University of Torino
Computer Science Department

Doctoral School of Science and High Technology
XXV Cycle

Ph.D. Thesis

States, actions and path properties in Markov chains

Author:
Elvio Gilberto Amparore

Advisor:
Susanna Donatelli

July 22, 2013

Contents

Chapter 1

Introduction and motivations

In the domain of performance and reliability analysis, high-level specification languages have become the standard approach for the formal verification of system properties.

In traditional performance analysis, only a limited number of simple properties are expressible, like the mean number of clients in a system, the probability that a failure happens in a time frame, or some reliability indexes like the *mean time to failure* (MTTF) or the *mean time to repair* (MTTR).

Not all properties have the same characteristics, and it is possible to identify some general classes of properties that share a similar schema, like properties of *correctness*, of *performance*, of *reliability* or *performability.* A property of *correctness* involves the guarantee that the system always satisfies some qualitative properties, like it does always respond (*liveness*), it does not block for invalid input of the user (it is *deadlock-free*), or it performs the requested operation (*safety*). Since these requirements are in some sense a yes-or-no question, they are usually referred to as *functional* or *qualitative* requirements.

In the context of real-time systems, a property does not only have to be true, but also has to respect certain time bounds. These are *performance* properties. Usually, these systems are classified as *hard* real-time, where the time bounds are strict, and *soft* real-time, where the time requirement is expressed within a probability of failure. For instance, the requirement that a hardware controller has to respond to interrupt signals is strict, to avoid malfunctions. On the contrary, a voice-over-IP service that wants to provide a smooth continuous voice stream, can only be modeled as a soft requirement due to the unpredictable behavior of the IP-level packet delivery mechanism, which does not support hard real-time requirements. Typical performance questions include querying the expected waiting time experienced by a customer in queue, the mean number of client waiting, the throughput and the utilization of a service, and so on. These are typically referred to as *quantitative* measures.

A *reliability* property assess that a system is operational and functional with high probability, and that in case of failure the downtime is predictable and short. For instance, a highly redundant transaction system may need to satisfy that at most it has 3 hours of downtime per year, no matter how many power losses, storage failures or network link crashes may happen.

Some systems employ redundancy to achieve a greater reliability, like a multicore system where CPUs may fail. In these scenarios, the system may undergo different degradation steps before failing, and in each step the system could be less able to accomplish its jobs. These requirement are expressed with a *performability* property. Typical performability questions have form like: “How long can the system be expected to work without interruption?” or: “What is the probability that the system operates above a certain level of efficiency during an observation period?”. These questions are a mix of qualitative/quantitative properties, since a certain qualitative property is ensured by having some capacity of the system above/below a certain threshold. Typically, performability questions are answered using a *reward* model.

A certain set of these properties are almost always requirements for many systems in use today in the industry, in medicine, in the software industry, etc. To verify if a system really meets these expectations, there are many methodologies. The system could be designed by only using the experience of the designer, which requires a trial-and-error process to discover what is good and what has to be redone. Usually, systems developed in this way require many iterations to achieve a reasonable functionality, and no guarantee of correctness can be stated on the system. However, for large systems this iterative process has met a large success and is nowadays an established practice in many engineering fields. This process typically involves a test phase to verify that the system actually works. *Testing* has the advantage of verifying a working and realistic system, so the results are realistic. Unfortunately, to test a system at least a prototype has to be developed. If the required properties do not hold, a redesign and reimplementation is often necessary. Also testing cannot provide an exhaustive verification of all the possible cases and scenarios that the system will face.

A different approach could be *simulating* the system before constructing it. Simulation is similar to testing, except that an abstract model of the system is used instead of the real one. The model itself is a simplified description of the system behavior. The advantage lies in the fact that designing an abstract model of a system is usually much cheaper than building the real one. On the other hand, if the modeled system is not descriptive enough or too simplified, the simulation results will be unrealistic. Given a system model, it is also possible to verify the properties with some kind of *formal verification*, which involves the generation of a formal proof of correctness of the property. Formal verification requires both the system and the properties to be specified with a formal language. A verification method is then used to prove that the property holds in every possible configuration of the system. Techniques for formal verification includes *proof-based* verification, where the system is provided as a set of logical formulae, and *model-based* verification, where the system is described as a transition system between discrete or continuous states.

It should be clear that while direct system design and testing are highly empirical and depend heavily on the capacity of the designer of verifying all the possible situations that the system will face, it will be impossible to state that the system has no design flaws, because such flaws could pass undetected by the modeler. Simulating a system is somewhat similar, but since it can be done automatically by a system (instead of doing it manually in a test phase) it is possible to check complex queries just from the model of the system with a high degree of accuracy. Accuracy can be very high but never 100%, since there is no guarantee that any possible behavior has been simulated. Formal verification is instead the most complex to specify, since it requires many details on the model, and usually requires large quantity of memory to store the data needed to perform the verification. However, the answer to a verified formula is completely accurate, since every possible behavior of the system is considered.

1.1 Subject of the thesis

The work of this thesis focuses on formal verification of model based systems. The advantage of formal verification is that the provided results are exact. The verification algorithm usually proves the property in every possible configuration of the modeled system, which is equivalent to an exhaustive testing that covers every possible situations. The set of all possible configurations is the *state space* of the system. Constructing and storing a large *state space* is an hard problem, since it usually grows exponentially in the number of system parameters. In addition, quantitative properties are answered by doing a numerical analysis on the state space of the model, which may encounter problems of accuracy and numerical stability.

Systems have to be designed accurately to model the real entity they represent. Otherwise, a property verified on an unrealistic model will not provide any guarantee on the real entity. Many formal languages exists for specifying system models. To do performance analysis, the specification has to include timing information. Petri Nets and process algebra are instances of these formalisms. We will consider systems where the timing is *stochastic*, i.e. its behavior is described by a statistical law. Techniques that verify stochastic systems are either analytical, based on a structural check, or numerical, based on the full exploration of all the possible states of the system.

A *Markov chain* is a low-level formalism to specify the state-transition graph of a stochastic system. The Markov chain describes both its state properties and its future evolution. We focuses on both discrete-time and continuous-time Markov chains, which represent systems sampled at discrete time instants or observed in continuum.

The thesis focuses on the specification of measures for Markov chains, and the techniques needed to compute these measures. Therefore, we will assume that systems are described in any language that can produce a Markov chain, and then the Markov chain is used as input system. We will focus on two different ways of specifying properties: properties on *states/transitions* and properties on *paths*.

Performance properties of states and transitions are usually described as the expected probability of finding the system in a given condition after a certain amount of time. Mean probabilities of state-based conditions, rewards assigned to system conditions, and throughput of transitions are examples of these performance properties.

Path-based properties specifies the set of allowed behaviors of the system, for any possible model execution. This may complicate the analysis even further, since the set of paths is larger that the set of states (and possibly infinite). Under some assumptions, an analysis of path properties can be carried out, as done by stochastic temporal logics. Typical stochastic logics include PCTL for the probabilistic case, and CSL or CSL$^{\text{TA}}$ for the stochastic case. An important topic that is considered in the thesis regards the efficiency of the model checking algorithm of the latter logic, CSL$^{\text{TA}}$. A series of numerical techniques are presented and studied, in order to improve the model checking algorithm.

1.2 Contribution of the thesis

The thesis tries to provide a treatment of the concept of measure for Markov chains. Therefore, the presented material contains both concepts already found and established in the literature of Markov processes and performance evaluation, plus some new techniques that contribute to improve the context of formal verification of stochastic processes. The thesis aims to contribute a solution to these problems.

Sections 2.2.3 and 2.3.3 provide an easy way to reformulate numerical problems that employs forward probabilities into problems that use backward probabilities. This result is used to simplify the demonstration of how strictly-forward temporal logics (like CSL or CSL$^{\text{TA}}$) are computed numerically using backward probabilities. We believe that this reformulation is easier than the demonstrations found in literature. Also, the forward-backward conversion allowed us to derive the backward conversion formula for EMC/MRP probabilities in 2.4.4, which was not present in literature.

Section 3.4.2 contains an improved reformulation of the CSL$^{\text{TA}}$ model checking procedure. The new formulation allows for backward and forward solutions, and is designed around the construction of a Markov regenerative transition system (MRTS).

Section 4.3 describes a problem-specific preconditioner[1] for MRPs. The proposed preconditioner may be used to improve both the accuracy and the convergence rate of steady-state MRP solution methods. This preconditioner can also be used in the model checking procedure of CSL$^{\text{TA}}$. A numerical assessment shows the behavior of the preconditioner.

Section 4.4 describes the *component-method* for the steady-state solution of non-ergodic MRPs. The component method breaks down the initial transient of the Markov process into smaller components, and compute for each of these components the vector of ingoing/outgoing probabilities. The component-method is empirically evaluated in section 4.4.6 with a set of non-ergodic DSPNs (a class of *Petri nets* with deterministic transitions, whose underlying stochastic process is a MRP).

Chapter 5 shows that CSL$^{\text{TA}}$ in its basic form has a computational cost that is higher than CSL. A set of techniques employing the component method and the problem specific-preconditioner described before are used to reduce the performance mismatch. The various techniques described in the previous

[1]A preconditioner is a numerical technique that improves the convergence and the conditioning of a problem expressed as system of linear equations.

chapters are used to build up an adaptive strategy that improves the numerical solution of a CSL^{TA} model checker. Some numerical tests are done to evaluate the resulting strategy against previous CSL^{TA} model checkers and against state-of-the-art CSL model checkers. Finally, a new on-the-fly state space construction technique is proposed, to help reduce the memory occupation.

1.3 Outline of the thesis

The thesis is structured around the general concept of *measure* for Markov chains. A special focus is then given to the stochastic temporal logic CSL^{TA} and on how to improve its model checking algorithm.

Chapter 2 introduces Markov chains and describes a set of classical performance properties. The presentation focuses on both forward and backward timing reasoning, for three classes [Kul95] of Markov chains: DTMCs, CTMCs and MRPs. Each of these classes is represented by a specific *labeled transition system* [BK08], which provides its formal description. Forward and backward measures based on steady-state and transient timings are derived, for both reducible and irreducible systems.

Chapter 3 describes path-based properties. The focus is on *stochastic temporal logics* and *probe*-based specifications. These properties are verified with model composition, i.e. a new modified system is constructed to extract the specified behavior. The treated logics include PCTL [Han91], CSL [Azi+00] and CSL^{TA} [DHS09]. Temporal logics are assimilated as a type of measures, although the strict definition of these logics does not provide any quantitative information: only the truth value of a logic expression could be tested. In practice, probabilistic and stochastic temporal logics are used to compute the probability of formulas of paths inside a Markov chain. The emerging topic of *stochastic probe* specifications is also covered.

Chapter 4 focuses on the numerical solution methods needed to compute the quantitative value of performance measures. The chapter focuses on both transient and stationary numerical solutions. The numerical stability is considered, and a set of preconditioning techniques are described as a possible improvement for hard-to-solve systems. The last part of Chapter 4 describes the *component-method* for the solution of non-ergodic MRPs.

Chapter 5 focuses on the improvement of the model checking algorithms for the stochastic temporal logic CSL^{TA}, over the original solution method. The presentation shows that CSL^{TA} can learn from CSL a set of techniques that can be used to improve its model checking algorithm. Moreover, it is shown that the *component-method* is a convenient technique for the model checking of CSL^{TA}, providing a significant improvement in efficiency. The section ends with a description of the MC4CSL^{TA} model checker and of its numerical solver, which implements the techniques described in the previous chapters.

Chapter 2

Measuring properties of Markov chains

This chapter is devoted to Markovian processes, which are introduced in section 2.1. The chapter focuses on three commonly used Markov processes: Markov chains in discrete time, Markov chains in continuous time and Markov regenerative processes, described in section 2.2, 2.3 and 2.4 respectively. For each of the three classes of processes, a linear algebra and an automaton-based representation is given, followed by the forward/backward properties and formulas (in transient and steady state). The presentation then extends to the case of non-ergodic processes. The class of Markov regenerative processes is also studied, because the presented forward/backward framework of measurement can be applied easily to the embedded discrete-time Markov chain that describes the regeneration points of the process, and because it is a central subject of the following chapters.

Most of the content is commonly found in most books on stochastic processes. However, the focus of the presentation is about how influential are the forward and backward Kolmogorov equations on Markov chains. Almost any measure of interest is essentially based on one of these two equations. Interestingly, it is less known that forward measures can be computed with the backward equation, and vice-versa, using a small equation. Following the established convention, we use $\boldsymbol{\pi}$ to denote the vector of forward probabilities. There is no similar conventions for backward measures, so we choose to use $\boldsymbol{\xi}$. The presentation points out the many similarities and the differences of these two measures.

Non-ergodic processes also enlighten some substantial differences between the forward solution process (based on forward Kolmogorov equations) and the backward one. This conversion is of large importance for the computation of stochastic logics, treated in chapter 3.

Finally, section 2.5 uses the defined formulas to describe a commonly used framework for state-based and event-based performance measures, based on *rewards*. These measures can be computed with both forward and backward formulas, but the choice of the Kolmogorov equation has a consequence on how many computations have to be done to compute a measurement and on the algorithmic complexity.

2.1 Stochastic processes

A *system* is a model of some real-world phenomenon, and takes value on a *sample space* $\mathcal{S}$ (also called *state space*). The *state* of a system is a random variable X with image in $\mathcal{S}$. A *stochastic process* is a probabilistic model that describes the evolution of the states of a system, and is represented as a collection of random variables $\{X_t\}$ indexed through a parameter t that usually represents the *time* of the system.

Definition 1 (Stochastic process)**.** A *stochastic process* $\{X_t \mid t \in \mathcal{T}\}$ is a collection of random variables defined over a common state space $\mathcal{S}$ and indexed by $t \in \mathcal{T}$ (with $\mathcal{T}$ usually is the set of non-negative natural numbers $\mathbb{N}_{\geq 0}$ or the set of non-negative real numbers $\mathbb{R}_{\geq 0}$).

When the state space $\mathcal{S}$ is finite or countable set, the process is a *discrete-state process*, or a *chain*. In all other cases, it is a *continuous-state process*, which is not a topic of this thesis. An important classification of stochastic processes concerns the conditional dependence of the random variables in the process. In an *independent chain*, each random variable X_t is independent. However, this is excessively restrictive, and is usually relaxed in the *Markov chain*:

Definition 2 (Markov chain)**.** A stochastic process $\{X_t \mid t \in \mathcal{T}\}$ with discrete state space $\mathcal{S}$ is a *Markov chain* iff:

$$\begin{aligned}\Pr\{X_{t_{k+1}} = x_{k+1} \mid X_{t_k} = x_k, \dots, X_{t_0} = x_0\} &= \\ = \Pr\{X_{t_{k+1}} = x_{k+1} \mid X_{t_k} = x_k\}\end{aligned}$$

so that the evolution of the chain (at t_{k+1}) depends only on the current state (at t_k). This property is known as the *Markov property* or the *memoryless property.*

In a Markov chain, only the current state is needed to probabilistically predict the future evolution. The Markov property implies [CL06] that:

(**M1**) all past information is irrelevant (no state memory);

(**M2**) the time elapsed in the current state is irrelevant (no age memory).

In section 2.4 the aspect (**M2**) is relaxed to allow for age memories to control the probabilistic evolution of the chain. When the Markov chain is observed at a set of discrete time points, it is described as a *discrete-time* Markov chain, while if it observed continuously in every instant, it is a *continuous-time* Markov chain. Discrete and continuous chains are described in sections 2.2 and 2.3.

Definitions of distribution vectors

A function $\mu : \mathcal{S} \to [0,1]$ that assigns to each element (i.e. state) of $\mathcal{S}$ a real value in $[0,1]$ is a *measure* over $\mathcal{S}$. Let *Measure*$(\mathcal{S})$ be the set of all measure functions over $\mathcal{S}$. Moreover, μ may be a probability distribution if $\sum_{s\in\mathcal{S}} \mu(s) = 1$. The set of all probability distributions over $\mathcal{S}$ is denoted as *Distr*$(\mathcal{S})$, and clearly *Distr*$(\mathcal{S}) \subset$ *Measure*$(\mathcal{S})$. Let *Distr*$^0(\mathcal{S})$ be the set of all probability distributions plus the zero vector. Given a state $s \in \mathcal{S}$ we denote with $\mathbf{i}_s$ the *indicator function* of s (or the Dirac function of s), which is a probability distribution with $\mathbf{i}_s(s) = 1$ and $\mathbf{i}_s(p) = 0, \forall\, p \in \mathcal{S} \setminus \{s\}$.

2.2 Discrete time Markov chains

We indicate with $\{Y_n \mid n \in \mathbb{N}_{\geq 0}\}$ a Discrete Time Markov Chain (DTMC) defined over a finite state space $\mathcal{S}$ observed at discrete non-negative time points $n \in \mathbb{N}_{\geq 0}$. Let $\mathbf{P}$ be the stochastic matrix of $\{Y_n\}$, which is a non-negative real square matrix defined over $\mathcal{S} \times \mathcal{S}$ such that each row $\mathbf{P}_i$ of $\mathbf{P}$ is in $\mathit{Distr}(\mathcal{S})$. The j-th entry $\mathbf{P}(i,j)$ of row $\mathbf{P}_i$ describes the probability of going from state i to state j in one discrete time step. Also let's assume that $\{Y_n\}$ is *time-homogeneous*, i.e. each probability value $\mathbf{P}(i,j)$ that describes the evolution of each Y_n from state i to state j is constant and does not depend on the time parameter n, so that:

$$\mathbf{P}(i,j) \overset{\text{def}}{=} \Pr\{X_{n+1} = j \mid X_n = i\} \;=\; \Pr\{X_1 = j \mid X_0 = i\}$$

DTMCs can be classified according to the connectivity of the states. If from every state i it is possible to reach every other state of $\mathcal{S}$ with a finite number of steps, then the DTMC is *irreducible*. Otherwise, if there exists (at least) a state i that cannot reach all the states of $\mathcal{S}$ after any number of steps, than the DTMC is *reducible*. In terms of linear algebra, a DTMC is irreducible if $\lim\limits_{n \to \infty} \mathbf{P}^n$ has no zero entries, i.e. there is a non-zero route probability $\mathbf{P}^n(i,j)$ for every pair of states i, j. For now we consider the irreducible case. The reducible case is treated in section 2.2.5

2.2.1 Discrete-time Markov Transition System

The usual representation of a DTMC in statistics is the one given before: a family of random variables governed by a stochastic matrix $\mathbf{P}$. Alternatively, a DTMC can be described as a *deterministic finite automaton* (DFA) whose state-transition relation is governed by a probabilistic law. The automaton representation has the advantage of being more compact and self-contained, and is a common representation in contexts like *model checking* [BK08]. In addition, it is useful to include labels attached to states, as used by *Kripke structures* (KS), and *actions* names to transitions, as used by *Labeled Transition Systems* (LTS or simply TS) [BK08, pp. 20-21].

Definition 3 (DMTS)**.** A *Discrete-time Markov Transition System* (DMTS) is a tuple $\mathcal{D} = \langle \mathcal{S}, \mathit{Act}, \mathit{AP}, \mathit{lab}, T, P, \boldsymbol{\alpha}_0 \rangle$ where:

- $\mathcal{S}$ is a finite set of *states*;
- Act is a finite set of *action* names;
- AP is a finite set of *atomic propositions*;
- $\mathit{lab} : \mathcal{S} \to 2^{\mathit{AP}}$ is a *state-labeling* function.
- $T \subseteq \mathcal{S} \times \mathit{Act} \times \mathcal{S}$ is a *transition* relation;
- $P : T \to \mathbb{R}_{(0,1]}$ is a *transition probability* function, with the restriction that for all states s it holds that: $\sum_{a \in \mathit{Act}} P(s,a) \in \mathit{Distr}^0(\mathcal{S})$;
- $\boldsymbol{\alpha}_0 \in \mathit{Distr}(\mathcal{S})$ is an initial distribution.

