its free variables.*
- *$\longrightarrow \subseteq S \times (Act'_{\mathrm{vp}} \times BExp \times Sub) \times S$ is called transition relation.*
- *$s_0 \in S$ is called the initial state.*
- *$\sigma_0 \equiv \underline{x}_0 := \underline{v}_0 \in Sub$, with $\underline{v}_0 \subseteq Val$ and $\underline{x}_0 \subseteq fv(\sigma_0)$, is called the initial assignment.*

The meaning of transition $s_1 \stackrel{\alpha,\tilde{\lambda},\beta,\sigma}{\longrightarrow} s_2$ with $\sigma \equiv \underline{x} := \underline{e}$, where $fv(\alpha) \cup fv(\beta) \cup fv(\underline{e}) \subseteq fv(s_1)$ and $fv(s_2) \subseteq fv(s_1) \cup \underline{x} \cup bv(\alpha)$, is that s_2 can be reached from s_1 by performing α whenever β evaluated at s_1 is satisfied and afterwards the free variables $\underline{x}$ of s_2 are assigned the values of $\underline{e}$ evaluated at s_1. ■

In the graphical representation of a STGLA, states are drawn as black dots with the appropriate labels and transitions are drawn as arrows between pairs of states with the appropriate labels; the initial state is pointed to by an arrow labeled with the initial assignment.

8.2.4 Integrated and Projected Interleaving Semantics

In this section we define a symbolic semantics that maps $\mathrm{EMPA}_{\mathrm{vp}}$ terms onto STGLA. The idea underlying such a semantics is to overcome the problems with infinite data domains by suitably modifying those rules of the usual concrete semantics [81] which would give rise to infinite branching and infinitely many states. Following the guidelines in [81, 122, 121], this is achieved by letting the input actions symbolic when they label the transitions (instead of making them concrete by replacing their parameters with every possible value) and by encoding the syntactical substitutions (that would be present in the concrete rules for the input action prefix and the constant invocation) through assignments labeling the transitions.

Before presenting the symbolic rules, some comments on the treatment of the variables are in order. When moving from the concrete to the symbolic semantics, the variables are no longer instantiated within the terms. Instead, they are given suitable values by means of assignments labeling the transitions. In order for such assignments to be correct, proper names should be given to the variables so that clashes are avoided. For this purpose, the symbolic rules that appeared in the literature [122, 121] assume disjoint name spaces for processes composed in parallel. Instead, the symbolic rules we shall introduce do not make any assumption on the variable names because of some technicalities (dot notation and localities) which are introduced in a way that is completely transparent to the user of $\mathrm{EMPA}_{\mathrm{vp}}$. This not only gives the freedom of choosing the same name for variables declared in different constants, but is necessary for easily automating in a software tool the correct management of the variables declared in several occurrences of the same constant composed in parallel.

To differentiate the variables with the same name that are declared in different constants, we use the *dot notation*. Therefore, an algebraic description is preprocessed in such a way that the name of every variable is

prefixed with the name of the constant in which the variable is declared followed by the dot. If we consider e.g.

$$\begin{array}{lcl} A(x) & \triangleq & <a!(x), \lambda>.(A_1(x+1) \parallel_\emptyset A_2(x+2)) \\ A_1(y) & \triangleq & <a_1!(y), \lambda_1>.A_1(y+1) \\ A_2(y) & \triangleq & <a_2!(y), \lambda_2>.A_2(y+1) \end{array}$$

where parameter y of A_1 must be initialized to $x+1$ while parameter y of A_2 must be initialized to $x+2$, we observe that no clash between the two parameters arise because the former is renamed $A_1.y$ while the latter is renamed $A_2.y$.

To differentiate the variables that are declared in different occurrences of the same constant composed in parallel, we further prefix the dotted variable names with *localities*. In the spirit of [22], localities express positions w.r.t. occurrences of the parallel composition operator within a term. We define the set of localities by $Loc = \{\swarrow, \searrow\}^*$ ranged over by ψ, we let $\swarrow x, \searrow x \in Var$ for all $x \in Var$, and we accordingly extend *Exp*, *BExp*, *Eval*, and *Sub*. When applied to a set, it is intended that $\swarrow$ and $\searrow$ must be applied to each element in the set. If we consider e.g.

$$\begin{array}{lcl} A(x) & \triangleq & <a!(x), \lambda>.(A'(x+1) \parallel_\emptyset A'(x+2)) \\ A'(y) & \triangleq & <a'!(y), \lambda'>.A'(y+1) \end{array}$$

where parameter y of the left occurrence of A' must be initialized to $x+1$ while parameter y of the right occurrence of A' must be initialized to $x+2$, we observe that no clash between the two parameters arise because the former is renamed $\swarrow A'.y$ while the latter is renamed $\searrow A'.y$. As we shall see shortly, while the dot notation is introduced statically via preprocessing, the localities are introduced dynamically during the application of the symbolic rules. We shall also see that, although localities would be needed in principle for every binary operator, their introduction is problematic in the case of dynamic operators like the alternative composition operator and the conditional operator. In our framework the problem is avoided by considering a syntax that allows only guarded summands and a semantics that makes conditional branches guarded, so that terms like $A'(x+1) + A'(x+2)$ are never encountered.

As said at the beginning of this section, the syntactical substitutions occurring in the rules of the concrete semantics must be encoded through assignments instead of being applied to terms. Once decided that the input actions labeling the transitions must stay symbolic, we have to deal with the syntactical substitutions occurring in the concrete rules. From the point of view of obtaining semantic models that are compact, the syntactical substitution occurring in the concrete rule for the constant invocation must be absolutely avoided in order to have just one state say $A(\underline{x})$ instead of a set of states $\{A(\underline{v}_j)\}_j$. This can be easily achieved by defining the symbolic rules in such a way that they look ahead in the derivative term of each generated transition in order to expand every constant invocation in the same way as the concrete rule for the constant invocation does.

Definition 8.3 *We define function unfold* : $\mathcal{G}_{\rm vp} \longrightarrow \mathcal{G}_{\rm vp}$ *by structural induction as follows:*

$$\begin{aligned}
unfold(\textstyle\sum_{i\in I} <\alpha_i, \tilde{\lambda}_i>.E_i) &= \textstyle\sum_{i\in I} <\alpha_i, \tilde{\lambda}_i>.E_i \\
unfold(\text{if } (\beta, p) \text{ then } E_1 \text{ else } E_2) &= \text{if } (\beta, p) \text{ then } E_1 \text{ else } E_2 \\
unfold(E/L) &= unfold(E)/L \\
unfold(E[\varphi]) &= unfold(E)[\varphi] \\
unfold(E_1 \parallel_S E_2) &= unfold(E_1) \parallel_S unfold(E_2) \\
unfold(A(\underline{e})) &= unfold(E) \qquad\qquad if\ A(\underline{x}) \triangleq E
\end{aligned}$$

■

It is worth noting that $A(\underline{e})$, where $A(\underline{x}) \triangleq E$, is replaced neither by $A(\underline{x})$ nor by E, but by $unfold(E)$. This is necessary to deal with sequences of constant invocations like e.g.

$$\begin{aligned}
A(x) &\triangleq A'(x+3) \\
A'(y) &\triangleq <a'!(y), \lambda'>.A'(y+1)
\end{aligned}$$

We also observe that the unfolding process terminates as soon as it is guaranteed that no constant invocation occurs outside the scope of an action prefix. This will be reflected in the symbolic semantics by the absence of a rule for the constant invocation.

The unfolding process must be accompanied by a process to record the generated assignments. Observed that the assignments generated in the case of a synchronization will be handled by the corresponding symbolic rule, there are two sources of assignment generation at unfolding time. First, we have to keep track of the assignments of actual parameters to formal parameters arising from the constant invocations. Since such assignments are built just before the constant invocation takes place, this procedure is consistent with the fact that in STGLA (and STGA') the assignments are evaluated after executing the actions, whereas it cannot be applied to STGA as it may cause errors. Second, when unfolding a parallel composition, it is necessary to generate suitable technical assignments. Consider for instance

$$\begin{aligned}
A(x, z) &\triangleq <a!(x), \lambda>.A'(x+1) \parallel_{\{a\}} <a?(z), *>.A'(z+2) \\
A'(y) &\triangleq <a'!(y), \lambda'>.A'(y+1)
\end{aligned}$$

Executing $<a!(A.x), \lambda>$ and unfolding $A'(A.x+1) \parallel_{\{a\}} A'(A.z+2)$ produce the assignment $\searrow A.z := \swarrow A.x \triangleright \swarrow A'.y := \swarrow A.x + 1 \triangleright \searrow A'.y := \searrow A.z + 2$. However, the initial variables are $A.x$ and $A.z$. In order for the assignment above to be correctly evaluated, a link has to be established between $A.x$ and $\swarrow A.x$. This is achieved by means of the insertion of technical assignment $\swarrow A.x := A.x$ before $\swarrow A'.y := \swarrow A.x + 1$.

The assignments generated during the unfolding process are recorded as follows, where $CLoc = \{\xi \mid \xi : Loc \rightharpoonup Const \cup \{\perp\}\}$ is a set of partial locality functions associating a constant with each locality.

Definition 8.4 *We define function* $record : \mathcal{G}_{\text{vp}} \times CLoc \times Loc \longrightarrow CLoc \times Sub$ *by structural induction as follows:*

$$
\begin{array}{rcll}
record(\sum_{i\in I} <\alpha_i, \tilde{\lambda}_i>.E_i, \xi, \psi) & = & (\xi, \varepsilon) & \\
record(\text{if } (\beta, p) \text{ then } E_1 \text{ else } E_2, \xi, \psi) & = & (\xi, \varepsilon) & \\
record(E/L, \xi, \psi) & = & record(E, \xi, \psi) & \\
record(E[\varphi], \xi, \psi) & = & record(E, \xi, \psi) & \\
record(E_1 \parallel_S E_2, \xi, \psi) & = & (\xi_1 \cup \xi_2, \sigma_{\text{tech}} \triangleright \swarrow\sigma_1 \triangleright \searrow\sigma_2) & \\
record(A(\underline{e}), \xi, \psi) & = & (\xi', \underline{x} := \underline{e} \triangleright \sigma') & \text{if } A(\underline{x}) \triangleq E
\end{array}
$$

where:

- $(\xi_1, \sigma_1) = record(E_1, \xi \cup \{(\swarrow\psi, \xi(\psi))\}, \swarrow\psi)$
  $(\xi_2, \sigma_2) = record(E_2, \xi \cup \{(\searrow\psi, \xi(\psi))\}, \searrow\psi)$
  $\sigma_{\text{tech}} = \begin{cases} \{\swarrow\psi\underline{x} := \psi\underline{x}, \searrow\psi\underline{x} := \psi\underline{x}\} & \text{if } \xi(\psi) \neq \bot \wedge \nexists A \in Const \cup \{\bot\}.\, (\swarrow\psi, A) \in \xi \wedge \xi(\psi)(\underline{x}) \triangleq E \\ \varepsilon & \text{if } \xi(\psi) = \bot \vee \exists A \in Const \cup \{\bot\}.\, (\swarrow\psi, A) \in \xi \end{cases}$
- $(\xi', \sigma') = record(E, \xi - \{(\psi, \xi(\psi))\} \cup \{(\psi, A)\}, \psi)$

In general, if $record(E, \xi, \psi) = (\xi', \sigma')$, ξ' *is denoted by* $loc(E, \xi, \psi)$ *while* σ' *is denoted by* $assign(E, \xi, \psi)$. ■

Let us explain the use of parameters ξ and ψ of function *record* and its clause for the parallel composition operator. Parameter ξ represents the locality tree of a term, i.e. it is a binary tree that keeps track of the positions in the syntactical structure of the term in which parallel composition operators have been encountered, together with the names of the constants in which those operators occur (or $\bot$ if they do not occur within the scope of a constant definition). Parameter ψ, instead, is used to navigate the locality tree described by ξ, i.e. it denotes the current locality. Given a term E, its initial locality tree is $\{(\varepsilon, const(E))\}$, where ε is the empty locality and

$$const(E) = \begin{cases} A & \text{if } E \equiv A(\underline{e}) \\ \bot & \text{otherwise} \end{cases}$$

The locality tree of a term is then updated by the symbolic rules whenever the unfolding of a derivative term of the given term takes place, with the navigator initially set to ε before each application of the rules. An update can refer to the structure of the tree, in the case when leaves become subtrees because new occurrences of the parallel composition operator are encountered, or to the constant names stored in the leaves, in the case when new constant invocations are encountered. If a parallel composition is encountered in the locality tree ξ at locality ψ, then the two leaves $\swarrow\psi$ and $\searrow\psi$ are added to the leaf ψ and labeled with constant $\xi(\psi)$, i.e. with the same constant as their parent. Moreover, as can be seen from the definition of σ_{tech}, technical assignments must be generated if the subterm under examination is within the scope of a constant definition ($\xi(\psi) \neq \bot$) and the parallel composition operator under consideration has not been encountered before ($\nexists A \in Const \cup \{\bot\}.\, (\swarrow\psi, A) \in \xi$). In this case, two technical assignments must be generated for every variable declared in constant $\xi(\psi)$. If an invocation of constant A is encountered in the locality tree ξ at locality ψ, the constant labeling leaf ψ is changed from $\xi(\psi)$ to A.

$$\frac{(<\alpha,\tilde{\lambda}>,\beta,\sigma,E',\xi') \in Melt_{\rm vp}(Select_{\rm vp}(PM_{\rm vp}(E,\xi,\varepsilon)))}{\langle E,\xi\rangle \xrightarrow{\alpha,\tilde{\lambda},\beta,\sigma}_{\rm vp} \langle E',\xi'\rangle}$$

$$PM_{\rm vp}(\sum_{i\in I} <\alpha_i,\tilde{\lambda}_i>.E_i,\xi,\psi) = \bigoplus_{i\in I} \{\!|\, (<\alpha_i,\tilde{\lambda}_i>, true, assign(E_i,\xi,\psi), unfold(E_i), loc(E_i,\xi,\psi)) \,|\!\}$$

$$PM_{\rm vp}(\text{if } (\beta,p) \text{ then } E_1 \text{ else } E_2,\xi,\psi) = \{\!|\, (<\tau,\infty_{\omega,p}>,\beta, assign(E_1,\xi,\psi), unfold(E_1), loc(E_1,\xi,\psi)),$$
$$(<\tau,\infty_{\omega,1-p}>,\neg\beta, assign(E_2,\xi,\psi), unfold(E_2), loc(E_2,\xi,\psi)) \,|\!\}$$

$$PM_{\rm vp}(E/L,\xi,\psi) = \{\!|\, (<\alpha,\tilde{\lambda}>,\beta,\sigma,E'/L,\xi') \mid (<\alpha,\tilde{\lambda}>,\beta,\sigma,E',\xi') \in PM_{\rm vp}(E,\xi,\psi) \wedge name(\alpha) \notin L \,|\!\} \oplus$$
$$\{\!|\, (<\tau,\tilde{\lambda}>,\beta,\sigma,E'/L,\xi') \mid \exists \alpha.\, (<\alpha,\tilde{\lambda}>,\beta,\sigma,E',\xi') \in PM_{\rm vp}(E,\xi,\psi) \wedge name(\alpha) \in L \,|\!\}$$

$$PM_{\rm vp}(E[\varphi],\xi,\psi) = \{\!|\, (<\varphi(\alpha),\tilde{\lambda}>,\beta,\sigma,E'[\varphi],\xi') \mid (<\alpha,\tilde{\lambda}>,\beta,\sigma,E',\xi') \in PM_{\rm vp}(E,\xi,\psi) \,|\!\}$$

$$PM_{\rm vp}(E_1 \parallel_S E_2,\xi,\psi) = \{\!|\, (<\alpha,\tilde{\lambda}>,\swarrow\beta,\swarrow\sigma,E_1' \parallel_S E_2,\xi') \mid name(\alpha) \notin S \wedge (<\alpha,\tilde{\lambda}>,\beta,\sigma,E_1',\xi') \in PM_{\rm vp}(E_1,\xi,\swarrow\psi) \,|\!\} \oplus$$
$$\{\!|\, (<\alpha,\tilde{\lambda}>,\searrow\beta,\searrow\sigma,E_1 \parallel_S E_2',\xi') \mid name(\alpha) \notin S \wedge (<\alpha,\tilde{\lambda}>,\beta,\sigma,E_2',\xi') \in PM_{\rm vp}(E_2,\xi,\searrow\psi) \,|\!\} \oplus$$
$$\{\!|\, (<\alpha,\tilde{\gamma}>,\swarrow\beta_1 \wedge \searrow\beta_2,\sigma,E_1' \parallel_S E_2',\xi_1 \cup \xi_2) \mid \exists \alpha_1,\alpha_2,\tilde{\lambda}_1,\tilde{\lambda}_2,\sigma_1,\sigma_2.$$
$$(<\alpha_1,\tilde{\lambda}_1>,\beta_1,\sigma_1,E_1',\xi_1) \in PM_{\rm vp}(E_1,\xi,\swarrow\psi) \wedge$$
$$(<\alpha_2,\tilde{\lambda}_2>,\beta_2,\sigma_2,E_2',\xi_2) \in PM_{\rm vp}(E_2,\xi,\searrow\psi) \wedge$$
$$name(\alpha_1) = name(\alpha_2) \in S \wedge$$
$$\tilde{\gamma} = Norm_{\rm vp}(\alpha_1,\alpha_2,\tilde{\lambda}_1,\tilde{\lambda}_2,PM_{\rm vp}(E_1),PM_{\rm vp}(E_2)) \wedge$$
$$\alpha = \begin{cases} \alpha_1 & \text{if } \alpha_1,\alpha_2 \in AType_{\rm U} \\ \alpha_1 & \text{if } \alpha_1,\alpha_2 \in AType_{\rm I} \\ \alpha_1 & \text{if } \alpha_1 \in Act_{\rm O} \wedge \alpha_2 \in AType_{\rm I} \\ \alpha_2 & \text{if } \alpha_1 \in Act_{\rm I} \wedge \alpha_2 \in AType_{\rm O} \end{cases} \wedge \sigma = \left\{ \begin{array}{ll} \varepsilon & \text{if } \alpha_1,\alpha_2 \in AType_{\rm U} \\ \searrow\underline{x}_2 := \swarrow\underline{x}_1 & \text{if } \alpha_1 = a?(\underline{x}_1) \wedge \alpha_2 = a?(\underline{x}_2) \\ \searrow\underline{x} := \swarrow\underline{e} & \text{if } \alpha_1 = a!(\underline{e}) \wedge \alpha_2 = a?(\underline{x}) \\ \swarrow\underline{x} := \searrow\underline{e} & \text{if } \alpha_1 = a?(\underline{x}) \wedge \alpha_2 = a!(\underline{e}) \end{array} \right\} \triangleright \swarrow\sigma_1 \triangleright \searrow\sigma_2 \,|\!\}$$

$$Select_{\rm vp}(PM) = \{\!|\, (<\alpha,\tilde{\lambda}>,\beta,\sigma,E,\xi) \in PM \mid (PL(<\alpha,\tilde{\lambda}>) = -1 \wedge \forall (<\alpha',\tilde{\lambda}'>,\beta',\sigma',E',\xi') \in PM.\, PL(<\alpha',\tilde{\lambda}'>) < \omega) \vee$$
$$\forall (<\alpha',\tilde{\lambda}'>,\beta',\sigma',E',\xi') \in PM.\, PL(<\alpha,\tilde{\lambda}>) \geq PL(<\alpha',\tilde{\lambda}'>) \,|\!\}$$

$$PL(<\alpha,*>) = -1 \qquad PL(<\alpha,\lambda>) = 0 \qquad PL(<\alpha,\infty_{l,w}>) = l$$

$$Melt_{\rm vp}(PM) = \{(<\alpha,\tilde{\lambda}>,\beta,\sigma,E,\xi) \mid \exists \tilde{\lambda}'.\, (<\alpha,\tilde{\lambda}'>,\beta,\sigma,E,\xi) \in PM \wedge$$
$$\tilde{\lambda} = Aggr\{\!|\, \tilde{\lambda}'' \mid (<\alpha,\tilde{\lambda}''>,\beta,\sigma,E,\xi) \in PM \wedge PL(<\alpha,\tilde{\lambda}''>) = PL(<\alpha,\tilde{\lambda}'>) \,|\!\}\}$$

$$* \; Aggr \; * = * \qquad \lambda_1 \; Aggr \; \lambda_2 = \lambda_1 + \lambda_2 \qquad \infty_{l,w_1} \; Aggr \; \infty_{l,w_2} = \infty_{l,w_1+w_2}$$

$$Norm_{\rm vp}(\alpha_1,\alpha_2,\tilde{\lambda}_1,\tilde{\lambda}_2,PM_1,PM_2) = \begin{cases} Split(\tilde{\lambda}_1, 1/(\pi_1(PM_2))(<\alpha_2,*>)) & \text{if } \alpha_1 = \alpha_2 \in AType_U \wedge \tilde{\lambda}_2 = * \\ Split(\tilde{\lambda}_2, 1/(\pi_1(PM_1))(<\alpha_1,*>)) & \text{if } \alpha_1 = \alpha_2 \in AType_U \wedge \tilde{\lambda}_1 = * \\ * & \text{if } \alpha_1 = a?(\underline{x}_1) \wedge \alpha_2 = a?(\underline{x}_2) \\ Split(\tilde{\lambda}_1, 1/\sum\limits_{\underline{y}\in\underline{Var}} (\pi_1(PM_2))(<a?(\underline{y}),*>)) & \text{if } \alpha_1 = a!(\underline{e}) \wedge \alpha_2 = a?(\underline{x}) \\ Split(\tilde{\lambda}_2, 1/\sum\limits_{\underline{y}\in\underline{Var}} (\pi_1(PM_1))(<a?(\underline{y}),*>)) & \text{if } \alpha_1 = a?(\underline{x}) \wedge \alpha_2 = a!(\underline{e}) \end{cases}$$

$$Split(*,p) = * \qquad Split(\lambda,p) = \lambda \cdot p \qquad Split(\infty_{l,w},p) = \infty_{l,w\cdot p}$$

Table 8.1: Inductive rules for EMPA$_{\rm vp}$ integrated interleaving semantics

The integrated interleaving semantics of $\mathrm{EMPA}_{\mathrm{vp}}$ terms is defined by exploiting again the idea of potential move, which is now a quintuple $(<\alpha, \tilde{\lambda}>, \beta, \sigma, E, \xi)$. We denote by $PMove_{\mathrm{vp}} = Act'_{\mathrm{vp}} \times BExp \times Sub \times \mathcal{G}_{\mathrm{vp}} \times CLoc$ the set of all the potential moves. The formal definition is based on the transition relation $\longrightarrow_{\mathrm{vp}}$, which is the least subset of $(\mathcal{G}_{\mathrm{vp}} \times CLoc) \times (Act'_{\mathrm{vp}} \times BExp \times Sub) \times (\mathcal{G}_{\mathrm{vp}} \times CLoc)$ satisfying the inference rule reported in the first part of Table 8.1, where it can be noted that each state is a pair $\langle E, \xi \rangle$. This rule selects the potential moves having the highest priority level and then merges together those having the same action type, the same priority level, the same boolean guard, the same assignment, and the same derivative term. The first operation is carried out through functions $Select_{\mathrm{vp}} : \mathcal{M}u_{\mathrm{fin}}(PMove_{\mathrm{vp}}) \longrightarrow \mathcal{M}u_{\mathrm{fin}}(PMove_{\mathrm{vp}})$ and $PL : Act'_{\mathrm{vp}} \longrightarrow APLev_{\mathrm{vp}}$, which are defined in the third part of Table 8.1. It is worth noting that the selection of the potential moves with the highest priority level does not interfere with boolean guards because nontrivial boolean guards are associated only with potential moves generated by the conditional operator (see the second part of Table 8.1) and these have the same priority level and take precedence over any other potential move. The second operation is carried out through function $Melt_{\mathrm{vp}} : \mathcal{M}u_{\mathrm{fin}}(PMove_{\mathrm{vp}}) \longrightarrow \mathcal{P}_{\mathrm{fin}}(PMove_{\mathrm{vp}})$ and partial function $Aggr : ARate_{\mathrm{vp}} \times ARate_{\mathrm{vp}} \not\longrightarrow ARate_{\mathrm{vp}}$, which are defined in the fourth part of Table 8.1.

The multiset $PM_{\mathrm{vp}}(E) \in \mathcal{M}u_{\mathrm{fin}}(PMove_{\mathrm{vp}})$ of potential moves of $E \in \mathcal{G}_{\mathrm{vp}}$ is defined by structural induction in the second part of Table 8.1. Note that the three auxiliary functions *unfold*, *loc*, and *assign* are used only in the rules for the guarded alternative composition operator and the conditional operator, while localities prefixing variable names occurring in assignments are recovered by the rule for the parallel composition operator. In particular, the synchronization can also generate new assignments. Moreover, note that there is no rule for the constant invocation because of the a priori application of function *unfold*.

The normalization of rates of potential moves resulting from the synchronization of an active action with several independent or alternative passive actions of the same name is carried out through partial function $Norm_{\mathrm{vp}} : AType_{\mathrm{vp}} \times AType_{\mathrm{vp}} \times ARate_{\mathrm{vp}} \times ARate_{\mathrm{vp}} \times \mathcal{M}u_{\mathrm{fin}}(PMove_{\mathrm{vp}}) \times \mathcal{M}u_{\mathrm{fin}}(PMove_{\mathrm{vp}}) \not\longrightarrow ARate_{\mathrm{vp}}$ and function $Split : ARate_{\mathrm{vp}} \times \mathbb{R}_{]0,1]} \longrightarrow ARate_{\mathrm{vp}}$, which are defined in the fifth part of Table 8.1.

Definition 8.5 *The integrated interleaving semantics of $E \in \mathcal{G}_{\mathrm{vp}}$ is the STGLA*

$$\mathcal{I}_{\mathrm{vp}}[\![E]\!] = (S_{E,\mathcal{I}_{\mathrm{vp}}}, \longrightarrow_{E,\mathcal{I}_{\mathrm{vp}}}, \langle unfold(E), loc(E, \{(\varepsilon, const(E))\}, \varepsilon)\rangle, assign(E, \{(\varepsilon, const(E))\}, \varepsilon); Var, Act'_{\mathrm{vp}} \times BExp \times Sub)$$

where:

- *$S_{E,\mathcal{I}_{\mathrm{vp}}}$ is the least subset of $\mathcal{G}_{\mathrm{vp}} \times CLoc$ such that:*
  - *$\langle unfold(E), loc(E, \{(\varepsilon, const(E))\}, \varepsilon)\rangle \in S_{E,\mathcal{I}_{\mathrm{vp}}}$;*
  - *if $\langle E_1, \xi_1\rangle \in S_{E,\mathcal{I}_{\mathrm{vp}}}$ and $\langle E_1, \xi_1\rangle \stackrel{\alpha,\tilde{\lambda},\beta,\sigma}{\longrightarrow}_{\mathrm{vp}} \langle E_2, \xi_2\rangle$, then $\langle E_2, \xi_2\rangle \in S_{E,\mathcal{I}_{\mathrm{vp}}}$.*
- *$\longrightarrow_{E,\mathcal{I}_{\mathrm{vp}}}$ is the restriction of $\longrightarrow_{\mathrm{vp}}$ to $S_{E,\mathcal{I}_{\mathrm{vp}}} \times (Act'_{\mathrm{vp}} \times BExp \times Sub) \times S_{E,\mathcal{I}_{\mathrm{vp}}}$.* ■

Definition 8.6 *$E \in \mathcal{G}_{\rm vp}$ is performance closed if and only if $\mathcal{I}_{\rm vp}[\![E]\!]$ does not contain passive transitions. We denote by $\mathcal{E}_{\rm vp}$ the set of performance closed terms of $\mathcal{G}_{\rm vp}$.* ■

Proposition 8.1 *For all $E \in \mathcal{G}_{\rm vp}$, every nonabsorbing state of $\mathcal{I}_{\rm vp}[\![E]\!]$ is such that all of its outgoing transitions either have type τ and priority level ω or have priority level less than ω and boolean guard true.* ■

Definition 8.7 *The functional semantics of $E \in \mathcal{G}_{\rm vp}$ is the STGLA*

$$\begin{aligned}\mathcal{F}_{\rm vp}[\![E]\!] \;=\; & (S_{E,\mathcal{F}_{\rm vp}},\, \longrightarrow_{E,\mathcal{F}_{\rm vp}}, \langle \mathit{unfold}(E), \mathit{loc}(E,\{(\varepsilon,\mathit{const}(E))\},\varepsilon)\rangle, \mathit{assign}(E,\{(\varepsilon,\mathit{const}(E))\},\varepsilon);\\ & \mathit{Var}, \mathit{AType}_{\rm vp} \times \mathit{BExp} \times \mathit{Sub})\end{aligned}$$

where:

- $S_{E,\mathcal{F}_{\rm vp}} = S_{E,\mathcal{I}_{\rm vp}}$.
- $\longrightarrow_{E,\mathcal{F}_{\rm vp}}$ *is the restriction of* $\longrightarrow_{E,\mathcal{I}_{\rm vp}}$ *to* $S_{E,\mathcal{F}_{\rm vp}} \times (\mathit{AType}_{\rm vp} \times \mathit{BExp} \times \mathit{Sub}) \times S_{E,\mathcal{F}_{\rm vp}}$. ■

Definition 8.8 *The Markovian semantics of $E \in \mathcal{E}_{\rm vp}$ is the p-LTS*

$$\mathcal{M}_{\rm vp}[\![E]\!] = (S_{E,\mathcal{M}_{\rm vp}}, \mathbb{R}_+, \longrightarrow_{E,\mathcal{M}_{\rm vp}}, P_{E,\mathcal{M}_{\rm vp}}, H_{E,\mathcal{M}_{\rm vp}})$$

where:

- $S_{E,\mathcal{M}_{\rm vp}} = S_{E,\mathcal{I}_{\rm vp}}$.
- $\longrightarrow_{E,\mathcal{M}_{\rm vp}}$ *is the least subset of* $S_{E,\mathcal{M}_{\rm vp}} \times \mathbb{R}_+ \times S_{E,\mathcal{M}_{\rm vp}}$ *such that:*
  - $\langle F,\xi\rangle \xrightarrow{\lambda}_{E,\mathcal{M}_{\rm vp}} \langle F',\xi'\rangle$ *whenever* $\lambda = \sum\{\!|\, \mu \mid \exists \alpha,\beta,\sigma.\, \langle F,\xi\rangle \xrightarrow{\alpha,\mu,\beta,\sigma}_{E,\mathcal{I}_{\rm vp}} \langle F',\xi'\rangle \,|\!\}$;
  - $\langle F,\xi\rangle \xrightarrow{p}_{E,\mathcal{M}_{\rm vp}} \langle F',\xi'\rangle$ *whenever* $p = \sum\{\!|\, w \mid \exists \alpha,l,\beta,\sigma.\, \langle F,\xi\rangle \xrightarrow{\alpha,\infty_{l,w},\beta,\sigma}_{E,\mathcal{I}_{\rm vp}} \langle F',\xi'\rangle \,|\!\} /$ $\sum\{\!|\, w \mid \exists \alpha,l,\beta,\sigma,F'',\xi''.\, \langle F,\xi\rangle \xrightarrow{\alpha,\infty_{l,w},\beta,\sigma}_{E,\mathcal{I}_{\rm vp}} \langle F'',\xi''\rangle \,|\!\}$.
- $P_{E,\mathcal{M}_{\rm vp}} : S_{E,\mathcal{M}_{\rm vp}} \longrightarrow \mathbb{R}_{[0,1]}$, $P_{E,\mathcal{M}_{\rm vp}}(F,\xi) = \begin{cases} 1 & \text{if } \langle F,\xi\rangle \equiv \langle E,\{(\varepsilon,\mathit{const}(E))\}\rangle \\ 0 & \text{if } \langle F,\xi\rangle \not\equiv \langle E,\{(\varepsilon,\mathit{const}(E))\}\rangle \end{cases}$
- $H_{E,\mathcal{M}_{\rm vp}} : S_{E,\mathcal{M}_{\rm vp}} \longrightarrow \{v,t,a\}$, $H_{E,\mathcal{M}_{\rm vp}}(F,\xi) = \begin{cases} v & \text{if } \exists \alpha,l,w,\beta,\sigma,F',\xi'.\, \langle F,\xi\rangle \xrightarrow{\alpha,\infty_{l,w},\beta,\sigma}_{E,\mathcal{I}_{\rm vp}} \langle F',\xi'\rangle \\ t & \text{if } \exists \alpha,\lambda,\beta,\sigma,F',\xi'.\, \langle F,\xi\rangle \xrightarrow{\alpha,\lambda,\beta,\sigma}_{E,\mathcal{I}_{\rm vp}} \langle F',\xi'\rangle \\ a & \text{if } \nexists \alpha,\tilde{\lambda},\beta,\sigma,F',\xi'.\, \langle F,\xi\rangle \xrightarrow{\alpha,\tilde{\lambda},\beta,\sigma}_{E,\mathcal{I}_{\rm vp}} \langle F',\xi'\rangle \end{cases}$ ■

It is worth observing that the definitions of $\mathcal{I}_{\rm vp}[\![E]\!]$, $\mathcal{F}_{\rm vp}[\![E]\!]$, and $\mathcal{M}_{\rm vp}[\![E]\!]$ are meaningful, in the sense that such models do not exhibit undesired behaviors, only for those terms E which do not contain a cascade of conditional operators with conflicting boolean expressions. As an example, if we consider term

$$\text{if } (x > 10, 0.2) \text{ then } (\text{if } (x \leq 10, 0.2) \text{ then } E_1 \text{ else } E_2) \text{ else } E_3$$

we see that a sequence of two transitions reaching E_1 is generated, each labeled with action $<\tau, \infty_{\omega,0.2}>$. However, since the conjunction of their boolean guards is $x > 10 \wedge x \leq 10$, the second transition in the sequence should not be considered. Unfortunately, detecting such situations is possible only in case of simulation based analysis, but not in general because the underlying semantic models are symbolic.

We now comment on the way in which localities have been introduced. While the dot notation is related to the definition of a constant (which is unique), the localities are related to the occurrences of a constant (which can be multiple). Therefore, the localities cannot be introduced statically but during the application of the symbolic rules. This can be accomplished by means of two different techniques.

With the first technique, localities are introduced in the assignments but not in the terms, i.e. function *unfold* is defined exactly as in Def. 8.3. In this case, the symbolic rules for the parallel composition operator have to take care of recovering the localities prefixing the variable names in the assignments. This works for the parallel composition operator because it is a static operator, but would not work for the standard alternative composition operator and the standard conditional operator because they are dynamic. As an example, if we consider

$$\begin{array}{rcl} A(x) & \triangleq & <a!(x), \lambda>.(A'(x+1) + A'(x+2)) \\ A'(y) & \triangleq & <a'!(y), \lambda'>.A'(y+1) \end{array}$$

and we use $\rfloor, \lfloor$ as localities for the alternative composition operator, after the execution of $<a!(A.x), \lambda>$ we have derivative term $unfold(A'(A.x+1) + A'(A.x+2)) = a'!(A'.y).A'(A'.y+1) + a'!(A'.y).A'(A'.y+1)$ with assignment $\rfloor A'.y := A.x + 1 \triangleright \lfloor A'.y := A.x + 2$. At this point, if the action of the left summand is executed, the new derivative term is $unfold(A'(A'.y+1)) = a'!(A'.y).A'(A'.y+1)$ with assignment $A'.y := A'.y + 1$ where the link with $\rfloor A'.y$ has been lost because the related alternative composition operator has disappeared.

With the second technique, instead, localities are introduced both in the assignments and in the terms as long as the symbolic rules are applied. In this case, the unfolding clause for the parallel composition operator must be changed by posing

$$unfold(E_1 \parallel_S E_2) = \swarrow unfold(E_1) \parallel_S \searrow unfold(E_2)$$

where $\swarrow E$ ($\searrow E$) is intended as the term obtained from E by replacing each variable x occurring in it with $\swarrow x$ ($\searrow x$), while the symbolic rules for the parallel composition operator do not have to recover localities because they are already present. From this observation about the symbolic rules, it follows that this technique would work for the standard alternative composition operator and the standard conditional operator, but may lead to the generation of a number of states greater than necessary. As an example, if we consider again $A(x)$ above, then after the execution of $<a!(A.x), \lambda>$ we have derivative term $unfold(A'(A.x+1) + A'(A.x+2)) = a'!(\rfloor A'.y).A'(\rfloor A'.y+1) + a'!(\lfloor A'.y).A'(\lfloor A'.y+1)$ from which two different states can be reached instead of the same state as expected by looking at the process description.

The proof of correctness of the STGLA based symbolic semantics for EMPA_{vp} proceeds in two steps. First, we observe that, for every term, its symbolic semantics and its concrete semantics are able to mimic each other. The reason is that every concrete state can be related to the symbolic state obtainable from

it by applying function *unfold*, because every application of function *unfold* to a symbolic derivative state corresponds to the application of the concrete rule for the constant invocation to a concrete source state. Second, we establish that the order in which the assignments occur in the transitions is correct. This is important e.g. in the case of simulation based analysis of a process description, because simulation requires the transitions to be actually executed, hence the related assignments to be actually evaluated.

Since the symbolic rules are always applied to terms obtained from the application of function *unfold*, the assignments stemming from synchronizations must be evaluated before the assignments generated by the unfolding of the corresponding derivative terms. As can be seen from the symbolic rule for the synchronization, this is the order in which such assignments are generated. As an example, consider

$$\begin{aligned} A(x,z) &\triangleq <a!(x),\lambda>.A'(x+1) \parallel_{\{a\}} <a?(z),*>.A'(z+2) \\ A'(y) &\triangleq <a'!(y),\lambda'>.A'(y+1) \end{aligned}$$

The transition originated by the synchronization of $<a!(\swarrow A.x),\lambda>$ with $<a?(\searrow A.z),*>$ is labeled with assignment $\searrow A.z := \swarrow A.x \triangleright \swarrow A'.y := \swarrow A.x + 1 \triangleright \searrow A'.y := \searrow A.z + 2$. Assignment $\searrow A.z := \swarrow A.x$ must be evaluated before the other two, otherwise $\searrow A'.y$ would get a wrong value.

As far as assignments related to multiway structured synchronizations are concerned, we note that they must be evaluated starting from those having the expression of the involved output action in their right hand side. The symbolic rule for the synchronization generates the related assignments in the appropriate order, because in case of synchronization of $<a?(\underline{x}_1),*>$ with $<a?(\underline{x}_2),*>$ it sets $<\underline{x}_2,*>$ to $\underline{x}_1$ and propagates $<a?(\underline{x}_1),*>$ up for further synchronizations with structured actions, while in case of synchronization of $<a!(\underline{e}),\tilde{\lambda}>$ with $<a?(\underline{x}),*>$ it sets $\underline{x}$ to $\underline{e}$ and propagates $<a!(\underline{e}),\tilde{\lambda}>$ up for further synchronizations with input actions.

The assignments generated during the unfolding process must be evaluated in the same order as they are produced by function *assign*. In particular, in the clause for the parallel composition operator, the technical assignments must be evaluated before the other assignments. Similarly, in the rule for the constant invocation, the parameter setting assignments must be evaluated before the other assignments. As an example, consider

$$\begin{aligned} A(x) &\triangleq <a!(x),\lambda>.(A_1(x+1) \parallel_{\emptyset} A_2(x+2)) \\ A_1(y) &\triangleq <a_1!(y),\mu>.A_1(y+3) \\ A_2(z) &\triangleq A_3(z+4) \\ A_3(z) &\triangleq <a_3!(z),\gamma>.A_3(z+5) \end{aligned}$$

The transition originated by the execution of $<a!(A.x),\lambda>$ is labeled with assignment $\swarrow A.x := A.x \triangleright \searrow A.x := A.x \triangleright \swarrow A_1.y := \swarrow A.x + 1 \triangleright \searrow A_2.z := \searrow A.x + 2 \triangleright \searrow A_3.z := \searrow A_2.z + 4$ which has to be evaluated exactly in this order to avoid errors.

Moreover, we observe that the parameter setting assignments that refer to variables with the same prefix, i.e. declared in the same constant invoked in the same locality, must be simultaneously treated in two steps. In the first step, all of their right hand sides must be evaluated. In the second step, all of their left hand

sides must be set. To see why, consider

$$A(x,y) \triangleq <a!(x,y),\lambda>.A(x+1,x)$$

The transition originated by the execution of $<a!(A.x,A.y),\lambda>$ is labeled with assignment $A.x := A.x+1 \triangleright A.y := A.x$. If $A.x$ is immediately set after evaluating $A.x+1$, then $A.y$ gets a wrong value. In general, for this kind of assignments (which can be easily recognized during the application of function *assign*) it is not possible to fix an order which allows them to be treated individually, as can be seen in the term above by substituting $A(y,x)$ for $A(x+1,x)$.

We can thus conclude that the assignment generation order imposed by function *assign* and the symbolic rules is correct w.r.t. the assignment evaluation order in the sense below, where *prefix*(x) denotes the string over the concatenation of Loc^* and *Const* which precedes the actual name of x.

Theorem 8.1 *Whenever* $\langle E,\xi\rangle \stackrel{\alpha,\lambda,\beta,\sigma}{\longrightarrow} \langle E',\xi'\rangle$, *then for all* $x \in Var$ *set by* $x := e$ *in* σ, *there is no* $x' := e'$ *preceding* $x := e$ *in* σ *with* $prefix(x') \neq prefix(x)$ *such that* x *occurs in* e'. ■

In Sect. 8.1 we have illustrated some improvements that have been made to the original symbolic transition graphs in such a way that they are now able to represent value passing terms in a more compact way. In order to benefit from the inherent parametricity of value passing as much as possible, no syntactical substitution should be applied to value passing process terms and every constant invocation should be suitably replaced with its definition so that the formal parameters (which are unique) replace the actual parameters (which can vary from invocation to invocation). Our symbolic rules fulfil not only the first requirement but also partially the second one. As far as the second requirement is concerned, since no syntactical substitution is performed by our symbolic rules, we observe that the problem of generating more states than necessary only arises when the derivatives of several transitions are identical up to the values of the actual parameters of the same constant invocations. A simple way of reducing the number of unnecessary states is to rewrite terms in such a way that every constant invocation occurs within the scope of an action prefix.

Definition 8.9 *Let* $Z = (S, \longrightarrow, s_0, \sigma_0; Var, Act'_{\rm vp} \times BExp \times Sub)$ *be a STGLA associated with a term of* $EMPA_{\rm vp}$. *Z is compact if and only if none of its states contains a substate of the form* $A(\underline{e})$ *outside the scope of the outermost guarded alternative composition operator or conditional operator.* ■

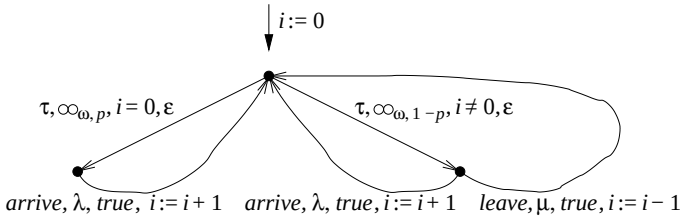
Theorem 8.2 *For all* $E \in \mathcal{G}_{\rm vp}$, $\mathcal{I}_{\rm vp}[\![E]\!]$ *is compact.* ■

The theorem above shows that our symbolic rules improve those of [121] because the latter do not produce compact STGA'.

Example 8.1 Let us consider again the unbounded buffer introduced in Sect. 8.1. According to the syntax of $\text{EMPA}_{\rm vp}$, the buffer is specified as follows:

$$\begin{aligned} \mathit{Buffer}(i) \quad \triangleq \quad & \text{if } (i = 0, p) \text{ then} \\ & \quad <\mathit{arrive}, \lambda>.\mathit{Buffer}(i+1) \\ & \text{else} \\ & \quad <\mathit{arrive}, \lambda>.\mathit{Buffer}(i+1) + <\mathit{leave}, \mu>.\mathit{Buffer}(i-1) \end{aligned}$$

where $p = 1 - \lambda/\mu$ since the buffer basically represents a QS $M/M/1$ with arrival rate λ and service rate μ hence p must reflect the probability that the system is empty. Below we show $\mathcal{I}_{\mathrm{vp}}[\![\mathit{Buffer}(0)]\!]$:



This STGLA has three states: the initial one corresponds to term $\mathit{Buffer}(i)$, while the others correspond to terms $<\mathit{arrive}, \lambda>.\mathit{Buffer}(i+1)$ and $<\mathit{arrive}, \lambda>.\mathit{Buffer}(i+1) + <\mathit{leave}, \mu>.\mathit{Buffer}(i-1)$, respectively. ■

8.2.5 Integrated Equivalence

The purpose of this section is to adapt $\sim_{\mathrm{EMB}}$ to the symbolic framework of $\mathrm{EMPA}_{\mathrm{vp}}$. Following the guidelines of [81, 122, 121], it turns out that this is not a trivial task for two reasons. First, the standard bisimulation equivalence is given for operational semantics defined on closed terms, while the symbolic semantics for $\mathrm{EMPA}_{\mathrm{vp}}$ necessarily deals with open terms, i.e. terms possibly containing free variables like e.g. $<a!(x), \tilde{\lambda}>.\underline{0}$. This suggests that such a semantics should be redefined e.g. relative to an evaluation to reobtain closed terms. Second, the standard (early concrete) definition of operational semantics for value passing process algebras is based on syntactical substitutions like e.g. in the concrete rules of the form $<a?(\underline{x}), \tilde{\lambda}>.E \stackrel{a?(\underline{v}),\tilde{\lambda}}{\longrightarrow} E\{\underline{v}/\underline{x}\}$ for all $v \in \underline{\mathit{Var}}$, whereas the semantics for $\mathrm{EMPA}_{\mathrm{vp}}$ does not apply syntactical substitutions of this kind. This suggests that such a semantics should be redefined in such a way that this syntactical substitution related information can be retrieved, e.g. using assignments to be associated with STGLA states.

In the following, each state $\langle E, \xi \rangle$ of a STGLA representing the integrated interleaving semantics of an $\mathrm{EMPA}_{\mathrm{vp}}$ term will be denoted simply by E with abuse of notation. Moreover, we let $\mathcal{O} = \mathcal{G}_{\mathrm{vp}} \times \mathit{Sub}$ be the set of *open terms*, ranged over by O. In the following, we shall also make use of syntactical substitutions restricted to variables, i.e. of the form $\{\underline{y}/\underline{x}\}$. When applied to a term/substitution, it means that each occurrence of $\underline{x}$ in the term/substitution must be replaced with $\underline{y}$. When applied to an open term, it must be applied both to the term and to the substitution composing the open term.

To redefine the semantics for $\mathrm{EMPA}_{\mathrm{vp}}$ over $\mathcal{O}$ we can follow two different approaches named *early* and *late*, respectively, where the difference relies on whether the behavior of processes w.r.t. input actions is related or not to the fact that each of their variables can take on several values. We shall take the late

view since the early view (which is based on rules like $<a?(\underline{x}), \tilde{\lambda}>.E \stackrel{a?(\underline{v}),\tilde{\lambda}}{\longrightarrow} E\{\underline{v}/\underline{x}\}$ for all $v \in \underline{Var}$) can lead to wrong rate increases or infinite branching. Actually, we shall investigate the relationships between two different semantics, hence two different extensions of $\sim_{\mathrm{EMB}}$, defined on $\mathcal{O}$ called *concrete late* and *symbolic late*, respectively, where the difference relies on whether expressions in boolean guards and output actions are left symbolic or instantiated to values.

In the concrete late case, expressions occurring in boolean guards and output actions are evaluated according to a given evaluation ρ supplying values for free variables, so that a more concrete instance of the symbolic model $\mathcal{I}_{\mathrm{vp}}[\![E]\!]$ is obtained where only the transitions whose boolean guards evaluate to true according to ρ are kept, hence boolean guards no longer appear in transition labels and the parameters of output actions are just values. We define $AType'_{\mathrm{O}} = \{a!(\underline{v}) \mid a \in AName_{\mathrm{IO}} \wedge \underline{v} \in \underline{Val}\}$, $AType'_{\mathrm{vp}} = AType_{\mathrm{U}} \cup AType_{\mathrm{I}} \cup AType'_{\mathrm{O}}$, and $Act''_{\mathrm{vp}} = \{<\alpha, \tilde{\lambda}> \in AType'_{\mathrm{vp}} \times ARate_{\mathrm{vp}} \mid \alpha \in AType_{\mathrm{I}} \Longrightarrow \tilde{\lambda} = *\}$.

Definition 8.10 *The concrete late integrated interleaving semantics of $E \in \mathcal{G}_{\mathrm{vp}}$ w.r.t. $\rho \in Eval$ is the LTS*

$$\mathcal{I}^{\mathrm{CL},\rho}_{\mathrm{vp}}[\![E]\!] = (S_{E,\mathrm{CL},\rho}, Act''_{\mathrm{vp}}, \longrightarrow_{E,\mathrm{CL},\rho}, \langle unfold(E), assign(E, \{(\varepsilon, const(E))\}, \varepsilon)\rangle)$$

where:

- *$S_{E,\mathrm{CL},\rho}$ is the least subset of $\mathcal{O}$ such that:*
  - *$\langle unfold(E), assign(E, \{(\varepsilon, const(E))\}, \varepsilon)\rangle \in S_{E,\mathrm{CL},\rho}$;*
  - *if $O_1 \in S_{E,\mathrm{CL},\rho}$ and $O_1 \stackrel{\alpha,\tilde{\lambda}}{\longrightarrow}_{\mathrm{CL},\rho} O_2$, then $O_2 \in S_{E,\mathrm{CL},\rho}$.*
- *$\longrightarrow_{E,\mathrm{CL},\rho}$ is the restriction of $\longrightarrow_{\mathrm{CL},\rho}$ defined in Table 8.2 to $S_{E,\mathrm{CL},\rho} \times Act''_{\mathrm{vp}} \times S_{E,\mathrm{CL},\rho}$.* ■

$$\frac{E \stackrel{\alpha,\tilde{\lambda},\beta,\sigma'}{\longrightarrow}_{\mathrm{vp}} E'}{\langle E,\sigma\rangle \stackrel{\alpha,\tilde{\lambda}}{\longrightarrow}_{\mathrm{CL},\rho} \langle E', \sigma \triangleright \sigma'\rangle} \quad \text{if } \alpha \in AType_{\mathrm{U}} \wedge \rho(\beta\sigma) = true$$

$$\frac{E \stackrel{a?(\underline{x}),\tilde{\lambda},true,\sigma'}{\longrightarrow}_{\mathrm{vp}} E'}{\langle E,\sigma\rangle \stackrel{a?(\underline{x}),\tilde{\lambda}}{\longrightarrow}_{\mathrm{CL},\rho} \langle E', \sigma \triangleright \sigma'\rangle}$$

$$\frac{E \stackrel{a!(\underline{e}),\tilde{\lambda},true,\sigma'}{\longrightarrow}_{\mathrm{vp}} E'}{\langle E,\sigma\rangle \stackrel{a!(\rho(\underline{e}\sigma)),\tilde{\lambda}}{\longrightarrow}_{\mathrm{CL},\rho} \langle E', \sigma \triangleright \sigma'\rangle}$$

Table 8.2: Rules for $\mathrm{EMPA}_{\mathrm{vp}}$ concrete late semantics

Definition 8.11 *For each $\rho \in Eval$ we define partial function $Rate_{\mathrm{CL},\rho} : (\mathcal{O} \times AType'_{\mathrm{vp}} \times APLev_{\mathrm{vp}} \times \mathcal{P}(\mathcal{O}) \times \underline{Var} \times \underline{Var}) \nrightarrow ARate_{\mathrm{vp}}$ by*

$$Rate_{\mathrm{CL},\rho}(O, \alpha, l, C, \underline{x}, \underline{y}) = Aggr\{\!|\, \tilde{\lambda} \mid \exists O'.\, O \xrightarrow{\alpha,\tilde{\lambda}}_{\mathrm{CL},\rho} O' \wedge PL(<\alpha, \tilde{\lambda}>) = l \wedge O'\{\underline{y}/\underline{x}\} \in C \,|\!\}$$ ■

Definition 8.12 *Let $\mathcal{R} = \{\mathcal{B}^\rho \subseteq \mathcal{O} \times \mathcal{O} \mid \mathcal{B}^\rho$ equivalence $\wedge\, \rho \in Eval\}$ and let $CLB(\mathcal{R}) = \{CLB(\mathcal{R})^\rho \subseteq \mathcal{O} \times \mathcal{O} \mid CLB(\mathcal{R})^\rho$ equivalence $\wedge\, \rho \in Eval\}$ be defined by $(O_1, O_2) \in CLB(\mathcal{R})^\rho$ if and only if:*

(i) $\forall \alpha \in AType_{\mathrm{U}}.\, \forall l \in APLev_{\mathrm{vp}}.\, \forall C \in \mathcal{O}/\mathcal{B}^\rho$

$$Rate_{\mathrm{CL},\rho}(O_1, \alpha, l, C, \underline{x}, \underline{x}) = Rate_{\mathrm{CL},\rho}(O_2, \alpha, l, C, \underline{x}, \underline{x})$$

with $\underline{x} \in \underline{Var}$ arbitrary.

(ii) $\forall a \in AName_{\mathrm{IO}}.\, \forall \underline{x}, \underline{y} \in \underline{Var}.\, \forall \underline{v} \in \underline{Val}.\, \forall l \in APLev_{\mathrm{vp}}.\, \forall C \in \mathcal{O}/\mathcal{B}^{\rho[\underline{x} \mapsto \underline{v}]}$

$$Rate_{\mathrm{CL},\rho}(O_1, a?(\underline{x}), l, C, \underline{x}, \underline{z}) = Rate_{\mathrm{CL},\rho}(O_2, a?(\underline{y}), l, C, \underline{y}, \underline{z})$$

with $\underline{z} \in \underline{Var}$ fresh.

(iii) $\forall \alpha \in AType'_{\mathrm{O}}.\, \forall l \in APLev_{\mathrm{vp}}.\, \forall C \in \mathcal{O}/\mathcal{B}^\rho$

$$Rate_{\mathrm{CL},\rho}(O_1, \alpha, l, C, \underline{x}, \underline{x}) = Rate_{\mathrm{CL},\rho}(O_2, \alpha, l, C, \underline{x}, \underline{x})$$

with $\underline{x} \in \underline{Var}$ arbitrary.

We say that $\mathcal{R}$ is a strong concrete late extended Markovian bisimulation (strong CLEMB) if and only if $\mathcal{R} \subseteq CLB(\mathcal{R})$, i.e. $\mathcal{R}^\rho \subseteq CLB(\mathcal{R})^\rho$ for each $\rho \in Eval$. ■

Definition 8.13 *We define the strong concrete late extended Markovian bisimulation equivalence (strong CLEMBE) w.r.t. $\rho \in Eval$, denoted $\sim^\rho_{\mathrm{CLEMB}}$, as the ρ-th component of the greatest fixed point of functional CLB.* ■

Since functional CLB is componentwise monotonic, the definition above is well founded. Furthermore, $\sim^\rho_{\mathrm{CLEMB}}$ turns out to be an equivalence relation.

In the symbolic late case, instead, expressions occurring in boolean guards and output actions are carried in transitions instead of getting evaluated, so the resulting semantic model (a STG) is more abstract because it does not refer to evaluations. On the other hand, the notion of equivalence must take into account also the boolean guards of the sets of transitions whose aggregated rates are compared. This is because e.g. a transition labeled with a certain action whose boolean guard is $x = 1$ is not matched by a transition labeled with the same action whose boolean guard is $x = 2$. According to [81] the idea is to define a notion of equivalence parametrized by a boolean expression β and to identify a set B of cases β' of β, where $\beta = \bigvee B$, under which transitions can be properly matched.