State labels are taken from a set of atomic propositions AP and are used to express properties that hold in states. Given a state $s \in \mathcal{S}$, the value of $\mathit{lab}(s)$ is the set of atomic propositions that hold in s. The notation $s \models a$ with $a \in \mathit{AP}$ means that $a \in \mathit{lab}(s)$. Action names are taken from a set Act of actions. The notation $i \xrightarrow{a} j$ means that $(i, a, j) \in T$. We assume that $P(i, a, j) = 0$ if the transition $i \xrightarrow{a} j$ is not in T. The notation $i \xrightarrow{a} j \sim \lambda$ means that the transition $\tau = i \xrightarrow{a} j$ has a probability $P(\tau)$ equal to $\lambda \in \mathbb{R}_{(0,1]}$.

The DMTS $\mathcal{D}$ starts in some state $s_0 \sim \boldsymbol{\alpha}_0$ and evolves according to T. That is, if the current state is i, then a transition $\tau = i \xrightarrow{a} j$ is taken with probability $P(\tau)$. In $\mathcal{D}$ the process may go from a state i to a state j with different actions, each with its own probability $P(i, a, j)$. The non-zero entries of the stochastic matrix are derived from the distributions of P by summing all actions, denoted as: $\mathbf{P}(i, j) = \sum_{a \in Act} P(i, a, j)$. Hence, a DMTS has the behavior of a DTMC with stochastic matrix $\mathbf{P}$.

2.2.2 Forward DTMC probabilities

The dynamic behavior of a DTMC can be described in terms of *forward* probability or in terms of *backward* probability distributions, based on the forward and backward *Chapman-Kolmogorov* equations [Tri02, p. 342].

Forward probabilities give the behavior of the evolution of the system after t time units, for a given *initial* fixed state i The probability of being in state j after n steps, knowing that at time 0 the state was i, is denoted by:

$$\pi^{\mathcal{D}}(i, j, n) \overset{\text{def}}{=} \Pr\{Y_n = j \mid Y_0 = i\} \tag{2.1}$$

When we consider an initial distribution $\boldsymbol{\alpha} \in \mathit{Distr}(\mathcal{S})$ rather than a single initial state, the vector $\boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha}, t)$ of forward state probabilities conditioned by $\boldsymbol{\alpha}$ becomes:

$$\begin{aligned} \Big[\boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha}, n)\Big](j) &\overset{\text{def}}{=} \Pr\{Y_n \mid Y_0 \sim \boldsymbol{\alpha}\} \;= \\ &= \sum_{i \in \mathcal{S}} \boldsymbol{\alpha}(i) \cdot \pi^{\mathcal{D}}(i, j, n) \end{aligned} \tag{2.2}$$

Equation (2.2) is subject to the *forward Chapman-Kolmogorov* equations (for the time-homogeneous case), for $n \geq 0$:

$$\boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha}, n) \;=\; \boldsymbol{\alpha} \cdot \mathbf{P}^n \tag{2.3}$$

The j-th element of the vector $\boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha}, n)$ is the probability of reaching state j in n steps starting from an initial distribution $\boldsymbol{\alpha}$. The irreducibility of the DTMC ensures that $\boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha}, n) \in \mathit{Distr}(\mathcal{S})$ for all vectors $\boldsymbol{\alpha} \in \mathit{Distr}(\mathcal{S})$, since a product with $\mathbf{P}^n$ is an ergodic function that preserves the magnitude of $\boldsymbol{\alpha}$ (and therefore $|\boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha}, n) = |\boldsymbol{\alpha}|$). When we need instead to compute the probability of reaching the fixed state j in n steps from *every other state* i, we need to recompute the above formula $|\mathcal{S}|$ times, one for each $\boldsymbol{\alpha} = \mathbf{i}_i$.

2.2.3 Backward DTMC probabilities

Backward probabilities represent the probability that the system started in state i at time 0, given that at step n it is observed in a given *destination* state j:

$$\xi^{\mathcal{D}}(i, j, n) \overset{\text{def}}{=} \Pr\{Y_0 = i \mid Y_n = j\} \tag{2.4}$$

If we now consider a measure vector $\boldsymbol{\rho} \in \mathit{Measure}(\mathcal{S})$ over a target set of states at step n, we can introduce the backward probability vector $\boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}, n)$ that represents the backward probabilities conditioned by the target vector $\boldsymbol{\rho}$:

$$\begin{aligned} \Big[\boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}, n)\Big](i) &\overset{\text{def}}{=} \Pr\{Y_0 \mid Y_n \sim \boldsymbol{\rho}\} \ = \\ &= \sum_{j \in \mathcal{S}} \boldsymbol{\rho}(j) \cdot \xi^{\mathcal{D}}(i, j, m) \end{aligned} \tag{2.5}$$

The measure vector $\boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}, n)$ does not represent a probability distribution: indeed it does not sum to one, and each entry is an independent quantity. When $\boldsymbol{\rho} = \mathbf{i}_j$ then the vector $\boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}, n)$ gives the probability of reaching the fixed state j in n steps from *each* possible initial state i. Equation (2.5) is governed by the *backward Chapman-Kolmogorov* equation:

$$\boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}, n) \ = \ \mathbf{P}^n \cdot \boldsymbol{\rho} \tag{2.6}$$

and it is important to observe that forward and backward probabilities are tied together by the relation:

$$\boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha}, n) \cdot \boldsymbol{\rho} \ = \ \boldsymbol{\alpha} \cdot \boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}, n) \tag{2.7}$$

which can be easily proven since: $\big(\boldsymbol{\alpha} \cdot \mathbf{P}^n\big) \cdot \boldsymbol{\rho} = \boldsymbol{\alpha} \cdot \big(\mathbf{P}^n \cdot \boldsymbol{\rho}\big)$.

2.2.4 Stationary behavior of DTMCs

Under the restriction of aperiodicity, the limiting behaviors $\boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha}, n)$ and $\boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}, n)$ for $n \to \infty$ do not depend on the time parameter n, but are somewhat constant vectors that depend only on $\mathbf{P}$. We denote the forward/backward limiting state probability with:

$$\boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha}) \overset{\text{def}}{=} \lim_{n\to\infty} \boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha}, n) \ = \ \boldsymbol{\alpha} \cdot \lim_{n\to\infty} \mathbf{P}^n \tag{2.8}$$

$$\boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}) \overset{\text{def}}{=} \lim_{n\to\infty} \boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}, n) \ = \ \lim_{n\to\infty} \mathbf{P}^n \cdot \boldsymbol{\rho} \tag{2.9}$$

and again forward and backward are tied together by the relation:

$$\boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha}) \cdot \boldsymbol{\rho} \ = \ \boldsymbol{\alpha} \cdot \boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}) \tag{2.10}$$

We recall that for irreducible DTMCs, $\boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha})$ always exists, it is unique and independent of $\boldsymbol{\alpha}$, i.e. $\boldsymbol{\pi}^{\mathcal{D}} = \boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha})$, $\forall \boldsymbol{\alpha} \in \mathit{Distr}(\mathcal{S})$. Non-null entries of $\boldsymbol{\pi}^{\mathcal{D}}$ are said to be *recurrent nonull*, or *positive recurrent*. It is less known instead that the limiting backward vector $\boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho})$ always depends on $\boldsymbol{\rho}$ ($\boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}) \neq \boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}')$ for $\boldsymbol{\rho} \neq \boldsymbol{\rho}'$), and that each entry of the vector has the same scalar value $\xi_{\boldsymbol{\rho}}$ ($\boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho})[s] = \xi_{\boldsymbol{\rho}}$, for every $s \in \mathcal{S}$) and $\xi_{\boldsymbol{\rho}}$ is uniquely dependent on $\boldsymbol{\rho}$.

2.2.5 Reducible DTMCs

When the DTMC is reducible, its stochastic matrix $\mathbf{P}$ can be reordered as an upper triangular block form (called *reducible normal form*):

$$\mathbf{P} = \begin{bmatrix} \mathbf{T} & \mathbf{F}_1 & \cdots & \mathbf{F}_m \\ 0 & \mathbf{R}_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{R}_m \end{bmatrix} \tag{2.11}$$

The state space $\mathcal{S}$ is partitioned into the set of *transient* states $\mathcal{S}_\mathrm{T}$ and m sets of *recurrent* states $\mathcal{S}_{\mathrm{R}_i}$ (*recurrent classes*). The sub-matrix T is the sub-stochastic DTMC of the *transient* states $\mathcal{S}_\mathrm{T}$. Each rectangular sub-matrix F_i is the probability matrix of going into the i-th recurrent class from $\mathcal{S}_\mathrm{T}$. Each square sub-matrix R_i is the DTMC of the i-th recurrent class.

For simplicity, the notation of [Ger00] is followed, for which all matrices (and vectors) have the same size of $\mathcal{S}\times\mathcal{S}$ (and $\mathcal{S}$). Therefore $\mathbf{T}$, $\mathbf{F}$ and $\mathbf{R}_i$ have the same size $\mathcal{S}\times\mathcal{S}$, and actually $\mathbf{P}$ (and $\mathbf{Q}$) should be regarded as a simple summation of $\mathbf{T}+\mathbf{F}+\sum_{i=1}^m \mathbf{R}_i$.

As mentioned before, $\mathbf{P}$ is stochastic, which means that each row of $\mathbf{P}$ is a probability distribution. Rows corresponding to transient states are split into $\mathbf{T}$ and the various $\mathbf{F}_i$ sub-matrices, therefore each of these matrices is not stochastic but *sub-stohastic*, having (at least) one row that sums up to a value in the range $[0,1)$. A sub-stochastic matrix $\mathbf{A}$ has many interesting properties: the most important is that the spectral radius $\rho(\mathbf{A})$ is strictly less than one. This implies that $\lim\limits_{k\to 0}\mathbf{A}^k$ is the zero matrix. Also, many numerical methods whose convergence depend on the value of the spectral radius to be less than 1 will typically work well with sub-stocastic matrices.

We now derive the structure of $\lim\limits_{n\to\infty}\mathbf{P}^n$, as in equations (2.8) and (2.9) when $\mathbf{P}$ is reducible. The powers of $\mathbf{P}$ are:

$$\mathbf{P}^0 = \mathbf{I}, \qquad \mathbf{P}^n = \begin{bmatrix} \mathbf{T}^n & \mathbf{\Lambda}_1(n) & \cdots & \mathbf{\Lambda}_m(n) \\ 0 & \mathbf{R}_1^n & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{R}_m^n \end{bmatrix}$$

with $\mathbf{\Lambda}_i(n) = \sum\limits_{k=0}^{n-1} \mathbf{T}^k\,\mathbf{F}_i\,\mathbf{R}_i^{n-k-1}$ and $\mathbf{\Lambda}_i(0)=0$.

The n-th power of $\mathbf{P}$ reveals the well-known structure of a reducible process after n steps: the process stays for k steps ($k\le n$) in the transient class $\mathcal{S}_\mathrm{T}$, then a single $\mathbf{F}_i$ transition occurs and moves the process in the i-th recurrent class $\mathcal{S}_{\mathrm{R}_i}$, after which the process remains in $\mathcal{S}_{\mathrm{R}_i}$ for the remaining $(n-k-1)$ steps.

Transient forward and backward probabilities can then be rewritten from equations (2.3) and (2.6) using the expression for $\mathbf{P}^k$ derived above; it is convenient to separate the part for the recurrent and transient states, to obtain:

$$\begin{aligned} \boldsymbol{\pi}^{\mathcal{D}}_\mathrm{T}(\boldsymbol{\alpha},n) &= \boldsymbol{\alpha}_\mathrm{T}\cdot\mathbf{T}^n \\ \boldsymbol{\pi}^{\mathcal{D}}_{\mathrm{R}_i}(\boldsymbol{\alpha},n) &= \boldsymbol{\alpha}_\mathrm{T}\cdot\mathbf{\Lambda}_i(n) + \boldsymbol{\alpha}_{\mathrm{R}_i}\cdot\mathbf{R}_i^n \\ \boldsymbol{\xi}^{\mathcal{D}}_\mathrm{T}(\boldsymbol{\rho},n) &= \sum_{i=1}^{n}\big(\mathbf{\Lambda}_i(n)\cdot\boldsymbol{\rho}_{\mathrm{R}_i}\big) + \mathbf{T}^n\cdot\boldsymbol{\rho}_\mathrm{T} \\ \boldsymbol{\xi}^{\mathcal{D}}_{\mathrm{R}_i}(\boldsymbol{\rho},n) &= \mathbf{R}_i^n\cdot\boldsymbol{\rho}_{\mathrm{R}_i} \end{aligned} \tag{2.12}$$

Let $\mathbf{J}$ be the *fundamental matrix* [Tri02, p. 7.9] of $\mathbf{T}$, defined as:

$$\mathbf{J} = \sum_{k=0}^{\infty}\mathbf{T}^k = (\mathbf{I}-\mathbf{T})^{-1} \tag{2.13}$$

The existence of $\mathbf{J}$ is ensured by the fact that $\mathbf{T}$ is substochastic, thus invertible (the spectral radius $\rho(\mathbf{T})$ is <1, therefore the series of $\mathbf{T}^k$ converges). The fundamental matrix carries an important information for the reducible process: each entry $\mathbf{J}(i,j)$ is the expected number of times the process visits state j, before entering an absorbing set, given that it started from state i. Observe that since $\mathbf{J} = (\mathbf{I} - \mathbf{T})^{-1}$, then it is possible to compute any vector-matrix products with $\mathbf{J}$ as the solutions of a linear equation system in $(\mathbf{I} - \mathbf{T})$, instead of computing a product with $\mathbf{J}$ directly, i.e.:

$$\begin{aligned} \mathbf{x} = \mathbf{b} \cdot \mathbf{J} \quad &\Rightarrow \quad \text{solution of: } (\mathbf{I} - \mathbf{T}) \cdot \mathbf{x} = \mathbf{b} \\ \mathbf{x} = \mathbf{J} \cdot \mathbf{b} \quad &\Rightarrow \quad \text{solution of: } \mathbf{x} \cdot (\mathbf{I} - \mathbf{T}) = \mathbf{b} \end{aligned}$$

for any measure vector $\mathbf{b}$ in the $\mathbb{R}^{|\mathcal{S}|}$ space.

Considering that $\lim_{n\to\infty} \mathbf{T}^n = 0$, using the definition of $\mathbf{J}$ we can rewrite the limiting behavior of $\mathbf{P}^n$ as:

$$\lim_{n\to\infty} \mathbf{P}^n = \begin{bmatrix} 0 & \mathbf{J} \cdot \mathbf{F}_1 \cdot \lim_{n\to\infty} \mathbf{R}_1^n & \cdots & \mathbf{J} \cdot \mathbf{F}_m \cdot \lim_{n\to\infty} \mathbf{R}_m^n \\ 0 & \lim_{n\to\infty} \mathbf{R}_1^n & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lim_{n\to\infty} \mathbf{R}_m^n \end{bmatrix}$$

from which the limiting behaviors of (2.12) result in:

$$\begin{aligned} \boldsymbol{\pi}_{\mathrm{T}}^{\mathcal{D}}(\boldsymbol{\alpha}) &= 0 \\ \boldsymbol{\pi}_{\mathrm{R}_i}^{\mathcal{D}}(\boldsymbol{\alpha}) &= \big(\boldsymbol{\alpha}_{\mathrm{T}} \cdot \mathbf{J} \cdot \mathbf{F}_i \ + \ \boldsymbol{\alpha}_{\mathrm{R}_i}\big) \cdot \lim_{n\to\infty} \mathbf{R}_i^n \\ \boldsymbol{\xi}_{\mathrm{T}}^{\mathcal{D}}(\boldsymbol{\rho}) &= \mathbf{J} \cdot \sum_{i=1}^{m} \Big(\mathbf{F}_i \cdot \lim_{n\to\infty} \mathbf{R}_i^n \cdot \boldsymbol{\rho}_{\mathrm{R}_i}\Big) \\ \boldsymbol{\xi}_{\mathrm{R}_i}^{\mathcal{D}}(\boldsymbol{\rho}) &= \lim_{n\to\infty} \mathbf{R}_i^n \cdot \boldsymbol{\rho}_{\mathrm{R}_i} \end{aligned} \tag{2.14}$$

The vector $\boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha})$ is zero in every transient state, which is an expected result since the process cannot be found in a transient state in the long run. The vector $\boldsymbol{\alpha}_{\mathrm{T}} \cdot \mathbf{J} \cdot \mathbf{F}_i$ can be interpreted as the probability of entering $\mathcal{S}_{\mathrm{R}_i}$ in the long run, from the transient set $\mathcal{S}_{\mathrm{T}}$. Given a recurrent class $\mathcal{S}_{\mathrm{R}_i}$, the long-run probabilities $\boldsymbol{\pi}_{\mathrm{R}_i}^{\mathcal{D}}(\boldsymbol{\alpha})$ are obtained by multiplying the steady state solution in isolation of that recurrent class ($\lim_{n\to\infty} \mathbf{R}_i^n$) with the vector of the probability that reaches that class in the long run.

For backward probabilities we can observe that $\boldsymbol{\rho}$ values associated with $\mathcal{S}_{\mathrm{T}}$ ($\boldsymbol{\rho}_{\mathrm{T}}$) have no influence on the probability of neither the transient states ($\boldsymbol{\xi}_{\mathrm{T}}^{\mathcal{D}}(\boldsymbol{\rho})$) nor the recurrent states ($\boldsymbol{\xi}_{\mathrm{R}_i}^{\mathcal{D}}(\boldsymbol{\rho})$). This is indeed a consequence of the fact that a transient state cannot be encountered as a target on the long run. Moreover the steady state backward probability of a recurrent state can be computed on the recurrent class in isolation (all the quantities in the equation for $\boldsymbol{\xi}_{\mathrm{R}_i}^{\mathcal{D}}(\boldsymbol{\rho})$ refer only to the recurrent class $\mathcal{S}_{\mathrm{R}_i}$). Note that, as explained before, all the states of the same recurrent class have the same value of backward probability. More interesting is the case of backward probability of transient states $\boldsymbol{\xi}_{\mathrm{T}}^{\mathcal{D}}(\boldsymbol{\rho})$, in which the probability of each recurrent class is "projected back" to the initial transient states through the multiplication with matrix $\mathbf{F}_i$ (one step probability of reaching $\mathcal{S}_{\mathrm{R}_i}$ from $\mathcal{S}_{\mathrm{T}}$) and matrix $\mathbf{J}$ (transient behavior).

The two relations (2.7) and (2.10) still hold for reducible Markov chains, which can be proven easily by expanding them with the terms of (2.14) and (2.12).

Note that when $\boldsymbol{\rho}$ is the indicator vector of a set of absorbing states then $\boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}, n)$ is commonly known as "transient absorption probability" [Kul95].