Definition 8.14 *The symbolic late integrated interleaving semantics of $E \in \mathcal{G}_{\mathrm{vp}}$ is the STG*

$$\mathcal{I}^{\mathrm{SL}}_{\mathrm{vp}}[\![E]\!] = (S_{E,\mathrm{SL}},\ \longrightarrow_{E,\mathrm{SL}},\ \langle \mathit{unfold}(E), \mathit{assign}(E, \{(\varepsilon, \mathit{const}(E))\}, \varepsilon)\rangle;\ \mathit{Var},\ \mathit{Act}'_{\mathrm{vp}} \times \mathit{BExp})$$

where:

- *$S_{E,\mathrm{SL}}$ is the least subset of $\mathcal{O}$ such that:*
  - *$\langle \mathit{unfold}(E), \mathit{assign}(E, \{(\varepsilon, \mathit{const}(E))\}, \varepsilon)\rangle \in S_{E,\mathrm{SL}}$;*
  - *if $O_1 \in S_{E,\mathrm{SL}}$ and $O_1 \xrightarrow{\alpha,\tilde{\lambda},\beta}_{\mathrm{SL}} O_2$, then $O_2 \in S_{E,\mathrm{SL}}$.*
- *$\longrightarrow_{E,\mathrm{SL}}$ is the restriction of $\longrightarrow_{\mathrm{SL}}$ defined in Table 8.3 to $S_{E,\mathrm{SL}} \times (\mathit{Act}'_{\mathrm{vp}} \times \mathit{BExp}) \times S_{E,\mathrm{SL}}$.* ■

$$\frac{E \xrightarrow{\alpha,\tilde{\lambda},\beta,\sigma'}_{\mathrm{vp}} E'}{\langle E,\sigma\rangle \xrightarrow{\alpha,\tilde{\lambda},\beta\sigma}_{\mathrm{SL}} \langle E', \sigma \triangleright \sigma'\rangle} \quad \text{if } \alpha \in \mathit{AType}_{\mathrm{U}}$$

$$\frac{E \xrightarrow{a?(\underline{x}),\tilde{\lambda},\mathit{true},\sigma'}_{\mathrm{vp}} E'}{\langle E,\sigma\rangle \xrightarrow{a?(\underline{x}),\tilde{\lambda},\mathit{true}}_{\mathrm{SL}} \langle E', \sigma \triangleright \sigma'\rangle}$$

$$\frac{E \xrightarrow{a!(\underline{e}),\tilde{\lambda},\mathit{true},\sigma'}_{\mathrm{vp}} E'}{\langle E,\sigma\rangle \xrightarrow{a!(\underline{e}\sigma),\tilde{\lambda},\mathit{true}}_{\mathrm{SL}} \langle E', \sigma \triangleright \sigma'\rangle}$$

Table 8.3: Rules for $\mathrm{EMPA}_{\mathrm{vp}}$ symbolic late semantics

Definition 8.15 *We define partial function $\mathit{Rate}_{\mathrm{SL}} : (\mathcal{O} \times \mathit{AType}_{\mathrm{vp}} \times \mathit{APLev}_{\mathrm{vp}} \times \mathcal{P}(\mathcal{O}) \times \mathit{BExp} \times \mathcal{P}_{\mathrm{fin}}(\mathit{BExp}) \times \mathit{BExp} \times \underline{\mathit{Var}} \times \underline{\mathit{Var}}) \nrightarrow \mathit{ARate}_{\mathrm{vp}}$ by*

$$\mathit{Rate}_{\mathrm{SL}}(O,\alpha,l,C,\beta,B,\beta',\underline{x},\underline{y}) = \mathit{Aggr}\{\!|\ \tilde{\lambda} \mid \exists \beta'', O'.\ O \xrightarrow{\alpha,\tilde{\lambda},\beta''}_{\mathrm{SL}} O' \wedge \mathit{PL}(<\alpha,\tilde{\lambda}>) = l \wedge O'\{\underline{y}/\underline{x}\} \in C \wedge (\beta \wedge \beta'' \Longrightarrow \bigvee B) \wedge (\beta' \Longrightarrow \beta'')\ |\!\}$$ ■

Definition 8.16 *Let $\mathcal{R} = \{\mathcal{B}^{\beta} \subseteq \mathcal{O} \times \mathcal{O} \mid \mathcal{B}^{\beta}$ equivalence $\wedge\ \beta \in \mathit{BExp}\}$ and let $\mathit{SLB}(\mathcal{R}) = \{\mathit{SLB}(\mathcal{R})^{\beta} \subseteq \mathcal{O} \times \mathcal{O} \mid \mathit{SLB}(\mathcal{R})^{\beta}$ equivalence $\wedge\ \beta \in \mathit{BExp}\}$ be defined by $(O_1, O_2) \in \mathit{SLB}(\mathcal{R})^{\beta}$ if and only if there exists $B \subseteq \mathit{BExp}$ such that for all $\beta' \in B$:*

(i) $\forall \alpha \in \mathit{AType}_{\mathrm{U}}.\ \forall l \in \mathit{APLev}_{\mathrm{vp}}.\ \forall C \in \mathcal{O}/\mathcal{B}^{\beta'}$

$$\mathit{Rate}_{\mathrm{SL}}(O_1,\alpha,l,C,\beta,B,\beta',\underline{x},\underline{x}) = \mathit{Rate}_{\mathrm{SL}}(O_2,\alpha,l,C,\beta,B,\beta',\underline{x},\underline{x})$$

with $\underline{x} \in \underline{\mathit{Var}}$ arbitrary.

(*ii*) $\forall a \in AName_{\mathrm{IO}}.\, \forall \underline{x}, \underline{y} \in \underline{Var}.\, \forall l \in APLev_{\mathrm{vp}}.\, \forall C \in \mathcal{O}/\mathcal{B}^{\beta'}$

$$Rate_{\mathrm{SL}}(O_1, a?(\underline{x}), l, C, \beta, B, \beta', \underline{x}, \underline{z}) = Rate_{\mathrm{SL}}(O_2, a?(\underline{y}), l, C, \beta, B, \beta', \underline{y}, \underline{z})$$

with $\underline{z} \in \underline{Var}$ *fresh.*

(*iii*) $\forall a \in AName_{\mathrm{IO}}.\, \forall \underline{e}_1, \underline{e}_2 \in \underline{Exp}.\, \forall l \in APLev_{\mathrm{vp}}.\, \forall C \in \mathcal{O}/\mathcal{B}^{\beta'}$

$$\begin{array}{rcl} Rate_{\mathrm{SL}}(O_1, a!(\underline{e}_1), l, C, \beta, B, \beta', \underline{x}, \underline{x}) & = & Rate_{\mathrm{SL}}(O_2, a!(\underline{e}_2), l, C, \beta, B, \beta', \underline{x}, \underline{x}) \\ \beta' & \Longrightarrow & \underline{e}_1 = \underline{e}_2 \end{array}$$

with $\underline{x} \in \underline{Var}$ *arbitrary.*

We say that $\mathcal{R}$ *is a strong symbolic late extended Markovian bisimulation (strong SLEMB) if and only if* $\mathcal{R} \subseteq SLB(\mathcal{R})$*, i.e.* $\mathcal{R}^\beta \subseteq SLB(\mathcal{R})^\beta$ *for each* $\beta \in BExp$*.* ■

Definition 8.17 *We define the strong symbolic late extended Markovian bisimulation equivalence (strong SLEMBE) with respect to* $\beta \in BExp$*, denoted* $\sim^\beta_{\mathrm{SLEMB}}$*, as the* β*-th component of the greatest fixed point of functional SLB.* ■

Since functional *SLB* is componentwise monotonic, the definition above is well founded. Furthermore, $\sim^\beta_{\mathrm{SLEMB}}$ turns out to be an equivalence relation.

We conclude by showing that the strong SLEMBE captures the strong CLEMBE. Similarly to [81], this has the advantage that the checking of strong CLEMBE can be reduced to the checking of strong SLEMBE on the more abstract (hence hopefully finite) semantic model.

Lemma 8.1 *We have that:*

(*i*) $O_1 \xrightarrow{\alpha,\tilde{\lambda}}_{\mathrm{CL},\rho} O_2$ *if and only if* $O_1 \xrightarrow{\alpha,\tilde{\lambda},\beta}_{\mathrm{SL}} O_2$ *whenever* $\rho(\beta) = true$*, where* $\alpha \in AType_{\mathrm{U}}$*.*

(*ii*) $O_1 \xrightarrow{\alpha,\tilde{\lambda}}_{\mathrm{CL},\rho} O_2$ *if and only if* $O_1 \xrightarrow{\alpha,\tilde{\lambda},true}_{\mathrm{SL}} O_2$*, where* $\alpha \in AType_{\mathrm{I}}$*.*

(*iii*) $O_1 \xrightarrow{a!(\underline{v}),\tilde{\lambda}}_{\mathrm{CL},\rho} O_2$ *if and only if* $O_1 \xrightarrow{a!(\underline{e}),\tilde{\lambda},true}_{\mathrm{SL}} O_2$ *whenever* $\rho(\underline{e}) = \underline{v}$*, where* $a \in AName_{\mathrm{IO}}$*.* ■

Lemma 8.2 *Let* $\mathcal{R} = \{\mathcal{B}^\beta \subseteq \mathcal{O} \times \mathcal{O} \mid \mathcal{B}^\beta \text{ equivalence} \wedge \beta \in BExp\}$ *and* $\mathcal{R}' = \{\mathcal{B}^\rho \subseteq \mathcal{O} \times \mathcal{O} \mid \mathcal{B}^\rho \text{ equivalence} \wedge \rho \in Eval\}$ *where* $\mathcal{B}^\rho = \{(O_1, O_2) \in \mathcal{O} \times \mathcal{O} \mid \exists \beta.\, \rho(\beta) = true \wedge (O_1, O_2) \in \mathcal{B}^\beta\}$*. If* $\mathcal{R}$ *is a strong SLEMB, then* $\mathcal{R}'$ *is a strong CLEMB.* ■

Lemma 8.3 *Let* $\mathcal{R} = \{\mathcal{B}^\rho \subseteq \mathcal{O} \times \mathcal{O} \mid \mathcal{B}^\rho \text{ equivalence} \wedge \rho \in Eval\}$ *and* $\mathcal{R}' = \{\mathcal{B}^\beta \subseteq \mathcal{O} \times \mathcal{O} \mid \mathcal{B}^\beta \text{ equivalence} \wedge \beta \in BExp\}$ *where* $\mathcal{B}^\beta = \{(O_1, O_2) \in \mathcal{O} \times \mathcal{O} \mid \forall \rho.\, \rho(\beta) = true \Longrightarrow (O_1, O_2) \in \mathcal{B}^\rho\}$*. If* $\mathcal{R}$ *is a strong CLEMB, then* $\mathcal{R}'$ *is a strong SLEMB.* ■

Theorem 8.3 $O_1 \sim^\beta_{\mathrm{SLEMB}} O_2$ *if and only if* $\forall \rho.\, \rho(\beta) = true \Longrightarrow O_1 \sim^\rho_{\mathrm{CLEMB}} O_2$*.* ■

function $SLEMBEcheck(E_1, \sigma_1, E_2, \sigma_2) =$
  $PES := \emptyset$;
  $close(E_1, E_2, \sigma_1 \triangleright \sigma_2, \emptyset)$;
  return(PES);

function $close(E_1, E_2, \sigma,\ Visited) =$
  if $(E_1, E_2) \notin Visited$ **then begin**
    $ATPLSet := \{(\alpha, l) \in AType_{\rm vp} \times APLev_{\rm vp} \mid \exists i \in \{1,2\}.\, \exists \tilde{\lambda}, \beta, \sigma', E'.\, E_i \xrightarrow{\alpha, \tilde{\lambda}, \beta, \sigma'}_{\rm vp} E' \wedge PL(<\alpha, \tilde{\lambda}>) = l\}$;
    $PES := PES \cup \{X_{E_1,E_2} = \bigwedge_{(\alpha,l) \in ATPLSet} match(E_1, E_2, \sigma, \alpha, l,\ Visited)\}$
  end;
  if $Visited \neq \emptyset$ **then**
    return$(X_{E_1,E_2}(\sigma))$;

function $match(E_1, E_2, \sigma, \alpha, l,\ Visited) =$
  if $Rate(E_1, a, l, \mathcal{G}_{\rm vp}) \neq Rate(E_2, a, l, \mathcal{G}_{\rm vp})$ **then**
    return$(false)$
  else
    if $(l = \omega)$ **then begin**
      for each $E_1 \xrightarrow{\tau, \infty_{\omega,w}, \beta_{1,i}, \sigma_{1,i}}_{\rm vp} E_{1,i}$ **do**
        for each $E_2 \xrightarrow{\tau, \infty_{\omega,w}, \beta_{2,j}, \sigma_{2,j}}_{\rm vp} E_{2,j}$ **do**
          $B_{i,j} = close(E_{1,i}, E_{2,j}, \sigma_{1,i} \triangleright \sigma_{2,j},\ Visited \cup \{(E_1, E_2)\})$;
      return$(\bigwedge_i(\beta_{1,i} \Longrightarrow \bigvee_j(\beta_{2,j} \wedge B_{i,j})) \wedge \bigwedge_j(\beta_{2,j} \Longrightarrow \bigvee_i(\beta_{1,j} \wedge B_{i,j})))$
    end else begin
      $\langle$let $\{C_1, C_2, \ldots, C_k\}$ be a partition of the derivative states of E_1 and E_2 which preserves aggregated rates$\rangle$;
      for each $h := 1$ **to** k **do**
        for each $E_1 \xrightarrow{\alpha_{1,i}, \tilde{\lambda}_{1,i}, true, \sigma_{1,i}}_{\rm vp} E_{1,i} \in C_h$ **do**
          for each $E_2 \xrightarrow{\alpha_{2,j}, \tilde{\lambda}_{2,j}, true, \sigma_{2,j}}_{\rm vp} E_{2,j} \in C_h$ **do**
            case $\alpha_{1,i}, \alpha_{2,j}$ **of**
              $\alpha_{1,i} = \alpha_{2,j} = a$:
                $B_{h,i,j} = close(E_{1,i}, E_{2,j}, \sigma_{1,i} \triangleright \sigma_{2,j},\ Visited \cup \{(E_1, E_2)\})$;
              $\alpha_{1,i} = a?(\underline{x}_1) \wedge \alpha_{2,j} = a?(\underline{x}_2)$:
                $\langle$let $\underline{z} \in \underline{Var}$ fresh$\rangle$;
                $B_{h,i,j} = \forall \underline{z}.\, close(E_{1,i}, E_{2,j}, \sigma_{1,i}\{\underline{z}/\underline{x}_1\} \triangleright \sigma_{2,j}\{\underline{z}/\underline{x}_2\},\ Visited \cup \{(E_1, E_2)\})$;
              $\alpha_{1,i} = a!(\underline{e}_1) \wedge \alpha_{2,j} = a!(\underline{e}_2)$:
                $B_{h,i,j} = close(E_{1,i}, E_{2,j}, \sigma_{1,i} \triangleright \sigma_{2,j},\ Visited \cup \{(E_1, E_2)\}) \wedge \underline{e}_1 = \underline{e}_2$
            end;
      return$(\bigvee_{1 \leq h \leq k} \bigwedge_{i,j} B_{h,i,j})$
    end

Table 8.4: Algorithm to check for strong SLEMBE

We conclude this section by presenting an algorithm to check for strong SLEMBE. Such an algorithm, inspired by that of [121], takes as input two finite state STGLA and produces as output a predicate equation system (PES for short). A PES is of the form $\{X_i = \Lambda_i \mid i \in I\}$ where for all $i \in I$ X_i is a predicate variable and Λ_i is a predicate built out of boolean expressions and data and predicate variables, operators such as conjunction, disjunction, and implication, and binders such as universal quantifier and abstraction. The idea underlying the algorithm is to reduce the problem of deciding strong SLEMBE to the problem of computing the greatest solution of a PES. Unfortunately, since a PES may involve arbitrary data and boolean expressions, it is in general not possible to automatically compute its greatest solution except for some special cases.

The algorithm, whose structure is shown in Table 8.4, assumes disjoint name spaces for the variables of the two STGLA and starts with their initial states and assignments. At each step, the algorithm makes use of transition relation $\longrightarrow_{\rm vp}$ and tries to match two states satisfying the necessary condition for $\sim_{\rm EMB}$ of Prop. 5.2 by creating a corresponding predicate variable and predicate equation. As in [121], it can be shown that the correctness of the algorithm follows from the following result.

Theorem 8.4 *Let $\{X_{E_{1,i},E_{2,i}} = \Lambda_{E_{1,i},E_{2,i}} \mid i \in I\}$ be the PES returned by the algorithm of Table 8.4 when applied to two finite state STGLA with disjoint name spaces.*

- *If η is a symbolic solution of the PES and $\eta(X_{E_{1,i},E_{2,i}})(\sigma) = \beta$, then $\langle E_{1,i}, \sigma\rangle \sim^{\beta}_{\rm SLEMB} \langle E_{2,i}, \sigma\rangle$.*
- *If η is the greatest symbolic solution of the PES and $\langle E_{1,i}, \sigma_{1,i}\rangle \sim^{\beta}_{\rm SLEMB} \langle E_{2,i}, \sigma_{2,i}\rangle$, then $\beta \Longrightarrow \eta(X_{E_{1,i},E_{2,i}})(\sigma_{1,i} \triangleright \sigma_{2,i})$.* ■

8.2.6 Compatibility Results

We conclude by showing some results which prove that $\text{EMPA}_{\rm vp}$ is a conservative extension of EMPA.

Proposition 8.2 *Let $E \in \mathcal{G}_{\rm vp}$ have no actions whose type belongs to $AType_I \cup AType_O$, no conditional operators, and no parametrized constants. Then $\mathcal{I}_{\rm vp}[\![E]\!]$ is isomorphic to $\mathcal{I}[\![E]\!]$ up to (true) boolean guards and (empty) assignments.* ■

Proposition 8.3 *Let $E \in \mathcal{G}_{\rm vp}$ have no actions whose type belongs to $AType_I \cup AType_O$, no conditional operators, and no parametrized constants. Then $\mathcal{F}_{\rm vp}[\![E]\!]$ is isomorphic to $\mathcal{F}[\![E]\!]$ up to (true) boolean guards and (empty) assignments.* ■

Proposition 8.4 *Let $E \in \mathcal{E}_{\rm vp}$ have no actions whose type belongs to $AType_I \cup AType_O$, no conditional operators, and no parametrized constants. Then $\mathcal{M}_{\rm vp}[\![E]\!]$ is p-isomorphic to $\mathcal{M}[\![E]\!]$.* ■

Proposition 8.5 *Let $E_1, E_2 \in \mathcal{E}_{\rm vp}$ have no actions whose type belongs to $AType_I \cup AType_O$, no conditional operators, and no parametrized constants. Then $\langle E_1, \sigma_1\rangle \sim^{\beta}_{\rm SLEMB} \langle E_2, \sigma_2\rangle$ if and only if $E_1 \sim_{\rm EMB} E_2$, for arbitrary $\beta \in BExp$ and $\sigma_1, \sigma_2 \in Sub$.* ■

8.3 Modeling Generally Distributed Durations

The purpose of this section is to show that value passing, besides allowing to model systems where data plays a fundamental role, permits dealing with generally distributed durations thereby overcoming the limitation to exponentially timed actions. To achieve this, we consider a discrete time subcalculus of EMPA_{vp} where only immediate (and passive) actions are allowed and we proceed as follows. When modeling a system with generally distributed durations, clock variables are employed in the related terms (of the subcalculus) in order to mark the discrete time steps. The timed events of the system are treated by means of suitable variables which store their occurrence times, denoted by appropriate value passing based expressions possibly involving the invocation of general probability distribution functions. Clock variables are thus used in the terms to detect when the occurrence time of a timed event has come.

As an example, consider a QS $M/G/1$ with arrival rate λ and service time following a Gaussian distribution with mean μ and standard deviation σ. This QS can be modeled as follows:

$$
\begin{array}{rcl}
QS_{M/G/1} & \triangleq & Arrivals(0, \exp(\lambda)) \parallel_{\{a,tick\}} (Queue(0) \parallel_{\{d,tick\}} Server) \\
Arrivals(clock, a\_time) & \triangleq & \text{if } (clock = a\_time, p_a) \text{ then} \\
 & & \quad <a, \infty_{2,1}>.<tick, \infty_{1,1}>.Arrivals(0, \exp(\lambda)) \\
 & & \text{else} \\
 & & \quad <tick, \infty_{1,1}>.Arrivals(clock + 1, a\_time) \\
Queue(h) & \triangleq & \text{if } (h = 0, p_q) \text{ then} \\
 & & \quad <a, *>.(<d, *>.<tick, *>.Queue(h) + <tick, *>.Queue(h + 1)) + \\
 & & \quad <tick, *>.Queue(h) \\
 & & \text{else} \\
 & & \quad <a, *>.(<d, *>.<tick, *>.Queue(h) + <tick, *>.Queue(h + 1)) + \\
 & & \quad <d, *>.(<a, *>.<tick, *>.Queue(h) + <tick, *>.Queue(h - 1)) + \\
 & & \quad <tick, *>.Queue(h) \\
Server & \triangleq & <d, \infty_{2,1}>.<tick, *>.Server'(0, \text{gauss}(\mu, \sigma)) + \\
 & & <tick, *>.Server \\
Server'(clock, s\_time) & \triangleq & \text{if } (clock = s\_time, p_s) \text{ then} \\
 & & \quad <s, \infty_{2,1}>.Server \\
 & & \text{else} \\
 & & \quad <tick, *>.Server'(clock + 1, s\_time)
\end{array}
$$

where a stands for arrive, d for deliver, s for serve, and *tick* represents a clock tick. Note that all the three components synchronize on action type *tick*, whose associated priority level is less than the priority level associated with actions having type a, d, or s so that these actions are not erroneously prevented from being executed when enabled. Component *Arrivals* schedules the time of the next arrival by sampling a number from an exponential distribution and storing it into variable *a_time*, while component *Server* schedules the

time of the next service completion by sampling a number from a Gaussian distribution and storing it into variable s_time. Action a (s) is executed only after a_time (s_time) time units have elapsed.

It is worth noting that, since in this framework semantic models are symbolic, expressions like gauss(μ, σ) are not evaluated. This means that MCs underlying EMPA_{vp} terms where generally distributed durations come into play lack information about the value of the durations. Therefore, whenever using EMPA_{vp} to model a system with generally distributed durations, a numerical performance analysis cannot be carried out on the underlying MCs and we have to resort to a simulative analysis of performance since in this case expressions are evaluated, hence durations encoded within assignments are taken into account.

Chapter 9

TwoTowers: A Software Tool Based on EMPA

This chapter describes our efforts to provide automatic support for the integrated approach of Fig. 1.1 through the implementation of a software tool called TwoTowers [23]. To use TwoTowers, the designer first specifies the system as a set of EMPA_{vp} constant defining equations. Since EMPA_{vp} supports compositional modeling, the designer can develop the algebraic representation of the system in a modular way; subsystems can be modeled separately, then combined using the appropriate operators. Another nice feature is that many functional properties of a system may be examined independently of performance properties and vice versa. Our implementation of the first phase of the integrated approach through TwoTowers exploits this fact by using two existing tools (that we have retargeted to EMPA_{vp}) for the analysis of these types of properties. The Concurrency Workbench of North Carolina (CWB-NC) [56] provides a variety of different types of analysis (e.g., model checking, equivalence checking, preorder checking) of the functional behavior of systems, while the Markov Chain Analyzer (MarCA) [180] conducts stationary and transient performance analysis. As far as the implementation of the second phase of the integrated approach is concerned, the analysis of GSPNs obtained from EMPA_{vp} terms is conducted through the Graphical editor and analyzer for timed and Stochastic Petri Nets (GreatSPN) [46]. Fig. 9.1 shows that the development of TwoTowers can be seen as an exercise in component based software engineering with emphasis on tool reuse.

The purpose of TwoTowers is similar to that of other stochastically timed process algebra based tools, such as the PEPA Workbench [69] and the TIPP-tool [91]. However, besides the difference in the expressive power of the algebras used by these three tools and the fact that TwoTowers deals also with stochastically timed Petri nets, the underlying philosophies are quite different. In particular, as we have seen TwoTowers profits from well known software tools to carry out the functional and performance analysis. This is advantageous both from the point of view of the implementors and from the point of view of the users. We did not need to write code for implementing most of the analysis routines, thereby making the development of TwoTowers itself easier and faster, and we provided users with a full range of automated techniques implemented in widely used tools.

The architecture of TwoTowers is depicted in Fig. 9.2. TwoTowers is composed of 20,000 lines of code

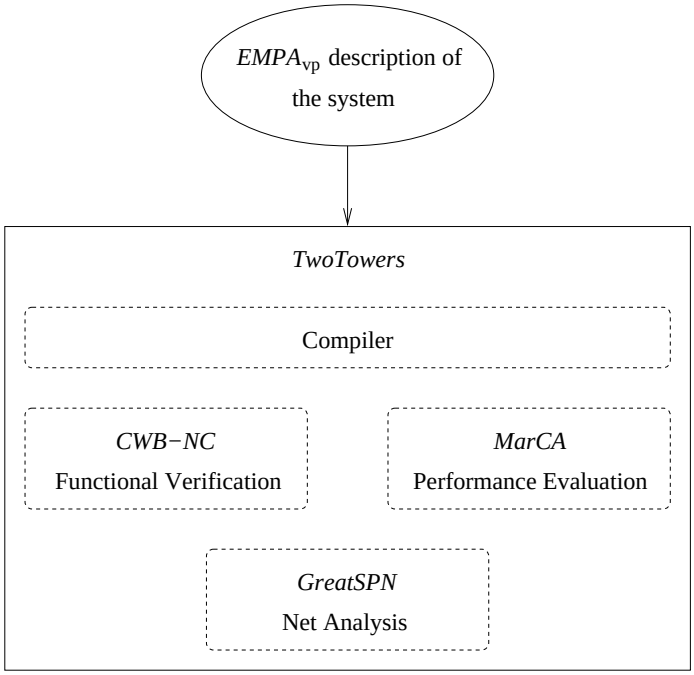


Figure 9.1: TwoTowers builds on CWB-NC, MarCA, and GreatSPN

(excluding the tools it builds on) divided into a graphical user interface, a tool driver, an integrated kernel, a functional kernel, a performance kernel, and a net kernel. In this chapter we illustrate the various components of TwoTowers.

9.1 Graphical User Interface

The graphical user interface is written in Tcl/Tk [146] and allows the user to edit EMPA$_{vp}$ specifications and auxiliary files, compile EMPA$_{vp}$ specifications, and run the various analysis routines. More precisely, the graphical user interface is composed of an editing/display area, eight popup menus, and three message fields: one reporting the path name of the file currently loaded in the editing/display area (initially it indicates the directory from which the tool has been invoked), one reporting a brief description of the command which shows up when browsing on the menus, and one reporting the result of the last invoked command in a concise form as well as the execution time. After the successful execution of a compilation or analysis routine, the result of the computation is shown also in extended form by means of a suitable window.

The first menu allows the user to open a new TwoTowers window or close the current one. Moreover, it permits the user to call for CWB-NC, MarCA, and GreatSPN graphical user interfaces in order to use directly such tools.

The second menu allows the user to deal with TwoTowers related files. The extensions for input files are .empa for EMPA$_{vp}$ specifications, .sim for the specification of performance measures and other information

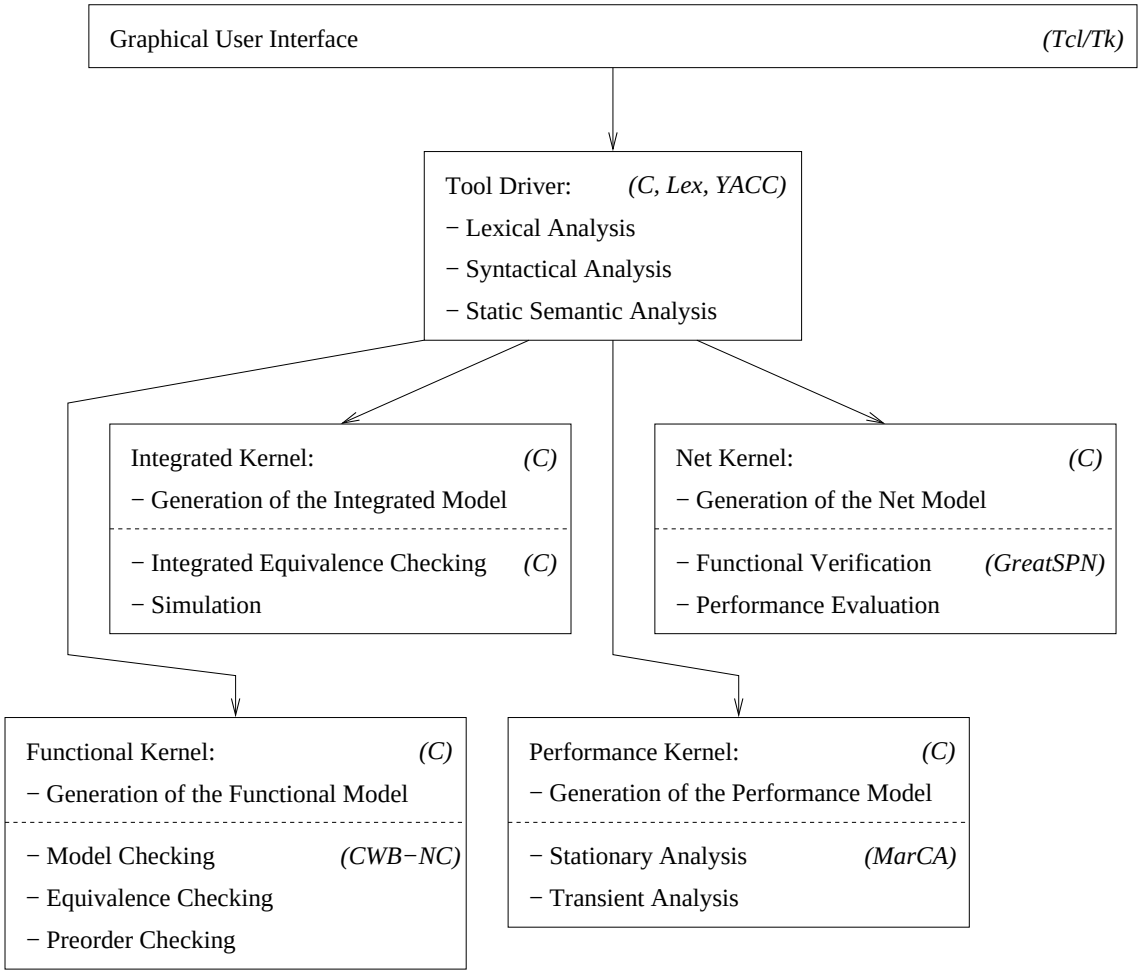


Figure 9.2: Architecture of TwoTowers

for simulation, .trace for the specification trace files for simulation, .mu for the specification of modal logic formulas, and .rew for the specification of rewards. The extensions for output files are .lis for listings resulting from parsing operations, .size for semantic model sizes, .iism for integrated interleaving semantic models, .iver for integrated verification results, .est for simulation estimates, .fsm for functional semantic models, .fsm.empa for functional semantic models in the format expected by CWB-NC, .fver for functional verification results, .psm for performance semantic models, .mc and .ip for performance semantic models and initial state probabilities in the format expected by MarCA, .dis for state probability distributions, .mea for performance measures, .insm for integrated net semantic models, .net and .def for the structure of integrated net semantic models and information about marking dependent rates in the format expected by GreatSPN, .pinv for P-invariants, .tinv for T-invariants, .tok for the average and maximum numbers of tokens in net places, and .thr for the maximum throughput of net transitions.

The third menu permits the user to check for the correctness of an EMPA_{vp} specification and compile

an EMPA_{vp} specification into the corresponding integrated interleaving semantic model, functional semantic model, performance semantic model, or integrated net semantic model. In case of compilation, the size of the model is shown in the field reporting the result of the compilation. It is also possible to compute just the size of a specified semantic model without writing the model into a file.

The fourth menu allows the user to call for the routine for $\sim_{\text{EMB}}$ checking as well as the routine for the simulative analysis of the performance of an EMPA_{vp} specification.

The fifth menu permits the user to obtain the functional semantic model of an EMPA_{vp} specification in the format expected by CWB-NC and to call for routines for model checking, equivalence checking, and preorder checking which are implemented in CWB-NC.

The sixth menu allows the user to derive the performance semantic model of an EMPA_{vp} specification in the format expected by MarCA and to call for routines for calculating stationary/transient state probability distributions or reward based performance measures which are implemented in MarCA.

The seventh menu allows the user to derive the integrated net semantic model of an EMPA_{vp} specification in the format expected by GreatSPN and to call for routines for computing P-invariants, T-invariants, the average and maximum numbers of tokens in net places, the maximum throughput of net transitions, and reward based performance measures which are implemented in GreatSPN.

Finally, the eighth menu helps the user with the on line TwoTowers manual.

We conclude by recalling that there are two cases in which the execution of all the menu items above fails: not enough memory and inability to open a given file. Other crashes specific to particular menu items will be indicated when illustrating those items.

9.2 Tool Driver

The tool driver, which is written in C [114], receives commands from the graphical user interface and passes the control to the appropriate kernel after invoking a parser for EMPA_{vp} specifications built by means of Lex [120] and YACC [110]. The syntax for EMPA_{vp} adopted in TwoTowers is that given in Def. 8.1 together with the rules for operator associativity and precedence given in Sect. 8.2.2 (which can be altered by means of parentheses), with the following differences:

- Each constant definition is of the form

  $$@\, A(type_1\ par_1 := val_1, \ldots, type_n\ par_n := val_n; type_{n+1}\ var_1, \ldots, type_{n+m}\ var_m) = E$$

  where the header is composed of a comma separated list of formal parameters and a comma separated list of local variables, with the two lists separated by a semicolon, and the initial assignment of formal parameters is allowed only for the initial constant. In the right hand side of the constant definition, each action type set and action type relabeling function must be denoted by its name (or by symbol "‘" if empty). Moreover, the null term is denoted by "nil", the invisible action type is denoted by "tau",

the immediate rate is denoted by "inf(_, _)", the functional relabeling operator is denoted by "˜", and the parallel composition operator is denoted by "_ ||− _ −|| _".

- Each action type set definition is of the form

$$\#\, L = a_1\, a_2\, \ldots\, a_n$$

- Each action type relabeling function definition is of the form

$$\$\, \varphi = a_1 \tilde{} b_1\, a_2 \tilde{} b_2\, \ldots\, a_n \tilde{} b_n$$

- Comments can be inserted wherever in a specification and start with symbol "%".

- Each constant, action type, action type set, action type relabeling function, formal parameter, or local variable identifier is a sequence of letters, digits, underscores, hyphens, and primes beginning with a letter. Identifiers cannot be keywords [1] and can be defined only once in a given specification, except for formal parameters or local variables provided that they are declared within the scope of distinct constants.

- The order in which constants, action type sets, and action type relabeling functions are listed in a specification is irrelevant, with the exception that the first constant definition encountered in the specification is assumed to be the initial one.

Note that each formal parameter and local variable is declared in the constant definition header to be of a certain type. Since these are the only objects that can occur in possible input/output actions of the constant definition body, also action parameters turn out to be typed. The admitted types are given by the following syntax

type ::= int | real | bool | list(*type*) | array(*length*, *type*) | record(*type field_name*$_1$, ..., *type field_name*$_n$)

The related operators, which are composed of two symbols if infix in order to distinguish them from the algebraic operators of EMPA$_{\rm vp}$, are the following ones:

- Zeroary operators are integer numbers, real numbers, boolean values "ff" for false and "tt" for true, and the empty list "[ ]".

- Operators "_++_" for addition, "_−−_" for subtraction, and "_**_" for multiplication can be applied to int/real typed arguments and produce an int/real typed result. The result has type int if and only if both arguments have type int.

[1] Keywords are: nil, tau, inf, if, then, else, int, real, mod, min, max, abs, ceil, floor, power, epower, loge, log10, sqrt, sin, cos, exp, gauss, bool, tt, ff, not, A, AG, AF, E, EG, EF, U, W, V, D, list, first, tail, insert, remove, length, array, read, write, record, get, put.

- Operator "_//_" for division can be applied to two int/real typed arguments, with the second one different from zero, and produces a real typed result.
- Operator "mod(_, _)" computes the modulus of its first int typed argument w.r.t. its second int typed argument which must be grater than zero.
- Operator "min(_, _)" ("max(_, _)") computes the minimum (maximum) of its two int/real typed arguments.
- Operator "abs(_)" computes the absolute value of its int/real typed argument.
- Operator "ceil(_)" ("floor(_)") computes the smallest (greatest) integer greater (smaller) than its int/real typed argument.
- Operator "power(_, _)" ("epower(_)") computes the power of its first int/real typed argument (number e) to its second (single) int/real typed argument. It cannot be applied to a pair of arguments such that the former is zero and the latter is non positive, or the former is negative and the latter is real.
- Operator "loge(_)" ("log10(_)") computes the natural logarithm (base 10 logarithm) of its int/real typed argument which must be grater than zero.
- Operator "sqrt(_)" computes the square root of its int/real typed argument which must be nonnegative.
- Operator "sin(_)" ("cos(_)") computes the sine (cosine) of its int/real typed argument expressed in radians.
- Operators "exp(_)" and "gauss(_, _)" generate a random number following an exponential or Gaussian distribution, respectively, according to the specified int/real typed arguments which must be greater than zero.
- Operators "_&& _" for logical and, "_|| _" for logical or, and "!! _" for logical not can be applied to bool typed arguments and produce a bool typed result. The first two operators are subject to shortcircuitation.
- Operators "_ == _" for equal to, "_!= _" for not equal to, "_ << _" for less than, "_ <= _" for less than or equal to, "_ >> _" for greater than, and "_ >= _" for greater than or equal to can be applied to arguments of the same type (int and real types are assumed to be compatible with each other) including lists and arrays and produce a bool typed result.
- Operator "[_, _, . . . , _]" creates a list provided that all of its arguments have the same type.
- Operator "first(_)" ("tail(_)") returns the first element of its list typed argument (its list typed argument without its first element) which must be nonempty.

- Operator "insert(_, _)" ("remove(_, _)") inserts its first argument into its second list typed argument (removes the element whose position is specified by its first int typed argument from its second list typed argument) provided that the first argument has the expected type (provided that the specified position exists). Insertions respect a lexicographical ordering on arbitrarily nested lists.
- Operator "length(_)" returns the length of its list typed argument.
- Operator "{ _, _, ..., _ }" creates an array provided that all of its arguments have the same type.
- Operator "read(_, _)" ("write(_, _, _)") reads the element whose position is specified by its first int typed argument (writes its second argument in the position specified by its first int typed argument) from its second array typed argument (of its third array typed argument) provided that the specified position exists (and the second argument has the expected type). The index range for an array varies from 0 to the array length decremented by 1.
- Operator "(_, _, ..., _)" creates a record provided that all of its arguments have the expected type.
- Operator "get(_, _)" ("put(_, _, _)") gets the field whose name is specified by its first argument (puts its second argument in the field whose name is specified by its first argument) from its second record typed argument (of its third record typed argument) provided that the field exists (and the second argument has the expected type).

In order to avoid ambiguities, we assume the binary infix operators to be left associative (except for the relational ones) and we introduce the following operator precedence relation: times = divide > plus = minus > equal to = not equal to = less than = less than or equal to = greater than = greater than or equal to > not > and = or. Parentheses can be used to alter associativity and precedence.

While reading an EMPA_{vp} specification, the parser produces a listing file (.lis) which is terminated with the indication of the number of errors and warnings. In such a listing file, possible errors/warnings are pinpointed in order to ease their correction. The errors/warnings handled by the parser are the following ones:

- Illegal character, i.e. any character different from $* \; ` \; @ \; \# \; \$ \; = \; , \; (\;) \; ; \; < \; > \; ? \; ! \; . \; / \; \tilde{} \; + \; [\;] \; \{ \; \} \; - \; \& \; | \;$: tab newline %.
- Redefinition of constant, action type set, or action type relabeling function.
- Redeclaration of variable in the same constant header or record field.
- Definition of constant, action type set, action type relabeling function, variable, or record field as another object among these ones.
- Redefinition of constant, action type, action type set, action type relabeling function, variable, or record field as another object among these ones except action type.

- Use of constant, action type, action type set, action type relabeling function, variable, or record field as another object among these ones.
- Ill typed constant parameters, assignment, expression, list/array/record of values, or action parameters. [2]
- Length of array not integer or positive.
- Right hand side expression of an assignment occurring in the header of the initial constant containing variables.
- Exponentially timed action with nonpositive rate.
- Immediate action with noninteger or nonpositive priority level or nonpositive weight.
- Input action with specified rate.
- Probability value of conditional operator out of range $\mathbb{R}_{]0,1[}$.
- Action type tau occurring in an action type set or in an action type relabeling.
- Action type occurring several times in an action type set (this is just a warning).
- Action type relabeling violating the uniqueness of an action type relabeling function.
- Identical action type relabeling occurring in an action type relabeling function (this is just a warning if the uniqueness of the action type relabeling function is not violated).
- Action type relabeling occurring several times in an action type relabeling function (this is just a warning).
- Unexpected character and illegal definition. These errors, which are different from all the previous ones (e.g.: use of a keyword as constant identifier), are treated by indicating in the listing file the point at which they are detected and the point from which parsing is resumed upon encountering "@", "#", "$", ".", "/", "~", "then", "else", "+", "⊫, ⊣", or ")".

After reading an EMPA_{vp} specification, the following errors/warnings are handled:

- No constant definition.
- Use of undefined constant, action type set, action type relabeling function, formal parameter, or record field.

[2] Type checking action parameters allows occurrences of input/output internal actions, which are not permitted by Def. 8.1, to be detected.

- Definition of unused constant, unused action type set, unused action type relabeling function, redundant action type relabeling, variable, or record field (this is just a warning).

- Formal parameter referenced in an input action. This warning is reported to indicate the fact that, in case of simulation, the value passed in that parameter is lost after the execution of that action if it is involved in a synchronization with an output action. Local variables should be used in input actions to get values via synchronization.

- Local variable referenced before occurring in an input action. This warning is reported to indicate the fact that, in case of simulation, the value associated with the local variable the first time it is accessed is a default one among 0 if the variable has type int, 0.0 if the variable has type real, ff if the variable has type bool, [ ] if the variable has type list, or an array/record initialized according to the type of its components/fields otherwise.

- Ill typed action type relabeling.

- Action type occurring only in action type sets and left hand sides of action type relabelings (this is just a warning).

- Violation of alternative composition guardedness.

- Violation of guardedness. This is checked for each constant defining equation by making sure that its right hand side cannot be expanded, by recursively rewriting only constants critical w.r.t. guardedness (i.e., not appearing within the scope of an action prefix operator) with the right hand side of their defining equation, in such a way that the original constant does not appear in the resulting term within the scope of an action prefix operator. The rewriting is actually carried out by having, for each constant defining equation, a list of constants occurring in the right hand side which are critical w.r.t. guardedness. Starting from the right hand side of a given constant defining equation, we recursively consider for each constant critical w.r.t. guardedness the related defining equation until the original constant is encountered outside the scope of an action prefix operator (thereby raising a violation of guardedness) or all the involved constants have been examined.

- Possible violation of finiteness. This is checked for each constant defining equation by making sure that its right hand side cannot be expanded, by initially rewriting only constants critical w.r.t. finiteness (i.e., appearing within the scope of a static operator) with the right hand side of their defining equation, in such a way that the original constant appears in the resulting term within the scope of a static operator. The rewriting is actually carried out by having, for each constant defining equation, a list of constants occurring in the right hand side which are critical w.r.t. finiteness as well as a list of all the constants occurring in the right hand side. Starting from the right hand side of a given constant defining equation, we initially consider for each constant critical w.r.t. finiteness and then recursively

for each occurring constant the related defining equation until the original constant is encountered (thereby raising a possible violation of finiteness) or all the involved constants have been examined.

If the EMPA_{vp} specification is correct, i.e. it contains no errors, the tool driver passes the control to the appropriate kernel.

9.3 Integrated Kernel

The integrated kernel, which is implemented in C, contains the routine to generate in a breadth first manner the integrated interleaving semantic model of correct, finite state EMPA_{vp} specifications according to the rules in Table 8.1. While generating an integrated interleaving semantic model, a file (.iism) is written which shows each state together with the related outgoing transitions. At the end of the file it is reported the number of states (classified as tangible, vanishing, performance open, or absorbing) as well as the number of transitions (classified as observable or invisible according to their types, and as exponentially timed, immediate, or passive according to their rates). It is worth observing that the two layer definition of the semantics, with the calculation of the whole multiset of potential moves at once, is well suited to the implementation of the semantics itself.

The states are collected in a hash table for efficient retrieval, which is needed when generating the derivative state of a transition in order to check whether the state is a new one or not. The bucket for a given state contains the corresponding term in concise prefix notation, the list of its outgoing transitions, and other information. Each state is represented by the corresponding term in prefix notation in order to avoid storing the related abstract syntax tree. The term is written in a concise way by encoding every algebraic operator as a single character and every constant invocation, action type set identifier, action type relabeling function identifier, action, expression, and probability through a serial number stored in the hash table bucket for the symbol at hand (a further hash table is employed to efficiently retrieve the symbol associated with a given serial code). The encoding is defined by structural induction as follows:

$$
\begin{aligned}
code(\text{nil}) &= \_ \\
code(<a, \tilde{\lambda}>.E) &= .\ serial(<a, \tilde{\lambda}>)\ code(E) \\
code(E/L) &= /\, code(E)\ serial(L) \\
code(E\tilde{}\varphi) &= \tilde{}\ code(E)\ serial(\varphi) \\
code(E_1 + E_2) &= +\ code(E_1)\ code(E_2) \\
code(\text{if } (\beta, p) \text{ then } E_1 \text{ else } E_2) &= :\ serial(\beta)\ serial(p)\ code(E_1)\ code(E_2) \\
code(E_1 \,\|\!-\, S \,-\!\|\, E_2) &= |\ code(E_1)\ serial(S)\ code(E_2) \\
code(A(e_1, e_2, \ldots, e_n)) &= serial(A(e_1, e_2, \ldots, e_n))
\end{aligned}
$$

Since the semantic rules for EMPA_{vp} rely on function *unfold* of Def. 8.3, aliasing is avoided. Aliasing is a consequence of the mechanism of constant definition and may result in the creation of unnecessary states.

As an example, suppose we do not use function *unfold* and consider the following EMPA_{vp} specification

$$\begin{aligned} @\,A &= B \,\|\!-\, `\, -\!\|\, C \\ @\,B &= <b,1>.B \\ @\,C &= <c,2>.C \end{aligned}$$

Its initial state would be A. While building its integrated interleaving semantic model, state $B \,\|\!-\, `\, -\!\|\, C$ would be encountered after generating both transitions for the two actions occurring in the specification. If it were not detected that $B \,\|\!-\, `\, -\!\|\, C$ is the right hand side of the definition of constant A, a new state would be generated. In our framework, the detection is possible because the initial state is

$$\mathit{unfold}(A) = <b,1>.B \,\|\!-\, `\, -\!\|\, <c,2>.C$$

and state $B \,\|\!-\, `\, -\!\|\, C$ is never encountered. To make the application of function *unfold* efficient, before the generation of the semantic model we compute for each constant the unfolding of the right hand side of its defining equation and we store a pointer to it into the symbol table bucket for the constant.

The state space generation employs the transition caching strategy proposed in [54] to store in an additional hash table certain semantic information during model construction thereby avoiding recomputation of this information caused by parallel compositions. More precisely, the idea is that, given state $E_1 \,\|\!-\, S \,-\!\|\, E_2$, this can evolve into $E_1' \,\|\!-\, S \,-\!\|\, E_2$, $E_1 \,\|\!-\, S \,-\!\|\, E_2'$, or $E_1'' \,\|\!-\, S \,-\!\|\, E_2''$; due to the first two cases, it is worth caching into a suitable hash table the potential moves of both substates E_1 and E_2 in order to avoid their recomputation when considering the derivative states above. Transition caching is advantageous not only from the time efficiency viewpoint but also from the memory utilization viewpoint. If we consider again the terms above, we see that E_1 and E_2, beside occurring in the original state, may occur in the derivative states as well. This can be exploited by representing them through serial codes in the concise prefix notation for $E_1 \,\|\!-\, S \,-\!\|\, E_2$ and its derivatives. Since the additional hash table for the substates of parallel compositions contains information that can be easily recomputed, it is periodically cleared in case of shortage of memory.

The purpose of the integrated kernel is to make those analyses that require both functional and performance information and therefore cannot be conducted by factoring out information to be passed to either the functional or performance kernels. In other words, the analysis routines implemented in the integrated kernel need to work on the integrated interleaving semantic model.

More precisely, the integrated kernel contains a routine to check two correct, finite state EMPA_{vp} specifications for integrated equivalence. If the two specifications do not contain any value passing related feature, strong EMBE is checked by means of the algorithm whose structure is shown in Table 5.2. Such an algorithm repeatedly partitions the union of the state spaces of the two EMPA specifications according to the stratified definition of $\sim_{\text{EMB}}$. Starting from the initial partition given by the set of blocks of states equivalent according to $\sim_{\text{EMB},1}$ (i.e., the necessary condition for $\sim_{\text{EMB}}$ given by Prop. 5.2) and a single initial splitter composed of all of these blocks, the algorithm refines the current partition as long as the two initial states of the two EMPA specifications are in the same block and there are composite splitters. At each refining step, a composite splitter is selected and the current partition is refined w.r.t. one of the blocks of the selected

composite splitter containing at most half of the states in the composite splitter. The refinement is carried out by singling out the blocks that need to be refined by only considering the transitions entering the selected block. Thanks to this technique for choosing the refining block and identifying the blocks to be refined, the algorithm requires $O(m \log n)$ time and $O(m+n)$ space where n is the number of states and m is the number of transitions of the integrated interleaving semantic model [153]. If instead the two specifications contain value passing related features, strong SLEMBE is checked by means of the algorithm whose structure is shown in Table 8.4. In this case, a predicate equation system is output instead of a direct answer.

Moreover, the integrated kernel contains a routine for the simulative analysis of the performance of a correct, deadlock free, performance closed EMPA_{vp} specification according to the method of independent replications [188], which consists of statistically inferring performance measures from several independent executions of the specification under consideration. At each step of each execution, the transitions for the current state are computed according to the rules in Table 8.1 and one of them (together with the related derivative state) is chosen according to the evaluation of boolean guards and either the race policy or the preselection policy, then executed together with its assignments. Pseudo random number generators [177] are employed to solve the choice. In the former case, a pseudo random number generator is used to sample the duration of each exponentially timed transition of the current state, then the one with the least duration is chosen. In the latter case, after partitioning $\mathbb{R}_{[0,1]}$ in as many subintervals as there are enabled immediate transitions, where the length of each subinterval is proportional to the weight of the corresponding enabled immediate transition, a pseudo random number generator is used to sample a number uniformly distributed in $\mathbb{R}_{[0,1]}$ which determines the immediate transition to execute depending on the subinterval to which it belongs. Since at any time during the simulation only the current state of the integrated interleaving semantic model is needed, we are not required to build the whole state space a priori. Providing the outgoing transitions for the current state without having the whole state space generated is naturally supported as the semantics for EMPA_{vp} is defined in an operational style [11]. Actually, during simulation the hash table storing the states is periodically cleared in case of shortage of memory. Therefore, simulation can in general be employed to study the performance of specifications with a huge or even infinite state space.

The simulative analysis of an EMPA_{vp} specification requires the provision of additional information. This information, given in a suitable file (.sim), consists of the indication of the termination condition, the specification of the performance measures of interest, and possible trace files to be used in case of trace driven simulation. The estimates for the performance measures and the related 90% confidence intervals are written into a suitable file (.est).

The structure of the auxiliary file for simulation is as follows:

- The termination condition is of the form

$$@\, a\, n_1\, n_2$$

where a is an action type occurring in the EMPA_{vp} specification, $n_1 \geq 1$ is the number of times a must be executed before terminating each simulation run, and $1 \leq n_2 \leq 30$ is the number of simulation runs.

- Each performance measure has one of the following forms:

$$\begin{aligned} \#\, id &= E(measure\_expr, m_1, m_2) \\ \#\, id &= V(measure\_expr, m_1, m_2) \\ \#\, id &= D(measure\_expr, m_1, m_2, m_3) \end{aligned}$$

where *id* is the measure identifier and *measure_expr* is an expression involving action types occurring in the EMPA_{vp} specification, numbers, and operators such as addition, subtraction, multiplication, division, modulus, minimum, maximum, absolute value, ceil, floor, power, logarithm, square root, sine, and cosine. In the first (second) case, the expected value (variance) of the measure is estimated together with the related 90% confidence interval over the period of time ranging from the instant when the termination condition related action type a is executed for the m_1-th time to the instant when a is executed for the m_2-th time ($0 \leq m_1 < m_2 \leq n_1$). In the third case, instead, the distribution of the measure is estimated together with the related 90% confidence interval over the same period of time; the estimation is done every m_3 executions of a starting from the m_1-th execution of a ($m_2 - m_1$ multiple of m_3). Each action type b occurring in *measure_expr* has an accumulator and a counter associated with it and is followed in the expression by a pair as in

$$b(reward\_expr, flag)$$

where *reward_expr* is an expression involving variables occurring in the EMPA_{vp} specification preceded by their scope and locality (e.g. $l.r.A.x$ where l stands for $\swarrow$ and r stands for $\searrow$), numbers, the same operators as *measure_expr*, and operators such as first, tail, length, read. The meaning is that whenever an action with type b is executed, the b related accumulator is increased by *reward_expr* and the b related counter is increased by one. At the end of each run, if the b related sample is purely cumulative (*flag* is "‘") then the value to be used when evaluating *measure_expr* is the value of the b related accumulator, otherwise (*flag* is "/") the value of the b related accumulator must be divided by the value of the b related counter. Purely cumulative samples are useful to derive measures such as throughput and utilization, while the other samples are useful to estimate average response times. As can be noted, in case of simulation the technique of rewards for the algebraic specification of performance measures is not used, as information about state probabilities is no longer available or meaningless in case of general distributions. However, the method we have adopted can be seen as an extension of rewards because the collection of samples is associated with the execution of actions.

- Each trace specification is of the form

$$\$\, trace\_expr = trace\_file$$

where *trace_expr* is an expression like *reward_expr* which additionally can contain (and has as outermost

operator) exp or gauss, while *trace_file* is the complete path name of the trace file (.trace) from which values of *trace_expr* must be taken during simulation instead of being produced by pseudo random number generators.

While reading a .sim file, the parser produces a listing file (.lis) where the following specific errors/warnings are reported:

- Identifier involved in the termination condition or in a measure expression not an observable action type.
- Length of simulation run not integer or positive.
- Number of simulation runs not integer or within range $1, \ldots, 30$.
- Identifier of a measure already used for other purposes.
- Begin of observation period not integer, negative, or not less than length of simulation run.
- End of observation period not integer, not positive, greater than length of simulation run, or not greater than begin of observation period.
- Width of observation subintervals not integer, not positive, or not regular.
- Reward expression not int/real typed.
- Variable not defined.
- Measure identifier (trace expression) already occurred with a different measure expression (trace file).
- Redundant measure or trace (this is just a warning).

A simulation can fail for the following reasons:

- Division by zero.
- Modulus by nonpositive integer.
- Invalid parameter of power operation.
- Logarithm of nonpositive number.
- Square root of negative number.
- Invalid parameter of a given probability distribution function.
- Attempt to access an element not present in a given list/array.
- Generation of an absorbing or performance open state.
- Unable to read from trace file.

9.4 Functional Kernel

The functional kernel, which is written in C, contains the routine to generate a file (.fsm) containing the functional semantic model of correct, finite state EMPA_{vp} specifications in the same way as the integrated interleaving semantic model. The functional semantic model can also be produced in the format expected by CWB-NC and saved into a suitable file (.fsm.empa).

The functional kernel is then interfaced with a version of CWB-NC that has been retargeted for EMPA_{vp} using the Process Algebra Compiler of North Carolina (PAC-NC) [54] by encoding as its input the functional semantics of EMPA_{vp} along with a syntactic description of EMPA_{vp}. The functional kernel therefore comprises the analysis capabilities of CWB-NC, such as on-the-fly model checking in the μ-calculus or CTL [50], equivalence checking (strong and weak bisimulation equivalences [133] and may and must testing equivalences [63]), and preorder checking (may and must testing preorders [63]), and provides diagnostic information (in the form of distinguishing formulas/tests) in the case that an equivalence or a preorder relation does not hold between two specifications. Such capabilities can also be directly applied to EMPA_{vp} specifications. This is the reason why EMPA_{vp} specifications and functional semantic models in the format expected by CWB-NC have the same extension (.empa).

To be more precise, it is worth recalling that CWB-NC does not support value passing. Therefore, EMPA_{vp} specifications which do employ value passing related features cannot be directly analyzed by CWB-NC. However, since for such specifications we nonetheless wish to exploit the CWB-NC capabilities, we abstract action parameters, guards, and assignments away from transitions when generating the functional semantic models in the format expected by CWB-NC.

Whenever an analysis routine is invoked, a suitable command file (.fcom) is prepared and passed to CWB-NC which returns the results in another file (.cwb). This file is then manipulated in order to make the verification results more readable (.fver). In case of model checking, a further file (.mu) must be provided which specifies the modal logic formulas of interest. Each formula is of the form

$$\hat{}id = (formula\_expr)$$

where *id* is the formula identifier and *formula_expr* is composed of the following operators:

- Variable x holds for a state if the formula bound to the variable holds for the state.
- Zeroary operator "tt" ("ff") holds for all (no) states.
- Operator "_∧_" for logical conjunction ("_∨_" for logical disjunction) holds for a state if both operands hold (at least one of the operands holds) for the state.
- Operator "not _" for logical negation holds for a state if the operand does not hold for the state.
- Operator "$\langle$ *act_type_list* $\rangle$ _" ("$\langle\langle$ *act_type_list* $\rangle\rangle$ _") for (weak) possibility holds for a state if in that state it is possible to perform an action (possibly preceded and followed by internal actions) whose

type belongs to *act_type_list* evolving into a state which satisfies the operand. The action type list can be empty and can be preceded by operator "-" for set complementation w.r.t. *AType*.

- Operator "[*act_type_list*] _" ("[[*act_type_list*]] _") for (weak) necessity holds for a state if in that state, whenever it is possible to perform an action (possibly preceded and followed by internal actions) whose type belongs to *act_type_list*, then the derivative state satisfies the operand. The action type list can be empty and can be preceded by operator "-" for set complementation w.r.t. *AType*.

- Operator "$\min x =$ _" ("$\max x =$ _") for minimum (maximum) fixed point is used to define recursive formulas: let $\Phi_0 = \mathrm{ff}$ ($\Phi_0 = \mathrm{tt}$) and Φ_{i+1} be Φ where every free occurrence of x is replaced by Φ_i for all $i \in \mathbb{N}_+$, then $\min x = \Phi$ ($\max x = \Phi$) may be interpreted as $\vee_{i \in \mathbb{N}} \Phi_i$ ($\wedge_{i \in \mathbb{N}} \Phi_i$).

- Operator "AG _" ("EG _") for always holds for a state if all of its maximal computations are (at least one of its maximal computation is) such that each involved state satisfies the operand.

- Operator "AF _" ("EF _") for eventually holds for a state if all of its maximal computations are (at least one of its maximal computation is) such that at least one of the involved states satisfies the operand.

- Operator "A(_ U _)" ("E(_ U _)") for until holds for a state if all of its maximal computations are (at least one of its maximal computation is) such that at least one of the involved states satisfies the second operand and every preceding state satisfies the first operand.

- Operator "A(_ W _)" ("E(_ U _)") for weak until holds for a state if all of its maximal computations are (at least one of its maximal computation is) such that at least one of the involved states satisfies the second operand and every preceding state satisfies the first operand, or every involved state satisfies the first operand.

In order to avoid ambiguities, we assume conjunction, disjunction, negation, always, and eventually to be right associative and we introduce the following operator precedence relation: (weak) possibility = (weak) necessity > negation = always = eventually > conjunction > disjunction. Parentheses can be used to alter associativity and precedence. While reading a .mu file, the parser produces a listing file (.lis) where the following specific errors/warnings are reported:

- Identifier of a formula or formula variable already used for other purposes.

- Identifier involved in a possibility or necessity operator not an action type.

- Unbound formula variable.

- Identifier occurring in a formula not a formula or a formula variable.

- Formula identifier already occurred with a different formula expression.

- Redundant formula (this is just a warning).

9.5 Performance Kernel

The performance kernel, which is written in C, contains the routine to generate a file (.psm) containing the performance semantic model of correct, finite state, performance closed EMPA_{vp} specifications in the same way as the integrated interleaving semantic model. More precisely, a HDTMC or a HCTMC is produced depending on whether there are only vanishing states or not: in the case in which both tangible and vanishing states are present, the algorithm described in Sect. 4.5 is applied. The performance semantic model can also be produced in the format expected by MarCA and saved into two suitable files: one containing a representation of the transition matrix or infinitesimal generator (.mc) and one containing the initial state probabilities (.ip).

The performance kernel is then interfaced with a version of MarCA that has been retargeted to interact with TwoTowers. The performance kernel therefore comprises all the analysis methods of MarCA for the computation of stationary/transient state probability distributions and reward based performance measures.

Whenever an analysis routine is invoked, a suitable command file (.pcom) is prepared and passed to MarCA which returns the related state probability distribution in another file (.pro or .prosmc). In case of stationary/transient state probability distribution computation, this file is manipulated in order to make the performance evaluation results more readable (.dis). In case of stationary/transient reward based performance measures, this file is used to calculate the performance measures of interest (.mea). These are specified in an additional file (.rew). Each measure is of the form

$$\hat{}id = (reward\_list)$$

where *id* is the measure identifier and *reward_list* is a list of reward assignments of the form

$$a\ yield\_reward\ bonus\_reward$$

which means that every active action with type a must be given the two rewards above for the measure under specification. This way of specifying rewards is equivalent to attach vectors of reward pairs to actions, where vectors have as many pairs as there are specified measures. This means that in TwoTowers a generalization of $\text{EMPA}_{\text{vp,r}}$ is actually implemented which avoids burdening algebraic specifications with sparse information necessary only in case of performance evaluation and permits evaluating several performance measures at a time by constructing the underlying performance model only once. While reading a .rew file, the parser produces a listing file (.lis) where the following specific errors/warnings are reported:

- Identifier of a measure already used for other purposes.
- Identifier involved in a reward assignment not an observable action type.
- Measure identifier (action type) already occurred with a different reward assignment.
- Redundant measure or reward assignment to an action type (this is just a warning).

The obtainment of performance results fails if the specification is not performance closed, the invoked analysis method has to deal with too many states, or the invoked analysis method does not achieve the desired accuracy. In the last two cases, the user is suggested to change the value of the parameters of the method or to change the method itself.

9.6 Net Kernel

The net kernel, which is written in C, contains the routine to generate the optimized integrated label oriented net semantics of correct, finite state, performance closed EMPA_{vp} specifications according to the axiom in Table 6.2. While generating an integrated net semantic model, a file (.insm) is written which shows each place, transition, and arc. At the end of the file it is reported the number of places, the number of transitions (classified as exponentially timed or immediate), and the number of arcs (classified as input or output). Inhibitor arcs cannot be generated because alternative compositions are admitted only if guarded. The integrated net semantic model can also be produced in the format expected by GreatSPN and saved into two suitable files: one containing the structure of the net (.net) and one containing information on marking dependent rates (.def).

To generate the optimized integrated label oriented net semantics of a specification, first of all we compute via dec_{lab} the decomposition of its initial constant and we repeat this procedure for each term associated with a place in *GACom* after removing its outermost action prefixes. As an example, given $\sum_{h=1}^{n} <a_h, \tilde{\lambda}_h>.E_h$ we compute $dec_{\text{lab}}(E_h)$ for all $h = 1, \ldots, n$. Such places are stored in a hash table for efficient retrieval. The bucket of a term points to the buckets of the places resulting from its decomposition; in this way, it is straightforward to see whether the decomposition of a given term has already been computed or not. Similarly, the bucket of a place points to the bucket of the related term; this allows for an efficient reinterpretation at the algebraic level of analysis results computed at the net level. Once all the places in *GACom* have been generated, the transitions can be produced by examining such places and those in *Ren* to build transition presets according to condition 3 of Sect. 6.2.2. More precisely, in the first round we consider those places whose action types are not renamed because of synchronizations. Such places cannot synchronize with others and hence give rise to transitions with a singleton preset. In the next round we consider the remaining places two by two and we try to combine them by means of the places in *Ren* generated so far. Three cases arise:

- If two places combine and the resulting action type is new, an additional dummy place is generated which will be considered in the subsequent rounds instead of the two originating places. The dummy place contains the information about the two places that have originated it.

- If two places combine and the resulting action type is not new, a new synchronization transition is generated and the two places will no longer be considered in the subsequent rounds. The preset of

the newly generated transition is composed of the renamings used so far and the two combined places. When one or both of them are dummy places, their expansion is required.

- If two places do not combine, nothing happens.

The number of rounds is bounded by the number of places in *Ren* that have been generated. Finally, the redundant inhibitor and contextual arcs are removed together with the related upstream places.

The net kernel is then interfaced with GreatSPN. The net kernel therefore comprises all the functional and performance analysis capabilities of GreatSPN including the computation of P-invariants and T-Invariants [164], the average and maximum numbers of tokens in net places and the maximum throughput of net transitions [45], and reward based performance measures. Whenever an analysis routine is invoked, a suitable command file (.ncom) is prepared and passed to GreatSPN which returns the results in another file (.gspn). This file is then manipulated in order to make P-invariants (.pinv), T-invariants (.tinv), the average and maximum numbers of tokens in net places (.tok), the maximum throughput of net transitions (.thr), and reward based performance measures (.mea) more readable.

The obtainment of results fails if the specification is not performance closed.

It is worth recalling that, since the net semantics is defined for EMPA, EMPA_{vp} specifications which do employ value passing related features cannot be adequately mapped onto optimized integrated label oriented net semantic models. However, since for such specifications we nonetheless wish to exploit the GreatSPN analysis capabilities, we abstract action parameters, guards, and assignments away from transitions when generating the net semantic models in the format expected by GreatSPN.

Chapter 10

Case Studies

In this chapter several communication protocols and distributed algorithms will be modeled with EMPA and then analyzed with TwoTowers. Only the first case study will involve the application of the whole integrated approach of Fig. 1.1, while the others will be mainly concerned with performance evaluation via numerical or simulative analysis.

This chapter is organized as follows. In Sect 10.1 we consider the alternating bit protocol. From the algebraic model, the whole collection of its underlying semantic models (integrated interleaving, functional, performance, and integrated net semantics) is derived. The protocol is shown to be deadlock free and its behavior is proved to be equivalent to a buffer with capacity one. The throughput of the protocol is assessed. A value passing description of the protocol is also provided to illustrate the advantage gained by exploiting the inherent parametricity of value passing terms when analyzing symmetric systems. In Sect. 10.2 we analyze CSMA/CD be measuring its channel utilization and collision probability. In Sect. 10.3 we examine token ring. The protocol is shown to be deadlock free via equivalence checking. The channel utilization achieved by the protocol is evaluated. The fair circulation of the token is also proved by resorting to both functional and performance arguments. In Sect. 10.4 we study two ATM switches supporting different kinds of service by comparing the corresponding cell loss probabilities. Discrete time terms are developed to model the switches and Bernoulli distributions are described to represent the incoming traffic. In Sect. 10.5 we describe a novel adaptive mechanism for transmitting packetized audio over the Internet. A value passing model is provided in order to be able to represent delays following a Gaussian distribution. The percentage of packets that are played out is evaluated via simulation. In Sect. 10.6 we analyze the Lehmann-Rabin randomized distributed algorithm for dining philosophers. The algorithm is proved to be deadlock free and to guarantee the mutual exclusive use of resources via model checking. The mean number of philosophers who are simultaneously eating is also assessed. Finally, in Sect. 10.7 we model several mutual exclusion algorithms. They are compared by measuring the corresponding mean numbers of accesses per time unit to the critical section and the shared control variables.

The purpose of such case studies is to demonstrate the adequacy of the integrated approach, the expres-

siveness of EMPA, and the importance of developing automated tools such as TwoTowers to help designers in modeling and analyzing complex systems in a formal way. More generally, these case studies together with other case studies appeared in the literature (such as e.g. electronic mail system [86], multiprocessor mainframe [87], industrial production cell [107], robot control [70], mobile telephony network [161], multimedia stream [36], plain old telephone system [88]) show the usability of the stochastically timed process algebra approach to the formal description and assessment of functional and performance properties of concurrent systems.

10.1 Alternating Bit Protocol

In this section we apply the integrated approach of Fig. 1.1 to the alternating bit protocol [26, 27, 12, 17]. The protocol is modeled by means of an EMPA term and then analyzed by studying the semantic models associated with the term in order to show its correctness and to assess its throughput. We also provide a value passing description of the protocol which profits from the symmetry of the protocol itself. The reason why we have chosen the alternating bit protocol as a case study to illustrate the integrated approach is that such a protocol has become a standard example in the literature, so it can be used to compare the EMPA model with other models.

10.1.1 Informal Specification

The *alternating bit protocol* [9] is a data link level communication protocol that establishes a means whereby two stations, one acting as a sender and the other acting as a receiver, connected by a full duplex communication channel that may lose messages, can cope with message loss. The name of the protocol stems from the fact that each message is augmented with an additional bit. Since consecutive messages that are not lost are tagged with additional bits that are pairwise complementary, it is easy to distinguish between an original message and its possible duplicates. Initially, if the sender obtains a message from the upper level, it augments the message with an additional bit set to 0, sends the tagged message to the receiver, and starts a timer. If an acknowledgement tagged with 0 is received before the timeout expires, then the subsequent message obtained from the upper level will be sent with an additional bit set to 1, otherwise the current tagged message is sent again. On the other side, the receiver waits for a message tagged with 0. If it receives such a tagged message for the first time, then it passes the message to the upper level, sends an acknowledgement tagged with 0 back to the sender, and waits for a message tagged with 1, whereas if it receives a duplicate tagged message (due to message loss, acknowledgement loss, or propagation taking an arbitrarily long time), then it sends an acknowledgement tagged with the same additional bit back to the sender.

10.1.2 Formal Description with EMPA

Since it is helpful to take advantage from compositionality, we figure out how to deal with three interacting entities: $Sender, Receiver, and Channel$. The interaction between *Sender* and *Channel* is described by action types tm_i, $i \in \{0,1\}$, standing for "transmit message tagged with i" and da_i, $i \in \{0,1\}$, standing for "deliver acknowledgement tagged with i". The interaction between *Receiver* and *Channel* is described by action types dm_i, $i \in \{0,1\}$, standing for "deliver message tagged with i", and ta_i, $i \in \{0,1\}$, standing for "transmit acknowledgement tagged with i". The scenario can be modeled as follows:

$$\begin{aligned} ABP &\triangleq Sender_0 \parallel_S Channel \parallel_R Receiver_0 \\ S &= \{tm_0, tm_1, da_0, da_1\} \\ R &= \{dm_0, dm_1, ta_0, ta_1\} \end{aligned}$$

Thanks to compositionality, we can now focus our attention on the single entities separately. *Channel* is composed of two independent half duplex lines: $Line_m$ (for message transmission) and $Line_a$ (for acknowledgement transmission). The local activities of *Channel* are described by action types pm_i, $i \in \{0,1\}$, standing for "propagate message tagged with i" and pa_i, $i \in \{0,1\}$, standing for "propagate acknowledgement tagged with i". Additionally, there are two other activities local to *Channel* that are described by action type τ and represent the fact that a message or an acknowledgement is lost or not. As far as the timing of the actions in which *Channel* is involved is concerned, we assume that the length of a message and the length of an acknowledgement are exponentially distributed, so that message/acknowledgement transmission, propagation, and delivery times are exponentially distributed. However, the three phases given by transmission, propagation, and delivery are temporally overlapped, i.e. they constitute a pipeline. As a consequence, in order to correctly determine the time taken by a message/acknowledgement to reach *Receiver*/*Sender*, we model actions related to transmission and delivery as immediate and we associate the actual timing (i.e., the duration of the slowest stage of the pipeline) with actions related to propagation. We thus assume that the message propagation time is exponentially distributed with rate δ, the acknowledgement propagation time is exponentially distributed with rate γ, and the loss probability is $p \in \mathbb{R}_{]0,1[}$. *Channel* can be modeled as follows:

$$\begin{aligned} Channel &\triangleq Line_m \parallel_\emptyset Line_a \\ Line_m &\triangleq <tm_0, *>.<pm_0, \delta>.(<\tau, \infty_{1,1-p}>.<dm_0, \infty_{1,1}>.Line_m + \\ &\qquad <\tau, \infty_{1,p}>.Line_m) + \\ &\quad <tm_1, *>.<pm_1, \delta>.(<\tau, \infty_{1,1-p}>.<dm_1, \infty_{1,1}>.Line_m + \\ &\qquad <\tau, \infty_{1,p}>.Line_m) \\ Line_a &\triangleq <ta_0, *>.<pa_0, \gamma>.(<\tau, \infty_{1,1-p}>.<da_0, \infty_{1,1}>.Line_a + \\ &\qquad <\tau, \infty_{1,p}>.Line_a) + \\ &\quad <ta_1, *>.<pa_1, \gamma>.(<\tau, \infty_{1,1-p}>.<da_1, \infty_{1,1}>.Line_a + \\ &\qquad <\tau, \infty_{1,p}>.Line_a) \end{aligned}$$

Observe that the probabilistic choice between the reception and the loss of a message/acknowledgement has been easily represented by means of the weights associated with the two immediate actions $<\tau, \infty_{1,1-p}>$ and $<\tau, \infty_{1,p}>$.

The local activities of *Sender* are described by action types *gm* standing for "generate message" and *to* standing for "timeout". We assume that the message generation time is exponentially distributed with rate λ and that the timeout period is exponentially distributed with rate θ. Of course, this is not realistic, but EMPA does not enable us to express deterministic durations, and a Markovian analysis would not be possible otherwise. A good solution would consist of approximating the deterministic duration of the timeout period by means of a sequence of exponentially timed actions with the same rate (thereby implementing an Erlang distribution, which is a special case of the hypoexponential distribution depicted in Fig. 4.3) as done in [90], but the underlying semantic model would be much bigger than the one in Fig. 10.1. *Sender* can be modeled as follows:

$$
\begin{aligned}
Sender_0 &\triangleq <gm, \lambda>.<tm_0, \infty_{1,1}>.Sender'_0 \\
Sender'_0 &\triangleq <da_0, *>.Sender_1 + \\
&\quad\ <da_1, *>.Sender'_0 + \\
&\quad\ <to, \theta>.Sender''_0 \\
Sender''_0 &\triangleq <tm_0, \infty_{1,1}>.Sender'_0 + \\
&\quad\ <da_0, *>.Sender_1 + \\
&\quad\ <da_1, *>.Sender''_0, \\
Sender_1 &\triangleq <gm, \lambda>.<tm_1, \infty_{1,1}>.Sender'_1 \\
Sender'_1 &\triangleq <da_1, *>.Sender_0 + \\
&\quad\ <da_0, *>.Sender'_1 + \\
&\quad\ <to, \theta>.Sender''_1 \\
Sender''_1 &\triangleq <tm_1, \infty_{1,1}>.Sender'_1 + \\
&\quad\ <da_1, *>.Sender_0 + \\
&\quad\ <da_0, *>.Sender''_1
\end{aligned}
$$

An important observation (similar to the one reported in [55] for a CCS model of the same protocol) concerns terms $Sender''_0$ and $Sender''_1$. Since they model the situation after a timeout expiration, they should comprise the retransmission action only in order to be consistent with the definition of the protocol. The problem is that a deadlock may occur whenever, after a sequence of premature timeouts (i.e. timeouts expired although nothing is lost), the sender is waiting to be able to retransmit the message, the receiver is waiting to be able to retransmit the corresponding acknowledgement, the message line is waiting to be able to deliver a previous copy of the message, and the acknowledgement line is waiting to be able to deliver a previous copy of the acknowledgement. To destroy deadlock, $Sender''_0$ and $Sender''_1$ are allowed to receive possible acknowledgements, thereby avoiding unnecessary retransmissions.

The only local activity of *Receiver* is described by action type *cm* standing for "consume message" which

is taken to be immediate in that it is irrelevant from the performance viewpoint. *Receiver* can be modeled as follows:

$$\begin{aligned} Receiver_0 &\triangleq <dm_0,*>.<cm,\infty_{1,1}>.<ta_0,\infty_{1,1}>.Receiver_1 + \\ &\quad <dm_1,*>.<ta_1,\infty_{1,1}>.Receiver_0 \\ Receiver_1 &\triangleq <dm_1,*>.<cm,\infty_{1,1}>.<ta_1,\infty_{1,1}>.Receiver_0 + \\ &\quad <dm_0,*>.<ta_0,\infty_{1,1}>.Receiver_1 \end{aligned}$$

10.1.3 Comparison with Other Formal Descriptions

At the beginning of this section we said that we have chosen the alternating bit protocol in order to compare its EMPA model with others expressed with different formalisms. For example, it turns out that in [133, 55] performance aspects are completely neglected because a classical process algebra is used, while in [136] a stochastically timed Petri net model is adopted but the unrealistic assumption that the timeout expires only if a loss actually occurs is made. In [144, 4, 90] stochastically timed process algebraic descriptions are given, but they do not accurately take into account the division into three temporally overlapped phases. In [90], however, the deterministic duration of the timeout period has been better approximated by means of an Erlang distribution.

10.1.4 Functional Analysis

The integrated interleaving semantics of *ABP* is presented in Fig. 10.1. The LTS $\mathcal{I}[\![ABP]\!]$ has 302 states (76 tangible, 226 vanishing) and 464 transitions (284 observable, 180 invisible; 140 exponentially timed, 324 immediate). Due to the symmetry of the protocol, only half of the state space has been drawn (dashed transitions depict the link with the remaining states). Whenever neither losses nor premature timeouts occur, the states visited by the protocol are 1, 3, 5, 9, 15, 21, 25, 45, 51, 55 and the corresponding symmetric ones, i.e. 2, 4, 6, 10, 16, 22, 26, 46, 52, 56. Following the proposed approach, we can use the LTS $\mathcal{F}[\![ABP]\!]$ (obtained from $\mathcal{I}[\![ABP]\!]$ by dropping action rates) to detect some functional properties. For example, since there are no absorbing states, the protocol is deadlock free. If $Sender_0''$ and $Sender_1''$ had not been carefully designed (as explained in Sect. 10.1.2), then we would have obtained eight deadlock states: 113, 197, 213, 301 and the corresponding symmetric ones. Moreover, by resorting to equivalence checking we have proved that *ABP* behaves as a buffer with capacity one that can engage in a sequence of alternating actions with type *gm* and *cm*. This has been accomplished by showing that $\mathcal{F}[\![ABP/L]\!]$ is weakly bisimilar to $\mathcal{F}[\![ABPSpec]\!]$ where $L = AType - \{\tau, gm, cm\}$ and $ABPSpec \triangleq <gm,\lambda>.<cm,\infty_{1,1}>.ABPSpec$. Finally, we have proved via model checking that two consecutive actions of type *gm* (*cm*) can never be executed. A .mu file with the following CTL formula has been used: $AG([gm]\,A([gm]\mathrm{ff}\,W\,\langle cm\rangle\mathrm{tt}) \wedge [cm]\,A([cm]\mathrm{ff}\,W\,\langle gm\rangle\mathrm{tt}))$.

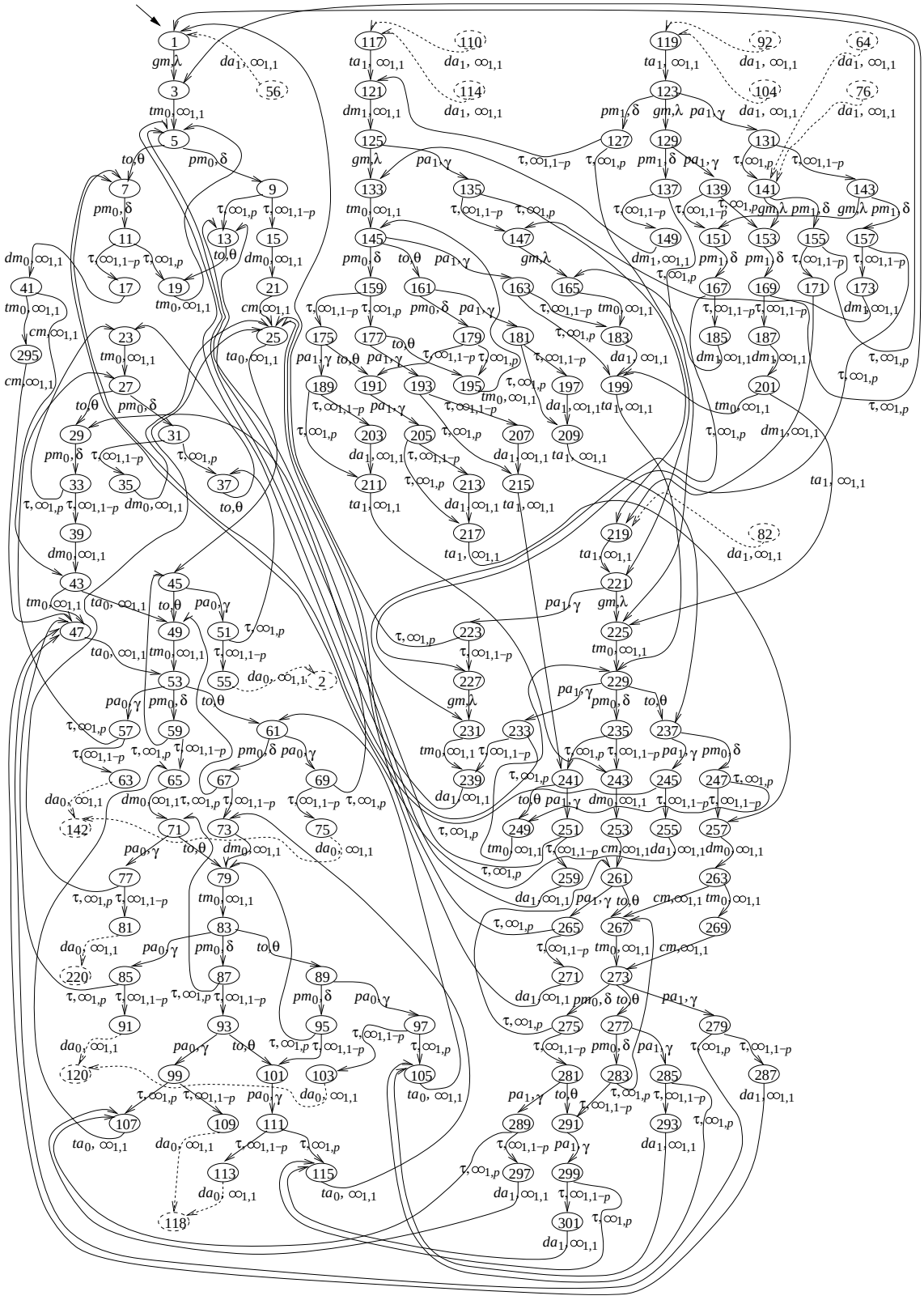


Figure 10.1: $\mathcal{I}[\![ABP]\!]$

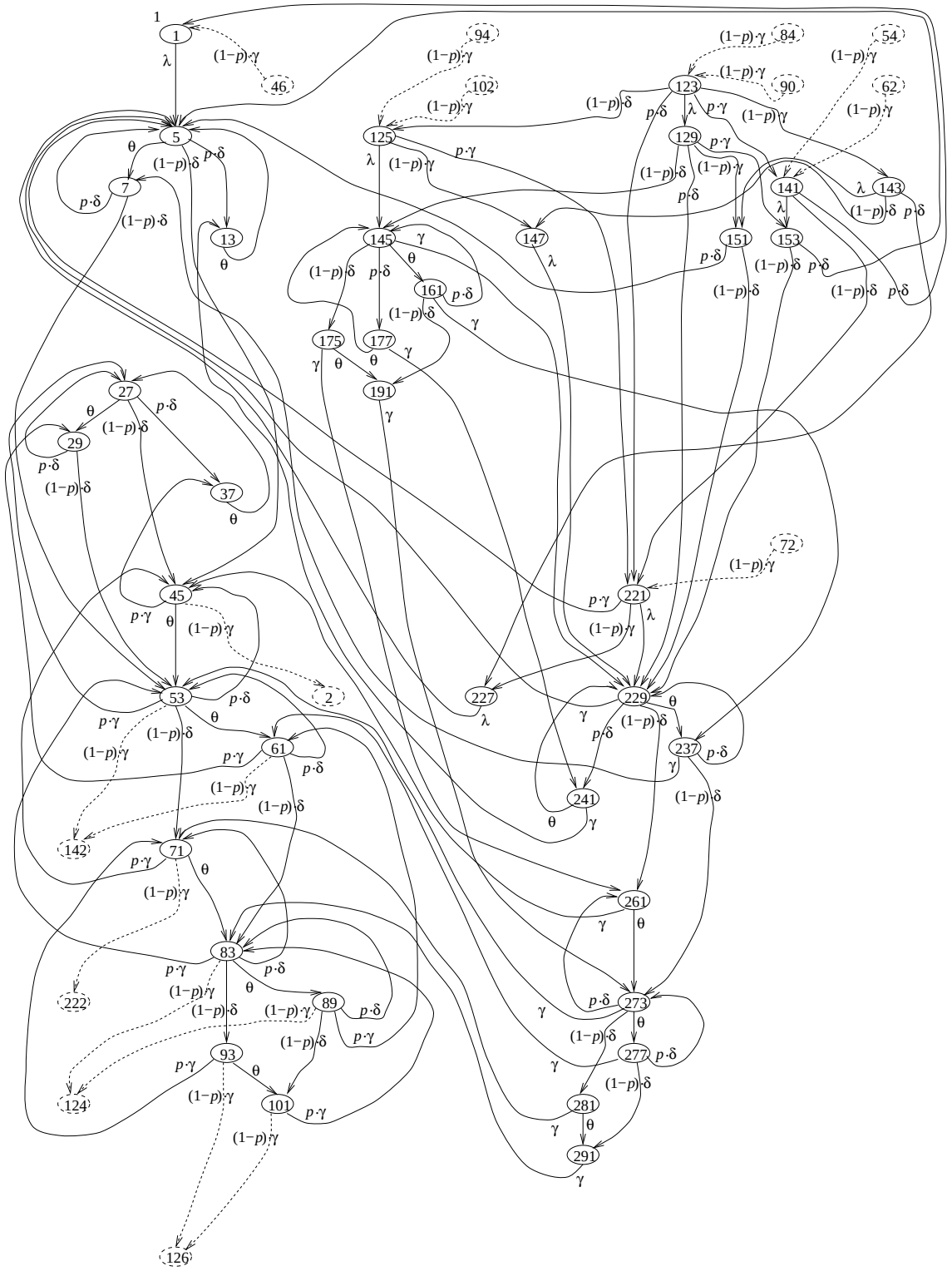


Figure 10.2: $\mathcal{M}[\![ABP]\!]$

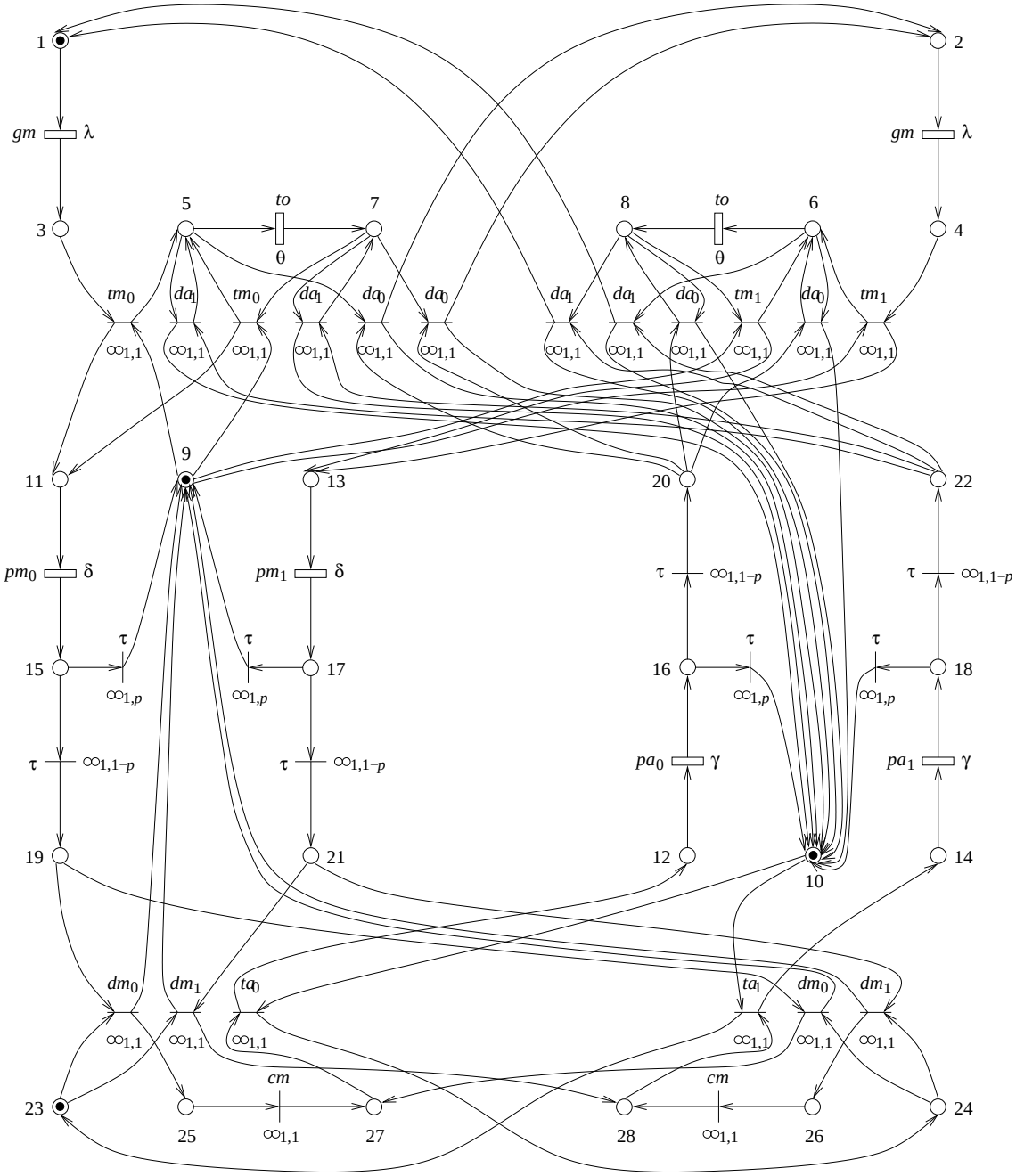


Figure 10.3: $\mathcal{N}_{\mathrm{lab,o}}[\![ABP]\!]$

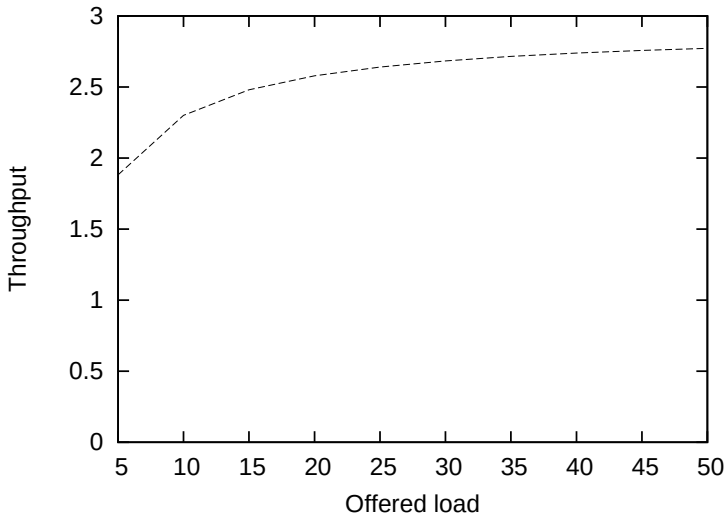


Figure 10.4: Throughput of ABP

10.1.5 Performance Analysis

The Markovian semantics of ABP is presented in Fig. 10.2. The p-LTS $\mathcal{M}[\![ABP]\!]$, which has 76 states and 204 transitions, has been obtained from $\mathcal{I}[\![ABP]\!]$ by applying the algorithm in Sect. 4.5. Since $\mathcal{M}[\![ABP]\!]$ is finite and strongly connected, it represents a HCTMC for which the stationary probability distribution function exists. Following the proposed approach, we can exploit such a HCTMC for assessing some performance indices. For example, the throughput of the protocol is given by the number λ of messages per second that arrive at the sender multiplied by the probability that the sender can accept a new message to send: this probability is given by the sum of the stationary probabilities of the states having an outgoing transition labeled with λ. The following .rew file has been provided:

$$\hat{}throughput = (gm\ \lambda\ 0)$$

In Fig. 10.4 we report the value of the actual throughput for different values of the offered load λ. We assume that the protocol uses two 9.6 Kbps lines and that the (mean) length of the packets is 1024 bits, so that the propagation rate is $\delta = \gamma = 9.375$ packets per second: we finally assume that the timeout period is 1 second ($\theta = 1$) and that the loss probability is $p = 0.05$.

10.1.6 The Equivalent Net Description

The optimized integrated label oriented net semantics of ABP is presented in Fig. 10.3. The GSPN $\mathcal{N}_{\mathrm{lab,o}}[\![ABP]\!]$ comprises 28 places and 36 transitions. Since the integrated net semantics for EMPA meets the retrievability principles, ABP and $\mathcal{N}_{\mathrm{lab,o}}[\![ABP]\!]$ model exactly the same protocol in two different ways: the algebraic description is compositional and more readable, the net description is more concrete and provides

a means to detect dependencies, conflicts, and synchronizations among activities which cannot be discovered in an interleaving model like $\mathcal{I}[\![ABP]\!]$. Also notice that $\mathcal{N}_{\mathrm{lab,o}}[\![ABP]\!]$ is considerably more compact than $\mathcal{I}[\![ABP]\!]$: this fact may turn out to be helpful in order to carry out a more efficient assessment of system characteristics.

As an example, some functional properties can be evaluated without generating the underlying state space. This is accomplished by computing invariants and by reinterpreting them at the algebraic level thanks to the correspondence between net places and sequential terms and between net transitions and actions. For instance, we provide below the P-invariants for $\mathcal{N}_{\mathrm{lab,o}}[\![ABP]\!]$:

$$\begin{aligned} M(V_1) + M(V_2) + M(V_3) + M(V_4) + M(V_5) + M(V_6) + M(V_7) + M(V_8) &= 1 \\ M(V_9) + M(V_{11}) + M(V_{13}) + M(V_{15}) + M(V_{17}) + M(V_{19}) + M(V_{21}) &= 1 \\ M(V_{10}) + M(V_{12}) + M(V_{14}) + M(V_{16}) + M(V_{18}) + M(V_{20}) + M(V_{22}) &= 1 \\ M(V_{23}) + M(V_{24}) + M(V_{25}) + M(V_{26}) + M(V_{27}) + M(V_{28}) &= 1 \end{aligned}$$

where M denotes an arbitrary marking while the constant occurring in the right hand side of each equality is determined by the initial marking. Note that the first P-invariant is concerned with the sender, the second one with the line for messages, the third one with the line for acknowledgements, and the fourth one with the receiver. Since each P-invariant identifies a subset of the places of the net whose token count is invariant under transition firing, from the equalities above we know that in our model at most one message/acknowledgement at a time is propagating along the message/acknowledgement line.

As another example, some performance properties can be similarly evaluated avoiding the construction of the whole state space. More precisely, we can efficiently approximate the throughput of the protocol (whose exact value is reported in Fig. 10.4) by computing upper bounds on the sum of the throughputs of the two net transitions labeled with gm, λ. As an example, in this way we can derive in polynomial time that λ is an upper bound on the throughput of the protocol.

10.1.7 Value Passing Description

In this section we illustrate the advantage that can be gained by exploiting the inherent parametricity of value passing terms when analyzing symmetric systems. Although it is not necessary to resort to value passing to model the alternating bit protocol, we shall see now that it is extremely advantageous to do so.

From the syntactical standpoint, the main difference is that message/acknowledgement tags become the parameters of actions and constants. The protocol can be modeled as follows:

$$\begin{aligned} ABP_{vp}(\mathrm{bool}\, b := \mathrm{ff};) &\triangleq Sender(b) \parallel_S Channel \parallel_R Receiver(b) \\ S &= \{tm, da\} \\ R &= \{dm, ta\} \end{aligned}$$

where:

$$
\begin{aligned}
Channel &\triangleq Line_m \parallel_\emptyset Line_a \\
Line_m(;\mathrm{bool}\, x) &\triangleq <tm?(x), *>.<pm, \delta>.(<\tau, \infty_{1,1-p}>.<dm!(x), \infty_{1,1}>.Line_m + \\
&\qquad <\tau, \infty_{1,p}>.Line_m) \\
Line_a(;\mathrm{bool}\, x) &\triangleq <ta?(x), *>.<pa, \gamma>.(<\tau, \infty_{1,1-p}>.<da!(x), \infty_{1,1}>.Line_a + \\
&\qquad <\tau, \infty_{1,p}>.Line_a)
\end{aligned}
$$

and:

$$
\begin{aligned}
Sender(\mathrm{bool}\, b;) &\triangleq <gm, \lambda>.<tm!(b), \infty_{1,1}>.Sender'(b) \\
Sender'(\mathrm{bool}\, b; \mathrm{bool}\, x) &\triangleq <da?(x), *>.Sender''(b, x) + \\
&\quad <to, \theta>.Sender'''(b) \\
Sender''(\mathrm{bool}\, b,\ \mathrm{bool}\, x;) &\triangleq \text{if } (x == b, 1 - p_{pt}) \text{ then} \\
&\quad\ Sender(!!b) \\
&\quad \text{else} \\
&\quad\ Sender'(b) \\
Sender'''(\mathrm{bool}\, b; \mathrm{bool}\, x) &\triangleq <tm!(b), \infty_{1,1}>.Sender'(b) + \\
&\quad <da?(x), *>.Sender''(b, x)
\end{aligned}
$$

with p_{pt} expressing (a guess about) the probability of a premature timeout, i.e. a timeout caused by slow propagation rather than actual loss, and:

$$
\begin{aligned}
Receiver(\mathrm{bool}\, b; \mathrm{bool}\, x) \triangleq\ & <dm?(x), *>.\text{if } (x == b, 1 - p_{pt}) \text{ then} \\
& \quad <cm, \infty_{1,1}>.<ta!(x), \infty_{1,1}>.Receiver(!!b) \\
& \quad \text{else} \\
& \quad <ta!(x), \infty_{1,1}>.Receiver(b)
\end{aligned}
$$

This description of the protocol is more compact than the previous one and, most importantly, the underlying state space is smaller. Now we have only 148 states (29 tangible, 119 vanishing) and 232 transitions (130 observable, 102 invisible; 52 exponentially timed, 180 immediate). This is due to the fact that by means of our treatment of constant parameters based on lookahead we are able to exploit all the parametricity inherent to value passing terms so as to lump together at semantic model construction time the ways in which the protocol works according to the two possible values of the alternating bit which labels messages and acknowledgements.

Finally, we plot in Fig. 10.5 the throughput of the protocol for different values of p_{pt} under the same assumptions made in Sect. 10.1.5. It is worth noting that by varying the probability associated with a conditional operator it is possible to carry out a sensitivity analysis of performance.

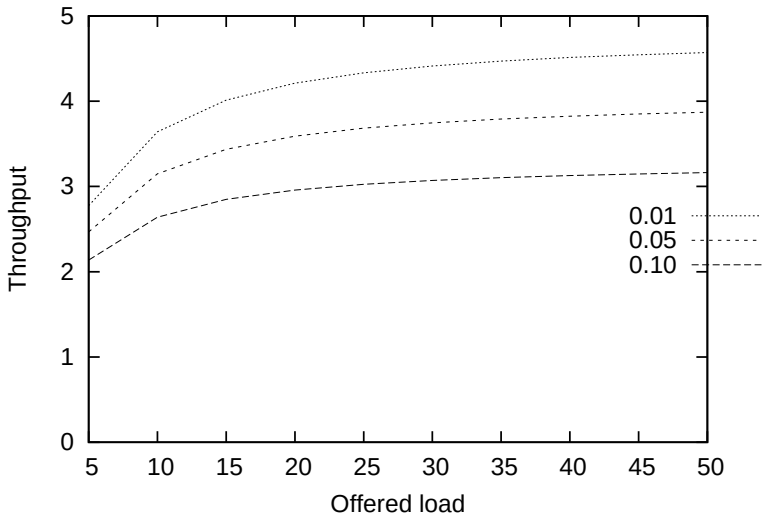


Figure 10.5: Throughput of ABP_{vp}

10.2 CSMA/CD

In this section we formally describe CSMA/CD, the IEEE 802.3 standard medium access control protocol for local area networks, in order to derive the related channel utilization and collision probability [13, 16].

10.2.1 Informal Specification

One of the most commonly used medium access control protocols for local area networks with bus/tree topology is *CSMA/CD* [182]: as an example, its original baseband version is seen in Ethernet. Given a set of n stations connected through a local area network, the nonpersistent version of CSMA/CD works as follows. When a station wants to transmit a message, it first listens to the channel in order to determine whether another transmission is in progress ("listen before talk"). If the channel is sensed to be idle then the station transmits its message, otherwise the station backs off a random amount of time and then senses the channel again. After starting transmission, the station continues to listen to the channel until it has finished ("listen while talk"). If a collision is detected, the station ceases transmitting its message, transmits a short signal in order to let all stations know there has been a collision, and backs off a random amount of time before attempting to transmit again. In case of collision, messages are lost.

10.2.2 Formal Description with EMPA

The entities involved in the protocol are n stations and the channel:

$$\begin{aligned} CSMA/CD_n &\triangleq (Station_1 \parallel_M \ldots \parallel_M Station_n) \parallel_C Channel \\ M &= \{sense\_idle_i, sense\_busy_i, start\_trans\_msg_i, trans\_msg_i \mid 1 \leq i \leq n\} \cup C \\ C &= \{signal\_coll\} \end{aligned}$$

where $sense\_idle_i$ ($sense\_busy_i$) is the action type describing the fact that *Channel* is sensed idle (busy) by $Station_i$, $start\_trans\_msg_i$ ($trans\_msg_i$) is the action type describing the beginning of transmission (transmission) of a message sent by $Station_i$, and $signal\_coll$ is the action type describing the propagation of a message indicating that there has been a collision.

By virtue of compositionality, we can now focus our attention on each of the involved entities separately. Let us consider $Station_i$, $1 \leq i \leq n$. We assume that upper levels generate at rate λ_i a stream of messages (gen_msg_i) that $Station_i$ has to send. After receiving a message to be sent, $Station_i$ listens to the channel. If the channel is sensed idle ($sense\_idle_i$) then the message starts to be transmitted ($start\_trans\_msg_i$), otherwise ($sense\_busy_i$) $Station_i$ backs off a random amount of time ($back\_off_i$) and then listens to the channel again. During transmission ($trans\_msg_i$) the message can collide with other messages ($signal\_coll$). $Station_i$ can be modeled as follows:

$$\begin{aligned} Station_i &\triangleq <gen\_msg_i, \lambda_i>.StationSense_i + \\ &\quad <signal\_coll, *, >.Station_i \\ StationSense_i &\triangleq <sense\_idle_i, \infty_{1,1}, >.<start\_trans\_msg_i, \infty_{1,1}, >.StationProp_i + \\ &\quad <sense\_busy_i, \infty_{1,1}, >.StationBackoff_i + \\ &\quad <signal\_coll, *>.StationSense_i \\ StationProp_i &\triangleq <trans\_msg_i, *>.Station_i + \\ &\quad <signal\_coll, *>.StationBackoff_i \\ StationBackoff_i &\triangleq <back\_off_i, \gamma>.StationSense_i + \\ &\quad <signal\_coll, *>.StationBackoff_i \end{aligned}$$

Note that $sense\_idle_i$ and $sense\_busy_i$ have been modeled as immediate actions, because they are irrelevant from the performance point of view, as well as $start\_trans\_msg_i$, since it only describes the beginning of the transmission while the duration of the whole operation will be attached to $trans\_msg_i$. Furthermore, the random amount of time during which $Station_i$ backs off has been described by means of an exponentially distributed random variable with rate γ. Finally, we observe that action $<signal\_coll, *>$ is enabled in any state because collisions are signaled also to stations that are not transmitting.

Let us consider now *Channel*. It can be viewed as being composed of a bidirectional error free communication line and n sensors, one for each station, reporting the status of the communication line. As a consequence, *Channel* can be modeled as follows:

$$\begin{aligned} Channel &\triangleq (Sensor_1 \parallel_\emptyset \ldots \parallel_\emptyset Sensor_n) \parallel_T Line \\ T &= \{set\_busy_i, set\_idle_i \mid 1 \leq i \leq n\} \end{aligned}$$

where set_busy_i (set_idle_i) is the action type describing the fact that $Sensor_i$ must be set in such a way that

$Station_i$ senses *Channel* to be busy (idle). The rationale behind the introduction of $Sensor_i$, $1 \leq i \leq n$, is that they should simulate the presence or the absence of a message propagating along the medium. Every component $Sensor_i$, $1 \leq i \leq n$, can be described as follows:

$$\begin{aligned} Sensor_i &\triangleq <sense\_idle_i, *>.Sensor_i + \\ &\quad <set\_busy_i, *>.SensorBusy_i \\ SensorBusy_i &\triangleq <sense\_busy_i, *>.SensorBusy_i + \\ &\quad <set\_idle_i, *>.Sensor_i \end{aligned}$$

Line initially waits for the beginning of the transmission of a message ($start\_trans\_msg_i$). If we denote by $1/\varsigma$ the propagation delay between the two farthest stations in the network, then a collision can occur only within $1/\varsigma$ time units from the beginning of the transmission (after $1/\varsigma$ time units all the stations will sense the channel busy). Thus, if no collisions occur within $1/\varsigma$ time units ($elapse\_prop\_delay$) then the message is successfully transmitted ($trans\_msg_i$), otherwise ($start\_trans\_msg_j$ for $j \neq i$) a collision is detected ($signal\_coll$). The length of messages sent by $Station_i$ is assumed to be exponentially distributed so that their transmission time will be exponentially distributed with mean $1/\mu$, where $1/\mu \geq 2/\varsigma$ according to the IEEE 802.3 standard. *Line* can then be represented as follows:

$$\begin{aligned} Line &\triangleq \sum_{i=1}^{n} <start\_trans\_msg_i, *>.LineTrans_i \\ LineTrans_i &\triangleq <elapse\_prop\_delay, \varsigma>.LineSuccess_i + \\ &\quad \sum_{j=1, j\neq i}^{n} <start\_trans\_msg_j, *>.LineCollision \\ LineSuccess_i &\triangleq <set\_busy_1, \infty_{2,1}>.\ldots.<set\_busy_n, \infty_{2,1}>.<trans\_msg_i, \mu>. \\ &\quad <set\_idle_1, \infty_{2,1}>.\ldots.<set\_idle_n, \infty_{2,1}>.Line \\ LineCollision &\triangleq <set\_busy_1, \infty_{2,1}>.\ldots.<set\_busy_n, \infty_{2,1}>.<elapse\_prop\_delay, \varsigma>. \\ &\quad <signal\_coll, \varsigma>.<set\_idle_1, \infty_{2,1}>.\ldots.<set\_idle_n, \infty_{2,1}>.Line \end{aligned}$$

Observe that the maximum propagation delay $1/\varsigma$ of a signal has been modeled by means of an exponentially distributed random variable with rate ς though the value of such a delay is fixed. This approximation can be made as accurate as we desire by means of a sequence of exponentially timed actions with the appropriate rates: the price to pay, as we can expect, is a state space growth. In case of success, the message is completely transmitted: due to the memoryless property of exponential distributions, the time to completion of the operation above (described by $<trans\_msg_i, \mu>$ in $LineSuccess_i$) is not affected by the fact that the maximum propagation delay has already elapsed. In case of collision, the worst case happens when one of the two farthest stations in the network is transmitting and the other one starts transmitting just before the maximum propagation delay $1/\varsigma$ has elapsed: the former station realizes that a collision has occurred only at around $2/\varsigma$ time units after it has started transmitting. This is the reason for action $<elapse\_prop\_delay, \varsigma>$ in *LineCollision* (the remark about the memoryless property of exponential distributions applies to this case too) as well as the requirement $1/\mu \geq 2/\varsigma$. Finally it is worth noting the use of priorities for actions

set_busy_i and set_idle_i, $1 \leq i \leq n$: since their priority level is 2, they cannot be preempted by any other action, thereby making $Sensor_i$, $1 \leq i \leq n$, work as expected. Actually, the introduction of n components $Sensor_i$ might seem cumbersome. However, they constitute the means whereby to obtain a more accurate description of the protocol. As it turns out, $Sensor_i$ should be set immediately after the passage of the leading part of the message ($1/\varsigma$ is only an upper bound): to do this, we should know the topology of the network as well as the maximum propagation delays between any pair of consecutive stations.

10.2.3 Performance Analysis

For CSMA/CD we have chosen to calculate the utilization of the medium, since it is a critical index which depends on the duration of the backoff period and the maximum propagation delay, as well as the collision probability. Both measures have been determined by varying the number of stations and the frame size. Let us assume that the channel capacity is 10 Mbps and that the average traffic generated by each of the stations equals the channel capacity, hence $\lambda_i = 10$ Mbps for all $1 \leq i \leq n$. According to the IEEE 802.3 standard the maximum propagation delay for a 10 Mbps local area network with a maximum length of 2.5 Km and 4 repeaters is $1/\varsigma = 0.0000256$ sec (which coincides with the transmission time of a 32 byte frame) and the mean backoff period is given by $1/\gamma = 5.4 \cdot 1/\varsigma = 0.00013824$ sec. We show in Table 10.1 the value of $1/\mu$ when varying the frame size and in Table 10.2 the size of the Markovian semantics when varying the number of stations. To compute the channel utilization, we have to assign yield reward 1 to action with type $trans\_msg_i$ in $LineSuccess_i$, whereas for the collision probability we have to assign yield reward 1 to action with type $signal\_coll$ in $LineCollision$. This has been accomplished by providing the following .rew file:

$$\begin{array}{rcl} \hat{}utilization & = & (\; trans\_msg_1 \quad 1 \quad 0 \\ & & \quad \ldots \qquad\quad \ldots \quad \ldots \\ & & \quad trans\_msg_n \quad 1 \quad 0 \;) \\ \hat{}coll\_prob & = & (\; signal\_coll \quad 1 \quad 0 \;) \end{array}$$

The results are shown in Fig. 10.6 and 10.7, respectively. It is worth noting that the channel utilization considerably decreases as the frame size decreases: this is because the longer is the frame, the smaller is the fraction of time during which a collision can occur. Moreover, the collision probability increases as the number of stations increases, as expected.

10.3 Token Ring

The purpose of this section is to study token ring, the IEEE 802.5 standard medium access control protocol for local area networks which avoids message collisions and guarantees a fair use of the medium. We prove a functional property (deadlock freeness) via equivalence checking and we evaluate a performance index (channel utilization) through numerical analysis. Moreover, we show a property (fair circulation of the token) that can be proven only by resorting to both functional and performance arguments [18].

frame size (bytes)	$1/\mu$ (sec)
64	0.0000512
128	0.0001024
256	0.0002048
512	0.0004096
1024	0.0008192

Table 10.1: Propagation rates for $CSMA/CD_n$

stations	states	transitions
2	14	26
3	44	117
4	128	432
5	352	1415
6	928	4284

Table 10.2: Size of $\mathcal{M}[\![CSMA/CD_n]\!]$

10.3.1 Informal Specification

Although the IEEE 802.3 standard for local area networks is widely used in offices, during the development of the 802 standard people from several companies interested in factory automation had serious reservations about it. Due to the probabilistic medium access control protocol, with a little bad luck a station might have to wait arbitrarily long to send a frame.

A simple system with a known worst case for the successful transmission of a frame is a ring in which the stations take turns sending frames. Another attractive feature is the fact that a ring is not really a broadcast medium, but a collection of individual point-to-point links that happen to form a circle. These links involve a well understood and field proven technology and can run on twisted pair, coaxial cable, or fiber optics. Ring engineering is also almost entirely digital, whereas CSMA/CD has a substantial analog component for collision detection. A ring is also fair and has a known upper bound on channel access. For these reasons, IBM chose the ring as its local area network and IEEE has included the *token ring* standard as 802.5 [182].

A ring really consists of a collection of ring interfaces connected by point-to-point lines. Each bit arriving at an interface is copied into a 1-bit buffer and then copied out onto the ring again. While in the buffer, the bit can be inspected and possibly modified before being written out. In a token ring a special 24 bit pattern, called the token, circulates around the ring whenever all the stations are idle. When a station wants to transmit a frame, it is required to seize the token and remove it from the ring before transmitting. Because

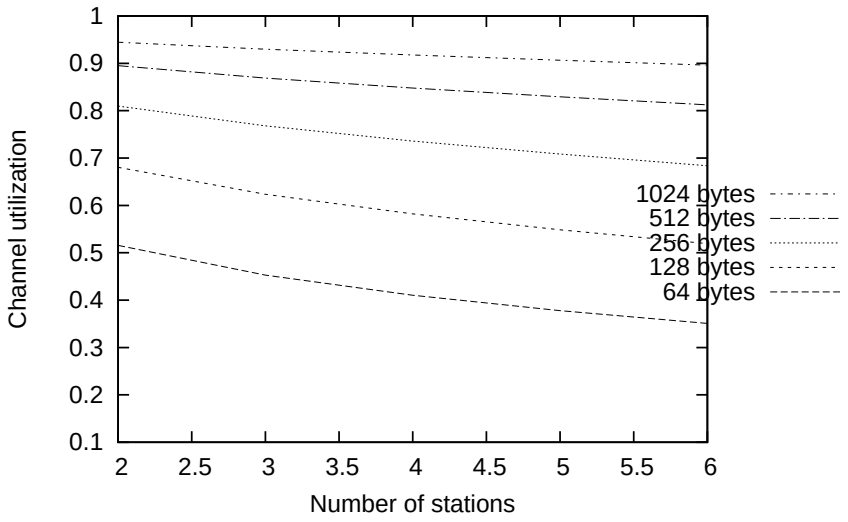


Figure 10.6: Channel utilization of $CSMA/CD_n$

there is only one token, only one station can transmit at a given instant, thus solving the channel access problem. An implication of the token ring design is that the ring itself must have a sufficient delay to contain a complete token to circulate when all the stations are idle.