2.3 Continuous-time Markov chains

Continuous-time Markov chains (CTMCs) are Markov chains defined over a continuous time domain in $\mathbb{R}_{\geq 0}$. The behavior of a CTMC $\{X_t \mid t \in \mathbb{R}_{\geq 0}\}$, defined over a finite state space $\mathcal{S}$, is described by the *infinitesimal generator* matrix $\mathbf{Q}$, which we assume to be time-homogeneous (i.e. it is constant for X_t and does not depend on t). An infinitesimal generator describes the instantaneous exit rates $\mathbf{Q}(i,j)$ for each pair of source and destination states i and j, and accounts in each diagonal entry $\mathbf{Q}(i,i)$ for the total exit rate (in negative sign) from state i. So, each row of $\mathbf{Q}$ sums up to 0, making $\mathbf{Q}$ a singular matrix.

Let $p_{i,j}(\Delta t)$ be the discrete probability of going from state i to state j in the time interval Δt. In this way, the value of Δt gives the length of the discrete time step of a DTMCs with stochastic matrix $\mathbf{P} = \left[p_{i,j}(\Delta t)\right]$. The tangential behavior of $p_{i,j}(\Delta t)$ in the rounding of a small time step ($\Delta t \to 0$) defines the mean *rate* of the transition from i to j, measured in events/time:

$$\begin{aligned} q_{i,j} &\overset{\text{def}}{=} -\frac{\mathrm{d}}{\mathrm{d}t} p_{i,j}(\Delta t)\bigg|_{\Delta t \to 0} = \lim_{\Delta t \to 0} \frac{p_{i,j}(0) - p_{i,j}(\Delta t)}{\Delta t} = \\ &= \lim_{\Delta t \to 0} \frac{p_{i,j}(\Delta t)}{\Delta t} \end{aligned} \tag{2.15}$$

and similarly for the rate of remaining in state i:

$$\begin{aligned} q_{i} &\overset{\text{def}}{=} -\frac{\mathrm{d}}{\mathrm{d}t} p_{i,i}(\Delta t)\bigg|_{\Delta t \to 0} = \lim_{\Delta t \to 0} \frac{p_{i,i}(0) - p_{i,i}(\Delta t)}{\Delta t} = \\ &= \lim_{\Delta t \to 0} \frac{1 - p_{i,i}(\Delta t)}{\Delta t} \end{aligned} \tag{2.16}$$

Given a matrix $\mathbf{Q} = \left[q_{i,j}\right]$ with the diagonal entries $\mathbf{Q}(i,i) = -q_i$, it is easy to see that $\sum_j \mathbf{Q}(i,j) = 0$ for all i. Given that definition, it is possible to write the matrix equations [Tri02, p. 408]:

$$\frac{\mathrm{d}\,\mathbf{P}}{\mathrm{d}t} = \mathbf{P} \cdot \mathbf{Q} = \mathbf{Q} \cdot \mathbf{P} \tag{2.17}$$

along with the forward and backward Kolmogorov differential equations that will be used in sections 2.3.2 and 2.3.3.

2.3.1 Continuous-time Markov Transition System

A CTMC can be described as a *transition system* similarly to a DMTS (as in definition (3)), with a timed transition relation, state labels and action names.

Definition 4 (CMTS)**.** A *Continuous-time Markov Transition System* (CMTS) is a tuple $\mathcal{M} = \langle \mathcal{S}, Act, AP, lab, T, P, E, \boldsymbol{\alpha}_0 \rangle$ where:

- $\langle \mathcal{S}, Act, AP, lab, T, P, \boldsymbol{\alpha}_0 \rangle$ is a DMTS;
- $E : \mathcal{S} \to \mathbb{R}_{\geq 0}$ is the *exit rate* function;

The value of $E(i)$ gives the Markovian exit rate from state i. The notation $i \xrightarrow{a} j \sim \lambda$ means that the transition $\tau = i \xrightarrow{a} j$ has a probability $P(\tau)$ equal to $\lambda \in \mathbb{R}_{(0,1]}$, while the notation $i \xrightarrow{a} j \sim_{\text{rate}} \mu$ means that the transition has an exit rate $E(i) \cdot P(\tau)$ equal to $\mu \in \mathbb{R}_{>0}$.

The evolution of a CMTS is analogous to that of a DMTS, except that the system remains in each state i for an amount of time that is distributed according to an exponential distribution with rate $E(i)$. The probability to exit state i in t time units is therefore: $\int_0^t E(i) \cdot \mathrm{e}^{-E(i)x}\, \mathrm{d}x$. Then a new transition $\tau = i \xrightarrow{a} j$ is chosen from T with probability $P(\tau)$. The entries of the infinitesimal generator matrix are derived from P and E as: $\mathbf{Q}(i,j) = \big(\sum_{a \in Act} P(i,a,j) - \mathbf{I}(i,j)\big) \cdot E(i)$. A CMTS has therefore the behavior of a CTMC with infinitesimal generator $\mathbf{Q}$.

2.3.2 Forward CTMC probabilities

The time evolution of a CTMC is governed by the forward/backward *Kolmogorov differential equations* [Kul95, th. 2.3]. The probability of being in state j at time t, knowing that at time 0 the state was i, is denoted by:

$$\pi^{\mathcal{M}}(i,j,t) \overset{\text{def}}{=} Pr\{X_t = j \mid X_0 = i\} \tag{2.18}$$

or, in vector form:

$$\begin{aligned} \Big[\boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha}, t)\Big](j) &\overset{\text{def}}{=} Pr\{X_t \mid X_0 \sim \boldsymbol{\alpha}\} = \\ &= \sum_{i \in \mathcal{S}} \boldsymbol{\alpha}(i) \cdot \pi^{\mathcal{M}}(i,j,t) \end{aligned} \tag{2.19}$$

with $\boldsymbol{\alpha} \in Distr(\mathcal{S})$ a probability distribution at time 0. Vector (2.19) is the solution of the forward Kolmogorov differential equation:

$$\frac{\mathrm{d}\,\boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha}, t)}{\mathrm{d}t} = \boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha}, t) \cdot \mathbf{Q} \tag{2.20}$$

with *entrance condition* $\boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha}, 0) = \boldsymbol{\alpha}$.

The solution of the linear first-order homogeneous matrix differential equation (2.20) for $t \geq 0$ is:

$$\boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha}, t) = \boldsymbol{\alpha} \cdot \mathrm{e}^{\mathbf{Q}t} \tag{2.21}$$

2.3.3 Backward CTMC probabilities

Backward probability gives the probability that the CTMC was in state i at time 0, given that at time t is observed in state j, is:

$$\xi^{\mathcal{M}}(i,j,t) \overset{\text{def}}{=} Pr\{X_0 = i \mid X_t = j\} \tag{2.22}$$

or, in vector form:

$$\begin{aligned} \Big[\boldsymbol{\xi}^{\mathcal{M}}(\boldsymbol{\rho}, v)\Big](i) &\overset{\text{def}}{=} Pr\{X_0 \mid X_t \sim \boldsymbol{\rho}\} = \\ &= \sum_{j \in \mathcal{S}} \boldsymbol{\rho}(j) \cdot \xi^{\mathcal{M}}(i,j,v) \end{aligned} \tag{2.23}$$

with $\boldsymbol{\rho}$ a measure vector over the target states at time t. The resulting vector is the solution of the backward Kolmogorov differential equation:

$$\frac{\mathrm{d}\,\boldsymbol{\xi}^{\mathcal{M}}(\boldsymbol{\rho},t)}{\mathrm{d}t} \;=\; \mathbf{Q}\cdot\boldsymbol{\xi}^{\mathcal{M}}(\boldsymbol{\rho},t) \tag{2.24}$$

with *exit condition* $\boldsymbol{\xi}^{\mathcal{M}}(\boldsymbol{\rho},t)=\boldsymbol{\rho}$. Solution of (2.24) is:

$$\boldsymbol{\xi}^{\mathcal{M}}(\boldsymbol{\rho},t) \;=\; \mathrm{e}^{\mathbf{Q}t}\cdot\boldsymbol{\rho} \tag{2.25}$$

Forward and backward formulas are tied by the relation:

$$\boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha},t)\cdot\boldsymbol{\rho} \;=\; \boldsymbol{\alpha}\cdot\boldsymbol{\xi}^{\mathcal{M}}(\boldsymbol{\rho},t) \tag{2.26}$$

which is equivalent to: $\left(\boldsymbol{\alpha}\cdot e^{\mathbf{Q}t}\right)\cdot\boldsymbol{\rho}=\boldsymbol{\alpha}\cdot\left(e^{\mathbf{Q}t}\cdot\boldsymbol{\rho}\right)$, and:

$$\boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha})\cdot\boldsymbol{\rho} \;=\; \boldsymbol{\alpha}\cdot\boldsymbol{\xi}^{\mathcal{M}}(\boldsymbol{\rho}) \tag{2.27}$$

The computation of the transient measures (2.21) and (2.25) may be carried out in many ways [ML78]; the popular uniformisation method can be defined for both forward and backward probabilities, leading to the two following Taylor series expansion of the solution of the first-order constant coefficients ODE of the Kolmogorov differential equations (2.21) and (2.25):

$$\begin{aligned} \boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha},t) \;&=\; \boldsymbol{\alpha}\cdot e^{\mathbf{Q}t} \;=\; \boldsymbol{\alpha}\cdot\sum_{n=0}^{\infty}\left(\frac{e^{-qt}(qt)^n}{n!}\mathbf{U}^n\right) \\ \boldsymbol{\xi}^{\mathcal{M}}(\boldsymbol{\rho},t) \;&=\; e^{\mathbf{Q}t}\cdot\boldsymbol{\rho} \;=\; \sum_{n=0}^{\infty}\left(\frac{e^{-qt}(qt)^n}{n!}\mathbf{U}^n\right)\cdot\boldsymbol{\rho} \end{aligned} \tag{2.28}$$

with $q\geq-\max_{i\in\mathcal{S}}\mathbf{Q}(i,i)$ and $\mathbf{U}={}^{1}\!/_{q}\mathbf{Q}+\mathbf{I}$ the *uniformized matrix* of $\mathbf{Q}$. As usual, $\mathbf{U}$ is the stochastic matrix of the mean behavior after a time step of duration ${}^{1}\!/_{q}$.

2.3.4 Stationary behavior of CTMCs

The stationary behavior can be defined also for CTMCs. Taking the limit to infinity of (2.21) and (2.25) we get:

$$\boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha}) \overset{\text{def}}{=} \lim_{t\to\infty}\boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha},t) \;=\; \lim_{t\to\infty}\boldsymbol{\alpha}\cdot\mathrm{e}^{\mathbf{Q}t} \tag{2.29}$$

$$\boldsymbol{\xi}^{\mathcal{M}}(\boldsymbol{\rho}) \overset{\text{def}}{=} \lim_{t\to\infty}\boldsymbol{\xi}^{\mathcal{M}}(\boldsymbol{\rho},t) \;=\; \lim_{t\to\infty}\mathrm{e}^{\mathbf{Q}t}\cdot\boldsymbol{\rho} \tag{2.30}$$

Therefore $\boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha})$ and $\boldsymbol{\xi}^{\mathcal{M}}(\boldsymbol{\rho})$ denote the stationary forward and backward vectors of the CTMC. Under the condition of irreducibility, the vector $\boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha})$ is unique and does not depend on $\boldsymbol{\alpha}$, and we denote it as $\boldsymbol{\pi}^{\mathcal{M}}$. Steady state formulas are computed as the solutions of the two linear equation systems in $\mathbf{Q}$:

$$\begin{aligned} \boldsymbol{\pi}^{\mathcal{M}}\cdot\mathbf{Q} \;&=\; 0 \\ \mathbf{Q}\cdot\boldsymbol{\xi}^{\mathcal{M}}(\boldsymbol{\rho}) \;&=\; \boldsymbol{\rho} \end{aligned} \tag{2.31}$$

Since $\boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha})$ does not depend on $\boldsymbol{\alpha}$, we write it simply as $\boldsymbol{\pi}^{\mathcal{M}}$. Backward solution instead depends on $\boldsymbol{\rho}$, since $\boldsymbol{\rho}$ is the given limiting distribution for $t\to\infty$. Many well-established algorithms exist [Ste94] for the computation of the limiting behavior of a CTMC.

2.3.5 Reducible CTMCs

When the CTMC is reducible, its infinitesimal generator $\mathbf{Q}$ can be written in reducible normal form:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{T} & \mathbf{F}_1 & \cdots & \mathbf{F}_m \\ 0 & \mathbf{R}_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{R}_m \end{bmatrix} \tag{2.32}$$

The sub-matrices $\mathbf{R}_i$ are infinitesimal generators, but the matrices $\mathbf{T}$ and $\mathbf{F}_i$ are not, because (at least) one of their rows does not sum up to 0. Like in section 2.2.5, this has various numerical implications. The most important is that $\mathbf{T}$ and $\mathbf{F}_i$ are not singular, and the matrix exponential $\lim\limits_{t\to\infty} \mathrm{e}^{\mathbf{T}t}$ is the zero matrix.

As for DTMCs, we start from the structure of $\lim\limits_{t\to\infty} \mathrm{e}^{\mathbf{Q}t}$ to show how to derive transient and stationary (for both forward and backward) formulas. The matrix powers of $\mathbf{Q}$ are:

$$\mathbf{Q}^0 = \mathbf{I}, \qquad \mathbf{Q}^n = \begin{bmatrix} \mathbf{T}^n & \mathbf{\Lambda}_1(n) & \cdots & \mathbf{\Lambda}_m(n) \\ 0 & \mathbf{R}_1^n & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{R}_m^n \end{bmatrix}$$

with $\mathbf{\Lambda}_i(n) = \sum\limits_{k=0}^{n-1} \mathbf{T}^k\, \mathbf{F}_i\, \mathbf{R}_i^{n-k-1}$ and $\mathbf{\Lambda}_i(0) = 0$. The exponential of $\mathbf{Q}\,t$ is then:

$$\mathrm{e}^{\mathbf{Q}t} = \sum_{n=0}^{\infty} \frac{\mathbf{Q}^n\, t^n}{n!} = \begin{bmatrix} \mathrm{e}^{\mathbf{T}t} & \mathbf{\Theta}_1(t) & \cdots & \mathbf{\Theta}_m(t) \\ 0 & \mathrm{e}^{\mathbf{R}_1 t} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathrm{e}^{\mathbf{R}_m t} \end{bmatrix}$$

with the term $\mathbf{\Theta}_i(t)$ defined as:

$$\begin{aligned} \mathbf{\Theta}_i(t) &= \sum_{n=0}^{\infty} \frac{\mathbf{\Lambda}_i(n)\, t^n}{n!} = \sum_{n=0}^{\infty} \sum_{k=0}^{n-1} \frac{\mathbf{T}^k\, \mathbf{F}_i\, \mathbf{R}_i^{n-k-1}\, t^n}{n!} = \\ &= \sum_{n=0}^{\infty} \sum_{k=0}^{\infty} \mathbf{T}^n\, \mathbf{F}_i\, \mathbf{R}_i^k \frac{t^{n+k+1}}{(n+k+1)!} \end{aligned}$$

The last factor can be rewritten:

$$\frac{t^{n+k+1}}{(n+k+1)!} = \frac{1}{n!\,k!} \int_0^t x^n\, (t-x)^k\, \mathrm{d}x$$

so that the term $\boldsymbol{\Theta}_i(t)$ can be reformulated:

$$\begin{aligned}\boldsymbol{\Theta}_i(t) &= \sum_{n=0}^{\infty} \frac{\mathbf{T}^n}{n!} \cdot \mathbf{F}_i \cdot \left(\sum_{k=0}^{\infty} \frac{\mathbf{R}_i^k}{k!} \int_0^t x^n \, (t-x)^k \, \mathrm{d}x \right) = \\ &= \int_0^t \left(\sum_{n=0}^{\infty} \frac{\mathbf{T}^n \, x^n}{n!} \right) \cdot \mathbf{F}_i \cdot \left(\sum_{k=0}^{\infty} \frac{\mathbf{R}_i^k \, (t-x)^k}{k!} \right) \mathrm{d}x = \\ &= \int_0^t \mathrm{e}^{\mathbf{T}x} \cdot \mathbf{F}_i \cdot \mathrm{e}^{\mathbf{R}_i(t-x)} \, \mathrm{d}x \end{aligned} \tag{2.33}$$

Equation (2.33) can be interpreted as follows: the process initially passes x time units in the transient class $\mathcal{S}_\mathrm{T}$, then one $\mathbf{F}_i$ transition occurs, after which the process spends the remaining time $(t-x)$ in the i-th recurrent class.

Transient forward and backward equations for $\mathcal{S}_\mathrm{T}$ and $\mathcal{S}_{\mathrm{R}_i}$ state partitions can then be written in vector form as:

$$\begin{aligned} \boldsymbol{\pi}^{\mathcal{M}}_{\mathrm{T}}(\boldsymbol{\alpha}, t) &= \boldsymbol{\alpha}_\mathrm{T} \cdot \mathrm{e}^{\mathbf{T}t} \\ \boldsymbol{\pi}^{\mathcal{M}}_{\mathrm{R}_i}(\boldsymbol{\alpha}, t) &= \boldsymbol{\alpha}_\mathrm{T} \cdot \boldsymbol{\Theta}_i(t) + \boldsymbol{\alpha}_{\mathrm{R}_i} \cdot \mathrm{e}^{\mathbf{R}_i t} \\ \boldsymbol{\xi}^{\mathcal{M}}_{\mathrm{T}}(\boldsymbol{\rho}, t) &= \sum_{i=1}^{m} \big(\boldsymbol{\Theta}_i(t) \cdot \boldsymbol{\rho}_{\mathrm{R}_i} \big) + \mathrm{e}^{\mathbf{T}t} \cdot \boldsymbol{\rho}_\mathrm{T} \\ \boldsymbol{\xi}^{\mathcal{M}}_{\mathrm{R}_i}(\boldsymbol{\rho}, t) &= \mathrm{e}^{\mathbf{R}_i t} \cdot \boldsymbol{\rho}_{\mathrm{R}_i} \end{aligned} \tag{2.34}$$

Let $\mathbf{W}$ be a matrix defined as:

$$\mathbf{W} = \int_0^{\infty} \mathrm{e}^{\mathbf{T}x} \, \mathrm{d}x \tag{2.35}$$

so that each entry $\mathbf{W}(i,j)$ gives the expected sojourn time left in state j, before entering an absorption class, given that the initial state was i. With $\mathbf{W}$ it is possible to rewrite the limit of $\boldsymbol{\Theta}_i(t)$ by integration by parts:

$$\begin{aligned} \lim_{t\to\infty} \boldsymbol{\Theta}_i(t) &= \int_0^{\infty} \mathrm{e}^{\mathbf{T}x} \, \mathrm{d}x \cdot \mathbf{F}_i \cdot \lim_{t\to\infty} \mathrm{e}^{\mathbf{R}_i t} = \\ &= \mathbf{W} \cdot \mathbf{F}_i \cdot \lim_{t\to\infty} \mathrm{e}^{\mathbf{R}_i t} \end{aligned} \tag{2.36}$$

The term $\lim\limits_{t\to\infty} \mathrm{e}^{\mathbf{R}_i t}$ is the *stationary* stochastic matrix of the i-th recurrent class. The limiting behavior of a reducible Markov chain is therefore:

$$\lim_{t\to\infty} \mathrm{e}^{\mathbf{Q}t} = \begin{bmatrix} 0 & \mathbf{W} \cdot \mathbf{F}_1 \cdot \lim\limits_{t\to\infty} \mathrm{e}^{\mathbf{R}_1 t} & \cdots & \mathbf{W} \cdot \mathbf{F}_m \cdot \lim\limits_{t\to\infty} \mathrm{e}^{\mathbf{R}_m t} \\ 0 & \lim\limits_{t\to\infty} \mathrm{e}^{\mathbf{R}_1 t} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lim\limits_{t\to\infty} \mathrm{e}^{\mathbf{R}_m t} \end{bmatrix}$$

The limiting behavior of $\lim\limits_{t\to\infty} \mathrm{e}^{\mathbf{T}t}$ tends to 0, since $\mathbf{T}$ has at least one row that has a negative row sum. Therefore $\lim\limits_{t\to\infty} \mathrm{e}^{\mathbf{Q}t}$ has the column of $\mathcal{S}_\mathrm{T}$ states zeroed. Observe that since $\lim\limits_{t\to\infty} \mathrm{e}^{\mathbf{T}t} = \mathbf{0}$, then the integral (2.35) is equivalent to: $\mathbf{W} = -\mathbf{T}^{-1}$, so that a product with $\mathbf{W}$ can be computed as the solution of a linear system in $\mathbf{T}$.