Ring interfaces have two operating modes: listen and transmit. In listen mode, the input bits are simply copied to output. In transmit mode, which is entered only after the token has been seized, the interface breaks the connection between input and output, entering its own data onto the ring. As bits that have propagated around the ring come back, they are removed from the ring by the sender. The sending station can either save them, to compare with the original data to monitor ring reliability, or discard them. Because the entire frame never appears on the ring at one instant, this ring architecture puts no limit on the size of the frames. After a station has finished transmitting the last bit of its last frame or the token holding time (usually 10 msec) has elapsed, the station must regenerate the token. When the last bit of the last frame has gone around and come back, it must be removed, and the interface must switch back into listen mode immediately, to avoid removing the token that might follow if no other station has removed it.

10.3.2 Formal Description with EMPA

The entities involved in the protocol are n stations, each equipped with a timer for the token holding time, and the token:

$$\begin{aligned} TokenRing_n &\triangleq ((Station_1 \parallel_R Timer_1) \parallel_\emptyset \ldots \parallel_\emptyset (Station_n \parallel_R Timer_n)) \parallel_T Token_1 \\ T &= \{get\_token_i, rel\_token_i \mid 1 \leq i \leq n\} \\ R &= \{start\_timer, interrupt\_timer, signal\_timeout\} \end{aligned}$$

where get_token_i (rel_token_i) is the action type describing the fact that $Station_i$ gets (releases) the token,

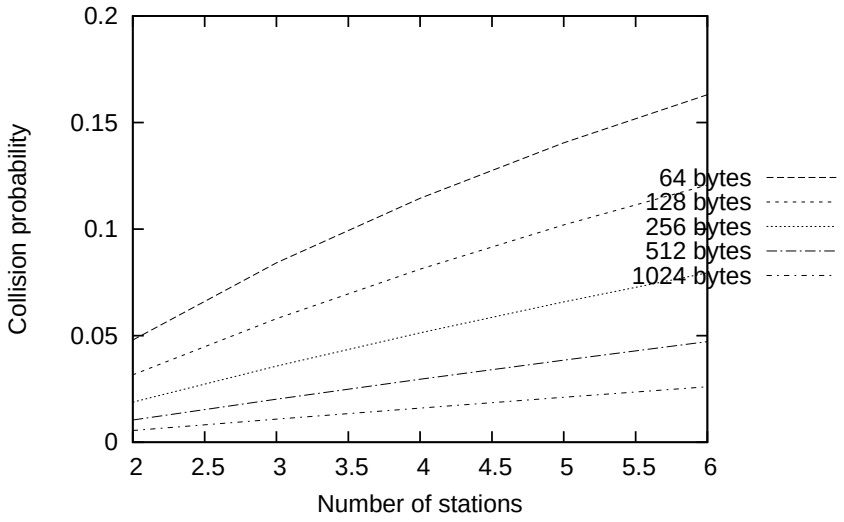


Figure 10.7: Collision probability of $CSMA/CD_n$

start_timer (*interrupt_timer*) is the action type describing the fact that the timer is started (interrupted), and *signal_timeout* is the action type describing the timeout expiration.

Now let us separately model the entities above. Each station is composed of a message generator, a queue, and a message transmitter:

$$Station_i \triangleq MsgGen_i \parallel_{\{gen\_msg_i\}} Queue_{i,0} \parallel_{\{fetch\_msg\}} MsgTrans_i$$

The message generator creates at rate λ_i a stream of messages (gen_msg_i) to be sent:

$$MsgGen_i \triangleq <gen\_msg_i, \lambda_i>.MsgGen_i$$

The queue has capacity c_i and is initially empty. The queue can either receive a new message to be sent (gen_msg_i) or give the next message to be sent (*fetch_msg*) to the message transmitter:

$$\begin{aligned} Queue_{i,0} &\triangleq <gen\_msg_i, *, >.Queue_{i,1} \\ Queue_{i,h} &\triangleq <gen\_msg_i, *, >.Queue_{i,h+1} + \\ &\quad <fetch\_msg, *>.Queue_{i,h-1}, \quad 0 < h < c_i \\ Queue_{i,c_i} &\triangleq <fetch\_msg, *>.Queue_{i,c_i-1} \end{aligned}$$

The message transmitter comes into play upon seizing the token (get_token_i). After starting the timer accounting for the token holding time (*start_timer*), either the first message in the queue is fetched (*fetch_msg*) or the token is released (rel_token_i) and the timer interrupted (*interrupt_timer*) if the queue is empty. If both cases are possible, the transmitter does fetch the message because action with type *fetch_msg* has been given higher priority than action with type rel_token_i. After fetching a message, the message is transmitted ($trans\_msg_i$): should a timeout be signaled in the meanwhile (*signal_timeout*), the message is transmitted

until completion, then the token is released (rel_token_i). Assuming that the length of a message is exponentially distributed, the message transmission time turns out to be exponentially distributed with rate γ_i:

$$\begin{aligned} MsgTrans_i &\triangleq <get\_token_i, *>.<start\_timer, \infty_{1,1}>.MsgTrans'_i \\ MsgTrans'_i &\triangleq <fetch\_msg, \infty_{2,1}>.(<trans\_msg_i, \gamma_i>.MsgTrans'_i + \\ &\qquad <signal\_timeout, *>.<trans\_msg_i, \gamma_i>. \\ &\qquad\quad <rel\_token_i, \infty_{1,1}>.MsgTrans_i) + \\ &\quad <rel\_token_i, \infty_{1,1}>.<interrupt\_timer, \infty_{1,1}>.MsgTrans_i \end{aligned}$$

The timer accounting for the token holding time is started upon seizing the token ($start\_timer$). Then either the token holding time elapses ($elapse\_tht_i$) and a timeout is signaled ($signal\_timeout$) or the timer is interrupted by the message transmitter ($interrupt\_timer$). The token holding time has been described by means of an exponentially distributed random variable with rate μ_i:

$$\begin{aligned} Timer_i &\triangleq <start\_timer, *>.(<elapse\_tht_i, \mu_i>.<signal\_timeout, \infty_{1,1}>.Timer_i + \\ &\qquad <interrupt\_timer, *>.Timer_i) \end{aligned}$$

Finally, the token circulating around the ring is modeled as follows, where the time it takes to get from one station to the subsequent one is represented by means of an exponentially distributed random variable with rate ς_i:

$$\begin{aligned} Token_k &\triangleq <get\_token_k, \varsigma_k>.<rel\_token_k, *>.Token_{k+1}, \qquad 1 \leq k \leq n-1 \\ Token_n &\triangleq <get\_token_n, \varsigma_n>.<rel\_token_n, *>.Token_1 \end{aligned}$$

10.3.3 Functional Analysis

As a functional property, we have proved that the protocol is deadlock free. This has been accomplished by verifying that $\mathcal{F}[\![TokenRing_n/L]\!]$ is weakly bisimilar to $\mathcal{F}[\![Token_1]\!]$, where $L = AType - \{\tau\} - T$.

10.3.4 Performance Analysis

Let us consider a 10 Mbps local area network and assume that all the stations have the same characteristics. We take $\lambda = 1$ msg / μsec, $c = 2$, $\gamma = 0.0195313$ (corresponding to an average message length of 512 bits), $\mu = 0.0001$ (corresponding to a 10 msec token holding time), and $\varsigma = 0.416666$ (corresponding to a 24 bit token). The number n of stations is assumed to vary between 2 and 6.

As a performance measure, we have computed the channel utilization. To this purpose, every action having type $trans\_msg_i$ must be given yield reward 1 in order to single out those states in which the channel is used. To achieve this, the following .rew file has been provided:

stations	states	transitions
2	54	144
3	243	810
4	972	3888
5	3645	17010
6	13122	69984

Table 10.3: Size of $\mathcal{M}[\![TokenRing_n]\!]$

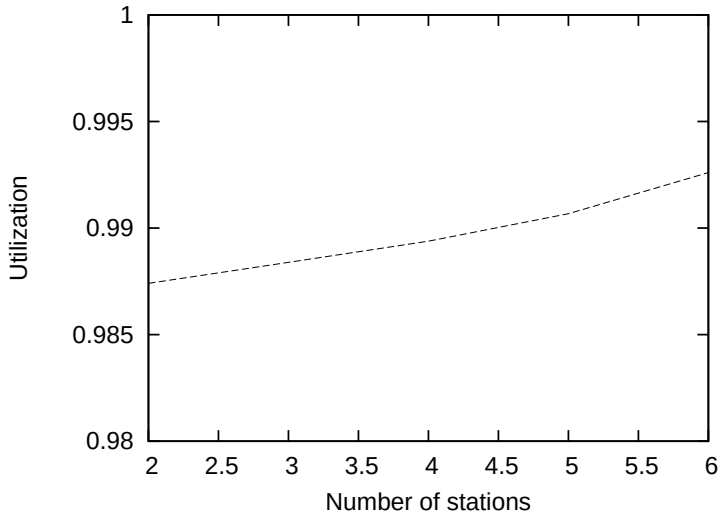


Figure 10.8: Channel utilization of $TokenRing_n$

$$\hat{}utilization = \left(\begin{array}{lll} trans\_msg_1 & 1 & 0 \\ \ldots & \ldots & \ldots \\ trans\_msg_n & 1 & 0 \end{array} \right)$$

We show in Fig. 10.3 the size of the Markovian semantics and in Fig. 10.8 the channel utilization. As expected, the utilization approaches 1 as the number of stations increases.

10.3.5 Combined Analysis

Finally, we prove the fairness of the protocol by taking into account both functional and performance aspects. Showing that token ring is fair amounts to demonstrate that the token does circulate and that the stations are cyclically given the chance to get the token. In order to highlight the relevant actions, the first property is proven by reasoning on $TokenRing_n/L$, where $L = AType - \{\tau\} - T$. The property holds because,

after executing an observable action, $TokenRing_n/L$ eventually performs another observable action without engaging forever in a τ loop. To verify this liveness property, it is sufficient to note that possible τ loops present in $\mathcal{I}[\![TokenRing_n/L]\!]$ are always escaped with probability 1. The second property is a straightforward consequence of the fact that $\mathcal{F}[\![TokenRing_n/L]\!]$ is weakly bisimilar to $\mathcal{F}[\![Token_1]\!]$, as already mentioned, and the structure of $Token_1$.

10.4 ATM Switch

The aim of this section is to formally model an ATM switch in different configurations in order to compute the related cell loss probabilities [15, 16]. More precisely, two algebraic descriptions will be provided and compared: the former concerns an ATM switch supporting UBR service only, the latter an ATM switch supporting UBR and ABR services.

10.4.1 Informal Specification

ATM [182] is the technology adopted to implement broadband integrated services digital networks, i.e. those broadband networks which provide end-to-end digital connectivity to support a wide range of services, including voice and nonvoice services, to which users have access by a limited set of standard multipurpose user network interfaces.

ATM is a packet switching transfer mode based on virtual connections which implements a slotted, asynchronous time division multiplexing. Virtual connections between sources and destinations are established before transmission takes place by suitably setting up the routing tables of the switches in order to allocate a given set of resources which should guarantee the quality of service, and are released once transmission is over. Information to be transmitted along the network is grouped in 53 byte long packets called cells which flow through the corresponding virtual connections by sharing the available physical links in a slotted manner without any predefined schema, thereby reflecting the real traffic load of the network.

An ATM switch is composed of n input ports as well as n output ports, and its task consists of forwarding incoming cells to the output ports according to the routing table that has been set up for the connections to which cells belong. ATM switches are generally synchronous in the sense that, during a cycle, one cell is taken from any input port (if one is present), passed into the internal switching fabric, and eventually transmitted on the appropriate output port. Switches may be pipelined, i.e. it may take several cycles before an incoming cell appears on its output port. Cells actually arrive on the input ports asynchronously, so there is a master clock that marks the beginning of a cycle. Any cell fully arrived when the clock ticks is eligible for switching during that cycle. A cell not fully arrived has to wait until the next cycle. The switch we are going to consider allows for unicast connections only, i.e. those requiring just one output port, is nonblocking, i.e. several incoming cells headed to the output ports can be delivered during the same clock

cycle without contention, and implements output queueing, i.e. a buffer for exceeding cells is located in each output port.

An ATM switch can support several kinds of service. Among them, we cite ABR and UBR. The ABR (Available Bit Rate) service is designed for bursty traffic whose bandwidth range is known roughly (e.g., browsing the web). Using ABR service avoids having to make a long term commitment to a fixed bandwidth. ABR is the service in which the network provides rate feedback to the sender, asking it to slow down when congestion occurs. Assuming that the sender complies with such requests, cell loss for ABR traffic is expected to be low. Instead, the UBR (Unspecified Bit Rate) service makes no promises and gives no feedback about congestion. This category is well suited to sending IP packets, since IP also makes no promises about delivery. All UBR cells are accepted, and if there is capacity left over in the buffers at output ports, they will also be delivered. If congestion occurs, UBR cells will be discarded, with no feedback to the sender and no expectation that the sender slows down.

Since in the following we consider a single ATM switch instead of an ATM network, we shall model the incoming traffic flow but not the source of each connection. As this incoming flow has to change according to the related class of service, we shall model part of the source behavior (such as the rate variations for ABR connections, which depend on the feedback mechanism) directly within the input ports. Analogously, the switch acts as a virtual destination, i.e. it turns around immediately rate variation information for ABR connections which thus depend on the feedback of this switch only. In other words, the load level for the switch depends on the incoming traffic simulated in the input port model and the incoming traffic partially depends on the congestion level and general load information of the switch through the feedback control mechanism simulated in the output port model.

10.4.2 ATM Switch: Formal Description of UBR Service

Because of the slotted nature of the transmission, we resort to a discrete time EMPA description of the switch containing only immediate actions. In order to model the abstraction of the time unit, we introduce the action type *clock* on which all the switch components must periodically synchronize. The switch can be modeled as follows:

$$\begin{aligned} UBRATMSwitch_n &\triangleq Clock \parallel_T ((InputPort_1 \parallel_T \ldots \parallel_T InputPort_n) \parallel_{I \cup T} \\ &\qquad (OutputPort_{1,0} \parallel_T \ldots \parallel_T OutputPort_{n,0})) \\ T &= \{clock\} \\ I &= \{arrive\_cell_{i,j} \mid 1 \leq i, j \leq n\} \end{aligned}$$

where $arrive\_cell_{i,j}$ is the action type describing the arrival of a cell at $InputPort_i$ and its subsequent forwarding to $OutputPort_j$.

We exploit compositionality in order to concentrate on the various components of the switch. First of all, we model *Clock* as follows:

$$Clock \triangleq <clock, \infty_{1,1}>.Clock$$

This timer endlessly performs action $<clock, \infty_{1,1}>$ on which every switch component must synchronize whenever no other local action logically belonging to the same time unit can occur. To enforce this, every term modeling a switch component will enable action $<clock, *>$ at any time, and will consist of actions whose priority level is greater than 1.

Now let us focus our attention on $InputPort_i$. Initially, it can receive a request for establishing a connection ($request\_ubr\_conn_i$). For the sake of simplicity, we assume that every input port can support at most one connection at a time:

$$InputPort_i \triangleq <request\_ubr\_conn_i, \infty_{8,1}>.UBRService_i$$

If the UBR connection can be accepted ($accept\_ubr\_conn_i$), the connection is established with $OutputPort_j$ with probability $1/n$ ($open\_ubr\_conn_{i,j}$), otherwise the connection is refused ($refuse\_ubr\_conn_i$). Whenever $InputPort_i$ is enabled to accept a cell, it can receive either a request for closing the connection with probability q_i ($close\_conn_i$) or a cell with probability $1 - q_i$ ($keep\_conn_i$). If a cell is enabled to arrive in a given time unit, it does arrive with probability p_i ($arrive\_cell_{i,j}$) otherwise $InputPort_i$ is idle ($idle_{i,j}$), hence we are assuming a Bernoulli distribution for the incoming traffic. Since the UBR service does not guarantee the quality of service as every portion of unused bandwidth is freely exploited without observing any flow control mechanism, $UBRService_i$ can be modeled as follows:

$$\begin{aligned}
UBRService_i &\triangleq <accept_ubr_conn_i, \infty_{7,1}>. \sum_{j=1}^{n} <open_ubr_conn_{i,j}, \infty_{6,1/n}>.UBRConn_{i,j} + \\
&\quad <refuse_ubr_conn_i, \infty_{7,1}>.InputPort_i \\
UBRConn_{i,j} &\triangleq <clock, *>.(<keep_conn_i, \infty_{5,1-q_i}>.UBRCell_{i,j} + \\
&\qquad\qquad <close_conn_i, \infty_{5,q_i}>.InputPort_i) \\
UBRCell_{i,j} &\triangleq <arrive_cell_{i,j}, \infty_{3,p_i}>.UBRConn_{i,j} + \\
&\quad <idle_{i,j}, \infty_{3,1-p_i}>.UBRConn_{i,j}
\end{aligned}$$

Finally, let us model output ports. $OutputPort_j$ is essentially a FIFO buffer with capacity c_j:

$$OutputPort_{j,m} \triangleq <clock, *>.ArriveCheck_{j,m,\{1,\ldots,n\}}, \quad 0 \leq m \leq c_j$$

At each time unit $OutputPort_j$ can do nothing ($wait_j$) if it is empty and there is no incoming cell, send the first cell in the buffer ($send\_cell_j$) if it is not empty and there is no incoming cell, or accept an incoming cell ($arrive\_cell_{k,j}$) and send the first cell in the buffer ($send\_cell_j$) if the buffer is not empty and there is an incoming cell. Since $OutputPort_j$ can be simultaneously connected to several input ports, it can receive several incoming cells during the same time unit, and those cells which cannot be accommodated in the buffer are lost ($lose\_cell_j$). For $|K| = n$ we have:

$$ArriveCheck_{j,0,K} \triangleq \sum_{k \in K} <arrive\_cell_{k,j}, *>.<send\_cell_j, \infty_{4,1}>.ArriveCheck_{j,0,K-\{k\}} + <wait_j, \infty_{2,1}>.OutputPort_{j,0}$$

$$ArriveCheck_{j,m,K} \triangleq \sum_{k \in K} <arrive\_cell_{k,j}, *>.<send\_cell_j, \infty_{4,1}>.ArriveCheck_{j,m,K-\{k\}} + <send\_cell_j, \infty_{2,1}>.OutputPort_{j,m-1}, \quad 1 \leq m \leq c_j$$

while for $1 < |K| < n$ we have:

$$ArriveCheck_{j,m,K} \triangleq \sum_{k \in K} <arrive\_cell_{k,j}, *>.ArriveCheck_{j,m+1,K-\{k\}} + OutputPort_{j,m}, \quad 0 \leq m < c_j$$

$$ArriveCheck_{j,c_j,K} \triangleq \sum_{k \in K} <arrive\_cell_{k,j}, *>.<lose\_cell_j, \infty_{4,1}>.ArriveCheck_{j,c_j,K-\{k\}} + OutputPort_{j,c_j}$$

and for $|K| = 1$ we have:

$$ArriveCheck_{j,m,\{i\}} \triangleq <arrive\_cell_{i,j}, *>.OutputPort_{j,m+1} + OutputPort_{j,m}, \quad 0 \leq m < c_j$$

$$ArriveCheck_{j,c_j,\{i\}} \triangleq <arrive\_cell_{i,j}, *>.<lose\_cell_j, \infty_{4,1}>.OutputPort_{j,c_j} + OutputPort_{j,c_j}$$

Note that actions with type $send\_cell_j$ or $wait_j$ which are alternative to actions $<arrive\_cell_{k,j}, *>$ (having priority level 3 in $UBRCell_{k,j}$) have been given priority level 2 so that all the cell arrivals are taken into account. Actions with type $send\_cell_j$ or $lose\_cell_j$ which follow actions $<arrive\_cell_{k,j}, *>$ have instead been given priority level 4 so that, in case of several input ports currently connected to the same output port, a deferred probabilistic choice between action $<arrive\_cell_{i,j}, \infty_{3,p_i}>$ and action $<idle_{i,j}, \infty_{3,1-p_i}>$ in $UBRCell_{i,j}$ is correctly preserved (instead of being biased in favor of the latter if action with type $send\_cell_j$ or $lose\_cell_j$ had priority level 3).

10.4.3 ATM Switch: Formal Description of UBR and ABR Services

In order to cope with bursty traffic, we now assume that the switch is able to provide the ABR service as well. Congestion control is based on the idea that each sender has a current rate that falls between the minimum cell rate and the peak cell rate agreed upon. When congestion occurs, the rate is reduced. When congestion is absent, the rate is increased. The switch can be modeled as follows:

$$ABRUBRATMSwitch_n \triangleq Clock \parallel_T ((InputPort_1 \parallel_T \ldots \parallel_T InputPort_n) \parallel_{I \cup T} (OutputPort_{1,0} \parallel_T \ldots \parallel_T OutputPort_{n,0}))$$

$$T = \{clock\}$$

$$I = \{arrive\_cell_{i,j}, increase\_speed_{i,j}, reduce\_speed_{i,j} \mid 1 \leq i, j \leq n\}$$

where $increase\_speed_{i,j}$ and $reduce\_speed_{i,j}$ are action types describing flow control messages from $OutputPort_j$ to $InputPort_i$.

In the following, we only present the terms whose structure has changed with respect to the previous section. $InputPort_i$ is modified as follows:

$$\begin{aligned} InputPort_i \triangleq\ & <request\_abr\_conn_i, \infty_{8,1}>.ABRService_i + \\ & <request\_ubr\_conn_i, \infty_{8,1}>.UBRService_i \end{aligned}$$

Let us now model the ABR service. The ABR service guarantees a minimal bandwidth provided that a suitable flow control algorithm is implemented. If the ABR connection request is accepted ($accept\_abr\_conn_i$), then the connection is established with $OutputPort_j$ with probability $1/n$ ($open\_abr\_conn_{i,j}$) and the transmission proceeds at high speed, which means that $InputPort_i$ can accept an incoming cell every time unit. If $InputPort_i$ receives from $OutputPort_j$ a reduce speed message ($reduce\_speed_{i,j}$), then the transmission proceeds at middle speed, which means that $InputPort_i$ can accept an incoming cell every two time units. If $InputPort_i$ receives from $OutputPort_j$ an additional reduce speed message, then the transmission proceeds at low speed, which means that $InputPort_i$ can accept an incoming cell every three time units. The transmission can subsequently proceed faster if $InputPort_i$ receives from $OutputPort_j$ an increase speed message ($increase\_speed_{i,j}$). $ABRService_i$ can then be modeled as follows:

$$\begin{aligned} ABRService_i \triangleq\ & <accept\_abr\_conn_i, \infty_{7,1}>.\sum_{j=1}^{n} <open\_abr\_conn_{i,j}, \infty_{6,1/n}>.ABRConn_{high,i,j} + \\ & <refuse\_abr\_conn_i, \infty_{7,1}>.InputPort_i \end{aligned}$$

where high speed traffic is given by:

$$\begin{aligned} ABRConn_{high,i,j} \triangleq\ & <clock, *>.(<keep\_conn_i, \infty_{5,1-q_i}>.ABRCell_{high,i,j} + \\ & \qquad <close\_conn_i, \infty_{5,q_i}>.InputPort_i) \\ ABRCell_{high,i,j} \triangleq\ & <arrive\_cell_{i,j}, \infty_{3,p_i}>.ReduceCheck_{high,i,j} + \\ & <idle_{i,j}, \infty_{3,1-p_i}>.IncreaseCheck_{high,i,j} \end{aligned}$$

middle speed traffic is given by:

$$\begin{aligned} ABRConn_{mid,i,j} \triangleq\ & <clock, *>.IncreaseCheck_{mid,i,j} \\ ABRConn'_{mid,i,j} \triangleq\ & <clock, *>.(<keep\_conn_i, \infty_{5,1-q_i}>.ABRCell_{mid,i,j} + \\ & \qquad <close\_conn_i, \infty_{5,q_i}>.InputPort_i) \\ ABRCell_{mid,i,j} \triangleq\ & <arrive\_cell_{i,j}, \infty_{3,p_i}>.ReduceCheck_{mid,i,j} + \\ & <idle_{i,j}, \infty_{3,1-p_i}>.IncreaseCheck'_{mid,i,j} \end{aligned}$$

and low speed traffic is given by:

$$\begin{aligned} ABRConn_{low,i,j} \triangleq\ & <clock, *>.IncreaseCheck_{low,i,j} \\ ABRConn'_{low,i,j} \triangleq\ & <clock, *>.IncreaseCheck'_{low,i,j} \\ ABRConn''_{low,i,j} \triangleq\ & <clock, *>.(<keep\_conn_i, \infty_{5,1-q_i}>.ABRCell_{low,i,j} + \\ & \qquad <close\_conn_i, \infty_{5,q_i}>.InputPort_i) \\ ABRCell_{low,i,j} \triangleq\ & <arrive\_cell_{i,j}, \infty_{3,p_i}>.ReduceCheck_{low,i,j} + \\ & <idle_{i,j}, \infty_{3,1-p_i}>.IncreaseCheck''_{low,i,j} \end{aligned}$$

where:

$$\begin{aligned}
IncreaseCheck_{high,i,j} &\triangleq <increase_speed_{i,j},*>.ABRConn_{high,i,j} + \\
&\quad ABRConn_{high,i,j} \\
IncreaseCheck_{mid,i,j} &\triangleq <increase_speed_{i,j},*>.ABRConn_{high,i,j} + \\
&\quad ABRConn'_{mid,i,j} \\
IncreaseCheck'_{mid,i,j} &\triangleq <increase_speed_{i,j},*>.ABRConn_{high,i,j} + \\
&\quad ABRConn_{mid,i,j} \\
IncreaseCheck_{low,i,j} &\triangleq <increase_speed_{i,j},*>.ABRConn'_{mid,i,j} + \\
&\quad ABRConn'_{low,i,j} \\
IncreaseCheck'_{low,i,j} &\triangleq <increase_speed_{i,j},*>.ABRConn'_{mid,i,j} + \\
&\quad ABRConn''_{low,i,j} \\
IncreaseCheck''_{low,i,j} &\triangleq <increase_speed_{i,j},*>.ABRConn_{mid,i,j} + \\
&\quad ABRConn_{low,i,j}
\end{aligned}$$

and:

$$\begin{aligned}
ReduceCheck_{high,i,j} &\triangleq <reduce_speed_{i,j},*>.ABRConn_{mid,i,j} + \\
&\quad ABRConn_{high,i,j} \\
ReduceCheck_{mid,i,j} &\triangleq <reduce_speed_{i,j},*>.ABRConn_{low,i,j} + \\
&\quad ABRConn_{mid,i,j} \\
ReduceCheck_{low,i,j} &\triangleq <reduce_speed_{i,j},*>.ABRConn_{low,i,j} + \\
&\quad ABRConn_{low,i,j}
\end{aligned}$$

As far as output ports are concerned, the flow control algorithm required by the ABR service introduces a lower threshold l_j and an upper threshold u_j: whenever u_j is exceeded reduce speed messages are issued, while increase speed messages are sent back to input ports if the number of cells in the buffer falls below l_j. For $|K| = n$ we have:

$$\begin{aligned}
ArriveCheck_{j,0,K} &\triangleq \sum_{k\in K} <arrive_cell_{k,j},*>.<send_cell_j,\infty_{4,1}>.ArriveCheck_{j,0,K-\{k\}} + \\
&\quad <wait_j,\infty_{2,1}>.OutputPort_{j,0} \\
ArriveCheck_{j,l_j,K} &\triangleq \sum_{k\in K} <arrive_cell_{k,j},*>.<send_cell_j,\infty_{4,1}>.ArriveCheck_{j,l_j,K-\{k\}} + \\
&\quad <send_cell_j,\infty_{2,1}>.IncreaseSpeedSignal_{j,K} \\
ArriveCheck_{j,m,K} &\triangleq \sum_{k\in K} <arrive_cell_{k,j},*>.<send_cell_j,\infty_{4,1}>.ArriveCheck_{j,m,K-\{k\}} + \\
&\quad <send_cell_j,\infty_{2,1}>.OutputPort_{j,m-1},\ \ 1 \le m < u_j \wedge m \neq l_j \\
ArriveCheck_{j,m,K} &\triangleq \sum_{k\in K} <arrive_cell_{k,j},*>.<send_cell_j,\infty_{4,1}>.ReduceSpeedSignal_{k,j,m,K} + \\
&\quad <send_cell_j,\infty_{2,1}>.OutputPort_{j,m-1},\ \ u_j \le m \le c_j
\end{aligned}$$

while for $1 < |K| < n$ we have:

$$
\begin{aligned}
ArriveCheck_{j,m,K} &\triangleq \sum_{k\in K} <arrive\_cell_{k,j},*>.ArriveCheck_{j,m+1,K-\{k\}} + \\
&\quad OutputPort_{j,m},\ \ 0 \le m < u_j \\
ArriveCheck_{j,m,K} &\triangleq \sum_{k\in K} <arrive\_cell_{k,j},*>.ReduceSpeedSignal_{k,j,m,K} + \\
&\quad OutputPort_{j,m},\ \ u_j \le m < c_j \\
ArriveCheck_{j,c_j,K} &\triangleq \sum_{k\in K} <arrive\_cell_{k,j},*>.<lose\_cell_j,\infty_{4,1}>.ReduceSpeedSignal_{k,j,m,K} + \\
&\quad OutputPort_{j,c_j}
\end{aligned}
$$

and for $|K| = 1$ we have:

$$
\begin{aligned}
ArriveCheck_{j,m,\{i\}} &\triangleq <arrive\_cell_{i,j},*>.OutputPort_{j,m+1} + \\
&\quad OutputPort_{j,m},\ \ 0 \le m < u_j \\
ArriveCheck_{j,m,\{i\}} &\triangleq <arrive\_cell_{i,j},*>.ReduceSpeedSignal_{i,j,m,\{i\}} + \\
&\quad OutputPort_{j,m},\ \ u_j \le m < c_j \\
ArriveCheck_{j,c_j,\{i\}} &\triangleq <arrive\_cell_{i,j},*>.<lose\_cell_j,\infty_{4,1}>.ReduceSpeedSignal_{i,j,c_j,\{i\}} + \\
&\quad OutputPort_{j,c_j}
\end{aligned}
$$

where:

$$
\begin{aligned}
IncreaseSpeedSignal_{j,K} &\triangleq \sum_{k\in K} <increase\_speed_{k,j},\infty_{4,1}>.IncreaseSpeedSignal_{j,K-\{k\}} + \\
&\quad OutputPort_{j,l_j-1},\ \ |K| > 1 \\
IncreaseSpeedSignal_{j,\{i\}} &\triangleq <increase\_speed_{i,j},\infty_{4,1}>.OutputPort_{j,l_j-1} + \\
&\quad OutputPort_{j,l_j-1}
\end{aligned}
$$

and:

$$
\begin{aligned}
ReduceSpeedSignal_{i,j,m,K} &\triangleq <reduce\_speed_{i,j},\infty_{4,1}>.ArriveCheck_{j,m,K-\{i\}} + \\
&\quad ArriveCheck_{j,m,K-\{i\}},\ \ |K| = n \wedge u_j \le m \le c_j \\
ReduceSpeedSignal_{i,j,m,K} &\triangleq <reduce\_speed_{i,j},\infty_{4,1}>.ArriveCheck_{j,m+1,K-\{i\}} + \\
&\quad ArriveCheck_{j,m+1,K-\{i\}},\ \ 1 < |K| < n \wedge u_j \le m < c_j \\
ReduceSpeedSignal_{i,j,c_j,K} &\triangleq <reduce\_speed_{i,j},\infty_{4,1}>.ArriveCheck_{j,c_j,K-\{i\}} + \\
&\quad ArriveCheck_{j,c_j,K-\{i\}},\ \ 1 < |K| < n \\
ReduceSpeedSignal_{i,j,m,\{i\}} &\triangleq <reduce\_speed_{i,j},\infty_{4,1}>.OutputPort_{j,m+1} + \\
&\quad OutputPort_{j,m+1},\ \ u_j \le m < c_j \\
ReduceSpeedSignal_{i,c_j,m,\{i\}} &\triangleq <reduce\_speed_{i,j},\infty_{4,1}>.OutputPort_{j,c_j} + \\
&\quad OutputPort_{j,c_j}
\end{aligned}
$$

Observe that modeling speed message related operations is made complicated by the fact that, for the sake of simplicity, $OutputPort_j$ does not keep track of the currently established connections nor the related kind of service. As a consequence, increase speed messages must be sent only to those input ports currently connected because of an ABR service request, and reduce speed messages must be sent to the input port causing the upper threshold overflow only in case of ABR service.

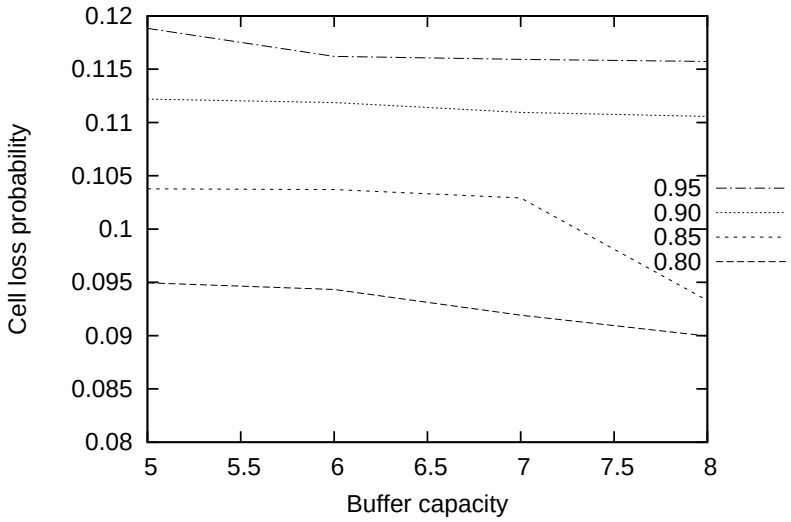


Figure 10.9: Cell loss probability for $UBRATMSwitch_2$

10.4.4 Performance Analysis

We now compare the performance of the two different switches by computing their cell loss probabilities for several buffer capacities c and parameters p of the Bernoulli incoming traffic. Let us assume that $n = 2$, i.e. there are only two input ports and two output ports, that the input ports all have the same characteristics ($q = 10^{-6}$), and that the output ports all have the same characteristics as well. We show in Table 10.4 the size of the Markovian semantics for the two switches, respectively.

buffer capacity	states	transitions	states	transitions
5	2079	3440	18983	29606
6	2821	4666	24653	38548
7	3693	6106	30975	48538
8	4695	7760	37949	59576

Table 10.4: Size of $\mathcal{M}[\![UBRATMSwitch_2]\!]$ and $\mathcal{M}[\![ABRUBRATMSwitch_2]\!]$

In order to measure the cell loss probability, we assign yield reward 1 to every action having type $lose\_cell_j$. To achieve this, the following .rew file has been provided:

$$\begin{array}{rcllll} \hat{}cell\_loss\_prob & = & (& lose\_cell_1 & 1 & 0 \\ & & & \ldots & \ldots & \ldots \\ & & & lose\_cell_n & 1 & 0 \quad) \end{array}$$

The results are plotted in Fig. 10.9 and 10.10, respectively. As can be observed, the cell loss probability

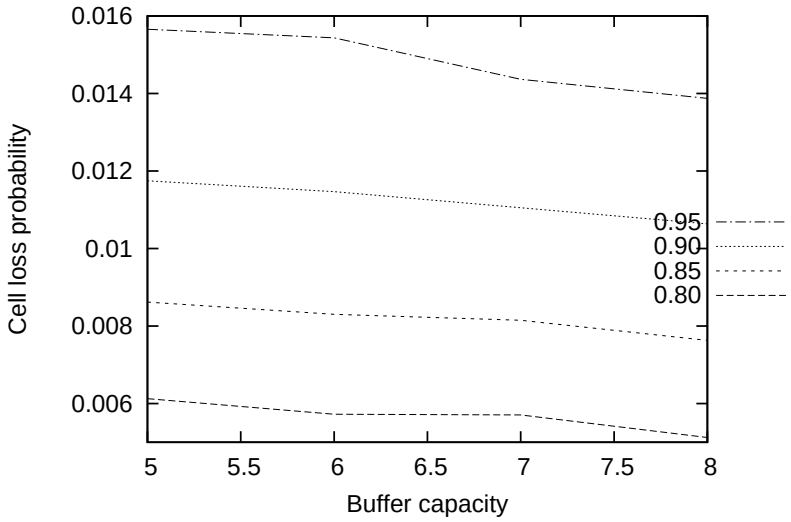


Figure 10.10: Cell loss probability for $UBRABRATMSwitch_2$

is reduced by one order of magnitude when the ABR congestion control mechanism comes into play. As expected, in both cases the cell loss probability increases (decreases) as the traffic increases (the buffer capacity increases).

We conclude by noting the considerable number of states and transitions of the performance model underlying $ABRUBRATMSwitch_2$. In order to analyze the performance of switches with a more realistic number of ports, it is possible to use the routines for the simulative analysis of performance implemented in TwoTowers so that the construction of the entire state space can be avoided.

10.5 Adaptive Mechanism for Packetized Audio

In this section we report on the use of EMPA_{vp} to formally describe an adaptive mechanism for packetized audio over the Internet in order to predict its performance [169, 30]. The percentage of audio packets that are played out by two different versions of the audio mechanism is assessed via simulative analysis and then compared with results obtained from experiments conducted on the field with a prototype implementation of the mechanism itself.

The contribution of this case study is twofold. On the one hand, the usefulness of value passing is emphasized. EMPA_{vp} has been used to model the audio mechanism because its execution strongly depends on the use of clock values and packet timestamps. Moreover, the possibility of modeling general distributions is a requirement in order to describe the audio mechanism of interest, as packet transmission delays are typically assumed to follow a Gaussian distribution. Such a requirement is met by EMPA_{vp} value passing based expressions, provided that performance is evaluated via simulation.

On the other hand, the construction of the formal algebraic model was part of the design process of the audio mechanism. The predicted performance figures obtained from the algebraic model have illustrated in advance the adequacy of the approach adopted in the design of the audio playout delay control mechanism. Based on these performance figures, it has been possible to implement and develop the complete mechanism without incurring additional costs due to the late discovery of unexpected errors or inefficiency.

10.5.1 Informal Specification

Since the early experiments with packetized voice with Arpanet [57], packetized audio applications have now turned out into tools that many Internet (and Intranet) users try to use frequently. For example, the audio conversations of many international conferences and workshops are now usually transmitted over the Mbone, an experimental overlay network of the Internet [126]. Typically, an audio segment may be considered as being constituted of talkspurt periods, during which the audio activity is carried out, and silence periods, during which no audio packets are generated. In order for the receiving site to reconstruct the audio conversation, the audio packets of a talkspurt must be played out in the order they were emitted at the sending site. If the delay between the arrival of subsequent packets is constant (i.e. there is no jitter), a receiving site may simply play out the arriving audio packets as soon as they are received. However, this is only rarely the case, since jitter free, ordered, on-time packet delivery almost never occurs in today's packet switched networks. Those variations in the arrivals of subsequent packets strongly depend on the traffic conditions of the underlying network. Packet loss rates (due to the effective loss and damage of packets as well as late arrivals) have been measured over the Internet as varying between 15% and 40% [138]. In addition, extensive experiments with wide area network testbeds have shown that the delays between consecutive packets may also be as much as 1.5 sec, thus impairing real time interactive human conversations. Hence, the most used approach adopted by all the most popular audio tools consists of adapting the applications to the jitter present on the network. In order to compensate for these variable delays, a smoothing playout buffer is used at the receiver. Received audio packets are first queued into the buffer, then the playout of a packet of a given talkspurt is delayed for some quantity of time beyond the reception of the first packet of that talkspurt. Needless to say a crucial tradeoff exists between the length of the imposed playout delay and the amount of lost packets due to their late arrival: the longer the additional playout delay, the more likely it is that a packet will arrive before its scheduled playout deadline. However, too long playout delays may in turn seriously compromise the quality of the conversation over the network. Typical acceptable values for the end-to-end delay between packet audio generation at the sending site and its playout time at the receiver are in the range of 300-400 msec, instead a percentage of no more than 5-10% of packet loss is considered quite tolerable in human conversations [34].

As mentioned above, a receiving site in an audio application buffers packets and delays their playout time. Such a playout delay is usually adaptively adjusted from one talkspurt to the next. In order to implement this playout control policy, all the audio applications make use of the two following strong assumptions:

i) an external mechanism is assumed to exist that keeps synchronized the two system clocks at both the sending and the receiving site (for example, the Internet based Network Time Protocol NTP); ii) the delays experienced by audio packets on the network are assumed to follow a Gaussian distribution. Based on these assumptions, the playout control mechanisms work by calculating the playout time for any packet according to given formulas.

Recently, a new on-line, adaptive mechanism for the control of the playout delay has been proposed that overcomes the drawbacks of the existing audio tools, while maintaining satisfiable values of the tradeoff between the average playout delay and the packet loss due to late arrivals [170]. More precisely, this policy does not assume a Gaussian distribution for the end-to-end transmission delays of the audio packets and provides: i) an internal and accurate technique that maintains tight time synchronization between the system clocks of both the sending and the receiving hosts, ii) a method for adaptively estimating the audio packet playout time (on a per-talkspurt basis) with an associated minimal computational complexity, iii) an exact and simple method for dimensioning the playout buffer depending on the traffic conditions on the underlying network.

A simplified description of the mechanism is as follows. When the sender transmits a packet, the sender timestamps that packet with the value of the reading of its own clock. When the packet arrives at the receiver, its attached timestamp is compared with the value of the reading of the receiver's clock. If the timestamp is equal to the value of the clock of the receiver, that packet is immediately played out. If the timestamp is greater than the value of the clock of the receiver, that packet is buffered and its playout time is scheduled after a time interval equal to the positive difference between the value of the timestamp and the current value of the receiver's clock. Finally, if the timestamp is less than the value of the clock of the receiver, the packet is simply discarded since it is too late for presentation. However, due to the fluctuant delays in real transmissions, the value of the clocks of the sender and the receiver at a given time instant may differ of the following quantity $T_S - T_R = \Delta$, where T_S and T_R are, respectively, the readings of the local clocks at the sender and at the receiver, and Δ (assuming identical drifts for both clocks) is a non negative quantity ranging between 0, a theoretical lower bound, and Δ_{max}, a theoretical upper bound on the transmission delays introduced by the network between the sender and the receiver.

In the mechanism proposed in [170], a technique has been devised that is used to estimate an upper bound for the maximum delay transmission. This technique exploits the so called Round Trip Time (RTT) and is based on a three-way handshake protocol. It works as follows. Prior to the beginning of the first talkspurt in an audio conversation, a probe packet is sent from the sender to the receiver timestamped with the clock value of the sender (say C). At the reception of this probe packet, the receiver sets its own clock to the value of the timestamp attached to the probe packet and sends immediately back to the sender a response packet with attached the same timestamp C. Upon the receiving of this response packet, the sender computes the value of the RTT by subtracting from the current value of its local clock the value of the timestamp C. At that moment, the difference between the two clocks, respectively at the sender and at the receiver, is

equal to an unknown quantity (say t_0) which may range from a theoretical lower bound of 0 (that is all the RTT value has been consumed on the way back from the receiver to the sender) and a theoretical upper bound of RTT (that is all the RTT has been consumed on the way in during the transmission of the probe packet). Unfortunately, as already mentioned, t_0 is unknown and a scarce approximation of this value (e.g. $t_0 = RTT/2$) might result in both playout buffer underflow problems and packet loss due to late arrivals. Based on these considerations, in the proposed mechanism the sender, after receiving the response packet from the receiver and calculating the RTT value, sends to the receiver a final installation packet, with attached the previously calculated RTT value. Upon receiving this installation packet, the receiver sets the time of its local clock by subtracting from the value shown at its clock at the arrival of the installation packet the value of the transmitted RTT. Hence, at that precise moment, the difference between the two clocks at the receiver and at the sender is equal to a value Δ given by $\Delta = T_S - T_R = t_0 + RTT$, where Δ ranges in the interval $[RTT, 2 \cdot RTT]$ depending on the unknown value of t_0, which, in turn, may range in the interval $[0, RTT]$. In order for the proposed policy to adaptively adjust to the highly fluctuant end-to-end delays experienced over wide area packet switched networks (like the Internet), the above mentioned synchronization technique is first carried out prior to the beginning of the first talkspurt of the audio conversation, then periodically repeated throughout the entire conversation (the adopted period is about 1 sec).

Hence, each time a new value for RRT is computed by the sender, it may be used by the receiver for adaptively setting the value of its local clock and the playout buffer dimension, as mentioned above. This method guarantees that both the introduced additional playout time and the buffer dimension are always proportional to the traffic conditions.

10.5.2 Formal Description of Initial Synchronization

In this section we formally describe the audio mechanism proposed in [170] in the case in which the synchronization is executed only once prior to the beginning of the first talkspurt of the audio conversation. Three entities are involved in the modeling of the audio mechanism: the sender (termed *ISSender* in the specification below), the receiver (termed *ISReceiver*), and the channel (termed *ISChannel*) over which packets are transmitted. The interaction between the sender and the channel is modeled through the set SC composed of the following actions: *prepare_packet* denoting the fact that a packet is put on the channel, *prepare_probe* denoting the fact that a probe message is put on the channel, *trans_response* denoting the reception of a synchronization response at the sender, *prepare_install* denoting the fact that an install message is put on the channel, and *trans_ack* denoting the reception of an acknowledgement at the sender. The interaction between the receiver and the channel is modeled through the set CR of the following actions: *trans_packet* denoting the reception of a packet at the receiver, *trans_probe* denoting the reception of a probe message at the receiver, *prepare_response* denoting the fact that a synchronization response is put on the channel, *trans_install* denoting the reception of an install message at the receiver, and *prepare_ack* denoting the fact that an acknowledgement is put on the channel. Hence, based on the above actions, the overall scenario for

the audio mechanism can be modeled as follows:

- $PacketizedAudio(\text{int } clock\_s;) \triangleq$
  $ISSender(clock\_s) \parallel_{SC} ISChannel \parallel_{CR} ISReceiver$
- $SC = \{prepare\_packet, prepare\_probe, trans\_response, prepare\_install, trans\_ack\}$
- $CR = \{trans\_packet, trans\_probe, prepare\_response, trans\_install, prepare\_ack\}$

At the beginning the clock of the sender and the clock of the receiver get synchronized. To accomplish this, the sender sends to the receiver a so called probe message carrying as a timestamp the value of its clock. When receiving such a probe message, the receiver sets its clock to the value of the timestamp and sends a response message back to the sender. Upon reception of the response, the sender transmits to the receiver an install message carrying as a timestamp the difference between the current value of its clock and the value of its clock when it sent the probe message (i.e. the RTT value). Finally, upon reception of the install message, the receiver decreases its clock by the value of the timestamp and sends an acknowledgement back to the sender. The initial synchronization of the two clocks may be therefore modeled as follows:

- $ISSender(\text{int } clock\_s;) \triangleq$
  $<prepare\_probe!(clock\_s), \infty_{1,1}>.ISSender'(clock\_s\ ++1, clock\_s)$

  $ISSender'(\text{int } clock\_s, \text{int } old\_clock\_s;) \triangleq$
  $<trans\_response, *>.<prepare\_install!(clock\_s\ --old\_clock\_s), \infty_{1,1}>.ISSender''(clock\_s\ ++1)\ +$
  $<elapse\_tick, \infty_{1,1}>.ISSender'(clock\_s\ ++1, old\_clock\_s)$

  $ISSender''(\text{int } clock\_s;) \triangleq$
  $<trans\_ack, *>.Sender(clock\_s\ ++1)\ +$
  $<elapse\_tick, \infty_{1,1}>.ISSender''(clock\_s\ ++1)$
- $ISChannel(;\text{int } timestamp) \triangleq$
  $<prepare\_probe?(timestamp), *>.ISChannel'(timestamp, 0, \text{gauss}(100, 7))$

  $ISChannel'(\text{int } timestamp, \text{int } clock\_c, \text{int } delivery\_time;) \triangleq$
  if $(delivery\_time <= clock\_c)$ then
  $<trans\_probe!(timestamp), \infty_{1,1}>.<prepare\_response, *>.ISChannel''(0, \text{gauss}(100, 7))$
  else
  $<elapse\_tick, \infty_{1,1}>.ISChannel'(timestamp, clock\_c\ ++1, delivery\_time)$

  $ISChannel''(\text{int } clock\_c, \text{int } delivery\_time; \text{int } timestamp) \triangleq$
  if $(delivery\_time <= clock\_c)$ then

$<trans\_response, \infty_{1,1}>.<prepare\_install?(timestamp), *>.$
$ISChannel'''(timestamp, 0, \text{gauss}(100, 7))$
else
$<elapse\_tick, \infty_{1,1}>.ISChannel''(clock\_c\ ++1, delivery\_time)$

$ISChannel'''(\text{int } timestamp, \text{int } clock\_c, \text{int } delivery\_time;) \triangleq$
if $(delivery\_time <= clock\_c)$ then
$<trans\_install!(timestamp), \infty_{1,1}>.<prepare\_ack, *>.ISChannel''''(0, \text{gauss}(100, 7))$
else
$<elapse\_tick, \infty_{1,1}>.ISChannel'''(timestamp, clock\_c\ ++1, delivery\_time)$

$ISChannel''''(\text{int } clock\_c, \text{int } delivery\_time;) \triangleq$
if $(delivery\_time <= clock\_c)$ then
$<trans\_ack, \infty_{1,1}>.Channel(0, [\,])$
else
$<elapse\_tick, \infty_{1,1}>.ISChannel''''(clock\_c\ ++1, delivery\_time)$

- $ISReceiver(;\text{int } timestamp) \triangleq$
$<trans\_probe?(timestamp), *>.<prepare\_response, \infty_{1,1}>.ISReceiver'(timestamp)$

$ISReceiver'(\text{int } clock\_r; \text{int } timestamp) \triangleq$
$<trans\_install?(timestamp), *>.<prepare\_ack, \infty_{1,1}>.Receiver(clock\_r\ --timestamp, [\,])\ +$
$<elapse\_tick, \infty_{1,1}>.ISReceiver'(clock\_r\ ++1)$

A few comments about the specification above are now in order. In particular, note that all the nonpassive actions are immediate. Moreover, for the purpose of this paper, we have decided to use a Gaussian distribution with mean 100 msec and standard deviation 7 msec to model the message transmission time in our simulation. The reasons for these choices are the following. First it is worth mentioning that, even if the audio mechanism has been designed and implemented to cope with general network traffic scenarios, a Gaussian distribution has been selected to carry out the simulation, following the widely accepted audio traffic assumption detailed in Sect. 10.5.1, so that performance comparisons with other audio mechanisms have been made possible. Second, the specific values of the Gaussian distribution parameters (the mean delay and its associated standard deviation) have been chosen based on several experiments conducted over a UDP connection between Cesena (a remote site of the University of Bologna) and Geneva (Switzerland). Finally, we would like to point out that the modeling of the Gaussian distribution above has been made possible by the value passing capability embedded in EMPA_{vp}. In fact, even if only exponential distributions might be directly expressed, Gaussian durations have been represented by storing their values into constant parameters (e.g. *delivery_time*) and by using clock variables (e.g. *clock_c*) to determine when the

corresponding actions (e.g. *trans_probe*) have to be executed. The same modeling technique will be used in the rest of the formal description of the mechanism.

After the initial synchronization has been carried out, the sender begins generating and transmitting audio packets over the channel every 20 msec. The following is the formal specification of the above mentioned sender's activity:

- $Sender(\text{int } clock\_s;) \triangleq$
  $PacketGen(clock\_s, clock\_s\ ++20)$

- $PacketGen(\text{int } clock\_s, \text{int } trans\_time;) \triangleq$
  if $(clock\_s == trans\_time)$ then
  $<prepare\_packet!(clock\_s), \infty_{1,1}>.PacketGen(clock\_s\ ++1, trans\_time\ ++20)$
  else
  $<elapse\_tick, \infty_{1,1}>.PacketGen(clock\_s\ ++1, trans\_time)$

The main activity of the channel consists of: (i) accepting audio packets from the sender, (ii) sampling their delivery time according to the Gaussian distribution above, (iii) queueing them in an unbounded buffer according to their delivery time, and (iv) finally delivering them at the scheduled time. The specification of the activity of the channel follows below:

- $Channel(\text{int } clock\_c, \text{list(list(int))}\ packet\_l; \text{int } timestamp) \triangleq$
  $<prepare\_packet?(timestamp), *>.$
  $DeliveryCheck(clock\_c, \text{insert}([clock\_c\ ++\text{gauss}(100, 7), timestamp], packet\_l))\ +$
  $<elapse\_tick, \infty_{1,1}>.DeliveryCheck(clock\_c, packet\_l)$

- $DeliveryCheck(\text{int } clock\_c, \text{list(list(int))}\ packet\_l;) \triangleq$
  if $((packet\_l\ != [\,]) \,\&\&\, (\text{first}(\text{first}(packet\_l)) <= clock\_c))$ then
  $<trans\_packet!(\text{first}(\text{tail}(\text{first}(packet\_l)))), \infty_{1,1}>.DeliveryCheck(clock\_c, \text{tail}(packet\_l))$
  else
  $Channel(clock\_c\ ++1, packet\_l)$

The receiver activity may be summarized as follows. The receiver is continuously waiting for audio packets. Upon reception, the packet is discarded if the value of its timestamp (which corresponds to the value of the sender's clock at the time the packet was sent) is smaller of the value of the receiver's clock. Every msec, the queue of accepted packets is checked in order to detect whether there is any packet to be played out. The specification of this activity is shown below:

- $Receiver(\text{int } clock\_r, \text{list(int)}\ packet\_l; \text{int } timestamp) \triangleq$
  $<trans\_packet?(timestamp), *>.$
  (if $(timestamp >= clock\_r)$ then

$PlayoutCheck(clock\_r, \text{insert}(timestamp, packet\_l))$

else

$PlayoutCheck(clock\_r, packet\_l))$ +

$<elapse\_tick, \infty_{1,1}>.PlayoutCheck(clock\_r, packet\_l)$

- $PlayoutCheck(\text{int } clock\_r, \text{list(int) } packet\_l;) \triangleq$

  if $((packet\_l \mathrel{!}= [\,]) \;\&\&\; (\text{first}(packet\_l) == clock\_r))$ then

  $<playout\_packet, \infty_{1,1}>.PlayoutCheck(clock\_r, \text{tail}(packet\_l))$

  else

  $Receiver(clock\_r \mathrel{++}1, packet\_l)$

10.5.3 Formal Description of Periodic Synchronization

In this section the activity of periodic synchronization is actually modeled. In fact, unlike the previous scenario, the mechanism of periodic synchronization mentioned above requires the sender's clock and the receiver's clock to get synchronized every second after the initial synchronization has taken place. A nice feature of EMPA_{vp} is that we do not have to start from scratch to develop the formal description of the mechanism, but we can exploit the model for the case of initial synchronization only. Moreover, since EMPA_{vp} supports compositional modeling, all we have to do in order to accommodate periodic synchronizations is to locally modify the already specified terms *Sender*, *Channel*, and *Receiver*.