Stationary expressions for $\boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha})$ and $\boldsymbol{\xi}^{\mathcal{M}}(\boldsymbol{\rho})$ are:

$$\begin{array}{rcl} \boldsymbol{\pi}^{\mathcal{M}}_{\mathrm{T}}(\boldsymbol{\alpha}) &=& 0 \\ \boldsymbol{\pi}^{\mathcal{M}}_{\mathrm{R}_i}(\boldsymbol{\alpha}) &=& \big(\boldsymbol{\alpha}_{\mathrm{T}} \cdot \mathbf{W} \cdot \mathbf{F}_i \;+\; \boldsymbol{\alpha}_{\mathrm{R}_i}\big) \cdot \lim\limits_{t\to\infty} \mathrm{e}^{\mathbf{R}_i t} \\ \boldsymbol{\xi}^{\mathcal{M}}_{\mathrm{T}}(\boldsymbol{\rho}) &=& \mathbf{W} \cdot \sum\limits_{i=1}^{m} \Big(\mathbf{F}_i \cdot \lim\limits_{t\to\infty} \mathrm{e}^{\mathbf{R}_i t} \cdot \boldsymbol{\rho}_{\mathrm{R}_i}\Big) \\ \boldsymbol{\xi}^{\mathcal{M}}_{\mathrm{R}_i}(\boldsymbol{\rho}) &=& \lim\limits_{t\to\infty} \mathrm{e}^{\mathbf{R}_i t} \cdot \boldsymbol{\rho}_{\mathrm{R}_i} \end{array} \tag{2.37}$$

and the same considerations done for the corresponding discrete equations (2.14) hold.

2.4 Markov Regenerative Processes

In this section we consider a class of stochastic processes that extends the CTMC definition (Def. 2) by allowing firing times to have any *general* distribution (*general events*), thus weakening the **M2** requirement. These processes, known as *Markov regenerative processes*, have been introduced in [Pyk59], and then studied in detail in [Ger00]. Events with an exponentially distributed firing times are called *exponential events.* General events that can fire from a state s are said to be *enabled.* A set of random variables G accounts for the age of the enabled general events. A *renewal time* is a point in time where the value of every random variable g is zero. States of the process encountered at renewal times are called *regeneration points.* A stochastic process $\{Z_t \mid t \in \mathbb{R}_{\geq 0}\}$ where general events are restricted to be enabled at most one per state [Cox55] is a *Markov Regenerative Process* (MRP), which can be described upon a *Markov Renewal Sequence* (MRS).

Definition 5 (Markov renewal sequence). Let $\mathcal{S}$ be a finite state space. A sequence of bivariate random variables $\{\langle Y_n, T_n\rangle \mid n \in \mathbb{N}\}$ is called a *Markov renewal sequence* with *regeneration points* $Y_n \in \mathcal{S}$ encountered at *renewal times* $T_n \in \mathbb{R}_{\geq 0}$ iff:

- $0 = T_0 < T_1 < T_2 < \dots$
- $Pr\{Y_{n+1}{=}j, T_{n+1}{-}T_n \leq t \mid Y_n{=}i, T_n, \dots, Y_0, T_0\} =$ (*Markov property*)
  $= Pr\{Y_{n+1}{=}j, T_{n+1}{-}T_n \leq t \mid Y_n{=}i\} =$ (*Time homogeneity*)
  $= Pr\{Y_1{=}j, T_1 \leq t \mid Y_0{=}i\}$

The *renewal sequence* is a stochastic process whose purpose is to count state transitions of a continuous-time process Z_t. When the renewal sequence is *memoryless*, the process Y_n is a discrete-time Markov chain, called the *embedded Markov chain* (EMC), while the process T_n is the time where the process Z_t jumps to a new state. Therefore, Z_t and Y_n are bound together by the relation $Z_{T_n} = Y_n$ for all $n \in \mathbb{N}$. In general, the process T_n is not a Markov process, since the durations $T_{n+1} - T_n$ are not i.i.d.[1], but depend on Y_n.

[1]Identically and independently distributed.

Definition 6 (**Markov regenerative process).** A stochastic process $\{Z_t \mid t \in \mathbb{R}_{\geq 0}\}$ is a *Markov regenerative process* if there exists an MRS $\{\langle Y_n, T_n\rangle \mid n \in \mathbb{N}\}$ such that all the conditional finite dimensional distributions of $\{Z_{T_n+t} \mid t \geq 0\}$ given the past history $\{Z_u \mid 0 \leq u \leq T_n, Y_n = i\}$ only depend on the last regeneration point $\langle Y_n, T_n\rangle$, i.e. are the same as $\{Z_t \mid t \geq 0\}$ given $Y_0 = i$, so that:

$$\begin{aligned} Pr\{Z_{T_n+t} = j \mid Z_u, 0 \leq u \leq T_n, Z_{T_n} = i\} &= \\ = \quad Pr\{Z_{T_n+t} = j \mid Z_{T_n} = i\} &= \\ = \quad Pr\{Z_t = j \mid Z_0 = i\} & \end{aligned}$$

Definition 6 implies that Z_t may not be memoryless between two renewal times T_n and T_{n+1}, but at T_{n+1} the process has no age memory, so it behavior depends only on the state reached. Therefore, the process Y_n that follows Z_{T_n} is a Markov chain.

The process behavior $\{Z_t \mid T_n \leq t < T_{n+1}\}$ between two regeneration points Y_n and Y_{n+1} may be described by any process: it is commonly described by a CTMC (called the *subordinated process* of Y_n), but other choices are possible.

An MRP may have an important property that is known as the *enabling-retriction*. In those processes, there is at most one general (non-memoryless) event enabled between two regeneration points Y_{n+1} and Y_n. All the MRPs considered in this thesis will have the *enabling-retriction*, so we drop the term "MRP with enabling retriction" ad we just use the term MRP.

2.4.1 Linear representation of a MRP

MRP analysis can be done at two levels: at the level of the continuous-time non-Markovian process Z_t, and at the level of the embedded DTMC Y_n. It is not possible to give a simple linear representation to Z_t (like an infinitesimal generator $\mathbf{Q}$), because it is not *memoryless*. On the contrary, Y_n is a Markov chain, for which it is possible to derive a stochastic matrix $\mathbf{P}$, which gives the transition probabilities between two consecutive regeneration points:

$$\mathbf{P}(i,j) \overset{\text{def}}{=} \Pr\{Y_{n+1} = j \mid Y_n = i\} = \Pr\{Y_1 = j \mid Y_0 = i\} \tag{2.38}$$

Every single discrete step of $\mathbf{P}$ gives the next regeneration point in the MRS. It is possible to convert the distributions of the regeneration points into the MRP distribution. However, the time elapsed between two regeneration points (the T_n sequence) is not uniformly distributed. Therefore, it is necessary to count how much time the embedded process spends in every subordinated process. A matrix of *conversion factors* is defined to count for these elapsed times:

$$\begin{aligned} \mathbf{C}(i,j) &\overset{\text{def}}{=} E\big[\text{time of } Z_t \text{ in state } j \text{ during } [0,T_1) \mid Y_0 = i\big] = \\ &= \int_0^{T_1} \Pr\{Z_t = j \mid Y_0 = i\}\,\mathrm{d}t \end{aligned} \tag{2.39}$$

The EMC definition is given with the notation of [Ger00]. Let G be the set of random variables that describe the ages of the general events. Given $g \in \text{G}$, we denote with $F^g(x)$ the *firing time distribution*, with $f^g(x)$ the *density function* of g, and with $x^g_{\max}$ the maximum support.

The set $\mathcal{S}^{\mathrm{E}} \subseteq \mathcal{S}$ is the *exponential state subset*, where no general event is enabled; $\mathcal{S}^g$ is the set of states where g is enabled, and $\mathcal{S}^{\mathrm{G}} = \bigcup_{g \in \mathrm{G}} \mathcal{S}^g$ is the *general state subset.*

State transitions are classified into three *kinds*:

[$\boldsymbol{\delta}$] Transition due to a general event completion (*firing* of a general event).

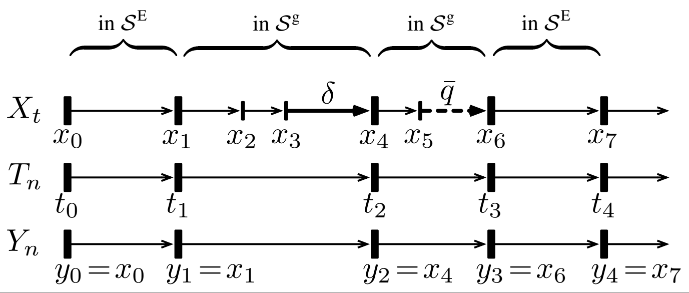
[$\mathbf{q}$] Exponential event whose firing has no effect on the age of the enabled general transition (*non-preemptive* firing of an exponential event).

[$\overline{\mathbf{q}}$] Exponential event whose firing resets the age of the currently enabled general transition (*preemptive* firing of an exponential event).

Events of type $\boldsymbol{\delta}$ and $\overline{\mathbf{q}}$ may only happen in a general state $i \in \mathcal{S}^{\mathrm{G}}$. The next state j depends on the current state i and on the age of the currently enabled general event g (if any), but not on the past history (the *semi-Markov* property, which relaxes the constraint **M2** of section 2). We assume a *preemptive repeat different* (*prd* policy) for general events: when a general event g is preempted, its memory is lost, so, when it will become active again, a new duration for that event is resampled from its random variable.

A sample dynamic of an MRP is depicted in Figure 2.1.

Figure 2.1 Sample MRP dynamic.


in $\mathcal{S}^{\mathrm{E}}$
in $\mathcal{S}^{g}$
in $\mathcal{S}^{g}$
in $\mathcal{S}^{\mathrm{E}}$
X_t
δ
$\bar{q}$
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7
T_n
t_0 t_1 t_2 t_3 t_4
Y_n
$y_0 = x_0$ $y_1 = x_1$ $y_2 = x_4$ $y_3 = x_6$ $y_4 = x_7$

In Figure 2.1 the MRP process Z_t passes through various states: from state x_1 to state x_4 a general event g is enabled, and this enabling ends with the firing event of g. The same happens from state x_4 to x_6, where the enabling of g ends due to an exponential event that disables g. Arcs of type $\mathbf{q}$ are not labeled. The age of g is not 0 in states x_2, x_3 and x_5, therefore these states are not regeneration points, and do not appear in the embedded process Y_n.

The dynamic of Z_t can be described [Ger00] by 3 matrices $\mathbf{Q}$, $\bar{\mathbf{Q}}$ and $\boldsymbol{\Delta}$:

- $\mathbf{Q}(i,j)$, $i \neq j$: non-preemptive exponential events rate from state i to state j;
- $\bar{\mathbf{Q}}(i,j), i \in \mathcal{S}^g$: preemptive exponential events rate from state i to state j that disable g enabled in i;
- $\mathbf{Q}(i,i)$ is the negative sum of all the rates of exponential events leaving state i;
- $\boldsymbol{\Delta}(i,j), i \in \mathcal{S}^g$: probability that the firing of g in state i leads to state j.

Observe that $\bar{\mathbf{Q}}(i,i)$ can indeed be non-zero as well (a self-loop with preemption), which represents a state transition that does not change the state but resets the age of g, thus making i a new regeneration point in the MRS. The diagonal of $\mathbf{Q}$ accounts for the rates of both $\mathbf{Q}$ and $\bar{\mathbf{Q}}$, i.e. $\mathbf{Q}(i,i) =$

$-\sum_{j\neq i}\big(\mathbf{Q}(i,j) + \bar{\mathbf{Q}}(i,j)\big)$. Rows of $\bar{\mathbf{Q}}$ and $\boldsymbol{\Delta}$ corresponding to exponential states are zero. The matrix $\boldsymbol{\Delta}$ is a stochastic matrix, while $\mathbf{Q}$ and $\bar{\mathbf{Q}}$ are rate matrixes.

MRP formulas are simpler with the help of a *filtering* notation.

Definition 7 (Filtering)**.** Let $\mathcal{S}^U \subseteq \mathcal{S}$ be a subset of states of $\mathcal{S}$, and let $\mathbf{A}$ be a matrix. With $\mathbf{A}^U$ we denote the *filtered matrix* where all rows which do not correspond to $\mathcal{S}^U$ states are zeroed. With $\mathbf{A}^{U,V}$ we denote a *doubly-filtered matrix* where the rows not in U and the columns not in V are zeroed.

For instance, $\mathbf{I}^{\mathrm{G}}$, $\bar{\mathbf{Q}}^g$ and $\mathbf{Q}^{\mathrm{E}}$ are all filtered matrices w.r.t. the set of states with g enabled, the general subset $\mathcal{S}^{\mathrm{G}}$, and the exponential subsets of $\mathcal{S}$. The filtering notation will be used extensively also for vectors.

2.4.2 Markov Regenerative Transition System

An MRP can be represented more compactly as a *transition system*, as done for def. (3) and (4). The three *transition types* $\{\mathbf{q}, \overline{\mathbf{q}}, \boldsymbol{\delta}\}$ are represented as *action types.* The set of actions Act is partitioned into three disjoint subsets:

- Act^{G} is the set of general actions (type $\boldsymbol{\delta}$), one $a_g \in Act^{\mathrm{G}}$ for each $g \in \mathrm{G}$;
- Act^{E} is the set of *non-preemptive* Markovian transitions (type $\mathbf{q}$);
- $\overline{Act}^{\mathrm{E}}$ is the set of *preemptive* Markovian transitions (type $\overline{\mathbf{q}}$).

Let $Act^{\mathrm{M}} = Act^{\mathrm{E}} \cup \overline{Act}^{\mathrm{E}}$ be the set of Markovian actions.

Definition 8 (MRTS)**.** A *Markov Regenerative Transition System* (MRTS) is a tuple $\mathcal{R} = \langle \mathcal{S}, \mathrm{G}, \Gamma, Act, AP, lab, T, P, E, \boldsymbol{\alpha}_0\rangle$ where:

- $\mathcal{S}$ is a finite set of *states.*
- G is a finite set of *general distributions*, with $F_g(x)$ the probability mass function of $g \in \mathrm{G}$.
- $\Gamma : \mathcal{S} \to \mathrm{G} \uplus \{\mathrm{E}\}$ is a *state-partition* function that assigns to each state s a general transition enabled in s, or E if no general transition is enabled in s; Function Γ induces the partitioning of $\mathcal{S}$ into the set of *exponential states* $\mathcal{S}^{\mathrm{E}} = \{s \in \mathcal{S} \mid \Gamma(s) = \mathrm{E}\}$ and, for each $g \in \mathrm{G}$, the set where g is enabled $\mathcal{S}^g = \{s \in \mathcal{S} \mid \Gamma(s) = g\}$.
- $Act = Act^{\mathrm{G}} \uplus Act^{\mathrm{E}} \uplus \overline{Act}^{\mathrm{E}}$ is a finite set of *action* names;
- AP is a finite set of *atomic propositions.*
- $lab : \mathcal{S} \to 2^{AP}$ is a *state-labeling* function.
- $T \subseteq \mathcal{S} \times Act \times \mathcal{S}$ is a *transition* relation, such that:
$$T \setminus \Big[\Big(\bigcup_{g\in \mathrm{G}}\Big(\big(\mathcal{S}^g \times Act^{\mathrm{E}} \times \mathcal{S}^g\big) \cup \big(\mathcal{S}^g \times (\overline{Act}^{\mathrm{E}} \cup \{a_g\}) \times \mathcal{S}\big)\Big)\Big) \cup \\ \cup \big(\mathcal{S}^{\mathrm{E}} \times Act^{\mathrm{E}} \times \mathcal{S}\big)\Big] = \varnothing$$
- $P : T \to \mathbb{R}_{(0,1]}$ is a *transition probability* function, subject to:
  1. for all states $s \in \mathcal{S}$ it holds that: $\sum_{a\in Act^{\mathrm{M}}} P(s,a) \in Distr^0(\mathcal{S})$;
  2. for all states $s \in \mathcal{S}^{\mathrm{G}}$ it holds that: $\sum_{a\in Act^{\mathrm{G}}} P(s,a) \in Distr(\mathcal{S})$;
- $E : \mathcal{S} \to \mathbb{R}_{>0}$ is the Markovian *exit rate* function.
- $\boldsymbol{\alpha}_0 \in Distr(\mathcal{S})$ is an initial distribution.

As before, the set $\mathcal{S}^{\mathrm{G}}$ is defined as $\cup_{g \in \mathrm{G}} \mathcal{S}^g$. The two notations $i \xrightarrow{a} j \sim \lambda$ and $i \xrightarrow{a} j \sim_{\text{rate}} \mu$ mean that the transition $\tau = i \xrightarrow{a} j$ has a probability $P(\tau) = \lambda$ and an exit rate $E(i) \cdot P(\tau) = \mu$, respectively, with $\lambda \in \mathbb{R}_{(0,1]}$ and $\mu \in \mathbb{R}_{>0}$. The rate notation is used only for Markovian transitions.

The restrictions on the T relation describes which kind of transitions may depart from exponential and general states. Note that in a MRTS the probability relation P accounts separately for general and Markovian actions.

An MRTS $\mathcal{R}$ starts in a state $s \sim \boldsymbol{\alpha}_0$. States where $\Gamma(s) = g$ are represented by a tuple $\langle s, x \rangle$, where x is the *age* of the enabled general event g. The value of the variable x increases linearly over time. Let $\bar{x}$ be the value of x. The evolution of $\mathcal{R}$ is governed by the transition relation T. In each state, either an exponential event or a general event may fire. If either a general event or preemptive exponential event fire, the value of x is reset to zero. The probability of taking a transition $\tau \in T(s)$ is given by $P(\tau)$. General transitions and exponential transitions have separate probability distributions.

2.4.3 The embedded DTMC

The time evolution of the EMC follows that of the regeneration points, and the evolution from one regeneration point Y_n to the next Y_{n+1} is given by the subordinated CTMC of state Y_n (i.e. each state i encountered as a regeneration point Y_n has its own subordinated CTMC). Therefore it is possible to isolate the behaviour of the sole embedded process Y_n, for which a DTMC can be provided. The steady-state solution of the EMC is called the *embedded solution*, and gives the expected probability of encountering a state as a regeneration point. The usefulness of the conversion factors matrix $\mathbf{C}$ is that it allows to convert from the *embedded solution* $\boldsymbol{\pi}^{\mathcal{D}}$ to the *MRP solution* $\boldsymbol{\pi}^{\mathcal{R}}$, thus taking the expected probabilities of the Z_n process.

We now introduce some basic definitions, in order to derive a closed-form expression for $\mathbf{P}$ and $\mathbf{C}$. Following [Ger00], let $\boldsymbol{\Omega}^g$ be the *state probability matrix in the instant before g fires*, defined for a subset $\mathcal{S}^g$ as:

$$\begin{aligned} \boldsymbol{\Omega}^g(i,j) &\overset{\text{def}}{=} Pr\{Z_{T_1^-} = j \mid Y_0 = i,\, i \in \mathcal{S}^g\} = \\ &= \mathbf{I}^g \int_0^{x^g_{\max}} \mathrm{e}^{\mathbf{Q}^g x} \cdot f^g(x)\, \mathrm{d}x \end{aligned} \tag{2.40}$$

Let $\boldsymbol{\Psi}^g$ be the *conditional expected sojourn time matrix* in the states of Z_t from the enabling of g to the firing, defined for a subset $\mathcal{S}^g$ as:

$$\begin{aligned} \boldsymbol{\Psi}^g(i,j) &\overset{\text{def}}{=} E\big[\text{sojourn time of } X \text{ in } j \text{ during } [0,T_1),\ \mid Y_0 = i,\, i \in \mathcal{S}^g\big] = \\ &= \int_0^{T_1} Pr\{X(\tau) = j \mid Y_0 = i,\, i \in \mathcal{S}^g\}\, \mathrm{d}\tau = \\ &= \mathbf{I}^g \int_0^{x^g_{\max}} \mathrm{e}^{\mathbf{Q}^g x} \cdot (1 - F^g(x))\, \mathrm{d}x \end{aligned} \tag{2.41}$$

The two shorthand notations:

$$\boldsymbol{\Omega} = \sum_{g \in \mathrm{G}} \boldsymbol{\Omega}^g, \qquad \boldsymbol{\Psi} = \sum_{g \in \mathrm{G}} \boldsymbol{\Psi}^g$$

denote the summation of (2.40) and (2.41) for each general event.

Let $\mathsf{diag}^{-1}(\mathbf{Q}^{\mathrm{E}})$ be the diagonal matrix where the non-zero diagonal entry of every exponential state $i \in \mathcal{S}^{\mathrm{E}}$ is $\mathbf{Q}(i,i)^{-1}$, and 0 in every other entry. In the exponential state subset $\mathcal{S}^{\mathrm{E}}$, the process $\{Z_t\}$ behaves like a CTMC. Therefore the next renewal state is reached with the next exponential event fires. Expected sojourn times are just the reciprocal of the rates in each state:

$$\mathbf{C}^{\mathrm{E}} \;=\; -\mathsf{diag}^{-1}(\mathbf{Q}^{\mathrm{E}})$$

so that $\mathbf{C}^{\mathrm{E}}(i,i) = -1/\mathbf{Q}^{\mathrm{E}}(i,i)$ when $\mathbf{Q}^{\mathrm{E}}(i,i) \neq 0$. The rows of $\mathbf{P}^{\mathrm{E}}$ are:

$$\mathbf{P}^{\mathrm{E}}(i,j) \;=\; \begin{cases} \frac{\mathbf{Q}(i,j)}{\mathbf{Q}(i,i)} & \text{if } i \neq j,\ \mathbf{Q}(i,i) \neq 0,\ i \in \mathcal{S}^{\mathrm{E}} \\ 0 & \text{otherwise} \end{cases}$$

or, alternatively, in matrix notation:

$$\mathbf{P}^{\mathrm{E}} \;=\; \mathbf{I}^{\mathrm{E}} - \mathsf{diag}^{-1}(\mathbf{Q}^{\mathrm{E}})\mathbf{Q}^{\mathrm{E}} \;=\; \mathbf{I}^{\mathrm{E}} + \mathbf{C}^{\mathrm{E}}\,\mathbf{Q}^{\mathrm{E}} \tag{2.42}$$

When Z_t enters a general state $s \in \mathcal{S}^g$, the instant when either g will be preempted or g will fire constitutes the next renewal time. Let $\bar{g}$ be the valuation of event g, randomly sampled from the distribution $F_g(x)$ of g at time T_0. Starting from (2.38) we split the age reset case (when the renewal is $T_1 < \bar{g}$) from the activation case (when $T_1 = \bar{g}$), obtaining:

$$\begin{aligned}\mathbf{P}^g(i,j) \;=\;& \underbrace{Pr\{Y_1 = j \mid Y_0 = i, i \in \mathcal{S}^g, T_1 < \bar{g}\}}_{\text{Probability that } g \text{ gets preempted}} + \\ &+ \underbrace{Pr\{Y_1 = j \mid Y_0 = i, i \in \mathcal{S}^g, T_1 = \bar{g}\}}_{\text{Probability that } g \text{ fires}}\end{aligned}$$

A matrix form for $\mathbf{P}^g$ is then derivable with the help of a SMC matrix.

Definition 9 (Subordinated Markov chain matrix)**.** Let $\mathbf{B}^g$ be a generator matrix of the subordinated Markov process of Y_0 for the $\mathcal{S}^g$ state subset, defined over a duplicated state space $\{\mathcal{S} \cup \bar{\mathcal{S}}\}$ as:

$$\mathbf{B}^g \;=\; \left[\begin{array}{c|c} \mathbf{Q}^g & \bar{\mathbf{Q}}^g \\ \hline \mathbf{0} & \mathbf{0} \end{array}\right]$$

where the set of states $\mathcal{S}$ gives the behavior before the firing of g, and the set of absorbing states $\bar{\mathcal{S}}$ are the destination states when a preemptive transition fires, disabling g.

The EMC matrix $\mathbf{P}^g$ for $\mathcal{S}^g$-states is:

$$\mathbf{P}^g \;=\; \begin{bmatrix}\mathbf{I}^g & \mathbf{0}\end{bmatrix} \left(\lim_{t\to\infty} e^{\mathbf{B}^g t}\right) \begin{bmatrix}\mathbf{\Delta}^g \\ \mathbf{I}\end{bmatrix} \tag{2.43}$$

The subordinated process starts in the $\mathcal{S}$-states of $\mathbf{B}^g$, as described by the left matrix factor $[\mathbf{I}^g \;\; \mathbf{0}]$. If no preemptive transition in $\bar{\mathbf{Q}}^g$ is taken up to time t, the process goes to the next regeneration point according to the branching distribution in $\mathbf{\Delta}^g$. Otherwise, when a $\bar{\mathbf{Q}}^g$ transition fires, the process goes to an absorbing $\bar{\mathcal{S}}$ state, which gives the next regeneration point.

Theorem 1. *Given the SMC $\mathbf{B}^g$ of definition 9, the limiting behavior of the matrix exponential of $\mathbf{B}^g$ is given by:*

$$\lim_{t\to\infty} \int_0^t e^{\mathbf{B}^g x} \cdot f^g(x)\, \mathrm{d}x \;=\; \left[\begin{array}{c|c} \mathbf{\Omega}^g & \mathbf{\Psi}^g\ \bar{\mathbf{Q}}^g \\ \hline \mathbf{0} & \mathbf{I} \end{array}\right] \tag{2.44}$$

with the matrixes $\mathbf{\Omega}^g$ and $\mathbf{\Psi}^g$ as defined in Eq. (2.40) and (2.41).

Proof. The powers of the subordinated Markov chain matrix $\mathbf{B}^g$ are:

$$(\mathbf{B}^g)^0 = \left[\begin{array}{c|c} \mathbf{I} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{I} \end{array}\right], \qquad (\mathbf{B}^g)^k = \left[\begin{array}{c|c} (\mathbf{Q}^g)^k & (\mathbf{Q}^g)^{k-1}\ \bar{\mathbf{Q}}^g \\ \hline \mathbf{0} & \mathbf{0} \end{array}\right]$$

Expanding the Taylor series of the matrix exponential $e^{\mathbf{B}^g t}$ results in:

$$e^{\mathbf{B}^g x} = \sum_{k=0}^{\infty} \frac{(\mathbf{B}^g x)^k}{k!} = \left[\begin{array}{c|c} \sum_{k=0}^{\infty} \frac{(\mathbf{Q}^g x)^k}{k!} & \sum_{k=1}^{\infty} \frac{x^k (\mathbf{Q}^g)^{k-1} \bar{\mathbf{Q}}^g}{k!} \\ \hline \mathbf{0} & \mathbf{I} \end{array}\right] = \left[\begin{array}{c|c} e^{\mathbf{Q}^g x} & \star \\ \hline \mathbf{0} & \mathbf{I} \end{array}\right]$$

Equation (2.44) is therefore:

$$\lim_{t\to\infty} \int_0^t e^{\mathbf{B}^g x} \cdot f^g(x)\, \mathrm{d}x \;=\; \lim_{t\to\infty} \int_0^t \left[\begin{array}{c|c} e^{\mathbf{Q}^g x} & \star \\ \hline \mathbf{0} & \mathbf{I} \end{array}\right] \cdot f^g(x)\, \mathrm{d}x$$

The term $\lim\limits_{t\to\infty} \int_0^t e^{\mathbf{Q}^g t} \cdot f^g(x)\, \mathrm{d}x$ is equivalent to $\mathbf{\Omega}^g$, by definition (2.40). The $\star$ term can be rewritten as:

$$\sum_{k=1}^{\infty} \frac{x^k\ (\mathbf{Q}^g)^{k-1}\ \bar{\mathbf{Q}}^g}{k!} \;=\; \sum_{k=0}^{\infty} \frac{x^{k+1}\ (\mathbf{Q}^g)^k}{(k+1)!} \bar{\mathbf{Q}}^g$$

The conditional expected sojourn time $\mathbf{\Psi}^g$ of Eq. (2.41) can be rewritten as:

$$\begin{aligned}
\mathbf{\Psi}^g \;&=\; \lim_{t\to\infty} \mathbf{I}^g \int_0^t e^{\mathbf{Q}^g x} \cdot \bar{F}^g(x)\, \mathrm{d}x = \lim_{t\to\infty} \mathbf{I}^g \int_0^t \sum_{k=0}^{\infty} \frac{(\mathbf{Q}^g x)^k}{k!} \cdot \bar{F}^g(x)\, \mathrm{d}x \;= \\
&=\; \lim_{t\to\infty} \mathbf{I}^g \sum_{k=0}^{\infty} \frac{(\mathbf{Q}^g)^k}{k!} \int_0^t x^k \cdot \bar{F}^g(x)\, \mathrm{d}x \;= \\
&=\; \lim_{t\to\infty} \mathbf{I}^g \sum_{k=0}^{\infty} \frac{(\mathbf{Q}^g)^k}{k!} \left(\frac{x^{k+1} \cdot \bar{F}^g(x)}{k+1} \bigg|_0^t \right) \;= \\
&=\; \lim_{t\to\infty} \mathbf{I}^g \sum_{k=0}^{\infty} \frac{(\mathbf{Q}^g)^k\ x^{k+1}}{(k+1)!} \;=\; \lim_{t\to\infty} \mathbf{I}^g \sum_{k=0}^{\infty} \frac{x^{k+1}\ (\mathbf{Q}^g)^k}{(k+1)!}
\end{aligned}$$

which implies that the term $\lim\limits_{t\to\infty} \int_0^t \star \cdot f^g(x)\, \mathrm{d}x$ is exactly $\mathbf{\Psi}^g\ \bar{\mathbf{Q}}^g$. □

The relation (2.44) of theorem 1 allows to simplify the formula for $\mathbf{P}^g$.

Corollary 2. *Theorem 1 allows to derive a compact analytical formula for the EMC matrix (2.43) for $\mathcal{S}^g$ states:*

$$\begin{aligned}
\mathbf{P}^g \;&=\; \begin{bmatrix} \mathbf{I}^g & \mathbf{0} \end{bmatrix} \left(\lim_{t\to\infty} e^{\mathbf{B}^g t} \right) \begin{bmatrix} \mathbf{\Delta}^g \\ \mathbf{I} \end{bmatrix} \;= \\
&=\; \begin{bmatrix} \mathbf{I}^g & \mathbf{0} \end{bmatrix} \left[\begin{array}{c|c} \mathbf{\Omega}^g & \mathbf{\Psi}^g\ \bar{\mathbf{Q}}^g \\ \hline \mathbf{0} & \mathbf{I} \end{array}\right] \begin{bmatrix} \mathbf{\Delta}^g \\ \mathbf{I} \end{bmatrix} \;= \\
&=\; \mathbf{\Omega}^g\, \mathbf{\Delta}^g + \mathbf{\Psi}^g\, \bar{\mathbf{Q}}^g
\end{aligned}$$

The stochastic matrix $\mathbf{P}$ of the DTMC embedded at renewal points is then [Ger00]:

$$\mathbf{P} \;=\; \mathbf{I}^{\mathrm{E}} - \mathsf{diag}^{-1}(\mathbf{Q}^{\mathrm{E}})\mathbf{Q}^{\mathrm{E}} + \mathbf{\Omega}\mathbf{\Delta} + \mathbf{\Psi}\bar{\mathbf{Q}} \tag{2.45}$$

which is a compact formulation for $\mathbf{P}$. Such a matrix can be used to compute the steady-state distribution of the MRP.

For general states, the matrix of conversion factors $\mathbf{C}^g$ is easily given by:

$$\mathbf{C}^g \;=\; \mathbf{\Psi}^g$$

since the definition of $\mathbf{C}$ of Eq. (2.39) is the same as $\mathbf{\Psi}$ (2.41) for $\mathcal{S}^g$ states. Therefore, the conversion factors matrix $\mathbf{C}$ of the MRP is:

$$\mathbf{C} \;=\; -\mathsf{diag}^{-1}(\mathbf{Q}^{\mathrm{E}}) + \mathbf{\Psi} \tag{2.46}$$

2.4.4 Stationary behavior of MRPs

Once the EMC matrix $\mathbf{P}$ and the conversion factors matrix $\mathbf{C}$ have been derived with (2.45) and (2.46), the stationary distribution of the embedded process Y_n can be computed using the DTMC formula (2.8) and (2.9). We assume, for now, that Y_n is irreducible.

The *forward* and *backward* stationary behaviors of the embedded process $\mathcal{D}$ of an MRP $\mathcal{R}$ are:

$$\boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha}) \;=\; \boldsymbol{\alpha} \cdot \lim_{n\to\infty} \mathbf{P}^n \tag{2.47}$$

$$\boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}) \;=\; \lim_{n\to\infty} \mathbf{P}^n \cdot \boldsymbol{\rho} \tag{2.48}$$

We may also write $\boldsymbol{\pi}^{\mathcal{D}}$ instead of $\boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha})$ since it does not depend on $\boldsymbol{\alpha}$.

The forward behavior of the MRP is then derived from that of the EMC as:

$$\boldsymbol{\pi}^{\mathcal{R}} \;=\; \frac{\boldsymbol{\pi}^{\mathcal{D}} \cdot \mathbf{C}}{\boldsymbol{\pi}^{\mathcal{D}} \cdot \mathbf{C} \cdot \mathbf{e}} \tag{2.49}$$

where $\boldsymbol{\pi}^{\mathcal{D}} \cdot \mathbf{C}$ is the conversion of the embedded probability into a (not normalized) MRP probability, and $\boldsymbol{\pi}^{\mathcal{D}} \cdot \mathbf{C} \cdot \mathbf{e}$ is the normalization coefficient. This normalization is needed because a product with $\mathbf{C}$ turns a probability distribution vector into a vector of mean sojourn times. Therefore, the product has to be re-normalized to be a probability distribution. When $\mathcal{R}$ is reducible, the normalization has to be done separately for each recurrent class and not globally.

Given a product $\boldsymbol{\pi}^{\mathcal{R}}(\boldsymbol{\alpha}) \cdot \boldsymbol{\rho}$, it is possible to derive a backward formulation of (2.49), by taking the limiting condition $\boldsymbol{\rho}$:

$$\boldsymbol{\pi}^{\mathcal{R}}(\boldsymbol{\alpha}) \cdot \boldsymbol{\rho} \;=\; \frac{\boldsymbol{\pi}^{\mathcal{D}} \cdot \mathbf{C} \cdot \boldsymbol{\rho}}{\boldsymbol{\pi}^{\mathcal{D}} \cdot \mathbf{C} \cdot \mathbf{e}} \;=\; \frac{\boldsymbol{\alpha} \cdot \lim\limits_{n\to\infty} \mathbf{P}^n \cdot \mathbf{C} \cdot \boldsymbol{\rho}}{\boldsymbol{\alpha} \cdot \lim\limits_{n\to\infty} \mathbf{P}^n \cdot \mathbf{C} \cdot \mathbf{e}} \;=\; \frac{\boldsymbol{\alpha} \cdot \boldsymbol{\xi}^{\mathcal{D}}(\mathbf{C} \cdot \boldsymbol{\rho})}{\boldsymbol{\alpha} \cdot \boldsymbol{\xi}^{\mathcal{D}}(\mathbf{C} \cdot \mathbf{e})} \tag{2.50}$$

With (2.50) the same product $\boldsymbol{\pi}^{\mathcal{R}}(\boldsymbol{\alpha}) \cdot \boldsymbol{\rho}$ for multiple initial distributions $\boldsymbol{\alpha}$ can be computed more efficiently by precomputing the two backward distributions $\boldsymbol{\xi}^{\mathcal{D}}(\mathbf{C} \cdot \boldsymbol{\rho})$ and $\boldsymbol{\xi}^{\mathcal{D}}(\mathbf{C} \cdot \mathbf{e})$. If $\mathcal{R}$ is irreducible, then $\boldsymbol{\xi}^{\mathcal{D}}(\mathbf{C} \cdot \mathbf{e})$ is a uniform vector $\mathbf{e}\,\xi$, for some value $\xi \in \mathbb{R}_{\geq 0}$, and any product $\boldsymbol{\alpha} \cdot \boldsymbol{\xi}^{\mathcal{D}}(\mathbf{C} \cdot \mathbf{e})$ has value $\|\boldsymbol{\alpha}\| \cdot \xi$. For reducible MRPs, the normalization coefficient is not unique but instead depends on $\boldsymbol{\alpha}$.

2.5 Measuring indexes on Markov Chains

The usual purpose for computing the forward distribution or the backward measure on a Markov chain at time t is that we want to use the computed distribution with some form of knowledge on the state space to determine quantitatively a scalar measure. Two important classes of scalar measures may be computed from a stochastic process, namely *state*-based measures and *event*-based measures (or *action*-based measures). Following the conventions of [Ger00, sec. 4.3], a formalism can be derived which is based on *rate rewards* and *impulse rewards*. States and actions can be taken from the labels of the transition system that describes the process.

State-based measures

A *rate reward* $\mathbf{rr} : \mathcal{S} \to \mathbb{R}$ is a function that assigns to each state s in $\mathcal{S}$ a real value. Typically, rate rewards are defined as expression over the set of atomic propositions AP, like $\mathbf{rr}(i) = (3 \text{ if } i \models \Phi)$, with Φ a proposition expression over AP.