In particular, as far as the modeling of the sender is concerned, track must be kept of the next synchronization time at the sending site which is now in charge of implementing the corresponding synchronization procedure accordingly. The specification reported below represents the above mentioned activity of the sender:

- $Sender(\text{int } clock\_s;) \triangleq$

  $PacketGen(clock\_s, clock\_s \mathrel{++}20, clock\_s \mathrel{++}1000)$

- $PacketGen(\text{int } clock\_s, \text{int } trans\_time, \text{int } synch\_time;) \triangleq$

  if $(clock\_s >= synch\_time)$ then

  $<prepare\_probe!(clock\_s), \infty_{2,1}>.PacketGen'(clock\_s \mathrel{++}1, clock\_s, trans\_time, synch\_time)$

  else

  if $(clock\_s >= trans\_time)$ then

  $<prepare\_packet!(clock\_s), \infty_{2,1}>.PacketGen(clock\_s \mathrel{++}1, trans\_time \mathrel{++}20, synch\_time)$

  else

  $<elapse\_tick, \infty_{2,1}>.PacketGen(clock\_s \mathrel{++}1, trans\_time, synch\_time)$

$PacketGen'(\text{int } clock\_s, \text{int } old\_clock\_s, \text{int } trans\_time, \text{int } synch\_time;) \triangleq$

if $(clock\_s >= trans\_time)$ then

$<prepare\_packet!(clock\_s), \infty_{2,1}>.$
$PacketGen'(clock\_s ++1, old\_clock\_s, trans\_time ++20, synch\_time)$
else
$(<trans\_response, *>.<prepare\_install!(clock\_s --old\_clock\_s), \infty_{2,1}>.$
$PacketGen''(clock\_s ++1, trans\_time, synch\_time) +$
$<elapse\_tick, \infty_{2,1}>.PacketGen'(clock\_s ++1, old\_clock\_s, trans\_time, synch\_time))$

$PacketGen''(\text{int } clock\_s, \text{int } trans\_time, \text{int } synch\_time;) \triangleq$
if $(clock\_s >= trans\_time)$ then
$<prepare\_packet!(clock\_s), \infty_{2,1}>.PacketGen''(clock\_s ++1, trans\_time ++20, synch\_time)$
else
$(<trans\_ack, *>.PacketGen(clock\_s ++1, trans\_time, synch\_time ++1000) +$
$<elapse\_tick, \infty_{2,1}>.PacketGen''(clock\_s ++1, trans\_time, synch\_time))$

The channel behaves as in the case of initial synchronization only, but in addition it must also manage probe, response, install, and acknowledgement messages due to periodic synchronization activities. To accomplish this, the model for the channel given in Sect. 10.5.2 may be simply reused for each of the above mentioned types of message, with in addition the value of the delivery time of the message stored in an appropriate variable. Thus, the formal specification of *Channel* is given below:

- $Channel(\text{int } clock\_c, \text{list(list(int))} \ packet\_l; \text{int } timestamp) \triangleq$
$<prepare\_packet?(timestamp), *>.$
$DeliveryCheck(clock\_c, \text{insert}([clock\_c ++\text{gauss}(100, 7), timestamp], packet\_l)) +$
$<prepare\_probe?(timestamp), *>.$
$DeliveryCheckProbe(clock\_c, packet\_l, timestamp, clock\_c ++\text{gauss}(100, 7)) +$
$<prepare\_response, *>.DeliveryCheckResponse(clock\_c, packet\_l, clock\_c ++\text{gauss}(100, 7)) +$
$<prepare\_install?(timestamp), *>.$
$DeliveryCheckInstall(clock\_c, packet\_l, timestamp, clock\_c ++\text{gauss}(100, 7)) +$
$<prepare\_ack, *>.DeliveryCheckAck(clock\_c, packet\_l, clock\_c ++\text{gauss}(100, 7)) +$
$<elapse\_tick, \infty_{2,1}>.DeliveryCheck(clock\_c, packet\_l)$

$ChannelProbe(\text{int } clock\_c, \text{list(list(int))} \ packet\_l, \text{int } timestamp\_synch, \text{int } delivery\_time; \text{int } timestamp) \triangleq$
if $(delivery\_time <= clock\_c)$ then
$<trans\_probe!(timestamp\_synch), \infty_{2,1}>.DeliveryCheck(clock\_c, packet\_l)$
else
$(<prepare\_packet?(timestamp), *>.$
$DeliveryCheckProbe(clock\_c, \text{insert}([clock\_c ++\text{gauss}(100, 7), timestamp], packet\_l),$
$timestamp\_synch), delivery\_time) +$

$<elapse\_tick, \infty_{2,1}>.DeliveryCheckProbe(clock\_c, packet\_l, timestamp\_synch, delivery\_time))$

$ChannelResponse(\text{int } clock\_c, \text{list(list(int))} \ packet\_l, \text{int } delivery\_time; \text{int } timestamp) \triangleq$

if $(delivery\_time <= clock\_c)$ then

$(<trans\_response, \infty_{2,1}>.DeliveryCheck(clock\_c, packet\_l) \ +$

$<prepare\_packet?(timestamp), *>.<trans\_response, \infty_{2,1}>.$

$DeliveryCheck(clock\_c, \text{insert}([clock\_c \ ++\text{gauss}(100, 7), timestamp], packet\_l)))$

else

$(<prepare\_packet?(timestamp), *>.$

$DeliveryCheckResponse(clock\_c, \text{insert}(timestamp, packet\_l),$

$\text{insert}([clock\_c{+}{+}\text{gauss}(100, 7), timestamp], packet\_l), delivery\_time) \ +$

$<elapse\_tick, \infty_{2,1}>.DeliveryCheckResponse(clock\_c, packet\_l, delivery\_time))$

$ChannelInstall(\text{int } clock\_c, \text{list(list(int))} \ packet\_l, \text{int } timestamp\_synch, \text{int } delivery\_time; \text{int } timestamp) \triangleq$

if $(delivery\_time <= clock\_c)$ then

$<trans\_install!(timestamp\_synch), \infty_{2,1}>.DeliveryCheck(clock\_c, packet\_l)$

else

$(<prepare\_packet?(timestamp), *>.$

$DeliveryCheckInstall(clock\_c, \text{insert}([clock\_c \ ++\text{gauss}(100, 7), timestamp], packet\_l),$

$timestamp\_synch), delivery\_time) \ +$

$<elapse\_tick, \infty_{2,1}>.DeliveryCheckInstall(clock\_c, packet\_l, timestamp\_synch, delivery\_time))$

$ChannelAck(\text{int } clock\_c, \text{list(list(int))} \ packet\_l, \text{int } delivery\_time; \text{int } timestamp) \triangleq$

if $(delivery\_time <= clock\_c)$ then

$(<trans\_ack, \infty_{2,1}>.DeliveryCheck(clock\_c, packet\_l) \ +$

$<prepare\_packet?(timestamp), *>.<trans\_response, \infty_{2,1}>.$

$DeliveryCheck(clock\_c, \text{insert}([clock\_c \ ++\text{gauss}(100, 7), timestamp], packet\_l)))$

else

$(<prepare\_packet?(timestamp), *>.$

$DeliveryCheckAck(clock\_c, \text{insert}(timestamp, packet\_l),$

$\text{insert}([clock\_c \ ++\text{gauss}(100, 7), timestamp], packet\_l), delivery\_time) \ +$

$<elapse\_tick, \infty_{2,1}>.DeliveryCheckAck(clock\_c, packet\_l, delivery\_time))$

- $DeliveryCheck(\text{int } clock\_c, \text{list(list(int))} \ packet\_l;) \triangleq$

if $((packet\_l \ != [\,]) \ \&\& \ (\text{first}(\text{first}(packet\_l)) <= clock\_c))$ then

$(<trans\_packet!(\text{first}(\text{tail}(\text{first}(packet\_l)))), \infty_{2,1}>.DeliveryCheck(clock\_c, \text{tail}(packet\_l)) \ +$

$<elapse\_tick, \infty_{1,1}>.Channel(clock\_c \ ++1, packet\_l))$

else

$Channel(clock\_c\ ++1, packet\_l)$

$DeliveryCheckProbe(\text{int } clock\_c, \text{list(list(int))}\ packet\_l, \text{int } timestamp\_synch, \text{real } delivery\_time;) \triangleq$

if $((packet\_l\ != [\,]) \,\&\&\, (\text{first(first}(packet\_l)) \leq clock\_c))$ then

$<trans\_packet!(\text{first(tail(first}(packet\_l)))), \infty_{2,1}>.$

$DeliveryCheckProbe(clock\_c, \text{tail}(packet\_l), timestamp\_synch, delivery\_time)$

else

$ChannelProbe(clock\_c\ ++1, packet\_l, timestamp\_synch, delivery\_time)$

$DeliveryCheckResponse(\text{int } clock\_c, \text{list(list(int))}\ packet\_l, \text{int } delivery\_time;) \triangleq$

if $((packet\_l\ != [\,]) \,\&\&\, (\text{first(first}(packet\_l)) <= clock\_c))$ then

$<trans\_packet!(\text{first(tail(first}(packet\_l)))), \infty_{2,1}>.$

$DeliveryCheckResponse(clock\_c, \text{tail}(packet\_l), delivery\_time)$

else

$ChannelResponse(clock\_c\ ++1, packet\_l, delivery\_time)$

$DeliveryCheckInstall(\text{int } clock\_c, \text{list(list(int))}\ packet\_l, \text{int } timestamp\_synch, \text{real } delivery\_time;) \triangleq$

if $((packet\_l\ != [\,]) \,\&\&\, (\text{first(first}(packet\_l)) \leq clock\_c))$ then

$<trans\_packet!(\text{first(tail(first}(packet\_l)))), \infty_{2,1}>.$

$DeliveryCheckInstall(clock\_c, \text{tail}(packet\_l), timestamp\_synch, delivery\_time)$

else

$ChannelInstall(clock\_c\ ++1, packet\_l, timestamp\_synch, delivery\_time)$

$DeliveryCheckAck(\text{int } clock\_c, \text{list(list(int))}\ packet\_l, \text{int } delivery\_time;) \triangleq$

if $((packet\_l\ != [\,]) \,\&\&\, (\text{first(first}(packet\_l)) <= clock\_c))$ then

$<trans\_packet!(\text{first(tail(first}(packet\_l)))), \infty_{2,1}>.$

$DeliveryCheckAck(clock\_c, \text{tail}(packet\_l), delivery\_time)$

else

$ChannelAck(clock\_c\ ++1, packet\_l, delivery\_time)$

Finally, in the last part of this section the formal modeling of the receiver is reported. Needless to say, also the receiver is involved in the synchronization procedure. From the receiver's viewpoint, the modeling of the periodic synchronization has a crucial difference w.r.t. the modeling of the initial synchronization. In particular, the value of the timestamp of the probe message of one of the subsequent synchronizations is not assigned to the receiver's clock but stored into a variable which is incremented each time the receiver's clock is incremented. The new value of the receiver's clock is then installed upon reception of the install message. At that time, all the audio packets in the receiver's queue whose timestamp is smaller than the new value

are discarded. The specification of the activity of the receiver is as follows:

- $Receiver(\text{int } clock\_r, \text{list(int) } packet\_l; \text{int } timestamp) \triangleq$

  $<trans\_probe?(timestamp), *>.<prepare\_response, \infty_{2,1}>.$

  $\quad PlayoutCheck'(clock\_r, timestamp, packet\_l)\ +$

  $<trans\_packet?(timestamp), *>.$

  $\quad (\text{if } (timestamp >= clock\_r) \text{ then}$

  $\quad\quad PlayoutCheck(clock\_r, \text{insert}(timestamp, packet\_l))$

  $\quad \text{else}$

  $\quad\quad PlayoutCheck(clock\_r, packet\_l))\ +$

  $<elapse\_tick, \infty_{2,1}>.PlayoutCheck(clock\_r, packet\_l)$

  $Receiver'(\text{int } clock\_r, \text{int } synch\_time, \text{list(int) } packet\_l; \text{int } timestamp) \triangleq$

  $<trans\_install?(timestamp), *>.<prepare\_ack, \infty_{2,1}>.$

  $\quad PlayoutCheck''(synch\_time --timestamp, packet\_l)\ +$

  $<trans\_packet?(timestamp), *>.$

  $\quad (\text{if } (timestamp >= clock\_r) \text{ then}$

  $\quad\quad PlayoutCheck'(clock\_r, synch\_time, \text{insert}(timestamp, packet\_l))$

  $\quad \text{else}$

  $\quad\quad PlayoutCheck'(clock\_r, synch\_time, packet\_l))\ +$

  $<elapse\_tick, \infty_{2,1}>.PlayoutCheck'(clock\_r, synch\_time, packet\_l)$

- $PlayoutCheck(\text{int } clock\_r, \text{list(int) } packet\_l;) \triangleq$

  $\text{if } ((packet\_l \text{ != } [\,]) \text{ \&\& } (\text{first}(packet\_l) == clock\_r)) \text{ then}$

  $\quad <playout\_packet, \infty_{2,1}>.PlayoutCheck(clock\_r, \text{tail}(packet\_l))$

  else

  $\quad Receiver(clock\_r ++1, packet\_l)$

  $PlayoutCheck'(\text{int } clock\_r, \text{int } synch\_time, \text{list(int) } packet\_l;) \triangleq$

  $\text{if } ((packet\_l \text{ != } [\,]) \text{ \&\& } (\text{first}(packet\_l) == clock\_r)) \text{ then}$

  $\quad <playout\_packet, \infty_{2,1}>.PlayoutCheck'(clock\_r, synch\_time, \text{tail}(packet\_l))$

  else

  $\quad Receiver'(clock\_r ++1, synch\_time ++1, packet\_l)$

  $PlayoutCheck''(\text{int } clock\_r, \text{list(int) } packet\_l;) \triangleq$

  $\text{if } ((packet\_l \text{ != } [\,]) \text{ \&\& } (\text{first}(packet\_l) <= clock\_r)) \text{ then}$

  $\quad PlayoutCheck''(clock\_r, \text{tail}(packet\_l))$

  else

$Receiver(clock\_r\ ++1, packet\_l)$

10.5.4 Performance Analysis

To analyze the performance of the two specifications above, we have measured the percentage of packets that are played out by conducting 10 experiments, each consisting of 20 simulation runs concerning a period of 30 sec. The following .tmt file has been provided:

$@\ elapse\_tick\ 30000\ 20$

$\#\ playout\_packet\_perc = (100\ **\ playout\_packet(1, `)\ //\ trans\_packet(1, `))$

The results are reported in Table 10.5, where also 90% confidence intervals are shown. As can be seen, about 25% of packets are discarded in case of initial synchronization only. These results revealed during the design of the audio mechanism the inadequacy of performing only an initial synchronization. The results also show that the percentage of discarded packets falls down 4% when performing a synchronization every second.

exp.	estimate	90% conf. int.	estimate	90% conf. int.
1	76.527147	[76.329880, 76.724415]	96.702746	[96.335040, 97.070452]
2	73.829673	[73.684253, 73.975093]	96.712566	[96.490167, 96.934966]
3	71.786939	[71.606600, 71.967279]	96.453736	[96.183720, 96.723751]
4	71.598548	[71.406887, 71.790209]	96.583136	[96.243839, 96.922433]
5	78.935170	[78.775455, 79.094885]	96.600180	[96.296200, 96.904160]
6	70.829697	[70.626774, 71.032620]	96.612087	[96.282160, 96.942014]
7	72.368819	[72.159914, 72.577724]	96.481827	[96.183756, 96.779897]
8	78.268155	[78.073999, 78.462310]	96.594055	[96.233550, 96.954561]
9	74.422198	[74.214966, 74.629431]	96.480821	[96.154645, 96.806997]
10	74.518115	[74.290436, 74.745794]	96.745196	[96.461490, 97.028903]

Table 10.5: Simulation results for *PacketizedAudio* in case of initial and periodic synchronization

A prototype implementation of the playout control mechanism has been carried out through the Unix socket interface and the datagram based UDP protocol. Using such a prototype implementation, an initial extensive experimentation has been conducted in order to test the real performances of the mechanism. In particular, several experiments of audio conversation have been carried out during daytime using a SUN workstation SPARC 5 situated at the Laboratory of Computer Science of Cesena (a remote site of the Computer Science Department of the University of Bologna) and a SUN workstation SPARC 10 situated at the CERN in Geneva (Switzerland).

A percentage of no more than 4-5% of loss audio packets was observed due to late arrivals. In order to test the efficacy of the proposed method, also contemporary experiments were conducted on the same testbed

without using any playout control mechanism. In those experiments, a percentage of lost audio packets (due to late arrivals) was experienced ranging from about 10% to almost 40% depending on the traffic conditions.

As can be noted, the results obtained from the simulative analysis of the formal model of the mechanism correctly predicted the actual performance of the mechanism itself measured on the field. This highlights the importance of using formal description techniques in the initial phase of the design of a system, in order to detect in advance errors or inefficiency which may result in cost increases if discovered too late.

10.6 Lehmann-Rabin Algorithm for Dining Philosophers

In this section we analyze the Lehmann-Rabin randomized distributed algorithm for the dining philosopher problem. The purpose is to prove the correctness of the algorithm, by verifying the absence of deadlock and the mutual exclusive use of resources, and to evaluate the mean number of philosophers who are simultaneously eating [23].

10.6.1 Informal Specification

The problem is characterized as follows. Suppose there are n philosophers P_i, $0 \leq i \leq n-1$, sitting at a round table each with a plate in front of him/her. Furthermore, let there also be n chopsticks C_i, $0 \leq i \leq n-1$, each shared by two neighbor philosophers and used to get the rice at the center of the table. Philosophers carry out two alternating activities: thinking and eating. Each philosopher must pick up both chopsticks before eating and must release them after eating. A philosopher may pick up only one chopstick at a time and cannot pick up a chopstick already in the hand of a neighbor. The time taken by a philosopher to eat is assumed to be finite. Given that philosophers can be thought of as concurrent processes sharing resources represented by the chopsticks, the mutual exclusive use of the chopsticks must be guaranteed in such a way that no adjacent philosophers are simultaneously eating and deadlock does not occur.

A fair and elegant solution to this problem has been proposed in [119]. The idea is that P_i flips a fair coin to choose between C_i and C_{i+1}, gets the chosen chopstick as soon as it becomes free, and gets the other chopstick if it is free. If the second chopstick is not available, the philosopher replaces the first one and repeats the procedure.

10.6.2 Formal Description with EMPA

If we denote by "$\_ +_n \_$" ("$\_ -_n \_$") the addition (subtraction) modulo n and let $think_i$ be the action type "P_i is thinking", eat_i be "P_i is eating", pu_i be "C_i is being picked up", and pd_i be "C_i is being put down", the algorithm above can be modeled as follows with EMPA:

$$
\begin{array}{rcl}
LehmannRabinDP_n & \equiv & (P_0 \parallel_\emptyset \ldots \parallel_\emptyset P_{n-1}) \parallel_{\{pu_i, pd_i \mid 0 \leq i \leq n-1\}} (C_0 \parallel_\emptyset \ldots \parallel_\emptyset C_{n-1}) \\
P_i & \triangleq & <think_i, \lambda_i>.P_i' \\
P_i' & \triangleq & <\tau, \infty_{1,1/2}>.<pu_i, \infty_{1,1}>.(<pu_{i+_n 1}, \infty_{2,1}>.P_i'' + \\
 & & \qquad <pd_i, \infty_{1,1}>.P_i') + \\
 & & <\tau, \infty_{1,1/2}>.<pu_{i+_n 1}, \infty_{1,1}>.(<pu_i, \infty_{2,1}>.P_i'' + \\
 & & \qquad <pd_{i+_n 1}, \infty_{1,1}>.P_i') \\
P_i'' & \triangleq & <eat_i, \mu_i>.<pd_i, \infty_{1,1}>.<pd_{i+_n 1}, \infty_{1,1}>.P_i \\
C_i & \triangleq & <pu_i, *>.<pd_i, *>.C_i
\end{array}
$$

where the τ actions correspond to coin flips with the weights reflecting the respective probabilistic outcomes. Observe that internal actions have been used to model the two outcomes of a coin flip as they cannot be involved in any interaction. It is worth noting that only actions with type $think_i$ or eat_i are relevant from the performance viewpoint and so they have been given an exponential timing, whereas the time it takes P_i to flip the coin or to pick up or put down a chopstick can be neglected so the corresponding actions are immediate. We also point out that priority levels of immediate actions have proved to be quite helpful in the description of the choice between getting the second chopstick and releasing the first one in order to model the fact that, whenever the second chopstick is available, the philosopher does pick it up. This is achieved in the subterms of term P_i' enclosed in parentheses by assigning priority level 2 to the action with type pu and priority level 1 to the action with type pd.

10.6.3 Functional Analysis

We address the correctness of the algorithm by making sure that no adjacent philosophers be simultaneously eating and deadlock be avoided. To verify these two functional properties, we have resorted to model checking by providing a .mu file with the following μ-calculus formulas:

$$
\begin{array}{rcl}
\hat{}no\_adjacent\_eating & = & \max x = \bigwedge_{i=0}^{n-1} \neg(\langle eat_i \rangle \mathrm{tt} \wedge (\langle eat_{i+_n 1} \rangle \mathrm{tt} \vee \langle eat_{i-_n 1} \rangle \mathrm{tt})) \wedge [-]x \\
\hat{}can\_deadlock & = & \min x = [-]\mathrm{ff} \vee \langle - \rangle x
\end{array}
$$

It has turned out that $LehmannRabinDP_n$ satisfies $no\_adjacent\_eating$ while it does not satisfy $can\_deadlock$, as required.

10.6.4 Performance Analysis

The mean number of philosophers who are simultaneously eating is a performance index reflecting the degree of parallelism of the algorithm. Such a number has been been obtained by summing up the stationary probabilities of every state of the underlying Markovian semantics (whose size is shown in Table 10.6) multiplied by the number of philosophers eating in that state, which is derived by assigning yield reward 1 to the action having type eat_i in term P_i''. To achieve this, the following .rew file has been provided:

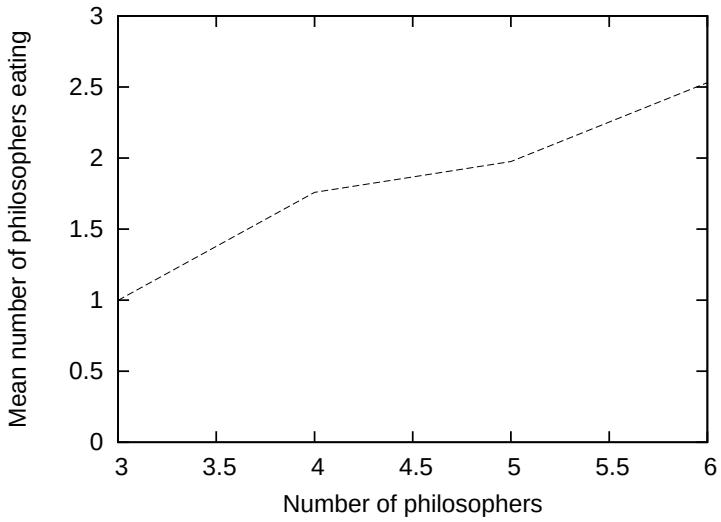


Figure 10.11: Mean number of philosophers simultaneously eating

$$\hat{}mean\_no\_eating\_phil \quad = \quad \begin{pmatrix} eat_0 & 1 & 0 \\ \dots & \dots & \dots \\ eat_{n-1} & 1 & 0 \end{pmatrix}$$

philosophers	states	transitions
3	13	30
4	35	112
5	81	305
6	199	924

Table 10.6: Size of $\mathcal{M}[\![LehmannRabinDP_n]\!]$

In Fig. 10.11 we report the results we have obtained by varying the number n of philosophers from 3 to 6, where $\lambda_i = 4$ and $\mu_i = 0.75$ for all $0 \leq i \leq n-1$. As expected, the mean number of philosophers who are simultaneously eating increases with n.

10.7 Mutual Exclusion Algorithms

The mutual exclusion problem is the problem of managing access to a single indivisible resource that can only support one user at a time. Alternatively, it can be viewed as the problem of ensuring that certain

portions of program code are executed within critical sections, where no two programs are permitted to be in critical sections at the same time.

In this section we consider six mutual exclusion algorithms taken from [124] and we compare their performance by evaluating the corresponding mean numbers of accesses per time unit to the critical section and to the shared control variables. The contribution of this case study is to show that EMPA constitutes a valid support to the analysis of the performance of distributed algorithms in the average case. This is important because in the literature only lower and upper bounds are usually provided.

10.7.1 Dijkstra Algorithm

This algorithm [65] makes use of two shared variables for controlling the access to the critical section. Variable *turn* (an integer in $\{1, \ldots, n\}$) indicates which program owns the turn to access the critical section, while $\mathit{flag}(i)$, $1 \leq i \leq n$, denotes the stage (an integer in $\{1, 2, 3\}$) of program i in accessing the critical section.

The Dijkstra algorithm works as follows. In the first stage, program i starts by setting $\mathit{flag}(i)$ to 1 and then repeatedly checks *turn* until it is equal to i. If not, and if the current owner of the turn is seen not to be currently active ($\mathit{flag}(\mathit{turn}) = 0$), program i sets *turn* to i. Once having seen $\mathit{turn} = i$, program i moves on to the second stage. In this stage, program i sets $\mathit{flag}(i)$ to 2 and then checks to see that no other program j has $\mathit{flag}(j) = 2$. If the check completes successfully, program i goes to its critical section, otherwise it returns to the first stage. Upon leaving the critical section, program i lowers $\mathit{flag}(i)$ to 0.

The Dijkstra algorithm can be modeled with EMPA as follows:

$$
\begin{array}{rcl}
\mathit{DijkstraME}_n & \triangleq & (\mathit{Program}_1 \parallel_\emptyset \ldots \parallel_\emptyset \mathit{Program}_n) \parallel_S \\
& & ((\mathit{Flag0}_1 \parallel_\emptyset \ldots \parallel_\emptyset \mathit{Flag0}_n) \parallel_R \\
& & \mathit{Turn1}) \\
S & = & \{\mathit{set\_flag\_to\_0}_i, \mathit{set\_flag\_to\_1}_i, \mathit{set\_flag\_to\_2}_i, \mathit{modify\_turn}_i, \\
& & \mathit{read\_turn\_eq\_i}, \mathit{turn\_eq\_i}, \mathit{turn\_neq\_i}, \\
& & \mathit{read\_flag\_turn\_eq\_0}_i, \mathit{flag\_turn\_eq\_0}_i, \mathit{flag\_turn\_neq\_0}_i, \\
& & \mathit{read\_flag\_eq\_2}_i, \mathit{flag\_eq\_2}_i, \mathit{flag\_neq\_2}_i \mid 1 \leq i \leq n\} \\
R & = & \{\mathit{read\_flag\_eq\_0}_i, \mathit{flag\_eq\_0}_i, \mathit{flag\_neq\_0}_i \mid 1 \leq i \leq n\}
\end{array}
$$

Let us denote by exec_i ($\mathit{exec\_cs}_i$) the action type describing the fact that $\mathit{Program}_i$ is executing outside (inside) the critical section. These actions are assumed to be exponentially timed with rate λ_i and δ_i, respectively. All the other actions related to reading or writing shared control variables are assumed to have the same duration for every program (the convention is that they are exponentially timed with rate 1).

$\mathit{Program}_i$ can be modeled as follows:

$$
\begin{array}{rcl}
\mathit{Program}_i & \triangleq & <\mathit{exec}_i, \lambda_i>.\mathit{SetFlag1}_i \\
\mathit{SetFlag1}_i & \triangleq & <\mathit{set\_flag\_to\_1}_i, 1>.\mathit{TestTurn}_i
\end{array}
$$

$$
\begin{aligned}
TestTurn_i &\triangleq <read\_turn\_eq\_i, 1>. \\
&\quad (<turn\_eq\_i, *>.SetFlag2_i + \\
&\quad <turn\_neq\_i, *>.TestFlag_i) \\
TestFlag_i &\triangleq <read\_flag\_turn\_eq\_0_i, 1>. \\
&\quad (<flag\_turn\_eq\_0_i, *>.<modify\_turn_i, 1>.SetFlag2_i + \\
&\quad <flag\_turn\_neq\_0_i, *>.TestTurn_i) \\
SetFlag2_i &\triangleq <set\_flag\_to\_2_i, 1>.TestFlag2_{\{1,\ldots,n\}-\{i\},i} \\
TestFlag2_{K,i} &\triangleq \sum_{k \in K} <read\_flag\_eq\_2_k, 1>. \\
&\quad (<flag\_eq\_2_k, *>.SetFlag1_i + \\
&\quad <flag\_neq\_2_k, *>.TestFlag2_{K-\{k\},i}),\ 1 < |K| < n \\
TestFlag2_{\{k\},i} &\triangleq <read\_flag\_eq\_2_k, 1>. \\
&\quad (<flag\_eq\_2_k, *>.SetFlag1_i + \\
&\quad <flag\_neq\_2_k, *>.CriticalSection_i),\ 1 \leq k \leq n \\
CriticalSection_i &\triangleq <exec\_cs_i, \delta_i>.<set\_flag\_to\_0_i, 1>.Program_i
\end{aligned}
$$

Flag s_i can be modeled as follows:

$$
\begin{aligned}
Flag0_i &\triangleq <set\_flag\_to\_1_i, *>.Flag1_i + \\
&\quad <read\_flag\_eq\_0_i, *>.<flag\_eq\_0_i, \infty_{1,1}>.Flag0_i + \\
&\quad <read\_flag\_eq\_2_i, *>.<flag\_neq\_2_i, \infty_{1,1}>.Flag0_i \\
Flag1_i &\triangleq <set\_flag\_to\_2_i, *>.Flag2_i + \\
&\quad <read\_flag\_eq\_0_i, *>.<flag\_neq\_0_i, \infty_{1,1}>.Flag1_i + \\
&\quad <read\_flag\_eq\_2_i, *>.<flag\_neq\_2_i, \infty_{1,1}>.Flag1_i \\
Flag2_i &\triangleq <set\_flag\_to\_0_i, *>.Flag0_i + \\
&\quad <set\_flag\_to\_1_i, *>.Flag1_i + \\
&\quad <read\_flag\_eq\_0_i, *>.<flag\_neq\_0_i, \infty_{1,1}>.Flag2_i + \\
&\quad <read\_flag\_eq\_2_i, *>.<flag\_eq\_2_i, \infty_{1,1}>.Flag2_i
\end{aligned}
$$

Note that testing shared control variables is modeled through two actions: reading and outcome (either true or false). The duration of the testing operation is associated with the former action, while the latter is described as immediate.

Finally, *Turn* i can be modeled as follows:

$$
Turn\ i \triangleq \sum_{j=1}^{n} <modify\_turn_j, *>.Turn\ j +
$$

$$\sum_{j=1}^{n} <read\_flag\_turn\_eq\_0_j, *>.<read\_flag\_eq\_0_i, \infty_{1,1}>.$$
$$(<flag\_eq\_0_i, *>.<flag\_turn\_eq\_0_j, \infty_{1,1}>.Turn\ i +$$
$$<flag\_neq\_0_i, *>.<flag\_turn\_neq\_0_j, \infty_{1,1}>.Turn\ i) +$$
$$<read\_turn\_eq\_i, *>.<turn\_eq\_i, \infty_{1,1}>.Turn\ i +$$
$$\sum_{j=1 \wedge j \neq i}^{n} <read\_turn\_eq\_j, *>.<turn\_neq\_j, \infty_{1,1}>.Turn\ i$$

10.7.2 Peterson Algorithm

This algorithm [155] is based on a series of $n-1$ competitions at levels $1, 2, \ldots, n-1$. At each successive competition, the algorithm ensures that there is at least one loser. Thus, all n programs may compete in the level 1 competition, but at most $n-1$ programs can win. In general, at most $n-k$ processes can win at level k. So at most one program can win at level $n-1$, which yields the mutual exclusion condition. This algorithm makes use of variables $turn(k)$ (integers in $\{1, \ldots, n\}$), indicating which program owns the turn to access the critical section at level $k \in \{1, \ldots, n-1\}$, and $flag(i)$, $1 \leq i \leq n$, denoting the stage (an integer in $\{0, \ldots, n-1\}$) of program i in accessing the critical section.

The Peterson algorithm works as follows. Program i engages in one competition for each level $k \in \{1, \ldots, n-1\}$ during which it sets $flag(i)$ to k and $turn(k)$ to i, then waits to discover either that for every other program j it holds $flag(j) < k$ or else that $turn(k) \neq i$. If and when this happens, which means either that no other program has reached level k or that $turn(k)$ has been reset by some other program since i most recently set it, program i enters its critical section. Upon leaving the critical section, program i lowers $flag(i)$ to 0.

The Peterson algorithm can be modeled with EMPA as follows:

$$\begin{aligned} PetersonME_n &\triangleq (Program_1 \parallel_\emptyset \ldots \parallel_\emptyset Program_n) \parallel_S \\ &\quad ((Flag0_1 \parallel_\emptyset \ldots \parallel_\emptyset Flag0_n) \parallel_\emptyset \\ &\quad (Turn_{1,\emptyset,\emptyset} \parallel_\emptyset \ldots \parallel_\emptyset Turn_{n-1,\emptyset,\emptyset})) \\ S &= \{set\_flag\_to\_s_i, set\_turn\_k\_to\_i, leave\_turn\_k_i, \\ &\quad read\_flag\_i\_lt\_k, flag\_i\_lt\_k, flag\_i\_ge\_k, \\ &\quad read\_turn\_k\_neq\_i, turn\_k\_neq\_i, turn\_k\_eq\_i \mid \\ &\qquad 1 \leq i \leq n \wedge 1 \leq k \leq n-1 \wedge 0 \leq s \leq n-1\} \end{aligned}$$

$Program_i$ can be modeled as follows:

$$\begin{aligned} Program_i &\triangleq <exec_i, \lambda_i>.SetFlag1_i \\ SetFlag\ k_i &\triangleq <set\_flag\_to\_k_i, 1>.SetTurn_{k,i},\ 1 \leq k \leq n-1 \\ SetFlag\ n_i &\triangleq <enter\_cs_i, \infty_{2,1}>.CriticalSection_i \end{aligned}$$

$$
\begin{aligned}
SetTurn_{k,i} &\triangleq <set\_turn\_k\_to\_i, 1>.CheckFlag_{k,\{1,\ldots,n\}-\{i\},i},\ 1 \leq k \leq n-1 \\
CheckFlag_{k,J,i} &\triangleq \sum_{j \in J} <read\_flag\_j\_lt\_k, 1>. \\
&\quad (<flag\_j\_lt\_k, *>.CheckFlag_{k,J-\{j\},i} + \\
&\quad\ <flag\_j\_ge\_k, *>.CheckTurn_{k,i}),\ 1 \leq k \leq n-1 \wedge 1 < |J| < n \\
CheckFlag_{k,\{j\},i} &\triangleq <read\_flag\_j\_lt\_k, 1>. \\
&\quad (<flag\_j\_lt\_k, *>.<leave\_turn\_k_i, \infty_{2,1}>.SetFlag\ k{+}1_{\,i} + \\
&\quad\ <flag\_j\_ge\_k, *>.CheckTurn_{k,i}),\ 1 \leq k \leq n-1 \wedge 1 \leq j \leq n \\
CheckTurn_{k,i} &\triangleq <read\_turn\_k\_neq\_i, 1>. \\
&\quad (<turn\_k\_neq\_i, *>.<leave\_turn\_k_i, \infty_{2,1}>.SetFlag\ k{+}1_{\,i} + \\
&\quad\ <turn\_k\_eq\_i, *>.CheckFlag_{k,\{1,\ldots,n\}-\{i\},i}),\ 1 \leq k \leq n-1 \\
CriticalSection_i &\triangleq <exec\_cs_i, \delta_i>.<set\_flag\_to\_0_{\,i}, 1>.Program_i
\end{aligned}
$$

Flag s_i can be modeled as follows:

$$
\begin{aligned}
Flag0_{\,i} &\triangleq <set\_flag\_to\_1_{\,i}, *>.Flag1_{\,i} + \\
&\quad \sum_{k=1}^{n-1} <read\_flag\_i\_lt\_k, *>.<flag\_i\_lt\_k, \infty_{1,1}>.Flag0_{\,i} \\
Flag\ s_i &\triangleq <set\_flag\_to\_s{+}1_{\,i}, *>.Flag\ s{+}1_{\,i} + \\
&\quad \sum_{k=s+1}^{n-1} <read\_flag\_i\_lt\_k, *>.<flag\_i\_lt\_k, \infty_{1,1}>.Flag\ s_i + \\
&\quad \sum_{k=1}^{s} <read\_flag\_i\_lt\_k, *>.<flag\_i\_ge\_k, \infty_{1,1}>.Flag\ s_i,\ 1 \leq s < n-1 \\
Flag\ n\text{-}1_{\,i} &\triangleq <set\_flag\_to\_0_{\,i}, *>.Flag0_{\,i} + \\
&\quad \sum_{k=1}^{n-1} <read\_flag\_i\_lt\_k, *>.<flag\_i\_ge\_k, \infty_{1,1}>.Flag\ n\text{-}1_{\,i}
\end{aligned}
$$

Finally $Turn_{k,I,J}$, where I is the set of programs i which have set $turn(k)$ to i and $J \subseteq I$ is the set containing the last program j which has set $turn(k)$ to j, can be modeled as follows:

$$
\begin{aligned}
Turn_{k,I,J} &\triangleq \sum_{j \notin I} <set\_turn\_k\_to\_j, *>.Turn_{k,I \cup \{j\},\{j\}} + \\
&\quad \sum_{j \in J} <read\_turn\_k\_neq\_j, *>.<turn\_k\_eq\_j, \infty_{1,1}>.Turn_{k,I,J} + \\
&\quad \sum_{j \in I-J} <read\_turn\_k\_neq\_j, *>.<turn\_k\_neq\_j, \infty_{1,1}>.Turn_{k,I,J} + \\
&\quad \sum_{j \in I-J} <leave\_turn\_k_j, *>.Turn_{k,I-\{j\},J} + \\
&\quad \sum_{j \in J} <leave\_turn\_k_j, *>.Turn_{k,I-\{j\},\emptyset}
\end{aligned}
$$

10.7.3 Tournament Algorithm

This algorithm [156] is similar to the previous one with the difference that competitions are arranged as in a tournament. Let us assume that the number n of programs is a power of 2 and let us number the programs starting with 0. Then, each program engages in a series of $\log n$ competitions in order to access the critical section. We can think of a binary tournament tree, where the n leaves correspond left-to-right to the n programs.

In order to name the various competitions, the roles played by the programs in the competitions, and the sets of opponents that programs can have in competitions, for $0 \leq i \leq n-1$ and $1 \leq k \leq \log n$ we define the following notions. The level k competition of program i $comp(i,k)$ is the string consisting of the high order $\log n - k$ bits of the binary representation of i. The role of program i in its level k competition $role(i,k)$ is the $(\log n - k + 1)$st bit of the binary representation of i. The opponents of program i in its level k competition $opp(i,k)$ is the set of program indices with the same high order $\log n - k$ bits as i and the opposite $(\log n - k + 1)$st bit. This algorithm makes use of variables $turn(x)$ (integers in $\{0,1\}$), indicating which program owns the turn to access the critical section among those programs for which the binary string x is a prefix of the binary representation of the indices, and $flag(i)$, $0 \leq i \leq n-1$, denoting the stage (an integer in $\{0,\ldots,\log n\}$) of program i in accessing the critical section.

The tournament algorithm works as follows. Program i engages in one competition for each level $k \in \{1,\ldots,\log n\}$ during which it sets $flag(i)$ to k and $turn(comp(i,k))$ to $role(i,k)$, then waits to discover either that for every other program $j \in opp(i,k)$ it holds $flag(j) < k$ or else that $turn(comp(i,k)) \neq role(i,k)$. If and when this happens, which means either that no other opponent program has reached level k or that $turn(comp(i,k))$ has been reset by some other program since i most recently set it, program i enters its critical section. Upon leaving the critical section, program i lowers $flag(i)$ to 0.

The tournament algorithm can be modeled with EMPA as follows:

$$
\begin{aligned}
TournamentME_n \;\triangleq\;& (Program_0 \parallel_\emptyset \ldots \parallel_\emptyset Program_{n-1}) \parallel_S \\
& ((Flag0_0 \parallel_\emptyset \ldots \parallel_\emptyset Flag0_{n-1}) \parallel_\emptyset \\
& (Turn_1 \parallel_\emptyset \ldots \parallel_\emptyset Turn_{\log n})) \\
S \;=\;& \{set\_flag\_to\_s_i, set\_turn_{comp(i,k),role(i,k),i}, leave\_turn_i, \\
& read\_flag\_i\_lt\_k, flag\_i\_lt\_k, flag\_i\_ge\_k, \\
& read\_turn_{comp(i,k),role(i,k),i}, turn\_neq\_role_i, turn\_eq\_role_i \mid \\
& 0 \leq i \leq n-1 \wedge 1 \leq k \leq \log n \wedge 0 \leq s \leq \log n\}
\end{aligned}
$$

$Program_i$ can be modeled as follows:

$$
\begin{aligned}
Program_i \;&\triangleq\; <exec_i, \lambda_i>.SetFlag1_i \\
SetFlag\,k_i \;&\triangleq\; <set\_flag\_to\_k_i, 1>.SetTurn_{k,i},\ 1 \leq k \leq \log n \\
SetFlag\,log\,n+1_i \;&\triangleq\; <enter\_cs_i, \infty_{2,1}>.CriticalSection_i \\
SetTurn_{k,i} \;&\triangleq\; <set\_turn_{comp(i,k),role(i,k),i}, 1>.CheckFlag_{k,opp(i,k),i},\ 1 \leq k \leq \log n
\end{aligned}
$$

$$
\begin{aligned}
CheckFlag_{k,J,i} &\triangleq \sum_{j \in J} <read\_flag\_j\_lt\_k, 1>. \\
&\quad (<flag\_j\_lt\_k, *>.CheckFlag_{k,J-\{j\},i} + \\
&\quad <flag\_j\_ge\_k, *>.CheckTurn_{k,i}),\ 1 \leq k \leq \log n \wedge 1 < |J| < n \\
CheckFlag_{k,\{j\},i} &\triangleq <read\_flag\_j\_lt\_k, 1>. \\
&\quad (<flag\_j\_lt\_k, *>.<leave\_turn_i, \infty_{2,1}>.SetFlag\ k{+}1_{\,i} + \\
&\quad <flag\_j\_ge\_k, *>.CheckTurn_{k,i}),\ 1 \leq k \leq \log n \wedge 0 \leq j \leq n-1 \\
CheckTurn_{k,i} &\triangleq <read\_turn_{comp(i,k),role(i,k),i}, 1>. \\
&\quad (<turn\_neq\_role_i, *>.<leave\_turn_i, \infty_{2,1}>.SetFlag\ k{+}1_{\,i} + \\
&\quad <turn\_eq\_role_i, *>.CheckFlag_{k,opp(i,k),i}),\ 1 \leq k \leq \log n \\
CriticalSection_i &\triangleq <exec\_cs_i, \delta_i>.<set\_flag\_to\_0_{\,i}, 1>.Program_i
\end{aligned}
$$

Flag s_i can be modeled as follows:

$$
\begin{aligned}
Flag0_{\,i} &\triangleq <set\_flag\_to\_1_{\,i}, *>.Flag1_{\,i} + \\
&\quad \sum_{k=1}^{\log n} <read\_flag\_i\_lt\_k, *>.<flag\_i\_lt\_k, \infty_{1,1}>.Flag0_{\,i} \\
Flag\ s_i &\triangleq <set\_flag\_to\_s{+}1_{\,i}, *>.Flag\ s{+}1_{\,i} + \\
&\quad \sum_{k=s+1}^{\log n} <read\_flag\_i\_lt\_k, *>.<flag\_i\_lt\_k, \infty_{1,1}>.Flag\ s_i + \\
&\quad \sum_{k=1}^{s} <read\_flag\_i\_lt\_k, *>.<flag\_i\_ge\_k, \infty_{1,1}>.Flag\ s_i,\ 1 \leq s < \log n \\
Flag\ log\ n_i &\triangleq <set\_flag\_to\_0_{\,i}, *>.Flag0_{\,i} + \\
&\quad \sum_{k=1}^{\log n} <read\_flag\_i\_lt\_k, *>.<flag\_i\_ge\_k, \infty_{1,1}>.Flag\ log\ n_i
\end{aligned}
$$

Finally, $Turn_k$ can be modeled as follows (let j', j'' be such that $comp(j', k) \neq comp(j'', k)$):

$$
\begin{aligned}
Turn_k &\triangleq Turn_{k,\emptyset,\emptyset,comp(j',k)} \parallel_\emptyset \ldots \parallel_\emptyset Turn_{k,\emptyset,\emptyset,comp(j'',k)},\ 1 \leq k < \log n \\
Turn_{\log n} &\triangleq Turn_{\log n,\emptyset,\emptyset,comp(i,\log n)}
\end{aligned}
$$

where for $J \subseteq I$ (let j be such that $comp(j, k) = comp(i, k)$):

$$
\begin{aligned}
Turn_{k,I,J,comp(i,k)} \triangleq\ & \sum_{j \notin I} <set\_turn_{comp(j,k),role(j,k),j}, *>.Turn_{k,I\cup\{j\},\{j\},comp(j,k)} + \\
& \sum_{j \in J} <read\_turn_{comp(j,k),role(j,k),j}, *>.<turn\_eq\_role_j, \infty_{1,1}>.Turn_{k,I,J,comp(i,k)} + \\
& \sum_{j \in I-J} <read\_turn_{comp(j,k),role(j,k),j}, *>.<turn\_neq\_role_j, \infty_{1,1}>.Turn_{k,I,J,comp(i,k)} + \\
& \sum_{j \in I-J} <leave\_turn_j, *>.Turn_{k,I-\{j\},J,comp(i,k)} + \\
& \sum_{j \in J} <leave\_turn_j, *>.Turn_{k,I-\{j\},\emptyset,comp(i,k)}
\end{aligned}
$$

10.7.4 Lamport Algorithm

This algorithm [116] makes use of variables x and y (integers in $\{1,\ldots,n\}$ and $\{0,\ldots,n\}$, respectively), implementing via software the effect of the atomic test-and-set operation based on the sequence write$_x$-read$_y$-write$_y$-read$_x$, and $\mathit{flag}(i)$, $1 \leq i \leq n$, denoting the stage (an integer in $\{0,1\}$) of program i in accessing the critical section.

The Lamport algorithm works as follows. Program i starts with setting $\mathit{flag}(i)$ to 1 and x to i, then it reads y. If $y \neq 0$ then program i resets $\mathit{flag}(i)$, waits for y to be reset, and returns to the beginning. If $y = 0$ then program i sets y to i and reads x. If $x = i$ then program i accesses its critical section, otherwise it resets $\mathit{flag}(i)$, waits for $\mathit{flag}(j)$ to be reset for every other $j \in \{1,\ldots,n\}$, and reads y. If $y = i$ then program i accesses its critical section, otherwise it waits for y to be reset and returns to the beginning. Upon leaving the critical section, program i resets y and $\mathit{flag}(i)$.

The Lamport algorithm can be modeled with EMPA as follows:

$$\begin{aligned}
\mathit{LamportME}_n \;\triangleq\;& (\mathit{Program}_1 \parallel_\emptyset \ldots \parallel_\emptyset \mathit{Program}_n) \parallel_S \\
& ((\mathit{Flag0}_1 \parallel_\emptyset \ldots \parallel_\emptyset \mathit{Flag0}_n) \parallel_\emptyset \\
& X1 \parallel_\emptyset Y0) \\
S \;=\;& \{\mathit{set_flag_to_0}_i, \mathit{set_flag_to_1}_i, \\
& \mathit{set_x_to_i}, \mathit{set_y_to_i}, \mathit{set_y_to_0}_i, \\
& \mathit{read_y_eq_0}_i, \mathit{y_eq_0}_i, \mathit{y_neq_0}_i, \\
& \mathit{wait_y_eq_0}_i, \mathit{now_y_eq_0}_i, \mathit{now_y_neq_0}_i, \\
& \mathit{read_x_eq_i}, \mathit{x_eq_i}, \mathit{x_neq_i}, \\
& \mathit{wait_flag_j_eq_0}_i, \mathit{flag_j_eq_0}_i, \mathit{flag_j_neq_0}_i, \\
& \mathit{read_y_eq_i}, \mathit{y_eq_i}, \mathit{y_neq_i} \mid 1 \leq i,j \leq n \wedge i \neq j\}
\end{aligned}$$

$\mathit{Program}_i$ can be modeled as follows:

$$\begin{aligned}
\mathit{Program}_i \;\triangleq\;& <\mathit{exec}_i, \lambda_i>.\mathit{Start}_i \\
\mathit{Start}_i \;\triangleq\;& <\mathit{set_flag_to_1}_i, 1>.<\mathit{set_x_to_i}, 1>.\mathit{FirstCheckY}_i \\
\mathit{FirstCheckY}_i \;\triangleq\;& <\mathit{read_y_eq_0}_i, 1>. \\
& \quad (<\mathit{y_neq_0}_i, *>.<\mathit{set_flag_to_0}_i, 1>.\mathit{WaitState}_i + \\
& \quad\; <\mathit{y_eq_0}_i, *>.<\mathit{set_y_to_i}, 1>.\mathit{CheckX}_i) \\
\mathit{WaitState}_i \;\triangleq\;& <\mathit{wait_y_eq_0}_i, 1>. \\
& \quad (<\mathit{now_y_neq_0}_i, *>.\mathit{WaitState}_i + \\
& \quad\; <\mathit{now_y_eq_0}_i, *>.\mathit{Start}_i) \\
\mathit{CheckX}_i \;\triangleq\;& <\mathit{read_x_eq_i}, 1>. \\
& \quad (<\mathit{x_neq_i}, *>.<\mathit{set_flag_to_0}_i, 1>.\mathit{CheckFlag}_{\{1,\ldots,n\}-\{i\},i} + \\
& \quad\; <\mathit{x_eq_i}, *>.\mathit{CriticalSection}_i)
\end{aligned}$$

$$
\begin{aligned}
CheckFlag_{K,i} &\triangleq \sum_{k\in K} <wait\_flag\_k\_eq\_0_i, 1>. \\
&\quad (<flag\_k\_neq\_0_i, *>.WaitFlag_{K,k,i} + \\
&\quad\ <flag\_k\_eq\_0_i, *>.CheckFlag_{K-\{k\},i}),\ 1 < |K| < n \\
CheckFlag_{\{k\},i} &\triangleq <wait\_flag\_k\_eq\_0_i, 1>. \\
&\quad (<flag\_k\_neq\_0_i, *>.CheckFlag_{\{k\},i} + \\
&\quad\ <flag\_k\_eq\_0_i, *>.SecondCheckY_i),\ 1 \leq k \leq n \\
WaitFlag_{K,k,i} &\triangleq <wait\_flag\_k\_eq\_0_i, 1>. \\
&\quad (<flag\_k\_neq\_0_i, *>.WaitFlag_{K,k,i} + \\
&\quad\ <flag\_k\_eq\_0_i, *>.CheckFlag_{K-\{k\},i}),\ 1 < |K| < n \wedge 1 \leq k \leq n \\
SecondCheckY_i &\triangleq <read\_y\_eq\_i, 1>. \\
&\quad (<y\_neq\_i, *>.WaitState_i + \\
&\quad\ <y\_eq\_i, *>.<set\_flag\_to\_1_i, 1>.CriticalSection_i) \\
CriticalSection_i &\triangleq <exec\_cs_i, \delta_i>.<set\_y\_to\_0_i, 1>.<set\_flag\_to\_0_i, 1>.Program_i
\end{aligned}
$$

$Flag\ s_i$ can be modeled as follows:

$$
\begin{aligned}
Flag0_i &\triangleq <set\_flag\_to\_1_i, *>.Flag1_i + \\
&\quad \sum_{j=1\wedge j\neq i}^{n} <wait\_flag\_i\_eq\_0_j, *>.<flag\_i\_eq\_0_j, \infty_{1,1}>.Flag0_i \\
Flag1_i &\triangleq <set\_flag\_to\_0_i, *>.Flag0_i + \\
&\quad \sum_{j=1\wedge j\neq i}^{n} <wait\_flag\_i\_eq\_0_j, *>.<flag\_i\_neq\_0_j, \infty_{1,1}>.Flag1_i
\end{aligned}
$$

$X\ i$ can be modeled as follows:

$$
\begin{aligned}
X\ i &\triangleq \sum_{j=1}^{n} <set\_x\_to\_j, *>.X\ j + \\
&\quad <read\_x\_eq\_i, *>.<x\_eq\_i, \infty_{1,1}>.X\ i + \\
&\quad \sum_{j=1\wedge j\neq i}^{n} <read\_x\_eq\_j, *>.<x\_neq\_j, \infty_{1,1}>.X\ i
\end{aligned}
$$

Finally, $Y\ s$ can be modeled as follows:

$$
\begin{aligned}
Y0 &\triangleq \sum_{j=1}^{n} <set\_y\_to\_j, *>.Y\ j + \\
&\quad \sum_{j=1}^{n} <read\_y\_eq\_j, *>.<y\_neq\_j, \infty_{1,1}>.Y0 + \\
&\quad \sum_{j=1}^{n} <read\_y\_eq\_0_j, *>.<y\_eq\_0_j, \infty_{1,1}>.Y0 + \\
&\quad \sum_{j=1}^{n} <wait\_y\_eq\_0_j, *>.<now\_y\_eq\_0_j, \infty_{1,1}>.Y0
\end{aligned}
$$

$$
\begin{aligned}
Y\,i \;\triangleq\;& \sum_{j=1}^{n} <set\_y\_to\_0_j, *>.Y0 + \\
& \sum_{j=1}^{n} <set\_y\_to\_j, *>.Y\,j + \\
& <read\_y\_eq\_i, *>.<y\_eq\_i, \infty_{1,1}>.Y\,i + \\
& \sum_{j=1}^{n} <read\_y\_eq\_0_j, *>.<y\_neq\_0_j, \infty_{1,1}>.Y\,i + \\
& \sum_{j=1 \wedge j \neq i}^{n} <read\_y\_eq\_j, *>.<y\_neq\_j, \infty_{1,1}>.Y\,i + \\
& \sum_{j=1}^{n} <wait\_y\_eq\_0_j, *>.<now\_y\_neq\_0_j, \infty_{1,1}>.Y\,i,\ 1 \leq i \leq n
\end{aligned}
$$

10.7.5 Burns Algorithm

This algorithm [42] does not require any variable (such as *turn*, x, and y) which is a multiwriter register often difficult to implement, but makes only use of $flag(i)$, $1 \leq i \leq n$, denoting the stage (an integer in $\{0, 1\}$) of program i in appropriate checks carried out before accessing the critical section.

The Burns algorithm works as follows. Three tests are executed by program i after setting $flag(i)$ to 0 and before accessing the critical section. The first two tests check that $flag(j) = 0$ for every $j < i$, while the third test checks that $flag(j) = 0$ for every $j > i$. If program i passes the first test, it sets $flag(i)$ to 1 and proceeds with the second test otherwise it resets $flag(i)$ and restarts the first test. If program i passes the second test, it proceeds with the third test (which is executed until it is passed) otherwise it returns to the first test after resetting $flag(i)$. When program i passes all the tests, it proceeds to its critical section. Upon leaving the critical section, program i lowers $flag(i)$ to 0.

The Burns algorithm can be modeled with EMPA as follows:

$$
\begin{aligned}
BurnsME_n \;\triangleq\;& (Program_1 \parallel_\emptyset \ldots \parallel_\emptyset Program_n) \parallel_S \\
& (Flag0_1 \parallel_\emptyset \ldots \parallel_\emptyset Flag0_n) \\
S \;=\;& \{set\_flag\_to\_0_i, set\_flag\_to\_1_i, \\
& \quad read\_flag\_i\_eq\_0_v, flag\_i\_eq\_0_v, flag\_i\_neq\_0_v \mid 1 \leq i \leq n \wedge 1 \leq v \leq 3\}
\end{aligned}
$$

$Program_i$ can be modeled as follows:

$$
\begin{aligned}
Program_i \;\triangleq\;& <exec_i, \lambda_i>.SetFlag0_i \\
SetFlag0_i \;\triangleq\;& <set\_flag\_to\_0_i, 1>.CheckFlag_{\{1,\ldots,i-1\},1,i} \\
SetFlag1_i \;\triangleq\;& <set\_flag\_to\_1_i, 1>.CheckFlag_{\{1,\ldots,i-1\},2,i} \\
CheckFlag_{K,v,i} \;\triangleq\;& \sum_{k \in K} <read\_flag\_k\_eq\_0_v, 1>. \\
& \quad (<flag\_k\_eq\_0_v, *>.CheckFlag_{K-\{k\},v,i} + \\
& \quad <flag\_k\_neq\_0_v, *>.SetFlag0_i),\ 1 < |K| \leq i-1 \wedge 1 \leq v \leq 2
\end{aligned}
$$

$$
\begin{aligned}
CheckFlag_{\{k\},1,i} &\triangleq <read\_flag\_k\_eq\_0_1, 1>. \\
&\quad (<flag\_k\_eq\_0_1, *>.SetFlag1_i + \\
&\quad\ <flag\_k\_neq\_0_1, *>.SetFlag0_i),\ 1 \leq k \leq i-1 \\
CheckFlag_{\{k\},2,i} &\triangleq <read\_flag\_k\_eq\_0_2, 1>. \\
&\quad (<flag\_k\_eq\_0_2, *>.CheckFlag_{\{i+1,\ldots,n\},3,i} + \\
&\quad\ <flag\_k\_neq\_0_2, *>.SetFlag0_i),\ 1 \leq k \leq i-1 \\
CheckFlag_{K,3,i} &\triangleq \sum_{k \in K} <read\_flag\_k\_eq\_0_3, 1>. \\
&\quad (<flag\_k\_eq\_0_3, *>.CheckFlag_{K-\{k\},3,i} + \\
&\quad\ <flag\_k\_neq\_0_3, *>.CheckFlag_{\{i+1,\ldots,n\},3}, i),\ 1 < |K| \leq n-i \\
CheckFlag_{\{k\},3,i} &\triangleq <read\_flag\_k\_eq\_0_3, 1>. \\
&\quad (<flag\_k\_eq\_0_3, *>.CriticalSection_i + \\
&\quad\ <flag\_k\_neq\_0_3, *>.CheckFlag_{\{i+1,\ldots,n\},3,i}),\ i+1 \leq k \leq n \\
CriticalSection_i &\triangleq <exec\_cs_i, \delta_i>.<set\_flag\_to\_0_i, 1>.Program_i
\end{aligned}
$$

Finally, *Flag* s_i can be modeled as follows:

$$
\begin{aligned}
Flag0_i &\triangleq <set\_flag\_to\_0_i, *>.Flag0_i + \\
&\quad <set\_flag\_to\_1_i, *>.Flag1_i + \\
&\quad <read\_flag\_i\_eq\_0_1, *>.<flag\_i\_eq\_0_1, \infty_{1,1}>.Flag0_i + \\
&\quad <read\_flag\_i\_eq\_0_2, *>.<flag\_i\_eq\_0_2, \infty_{1,1}>.Flag0_i + \\
&\quad <read\_flag\_i\_eq\_0_3, *>.<flag\_i\_eq\_0_3, \infty_{1,1}>.Flag0_i \\
Flag1_i &\triangleq <set\_flag\_to\_0_i, *>.Flag0_i + \\
&\quad <read\_flag\_i\_eq\_0_1, *>.<flag\_i\_neq\_0_1, \infty_{1,1}>.Flag1_i + \\
&\quad <read\_flag\_i\_eq\_0_2, *>.<flag\_i\_neq\_0_2, \infty_{1,1}>.Flag1_i + \\
&\quad <read\_flag\_i\_eq\_0_3, *>.<flag\_i\_neq\_0_3, \infty_{1,1}>.Flag1_i
\end{aligned}
$$

10.7.6 Ticket Algorithm

To guarantee a fair access to the critical section, programs could maintain a queue of program indices, initially empty, in a shared variable. A program wishing to access the critical section adds its index to the end of the queue, enters the critical section when finding itself at the beginning of the queue, and deletes itself from the queue when leaving the critical section.

The same type of FIFO behavior can be more efficiently achieved based on the idea of issuing tickets to the critical section [67]. In this case, only one shared variable holding a pair $(next, granted)$, with values in $\{0, \ldots, n-1\}$, is used. The *next* component represents the next ticket to the critical section that is to be

issued to a program, while the *granted* component represents the last ticket that has been granted permission to enter the critical section.

The ticket algorithm works as follows. When program i wants to access the critical section, it takes a ticket, i.e. it copies and increment the *next* component, and waits until such a ticket is equal to the *granted* component. When program i leaves the critical section, it increments the *granted* component modulo n.

The ticket algorithm can be modeled with EMPA as follows:

$$\begin{aligned}
TicketME_n &\triangleq ((Program_0 \parallel_\emptyset \ldots \parallel_\emptyset Program_{n-1}) \parallel_R \\
&\quad (Ticket_{0,null} \parallel_\emptyset \ldots \parallel_\emptyset Ticket_{n-1,null})) \parallel_S \\
&\quad (Next0 \parallel_\emptyset \\
&\quad Granted0) \\
S &= \{increment_granted, \\
&\quad read_next_i, return_next_{v,i}, \\
&\quad read_granted_eq_ticket_i_v, granted_eq_ticket_i, granted_neq_ticket_i \mid 0 \leq i, v \leq n-1\} \\
R &= \{read_ticket_i_eq_granted, ticket_i_eq_granted, ticket_i_neq_granted, \\
&\quad set_ticket_i_to_null, take_ticket_i \mid 0 \leq i \leq n-1\}
\end{aligned}$$

$Program_i$ can be modeled as follows:

$$\begin{aligned}
Program_i &\triangleq <exec_i, \lambda_i>.<take_ticket_i, \infty_{1,1}>.TestTicket_i \\
TestTicket_i &\triangleq <read_ticket_i_eq_granted, 1>. \\
&\quad (<ticket_i_eq_granted, *>.CriticalSection_i + \\
&\quad <ticket_i_neq_granted, *>.TestTicket_i) \\
CriticalSection_i &\triangleq <exec_cs, \delta_i>.<increment_granted, 1>.<set_ticket_i_to_null, \infty_{1,1}>.Program_i
\end{aligned}$$

$Ticket_{i,v}$ can be modeled as follows:

$$\begin{aligned}
Ticket_{i,null} &\triangleq <take_ticket_i, *>.<read_next_i, 1>. \sum_{v=0}^{n-1} <return_next_{v,i}, *>.Ticket_{i,v} \\
Ticket_{i,v} &\triangleq <set_ticket_i_to_null, *>.Ticket_{i,null} + \\
&\quad <read_ticket_i_eq_granted, *>.<read_granted_eq_ticket_i_v, 1>. \\
&\quad (<granted_eq_ticket_i, *>.<ticket_i_eq_granted, \infty_{1,1}>.Ticket_{i,v} + \\
&\quad <granted_neq_ticket_i, *>.<ticket_i_neq_granted, \infty_{1,1}>.Ticket_{i,v}),\ 0 \leq v \leq n-1
\end{aligned}$$

Next i can be modeled as follows (where "$\_ +_n \_$" is the addition modulo n):

$$Next\ i \triangleq \sum_{j=0}^{n-1} <read\_next_j, *>.<return\_next_{i,j}, \infty_{1,1}>.Next\ i +_n 1$$

Finally, *Granted i* can be modeled as follows:

$$\begin{aligned} \mathit{Granted}\ i \ \triangleq\ & <\mathit{increment\_granted}, *>.\mathit{Granted}\ i +_n 1 + \\ & \sum_{j=0}^{n-1} <\mathit{read\_granted\_eq\_ticket\_j}_i, *>.<\mathit{granted\_eq\_ticket\_j}, \infty_{1,1}>.\mathit{Granted}\ i + \\ & \sum_{j=0}^{n-1} \sum_{v=0 \wedge v \neq i}^{n-1} <\mathit{read\_granted\_eq\_ticket\_j}_v, *>.<\mathit{granted\_neq\_ticket\_j}, \infty_{1,1}>.\mathit{Granted}\ i \end{aligned}$$

10.7.7 Performance Analysis

We now compare the performance of the six mutual exclusion algorithms we have modeled. Before doing that, we have preliminarily investigated their correctness via model checking by providing the following .mu file:

$$\hat{}\,\mathit{mutual\_exclusion} \ =\ \max x = \neg(\bigvee_{i,j=1 \wedge i \neq j}^{n} \langle \mathit{exec\_cs}_i \rangle \mathrm{tt} \wedge \langle \mathit{exec\_cs}_j \rangle \mathrm{tt}) \wedge [-]x$$

The process algebraic model of each of the six algorithms satisfies *mutual_exclusion*.

The performance measures we are interested in are the mean numbers of accesses per time unit to the critical section and to the shared variables. They are computed on the Markovian semantic model of each algorithm; the size of such models in the case of two programs is shown in Table 10.7. The former performance index represents the throughput of the algorithm and has been specified by assigning bonus reward 1 to every action with type $\mathit{exec\_cs}_i$. The latter performance index represents instead the delay introduced by the algorithm in order to guarantee the mutual exclusive access to the critical section and has been specified by assigning bonus reward 1 to every action related to reading or writing shared control variables. As an example, for $\mathit{DijkstraME}_2$ the following .rew file has been provided:

algorithm	states	transitions
$\mathit{DjikstraME}_2$	148	296
$\mathit{PetersonME}_2$	49	98
$\mathit{TournamentME}_2$	49	98
$\mathit{LamportME}_2$	346	692
$\mathit{BurnsME}_2$	44	88
$\mathit{TicketME}_2$	72	144

Table 10.7: Size of the Markovian semantic models of the six mutual exclusion algorithms

$$
\begin{array}{lcll}
\hat{}critical\_section\_accesses & = & (& exec\_cs_1 \quad 0 \quad 1 \\
 & & & exec\_cs_2 \quad 0 \quad 1 \quad) \\
\hat{}shared\_variable\_accesses & = & (& set\_flag\_to\_0_1 \quad 0 \quad 1 \\
 & & & set\_flag\_to\_1_1 \quad 0 \quad 1 \\
 & & & set\_flag\_to\_2_1 \quad 0 \quad 1 \\
 & & & read\_turn\_eq\_1 \quad 0 \quad 1 \\
 & & & read\_flag\_turn\_eq\_0_1 \quad 0 \quad 1 \\
 & & & modify\_turn_1 \quad 0 \quad 1 \\
 & & & read\_flag\_eq\_2_1 \quad 0 \quad 1 \\
 & & & set\_flag\_to\_0_2 \quad 0 \quad 1 \\
 & & & set\_flag\_to\_1_2 \quad 0 \quad 1 \\
 & & & set\_flag\_to\_2_2 \quad 0 \quad 1 \\
 & & & read\_turn\_eq\_2 \quad 0 \quad 1 \\
 & & & read\_flag\_turn\_eq\_0_2 \quad 0 \quad 1 \\
 & & & modify\_turn_2 \quad 0 \quad 1 \\
 & & & read\_flag\_eq\_2_2 \quad 0 \quad 1 \quad)
\end{array}
$$

We report in Fig. 10.12 and 10.13 the curves concerning the mean number of accesses per time unit to the critical section and to the shared variables, respectively, for each of the six algorithms in the case of two programs. The curves are plotted for different values of the ratio $\delta_i^{-1}/\lambda_i^{-1}$ of the average time spent inside the critical section to the average time spent outside the critical section. Such values are those obtained by letting $\delta_i = 0.04$ (corresponding to 25 time units) and varying λ_i from 0.1 (10 time units) to 0.02 (50 time units). The two figures show that the throughput and the delay of each algorithm decrease as the above mentioned ratio decreases, i.e. as the frequency with which programs want to access the critical section decreases. Moreover, the two figures confirms that the Peterson algorithm and the tournament algorithm behave the same if two programs only are involved. Finally, the figures indicate that, in a scenario with only two programs, the six algorithms have comparable performance, with the ticket algorithm achieving the higher throughput and introducing the lower delay. Note that such a comparison is not based on lower or upper bounds for the performance of the algorithms, as usually happens, but on average values, so that it provides additional information to choose the most appropriate algorithm for a specific case.

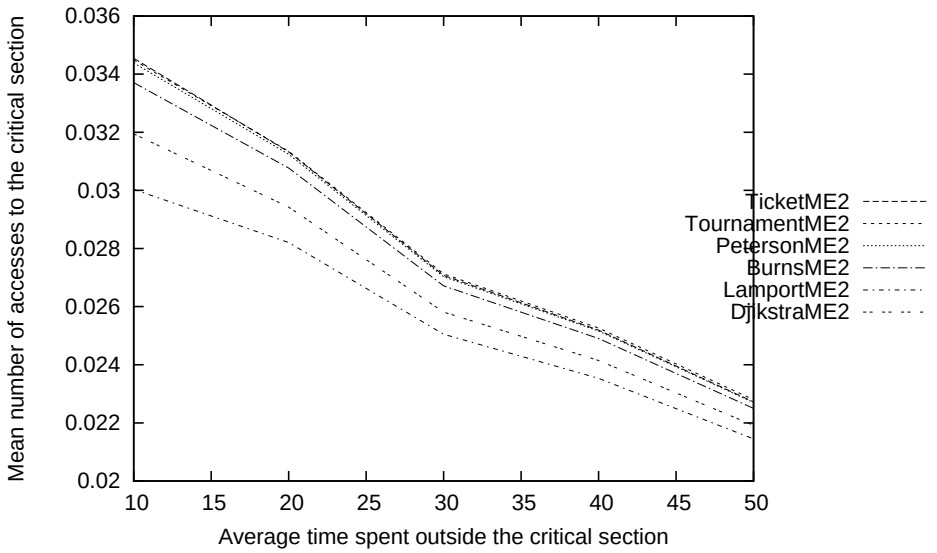


Figure 10.12: Mean number of accesses per time unit to the critical section

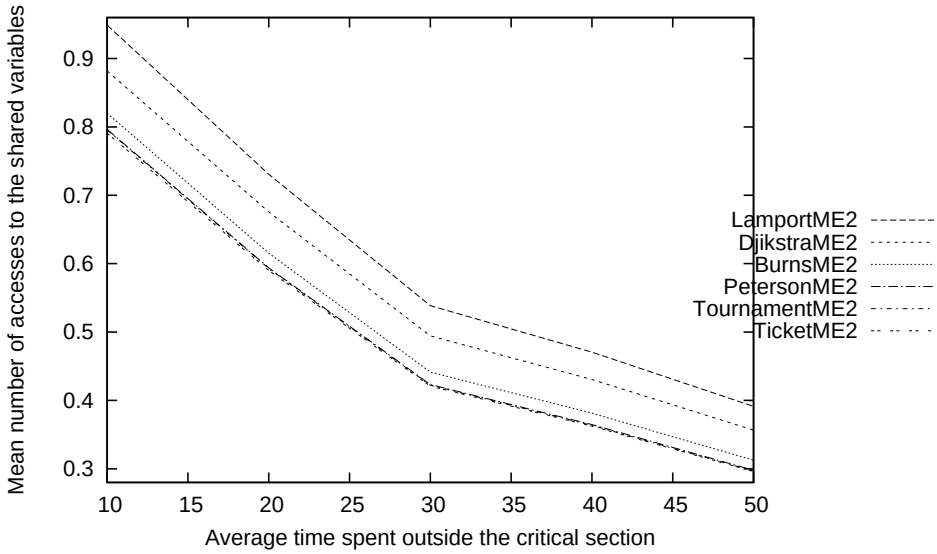


Figure 10.13: Mean number of accesses per time unit to the shared variables

Chapter 11

Conclusion

In this thesis we have proposed a formal integrated approach to modeling and analyzing functional and performance properties of concurrent systems during their design which profits from the complementary advantages of stochastically timed process algebras and stochastically timed Petri nets. In order to implement the integrated approach in the Markovian case, we have developed a new stochastically timed process algebra called EMPA as well as a software tool called TwoTowers that compiles EMPA descriptions of concurrent systems into their underlying semantic models and exploits already existing analysis tools to study such models.

This chapter concludes the thesis by making some comparisons with related work and outlining future research in the field of stochastically timed process algebras.

11.1 Related Work

The idea underlying the integrated approach of Fig. 1.1 comes from [145], where complementary views of concurrent systems, each one describing the systems at a different level of abstraction, are brought together in one uniform framework by establishing the appropriate semantic links. This realizes the stepwise development of complex systems through various levels of abstraction, which is good practice in software and hardware design. In this thesis we have extended the proposal of [145] with two orthogonal integrations. The former allows for the timely consideration of performance aspects during system design, thereby solving the major problem tackled in this thesis. The latter permits reusing already existing analysis tools, so that the approach can be implemented (as it has been done with TwoTowers) by constructing a compiler for algebraic terms which yields their underlying semantic models and by interfacing such a compiler with the analysis tools for the semantic models.

The development of EMPA, instead, has been influenced by the stochastically timed process algebras MTIPP [75] and PEPA [100] and by the formalism of GSPNs [3]. While designing EMPA, much emphasis

has been placed on expressiveness, as recognized in Chap. 4 where we have compared the four EMPA kernels with other process algebras appeared in the literature.

In EMPA action durations are mainly given by means of exponential distributions like in MTIPP and PEPA, but it is also possible to express immediate actions each of which is assigned a priority level and a weight like GSPN immediate transitions. Using exponentially timed actions only results in continuous time models, while using immediate actions only results in discrete time models. When both of them are used, immediate actions permit to model logical events as well as activities that are irrelevant from the performance viewpoint, thereby providing a mechanism for performance abstraction in the same way as action type τ provides a mechanism for functional abstraction. Furthermore, immediate actions allow to model systems whose activities may have different priorities and can be used to explicitly describe probabilistic choices avoiding the need of a new operator. Finally, the interplay of exponentially timed and immediate actions makes it possible, though not atomically, the description of activities whose durations follow or can be approximated by phase type distributions.

EMPA is also endowed with passive actions somewhat different from those of MTIPP and PEPA. Passive actions play a prominent role in EMPA, because they allow for nondeterministic choices and are essential in the synchronization discipline on action rates, since it requires that at most one active action is involved similarly to I/O automata [125]. Our master-slaves synchronization discipline, whose expressiveness is in most cases comparable to that of MTIPP and PEPA, leads to a more intuitive and modular treatment of the interactions among processes.

Because of the coexistence of different kinds of actions, the resulting expressive power of EMPA is considerable. Basically, it can be viewed as the union of a classical process algebra, a prioritized process algebra, a probabilistic process algebra, and an exponentially timed process algebra. On the other hand, this has required a great care in the definition of the integrated interleaving semantics (reflected by the use of functions *Melt*, *Select*, and *Norm* and the related computation of all the potential moves of a term at once), in the definition of the Markovian semantics (because of the possible coexistence of exponentially timed and immediate transitions), and in the definition of the integrated net semantics (witnessed by the handling of marking dependent rates).

Despite of the notable expressive power of EMPA, the notion of integrated equivalence $\sim_{\mathrm{EMB}}$ has been set up by assembling complementary proposals [117, 92, 100, 41, 184, 133] in an elegant and compact way, achieving the congruence property as far as a restricted form of nondeterminism is considered. Moreover, a new theoretical result has been established: $\sim_{\mathrm{EMB}}$ is the coarsest congruence contained in $\sim_{\mathrm{FP}}$ for a large class of terms, thereby emphasizing the necessity (beside the convenience) of defining a notion of equivalence directly on the integrated semantic model.

Finally, we observe that the development in this thesis of the integrated label oriented net semantics, the algebraic treatment of rewards for the high level specification of performance measures with the related performance measure sensitive integrated equivalence, and the symbolic handling of value passing terms

based on variable name independent semantic rules producing compact, improved versions of the symbolic models proposed in [81, 122, 121] are completely new in the field of (stochastically timed) process algebras. Unlike [159], mobility issues have not been addressed.

11.2 Future Research

We now conclude the thesis by outlining some relevant open problems left for future research, whose solution should lead to the development of a methodology for modeling and analyzing computer, communication and software systems that achieves an acceptable balance among formality, expressivity, usability, and efficiency. The first two problems affect EMPA only, while the others are concerned with stochastically timed process algebras in general.

1. As demonstrated in Chap. 5, the integrated equivalence $\sim_{\mathrm{EMB}}$ is a congruence only for those terms of EMPA exhibiting a restricted form of nondeterminism. This shows that nondeterminism, priority, probability, and exponentially distributed time fit well together in EMPA as long as the nondeterminism is confined to the choice among passive actions of different types or, under certain conditions, of the same type. Despite the practical relevance of such terms as demonstrated by the case studies presented in this thesis, because of the semantical nature of the characterization of the above mentioned conditions it is desirable to investigate a scenario in which the integrated equivalence is a congruence for the whole language. One possibility may be that of producing a generative-reactive (according to the terminology of [71]) variant of EMPA [37]. Exploiting the asymmetric form of synchronization between active actions and passive actions, the idea would be that of turning nondeterministic passive actions into reactive prioritized-probabilistic passive actions, whose rate is denoted by $*_{l,w}$. According to the reactive approach, the priority levels and the weights associated with passive actions would be used only to choose among passive actions of the same type, thus confining nondeterminism to the choice among passive actions of different types and avoiding discrepancies with the way rate normalization works. This hybrid generative-reactive approach would be similar to that used in [189] for probabilistic I/O automata and would be combined with an approach to modeling probabilistic choices among processes similar to that used in a probabilistic variant of ACP [7] in order to allow processes with different advancing speeds to be described in a purely probabilistic setting. Moreover this approach would encompass that of [100] as long as asymmetric synchronizations are concerned.

2. Abstraction mechanisms based on algebraic operators and suitable equivalences are, together with formality and compositionality, one of the major benefits of using stochastically timed process algebras w.r.t. notations traditionally adopted in the field of performance evaluation such as queueing networks and stochastically timed Petri nets. Unfortunately, the integrated equivalence $\sim_{\mathrm{EMB}}$ is strong, which means that, as far as terms where both exponentially timed and immediate actions coexist are concerned, it does not abstract from internal immediate actions, i.e. those actions which are not observable

and take no time. From the state space reduction standpoint, it would be profitable to define in our expressive framework an integrated equivalence which does abstract from those actions, as done for some extensions of MTIPP [165, 94, 93, 89]. Similarly, it would be advantageous defining an equivalence which abstracts from internal exponentially timed actions, as done for a sublanguage of PEPA [100].

3. As far as the calculation of performance measures is concerned, the problem of state space explosion must be tackled. Obviously, well known methods, such as generating the state space underlying a process term in a stepwise fashion along the structure imposed by the occurrences of the parallel composition operator and minimizing it at every step according to some congruence (possibly exploiting efficient data structures [95]), or operating at the syntactical level using the axioms of some congruence as rewriting rules, can be exploited. However, based on the compositional structure of process terms, aggregation and decomposition techniques leading to an efficient solution of the underlying MCs, as well as parallel and distributed simulation techniques, should be devised. Work in this direction can be found in [41, 167, 103, 175, 76, 134, 130, 185, 84, 128, 104, 183, 105, 32].

4. Similarly to the preorders for deterministically timed process algebras [135, 140, 52, 58, 109], a notion of Markovian preorder should be developed which sorts functionally equivalent terms according to their performance. In order to define such a Markovian preorder, it is necessary to understand whether it has to be measure specific or not as well as whether it refers to stationary or transient measures. Moreover, it has to be investigated whether it is convenient to define the preorder in the bisimulation style or e.g. according to the testing approach [63, 80]. As recognized in [102], an application of such a preorder may be the approximation of a term with another term amenable to efficient solution in the case in which the original term cannot be replaced with a suitable equivalent term. This would allow bounds on the performance of the original term to be efficiently derived. A study based on the testing approach can be found in [22].

5. The Markovian semantics is defined only for performance closed terms, i.e. for terms which do not admit nondeterminism. Unfortunately, it might be the case that, especially in the very early stages of system design, relevant quantities are unknown or known only with some approximation. Therefore, techniques should be developed which allow bounds on the performance measures to be derived also for underspecified systems. This can be done by extracting Markov decision processes [64] from terms that are not performance closed and computing bounds on such processes as proposed in [61].

6. The label oriented approach to the definition of net semantics outperforms the location oriented one w.r.t. the size of the resulting nets. However, the current formulation of the label oriented net semantics requires an additional processing step to eliminate unecessary inhibitor and contextual arcs which complicate the structure of the resulting nets. In order to avoid the introduction of additional places with inhibitor and contextual arcs altogether, one possibility may be that of suitably decorating action types within the sequential terms associated with net places in order to keep track of the occurrences

of static operators, without falling in the excess of information of the location oriented approach [20]. Furthermore, the definition of the net semantics should be extended to value passing terms.

7. In order for a formal description technique to be helpful in practice, its usability should be considered to ease the task of the designer. In this respect, it may be convenient to take the view of software architecture [176], which has emerged in the last decade as a discipline within software engineering to cope with the increasing size and complexity of software systems during the early stage of their development. To achieve this, the focus is turned from algorithmic and data structure related issues to the overall architecture of the system, meant to be a collection of computational components together with a description of their connectors, i.e. the interactions between these components. From the stochastically timed process algebra perspective, it is desirable to force the designer to model systems in a way that elucidates the basic architectural concepts of component and connector. Following [5], the distinction between dynamic and static operators can be exploited to enforce this controlled way of modeling. More precisely, the (easier) dynamic operators should be employed when defining the behavior of components and connectors. Instead, the (more difficult) static operators should be employed to define the semantics and the related static architectural checks, so they are transparent to the designer. Moreover, mechanisms for the hierarchical modeling of components and connectors should be provided and a companion graphical notation consistent with the concepts above should be devised; a good candidate here may be the flow graph model of [133]. A study on the development of a stochastically timed process algebra based architectural description language can be found in [21].

8. Does the integrated approach of Fig. 1.1 scale to generally distributed durations? In this thesis we have seen that the combined use of exponentially timed and immediate actions allows us to represent or approximate many frequently occurring probability distributions through phase type distributions. Moreover, we have observed that adding value passing to a process algebra with interleaving semantics allows systems with generally distributed durations to be dealt with, so from a modeling standpoint it is not strictly necessary to include actions which can be directly given arbitrarily distributed durations. However, from the designer viewpoint it may be advantageous to be able to directly and naturally express generally distributed durations and possibly analyze the related systems through numerical techniques. Probably, this is the most challenging open problem because we can no longer exploit the memoryless property of exponential distributions, which has allowed us to obtain MCs as performance models and to define the integrated semantics in the interleaving style. From a foundational point of view, this means that activities can no more be thought of as being started in the states where they are terminated. Therefore, it is necessary to understand how to define the semantics for stochastically timed process algebras with generally distributed durations and what performance models underlie process terms. To cope with this, several proposals have recently appeared [74, 77, 4, 40, 98, 160, 60, 38] which are quite different from one another. Among such proposals, we cite those stochastically timed process algebras with generally distributed durations for which a notion of equivalence (which is peculiar of

the process algebra approach) has been developed. CCS+ [77] solves the problem of identifying the start and the termination of an activity at the syntactic level by means of suitable operators which represent the random setting of a timer and the expiration of a timer, respectively. Semantic models are infinite LTSs from which performance measures can be derived via simulation. SPADES [60] solves the problem above in a similar way but the underlying semantic models are given by stochastic automata equipped with clocks. Performance measures can be obtained by simulating such automata. GSMPA [38] solves instead the problem of identifying the start and the termination of an activity at the semantic level through the ST approach [72] in the style of [1]. Additionally, the underlying performance models are explicitly defined to be generalized semi Markov processes (GSMPs) [127], which can be analyzed not only via simulation but also numerically. More precisely, if insensitivity conditions are met then exponential distributions can be substituted for the general distributions occurring in a GSMP thus obtaining an equivalent MC provided that expected values are preserved. Furthermore, even if insensitivity cannot be exploited, actions with a generally distributed duration can be approximated by terms where only exponentially timed actions occur in such a way that a suitable phase type distribution is represented; this action refinement process is naturally supported by the ST semantics. We conclude by noting that introducing generally distributed durations should hopefully lead to the ultimate integration: a uniform framework for deterministically timed process algebras [143], used for real time systems, and stochastically timed process algebras, used for performance evaluation purposes.

Appendix A

Proofs of Results

Proof of Prop. 4.1: It follows immediately from the fact that, for $E \in \mathcal{G}_{\mathrm{nd}}$, function *Norm* always evaluates to $*$, function *Select* boils down to the identity function, and the application of function *Melt* is irrelevant because the effect of $* \mathit{Aggr} * = *$ stems from the fact that $\longrightarrow$ is a relation hence it does not keep track of the multiplicity of its elements. ■

Proof of Prop. 4.2: It follows immediately from the fact that, for $E \in \mathcal{G}_{\mathrm{pt},w}$, function *Norm* has the same effect as side condition $(\tilde{\lambda}_1 = * \wedge \tilde{\gamma} = \tilde{\lambda}_2) \vee (\tilde{\lambda}_2 = * \wedge \tilde{\gamma} = \tilde{\lambda}_1)$ and the application of function *Melt* is irrelevant, because we are interested in action types only. ■

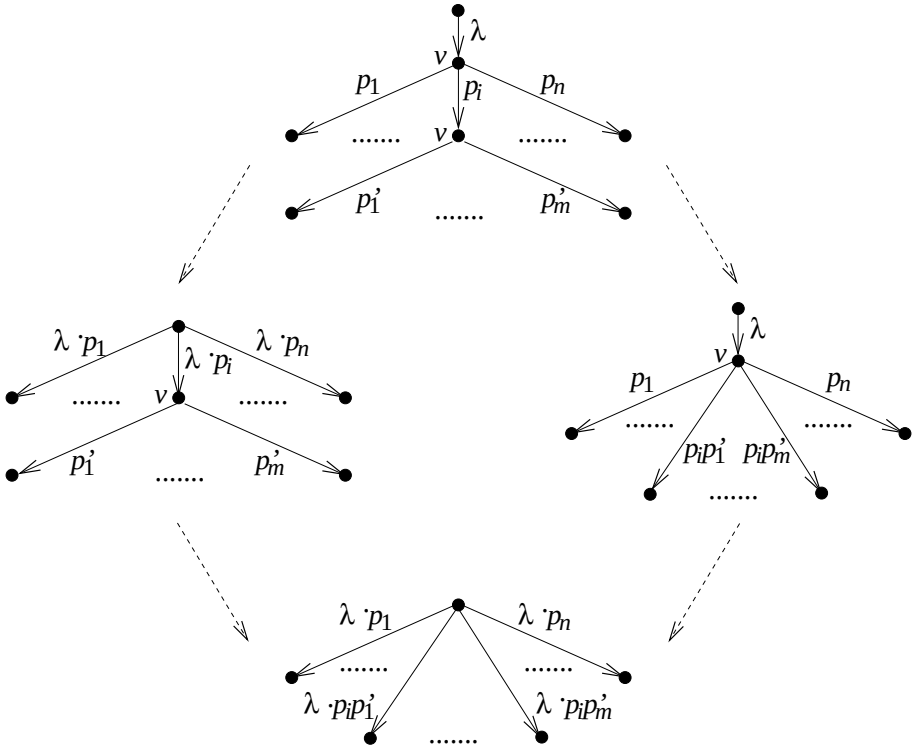
Proof of Prop. 4.3: It is a straightforward consequence of the fact that, for $E \in \mathcal{G}_{\mathrm{pb},l}$, function *Select* boils down to the identity function. ■

Proof of Prop. 4.4: It is a straightforward consequence of the fact that, for $E \in \mathcal{G}_{\mathrm{et}}$, function *Select* boils down to the identity function. ■

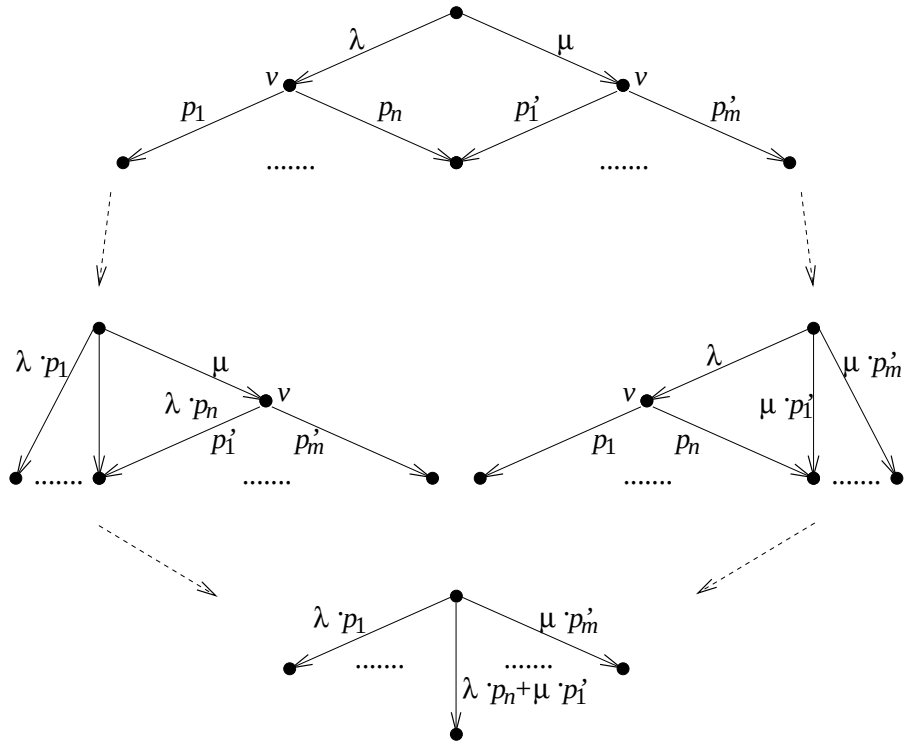
Proof of Thm. 4.1: Let $E \in \mathcal{E}$ be such that $\mathcal{I}[\![E]\!]$ contains both exponentially timed and immediate transitions.

(i) We proceed by induction on $k \in \mathbb{N}_+$:

- If $k = 1$ then the result immediately follows from the definition of $\longrightarrow_{E,1}$.
- Let $k > 1$ and let the result hold for $k - 1$. Suppose that the fork considered at step k is the one depicted in Fig. 4.2 and let $s \in S_{E,k}$:



(a)



(b)

Figure A.1: Confluence of the graph reduction rule in absence of immediate selfloops

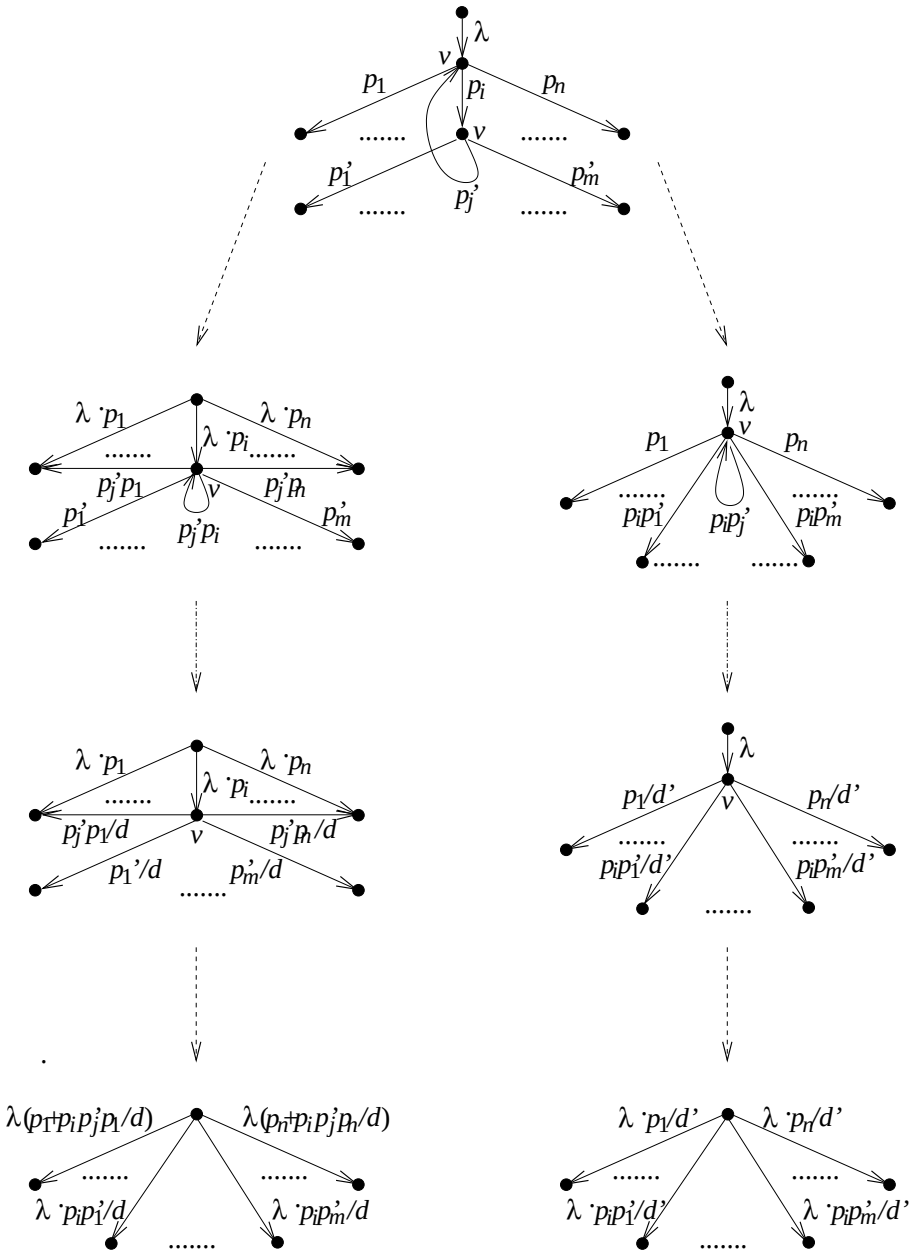


Figure A.2: Confluence of the graph reduction rule in presence of immediate selfloops

* If $s \notin \{s' \in S_{E,k-1} \mid s' \xrightarrow{p_0}_{E,k-1} s_0 \wedge H_{E,k}(s') = v\}$ then either s is tangible (hence the result is not concerned with it) or s is vanishing but none of its immediate transitions enters s_0, so the result holds by the induction hypothesis or, if it is downstream the fork, by the normalization performed at step k.
* Let $s \in \{s' \in S_{E,k-1} \mid s' \xrightarrow{p_0}_{E,k-1} s_0 \wedge H_{E,k}(s') = v\}$. If s is downstream the fork, then the result trivially follows by the normalization carried out at step k. Assume that s is not downstream the fork. From the induction hypothesis it follows that

$$\begin{aligned}
&\sum\{\!\!\{\, p \mid s \xrightarrow{p}_{E,k} s' \,\}\!\!\} = \\
&\qquad \sum\{\!\!\{\, p \mid s \xrightarrow{p}_{E,k} s' \wedge s' \not\equiv s_0 \,\}\!\!\} + \sum\{\!\!\{\, p_0 \cdot p_i \mid s \xrightarrow{p_0 \cdot p_i}_{E,k} s_i \,\}\!\!\} = \\
&\qquad \sum\{\!\!\{\, p \mid s \xrightarrow{p}_{E,k-1} s' \wedge s' \not\equiv s_0 \,\}\!\!\} + p_0 \cdot \sum\{\!\!\{\, p_i \mid s_0 \xrightarrow{p_i}_{E,k-1} s_i \,\}\!\!\} = \\
&\qquad \sum\{\!\!\{\, p \mid s \xrightarrow{p}_{E,k-1} s' \wedge s' \not\equiv s_0 \,\}\!\!\} + p_0 = \\
&\qquad \sum\{\!\!\{\, p \mid s \xrightarrow{p}_{E,k-1} s' \,\}\!\!\} = 1
\end{aligned}$$

(*ii*) We proceed by induction on $k \in \mathbb{N}_+$:

- If $k = 1$ then the result immediately follows from the definition of $P_{E,1}$.
- Let $k > 1$ and let the result hold for $k - 1$. Suppose that the fork considered at step k is the one depicted in Fig. 4.2. From the induction hypothesis and (*i*) it follows that

$$\begin{aligned}
&\textstyle\sum_{s \in S_{E,k}} P_{E,k}(s) = \\
&\qquad \textstyle\sum_{s \in S_{E,k} - \{s_i \mid 1 \le i \le n\}} P_{E,k-1}(s) + \sum_{1 \le i \le n} (P_{E,k-1}(s_i) + P_{E,k-1}(s_0) \cdot p_i) = \\
&\qquad \textstyle\sum_{s \in S_{E,k} - \{s_i \mid 1 \le i \le n\}} P_{E,k-1}(s) + \sum_{1 \le i \le n} P_{E,k-1}(s_i) + P_{E,k-1}(s_0) \cdot \sum_{1 \le i \le n} p_i = \\
&\qquad \textstyle\sum_{s \in S_{E,k}} P_{E,k-1}(s) + P_{E,k-1}(s_0) = \\
&\qquad \textstyle\sum_{s \in S_{E,k-1}} P_{E,k-1}(s) = 1
\end{aligned}$$

(*iii*) Let us modify the fork of immediate transitions depicted in Fig. 4.2 by assuming that s_0 has also an immediate selfloop labeled with q, where $\sum_{i=1}^{n} p_i + q = 1$ due to (*i*). Let us unfold the immediate selfloop by introducing the set of states $\{s_{0,j} \mid j \in \mathbb{N}_+\}$ such that:

- the immediate selfloop is replaced by a transition labeled with q from s_0 to $s_{0,1}$;
- for all $j \in \mathbb{N}_+$, $s_{0,j}$ has a transition labeled with p_i reaching s_i and a transition labeled with q reaching $s_{0,j+1}$.

Starting from s_0, the probability of reaching $s_{0,j}$ after j transition executions is q^j, while the probability of reaching s_i within j transition executions is $\sum_{h=0}^{j-1} p_i \cdot q^h$. As j grows, these probabilities approach 0 and $p_i/(1-q) = p_i / \sum_{r=1}^{n} p_r$, respectively.

(*iv*) The uniqueness of $\mathcal{M}[\![E]\!]$ stems from the confluence of the graph reduction rule in Fig. 4.2. To prove confluence, we proceed by induction on the length of the longest cycle of immediate transitions in $\mathcal{I}[\![E]\!]$.

– If the length of the longest cycle of immediate transitions is $c \leq 1$, then the first step eliminates all the cycles of immediate transitions (if any). In this case, at each step no immediate selfloop arises, thus making unnecessary the possible normalization of execution probabilities at states downstream the fork. Given two forks of immediate transitions, there are three cases:

  * There exists a state downstream a fork and upstream the other fork. Fig. A.1(a) shows that confluence holds in this case.
  * There exists a state downstream both forks. Fig. A.1(b) shows that confluence holds in this case as well.
  * There is no state shared by the two forks. In such a case, it is obvious that the order in which the two forks are considered is irrelevant.

– Suppose that the length of the longest cycle of immediate transitions is $c \geq 2$ and assume that the result holds whenever the length of the longest cycle of immediate transitions is $< c$. Consider the application of the graph reduction rule to one of the states in the cycle:

  * If no immediate selfloop arises, the confluence is preserved by this step as shown above.
  * If an immediate selfloop arises, the confluence is still preserved by this step as shown in Fig. A.2. In fact, by exploiting (i), it turns out that

$$d = \sum_{1 \leq h \leq n \wedge h \neq i} p'_j \cdot p_h + \sum_{1 \leq r \leq m \wedge r \neq j} p'_r =$$

$$\sum_{1 \leq h \leq n} p'_j \cdot p_h - p'_j \cdot p_i + \sum_{1 \leq r \leq m \wedge r \neq j} p'_r =$$

$$p'_j - p'_j \cdot p_i + \sum_{1 \leq r \leq m \wedge r \neq j} p'_r =$$

$$\sum_{1 \leq r \leq m} p'_r - p'_j \cdot p_i = 1 - p'_j \cdot p_i$$

and

$$d' = \sum_{1 \leq h \leq n \wedge h \neq i} p_h + \sum_{1 \leq r \leq m \wedge r \neq j} p_i \cdot p'_r =$$

$$\sum_{1 \leq h \leq n} p_h - p_i + \sum_{1 \leq r \leq m} p_i \cdot p'_r - p_i \cdot p'_j =$$

$$1 - p_i + p_i - p_i \cdot p'_j = 1 - p_i \cdot p'_j$$

and for each $h = 1, \ldots, n$ such that $h \neq i$

$$p_h + p_i \cdot p'_j \cdot p_h / d = p_h(1 + p_i \cdot p'_j / (1 - p'_j \cdot p_i)) =$$
$$p_h(1 - p'_j \cdot p_i + p_i \cdot p'_j)/(1 - p'_j \cdot p_i) = p_h / d'$$

  The effect of such an application of the graph reduction rule is to shorten the longest cycle of immediate transitions, so the induction hypothesis can be exploited.

(v) If $\mathcal{I}[\![E]\!]$ has finitely many states, then $\mathcal{I}[\![E]\!]$ has finitely many transitions because E is closed and guarded. Therefore, the algorithm terminates and the number of steps is bounded by the number of vanishing states in $S_{E,\mathcal{I}}$. ■

Proof of Lemma 5.1: Once observed that $\mathcal{B}$ is an equivalence relation because it is the transitive closure of the union of equivalence relations, assume that $(E_1, E_2) \in \mathcal{B}$. Since $\mathcal{B} = \cup_{n\in\mathbb{N}_+}\mathcal{B}^{(n)}$ where $\mathcal{B}^{(n)} = (\cup_{i\in I}\mathcal{B}_i)^n$, we have $(E_1, E_2) \in \mathcal{B}^{(n)}$ for some $n \in \mathbb{N}_+$. The result follows by proving by induction on $n \in \mathbb{N}_+$ that, whenever $(E_1, E_2) \in \mathcal{B}^{(n)}$, then $\mathit{Rate}(E_1, a, l, C) = \mathit{Rate}(E_2, a, l, C)$ for all $a \in \mathit{AType}$, $l \in \mathit{APLev}$, and $C \in \mathcal{G}/\mathcal{B}$.

- If $n = 1$ then $(E_1, E_2) \in \mathcal{B}_i$ for some $i \in I$. Let $\mathcal{G}/\mathcal{B}_i = \{C_{i,j} \mid j \in J_i\}$. Since $(E_1, E_2) \in \mathcal{B}_i$ implies $(E_1, E_2) \in \mathcal{B}$, we have that for each $C_{i,j} \in \mathcal{G}/\mathcal{B}_i$ there exists $C \in \mathcal{G}/\mathcal{B}$ such that $C_{i,j} \subseteq C$, so each equivalence class of $\mathcal{B}$ can be written as the union of a set of equivalence classes of $\mathcal{B}_i$. As a consequence, for all $a \in \mathit{AType}$, $l \in \mathit{APLev}$, and $C \in \mathcal{G}/\mathcal{B}$, if $C = \cup_{j\in J'_i} C_{i,j}$ where $J'_i \subseteq J_i$, then $\mathit{Rate}(E_1, a, l, C) = \mathit{Aggr}\{\!|\, \mathit{Rate}(E_1, a, l, C_{i,j}) \mid j \in J'_i \,|\!\} = \mathit{Aggr}\{\!|\, \mathit{Rate}(E_2, a, l, C_{i,j}) \mid j \in J'_i \,|\!\} = \mathit{Rate}(E_2, a, l, C)$ because $\mathcal{B}_i$ is a strong EMB.
- If $n > 1$ from $(E_1, E_2) \in \mathcal{B}^{(n)}$ we derive that there exists $F \in \mathcal{G}$ such that $(E_1, F) \in \mathcal{B}^{(n-1)}$ and $(F, E_2) \in \mathcal{B}_i$ for some $i \in I$. Thus for all $a \in \mathit{AType}$, $l \in \mathit{APLev}$, and $C \in \mathcal{G}/\mathcal{B}$, it turns out $\mathit{Rate}(E_1, a, l, C) = \mathit{Rate}(F, a, l, C)$ by the induction hypothesis and $\mathit{Rate}(F, a, l, C) = \mathit{Rate}(E_2, a, l, C)$ by applying the same argument as the previous point. ■

Proof of Prop. 5.1: By definition, $\sim_{\mathrm{EMB}}$ contains the largest strong EMB. If we prove that $\sim_{\mathrm{EMB}}$ is a strong EMB, then we are done. Since $\sim_{\mathrm{EMB}} \subseteq \sim^+_{\mathrm{EMB}}$ trivially holds and $\sim^+_{\mathrm{EMB}} \subseteq \sim_{\mathrm{EMB}}$ is due to the fact that $\sim^+_{\mathrm{EMB}}$ is a strong EMB by virtue of Lemma 5.1 and that $\sim_{\mathrm{EMB}}$ contains all the strong EMBs by definition, we have $\sim_{\mathrm{EMB}} = \sim^+_{\mathrm{EMB}}$. Again $\sim^+_{\mathrm{EMB}}$ is a strong EMB because of Lemma 5.1, hence so is $\sim_{\mathrm{EMB}}$. ■

Proof of Prop. 5.3: Given $\mathcal{B} \subseteq \mathcal{G} \times \mathcal{G}$ strong EMB up to $\sim_{\mathrm{EMB}}$ and let $\mathcal{B}' = \mathcal{B} \cup \mathcal{B}^{-1}$, we first prove that $(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+$ is a strong EMB. Let $(E_1, E_2) \in (\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+$. Since $(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+ = \cup_{n\in\mathbb{N}_+}(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^n$, we have $(E_1, E_2) \in (\mathcal{B}' \cup \sim_{\mathrm{EMB}})^n$ for some $n \in \mathbb{N}_+$. The result follows by proving by induction on $n \in \mathbb{N}_+$ that, whenever $(E_1, E_2) \in (\mathcal{B}' \cup \sim_{\mathrm{EMB}})^n$, then $\mathit{Rate}(E_1, a, l, C) = \mathit{Rate}(E_2, a, l, C)$ for all $a \in \mathit{AType}$, $l \in \mathit{APLev}$, and $C \in \mathcal{G}/(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+$.

- If $n = 1$ then $(E_1, E_2) \in \mathcal{B}' \cup \sim_{\mathrm{EMB}}$. If $(E_1, E_2) \in \mathcal{B}'$ then the result trivially follows from the fact that $\mathcal{B}$ is a strong EMB up to $\sim_{\mathrm{EMB}}$. If $(E_1, E_2) \in \sim_{\mathrm{EMB}}$ we observe that, since $\sim_{\mathrm{EMB}} \subseteq (\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+$, each equivalence class of $(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+$ can be written as the union of some equivalence classes of $\sim_{\mathrm{EMB}}$. As a consequence, for all $a \in \mathit{AType}$, $l \in \mathit{APLev}$, and $C \in \mathcal{G}/(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+$, it turns out $\mathit{Rate}(E_1, a, l, C) = \mathit{Rate}(E_2, a, l, C)$, because $E_1 \sim_{\mathrm{EMB}} E_2$.
- If $n > 1$ from $(E_1, E_2) \in (\mathcal{B}' \cup \sim_{\mathrm{EMB}})^n$ we derive that there exists $F \in \mathcal{G}$ such that $(E_1, F) \in (\mathcal{B}' \cup \sim_{\mathrm{EMB}})^{n-1}$ and $(F, E_2) \in \mathcal{B}' \cup \sim_{\mathrm{EMB}}$. Thus we have that for all $a \in \mathit{AType}$, $l \in \mathit{APLev}$, and $C \in \mathcal{G}/(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+$, it turns out $\mathit{Rate}(E_1, a, l, C) = \mathit{Rate}(F, a, l, C)$ by the induction hypothesis and $\mathit{Rate}(F, a, l, C) = \mathit{Rate}(E_2, a, l, C)$ by applying the same argument as the previous point.

To complete the proof, we observe that $\mathcal{B} \subseteq (\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+$ by construction and $(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+ \subseteq \sim_{\mathrm{EMB}}$ because $(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+$ is a strong EMB, hence $\mathcal{B} \subseteq \sim_{\mathrm{EMB}}$ by transitivity. ■

Proof of Thm. 5.1: Let $E_1, E_2 \in \mathcal{G}_{\Theta,\mathrm{rnd}}$.

(*i*) Let $\mathcal{B} \subseteq \mathcal{G}_{\Theta,\mathrm{rnd}} \times \mathcal{G}_{\Theta,\mathrm{rnd}}$ be a strong EMB such that $(E_1, E_2) \in \mathcal{B}$. Given $<a, \tilde{\lambda}> \in Act$, we prove that $\mathcal{B}' = (\mathcal{B} \cup \{(<a, \tilde{\lambda}>.E_1, <a, \tilde{\lambda}>.E_2), (<a, \tilde{\lambda}>.E_2, <a, \tilde{\lambda}>.E_1)\})^+$ is a strong EMB. Observed that $\mathcal{B}'$ is an equivalence relation, we have two cases.

- If $(<a, \tilde{\lambda}>.E_1, <a, \tilde{\lambda}>.E_2) \in \mathcal{B}$, then $\mathcal{B}' = \mathcal{B}$ and the result trivially follows.
- Assume that $(<a, \tilde{\lambda}>.E_1, <a, \tilde{\lambda}>.E_2) \notin \mathcal{B}$. Observed that

$$\mathcal{G}_{\Theta,\mathrm{rnd}}/\mathcal{B}' = (\mathcal{G}_{\Theta,\mathrm{rnd}}/\mathcal{B} - \{[<a, \tilde{\lambda}>.E_1]_{\mathcal{B}}, [<a, \tilde{\lambda}>.E_2]_{\mathcal{B}}\}) \cup \{[<a, \tilde{\lambda}>.E_1]_{\mathcal{B}} \cup [<a, \tilde{\lambda}>.E_2]_{\mathcal{B}}\}$$

  let $(F_1, F_2) \in \mathcal{B}'$, $b \in AType$, $l \in APLev$, and $C \in \mathcal{G}_{\Theta,\mathrm{rnd}}/\mathcal{B}'$.

  * If $(F_1, F_2) \in \mathcal{B}$ and $C \in \mathcal{G}_{\Theta,\mathrm{rnd}}/\mathcal{B} - \{[<a, \tilde{\lambda}>.E_1]_{\mathcal{B}}, [<a, \tilde{\lambda}>.E_2]_{\mathcal{B}}\}$, then trivially $Rate(F_1, b, l, C) = Rate(F_2, b, l, C)$.
  * If $(F_1, F_2) \in \mathcal{B}$ and $C = [<a, \tilde{\lambda}>.E_1]_{\mathcal{B}} \cup [<a, \tilde{\lambda}>.E_2]_{\mathcal{B}}$, then for $j \in \{1, 2\}$ we have

$$Rate(F_j, b, l, C) = Rate(F_j, b, l, [<a, \tilde{\lambda}>.E_1]_{\mathcal{B}}) \; Aggr \; Rate(F_j, b, l, [<a, \tilde{\lambda}>.E_2]_{\mathcal{B}})$$

    so $Rate(F_1, b, l, C) = Rate(F_2, b, l, C)$.
  * If $(F_1, F_2) \in \mathcal{B}' - \mathcal{B}$, i.e. $F_1 \in [<a, \tilde{\lambda}>.E_1]_{\mathcal{B}}$ and $F_2 \in [<a, \tilde{\lambda}>.E_2]_{\mathcal{B}}$, then for $j \in \{1, 2\}$ we have

$$Rate(F_j, b, l, C) = \begin{cases} \tilde{\lambda} & \text{if } \; b = a \wedge l = PL(<a, \tilde{\lambda}>) \wedge C = [E_j]_{\mathcal{B}'} \\ \bot & \text{otherwise} \end{cases}$$

    Since $[E_1]_{\mathcal{B}'} = [E_2]_{\mathcal{B}'}$, it turns out that $Rate(F_1, b, l, C) = Rate(F_2, b, l, C)$.

(*ii*) Given $L \subseteq AType - \{\tau\}$, we prove that $\mathcal{B}' = \mathcal{B} \cup Id_{\mathcal{G}_{\Theta,\mathrm{rnd}}}$, where $\mathcal{B} = \{(E_1/L, E_2/L) \mid E_1 \sim_{\mathrm{EMB}} E_2\}$, is a strong EMB. Observed that $\mathcal{B}'$ is an equivalence relation and that either each of the terms of an equivalence class has "$\_/L$" as outermost operator or none of them has, let $(F_1, F_2) \in \mathcal{B}'$, $a \in AType$, $l \in APLev$, and $C \in \mathcal{G}_{\Theta,\mathrm{rnd}}/\mathcal{B}'$.

- If $(F_1, F_2) \in Id_{\mathcal{G}_{\Theta,\mathrm{rnd}}}$, then trivially $Rate(F_1, a, l, C) = Rate(F_2, a, l, C)$.
- If $(F_1, F_2) \in \mathcal{B}$, then $F_1 \equiv E_1/L$ and $F_2 \equiv E_2/L$ where $E_1 \sim_{\mathrm{EMB}} E_2$.
  * If none of the terms in C has "$\_/L$" as outermost operator, then trivially $Rate(F_1, a, l, C) = \bot = Rate(F_2, a, l, C)$.
  * If each of the terms in C has "$\_/L$" as outermost operator, given $E/L \in C$ it turns out that $C = \{E'/L \mid E' \in [E]_{\sim_{\mathrm{EMB}}}\}$. Thus for $j \in \{1, 2\}$ we have

$$Rate(F_j,a,l,C) = \begin{cases} Rate(E_j,a,l,[E]_{\sim_{\mathrm{EMB}}}) \\ Rate(E_j,\tau,l,[E]_{\sim_{\mathrm{EMB}}})\ Aggr\ Aggr\{\!|\ Rate(E_j,b,l,[E]_{\sim_{\mathrm{EMB}}}) \mid b \in L\ |\!\} \end{cases}$$

depending on whether $a \notin L \cup \{\tau\}$ or $a = \tau$. From $E_1 \sim_{\mathrm{EMB}} E_2$ it follows $Rate(F_1,a,l,C) = Rate(F_2,a,l,C)$.