A *state reward variable* $R_{\mathbf{r}}(t)$ is a scalar variable that accumulates $\mathbf{rr}(i)$ units whenever a given process spends time in state $s \in \mathcal{S}$ in the interval $[0, t]$. State reward variables may be defined for DTMCs, CTMCs and MRPs. In a DTMC Y_n, the variable $R^{\mathcal{D}}_{\mathbf{r}}(n)$ accumulates $\mathbf{rr}(n)$ in every step, where i is the state of that step.

$$R^{\mathcal{D}}_{\mathbf{r}}(n) \;=\; \sum_{i=0}^{n} \mathbf{rr}(Y_i) \tag{2.51}$$

In a CTMC X_t (or in an MRP Z_t), the variable $R^{\mathcal{M}}_{\mathbf{r}}(t)$ accumulates $\mathbf{rr}(i) \cdot t$ every times the process stays in state i for an amount of time of t units.

$$R^{\mathcal{M}}_{\mathbf{r}}(t) \;=\; \int_0^t \mathbf{rr}(X_\tau)\,\mathrm{d}\tau \tag{2.52}$$

Event-based measures

An *impulse reward* $\mathbf{ir} : T \to \mathbb{R}$ is a function that assigns a positive real value to each transition event $\tau \in T$ of the transition system. An *impulse reward variable* $R_{\mathbf{i}}(t)$ is a scalar variable that accumulates impulse rewards whenever a transition $\tau \in T$ takes place in the process during the interval $[0, t]$. For instance, an impulse reward $\mathbf{ir}_A(i \xrightarrow{a} j) = (1 \text{ if } a \in A)$ assigns a value of 1 whenever an action $a \in A$ happens. An event-based measure can be computed by having the probability of each state combined with the event frequency from each state.

In a DTMC a transition event happens in every discrete step. An *impulse reward variable* $R^{\mathcal{D}}_{\mathbf{i}}(n)$ is defined as:

$$R^{\mathcal{D}}_{\mathbf{i}}(n) \;=\; \sum_{i=0}^{n-1} \sum_{a \in Act} \mathbf{ir}(Y_i, a, Y_{i+1}) \tag{2.53}$$

In a CTMC, the definition of reward variables is given by taking the sequence $\{T_n \mid n \in \mathbb{N}\}$ of time instants where X_t takes a transition event. An *impulse* *reward variable* $R_{\mathbf{i}}^{\mathcal{M}}(n)$ is then defined as:

$$R_{\mathbf{i}}^{\mathcal{M}}(t) \;=\; \sum_{n=0}^{T_{n-1}\le t} \sum_{a\in Act} \mathbf{ir}(X_{T_n}, a, X_{T_{n+1}}) \cdot (T_{n+1} - T_n) \tag{2.54}$$

The firing frequency of exponential event $\tau = i \xrightarrow{a} j$ in a CTMC is just the exponential rate λ of the event τ. Therefore, the *impulse reward variable* $R_{\mathbf{i}}^{\mathcal{M}}(t)$ that measures the event τ grows linearly in time, as shown in Figure 2.2(a).

Exponential events in an MRP behaves in the same way as CTMC events. A general event g is instead more complex, and its behavior is not constant during the time t because g is not memoryless.

Figure 2.2 Firing frequencies of Markovian and general (deterministic) events.

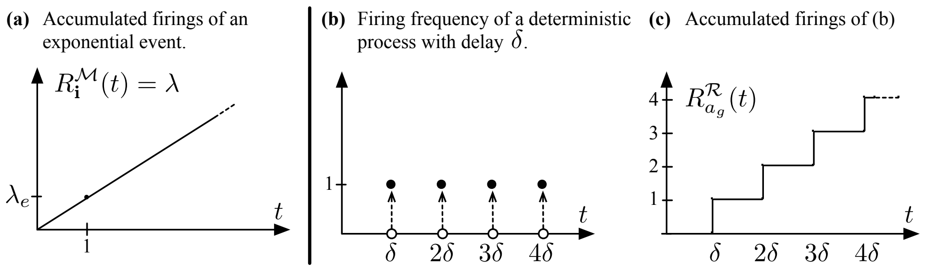


Figure 2.2(b) and (c) shows instantaneous and accumulated firing frequencies of a general event g with deterministic duration δ. In this case, firing frequencies can be represented as Dirac impulses in the roundings of $k\delta$, $k > 0$, where the probability of g firings is 1. Therefore, the values of $R_g^{\mathcal{R}}(t)$ strictly depends on the time where g has become active, i.e. it depends on t. Other general events with distributions different from the deterministic distribution have a different firing frequency diagram.

The reward variable for an exponential event $R_{\mathbf{i}}^{\mathcal{M}}(t)$ has the same diagram of a *state reward variable* with a reward $\mathbf{rr}(i)$ equivalent to the exit rate of the measured event in i. Therefore $R_{\mathbf{i}}^{\mathcal{M}}(t)$ can be assimilated to a state reward variable. Impulse rewards are a concept relevant only for general event firings, and are of no use for regular Markov chains. Since MRP restrict general events to be enabled at most one at a time, it is possible to redefine $\mathbf{i} : \mathcal{S} \to \mathbb{R}$ as a function that assigns a reward in a state i if the measured event g fires in that state. Let $\boldsymbol{\varphi}$ be a vector defined as:

$$\boldsymbol{\varphi}(i) \overset{\text{def}}{=} E[\text{probability of } g \text{ firing in state } i \text{ from 0 to } T_1] \tag{2.55}$$

The stationary value of $\boldsymbol{\varphi}$ has been derived in [Ger00, p. 272] from the forward stationary distribution $\boldsymbol{\pi}^{\mathcal{R}}$:

$$\boldsymbol{\varphi} \;=\; \boldsymbol{\pi}^{\mathcal{R}}\,\boldsymbol{\Omega} \;=\; \frac{\mathbf{u}\boldsymbol{\Omega}}{\mathbf{uCe}} \tag{2.56}$$

with $\mathbf{u} = \mathbf{uP}$ subject to $\mathbf{ue} = 1$.

2.5.1 Reward-based measure definitions

The advantage of defining quantitative properties with rate and impulse rewards is [Ger00, p. 47] that it provides a systematic framework that can express a large variety of measures. The structure is very similar for CTMCs, DTMCs and MRPs.

For MRPs, it is convenient to define a *reward structure* for a quantitative measure as a vector of rate rewards **rr** and, for each general event g, a vector of impulse rewards $\mathbf{ir}^g$: Given a reward structure, four measures can be computed in an MRP:

$$R_{\text{inst}}(t) \overset{\text{def}}{=} \mathbf{rr} \cdot \boldsymbol{\pi}(t) + \sum_{g \in \mathrm{G}} \mathbf{ir}^g \cdot \boldsymbol{\varphi}^g(t) \tag{2.57}$$

$$R_{\text{steady}} \overset{\text{def}}{=} \lim_{t \to \infty} R_{\text{inst}}(t) \tag{2.58}$$

$$R_{\text{acc}}(t) \overset{\text{def}}{=} \int_0^t R_{\text{inst}}(x)\, \mathrm{d}x \tag{2.59}$$

$$R_{\text{average}}(t) \overset{\text{def}}{=} \frac{1}{t} R_{\text{acc}}(t) \tag{2.60}$$

The value of $R_{\text{inst}}(t)$ gives the instantaneous reward in a specified time instant t. The expected instantaneous reward in the long-run is given by R_{steady}. The value of $R_{\text{acc}}(t)$ gives the accumulated reward until time t, and $R_{\text{average}}(t)$ gives the average reward in the interval $[0, t]$. For continuous-time Markov processes, the accumulated reward $R_{\text{acc}}(t)$ is a continuous measure, which increases over the time when the process stays in a state with non-zero reward rate, and for the reward of exponential transition firings. In a MRP, the value of $R_{\text{acc}}(t)$ may contain discontinuity due to the impulses of general event firings.

The same kind of measures can be defined for CTMCs and DTMCs, given that the value of **ir** is not needed.

A quite different approach to measuring performance indexes on Markov chains is that of assigning rewards to execution *paths* of the entire Markov chain, instead of taking only isolated states/events. In this way, performance indexes may follow a property that describes an accepted behaviour of the chain. This approach is discussed in chapter 3 and is commonly referred to as *stochastic model checking*.

Chapter 3

Measuring path-based properties

The framework introduced in chapter 2 defines three classes of processes, namely DTMCs, CTMS and MRPs, and allows to define and compute various performance indexes based on per-state and per-event reward measures. For each of these three classes, a *labeled transition system* is described, which incorporates the stochastic behavior along with state labels and action names.

In the area of formal methods, many powerful systems have been defined to express temporal properties of systems. Systems are usually modeled as *transition systems*. Temporal properties allows to specify the dynamics of those systems by describing the allowed executions of the system. An important temporal logics of this kind is CTL (*Computation Tree Logic*) [EC82], which allows the formulation of properties over path of transition systems. A typical CTL property is the verification that, when some initial conditions are met, all possible executions of a program avoid some undesirable condition. Verifying such a property requires a software tool, known as *model checker*, to construct all the possible execution of the model from the initial states and to verify that the condition holds in each path. Usually, the path tree is infinite, so it is represented as a language automaton. Other stochastic logics include LTL (*Linear Temporal Logic*) [Pnu77] and CTL* [EH86], among many others.

Temporal logics have been extended into both the probabilistic and stochastic domains. The temporal logic PCTL (*Probabilistic CTL*) [Han91] is a variant of CTL defined for DTMCs in which path properties accept/reject paths when the overall probability of the specified condition is above/below a given threshold. Stochastic temporal logics measure path durations instead of path probabilities, and are defined for CTMCs. The first of these logics, and probably the most influential, is CSL (*Continuous Stochastic Logic*) [Azi+00], introduced in the early 2000 and then improved algorithmically in 2003 with the work of [Bai+03]. CSL includes both probabilistic and timed constraints over path executions in CTMCs, so that a specified condition has to be met with a given probability in a specified amount of time t. In [MH06], CSL has been extended to the class of MRP (although with limitations), while in [Hor+11] the logic CSL has been derived in analytical form for the class of *generalized semi-Markov processes*, using the concept of *stochastic state classes*.

3.1 Path probabilities on the measurable space

Measuring paths is different from measuring rewards on set of states/events, because paths in a Markov transition system are intrinsically infinite. The formal structure of the set of paths of a Markov transition system is based on measure theory. A probability space can be constructed from the σ-algebra of the possible paths. This section introduces the basic formalism of σ-algebra and measurable spaces. These definitions are then used to define the measurability of set of paths in discrete-time and continuous-time Markov chains.

Definition 10 (σ-algebra)**.** Given a set H, let $\mathcal{F}_H$ be the σ-algebra of H obtained as some subset of the power set 2^H that satisfies the three properties:

- $\mathcal{F}_H \neq \varnothing$;
- Closed under *complementation*: if $E \in \mathcal{F}_H$ then $(H \setminus E) \in \mathcal{F}_H$;
- Closed under *countable union*: if $E_1, E_2, \ldots \in \mathcal{F}_H$ then $\bigcup_{n \geq 0} E_n \in \mathcal{F}_H$.

The pair $\langle H, \mathcal{F}_H \rangle$ is called a *measurable space*. The set H is interpreted as a collection of possible *outcomes*, or *events*, which can be assigned probabilities. In the context of this chapter, the set H is the set of *infinite paths* that starts from an initial state (or path prefix). Since paths are infinite in length, their probability will (in general) be infinitesimal. Entries in the set $\mathcal{F}_H$ are collection of paths. Probabilities are then assigned to set of paths $E \in \mathcal{F}_H$.

Definition 11 (Probability measure)**.** A function $\Pr : \mathcal{F}_H \to \mathbb{R}_{[0,1]}$ is called a *probability measure* for the σ-algebra $\mathcal{F}_H$ if $\Pr(\mathcal{F}_H) = 1$ and if it is additive over the union of disjoint events $\{E_n \in \mathcal{F}_H\}$, such that: $\sum_{n \geq 0} \Pr(E_n) = \Pr\big(\bigcup_{n \geq 0} E_n\big)$.

A proper *probability measure* is given for paths in a DMTS and in a CMTS in sections 3.2 and 3.3. A measurable space paired with a measure function Pr forms a triple $\langle H, \mathcal{F}_H, \Pr \rangle$ called a *measure space*. Sections 3.2 and 3.3 describe the measurable spaces of DTMC and CTMC paths, along with the formalisms used to express and compute measures on path probabilities.

3.2 Path probabilities in discrete-time

This section deals with the σ-algebra definition for discrete-time Markov chains, based on [Var85]. Let $\mathcal{D} = \langle \mathcal{S}, Act, AP, lab, T, P, \boldsymbol{\alpha}_0 \rangle$ be a DMTS with stochastic matrix $\mathbf{P}$. A *finite path* of length n is a sequence of states:

$$\sigma \;=\; s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{n-1}} s_n, \qquad \sigma \in \mathcal{S}^n \tag{3.1}$$

such that $P(s_{i-1}, a_{i-1} s_i) > 0$ for all $i \in [1, n]$. Let $Paths_n^{\mathcal{D}}$ be the set of all paths of length n with the structure of (3.1). Given a path $\sigma \in Paths_n^{\mathcal{D}}$, let $|\sigma| = n$ be the length of σ, and let $\sigma[i] = s_i$ be the i-th state of the path. Let $Paths_\omega^{\mathcal{D}} = \lim\limits_{n \to \infty} Paths_n^{\mathcal{D}}$ be the set of infinite paths of $\mathcal{D}$. The set H of the measurable space of $\mathcal{D}$ is given by $Paths_\omega^{\mathcal{D}}$.

The *cylinder set* $Cyl(\bar{\sigma})$ of a finite path $\bar{\sigma} \in Paths_n^{\mathcal{D}}$ is defined as:

$$Cyl(\bar{\sigma}) \overset{\text{def}}{=} \{\sigma \in Paths_\omega^{\mathcal{D}} \mid \bar{\sigma} \text{ is a prefix of } \sigma\} \tag{3.2}$$

The cylinder set that stems from $\bar{\sigma}$ is the set of all infinite paths that start with $\bar{\sigma}$ as a prefix. The σ-algebra $\mathcal{F}^{\mathcal{D}}$ of a DMTS is [BK08, def. 10.10] the smallest σ-algebra that contains all the cylinder sets $\mathit{Cyl}(\bar{\sigma})$, where $\bar{\sigma}$ ranges over all path fragments in $\mathcal{D}$.

There is a unique [BK08, p. 758] probability measure $\Pr^{\mathcal{D}}$ on the σ-algebra $\mathcal{F}_H^{\mathcal{D}}$, that gives the probability of any cylinder set as:

$$\Pr^{\mathcal{D}}(\mathit{Cyl}(s_0 \dots s_n)) \;=\; \boldsymbol{\alpha}(s_0) \cdot \prod_{0<i\leq n} P(s_{i-1}, a_{i-1}, s_i) \tag{3.3}$$

with $\boldsymbol{\alpha}(s_0)$ is the initial probability of state s_0, and P is the probability function of $\mathcal{D}$ transitions.

3.2.1 Probabilistic Computation Tree Logic

The probabilistic logic PCTL is a branching-time temporal logic, similar to CTL [EC82] with the addition of probabilistic operators specifically designed for model checking discrete-time Markov chains.

PCTL verifies properties of a given DMTS $\mathcal{D}$. PCTL formulas use state labels but do not use action names of $\mathcal{D}$, which are just ignored. The syntax of PCTL is given by state-formulas and path-formulas, which are interpreted over states and paths of the DMTS. The syntaxes are defined as:

Definition 12 (PCTL syntax)**.** A *state formula* Φ in the PCTL temporal logic is defined by:

$$\Phi \;::=\; p \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \mathcal{P}_{\bowtie\lambda}(\Psi) \mid$$

with Ψ a PCTL *path formula*, defined by:

$$\Psi \;::=\; \mathcal{X}\ \Phi \mid \Phi_1\ \mathcal{U}^{[n_1,n_2)}\ \Phi_2$$

where $p \in \mathit{AP}$, $\bowtie$ is a comparison operator in $\{<, \leq, \geq, >\}$, $\lambda \in \mathbb{R}_{[0,1]}$ is a probability, and $0 \leq n_1 \leq n_2$ is an integer bound.

The operator $\mathcal{P}_{\bowtie\lambda}(\Psi)$ means that the probability of following a path that satisfies the path formula Ψ is $\bowtie \lambda$. PCTL provides two fixed path formulas, namely $\mathcal{X}$ (*neXt*) and $\mathcal{U}$ (*Until*). The path formula $\mathcal{X}\Phi$ means that the next state of the DMTS satisfies Φ, while $\Phi_1\ \mathcal{U}^{[n_1,n_2)}\Phi_2$ means that a Φ_2-state is reached at the discrete step in the range $[n_1, n_2)$ passing only through Φ_1-states.

The boolean value of a PCTL state formula Φ in a state s is given by means of a satisfaction relation, denoted as $s \models \Phi$. The pair $\langle s, \Phi \rangle$ belongs to the relation $\models$ iff Φ is satisfied in s. The set of all states that satisfies a formula Φ is denoted as $\mathit{Sat}(\Phi)$ and obeys the following semantic:

Definition 13 (PCTL state-based semantic)**.** Let $\mathit{Sat}(\Phi) = \{s \in \mathcal{S} \mid s \models \Phi\}$ be the set of states that belongs to the satisfaction relation $\models$, defined recursively as:

$$\begin{array}{ll}
s \models a & \text{iff } a \in \mathit{lab}(s) \\
s \models \neg\Phi & \text{iff } s \not\models \Phi \\
s \models \Phi_1 \wedge \Phi_2 & \text{iff } s \models \Phi_1 \wedge s \models \Phi_2 \\
s \models \mathcal{P}_{\bowtie\lambda}(\Psi) & \text{iff } \mathit{Prob}^{\mathcal{D}}(s, \Psi) \bowtie \lambda
\end{array}$$

where $\mathit{Prob}^{\mathcal{D}}(s, \Psi)$ denotes the probability of all paths in the cylinder set $\mathit{Cyl}(s)$ that satisfies the path formula Ψ:

$$\mathit{Prob}^{\mathcal{D}}(s, \Psi) \overset{\text{def}}{=} \Pr\{\sigma \in \mathit{Cyl}(s) \mid \sigma \models \Psi\}$$

The measurability of the set $\{\sigma \in \mathit{Cyl}(s) \mid \sigma \models \Psi\}$ comes from the definition of the σ-algebra $\mathcal{F}_H^{\mathcal{D}}$ of $\mathcal{D}$. The semantic of state-based PCTL formulas of definition 13 requires the semantic of the the two path-based PCTL formulas.

Definition 14 (PCTL semantic of path operators)**.** Let $\{\sigma \in \mathit{Cyl}(s) \mid \sigma \models \Psi\}$ be the set of paths that satisfy the path-formula Ψ, defined as:

$$\begin{array}{ll} \sigma \models \mathcal{X}\ \Phi & \text{iff } |\sigma| \geq 1 \text{ and } \sigma[1] \models \Phi \\ \sigma \models \Phi_1\ \mathcal{U}^{[n_1,n_2)}\ \Phi_2 & \text{iff } \exists i \in [n_1, n_2) : \sigma[i] \models \Phi_2 \text{ and } \forall j \in [0, i) : \sigma[j] \models \Phi_1 \end{array}$$

The until operator with bound $[0, \infty)$ is called *unbounded* until, otherwise it is called *step-bounded* until.