(iii) Given $\varphi \in ATRFun$, the proof that $\mathcal{B}' = \mathcal{B} \cup Id_{\mathcal{G}_{\Theta,\mathrm{rnd}}}$, where $\mathcal{B} = \{(E_1[\varphi], E_2[\varphi]) \mid E_1 \sim_{\mathrm{EMB}} E_2\}$, is a strong EMB is similar to the one developed in *(ii)*. The main difference is that in the last subcase the result follows from the fact that for $j \in \{1,2\}$ we have

$$Rate(F_j,a,l,C) = Aggr\{\!|\ Rate(E_j,b,l,[E]_{\sim_{\mathrm{EMB}}}) \mid \varphi(b) = a\ |\!\}$$

(iv) The proof that $\mathcal{B}' = \mathcal{B} \cup Id_{\mathcal{G}_{\Theta,\mathrm{rnd}}}$, where $\mathcal{B} = \{(\Theta(E_1), \Theta(E_2)) \mid E_1 \sim_{\mathrm{EMB}} E_2\}$, is a strong EMB is similar to the one developed in *(ii)*. The main difference is that in the last subcase the result follows from the fact that for $j \in \{1,2\}$ we have

$$Rate(F_j,a,l,C) = \begin{cases} Rate(E_j,a,l,[E]_{\sim_{\mathrm{EMB}}}) & \text{if } \neg(E_j \xrightarrow{b,\tilde{\lambda}} F \wedge PL(<b,\tilde{\lambda}>) > l) \vee (l = -1) \\ \bot & \text{otherwise} \end{cases}$$

(v) Let $\mathcal{B} \subseteq \mathcal{G}_{\Theta,\mathrm{rnd}} \times \mathcal{G}_{\Theta,\mathrm{rnd}}$ be a strong EMB such that $(E_1,E_2) \in \mathcal{B}$. Given $F \in \mathcal{G}_{\Theta,\mathrm{rnd}}$, the proof that $\mathcal{B}' = (\mathcal{B} \cup \{(E_1+F, E_2+F), (E_2+F, E_1+F)\})^+$ is a strong EMB is similar to the one developed in *(i)*. The main difference is that in the last subcase the result follows from the fact that for $j \in \{1,2\}$ we have

$$Rate(F_j,b,l,C) = Rate(E_j,b,l,C)\ Aggr\ Rate(F,b,l,C)$$

and from the following considerations:

- If $C \in \mathcal{G}_{\Theta,\mathrm{rnd}}/\mathcal{B} - \{[E_1+F]_{\mathcal{B}}, [E_2+F]_{\mathcal{B}}\}$, then from $(E_1,E_2) \in \mathcal{B}$ we derive $Rate(E_1,b,l,C) = Rate(E_2,b,l,C)$ so $Rate(F_1,b,l,C) = Rate(F_2,b,l,C)$.
- If $C = [E_1+F]_{\mathcal{B}} \cup [E_2+F]_{\mathcal{B}}$, then for $j \in \{1,2\}$ we have

  $$Rate(E_j,b,l,C) = Rate(E_j,b,l,[E_1+F]_{\mathcal{B}})\ Aggr\ Rate(E_j,b,l,[E_2+F]_{\mathcal{B}})$$

  Since $(E_1,E_2) \in \mathcal{B}$, it turns out that $Rate(E_1,b,l,C) = Rate(E_2,b,l,C)$ so $Rate(F_1,b,l,C) = Rate(F_2,b,l,C)$.

(vi) Given $F \in \mathcal{G}_{\Theta,\mathrm{rnd}}$ and $S \subseteq AType - \{\tau\}$, the proof that $\mathcal{B}' = \mathcal{B} \cup Id_{\mathcal{G}_{\Theta,\mathrm{rnd}}}$, where $\mathcal{B} = \{(E_1 \parallel_S F, E_2 \parallel_S F) \mid E_1 \sim_{\mathrm{EMB}} E_2 \wedge E_1 \parallel_S F, E_2 \parallel_S F \in \mathcal{G}_{\Theta,\mathrm{rnd}}\}$, is a strong EMB is similar to the one developed in *(ii)*. The main difference is that in the last subcase, where given $E \parallel_S G \in C$ it turns out that $C = \{E' \parallel_S G \mid E' \in [E]_{\sim_{\mathrm{EMB}}}\}$, the result follows from the considerations below:

– If $a \notin S$, then for $j \in \{1,2\}$ we have that

$$Rate(F_j,a,l,C) = \begin{cases} Rate(E_j,a,l,[E]_{\sim_{\rm EMB}})\ Aggr\ Rate(F,a,l,\{G\}) \\ Rate(E_j,a,l,[E]_{\sim_{\rm EMB}}) \\ Rate(F,a,l,\{G\}) \\ \bot \end{cases}$$

depending on whether $E_j \in [E]_{\sim_{\rm EMB}} \wedge F \equiv G$ or $E_j \notin [E]_{\sim_{\rm EMB}} \wedge F \equiv G$ or $E_j \in [E]_{\sim_{\rm EMB}} \wedge F \not\equiv G$ or $E_j \notin [E]_{\sim_{\rm EMB}} \wedge F \not\equiv G$. Since $E_1 \sim_{\rm EMB} E_2$, it follows that $Rate(F_1,a,l,C) = Rate(F_2,a,l,C)$.

– If $a \in S$, then for $j \in \{1,2\}$ we have that

$$Rate(F_j,a,l,C) = \begin{cases} Split(Rate(E_j,a,l,[E]_{\sim_{\rm EMB}}),p_F)\ Aggr\ Split(Rate(F,a,l,\{G\}),p_{E_j}) \\ Split(Rate(E_j,a,l,[E]_{\sim_{\rm EMB}}),p_F) \\ Split(Rate(F,a,l,\{G\}),p_{E_j}) \\ \bot \end{cases}$$

depending on whether $Rate(F,a,-1,\{G\}) \neq \bot \wedge Rate(E_j,a,-1,[E]_{\sim_{\rm EMB}}) \neq \bot$ or $Rate(F,a,-1,\{G\}) \neq \bot \wedge Rate(E_j,a,-1,[E]_{\sim_{\rm EMB}}) = \bot$ or $Rate(F,a,-1,\{G\}) = \bot \wedge Rate(E_j,a,-1,[E]_{\sim_{\rm EMB}}) \neq \bot$ or $Rate(F,a,-1,\{G\}) = \bot \wedge Rate(E_j,a,-1,[E]_{\sim_{\rm EMB}}) = \bot$, where p_F (p_{E_j}) is the ratio of the number of passive potential moves with type a from F (E_j) to G ($[E]_{\sim_{\rm EMB}}$) to the number of passive potential moves with type a from F (E_j). Since $E_1 \sim_{\rm EMB} E_2$, and by virtue of the restriction to $\mathcal{G}_{\Theta,\rm rnd}$ (which causes p_{E_j} to be equal to 1 for $j \in \{1,2\}$ and $l \geq 0$), it follows that $Rate(F_1,a,l,C) = Rate(F_2,a,l,C)$. ■

Proof of Thm. 5.2: It suffices to prove that

$$\mathcal{B} = \{(F_1,F_2) \in \mathcal{G}_{\Theta,\rm rnd} \times \mathcal{G}_{\Theta,\rm rnd} \mid F_1 \equiv F\langle\langle B := A_1\rangle\rangle \wedge F_2 \equiv F\langle\langle B := A_2\rangle\rangle \wedge F \in \mathcal{L}_\Theta \text{ pcg with at most } B \in Const_{Def_\Theta}(F) \text{ free}\}$$

is a strong EMB up to $\sim_{\rm EMB}$: the result will follow by taking $F \equiv B$. Let $\mathcal{B}' = \mathcal{B} \cup \mathcal{B}^{-1}$. Given $(F_1,F_2) \in \mathcal{B}$, $a \in AType$, $l \in APLev$, and $C \in \mathcal{G}_{\Theta,\rm rnd}/(\mathcal{B}' \cup \sim_{\rm EMB})^+$, we must prove that $Rate(F_1,a,l,C) = Rate(F_2,a,l,C)$.

We start by showing that $Rate(F_1,a,l,C) \leq Rate(F_2,a,l,C)$ for all $a \in AType$, $l \in APLev$, and $C \in \mathcal{G}_{\Theta,\rm rnd}/(\mathcal{B}' \cup \sim_{\rm EMB})^+$. If $Rate(F_1,a,l,C) = \bot$ there is nothing to prove, otherwise we proceed by induction on the maximum depth d of the inferences of the potential moves for F_1 having type a, priority level l, and derivative term in C. Note that, because of the inductive definition of predicate *RND* which characterizes terms in $\mathcal{G}_{\Theta,\rm rnd}$, when proceeding by induction on the depth of the inferences of the potential moves for a term in $\mathcal{G}_{\Theta,\rm rnd}$ we are guaranteed that the involved subterms are in $\mathcal{G}_{\Theta,\rm rnd}$ as well, so the induction hypothesis can be safely applied.

- If $d = 1$ then only the rule for the action prefix operator has been used to deduce the existing potential move. Therefore $F \equiv <a,\tilde{\lambda}>.F'$ with $PL(<a,\tilde{\lambda}>) = l$ and for $j \in \{1,2\}$ we have $F_j \equiv <a,\tilde{\lambda}>.F'\langle\langle B :=$

$A_j\rangle\!\rangle$. Since $(F'\langle\!\langle B := A_1\rangle\!\rangle, F'\langle\!\langle B := A_2\rangle\!\rangle) \in \mathcal{B}$, it turns out that $C = [F'\langle\!\langle B := A_1\rangle\!\rangle]_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+} = [F'\langle\!\langle B := A_2\rangle\!\rangle]_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$ hence $Rate(F_1, a, l, C) = \tilde{\lambda} = Rate(F_2, a, l, C)$.

- If $d > 1$ then several subcases arise depending on the syntactical structure of F.

  - If $F \equiv F'/L$ then for $j \in \{1, 2\}$ we have $F_j \equiv F'\langle\!\langle B := A_j\rangle\!\rangle/L$. Since F_1 has a potential move having derivative term in C, there exists $G \in \mathcal{G}_\Theta$ such that G/L is the derivative term and $C = [G/L]_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$. Because of the congruence property of $\sim_{\mathrm{EMB}}$ and $\mathcal{B}$ w.r.t. "$\_/L$", we have that $H/L \in [G/L]_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$ implies $[H]_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}/L \subseteq [G/L]_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$. [1] As a consequence $[G/L]_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+} = (\cup_i C_i/L) \cup D$ where $C_i = [G_i]_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$ are distinguished equivalence classes with $G_i/L \in [G/L]_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$ and D is a set of terms not having "$\_/L$" as outermost operator, hence not reachable from F_1 or F_2. Since d is the maximum depth of the inferences of the potential moves for F_1 having type a, priority level l, and derivative term in C, for any i either (i) $F'\langle\!\langle B := A_1\rangle\!\rangle$ has no potential moves having type a, priority level l, and derivative term in C_i, or (ii) $d-1$ is the maximum depth of the inferences of the potential moves for $F'\langle\!\langle B := A_1\rangle\!\rangle$ having type a, priority level l, and derivative term in C_i. For $j \in \{1, 2\}$ we have

$$Rate(F_j, a, l, C) = \underset{i}{Aggr}\, Rate(F_j, a, l, C_i/L)$$

    where for any i

$$Rate(F_j, a, l, C_i/L) = \begin{cases} Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l, C_i) \\ Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, \tau, l, C_i)\ Aggr \\ \qquad Aggr\{\!|\ Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, b, l, C_i) \mid b \in L\ |\!\} \end{cases}$$

    depending on whether $a \notin L \cup \{\tau\}$ or $a = \tau$. By applying the induction hypothesis to each $F'\langle\!\langle B := A_1\rangle\!\rangle$, b (with $b \in L \cup \{\tau\}$ if $a = \tau$ and $b = a$ otherwise), l, and C_i such that $Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, b, l, C_i) \neq \bot$, we have that $Rate(F_1, a, l, C_i/L) \leq Rate(F_2, a, l, C_i/L)$ for any i. It follows that $Rate(F_1, a, l, C) \leq Rate(F_2, a, l, C)$.

  - If $F \equiv F'[\varphi]$ then the proof is similar to the one developed in the first subcase. The result follows by applying the induction hypothesis to the fact that for $j \in \{1, 2\}$ we have

$$Rate(F_j, a, l, C_i[\varphi]) = Aggr\{\!|\ Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, b, l, C_i) \mid \varphi(b) = a\ |\!\}$$

  - If $F \equiv \Theta(F')$ then the proof is similar to the one developed in the first subcase. The result follows by applying the induction hypothesis to the fact that for $j \in \{1, 2\}$ we have

$$Rate(F_j, a, l, \Theta(C_i)) = \begin{cases} Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l, C_i) \\ \bot \end{cases}$$

    depending on whether $\neg(F'\langle\!\langle B := A_j\rangle\!\rangle \xrightarrow{b, \tilde{\lambda}} F \wedge PL(<b, \tilde{\lambda}>) > l) \vee (l = -1)$.

[1] Given a set of terms $\mathcal{T}$, $\mathcal{T}/L$ is the set of terms $\{E/L \mid E \in \mathcal{T}\}$.

– If $F \equiv F' + F''$ then for $j \in \{1,2\}$ we have $F_j \equiv F'\langle\langle B := A_j\rangle\rangle + F''\langle\langle B := A_j\rangle\rangle$. Since d is the maximum depth of the inferences of the potential moves for F_1 having type a, priority level l, and derivative term in C, for any $G \in \{F'\langle\langle B := A_1\rangle\rangle, F''\langle\langle B := A_1\rangle\rangle\}$ either (i) G has no potential moves having type a, priority level l, and derivative term in C, or (ii) $d-1$ is the maximum depth of the inferences of the potential moves for G having type a, priority level l, and derivative term in C. For $j \in \{1,2\}$ we have:

$$Rate(F_j, a, l, C) = Rate(F'\langle\langle B := A_j\rangle\rangle, a, l, C)\ Aggr\ Rate(F''\langle\langle B := A_j\rangle\rangle, a, l, C)$$

By applying the induction hypothesis to $F'\langle\langle B := A_1\rangle\rangle$, a, l, and C if $Rate(F'\langle\langle B := A_1\rangle\rangle, a, l, C) \neq \bot$, and to $F''\langle\langle B := A_1\rangle\rangle$, a, l, and C if $Rate(F''\langle\langle B := A_1\rangle\rangle, a, l, C) \neq \bot$, we have that $Rate(F'\langle\langle B := A_1\rangle\rangle, a, l, C) \leq Rate(F'\langle\langle B := A_2\rangle\rangle, a, l, C)$ and $Rate(F''\langle\langle B := A_1\rangle\rangle, a, l, C) \leq Rate(F''\langle\langle B := A_2\rangle\rangle, a, l, C)$. It follows that $Rate(F_1, a, l, C) \leq Rate(F_2, a, l, C)$.

– If $F \equiv F' \parallel_S F''$ then for $j \in \{1,2\}$ we have $F_j \equiv F'\langle\langle B := A_j\rangle\rangle \parallel_S F''\langle\langle B := A_j\rangle\rangle$. Since F_1 has a potential move having derivative term in C, there exist $G', G'' \in \mathcal{G}_\Theta$ such that $G' \parallel_S G''$ is the derivative term and $C = [G' \parallel_S G'']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$.

  * If $a \notin S$ then either $G'' \equiv F''\langle\langle B := A_1\rangle\rangle$ or $G' \equiv F'\langle\langle B := A_1\rangle\rangle$. Because of the congruence property of $\sim_{\mathrm{EMB}}$ and $\mathcal{B}$ w.r.t. "$\_ \parallel_S \_$", we have that $H \parallel_S F''\langle\langle B := A_1\rangle\rangle \in [G' \parallel_S G'']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$ implies $[H]_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+} \parallel_S F''\langle\langle B := A_1\rangle\rangle \subseteq [G' \parallel_S G'']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$ and $F'\langle\langle B := A_1\rangle\rangle \parallel_S H \in [G' \parallel_S G'']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$ implies $F'\langle\langle B := A_1\rangle\rangle \parallel_S [H]_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+} \subseteq [G' \parallel_S G'']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$. Moreover, due to the congruence property of $\mathcal{B}$ w.r.t. "$\_ \parallel_S \_$" we have that, for all $\mathcal{T} \subseteq \mathcal{G}_{\Theta,\mathrm{rnd}}$, $\mathcal{T} \parallel_S F''\langle\langle B := A_1\rangle\rangle \subseteq [G' \parallel_S G'']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$ if and only if $\mathcal{T} \parallel_S F''\langle\langle B := A_2\rangle\rangle \subseteq [G' \parallel_S G'']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$ and $F'\langle\langle B := A_1\rangle\rangle \parallel_S \mathcal{T} \subseteq [G' \parallel_S G'']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$ if and only if $F'\langle\langle B := A_2\rangle\rangle \parallel_S \mathcal{T} \subseteq [G' \parallel_S G'']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$. As a consequence $[G' \parallel_S G'']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+} = (\cup_i C'_i \parallel_S F''\langle\langle B := A_1\rangle\rangle) \cup (\cup_i C'_i \parallel_S F''\langle\langle B := A_2\rangle\rangle) \cup (\cup_j F'\langle\langle B := A_1\rangle\rangle \parallel_S C''_j) \cup (\cup_j F'\langle\langle B := A_2\rangle\rangle \parallel_S C''_j) \cup D$ where $C'_i = [H'_i]_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$ are distinguished equivalence classes with $H'_i \parallel_S F''\langle\langle B := A_1\rangle\rangle \in [G' \parallel_S G'']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$ and $C''_j = [H''_j]_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$ are distinguished equivalence classes with $F'\langle\langle B := A_1\rangle\rangle \parallel_S H''_j \in [G' \parallel_S G'']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$. Besides, classes C'_i and C''_j are such that if $F_1 \equiv F'\langle\langle B := A_1\rangle\rangle \parallel_S F''\langle\langle B := A_1\rangle\rangle \in [G' \parallel_S G'']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$ then there exist i such that $F'\langle\langle B := A_1\rangle\rangle \in C'_i$ and j such that $F''\langle\langle B := A_1\rangle\rangle \in C''_j$. Moreover D is a set of terms not of the form $H \parallel_S F''\langle\langle B := A_j\rangle\rangle$ or $F'\langle\langle B := A_j\rangle\rangle \parallel_S H$ for any term H, hence not reachable from F_1 or F_2. Since d is the maximum depth of the inferences of the potential moves for F_1 having type a, priority level l, and derivative term in C, considered term $F'\langle\langle B := A_1\rangle\rangle$ ($F''\langle\langle B := A_1\rangle\rangle$) we have that for any i (for any j) either (i) $F'\langle\langle B := A_1\rangle\rangle$ ($F''\langle\langle B := A_1\rangle\rangle$) has no potential moves having type a, priority level l, and derivative term in C'_i (C''_j), or (ii) $d-1$ is the maximum depth of the inferences of the potential moves for $F'\langle\langle B := A_1\rangle\rangle$ ($F''\langle\langle B := A_1\rangle\rangle$) having type a, priority level l, and derivative term in C'_i (C''_j). For $k \in \{1,2\}$ we have

$$Rate(F_k, a, l, C) = \underset{i}{Aggr}\ Rate(F_k, a, l, C'_i \parallel_S F''\langle\langle B := A_k\rangle\rangle)\ Aggr$$

$$\underset{j}{Aggr}\ Rate(F_k, a, l, F'\langle\langle B := A_k\rangle\rangle \parallel_S C''_j)$$

where for any i

$$Rate(F_k, a, l, C'_i \parallel_S F''\langle\langle B := A_k\rangle\rangle) = Rate(F'\langle\langle B := A_k\rangle\rangle, a, l, C'_i)$$

and for any j

$$Rate(F_k, a, l, F'\langle\langle B := A_k\rangle\rangle \parallel_S C''_j) = Rate(F''\langle\langle B := A_k\rangle\rangle, a, l, C''_j)$$

By applying the induction hypothesis to each $F'\langle\langle B := A_1\rangle\rangle$, a, l, and C'_i such that $Rate(F'\langle\langle B := A_1\rangle\rangle, a, l, C'_i) \neq \bot$ and to each $F''\langle\langle B := A_1\rangle\rangle$, a, l, and C''_j such that $Rate(F''\langle\langle B := A_1\rangle\rangle, a, l, C''_j) \neq \bot$, we have that $Rate(F_1, a, l, C'_i \parallel_S F''\langle\langle B := A_1\rangle\rangle) \leq Rate(F_2, a, l, C'_i \parallel_S F''\langle\langle B := A_2\rangle\rangle)$ for any i and that $Rate(F_1, a, l, F'\langle\langle B := A_1\rangle\rangle \parallel_S C''_j) \leq Rate(F_2, a, l, F'\langle\langle B := A_2\rangle\rangle \parallel_S C''_j)$ for any j. It follows that $Rate(F_1, a, l, C) \leq Rate(F_2, a, l, C)$.

∗ Let $a \in S$. Because of the congruence property of $\sim_{\mathrm{EMB}}$ and $\mathcal{B}$ w.r.t. "$\_ \parallel_S \_$", we have that $H' \parallel_S H'' \in [G' \parallel_S G'']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$ implies $[H']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+} \parallel_S [H'']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+} \subseteq [G' \parallel_S G'']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$.[2] As a consequence $[G' \parallel_S G'']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+} = (\cup_i C'_i \parallel_S C''_i) \cup D$ where $C'_i = [G'_i]_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$ and $C''_i = [G''_i]_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$ are distinguished pairs of equivalence classes with $G'_i \parallel_S G''_i \in [G' \parallel_S G'']_{(\mathcal{B}' \cup \sim_{\mathrm{EMB}})^+}$ and D is a set of terms not having "$\_ \parallel_S \_$" as outermost operator, hence not reachable from F_1 or F_2. Since d is the maximum depth of the inferences of the potential moves for F_1 having type a, priority level l, and derivative term in C, for any i if $Rate(F''\langle\langle B := A_1\rangle\rangle, a, -1, C''_i) \neq \bot$ ($Rate(F'\langle\langle B := A_1\rangle\rangle, a, -1, C'_i) \neq \bot$) then either (i) $F'\langle\langle B := A_1\rangle\rangle$ ($F''\langle\langle B := A_1\rangle\rangle$) has no potential moves having type a, priority level l, and derivative term in C'_i (C''_i), or (ii) $d-1$ is the maximum depth of the inferences of the potential moves for $F'\langle\langle B := A_1\rangle\rangle$ ($F''\langle\langle B := A_1\rangle\rangle$) having type a, priority level l, and derivative term in C'_i (C''_i). For $j \in \{1, 2\}$ we have

$$Rate(F_j, a, l, C) = \underset{i}{Aggr}\ Rate(F_j, a, l, C'_i \parallel_S C''_i)$$

where for any i

$$Rate(F_j, a, l, C'_i \parallel_S C''_i) = \begin{cases} Split(Rate(F'\langle\langle B := A_j\rangle\rangle, a, l, C'_i), p_{F''_j})\ Aggr \\ \quad Split(Rate(F''\langle\langle B := A_j\rangle\rangle, a, l, C''_i), p_{F'_j}) \\ Split(Rate(F'\langle\langle B := A_j\rangle\rangle, a, l, C'_i), p_{F''_j}) \\ Split(Rate(F''\langle\langle B := A_j\rangle\rangle, a, l, C''_i), p_{F'_j}) \\ \bot \end{cases}$$

depending on whether, defined $R'_j = Rate(F'\langle\langle B := A_j\rangle\rangle, a, -1, C'_i)$ and $R''_j = Rate(F''\langle\langle B :=$

[2] Given two sets of terms $\mathcal{T}_1$ and $\mathcal{T}_2$, $\mathcal{T}_1 \parallel_S \mathcal{T}_2$ is the set of terms $\{E_1 \parallel_S E_2 \mid E_1 \in \mathcal{T}_1 \wedge E_2 \in \mathcal{T}_2\}$.

$A_j\rangle\!\rangle, a, -1, C_i'')$, $R_j' \neq \bot \wedge R_j'' \neq \bot$ or $R_j' = \bot \wedge R_j'' \neq \bot$ or $R_j' \neq \bot \wedge R_j'' = \bot$ or $R_j' = \bot \wedge R_j'' = \bot$, where $p_{F_j'}$ ($p_{F_j''}$) is the ratio of the number of passive potential moves with type a from $F'\langle\!\langle B := A_j\rangle\!\rangle$ ($F''\langle\!\langle B := A_j\rangle\!\rangle$) to C_i' (C_i'') to the number of passive potential moves with type a from $F'\langle\!\langle B := A_j\rangle\!\rangle$ ($F''\langle\!\langle B := A_j\rangle\!\rangle$). By applying the induction hypothesis to each $F'\langle\!\langle B := A_1\rangle\!\rangle$, a, l, and C_i' such that $Rate(F'\langle\!\langle B := A_1\rangle\!\rangle, a, l, C_i') \neq \bot$ and to each $F''\langle\!\langle B := A_1\rangle\!\rangle$, a, l, and C_i'' such that $Rate(F''\langle\!\langle B := A_1\rangle\!\rangle, a, l, C_i'') \neq \bot$, and using the restriction to $\mathcal{G}_{\Theta,\mathrm{rnd}}$ (which causes $p_{F_j'}$ and $p_{F_j''}$ to be equal to 1 for $j \in \{1,2\}$ and $l \geq 0$), we have that $Rate(F_1, a, l, C_i' \parallel_S C_i'') \leq Rate(F_2, a, l, C_i' \parallel_S C_i'')$ for any i. It follows that $Rate(F_1, a, l, C) \leq Rate(F_2, a, l, C)$.

– If $F \equiv B'$ then for $j \in \{1,2\}$ we have $F_j \equiv B'\langle\!\langle B := A_j\rangle\!\rangle$.

  * If $B' \equiv B$ then for $j \in \{1,2\}$ we have $F_j \equiv A_j$. Since d is the maximum depth of the inferences of the potential moves for F_1 having type a, priority level l, and derivative term in C, then $d-1$ is the maximum depth of the inferences of the potential moves for $E_1\langle\!\langle A := A_1\rangle\!\rangle$ having type a, priority level l, and derivative term in C. For $j \in \{1,2\}$ we have

$$Rate(F_j, a, l, C) = Rate(E_j\langle\!\langle A := A_j\rangle\!\rangle, a, l, C)$$

    By applying the induction hypothesis to $E_1\langle\!\langle A := A_1\rangle\!\rangle$, a, l, and C we have that $Rate(E_1\langle\!\langle A := A_1\rangle\!\rangle, a, l, C) \leq Rate(E_1\langle\!\langle A := A_2\rangle\!\rangle, a, l, C)$. From the fact that $E_1 \sim_{\mathrm{EMB}} E_2$ we derive $Rate(E_1\langle\!\langle A := A_2\rangle\!\rangle, a, l, C) = Rate(E_2\langle\!\langle A := A_2\rangle\!\rangle, a, l, C)$. It follows that $Rate(F_1, a, l, C) \leq Rate(F_2, a, l, C)$.

  * If $B' \not\equiv B$ then let $B' \triangleq F'$ be the defining equation of B'. Since d is the maximum depth of the inferences of the potential moves for F_1 having type a, priority level l, and derivative term in C, then $d-1$ is the maximum depth of the inferences of the potential moves for $F'\langle\!\langle B := A_1\rangle\!\rangle$ having type a, priority level l, and derivative term in C. For $j \in \{1,2\}$ we have

$$Rate(F_j, a, l, C) = Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l, C)$$

    By applying the induction hypothesis to $F'\langle\!\langle B := A_1\rangle\!\rangle$, a, l, and C we have that $Rate(F'\langle\!\langle B := A_1\rangle\!\rangle, a, l, C) \leq Rate(F'\langle\!\langle B := A_2\rangle\!\rangle, a, l, C)$. It follows that $Rate(F_1, a, l, C) \leq Rate(F_2, a, l, C)$.

By applying a symmetric argument we also have that $Rate(F_2, a, l, C) \leq Rate(F_1, a, l, C)$, hence $Rate(F_1, a, l, C) = Rate(F_2, a, l, C)$. ∎

Proof of Thm. 5.3: Let $E_1 \sim_{\mathrm{EMB}} E_2$ via some strong EMB $\mathcal{B}$. The result follows by observing that $\mathcal{B} \cap (S_{E_1,\mathcal{I}} \times S_{E_2,\mathcal{I}})$ is a bisimulation and a p-bisimulation, too. ∎

Proof of Thm. 5.4: ($\Longrightarrow$) Since $E_1 \sim_{\rm EMB} E_2$ and $\sim_{\rm EMB}$ is a congruence over $\mathcal{E}'$ w.r.t. the alternative and parallel composition operators (problems with nondeterminism do not arise as terms in $\mathcal{E}'$ cannot execute passive actions at all), for all $F \in \mathcal{G}$ and $S \subseteq AType - \{\tau\}$ we have $E_1 + F \sim_{\rm EMB} E_2 + F$ and $E_1 \parallel_S F \sim_{\rm EMB} E_2 \parallel_S F$. The results follows by applying Thm. 5.3.

($\Longleftarrow$) We prove the contrapositive, so we assume that $E_1 \not\sim_{\rm EMB} E_2$ and we demonstrate that E_1 and E_2 are distinguishable w.r.t. $\sim_{\rm FP}$ by means of an appropriate context based on the alternative composition operator or the parallel composition operator. We preliminarily assume that $E_1 \sim_{\rm FP} E_2$ otherwise $E_1 + \underline{0} \not\sim_{\rm FP} E_2 + \underline{0}$ trivially. From $E_1 \not\sim_{\rm EMB} E_2$ it follows that E_1 and E_2, after executing n actions which pairwise have the same type and priority level, reach E_1' and E_2', respectively, which violate the necessary condition expressed by Prop. 5.2. [3] We prove the result by proceeding by induction on n:

- Let $n = 0$, i.e. assume that there exist $a \in AType$ and $l \in APLev$ such that $Rate(E_1, a, l, \mathcal{E}') \neq Rate(E_2, a, l, \mathcal{E}')$. Since both sides of the inequality cannot be $\bot$, we assume that $Rate(E_1, a, l, \mathcal{E}') \neq \bot$. There are several cases:
  - Assume that $Rate(E_2, a, l, \mathcal{E}') \neq \bot$.
    - Assume that $l = 0$.
      - If $a = \tau$, or $a \neq \tau$ and $Rate(E_1, \tau, 0, \mathcal{E}') \neq Rate(E_2, \tau, 0, \mathcal{E}')$, then
        $$E_1 \parallel_{AType - \{\tau\}} \underline{0} \not\sim_{\rm P} E_2 \parallel_{AType - \{\tau\}} \underline{0}$$
        because the aggregated rates of these two terms are $Rate(E_1, \tau, 0, \mathcal{E}')$ and $Rate(E_2, \tau, 0, \mathcal{E}')$, respectively, hence p-bisimilarity is violated.
      - If $a \neq \tau$ and $Rate(E_1, \tau, 0, \mathcal{E}') = Rate(E_2, \tau, 0, \mathcal{E}')$, then
        $$E_1 \parallel_{AType - \{\tau\}} <a, *>.\underline{0} \not\sim_{\rm P} E_2 \parallel_{AType - \{\tau\}} <a, *>.\underline{0}$$
        because the aggregated rates of these two terms are $Rate(E_1, a, 0, \mathcal{E}') \, Aggr \, Rate(E_1, \tau, 0, \mathcal{E}')$ and $Rate(E_2, a, 0, \mathcal{E}') \, Aggr \, Rate(E_2, \tau, 0, \mathcal{E}')$, respectively, hence p-bisimilarity is violated.
    - Assume that $l \geq 1$.
      - If $a = \tau$, or $a \neq \tau$ and $Rate(E_1, \tau, l, \mathcal{E}') \neq Rate(E_2, \tau, l, \mathcal{E}')$, given $b, c \in AType - \{\tau\}$ occurring neither in E_1 nor in E_2, we have that
        $$E_1 \parallel_{AType - \{\tau\} - \{b,c\}} <b, \infty_{l,w}>.<c, \infty_{l,w}>.\underline{0} \not\sim_P$$

[3] The contrapositive of this assertion can be proved by showing that the reflexive, symmetric and transitive closure $\mathcal{B}'$ of relation $\mathcal{B}$ composed of the pairs (F_1, F_2) of terms which satisfy the necessary condition expressed by Prop. 5.2 and are reachable from E_1 and E_2, respectively, after executing a sequence of actions which pairwise have the same type and priority level, is a strong EMB. In particular, whenever $(F_1, F_2) \in \mathcal{B}$, then for all $a \in AType$, $l \in APLev$, and $C \in \mathcal{E}'/\mathcal{B}'$ we have by construction that either $Rate(F_1, a, l, C) = \bot = Rate(F_2, a, l, C)$ or $Rate(F_1, a, l, C) = Rate(F_1, a, l, \mathcal{E}') = Rate(F_2, a, l, \mathcal{E}') = Rate(F_2, a, l, C)$.

$$E_2 \parallel_{AType-\{\tau\}-\{b,c\}} <b, \infty_{l,w}>.<c, \infty_{l,w}>.\underline{0}$$

because the former term reaches a class composed of a state having an outgoing transition labeled with $c, \infty_{l,w}$ with probability $w/(w_1 + w)$ where $\infty_{l,w_1} = \mathit{Rate}(E_1, \tau, l, \mathcal{E}')$, while the latter term reaches the same class with probability $w/(w_2 + w)$ where $\infty_{l,w_2} = \mathit{Rate}(E_2, \tau, l, \mathcal{E}')$.

· If $a \neq \tau$ and $\mathit{Rate}(E_1, \tau, l, \mathcal{E}') = \mathit{Rate}(E_2, \tau, l, \mathcal{E}')$, given $b, c \in \mathit{AType} - \{\tau\}$ occurring neither in E_1 nor in E_2, we have that

$$E_1 \parallel_{AType-\{\tau\}-\{b,c\}} (<a, *>.\underline{0} + <b, \infty_{l,w}>.<c, \infty_{l,w}>.\underline{0}) \not\sim_P$$

$$E_2 \parallel_{AType-\{\tau\}-\{b,c\}} (<a, *>.\underline{0} + <b, \infty_{l,w}>.<c, \infty_{l,w}>.\underline{0})$$

because the former term reaches a class composed of a state having an outgoing transition labeled with $c, \infty_{l,w}$ with probability $w/(w_1 + w)$ where $\infty_{l,w_1} = \mathit{Rate}(E_1, a, l, \mathcal{E}')\ \mathit{Aggr}\ \mathit{Rate}(E_1, \tau, l, \mathcal{E}')$, while the latter term reaches the same class with probability $w/(w_2+w)$ where $\infty_{l,w_2} = \mathit{Rate}(E_2, a, l, \mathcal{E}')\ \mathit{Aggr}\ \mathit{Rate}(E_2, \tau, l, \mathcal{E}')$.

– Assume that $\mathit{Rate}(E_2, a, l, \mathcal{E}') = \bot$.

∗ Assume that there exists $l' \in \mathit{APLev} - \{l\}$ such that $\mathit{Rate}(E_2, a, l', \mathcal{E}') \neq \bot$.

· If $a = \tau$, given $b \in \mathit{AType} - \{\tau\}$ occurring neither in E_1 nor in E_2, we have that

$$E_1 \parallel_{AType-\{\tau\}-\{b\}} <b, \infty_{\max(l,l'),w}>.\underline{0} \not\sim_F E_2 \parallel_{AType-\{\tau\}-\{b\}} <b, \infty_{\max(l,l'),w}>.\underline{0}$$

because one of these two terms can execute an action with type b while the other term cannot.

· If $a \neq \tau$, given $b \in \mathit{AType} - \{\tau\}$ occurring neither in E_1 nor in E_2, we have that

$$E_1 \parallel_{AType-\{\tau\}-\{b\}} (<a, *>.\underline{0} + <b, \infty_{\max(l,l'),w}>.\underline{0}) \not\sim_F$$

$$E_2 \parallel_{AType-\{\tau\}-\{b\}} (<a, *>.\underline{0} + <b, \infty_{\max(l,l'),w}>.\underline{0})$$

because one of these two terms can execute an action with type b while the other term cannot.

∗ Assume that $\mathit{Aggr}\{\!|\ \mathit{Rate}(E_2, a, l', \mathcal{E}') \mid l' \in \mathit{APLev}\ |\!\} = \bot$.

· If $a = \tau$ then

$$E_1 \parallel_{AType-\{\tau\}} \underline{0} \not\sim_F E_2 \parallel_{AType-\{\tau\}} \underline{0}$$

because the left hand side term can execute an action with type τ while the right hand side term cannot.

· If $a \neq \tau$ then

$$E_1 \parallel_{AType-\{\tau\}} <a,*>.\underline{0} \not\sim_F E_2 \parallel_{AType-\{\tau\}} <a,*>.\underline{0}$$

because the left hand side term can execute an action with type a while the right hand side term cannot.

- Let $n \geq 1$. Let a be the first action type on the trajectory leading from E_1 and E_2 to E_1' and E_2', respectively, and let F_1 and F_2 be the two a-derivative terms on the trajectory, respectively. By the induction hypothesis, there exists a context "$\_ \parallel_{AType-\{\tau\}-S} F$" that distinguishes F_1 and F_2 with respect to $\sim_{\mathrm{FP}}$. If $a = \tau$ then

$$E_1 \parallel_{AType-\{\tau\}-S} F \not\sim_{\mathrm{FP}} E_2 \parallel_{AType-\{\tau\}-S} F$$

else

$$E_1 \parallel_{AType-\{\tau\}-S} <a,*>.F \not\sim_{\mathrm{FP}} E_2 \parallel_{AType-\{\tau\}-S} <a,*>.F$$

with the appropriate action type renamings applied to S and F in order to avoid clashes with action types occurring in E_1 and E_2. ■

Proof of Lemma. 5.4: Given $E \in \mathcal{G}_{\Theta,\mathrm{rnd},\mathrm{nrec}}$, we proceed by induction on $size(E)$:

- Let $size(E) = 1$. The result follows by proving $\mathcal{A} \vdash E = \underline{0}$ by induction on the syntactical structure of E:
  - If $E \equiv \underline{0}$ then the result follows by reflexivity.
  - The case $E \equiv <a, \tilde{\lambda}>.E'$ is not possible because it contradicts the hypothesis $size(E) = 1$.
  - If $E \equiv E'/L$ then E' is a subterm of E such that $size(E') = 1$, hence $\mathcal{A} \vdash E' = \underline{0}$ by the structural induction hypothesis. The result follows by substitutivity, $\mathcal{A}_5$, and transitivity.
  - If $E \equiv E'[\varphi]$ or $E \equiv \Theta(E')$ then the result can be proved by proceeding as in the previous point and by exploiting $\mathcal{A}_8$ in the first case and $\mathcal{A}_{11}$ in the second case.
  - If $E \equiv E_1 + E_2$ then E_1 and E_2 are subterms of E such that $size(E_1) = 1$ and $size(E_2) = 1$, hence $\mathcal{A} \vdash E_1 = \underline{0}$ and $\mathcal{A} \vdash E_2 = \underline{0}$ by the structural induction hypothesis. The result follows by substitutivity, $\mathcal{A}_3$, and transitivity.
  - The case $E \equiv E_1 \parallel_S E_2$ is not possible because it contradicts the hypothesis $size(E) = 1$.
- Let the result hold whenever $size(E) \leq n \in \mathbb{N}_+$ and assume $size(E) = n + 1$. The result follows by proceeding by induction on the syntactical structure of E:
  - The case $E \equiv \underline{0}$ is not possible because it contradicts the hypothesis $size(E) = n + 1 \geq 2$.

– If $E \equiv <a, \tilde{\lambda}>.E'$ then $size(E') = n$ hence by the induction hypothesis there exists $F' \in \mathcal{G}_{\Theta,\mathrm{rnd},\mathrm{nrec}}$ in snf such that $\mathcal{A} \vdash E' = F'$. The result follows by substitutivity.

– If $E \equiv E'/L$ then E' is a subterm of E hence by the structural induction hypothesis there exists $F' \in \mathcal{G}_{\Theta,\mathrm{rnd},\mathrm{nrec}}$ in snf such that $\mathcal{A} \vdash E' = F'$. By substitutivity we obtain $\mathcal{A} \vdash E = F'/L$. Assuming $F' \equiv \sum_{i \in I} <a_i, \tilde{\lambda}_i>.F'_i$ where every F'_i is in snf, by exploiting $\mathcal{A}_7$, $\mathcal{A}_6$, and transitivity we obtain $\mathcal{A} \vdash F'/L = \sum_{i \in I}(<a_i, \tilde{\lambda}_i>.F'_i)/L = \sum_{i \in I \wedge a_i \in L} <\tau, \tilde{\lambda}_i>.F'_i/L + \sum_{i \in I \wedge a_i \notin L} <a_i, \tilde{\lambda}_i>.F'_i/L$. Since for every $i \in I$ we have $size(F'_i/L) = size(F'_i) < size(F') = size(F'/L) \leq size(E) = n + 1$, by the induction hypothesis it follows that for every $i \in I$ there exists $F''_i \in \mathcal{G}_{\Theta,\mathrm{rnd},\mathrm{nrec}}$ in snf such that $\mathcal{A} \vdash F'_i/L = F''_i$. If we take $F \equiv \sum_{i \in I \wedge a_i \in L} <\tau, \tilde{\lambda}_i>.F''_i + \sum_{i \in I \wedge a_i \notin L} <a_i, \tilde{\lambda}_i>.F''_i$, then the result follows by substitutivity and possible applications of $\mathcal{A}_4$ preceded by $\mathcal{A}_1$ and $\mathcal{A}_2$.

– If $E \equiv E'[\varphi]$ or $E \equiv \Theta(E')$ then the result can be proved by proceeding as in the previous point and by exploiting $\mathcal{A}_{10}$ and $\mathcal{A}_9$ in the first case and $\mathcal{A}_{12}$ in the second case.

– If $E \equiv E_1 + E_2$ then E_1 and E_2 are subterms of E hence by the structural induction hypothesis there exist $F_1, F_2 \in \mathcal{G}_{\Theta,\mathrm{rnd},\mathrm{nrec}}$ in snf such that $\mathcal{A} \vdash E_1 = F_1$ and $\mathcal{A} \vdash E_2 = F_2$. By substitutivity we obtain $\mathcal{A} \vdash E = F_1 + F_2$ and the result follows after possible applications of $\mathcal{A}_3$ and $\mathcal{A}_4$ preceded by $\mathcal{A}_1$ and $\mathcal{A}_2$.

– If $E \equiv E_1 \parallel_S E_2$ then $size(E_1) \leq n$ and $size(E_2) \leq n$ hence by the induction hypothesis there exist $F_1, F_2 \in \mathcal{G}_{\Theta,\mathrm{rnd},\mathrm{nrec}}$ in snf such that $\mathcal{A} \vdash E_1 = F_1$ and $\mathcal{A} \vdash E_2 = F_2$. By substitutivity we obtain $\mathcal{A} \vdash E = F_1 \parallel_S F_2$. There are three cases:

  * If $size(E_1) = size(E_2) = 1$ then $F_1 \equiv F_2 \equiv \underline{0}$ hence $\mathcal{A} \vdash E = \underline{0}$ by $\mathcal{A}_{13}$, $\mathcal{A}_3$, and transitivity.
  * If $size(E_1) = 1$ and $size(E_2) > 1$ then $F_1 \equiv \underline{0}$ and $F_2 \equiv \sum_{i \in I} <a_i, \tilde{\lambda}_i>.F_{2,i}$ where every $F_{2,i}$ is in snf. By $\mathcal{A}_{13}$, $\mathcal{A}_3$, and transitivity we obtain $\mathcal{A} \vdash F_1 \parallel_S F_2 = \sum_{j \in J_2} <a_j, \tilde{\lambda}_j>.(\underline{0} \parallel_S F_{2,j})$. Since for every $j \in J_2$ we have $size(\underline{0} \parallel_S F_{2,j}) < size(\underline{0} \parallel_S F_2) \leq size(E) = n + 1$, by the induction hypothesis it follows that for every $j \in J_2$ there exists $F'_{2,j}$ in snf such that $\mathcal{A} \vdash \underline{0} \parallel_S F_{2,j} = F'_{2,j}$, hence the result follows by substitutivity and possible applications of $\mathcal{A}_4$ preceded by $\mathcal{A}_1$ and $\mathcal{A}_2$. The proof for the symmetric case is similar.
  * If $size(E_1) > 1$ and $size(E_2) > 1$ then $F_1 \equiv \sum_{i \in I_1} <a_i, \tilde{\lambda}_i>.F_{1,i}$ and $F_2 \equiv \sum_{i \in I_2} <a_i, \tilde{\lambda}_i>.F_{2,i}$ where every $F_{1,i}$ and $F_{2,i}$ is in snf. By $\mathcal{A}_{13}$ and transitivity we obtain $\mathcal{A} \vdash F_1 \parallel_S F_2 = \sum_{j \in J_1} <a_j, \tilde{\lambda}_j>.(F_{1,j} \parallel_S F_2) + \sum_{j \in J_2} <a_j, \tilde{\lambda}_j>.(F_1 \parallel_S F_{2,j}) + \sum_{k \in K_1} \sum_{h \in H_k} <a_k, Split(\tilde{\lambda}_k, 1/n_k)>.(F_{1,k} \parallel_S F_{2,h}) + \sum_{k \in K_2} \sum_{h \in H_k} <a_k, Split(\tilde{\lambda}_k, 1/n_k)>.(F_{1,h} \parallel_S F_{2,k})$. Since the size of every term surrounded by parentheses is at most n, by the induction hypothesis each such term can be proved equal via $\mathcal{A}$ to a term in snf, hence the result follows by substitutivity and possible applications of $\mathcal{A}_4$ preceded by $\mathcal{A}_1$ and $\mathcal{A}_2$. ■

Proof of Thm. 5.5: We must prove that for all $E_1, E_2 \in \mathcal{G}_{\Theta,\mathrm{rnd},\mathrm{nrec}}$ we have $\mathcal{A} \vdash E_1 = E_2 \iff E_1 \sim_{\mathrm{EMB}} E_2$.

($\Longrightarrow$) It is a straightforward consequence of the fact that reflexivity, symmetry, transitivity, and substitutivity of $Ded(\mathcal{A})$ are matched by the reflexive, symmetric, transitive and congruence properties of $\sim_{\mathrm{EMB}}$ and the fact that every axiom in $\mathcal{A}$ can be restated as a property of $\sim_{\mathrm{EMB}}$.

($\Longleftarrow$) Assume $E_1 \sim_{\mathrm{EMB}} E_2$. There are two cases:

- If E_1 and E_2 are both in snf, the result follows by proceeding by induction on $size(E_1)$:
  - If $size(E_1) = 1$ then $E_1 \equiv \underline{0} \equiv E_2$ since they are both in snf. The result follows by reflexivity.
  - If $size(E_1) > 1$ then $E_1 \equiv \sum_{i \in I_1} <a_{1,i}, \tilde{\lambda}_{1,i}>.E_{1,i}$ and $E_2 \equiv \sum_{i \in I_2} <a_{2,i}, \tilde{\lambda}_{2,i}>.E_{2,i}$. Since $E_1 \sim_{\mathrm{EMB}} E_2$, from the definition of snf we have that $|I_1| = |I_2|$ and for every summand $<a_{1,i}, \tilde{\lambda}_{1,i}>.E_{1,i}$ in E_1 there exists exactly one summand $<a_{2,j}, \tilde{\lambda}_{2,j}>.E_{2,j}$ in E_2 such that $a_{1,i} = a_{2,j} \wedge \tilde{\lambda}_{1,i} = \tilde{\lambda}_{2,j} \wedge E_{1,i} \sim_{\mathrm{EMB}} E_{2,j}$, and vice versa. By the induction hypothesis we obtain that $\mathcal{A} \vdash E_{1,i} = E_{2,j}$ and the result follows by substitutivity.
- If E_1 or E_2 is not in snf, then by Lemma 5.4 there exist $F_1, F_2 \in \mathcal{G}_{\Theta,\mathrm{nrec}}$ in snf such that $\mathcal{A} \vdash E_1 = F_1$ and $\mathcal{A} \vdash E_2 = F_2$. We then obtain $E_1 \sim_{\mathrm{EMB}} F_1$ and $E_2 \sim_{\mathrm{EMB}} F_2$ by soundness and $F_1 \sim_{\mathrm{EMB}} F_2$ by the symmetric and the transitive properties, hence $\mathcal{A} \vdash F_1 = F_2$ by the previous point. By symmetry and transitivity we finally obtain $\mathcal{A} \vdash E_1 = E_2$. ■

Proof of Thm. 5.6: From Thm. 5.3 it follows that $\mathcal{M}[\![E_1]\!]$ is p-bisimilar to $\mathcal{M}[\![E_2]\!]$ via a given p-bisimulation $\mathcal{B}_{\mathcal{M}}$. The result follows by proving that $\mathcal{B}_{\mathcal{M}_l} = \{(C_1, C_2) \in S_{E_1,\mathcal{M}_l} \times S_{E_2,\mathcal{M}_l} \mid \exists s_1 \in C_1.\, \exists s_2 \in C_2.\, (s_1, s_2) \in \mathcal{B}_{\mathcal{M}}\}$ is a p-bisimulation between $\mathcal{M}_l[\![E_1]\!]$ and $\mathcal{M}_l[\![E_2]\!]$. Let $\mathcal{B}'_{\mathcal{M}_l}$ ($\mathcal{B}'_{\mathcal{M}}$) be the reflexive, symmetric and transitive closure of $\mathcal{B}_{\mathcal{M}_l}$ ($\mathcal{B}_{\mathcal{M}}$).

- Let $D \in (S_{E_1,\mathcal{M}_l} \cup S_{E_2,\mathcal{M}_l})/\mathcal{B}'_{\mathcal{M}_l}$. For $j \in \{1, 2\}$ we have

$$\sum_{C \in D \cap S_{E_j,\mathcal{M}_l}} P_{E_j,\mathcal{M}_l}(C) = \sum_{C \in D \cap S_{E_j,\mathcal{M}_l}} \sum_{s \in C} P_{E_j,\mathcal{M}}(s) = \sum_{s \in (\cup_{C \in D} C) \cap S_{E_j,\mathcal{M}}} P_{E_j,\mathcal{M}}(s)$$

  Since $\cup_{C \in D} C$ is the union of some equivalence classes of $\mathcal{B}'_{\mathcal{M}}$ and $\mathcal{B}_{\mathcal{M}}$ is a p-bisimulation, it follows that

$$\sum_{C \in D \cap S_{E_1,\mathcal{M}_l}} P_{E_1,\mathcal{M}_l}(C) = \sum_{C \in D \cap S_{E_2,\mathcal{M}_l}} P_{E_2,\mathcal{M}_l}(C)$$

- If $(C_1, C_2) \in \mathcal{B}_{\mathcal{M}_l}$ then there exist $s_1 \in C_1$ and $s_2 \in C_2$ such that $(s_1, s_2) \in \mathcal{B}_{\mathcal{M}}$. Then $H_{E_1,\mathcal{M}}(s_1) = H_{E_2,\mathcal{M}}(s_2)$ since $\mathcal{B}_{\mathcal{M}}$ is a p-bisimulation, hence $H_{E_1,\mathcal{M}_l}(C_1) = H_{E_2,\mathcal{M}_l}(C_2)$ by construction.

- Let $(C_1, C_2) \in \mathcal{B}_{\mathcal{M}_l}$ due to the existence of $s_1 \in C_1$ and $s_2 \in C_2$ such that $(s_1, s_2) \in \mathcal{B}_{\mathcal{M}}$. Let $D \in (S_{E_1,\mathcal{M}_l} \cup S_{E_2,\mathcal{M}_l})/\mathcal{B}'_{\mathcal{M}_l}$. For $j \in \{1, 2\}$ we have

$$\sum \{\!|\, \lambda \mid \exists C'_j \in D \cap S_{E_j,\mathcal{M}_l}.\, C_j \xrightarrow{\lambda}_{E_j,\mathcal{M}_l} C'_j \,|\!\} =$$

$$\sum \{\!|\, \lambda \mid \exists s'_j \in C'_j \in D \cap S_{E_j,\mathcal{M}_l}.\, s_j \xrightarrow{\lambda}_{E_j,\mathcal{M}} s'_j \,|\!\} =$$

$$\sum\{\!|\, \lambda \mid \exists s'_j \in (\cup_{C\in D} C) \cap S_{E_j,\mathcal{M}}.\, s_j \xrightarrow{\lambda}_{E_j,\mathcal{M}} s'_j \,|\!\}$$

where the first equality holds whichever is $s_j \in C_j$ because $\mathcal{M}_l[\![E_j]\!]$ is obtained from $\mathcal{M}[\![E_j]\!]$ via ordinary lumping. Since $\cup_{C\in D} C$ is the union of some equivalence classes of $\mathcal{B}'_{\mathcal{M}}$ and $\mathcal{B}_{\mathcal{M}}$ is a p-bisimulation, it follows that

$$\sum\{\!|\, \lambda \mid \exists C'_1 \in D \cap S_{E_1,\mathcal{M}_l}.\, C_1 \xrightarrow{\lambda}_{E_1,\mathcal{M}_l} C'_1 \,|\!\} =$$
$$\sum\{\!|\, \lambda \mid \exists C'_2 \in D \cap S_{E_2,\mathcal{M}_l}.\, C_2 \xrightarrow{\lambda}_{E_2,\mathcal{M}_l} C'_2 \,|\!\}$$ ■

Proof of Cor. 5.1: The searched bijection is relation $\mathcal{B}_{\mathcal{M}_l}$ built in the proof of the previous theorem. The reason is that $\mathcal{B}_{\mathcal{M}_l}$ is a p-bisimulation and $\mathcal{M}_l[\![E_1]\!]$ and $\mathcal{M}_l[\![E_2]\!]$ are the coarsest ordinary lumpings of $\mathcal{M}[\![E_1]\!]$ and $\mathcal{M}[\![E_2]\!]$, respectively. ■

Proof of Thm. 6.2: The proof is divided into three parts.

(1st part) Suppose that priority levels are taken into account neither in EMPA nor in PGSPNs. More accurately, assume that the active transitions of PGSPNs are not divided into different priority levels and consider $\mathcal{I}'[\![E]\!]$ instead of $\mathcal{I}[\![E]\!]$, i.e. consider the LTS (whose set of states is denoted by S'_E) representing the integrated interleaving semantics of E if function *Select* were not applied. Then we can demonstrate, by following the proof developed in [145] Thm. 3.7.18, that $\mathcal{RG}[\![\mathcal{N}_{\rm loc}[\![E]\!]]\!]$ is bisimilar to $\mathcal{I}'[\![E]\!]$ through relation

$$\mathcal{B} = \{(F, Q) \in S'_E \times RS(dec_{\rm loc}(E)) \mid Q \,\text{swf} \wedge dec_{\rm loc}(F) = upd(Q)\}$$

where:

- The definition of *strongly well formed (swf)* marking is the following:
  - $\{\!|\, \underline{0} \,|\!\}$ and $\{\!|\, <a, \tilde{\lambda}>.E \,|\!\}$ are swf.
  - If Q is swf, then so are Q/L and $Q[\varphi]$.
  - If $Q_1 \oplus Q_3$ is swf, $dom(Q_1) \cap dom(Q_3) = \emptyset$, either $Q_3 = \emptyset$ or not all components in Q_3 contain "+" as their outermost operator, and Q_2 is complete, then $(Q_1 + Q_2) \oplus Q_3$ and $(Q_2 + Q_1) \oplus Q_3$ are swf.
  - If Q_1 and Q_2 are swf, then so is $Q_1 \parallel_S id \oplus id \parallel_S Q_2$.

  This property is satisfied by complete elements of $\mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm loc})$ and is invariant for transition firing.
- The definition of the *update operation (upd)* on swf markings is the following:
  - If Q is complete, then $upd(Q) = Q$.
  - If $Q \equiv Q'/L$ is incomplete, then $upd(Q) = upd(Q')/L$.

- If $Q \equiv Q'[\varphi]$ is incomplete, then $upd(Q) = upd(Q')[\varphi]$.
- If $Q \equiv (Q_1 + Q_2) \oplus Q_3$ or $Q \equiv (Q_2 + Q_1) \oplus Q_3$ is incomplete, and Q_2 is complete, then $upd(Q) = upd(Q_1 \oplus Q_3)$.
- If $Q \equiv Q_1 \parallel_S id \oplus id \parallel_S Q_2$ is incomplete, then $upd(Q) = upd(Q_1) \parallel_S id \oplus id \parallel_S upd(Q_2)$.

For each swf marking Q, it turns out that $upd(Q)$ is complete.

(2nd part) Now we want to prove, under the same assumption made at the beginning of the previous part, that bisimulation $\mathcal{B}$ is actually an isomorphism between $\mathcal{RG}[\![\mathcal{N}_{\rm loc}[\![E]\!]]\!]$ and $\mathcal{I}'[\![E]\!]$.