Many temporal operators of CTL can be derived from the basic operators of PCTL. The temporal operator $\diamond$ that means "eventually/in the future" (and its *step-bounded* variant $\diamond^{[n_1,n_2)}$) are derived as:

$$\diamond\ \Phi = \text{true}\ \mathcal{U}\ \Phi, \qquad \diamond^{[n_1,n_2)}\ \Phi = \text{true}\ \mathcal{U}^{[n_1,n_2)}\ \Phi$$

The *step-bounded* operator "always" $\Box^{[n_1,n_2)}\ \Phi$ is derived as:

$$\mathcal{P}_{\geq\lambda}(\Box^{[n_1,n_2)}\ \Phi) = \mathcal{P}_{\leq 1-\lambda}(\diamond^{[n_1,n_2)}\ \neg\Phi)$$

which requires to invert the probability bound of $\mathcal{P}$. The *unbounded* operator $\Box\ \Phi$ is derived analogously.

3.3 Path probabilities in continuous-time

This section deals with the σ-algebra definition for continuous-time Markov chains, as in [Bai+03, sec. 2.3]. Let $\mathcal{M} = \langle \mathcal{S}, \mathit{Act}, \mathit{AP}, \mathit{lab}, T, P, E, \boldsymbol{\alpha}_0 \rangle$ be a CMTS with infinitesimal generator matrix $\mathbf{Q}$. A *finite timed path* of length n is a sequence of states:

$$\sigma\ :\ s_0 \xrightarrow{a_0,t_0} s_1 \xrightarrow{a_1,t_1} \ldots \xrightarrow{a_{n-1},t_{n-1}} s_n \tag{3.4}$$

with $\sigma \in (\mathcal{S} \times \mathit{Act} \times \mathbb{R}_{>0})^{n-1} \times (\mathcal{S} \times \{\infty\})$, such that $P(s_{i-1}, a_{i-1}, s_i) > 0$ and $t_{i-1} > 0$ for all $i \in [1, n]$. Let $\mathit{Paths}_n^{\mathcal{M}}$ be the set of all timed paths of length n with the structure of (3.4). Given a path $\sigma \in \mathit{Paths}_n^{\mathcal{M}}$, with $|\sigma|$, $\sigma[i]$, $\sigma@t$ and $\tau(\sigma, k)$ we denote: the length n of σ, the i-th state of the path, the state of the path at time t, and the total time $\sum_{i=0}^{k-1} t_i$ up to step k, respectively. Let $\mathit{Paths}_\omega^{\mathcal{M}} = \lim\limits_{n\to\infty} \mathit{Paths}_n^{\mathcal{M}}$ be the set of infinite paths of $\mathcal{M}$. The set H of the measurable space of $\mathcal{M}$ is given by $\mathit{Paths}_\omega^{\mathcal{M}}$.

The definition of *cylinder set* $\mathit{Cyl}(\bar{\sigma})$ of a finite timed path $\bar{\sigma} \in \mathit{Paths}_n^{\mathcal{M}}$ is analogous to that of 3.2, and is:

$$\mathit{Cyl}(\bar{\sigma}) \overset{\text{def}}{=} \{\sigma \in \mathit{Paths}_\omega^{\mathcal{M}} \mid \bar{\sigma} \text{ is a prefix of } \sigma\} \tag{3.5}$$

The σ-algebra $\mathcal{F}^{\mathcal{M}}$ of a CMTS is given by the smallest σ-algebra that contains all the cylinder sets $\mathit{Cyl}(\bar{\sigma})$ of $\mathcal{M}$.

The unique probability measure $\Pr^{\mathcal{M}}$ on the timed σ-algebra $\mathcal{F}_H^{\mathcal{M}}$ is defined as follows: let $\mathit{Cyl}(s_0, I_0, \ldots, I_{n-1}, s_n)$ be a cylinder set of a finite path of length n, with $I_0, \ldots, I_{n-1}$ nonempty intervals in $\mathbb{R}_{\geq 0}$. The probability is defined recursively as:

$$\begin{aligned}\Pr^{\mathcal{M}}(\mathit{Cyl}(s_0, I_0, \ldots, I_{n-1}, s_n)) \; = \\ \Pr^{\mathcal{M}}(\mathit{Cyl}(s_0, I_0, \ldots, I_{n-2}, s_{n-1})) \cdot \int_{I_{n-1}} \mathbf{Q}(s_{n-1}, s_n) \cdot \mathrm{e}^{-\mathbf{Q}(s_n)\tau} \, \mathrm{d}\tau\end{aligned} \tag{3.6}$$

with $\Pr^{\mathcal{M}}\big(\mathit{Cyl}(s_0)\big) = \boldsymbol{\alpha}(s_0)$ the initial probability of state s_0. Following the definition of [Bai+03, prop. 1], any infinite timed path $\sigma \in \mathit{Paths}_{\omega}^{\mathcal{M}}$ with finite durations $\tau(\sigma)$ is considered “unrealistic”, since the probability of such path σ converges to zero. Therefore, infinite paths must have infinite duration.

3.3.1 Continuous Stochastic Logic

The stochastic logic CSL is a branching-time temporal logic similar to CTL [EC82] initially developed in [Azi+00]. The formal forward solution has been given in [Bai+03], while the backward solution is in [Kat+01]. The relation between these two methods is given in [AD12b]. CSL has many points in common with PCTL; the main difference is that it is defined in continuous-time for CMTSs, so that CSL path formulas specify *time bounds* instead of *step bounds*.

CSL verifies properties on a given CMTS $\mathcal{M}$. The definition of CSL uses state labels but not action names, which are just ignored. CSL provides both state-formulas and path-formulas, interpreted over states and paths of the CMTS. Like PCTL, CSL has two probabilistic operators that check the probability of the specified formulas. Since CSL reads timed paths, the path formulas are extended with time constraints.

Definition 15 (CSL syntax)**.** A *state formula* Φ in the CSL temporal logic is defined by:

$$\Phi \;::=\; p \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \mathcal{S}_{\bowtie\lambda}(\Phi) \mid \mathcal{P}_{\bowtie\lambda}(\Psi)$$

with Ψ a *path formula* defined by:

$$\Psi \;::=\; \mathcal{X}^{[\alpha,\beta]}\;\Phi \mid \Phi_1\;\mathcal{U}^{[\alpha,\beta]}\;\Phi_2$$

where $p \in AP$, $\bowtie$ is a comparison operator in $\{<, \leq, \geq, >\}$ and $\lambda \in \mathbb{R}_{[0,1]}$ is a probability.

Like PCTL, the operator $\mathcal{P}_{\bowtie\lambda}(\Psi)$ means that the probability of following a path that satisfies the path formula Ψ is $\bowtie \lambda$. Additionaly, CSL offers an operator for steady-state probabilities, introduced in [Bai+03]. The operator $\mathcal{S}_{\bowtie\lambda}(\Phi)$ means that the sub-expression Φ holds in steady-state with probability $\bowtie \lambda$.

CSL provides two fixed path formulas, namely $\mathcal{X}$ (*neXt*) and $\mathcal{U}$ (*Until*). The path formula $\mathcal{X}^{[\alpha,\beta]}\;\Phi$ means that the next state is reached at time $[\alpha, \beta]$ and satisfies Φ, while $\Phi_1 \, \mathcal{U}^{[\alpha,\beta]} \, \Phi_2$ means that a Φ_2-state is reached in the time interval $[\alpha, \beta]$ passing only through Φ_1-states.

The boolean value of a CSL state formula Φ in a state s is given by means of a satisfaction relation, denoted as $s \models \Phi$. The pair $\langle s, \Phi \rangle$ belongs to the relation $\models$ iff Φ is satisfied in s. The set of all states that satisfies a formula Φ is denoted as $\mathit{Sat}(\Phi)$ and obeys the following semantic:

Definition 16 (CSL state-based semantic)**.** Let $\mathit{Sat}(\Phi) = \{s \in \mathcal{S} \mid s \models \Phi\}$ be the set of states that belongs to the satisfaction relation $\models$, defined recursively as:

$$\begin{array}{ll}
s \models a & \text{iff } a \in \mathit{lab}(s) \\
s \models \neg\Phi & \text{iff } s \not\models \Phi \\
s \models \Phi_1 \wedge \Phi_2 & \text{iff } s \models \Phi_1 \wedge s \models \Phi_2 \\
s \models \mathcal{S}_{\bowtie\lambda}(\Phi) & \text{iff } \boldsymbol{\pi}^{\mathcal{M}}(\mathbf{i}_s) \cdot \mathbf{i}_\Phi \bowtie \lambda \\
s \models \mathcal{P}_{\bowtie\lambda}(\Psi) & \text{iff } \mathit{Prob}^{\mathcal{M}}(s, \Psi) \bowtie \lambda
\end{array}$$

where $\mathbf{i}_\Phi$ is an indicator vector with value 1 in each state of $\mathit{Sat}(\Phi)$, and $\mathit{Prob}^{\mathcal{M}}(s, \Psi)$ denotes the probability of all paths in $\mathit{Cyl}(s)$ that satisfies the path formula Ψ:

$$\mathit{Prob}^{\mathcal{M}}(s, \Psi) \overset{\text{def}}{=} \Pr\{\sigma \in \mathit{Cyl}(s) \mid \sigma \models \Psi\}$$

The measurability of the set $\{\sigma \in \mathit{Cyl}(s) \mid \sigma \models \Psi\}$ comes from the definition of the σ-algebra $\mathcal{F}_H^{\mathcal{M}}$ of $\mathcal{M}$. The semantic of state-based CSL formulas of definition 16 requires the semantic of the path-based CSL formulas.

Definition 17 (CSL path-based semantic)**.** Let $\{\sigma \in \mathit{Cyl}(s) \mid \sigma \models \Psi\}$ be the set of paths that satisfy the path-formula Ψ, defined as:

$$\begin{array}{ll}
\sigma \models \mathcal{X}^{[\alpha,\beta]} \, \Phi & \text{iff } |\sigma| \geq 1 \text{ and } \sigma[1] \models \Phi \text{ and } \tau(\sigma, 0) \in [\alpha, \beta] \\
\sigma \models \Phi_1 \, \mathcal{U}^{[\alpha,\beta]} \, \Phi_2 & \text{iff } \exists t \in [\alpha, \beta] : \sigma@t \models \Phi_2 \text{ and } \forall t' \in [0, t) : \sigma@t' \models \Phi_1
\end{array}$$

The relation between CSL and CTL is provided in [Bai+03, sec. 3.2]. The untimed version of $\mathcal{X}$ and $\mathcal{U}$ are simply obtained using the time interval $[0, \infty)$. The temporal operator $\Diamond$, which means “eventually/in the future” (or the timed variant $\Diamond^I$) is formulated as: $\mathcal{P}_{\bowtie\lambda}(\Diamond^I \, \Phi) = \mathcal{P}_{\bowtie\lambda}(\text{true} \, \mathcal{U}^I \, \Phi)$. Analogously, the $\Box$ operator of “always” and its timed variant $\Box^I$ can be expressed as: $\mathcal{P}_{\geq\lambda}(\Box^I \, \Phi) = \mathcal{P}_{\leq 1-\lambda}(\Diamond^I \, \neg\Phi)$.

3.3.2 Computation of CSL formulas

The calculation of a CSL formula Φ for a state s is done as a recursive bottom-up evaluation of the CSL sub-formulas, for which the $\mathit{Sat}(\Phi)$ set of states has to be computed. Non probabilistic operators (the first three of Def. 16) are computed by applying their logical formula, in a way that is analogous to CTL. Terminal leaves of the evaluation tree are made by atomic propositions, which are evaluated using the labeling function lab of the CMTS. The $\mathcal{S}_{\bowtie\lambda}(\Phi)$ and $\mathcal{P}_{\bowtie\lambda}(\Psi)$ operators instead require a numerical computation that can be derived from the formulas of section 2.3.

Stationary measures with $\mathcal{S}_{\bowtie\lambda}(\Phi)$

The semantic of the $s \models \mathcal{S}_{\bowtie\lambda}(\Phi)$ formula of Def. 16 says that the formula is true in state s if the long run probability $\boldsymbol{\pi}^{\mathcal{M}}(\mathbf{i}_s)$ starting from state s has a mean probability of satisfying Φ that is $\bowtie \lambda$. By definition, such a probability is:

$$s \models \mathcal{S}_{\bowtie\lambda}(\Phi) \quad \Leftrightarrow \quad \left(\boldsymbol{\pi}^{\mathcal{M}}(\mathbf{i}_s) \cdot \mathbf{i}_\Phi\right) \bowtie \lambda \tag{3.7}$$

with $\mathbf{i}_\Phi$ the indicator vector for $Sat(\Phi)$ states. Another solution method, based on the BSCC (*bottom strongly connected components*) isolation, is given in [Bai+03, sec. 3.4]. The key observation is that the computation depends from s when the CMTS $\mathcal{M}$ is reducible. Therefore, (3.7) may be computed by separating the computation of the steady-state probabilities of each BSCC from the computation of the probability of reaching the BSCCs from s.

A more interesting approach is the use of backward probabilities. Equation (2.26) can be applied to change the solution formula (3.7) into:

$$\boldsymbol{\pi}^{\mathcal{M}}(\mathbf{i}_s) \cdot \mathbf{i}_\Phi \quad \Leftrightarrow \quad \boldsymbol{\xi}^{\mathcal{M}}(\mathbf{i}_\Phi) \cdot \mathbf{i}_s \tag{3.8}$$

The backward formulation has the advantage that $\boldsymbol{\xi}^{\mathcal{M}}(\mathbf{i}_\Phi)$ only depends on the $Sat(\Phi)$ set, and not from the initial state s. Therefore, a single computation of $\boldsymbol{\xi}^{\mathcal{M}}(\mathbf{i}_\Phi)$ allows to compute easily the formula for each initial state, independently from the number of BSCCs.

Probability measures with $\mathcal{P}_{\bowtie\lambda}(\Psi)$

A path-based measure for state s is expressed with the CSL formula $s \models \mathcal{P}_{\bowtie\lambda}(\Psi)$. The formula is true if the probability of the set of paths $Cyl(s)$ that satisfies Ψ is $\bowtie \lambda$, indicated as $Prob^{\mathcal{M}}(s, \Psi)$. In [Bai+03, p. 8] a set of Volterra integral equations are derived for the computation of path-based measures, which are then reduced to transient/stationary measures on CTMCs. In this section, we summarize the latter.

Some formulas require the concept of *modified CMTS*, obtained through the *Sat*-based filtering operator $\mathcal{M}[\Phi]$ for a certain CSL formula Φ.

Definition 18 (Modified CMTS)**.** Given a labeled CMTS $\mathcal{M}$ and a CSL formula Φ, the labeled CMTS $\mathcal{M}[\Phi]$ is obtained by making absorbing all the states that satisfy Φ in $\mathcal{M}$.

Computing the timed *next* operator: $\Psi = \mathcal{X}^{[\alpha,\beta]}\ \Phi$

The next operator measures the probability that the next state-transition of the CMTS $\mathcal{M}$ happens in the time interval $[\alpha, \beta]$ and reaches a $Sat(\Phi)$-state. Therefore:

$$\begin{aligned} Prob^{\mathcal{M}}(s, \mathcal{X}^{[\alpha,\beta]}\ \Phi) \ &= \int_\alpha^\beta \mathrm{e}^{x\mathbf{Q}(s)}\,\mathrm{d}x \cdot \sum_{s' \models \Phi} \mathbf{P}(s, s') \ = \\ &= \left(\mathrm{e}^{\beta\mathbf{Q}(s)} - \mathrm{e}^{\alpha\mathbf{Q}(s)}\right) \cdot \sum_{s' \models \Phi} \mathbf{P}(s, s') \end{aligned} \tag{3.9}$$

where $\mathbf{P} = \mathsf{diag}^{-1}(\mathbf{Q})\mathbf{Q}$ is the transition probability matrix of $\mathcal{M}$. Recall that $\mathbf{Q}(s)$ is the total exit rate from s, with negative sign.

The intuition of (3.9) is that CSL allows the process to jump to a $\mathit{Sat}(\Phi)$ state (the probability collected in the summation of $\mathbf{P}(s, s'))$ conditioned by the probability that the Poisson process of rate $-\mathbf{Q}(s)$ makes a jump in the interval $[\alpha, \beta]$.

Computing the timed *until* operator: $\Psi = \Phi_1 \; \mathcal{U}^{[\alpha,\beta]} \; \Phi_2$

The until operator observes paths of length more than 1. An accepted path reaches a $\mathit{Sat}(\Phi_2)$ state in the time interval $[\alpha, \beta]$, having only visited $\mathit{Sat}(\Phi_1)$ states before. According to [AD12b], it is possible to split the problem into three cases, according to the bounds of the interval $[\alpha, \beta]$. The interval can be $[0, \beta]$, $[\beta, \beta]$, or $[\alpha, \beta]$, with $0 < \alpha < \beta$, leading to:

$$\begin{aligned}
\mathit{Prob}^{\mathcal{M}}(s, \Phi_1 \; \mathcal{U}^{[0,\beta]} \; \Phi_2) &= \mathbf{i}_{\Phi_2} \cdot \boldsymbol{\pi}^{\mathcal{M}[\neg\Phi_1 \vee \Phi_2]}\big(\mathbf{i}_s, \beta\big) \\
\mathit{Prob}^{\mathcal{M}}(s, \Phi_1 \; \mathcal{U}^{[\beta,\beta]} \; \Phi_2) &= \mathbf{i}_{\Phi_1 \wedge \Phi_2} \cdot \boldsymbol{\pi}^{\mathcal{M}[\neg\Phi_1]}\big(\mathbf{i}_s, \beta\big) \\
\mathit{Prob}^{\mathcal{M}}(s, \Phi_1 \; \mathcal{U}^{[\alpha,\beta]} \; \Phi_2) &= \\
&= \mathbf{i}_{\Phi_2} \cdot \boldsymbol{\pi}^{\mathcal{M}[\neg\Phi_1 \vee \Phi_2]}\Big(\mathbf{I}^{\Phi_1} \cdot \boldsymbol{\pi}^{\mathcal{M}[\neg\Phi_1]}\big(\mathbf{i}_s, \alpha\big), \; \beta - \alpha\Big)
\end{aligned} \tag{3.10}$$

Forward measures are computed using modified CMTSs, as in Def. 18. When $\beta = \infty$, the computed measures is a stationary solution of the modified CMTS. In the other cases, the solution is computed with a forward transient analysis. The vectors $\mathbf{i}_{\Phi_2}$, $\mathbf{i}_{\Phi_1 \wedge \Phi_2}$ and $\mathbf{i}_s$ are indicator vectors for the states of $\mathit{Sat}(\Phi_2)$, $\mathit{Sat}(\Phi_1 \wedge \Phi_2)$ and $\{s\}$, respectively, and 0 in every other state. The $\mathbf{I}^{\Phi_1}$ is the filtered identity matrix (as in Def. 7) where rows corresponding to states that do not satisfy Φ_1 are set to zero.