First, we have to prove that $\mathcal{B}$ is a function. Given $F \in S'_E$, since F is reachable from E and $\mathcal{B}$ is a bisimulation, there must exist $Q \in RS(dec_{\rm loc}(E))$ such that $(F, Q) \in \mathcal{B}$, i.e. $dec_{\rm loc}(F) = upd(Q)$. It remains to prove the uniqueness of such a swf reachable marking Q. Suppose that there exist $Q_1, Q_2 \in RS(dec_{\rm loc}(E))$ swf such that $Q_1 \neq Q_2$ and $upd(Q_1) = upd(Q_2) = dec_{\rm loc}(F)$. This can stem only from the fact that there exists at least a pair composed of a subterm G of a place V_1 in Q_1 and a subterm $G + G'$ of a place V_2 in Q_2 that reside in the same position of the syntactical structure of V_1 and V_2 (if such a pair did not exist, Q_1 and Q_2 could not be different from each other). The existence of this pair contradicts the reachability of Q_1. In fact, we recall that the decomposition function $dec_{\rm loc}$ distributes all the alternative composition operators between all the appropriate places and when one of these places is part of a marking involved in a transition firing, either it remains unchanged or it gives rise to a new place where the alternative composition operator disappears and only the alternative involved remains after it has been transformed (see the rules for the alternative composition operator).

Second, we have to prove that $\mathcal{B}$ is injective. This is trivial, because if there exist $F_1, F_2 \in S'_E$ and $Q \in RS(dec_{\rm loc}(E))$ such that $dec_{\rm loc}(F_1) = upd(Q) = dec_{\rm loc}(F_2)$, then necessarily $F_1 = F_2$ as $dec_{\rm loc}$ is injective.

Third, we have to prove that $\mathcal{B}$ is surjective. This is true because given $Q \in RS(dec_{\rm loc}(E))$, since Q is reachable from $dec_{\rm loc}(E)$ and $\mathcal{B}$ is a bisimulation, there must exist $F \in S'_E$ such that $(F, Q) \in \mathcal{B}$.

Finally, we have to prove that $\mathcal{B}$ satisfies the isomorphism clauses. This follows immediately from the fact that $\mathcal{B}$ is a bijection fulfilling the bisimilarity clauses.

(3rd part) Now let us take into account the priority levels. Since the priority mechanism for EMPA actions is exactly the same as the priority mechanism for PGSPN transitions, from the previous step it follows that $\mathcal{RG}[\![\mathcal{N}_{\rm loc}[\![E]\!]]\!]$ is isomorphic to $\mathcal{I}[\![E]\!]$. ■

Proof of Thm. 6.4: Let $(E, Q) \in \mathcal{B}$. We first show that $Rate(E, a, l, C) \leq Rate(Q, a, l, C)$ for all $a \in AType$, $l \in APLev$, and $C \in (\mathcal{G} \cup \mathcal{M}u_{\rm fin}(\mathcal{V}_{\rm lab}))/\mathcal{B}'$. If $Rate(E, a, l, C) = \bot$ there is nothing to prove, otherwise we proceed by induction on the maximum depth d of the inferences of the potential moves for E having type a, priority level l, and derivative term in C.

- If $d = 1$ then only the rule for the action prefix operator has been used to deduce the existing potential move. Therefore $E \equiv <a, \tilde{\lambda}>.E'$ with $PL(<a, \tilde{\lambda}>) = l$ and $E' \in C$. From the definition of $\mathcal{B}$ it follows that $Q = \{\!|\, \emptyset <a, \tilde{\lambda}>.E' \,|\!\}$, thus in the reachability graph Q has only one transition to $dec_{\rm lab}(E')$ labeled with $a, \tilde{\lambda}$. From $(E', dec_{\rm lab}(E')) \in \mathcal{B}$ it follows that $dec_{\rm lab}(E') \in C$, hence the result.

- If $d > 1$ then several subcases arise depending on the syntactical structure of E.

  - If $E \equiv E'/L$ then from the definition of $\mathcal{B}$ it follows that $Q = dec_{\rm lab}(E'\langle L := L_{new}\rangle) \oplus \{\!|\, I_{a_{new}} \mid a \in L \,|\!\} \oplus \{\!|\, [a_{new} \mapsto \tau] \mid a \in L \,|\!\}$. Since d is the maximum depth of the inferences of the potential moves for E having type a, priority level l, and derivative term in C, either E' has no potential moves having type a, priority level l, and derivative term in C' (where $C = C'/L$), or $d - 1$ is the maximum depth of the inferences of the potential moves for E' having type a, priority level l, and derivative term in C'. Since

    $$Rate(E, a, l, C) = \begin{cases} Rate(E', a, l, C') & \text{if } a \notin L \cup \{\tau\} \\ Rate(E', \tau, l, C')\ Aggr\ Aggr\{\!|\, Rate(E', b, l, C') \mid b \in L \,|\!\} & \text{if } a = \tau \end{cases}$$

    and similarly for Q and $dec_{\rm lab}(E')$, the result follows by the induction hypothesis.

  - If $E \equiv E'[\varphi]$ then the proof is similar to the one developed in the first subcase.

  - If $E \equiv E_1 + E_2$ then from the definition of $\mathcal{B}$ it follows that $Q = \{k_{new}\}(dec_{\rm lab}(E_1)) \oplus \{\bar{k}_{new}\}(dec_{\rm lab}(E_2))$ if $E_1 + E_2$ is not guarded, $Q = \{\!|\, \emptyset(E_1 + E_2) \,|\!\}$ otherwise. Since d is the maximum depth of the inferences of the potential moves for E having type a, priority level l, and derivative term in C, for any $j \in \{1, 2\}$ either E_j has no potential moves having type a, priority level l, and derivative term in C, or $d - 1$ is the maximum depth of the inferences of the potential moves for E_j having type a, priority level l, and derivative term in C. Note that $Rate(E, a, l, C) = Rate(E_1, a, l, C)\ Aggr\ Rate(E_2, a, l, C)$.

    - If $E_1 + E_2$ is not guarded then $Rate(Q, a, l, C) = Rate(\{k_{new}\}dec_{\rm lab}(E_1), a, l, C)\ Aggr\ Rate(\{\bar{k}_{new}\}dec_{\rm lab}(E_2), a, l, C) = Rate(dec_{\rm lab}(E_1), a, l, C)\ Aggr\ Rate(dec_{\rm lab}(E_2), a, l, C)$ since from the definition of $\mathcal{B}$ it follows that place multisets occurring in C can comprise any subset of conflicts. The result follows by the induction hypothesis.

    - If $E_1 + E_2$ is guarded then the result trivially holds because $Rate(Q, a, l, C) = Rate(E_1, a, l, C)\ Aggr\ Rate(E_2, a, l, C)$.

  - If $E \equiv E_1 \parallel_S E_2$ then from the definition of $\mathcal{B}$ it follows that $Q = dec_{\rm lab}(E_1\langle S := S_{new,l}\rangle) \oplus dec_{\rm lab}(E_2\langle S := S_{new,r}\rangle) \oplus \{\!|\, I_{a_{new,l}} \mid a \in S \,|\!\} \oplus \{\!|\, I_{a_{new,r}} \mid a \in S \,|\!\} \oplus \{\!|\, [a_{new,l}, a_{new,r} \mapsto a] \mid a \in S \,|\!\}$.

    - Let $a \notin S$. Since d is the maximum depth of the inferences of the potential moves for E having type a, priority level l, and derivative term in C, for any $j \in \{1, 2\}$ either E_j has no potential moves having type a, priority level l, and derivative term in C_j (where $C = C_1 \parallel_S C_2$), or $d - 1$ is the maximum depth of the inferences of the potential moves for E_j having type a,

priority level l, and derivative term in C_j. Since

$$Rate(E,a,l,C) = \begin{cases} Rate(E_1,a,l,C_1)\ Aggr\ Rate(E_2,a,l,C_2) & \text{if } E_1 \in C_1 \wedge E_2 \in C_2 \\ Rate(E_1,a,l,C_1) & \text{if } E_1 \notin C_1 \wedge E_2 \in C_2 \\ Rate(E_2,a,l,C_2) & \text{if } E_1 \in C_1 \wedge E_2 \notin C_2 \\ \bot & \text{if } E_1 \notin C_1 \wedge E_2 \notin C_2 \end{cases}$$

and similarly for Q, $dec_{\mathrm{lab}}(E_1)$, and $dec_{\mathrm{lab}}(E_2)$, the result follows by the induction hypothesis. Note that the result holds in the case $dec_{\mathrm{lab}}(E_1\langle S := S_{new,l}\rangle) = dec_{\mathrm{lab}}(E_2\langle S := S_{new,r}\rangle)$ thanks to the marking dependent factor f_1.

* Let $a \in S$. Since d is the maximum depth of the inferences of the potential moves for E having type a, priority level l, and derivative term in C, let $C = C_1 \|_S C_2$ we have that if $Rate(E_2,a,-1,C_2) \neq \bot$ ($Rate(E_1,a,-1,C_1) \neq \bot$) then either E_1 (E_2) has no potential moves having type a, priority level l, and derivative term in C_1 (C_2), or $d-1$ is the maximum depth of the inferences of the potential moves for E_1 (E_2) having type a, priority level l, and derivative term in C_1 (C_2). Since

$$Rate(E,a,l,C) = \begin{cases} Split(Rate(E_1,a,l,C_1),p_{E_2})\ Aggr\ Split(Rate(E_2,a,l,C_2),p_{E_1}) \\ Split(Rate(E_1,a,l,C_1),p_{E_2}) \\ Split(Rate(E_2,a,l,C_2),p_{E_1}) \\ \bot \end{cases}$$

depending on whether $Rate(E_2,a,-1,C_2) \neq \bot \wedge Rate(E_1,a,-1,C_1) \neq \bot$, $Rate(E_2,a,-1,C_2) \neq \bot \wedge Rate(E_1,a,-1,C_1) = \bot$, $Rate(E_2,a,-1,C_2) = \bot \wedge Rate(E_1,a,-1,C_1) \neq \bot$, or $Rate(E_2,a,-1,C_2) = \bot \wedge Rate(E_1,a,-1,C_1) = \bot$, where p_{E_1} (p_{E_2}) is the ratio of the number of passive potential moves with type a from E_1 (E_2) to C_1 (C_2) to the number of passive potential moves with type a from E_1 (E_2), and similarly for Q, $dec_{\mathrm{lab}}(E_1)$, and $dec_{\mathrm{lab}}(E_2)$, the result follows by the induction hypothesis. Note that the result holds thanks to the marking dependent factors f_1 and f_2.

– Let $E \equiv A$ with $A \triangleq E'$. Since d is the maximum depth of the inferences of the potential moves for E having type a, priority level l, and derivative term in C, $d-1$ is the maximum depth of the inferences of the potential moves for E' having type a, priority level l, and derivative term in C. From the induction hypothesis it follows that $Rate(E,a,l,C) = Rate(E',a,l,C) \leq Rate(dec_{\mathrm{lab}}(E'),a,l,C) = Rate(Q,a,l,C)$.

We now show that $Rate(Q,a,l,C) \leq Rate(E,a,l,C)$ for all $a \in AType$, $l \in APLev$, and $C \in (\mathcal{G} \cup \mathcal{M}u_{\mathrm{fin}}(\mathcal{V}_{\mathrm{lab}}))/\mathcal{B}'$. If $Rate(Q,a,l,C) = \bot$ there is nothing to prove, otherwise we proceed by induction on the depth d of the inference to obtain $Q = dec_{\mathrm{lab}}(E) \oplus K$ from E.

- If $d = 1$ then either $dec_{\mathrm{lab}}(E) = \{\!|\ \emptyset{<}a,\tilde{\lambda}{>}.E'\ |\!\}$ or $dec_{\mathrm{lab}}(E) = \{\!|\ \emptyset(E_1 + E_2)\ |\!\}$ with $E_1 + E_2$ guarded. The result trivially follows.

- If $d > 1$ then several subcases arise which are treated in a way similar to the first part of the proof. ■

Proof of Lemma 8.2: We must prove that $\mathcal{R}' \subseteq CLB(\mathcal{R}')$. Given $(O_1, O_2) \in \mathcal{B}^\rho$, i.e. assumed that there exists β such that $\rho(\beta) = \mathit{true} \wedge (O_1, O_2) \in \mathcal{B}^\beta$, let us demonstrate that $(O_1, O_2) \in CLB(\mathcal{R}')^\rho$.

- Let $\alpha \in AType_{\mathrm{U}}$. Given $l \in APLev_{\mathrm{vp}}$ and $C \in \mathcal{O}/\mathcal{B}^\rho$, we must prove that

$$Rate_{\mathrm{CL},\rho}(O_1, \alpha, l, C, \underline{x}, \underline{x}) = Rate_{\mathrm{CL},\rho}(O_2, \alpha, l, C, \underline{x}, \underline{x})$$

  with $\underline{x} \in \underline{Var}$ arbitrary. We observe that, for every transition considered in $Rate_{\mathrm{CL},\rho}(O_1, \alpha, l, C, \underline{x}, \underline{x})$ ($Rate_{\mathrm{CL},\rho}(O_2, \alpha, l, C, \underline{x}, \underline{x})$), by virtue of Lemma 8.1 there is an analogous symbolic late transition with boolean guard β_1 (β_2) such that $\rho(\beta_1) = \mathit{true}$ ($\rho(\beta_2) = \mathit{true}$). Since $(O_1, O_2) \in \mathcal{B}^\beta$ and $\mathcal{R}$ is a strong SLEMB, there must exist $B \subseteq BExp$ such that for all $\beta' \in B$ and $C' \in \mathcal{O}/\mathcal{B}^{\beta'}$

$$Rate_{\mathrm{SL}}(O_1, \alpha, l, C', \beta, B, \beta', \underline{x}, \underline{x}) = Rate_{\mathrm{SL}}(O_2, \alpha, l, C', \beta, B, \beta', \underline{x}, \underline{x})$$

  where the synbolic late transitions above are considered in $Rate_{\mathrm{SL}}(O_1, \alpha, l, C', \beta, B, \beta', \underline{x}, \underline{x})$ ($Rate_{\mathrm{SL}}(O_2, \alpha, l, C', \beta, B, \beta', \underline{x}, \underline{x})$). Since $\rho(\beta) = \mathit{true}$ by the initial hypothesis and $\rho(\beta_1) = \mathit{true}$, we have $\rho(\beta \wedge \beta_1) = \mathit{true}$. From $\beta \wedge \beta_1 \Longrightarrow \bigvee B$ in the definition of $Rate_{\mathrm{SL}}$, it follows that it must exist $\beta'_0 \in B$ such that $\rho(\beta'_0) = \mathit{true}$. From this fact and the definition of $\mathcal{B}^\rho$ it follows that every equivalence class w.r.t. $\mathcal{B}^\rho$ is the union of some equivalence classes w.r.t. $\mathcal{B}^{\beta'_0}$, because from $(O'_1, O'_2) \in \mathcal{B}^{\beta'_0}$ it follows that $(O'_1, O'_2) \in \mathcal{B}^\rho$. If we rewrite C as $\cup_{i \in I} C'_i$ where every such $C'_i \in \mathcal{O}/\mathcal{B}^{\beta'_0}$, by virtue of the fact that $\mathcal{R}$ is a strong SLEMB and Lemma 8.1 we obtain

$$Rate_{\mathrm{CL},\rho}(O_1, \alpha, l, C, \underline{x}, \underline{x}) = Aggr\{\!|\, Rate_{\mathrm{SL}}(O_1, \alpha, l, C'_i, \beta, B, \beta'_0, \underline{x}, \underline{x}) \mid i \in I \,|\!\} =$$
$$Aggr\{\!|\, Rate_{\mathrm{SL}}(O_2, \alpha, l, C', \beta, B, \beta', \underline{x}, \underline{x}) \mid i \in I \,|\!\} = Rate_{\mathrm{CL},\rho}(O_2, \alpha, l, C, \underline{x}, \underline{x})$$

- Let $a \in AName_{\mathrm{IO}}$. Given $\underline{x}, \underline{y} \in \underline{Var}$, $\underline{v} \in \underline{Val}$, $l \in APLev_{\mathrm{vp}}$, and $C \in \mathcal{O}/\mathcal{B}^{\rho[\underline{z} \mapsto \underline{v}]}$, we must prove that

$$Rate_{\mathrm{CL},\rho}(O_1, a?(\underline{x}), l, C, \underline{x}, \underline{z}) = Rate_{\mathrm{CL},\rho}(O_2, a?(\underline{y}), l, C, \underline{y}, \underline{z})$$

  with $\underline{z} \in \underline{Var}$ fresh. The proof is similar to that of the previous case, with in addition the observation that $\rho[\underline{z} \mapsto \underline{v}](\beta'_0) = \mathit{true}$ as $\underline{z}$ is fresh.

- Let $\alpha = a!(\underline{v}) \in AType'_{\mathrm{O}}$. Given $l \in APLev_{\mathrm{vp}}$ and $C \in \mathcal{O}/\mathcal{B}^\rho$, we must prove that

$$Rate_{\mathrm{CL},\rho}(O_1, \alpha, l, C, \underline{x}, \underline{x}) = Rate_{\mathrm{CL},\rho}(O_2, \alpha, l, C, \underline{x}, \underline{x})$$

  with $\underline{x} \in \underline{Var}$ arbitrary. The proof is similar to that of the first case, with in addition the observation that $\rho(\underline{e}_1) = \rho(\underline{e}_2) = \underline{v}$ because $\beta'_0 \Longrightarrow \underline{e}_1 = \underline{e}_2$. ■

Proof of Lemma 8.3: We must prove that $\mathcal{R}' \subseteq SLB(\mathcal{R}')$. Given $(O_1, O_2) \in \mathcal{B}^\beta$, i.e. assumed that $(O_1, O_2) \in \mathcal{B}^\rho$ for all ρ such that $\rho(\beta) = \mathit{true}$, let us demonstrate that $(O_1, O_2) \in SLB(\mathcal{R}')^\beta$, i.e. that there exists $B \subseteq BExp$ such that for all $\beta' \in B$ a certain property holds.

- Let $\alpha \in AType_{\mathrm{U}}$. Given $l \in APLev_{\mathrm{vp}}$ and $C \in \mathcal{O}/\mathcal{B}^{\beta'}$, we must prove that

$$Rate_{\mathrm{SL}}(O_1, \alpha, l, C, \beta, B, \beta', \underline{x}, \underline{x}) = Rate_{\mathrm{SL}}(O_2, \alpha, l, C, \beta, B, \beta', \underline{x}, \underline{x})$$

  Let B be the family of boolean expressions of the form

$$\bigvee\{\beta_1 \wedge \beta_2 \mid \exists \tilde{\lambda}_1, \tilde{\lambda}_2, O_1', O_2'.\, O_1 \xrightarrow{\alpha, \tilde{\lambda}_1, \beta_1}_{\mathrm{SL}} O_1' \wedge O_2 \xrightarrow{\alpha, \tilde{\lambda}_2, \beta_2}_{\mathrm{SL}} O_2' \wedge PL(<\alpha, \tilde{\lambda}_1>) = PL(<\alpha, \tilde{\lambda}_2>) \wedge$$
$$\forall \rho.\, \rho(\beta) = true \Longrightarrow \rho(\beta_1) = true \wedge \rho(\beta_2) = true \wedge (O_1', O_2') \in \mathcal{B}^{\rho}\}$$

  and let us prove that the property above holds for such a B. Given ρ such that $\rho(\beta) = true$, for every transition considered in $Rate_{\mathrm{SL}}(O_1, \alpha, l, C, \beta, B, \beta', \underline{x}, \underline{x})$ ($Rate_{\mathrm{SL}}(O_2, \alpha, l, C, \beta, B, \beta', \underline{x}, \underline{x})$) there is an analogous concrete late transition by virtue of Lemma 8.1. Since $(O_1, O_2) \in \mathcal{B}^{\rho}$ and $\mathcal{R}$ is a strong CLEMB, for all $C' \in \mathcal{O}/\mathcal{B}^{\rho}$ such concrete late transitions are related as follows

$$Rate_{\mathrm{CL},\rho}(O_1, \alpha, l, C', \underline{x}, \underline{x}) = Rate_{\mathrm{CL},\rho}(O_2, \alpha, l, C', \underline{x}, \underline{x})$$

  Since $\mathcal{B}^{\beta} = \bigcap_{\rho(\beta)=true} \mathcal{B}^{\rho}$ and every such $\mathcal{B}^{\rho}$ is a strong CLEMB, the result follows from the construction of B.

- Let $a \in AName_{\mathrm{IO}}$. Given $\underline{x}, \underline{y} \in \underline{Var}$, $l \in APLev_{\mathrm{vp}}$, and $C \in \mathcal{O}/\mathcal{B}^{\beta'}$, we must prove that

$$Rate_{\mathrm{SL}}(O_1, a?(\underline{x}), l, C, \beta, B, \beta', \underline{x}, \underline{z}) = Rate_{\mathrm{SL}}(O_2, a?(\underline{y}), l, C, \beta, B, \beta', \underline{y}, \underline{z})$$

  with $\underline{z} \in \underline{Var}$ fresh. The proof is similar to that of the previous case, with $(O_1', O_2') \in \mathcal{B}^{\rho[\underline{z} \mapsto \underline{v}]}$ in the definition of B.

- Let $a \in AName_{\mathrm{IO}}$. Given $\underline{e}_1, \underline{e}_2 \in \underline{Exp}$, $l \in APLev_{\mathrm{vp}}$, and $C \in \mathcal{O}/\mathcal{B}^{\beta'}$, we must prove that

$$\begin{array}{rcl} Rate_{\mathrm{SL}}(O_1, a!(\underline{e}_1), l, C, \beta, B, \beta', \underline{x}, \underline{x}) & = & Rate_{\mathrm{SL}}(O_2, a!(\underline{e}_2), l, C, \beta, B, \beta', \underline{x}, \underline{x}) \\ \beta' & \Longrightarrow & \underline{e}_1 = \underline{e}_2 \end{array}$$

  with $\underline{x} \in \underline{Var}$ arbitrary. The proof is similar to that of the first case, where in the definition of B we additionally require $\rho(\underline{e}_1) = \rho(\underline{e}_2) = \underline{v}$. ■

References

[1] L. Aceto, M.C.B. Hennessy, *"Adding Action Refinement to a Finite Process Algebra"*, in Information and Computation 115:179-247, 1994

[2] M. Ajmone Marsan, *"Stochastic Petri Nets: An Elementary Introduction"*, in *Advances in Petri Nets '89*, LNCS 424:1-29, 1990

[3] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, G. Franceschinis, *"Modelling with Generalized Stochastic Petri Nets"*, John Wiley & Sons, 1995

[4] M. Ajmone Marsan, A. Bianco, L. Ciminiera, R. Sisto, A. Valenzano, *"A LOTOS Extension for the Performance Analysis of Distributed Systems"*, in IEEE/ACM Trans. on Networking 2:151-164, 1994

[5] R. Allen, D. Garlan, *"A Formal Basis for Architectural Connection"*, in ACM Trans. on Software Engineering and Methodology 6:213-249, 1997

[6] J.C.M. Baeten, J.A. Bergstra, J.W. Klop, *"Syntax and Defining Equations for an Interrupt Mechanism in Process Algebra"*, in Fundamenta Informaticae IX:127-168, 1986

[7] J.C.M. Baeten, J.A. Bergstra, S.A. Smolka, *"Axiomatizing Probabilistic Processes: ACP with Generative Probabilities"*, in Information and Computation 121:234-255, 1995

[8] J.C.M. Baeten, W.P. Weijland, *"Process Algebra"*, Cambridge University Press, 1990

[9] K.A. Bartlett, R.A. Scantlebury, P.T. Wilkinson, *"A Note on Reliable Full-Duplex Transmission over Half-Duplex Links"*, in Comm. of the ACM 12:260-261, 1969

[10] M. Bernardo, *"On the Coexistence of Exponential, Immediate and Passive Actions in EMPA"*, in [150], pp. 58-76

[11] M. Bernardo, *"A Methodology Based on EMPA for Modeling and Simulating Concurrent Systems"*, in Proc. of the *Annual Conf. of the Italian Society for Computer Simulation (ISCS '96)*, pp. 146-151, Rome (Italy), 1996

[12] M. Bernardo, *"Enriching EMPA with Value Passing: A Symbolic Approach based on Lookahead"*, in [151], pp. 35-49

[13] M. Bernardo, *"Two Exercises with EMPA: Computing the Utilization of the CSMA/CD Protocol and Assessing the Performability of a Queueing System"*, in Proc. of the *2nd ERCIM Int. Workshop on Formal Methods for Industrial Critical Systems (FMICS '97)*, pp. 5-17, Cesena (Italy), 1997

[14] M. Bernardo, *"An Algebra-Based Method to Associate Rewards with EMPA Terms"*, in Proc. of the *24th Int. Coll. on Automata, Languages and Programming (ICALP '97)*, LNCS 1256:358-368, Bologna (Italy), 1997

[15] M. Bernardo, *"Using EMPA for the Performance Evaluation of an ATM Switch"*, in Proc. of the *3rd ERCIM Int. Workshop on Formal Methods for Industrial Critical Systems (FMICS '98)*, CWI, pp. 43-57, Amsterdam (The Netherlands), 1998

[16] M. Bernardo, *"Formal Specification of Performance Measures for Process Algebra Models of Concurrent Systems"*, Tech. Rep. UBLCS-98-08, University of Bologna (Italy), 1998

[17] M. Bernardo, *"Value Passing in Stochastically Timed Process Algebras: A Symbolic Approach based on Lookahead"*, Tech. Rep. UBLCS-98-10, University of Bologna (Italy), 1998

[18] M. Bernardo, M. Bravetti, *"Functional and Performance Modeling and Analysis of Token Ring using EMPA"*, in Proc. of the *6th Italian Conf. on Theoretical Computer Science (ICTCS '98)*, World Scientific, pp. 204-215, Prato (Italy), 1998

[19] M. Bernardo, N. Busi, R. Gorrieri, *"A Distributed Semantics for EMPA Based on Stochastic Contextual Nets"*, in [149], pp. 492-509

[20] M. Bernardo, N. Busi, M. Ribaudo, *"Compact Net Semantics for Process Algebras"*, Tech. Rep., University of Bologna (Italy), in preparation

[21] M. Bernardo, P. Ciancarini, L. Donatiello, *"Performance Analysis of Software Architectures via a Process Algebraic Description Language"*, Tech. Rep., University of Bologna (Italy), in preparation

[22] M. Bernardo, W.R. Cleaveland, *"A Theory of Efficiency for Markovian Processes"*, Tech. Rep. UBLCS-99-02, University of Bologna (Italy), 1999

[23] M. Bernardo, W.R. Cleaveland, S.T. Sims, W.J. Stewart, *"TwoTowers: A Tool Integrating Functional and Performance Analysis of Concurrent Systems"*, in Proc. of the *IFIP Joint Int. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing and Verification (FORTE/PSTV '98)*, Kluwer, pp. 457-467, Paris (France), 1998

[24] M. Bernardo, L. Donatiello, R. Gorrieri, *"Integrated Analysis of Concurrent Distributed Systems using Markovian Process Algebra"*, in Proc. of the *7th Int. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE '94)*, Chapman & Hall, pp. 455-457, Berne (Switzerland), 1994

[25] M. Bernardo, L. Donatiello, R. Gorrieri, *"Giving a Net Semantics to Markovian Process Algebra"*, in Proc. of the *6th Int. Workshop on Petri Nets and Performance Models (PNPM '95)*, IEEE-CS Press, pp. 169-178, Durham (NC), 1995

[26] M. Bernardo, L. Donatiello, R. Gorrieri, *"A Stochastic Process Algebra Model for the Analysis of the Alternating Bit Protocol"*, in Proc. of the *11th Int. Symp. on Computer and Information Sciences (ISCIS XI)*, METU, pp. 375-384, Antalya (Turkey), 1996

[27] M. Bernardo, L. Donatiello, R. Gorrieri, *"A Formal Approach to the Integration of Performance Aspects in the Modeling and Analysis of Concurrent Systems"*, in Information and Computation 144:83-154, 1998

[28] M. Bernardo, R. Gorrieri, *"Extended Markovian Process Algebra"*, in Proc. of the *7th Int. Conf. on Concurrency Theory (CONCUR '96)*, LNCS 1119:315-330, Pisa (Italy), 1996

[29] M. Bernardo, R. Gorrieri, *"A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time"*, in Theoretical Computer Science 202:1-54, 1998

[30] M. Bernardo, R. Gorrieri, M. Roccetti, *"Formal Performance Modelling and Evaluation of an Adaptive Mechanism for Packetised Audio over the Internet"*, in Formal Aspects of Computing 10:313-337, 1999

[31] G.V. Bochmann, J. Vaucher, *"Adding Performance Aspects to Specification Languages"*, in Proc. of the *8th Int. Conf. on Protocol Specification, Testing and Verification (PSTV VIII)*, North Holland, pp. 19-31, Atlantic City (NJ), 1988

[32] H.C. Bohnenkamp, B.R. Haverkort, *"Semi-Numerical Solution of Stochastic Process Algebra Models"*, in [152], pp. 71-84

[33] T. Bolognesi, E. Brinksma, *"Introduction to the ISO Specification Language LOTOS"*, in Computer Networks and ISDN Systems 14:25-59, 1987

[34] J. Bolot, H. Crepin, A. Vega Garcia, *"Analysis of Audio Packet Loss on the Internet"*, in Proc. of *Network and Operating System Support for Digital Audio and Video*, pp. 163-174, Durham (NC), 1995

[35] G. Boudol, I. Castellani, *"A Non-interleaving Semantics for CCS Based on Proved Transitions"*, in Fundamenta Informaticae XI:433-452, 1988

[36] H. Bowman, J.W. Bryans, J. Derrick, *"Analysis of a Multimedia Stream using Stochastic Process Algebra"*, in [152], pp. 51-69

[37] M. Bravetti, M. Bernardo, *"Compositional Asymmetric Cooperations for Process Algebras with Probabilities, Priorities, and Time"*, Tech. Rep., University of Bologna (Italy), in preparation

[38] M. Bravetti, M. Bernardo, R. Gorrieri, *"Towards Performance Evaluation with General Distributions in Process Algebras"*, in Proc. of the *9th Int. Conf. on Concurrency Theory (CONCUR '98)*, LNCS 1466:405-422, Nice (France), 1998

[39] M. Bravetti, M. Bernardo, R. Gorrieri, *"A Note on the Congruence Proof for Recursion in Markovian Bisimulation Equivalence"*, in [152], pp. 153-164

[40] E. Brinksma, J.-P. Katoen, R. Langerak, D. Latella, *"A Stochastic Causality-Based Process Algebra"*, in [149], pp. 553-565

[41] P. Buchholz, *"Markovian Process Algebra: Composition and Equivalence"*, in [148], pp. 11-30

[42] J.E. Burns, *"Mutual Exclusion with Linear Waiting using Binary Shared Variables"*, in ACM SIGACT News 10:42-47, 1978

[43] N. Busi, R. Gorrieri, *"A Petri Net Semantics for π-Calculus"*, in Proc. of the *6th Int. Conf. on Concurrency Theory (CONCUR '95)*, LNCS 962:145-159, Philadelphia (PA), 1995

[44] J. Camilleri, G. Winskel, *"CCS with Priority Choice"*, in Information and Computation 116:26-37, 1995

[45] G. Chiola, C. Anglano, J. Campos, J.M. Colom, M. Silva, *"Operational Analysis of Timed Petri Nets and Applications to the Computation of Performance Bounds"*, in *Quantitative Methods in Parallel Systems*, Springer, pp. 161-174, 1995

[46] G. Chiola, G. Franceschinis, R. Gaeta, M. Ribaudo, *"GreatSPN 1.7: Graphical Editor and Analyzer for Timed and Stochastic Petri Nets"*, in Performance Evaluation 24:47-68, 1995

[47] G. Ciardo, J. Muppala, K.S. Trivedi, *"On the Solution of GSPN Reward Models"*, in Performance Evaluation 12:237-253, 1991

[48] G. Clark, *"Formalising the Specification of Rewards with PEPA"*, in [150], pp. 139-160

[49] G. Clark, S. Gilmore, J. Hillston, *"Specifying Performance Measures for PEPA"*, to appear in Proc. of the *5th AMAST Int. Workshop on Formal Methods for Real Time and Probabilistic Systems (ARTS '99)*, Bamberg (Germany), 1999

[50] E.M. Clarke, O. Grumberg, D.A. Peled, *"Model Checking"*, MIT Press, 1999

[51] W.R. Cleaveland, M.C.B. Hennessy, *"Priorities in Process Algebras"*, in Information and Computation 87:58-77, 1990

[52] W.R. Cleaveland, I. Lee, P. Lewis, S.A. Smolka, *"A Theory of Testing for Soft Real-Time Processes"*, in Proc. of the *8th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE '96)*, pp. 474-479, Lake Tahoe (NE), 1996

[53] W.R. Cleaveland, G. Lüttgen, V. Natarajan, *"A Process Algebra with Distributed Priorities"*, in Proc. of the *7th Int. Conf. on Concurrency Theory (CONCUR '96)*, LNCS 1119:34-49, Pisa (Italy), 1996

[54] W.R. Cleaveland, E. Madelaine, S.T. Sims, *"A Front-End Generator for Verification Tools"*, in Proc. of the *1st Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '95)*, LNCS 1019:153-173, Aarhus (Denmark), 1995

[55] W.R. Cleaveland, J. Parrow, B. Steffen, *"The Concurrency Workbench: A Semantics-Based Tool for the Verification of Concurrent Systems"*, in ACM Trans. on Programming Languages and Systems 15:36-72, 1993

[56] W.R. Cleaveland, S.T. Sims, *"The NCSU Concurrency Workbench"*, in Proc. of the *8th Int. Conf. on Computer Aided Verification (CAV '96)*, LNCS 1102:394-397, New Brunswick (NJ), 1996

[57] D. Cohen, *"Issues in Transnet Packetized Voice Communications"*, in Proc. of the *5th Data Communication Symp.*, pp. 6.10 - 6.13, Snowbird (UT), 1977

[58] F. Corradini, R. Gorrieri, M. Roccetti, *"Performance Preorder and Competitive Equivalence"*, in Acta Informatica 34:805-835, 1997

[59] P.J. Courtois, *"Decomposability: Queueing and Computer System Applications"*, Academic Press, 1977

[60] P.R. D'Argenio, J.-P. Katoen, E. Brinksma, *"An Algebraic Approach to the Specification of Stochastic Systems"*, in Proc. of the *IFIP Working Conf. on Programming Concepts, Methods and Calculi (PROCOMET '98)*, Chapman & Hall, pp. 126-147, Shelter Islands (NY), 1998

[61] L. De Alfaro, *"Stochastic Transition Systems"*, in Proc. of the *9th Int. Conf. on Concurrency Theory (CONCUR '98)*, LNCS 1466:423-438, Nice (France), 1998

[62] P. Degano, R. De Nicola, U. Montanari, *"A Distributed Operational Semantics for CCS Based on Condition/Event Systems"*, in Acta Informatica 26:59-91, 1988

[63] R. De Nicola, M.C.B. Hennessy, *"Testing Equivalences for Processes"*, in Theoretical Computer Science 34:83-133, 1983

[64] C. Derman, *"Finite State Markovian Decision Processes"*, Academic Press, 1970

[65] E.W. Dijkstra, *"Solution of a Problem in Concurrent Programming Control"*, in Comm. of the ACM 8:569, 1965

[66] D. Ferrari, *"Considerations on the Insularity of Performance Evaluation"*, in IEEE Trans. on Software Engineering 12:678-683, 1986

[67] M.J. Fischer, N.A. Lynch, J.E. Burns, A. Borodin, *"Distributed FIFO Allocation of Identical Resources using Small Shared Space"*, in ACM Trans. on Programming Languages and Systems 11:90-114, 1989

[68] R. Gerber, I. Lee, *"A Resource-Based Prioritized Bisimulation for Real-Time Systems"*, in Information and Computation 113:102-142, 1994

[69] S. Gilmore, J. Hillston, *"The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling"*, in Proc. of the *7th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation (PERFORMANCE TOOLS '94)*, LNCS 794:353-368, Wien (Austria), 1994

[70] S. Gilmore, J. Hillston, D.R.W. Holton, M. Rettelbach, *"Specifications in Stochastic Process Algebra for a Robot Control Problem"*, in Journal of Production Research 34:1065-1080, 1996

[71] R.J. van Glabbeek, S.A. Smolka, B. Steffen, *"Reactive, Generative and Stratified Models of Probabilistic Processes"*, in Information and Computation 121:59-80, 1995

[72] R.J. van Glabbeek, F.W. Vaandrager, *"Petri Net Models for Algebraic Theories of Concurrency"*, in Proc. of the *Conf. on Parallel Architectures and Languages Europe (PARLE '87)*, LNCS 259:224-242, Eindhoven (The Netherlands), 1987

[73] N. Götz, *"Stochastische Prozeßalgebren – Integration von funktionalem Entwurf und Leistungsbewertung Verteilter Systeme"*, Ph.D. Thesis, University of Erlangen-Nürnberg (Germany), 1994

[74] N. Götz, U. Herzog, M. Rettelbach, *"TIPP - A Stochastic Process Algebra"*, in [147], pp. 31-36

[75] N. Götz, U. Herzog, M. Rettelbach, *"Multiprocessor and Distributed System Design: The Integration of Functional Specification and Performance Analysis Using Stochastic Process Algebras"*, in Proc. of the *16th Int. Symp. on Computer Performance Modelling, Measurement and Evaluation (PERFORMANCE '93)*, LNCS 729:121-146, Rome (Italy), 1993

[76] P.G. Harrison, J. Hillston, *"Exploiting Quasi-Reversible Structures in Markovian Process Algebra Models"*, in [149], pp. 510-520

[77] P.G. Harrison, B. Strulo, *"Stochastic Process Algebra for Discrete Event Simulation"*, in *Quantitative Methods in Parallel Systems*, Springer, pp. 18-37, 1995

[78] C. Harvey, *"Performance Engineering as an Integral Part of System Design"*, in BT Technology Journal 4:143-147, 1986

[79] B.R. Haverkort, K.S. Trivedi, *"Specification Techniques for Markov Reward Models"*, in Discrete Event Dynamic Systems: Theory and Applications 3:219-247, 1993

[80] M.C.B. Hennessy, *"Algebraic Theory of Processes"*, MIT Press, 1988

[81] M.C.B. Hennessy, H. Lin, *"Symbolic Bisimulations"*, in Theoretical Computer Science 138:353-389, 1995

[82] M.C.B. Hennessy, R. Milner, *"Algebraic Laws for Nondeterminism and Concurrency"*, in Journal of the ACM 32:137-161, 1985

[83] H. Hermanns, *"Leistungsvorhersage von Verhaltensbeschreibungen mittels Temporaler Logik"*, contribution to the *GI/ITG Fachgespraech '95: Formale Beschreibungstechniken fuer Verteilte Systeme*, 1995

[84] H. Hermanns, *"An Operator for Symmetry Representation and Exploitation in Stochastic Process Algebras"*, in [151], pp. 55-70

[85] H. Hermanns, *"Interactive Markov Chains"*, Ph.D. Thesis, University of Erlangen-Nürnberg (Germany), 1998

[86] H. Hermanns, U. Herzog, J. Hillston, V. Mertsiotakis, M. Rettelbach, *"Stochastic Process Algebras: Integrating Qualitative and Quantitative Modelling"*, Tech. Rep. 11/94, University of Erlangen-Nürnberg (Germany), 1994

[87] H. Hermanns, U. Herzog, V. Mertsiotakis, *"Stochastic Process Algebras as a Tool for Performance and Dependability Modelling"*, in Proc. of the *1st IEEE Int. Computer Performance and Dependability Symp. (IPDS '95)*, IEEE-CS Press, pp. 102-111, Erlangen (Germany), 1995

[88] H. Hermanns, J.-P. Katoen, *"Automated Compositional Markov Chain Generation for a Plain-Old Telephone System"*, to appear in Science of Computer Programming, 1999

[89] H. Hermanns, M. Lohrey, *"Priority and Maximal Progress Are Completely Axiomatisable"*, in Proc. of the *9th Int. Conf. on Concurrency Theory (CONCUR '98)*, LNCS 1466:237-252, Nice (France), 1998

[90] H. Hermanns, V. Mertsiotakis, M. Rettelbach, *"Performance Analysis of Distributed Systems Using TIPP - A Case Study"*, in Proc. of the *10th UK Performance Engineering Workshop for Computer and Telecommunication Systems*, Edinburgh (UK), 1994

[91] H. Hermanns, V. Mertsiotakis, M. Rettelbach, *"A Construction and Analysis Tool Based on the Stochastic Process Algebra TIPP"*, in Proc. of the *2nd Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '96)*, LNCS 1055:427-430, Passau (Germany), 1996

[92] H. Hermanns, M. Rettelbach, *"Syntax, Semantics, Equivalences, and Axioms for MTIPP"*, in [148], pp. 71-87

[93] H. Hermanns, M. Rettelbach, *"Towards a Superset of Basic LOTOS for Performance Prediction"*, in [150], pp. 77-94

[94] H. Hermanns, M. Rettelbach, T. Weiß, *"Formal Characterisation of Immediate Actions in SPA with Nondeterministic Branching"*, in [149], pp. 530-541

[95] H. Hermanns, M. Siegle, *"Computing Bisimulations for Stochastic Process Algebras using Symbolic Representations"*, in [152], pp. 103-118

[96] U. Herzog, *"Formal Description, Time and Performance Analysis – A Framework"*, in *Entwurf und Betrieb verteilter Systeme*, Informatik Fachberichte 264, Springer, 1990

[97] U. Herzog, *"EXL: Syntax, Semantics and Examples"*, Tech. Rep. 16/90, University of Erlangen-Nürnberg (Germany), 1990

[98] U. Herzog, *"A Concept for Graph-Based Stochastic Process Algebras, Generally Distributed Activity Times, and Hierarchical Modelling"*, in [150], pp. 1-20

[99] D.P. Heyman, M.J. Sobel, *"Stochastic Models in Operations Research"*, McGraw-Hill, 1982

[100] J. Hillston, *"A Compositional Approach to Performance Modelling"*, Ph.D. Thesis, University of Edinburgh (UK), 1994, Cambridge University Press, 1996

[101] J. Hillston, *"The Nature of Synchronisation"*, in [148], pp. 51-70

[102] J. Hillston, *"Exploiting Structure in Solution: Decomposing Composed Models"*, in [152], pp. 1-15

[103] J. Hillston, V. Mertsiotakis, *"A Simple Time Scale Decomposition Technique for Stochastic Process Algebras"*, in [149], pp. 566-577

[104] J. Hillston, N. Thomas, *"Product Form Solution for a Class of PEPA Models"*, in Proc. of the *3rd IEEE Int. Computer Performance and Dependability Symp. (IPDS '98)*, IEEE-CS Press, Durham (NC), 1998

[105] J. Hillston, N. Thomas, *"A Syntactical Analysis of Reversible PEPA Models"*, in [152], pp. 37-49

[106] C.A.R. Hoare, *"Communicating Sequential Processes"*, Prentice Hall, 1985

[107] D.R.W. Holton, *"A PEPA Specification of an Industrial Production Cell"*, in [149], pp. 542-551

[108] R.A. Howard, *"Dynamic Probabilistic Systems"*, John Wiley & Sons, 1971

[109] L. Jenner, W. Vogler, *"Comparing the Efficiency of Asynchronous Systems"*, in [152], pp. 137-151

[110] S.C. Johnson, *"YACC - Yet Another Compiler Compiler"*, Computing Science Tech. Rep. 32, AT&T Bell Labs, Murray Hill (NJ), 1975

[111] C.-C. Jou, S.A. Smolka, *"Equivalences, Congruences, and Complete Axiomatizations for Probabilistic Processes"*, in Proc. of the *1st Int. Conf. on Concurrency Theory (CONCUR '90)*, LNCS 458:367-383, Amsterdam (The Netherlands), 1990

[112] K. Kanani, *"A Unified Framework for Systematic Quantitative and Qualitative Analysis of Communicating Systems"*, Ph.D. Thesis, Imperial College (UK), 1998

[113] J.-P. Katoen *"Quantitative and Qualitative Extensions of Event Structures"*, Ph.D. Thesis, University of Twente (The Netherlands), 1996

[114] B.W. Kernighan, D.M. Ritchie, *"The C Programming Language"*, Prentice Hall, 1988

[115] L. Kleinrock, *"Queueing Systems"*, John Wiley & Sons, 1975

[116] L. Lamport, *"A Fast Mutual Exclusion Algorithm"*, in ACM Trans. on Computer Systems 5:1-11, 1987

[117] K.G. Larsen, A. Skou, *"Bisimulation through Probabilistic Testing"*, in Information and Computation 94:1-28, 1991

[118] K.G. Larsen, A. Skou, *"Compositional Verification of Probabilistic Processes"*, in Proc. of the *3rd Int. Conf. on Concurrency Theory (CONCUR '92)*, LNCS 630:456-471, Stony Brook (NY), 1992

[119] D. Lehmann, M. Rabin, *"On the Advantage of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem"*, in Proc. of the *8th ACM Symp. on Principles of Programming Languages (POPL '81)*, ACM Press, pp. 133-138, 1981

[120] M.E. Lesk, *"Lex - A Lexical Analyzer Generator"*, Computing Science Tech. Rep. 39, AT&T Bell Labs, Murray Hill (NJ), 1975

[121] Z. Li, H. Chen, *"Computing Strong/Weak Bisimulation Equivalences and Observation Congruence for Value-Passing Processes"*, to appear in Proc. of the *5th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99)*, Amsterdam (The Netherlands), 1999

[122] H. Lin, *"Symbolic Transition Graph with Assignment"*, in Proc. of the *7th Int. Conf. on Concurrency Theory (CONCUR '96)*, LNCS 1119:50-65, Pisa (Italy), 1996

[123] J.L. Lions, *"Ariane 5 – Flight 501 Failure – Report by the Inquiry Board"*, http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html, 1996

[124] N.A. Lynch, *"Distributed Algorithms"*, Morgan Kaufmann, 1996

[125] N.A. Lynch, M.R. Tuttle, *"Hierarchical Correctness Proofs for Distributed Algorithms"*, in Proc. of the *6th ACM Symp. on Principles of Distributed Computing (PODC '87)*, ACM Press, pp. 137-151, Vancouver (Canada), 1987

[126] M. Macedonia, D. Brutzmann, *"Mbone Provides Audio and Video across the Internet"*, in IEEE Computer Magazine 21:30-35, 1994

[127] K. Matthes, *"Zur Theorie der Bedienungsprozesse"*, in Trans. of the *3rd Prague Conf. on Information Theory, Stat. Dec. Fns. and Random Processes*, pp. 513-528, 1962

[128] V. Mertsiotakis, *"Time Scale Decomposition of Stochastic Process Algebra Models"*, in [151], pp. 71-93

[129] V. Mertsiotakis, *"Approximate Analysis Methods for Stochastic Process Algebras"*, Ph.D. Thesis, University of Erlangen-Nürnberg (Germany), 1998

[130] V. Mertsiotakis, M. Silva, *"A Throughput Approximation Algorithm for Decision Free Processes"*, in [150], pp. 161-178

[131] J.F. Meyer, *"Performability: A Retrospective and some Pointers to the Future"*, in Performance Evaluation 14:139-156, 1992

[132] C. Miguel, A. Fernandez, L. Vidaller, *"LOTOS Extended with Probabilistic Behaviours"*, in Formal Aspects of Computing 5:253-281, 1993

[133] R. Milner, *"Communication and Concurrency"*, Prentice Hall, 1989

[134] I. Mitrani, A. Ost, M. Rettelbach, *"TIPP and the Spectral Expansion Method"*, in *Quantitative Methods in Parallel Systems*, Springer, pp. 99-113, 1995

[135] F. Moller, C.M.N. Tofts, *"Relating Processes with Respect to Speed"*, in Proc. of the *2nd Int. Conf. on Concurrency Theory (CONCUR '91)*, LNCS 527:424-438, Amsterdam (The Netherlands), 1991

[136] M.K. Molloy, *"Performance Analysis using Stochastic Petri Nets"*, in IEEE Trans. on Computers 31:913-917, 1982

[137] U. Montanari, F. Rossi, *"Contextual Nets"*, in Acta Informatica 32:545-596, 1995

[138] S.B. Moon, J. Kurose, D. Towsley, *"Packet Audio Playout Delay Adjustment: Performance Bounds and Algorithms"*, in ACM Multimedia Systems 6:17-28, 1998

[139] T. Murata, *"Petri Nets: Properties, Analysis and Applications"*, in Proc. of the IEEE 77:541-580, 1989

[140] V. Natarajan, W.R. Cleaveland, *"Predictability of Real-Time Systems: A Process-Algebraic Approach"*, in Proc. of the *17th IEEE Real Time Systems Symp. (RTSS '96)*, IEEE-CS Press, pp. 82-91, Washington (DC), 1996

[141] M.F. Neuts, *"Matrix-Geometric Solutions in Stochastic Models - An Algorithmic Approach"*, John Hopkins University Press, 1981

[142] V.F. Nicola, *"Lumping in Markov Reward Processes"*, Tech. Rep. RC-14719, IBM T.J. Watson Research Center, Yorktown Heights (NY), 1990

[143] X. Nicollin, J. Sifakis, *"An Overview and Synthesis on Timed Process Algebras"*, in Proc. of the *3rd Int. Conf. on Computer Aided Verification (CAV '91)*, LNCS 575:376-398, Aalborg (Denmark), 1991

[144] N. Nounou, Y. Yemini, *"Algebraic Specification-Based Performance Analysis of Communication Protocols"*, in Proc. of the *5th Int. Conf. on Protocol Specification, Testing and Verification (PSTV V)*, North Holland, pp. 541-560, 1985

[145] E.-R. Olderog, *"Nets, Terms and Formulas"*, Cambridge University Press, 1991

[146] J.K. Ousterhout, *"Tcl and the Tk Toolkit"*, Addison-Wesley, 1994

[147] *Proc. of the 1st Int. Workshop on Process Algebra and Performance Modelling (PAPM '93)*, J. Hillston and F. Moller editors, Edinburgh (UK), 1993

[148] *Proc. of the 2nd Int. Workshop on Process Algebra and Performance Modelling (PAPM '94)*, U. Herzog and M. Rettelbach editors, Erlangen (Germany), 1994

[149] *Proc. of the 3rd Int. Workshop on Process Algebra and Performance Modelling (PAPM '95)*, S. Gilmore and J. Hillston editors, Computer Journal 38(7), Edinburgh (UK), 1995

[150] *Proc. of the 4th Int. Workshop on Process Algebra and Performance Modelling (PAPM '96)*, M. Ribaudo editor, CLUT, Torino (Italy), 1996

[151] *Proc. of the 5th Int. Workshop on Process Algebra and Performance Modelling (PAPM '97)*, E. Brinksma and A. Nymeyer editors, Enschede (The Netherlands), 1997

[152] *Proc. of the 6th Int. Workshop on Process Algebra and Performance Modelling (PAPM '98)*, C. Priami editor, Nice (France), 1998

[153] R. Paige, R.E. Tarjan, *"Three Partition Refinement Algorithms"*, in SIAM Journal of Computing 16:973-989, 1987

[154] D. Park, *"Concurrency and Automata on Infinite Sequences"*, in Proc. of the *5th GI-Conf. on Theoretical Computer Science*, LNCS 104:167-183, 1981

[155] G.L. Peterson, *"Myths about the Mutual Exclusion Problem"*, in Information Processing Letters 12:115-116, 1981

[156] G.L. Peterson, M.J. Fischer, *"Economical Solutions for the Critical Section Problem in a Distributed System"*, in Proc. of the *9th ACM Symp. on Theory of Computing*, pp. 91-97, Boulder (CO), 1977

[157] C.A. Petri, *"Communication with Automata"*, Tech. Rep. RADC-TR-65-377, Griffiss Air Force Base (NY), 1966

[158] G.D. Plotkin, *"A Structured Approach to Operational Semantics"*, Tech. Rep. DAIMI-FN-19, Aarhus University (Denmark), 1981

[159] C. Priami, *"Stochastic π-Calculus"*, in [149], pp. 578-589

[160] C. Priami, *"Stochastic π-Calculus with General Distributions"*, in [150], pp. 41-57

[161] C. Priami, *"Stochastic Analysis of Mobile Telephony Networks"*, in [151], pp. 145-171

[162] S. Purushothaman, P.A. Subrahmanyam, *"Reasoning about Probabilistic Behavior in Concurrent Systems"*, in IEEE Trans. on Software Engineering 13:740-745, 1987

[163] M.A. Qureshi, W.H. Sanders, *"Reward Model Solution Methods with Impulse and Rate Rewards: An Algorithm and Numerical Results"*, in Performance Evaluation 20:413-436, 1994

[164] W. Reisig, *"Petri Nets: An Introduction"*, Springer-Verlag, 1985

[165] M. Rettelbach, *"Probabilistic Branching in Markovian Process Algebras"*, in [149], pp. 590-599

[166] M. Rettelbach, *"Stochastische Prozeßalgebren mit zeitlosen Aktivitäten und probabilistischen Verzweigungen"*, Ph.D. Thesis, University of Erlangen-Nürnberg (Germany), 1996

[167] M. Rettelbach, M. Siegle, *"Compositional Minimal Semantics for the Stochastic Process Algebra TIPP"*, in [148], pp. 89-105

[168] M. Ribaudo, *"On the Relationship between Stochastic Process Algebras and Stochastic Petri Nets"*, Ph.D. Thesis, University of Torino (Italy), 1995

[169] M. Roccetti, M. Bernardo, R. Gorrieri, *"Packetized Audio for Industrial Applications: A Simulation Study"*, in Proc. of the *10th European Simulation Symp. (ESS '98)*, SCS International, pp. 495-500, Nottingham (UK), 1998

[170] M. Roccetti, V. Ghini, P. Salomoni, G. Pau, M.E. Bonfigli, *"Design, Development and Experimentation of an Adaptive Mechanism for Reliable Packetized Audio for Use over the Internet"*, to appear in Multimedia Tools and Applications, 1999

[171] W.H. Sanders, J.F. Meyer, *"A Unified Approach for Specifying Measures of Performance, Dependability, and Performability"*, in Dependable Computing and Fault Tolerant Systems 4:215-237, 1991

[172] P. Schweitzer, *"Aggregation Methods for Large Markov Chains"*, in Mathematical Computer Performance and Reliability, North Holland, pp. 275-286, 1984

[173] R. Segala, *"Modeling and Verification of Randomized Distributed Real-Time Systems"*, Ph.D. Thesis, MIT, Boston (MA), 1995

[174] K. Seidel, *"Probabilistic Communicating Processes"*, in Theoretical Computer Science 152:219-249, 1995

[175] M. Sereno, *"Towards a Product Form Solution for Stochastic Process Algebras"*, in [149], pp. 622-632

[176] M. Shaw, D. Garlan, *"Software Architecture: Perspectives on an Emerging Discipline"*, Prentice Hall, 1996

[177] G.S. Shedler, *"Generation Methods for Discrete Event Simulation"*, in *"Computer Performance Modeling Handbook"*, S.S. Lavenberg editor, Academic Press, pp. 223-266, 1983

[178] C.U. Smith, *"Performance Engineering of Software Systems"*, Addison-Wesley, 1990

[179] S.A. Smolka, B. Steffen, *"Priority as Extremal Probability"*, in Formal Aspects of Computing 8:585-606, 1996

[180] W.J. Stewart, *"Introduction to the Numerical Solution of Markov Chains"*, Princeton University Press, 1994

[181] B. Strulo, *"Process Algebra for Discrete Event Simulation"*, Ph.D. Thesis, Imperial College, University of London (UK), 1994

[182] A.S. Tanenbaum, *"Computer Networks"*, Prentice Hall, 1996

[183] N. Thomas, S. Gilmore, *"Applying Quasi-Separability to Markovian Process Algebra"*, in [152], pp. 27-36

[184] C.M.N. Tofts, *"Processes with Probabilities, Priority and Time"*, in Formal Aspects of Computing 6:536-564, 1994

[185] C.M.N. Tofts, *"Compositional Performance Analysis"*, in Proc. of the *3rd Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '97)*, LNCS 1217:290-305, Enschede (The Netherlands), 1997

[186] K.S. Trivedi, J.K. Muppala, S.P. Woolet, B.R. Haverkort, *"Composite Performance and Dependability Analysis"*, in Performance Evaluation 14:197-215, 1992

[187] C.A. Vissers, G. Scollo, M. van Sinderen, E. Brinksma, *"Specification Styles in Distributed Systems Design and Verification"*, in Theoretical Computer Science 89:179-206, 1991

[188] P.D. Welch, *"The Statistical Analysis of Simulation Results"*, in *"Computer Performance Modeling Handbook"*, S.S. Lavenberg editor, Academic Press, pp. 267-329, 1983

[189] S.-H. Wu, S.A. Smolka, E.W. Stark, *"Composition and Behaviors of Probabilistic I/O Automata"*, in Theoretical Computer Science 176:1-38, 1997

[190] Y. Yemini, J. Kurose, *"Towards the Unification of the Functional and Performance Analysis of Protocols, or Is the Alternating-Bit Protocol Really Correct?"*, in Proc. of the *2nd Int. Conf. on Protocol Specification, Testing and Verification (PSTV II)*, North Holland, 1982