The intuition behind is simple. The probability $\mathit{Prob}^{\mathcal{M}}(s, \Phi_1 \; \mathcal{U}^{[0,\beta]} \; \Phi_2)$ is computed as the probability of being, at time t, in any Φ_2-state on the modified CMTS $\mathcal{M}[\neg\Phi_1 \vee \Phi_2]$. In the modified chain the Φ_2 states are absorbing, so even if a Φ_2 state is reached before t, the chain will still be in that state at time t, the time horizon for the transient probability computation. Similarly, if a $\neg\Phi_1$-state s' is encountered before a Φ_2-state is reached, the modified chain stays trapped in s', and that path will not be counted (unless s' is also a Φ_2-state). Note the use of the inner product with $\mathbf{i}_{\Phi_2}$ to sum over all possible Φ_2 states, and that the computation of the transient probability assumes the modified chain is in state s at time 0 (initial vector $\mathbf{i}_s$). The more complicated case of $[\alpha, \beta]$ requires the path to stay in Φ_1-states during the time interval $[0, \alpha]$, and then to behave as a path that satisfies $\Phi_1 \; \mathcal{U}^{[0,(\beta-\alpha)]} \; \Phi_2$. This requires the transient solution of two modified CMTSs: at time α, assuming we start in s at time 0, for the chain $\boldsymbol{\pi}^{\mathcal{M}[\neg\Phi_1]}$ and at time $\beta - \alpha$, assuming we start at time 0 with a probability vector which is the result of the previous computation, for the chain $\boldsymbol{\pi}^{\mathcal{M}[\neg\Phi_1 \vee \Phi_2]}$. Note that the result of the first computation is filtered out using the $\mathbf{I}^{\Phi_1}$ vector, to put to zero the probability of all states which are not Φ_1-states: as a consequence the second transient analysis starts from an initial vector that does not necessarily sum up to one.

The work in [Kat+01] shows that the three equations (3.10) can be rewritten using the forward-backward relation (2.26), leading to:

$$\begin{aligned}
\underline{Prob}^{\mathcal{M}}(\Phi_1\ \mathcal{U}^{[0,\beta]}\ \Phi_2) &= \boldsymbol{\xi}^{\mathcal{M}[\neg\Phi_1\vee\Phi_2]}\big(\mathbf{i}_{\Phi_2},\beta\big) \\
\underline{Prob}^{\mathcal{M}}(\Phi_1\ \mathcal{U}^{[\beta,\beta]}\ \Phi_2) &= \boldsymbol{\xi}^{\mathcal{M}[\neg\Phi_1]}\big(\mathbf{i}_{\Phi_1\wedge\Phi_2},\beta\big) \\
\underline{Prob}^{\mathcal{M}}(\Phi_1\ \mathcal{U}^{[\alpha,\beta]}\ \Phi_2) &= \boldsymbol{\xi}^{\mathcal{M}[\neg\Phi_1]}\Big(\mathbf{I}^{\Phi_1}\cdot\boldsymbol{\xi}^{\mathcal{M}[\neg\Phi_1\vee\Phi_2]}\big(\mathbf{i}_{\Phi_2},\beta-\alpha\big),\ \alpha\Big)
\end{aligned} \tag{3.11}$$

with $\underline{Prob}^{\mathcal{M}}(\Psi)$ a vector of probabilities such that $\underline{Prob}^{\mathcal{M}}(\Psi)\cdot\mathbf{i}_s = Prob^{\mathcal{M}}(s,\Psi)$, for every state $s\in\mathcal{S}$. The third formula in (3.11) is derived in two steps as:

$$\begin{aligned}
Prob^{\mathcal{M}}(s,\Phi_1\ \mathcal{U}^{[\alpha,\beta]}\ \Phi_2) &= \mathbf{i}_{\Phi_2}\cdot\boldsymbol{\pi}^{\mathcal{M}[\neg\Phi_1\vee\Phi_2]}\Big(\mathbf{I}^{\Phi_1}\cdot\boldsymbol{\pi}^{\mathcal{M}[\neg\Phi_1]}\big(\mathbf{i}_s,\alpha\big),\ \beta-\alpha\Big) = \\
&= \boldsymbol{\pi}^{\mathcal{M}[\neg\Phi_1]}\big(\mathbf{i}_s,\alpha\big)\cdot\Big(\mathbf{I}^{\Phi_1}\cdot\boldsymbol{\xi}^{\mathcal{M}[\neg\Phi_1\vee\Phi_2]}\big(\mathbf{i}_{\Phi_2},\beta-\alpha\big)\Big) = \\
&= \boldsymbol{\xi}^{\mathcal{M}[\neg\Phi_1]}\Big(\mathbf{I}^{\Phi_1}\cdot\boldsymbol{\xi}^{\mathcal{M}[\neg\Phi_1\vee\Phi_2]}\big(\mathbf{i}_{\Phi_2},\beta-\alpha\big),\ \alpha\Big)\cdot\mathbf{i}_s = \\
&= \underline{Prob}^{\mathcal{M}}(\Phi_1\ \mathcal{U}^{[\alpha,\beta]}\ \Phi_2)\cdot\mathbf{i}_s
\end{aligned}$$

which proves the relation. With (3.11) a model checker can compute the until operator for every initial state s with one (or two for the case $[\alpha,\beta]$) backward computations.

The original definition of CSL [Azi+00] allows also cascades of multiple timed-until formulas, which are not easily treated. An innovative technique for dealing with this special case has been proposed in [Zha+12], by introducing of the concept of CTMC stratification. Further details can be found in that paper. However, complex path specification requires a richer language than the one proposed in CSL. Path specification done with timed automata is treated in the following sections.

3.4 Specifying path properties with automata

PCTL and CSL describe path properties with a fixed set of path operators, that provide the set of constraints for the measured cylinder sets. A more flexible approach is to allow for path operators to be specified by a *language* of accepted paths. This section considers two different approaches: the stochastic logic CSL$^{\text{TA}}$ [DHS09], which is a superset of CSL with path properties specified with automata, and *probe automata*[Amp+11], which is a specialized language for specifying path-based properties. In the literature, however, there are many other proposals for path-based measures: a survey of a number of approaches can be found in [DHS09, sec. 4] and in [Kun06, sec. II]. Automata-based extensions of PCTL will not be covered.

3.4.1 CSL with Timed Automata

CSL path properties can be specified with the two predefined timed operators, $\mathcal{X}^I$ and $\mathcal{U}^I$. More complex paths with additional constraints or with multiple time intervals cannot be described. The temporal stochastic logic CSL$^{\text{TA}}$ (*CSL with Timed Automata*) has been introduced in [DHS09] as an extension to CSL in which path properties are specified with a *Deterministic Timed Automaton* (DTA). In this way, the set of accepted paths can be described arbitrarily. Unlike CSL, the logic CSL$^{\text{TA}}$ reads both state labels and action names of the CMTS.

A DTA is an automaton that reads the language of CMTS paths. DTAs are deterministic in the sense that each CMTS path σ generates a unique DTA computation (or *run*) ς. Each DTA is equipped with a *clock*, named x, that runs constantly and whose value increases linearly over time. A *clock valuation* $\bar{x}$ is the actual value of the clock variable. In [Che+11a] DTAs with multiple clocks are defined, which will not be covered in this text. A DTA has a set of *locations* and a set of *edges*. Edges describe the transition relation and are labeled with a *clock constraint*.

The language CC of clock constraints is given by:

$$CC \; ::= \; \underbrace{x = c_1}_{Boundary} \mid \underbrace{c_1 < x < c_2}_{Inner} \qquad c_1, c_2 \in \mathbb{N},\, c_1 < c_2$$

An edge with a constraint in the form $x = c_1$ is a *Boundary* edge, while an edge with a constraint $c_1 < x < c_2$ is an *Inner* edge. The value of c_2 is allowed to be infinite, to represent unbounded *Inner* constraints. Given a constraint $\gamma \in CC$, let $\bar{x} \models \gamma$ denote that γ is satisfied by substituting x with $\bar{x}$.

Definition 19 (DTA)**.** A *deterministic timed automaton* $\mathcal{A}$ is a tuple:

$$\mathcal{A} \;=\; \langle \Sigma, Act, L, L_0, L_F, \Lambda, \rightarrow \rangle$$

where:

- Σ is a finite set of *state propositions*;
- Act is a finite set of *action names*;
- L is a finite set of *locations*;
- $L_0 \subseteq L$ is the set of *initial locations*;
- $L_F \subseteq L$ is the set of *final locations*;
- $\Lambda : L \rightarrow \mathcal{B}(\Sigma)$ is a *location labeling function*, with $\mathcal{B}(\Sigma)$ the language of boolean expressions over Σ;
- $\rightarrow \subseteq (L \setminus L_F) \times \big((Inner \times 2^{Act}) \uplus (Boundary \times \{\sharp\})\big) \times \{\varnothing, x\} \times L$ is a finite set of *edges*, where $l \xrightarrow{\gamma,A,r} l'$ denotes the edge $\langle l, \gamma, A, r, l' \rangle \in \rightarrow$;

that fulfills the following restrictions:

- **Initial determinism**: $\forall l, l' \in L_0, \Lambda(l) \wedge \Lambda(l') \Leftrightarrow$ *false*;
- **Determinism on edges**: $\forall l, l', l'' \in L$, if $l'' \xrightarrow{\gamma,A,r} l \wedge l'' \xrightarrow{\gamma',A',r'} l'$ then either $\Lambda(l) \wedge \Lambda(l') \Leftrightarrow$ *false* or $A \cap A' = \varnothing$ or $\gamma \wedge \gamma' \Leftrightarrow$ *false*;
- **No loops of *Boundary* edges**: there exists no sequence of *Boundary* edges: $l_0 \xrightarrow{\gamma_0,\sharp,r_0} l_1 \xrightarrow{\gamma_1,\sharp,r_1} \ldots \xrightarrow{\gamma_{n-1},\sharp,r_{n-1}} l_n$, with $l_n = l_0$.

Given an edge $e = l \xrightarrow{\gamma,A,r} l'$, let $\mathsf{source}(e) = l$, $\mathsf{guard}(e) = \gamma$, $\mathsf{action}(e) = A$, $\mathsf{reset}(e) = r$ and $\mathsf{target}(e) = l'$. The *reset set* $\mathsf{reset}(e)$ indicates whether the clock x is reset to zero when the DTA follows the edge e. Let the notation $\bar{x}[x := 0]$ indicate that the valuation $\bar{x}$ is equal to 0, and let $\bar{x}[\varnothing := 0]$ indicate that it is equal to $\bar{x}$. Assume also that the set Σ of state propositions is equivalent to the set AP of atomic proposition of the verified CMTS.

Boundary edges have priority over *Inner* edges, and are *urgent*. Urgency specifies that if a *Boundary* edge is enabled, it must be taken immediately.

A *configuration* of $\mathcal{A}$ is a pair $\langle l, \bar{x} \rangle$ with $l \in L$ and $\bar{x}$ a clock valuation. A configuration describes the current state of $\mathcal{A}$.

A *step* of $\mathcal{A}$ from a configuration $\langle l, \bar{x} \rangle$ is $\langle l, \bar{x} \rangle \xrightarrow{\tau, e} \langle l', \bar{x}' \rangle$ with $\tau \in \mathbb{R}_{\geq 0}$ an elapsed time, $e \in \rightarrow$ an edge, and such that: $l' = \mathsf{target}(e)$ and $\bar{x} + \tau \models \mathsf{guard}(e)$ and $\bar{x}' = (\bar{x} + \tau)[\mathsf{reset}(e) := 0]$. A *step* is a single transition of configuration in $\mathcal{A}$ after an elapsed time of τ instants, after which an edge e is taken. Note that τ could be zero. Boundary edges have a concept of *urgency*: as soon as the guard $x = c$ is satisfied, the edge is enabled and a step with that edge is immediately taken. In addition, boundary edges have priority, i.e. a step with an *Inner* edge is not allowed if there is a *Boundary* edge that can be taken instead.

Definition 20 (Run in a DTA)**.** A finite run ς of length n of a DTA $\mathcal{A}$ starting from an initial configuration $\langle l_0, \bar{x}_0 \rangle$ is a sequence of steps:

$$\varsigma \; : \; \langle l_0, \bar{x}_0 \rangle \xrightarrow{\tau_0, e_0} \langle l_1, \bar{x}_1 \rangle \xrightarrow{\tau_1, e_1} \ldots \xrightarrow{\tau_{n-1}, e_{n-1}} \langle l_n, \bar{x}_n \rangle$$

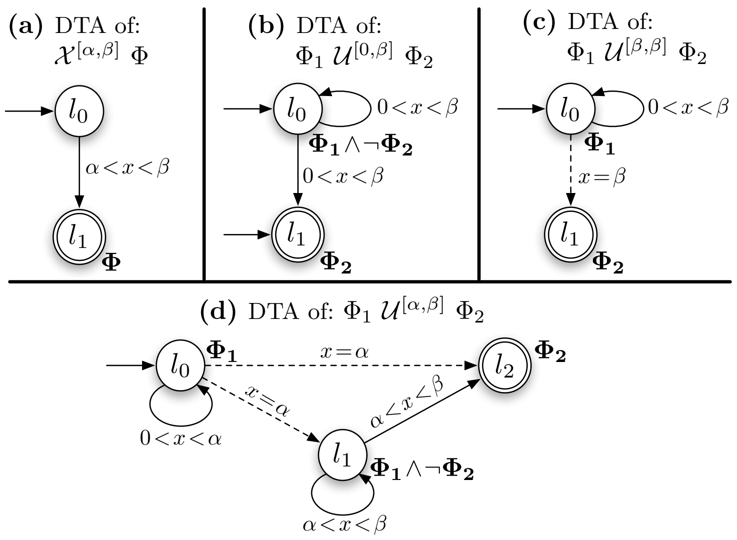
where, for all $0 \leq i < n$, $\tau_i \in \mathbb{R}_{\geq 0}$ and $e_i \in \rightarrow$.

Let $|\varsigma|$ be the length of ς. Let $\mathit{Runs}_n^{\mathcal{A}}$ be the set of all runs ς of length n that are valid, i.e. that satisfies the condition of Def. 19, and where *Boundary* edges are urgent and have priority over *Inner* edges. Let $\mathit{Runs}_\omega^{\mathcal{A}} = \bigcup_{n \geq 0} \mathit{Runs}_n^{\mathcal{A}}$ be the set of all runs of $\mathcal{A}$.

Examples of DTAs of the CSL$^{\mathbf{TA}}$ logic

Figure 3.1 shows the DTAs corresponding to the CSL *next* path formula and the three cases of the CSL *until* path formula.

Figure 3.1 DTAs for the CSL path formulas *neXt* and *Until.*



The DTA in Figure 3.1(a) is the DTA corresponding to the CSL path formula $\mathcal{X}^{[\alpha,\beta]}\ \Phi$. The initial location is l_0, and the CMTS path is accepted if the first transition satisfies the location constraint Φ of l_1 and happens in the time interval $[\alpha, \beta]$. Figure 3.1(b) is the DTA that accepts paths that satisfy $\Phi_1\ \mathcal{U}^{[0,\beta]}\ \Phi_2$. This DTA accepts all CMTS paths that reach a $\mathit{Sat}(\Phi_2)$ state before time β, while visiting only $\mathit{Sat}(\Phi_1)$ states, or that start at time 0 already in a state that satisfies Φ_2. There are therefore two initial locations, one of which is final, since any path that starts in a Φ_2 state should be accepted. If a paths starts instead in a $\mathit{Sat}(\Phi_1 \wedge \neg\Phi_2)$ state and it goes through another Φ_1-state, then the transition is read by the self loop over l_0, if it happens before time β. If the CMTS transition reaches a $\mathit{Sat}(\neg\Phi_1 \wedge \neg\Phi_2)$ state, then the path is rejected since neither the condition over l_0 nor that over l_1 are satisfied. If the transition goes to a $\mathit{Sat}(\Phi_2)$ state, before the automata clock x reaches β, then the automata moves to location l_1 and the path is accepted.

Figures 3.1(c) and (d) depict the other two cases for the Until interval ($[\beta, \beta]$ and $[\alpha, \beta]$). Paths are recognized in a similar manner, but it is worth noting the use of boundary edges with guard $x = t$ to mark the lower limit of the interval: it is indeed only after the time barrier has been reached that a CMTS transition from a $\mathit{Sat}(\Phi_1)$ state to a $\mathit{Sat}(\Phi_2)$ state can lead to the acceptance of the path (if it happens before β).

The example also shows that the CSL semantic of the Until requires multiple CSL$^{\mathrm{TA}}$ DTAs to express all the cases for the time intervals.

Acceptance of CMTS paths

Definition 20 provides the basis for accepting or rejecting CMTS paths. Intuitively, a DTA run ς has to follow a CMTS path $\sigma \in \mathit{Paths}^{\mathcal{M}}_{\omega}$. *Inner* edges in the run ς are triggered by the transitions in σ, while *Boundary* edges are triggered by the elapse of time of the clock x. If a transition in σ cannot be matched by any *Inner* edge, the path σ is rejected. When the run ς reaches a configuration $\langle l, \bar{x}\rangle$ with l a final location, the path σ is accepted and any further CMTS transition is ignored. Therefore, a run ς that accepts σ up to the n-th transition, accepts every path in $\mathit{Cyl}(\sigma[0]\dots\sigma[n])$, which ensures the measurability of accepted paths (see [Che+11a, th. 1] for a formal demonstration of measurability).

The initial configuration of ς is $\langle l, 0\rangle$, where $l \in L_0$ and such that $\sigma[0]$ satisfies the condition $\Lambda(l)$. The condition of initial determinism of $\mathcal{A}$ ensures that there exists at most one initial location l for a given initial state $\omega[0]$. If no such location l exists, then the entire path ω is rejected.

Definition 21 (Path acceptance)**.** Given a DTA $\mathcal{A}$ and a CMTS path $\sigma \in \mathit{Paths}^{\mathcal{M}}_{\omega}$ up to length n, we say that $\sigma \models \mathcal{A}$ if there exists a run $\varsigma \in \mathit{Runs}^{\mathcal{A}}_{\omega}$ of length m and a function $\kappa : \{0\dots m\} \to \{0\dots n\}$ that maps indices of ς into indices of σ and such that:

- *Initial*: $l_0 \in L_0$, $\bar{x}_0 = 0$, $\kappa(0) = 0$;
- *Final*: $l_m \in L_F$ and $\forall 0 < i < m : l_i \notin L_F$;
- *State propositions*: $\forall 0 \le i \le m : s_{\kappa(i)} \models \Lambda(l_i)$;
- *Index correspondence:* $\forall 0 \le i < m$ if e_i is *Inner* then $\kappa(i+1) = \kappa(i) + 1$ and $a_{\kappa(i)} \in \mathsf{action}(e_i)$, else $\kappa(i+1) = \kappa(i)$;
- *Time correspondence:* $\forall 0 \le i < n : \tau(\sigma, i) = \sum_{j=0}^{\kappa(j)\le i} \tau_j$;
- *Boundary edges are urgent:* $\forall 0 \le i < m$ if e_i is *Inner* then there does not exists any *Boundary* edge e' enabled in $[\bar{x}_i, \bar{x}_i + \tau_i)$;

- *Boundary edges have priority:* $\forall 0 \leq i < m$ if e_i is *Inner* then there does not exists any *Boundary* edge e' enabled in $\bar{x}_i + \tau_i$;

Figure 3.2 illustrates the correspondence of a path σ accepted by a run ς.

Figure 3.2 Acceptance of a CMTS path σ by a DTA run ς.

$$\begin{tikzcd}
\sigma & : & s_0 \arrow[r, "t_0"] \arrow[d, dash, dotted] & s_1 \arrow[rr, "t_1"] \arrow[d, dash, dotted] \arrow[dr, dash, dotted] & & s_2 \arrow[r, "t_2"] \arrow[d, dash, dotted] & \ldots \arrow[r, "t_{n-1}"] & s_n \arrow[d, dash, dotted] \\
\varsigma & : & \langle l_0, 0\rangle \arrow[r, "\tau_0, e_0"] & \langle l_1, \bar{x}_1\rangle \arrow[r, dashed, "\tau_1, e_1"] & \langle l_2, \bar{x}_2\rangle \arrow[r, "\tau_2, e_2"] & \langle l_3, \bar{x}_3\rangle \arrow[r, "\tau_3, e_3"] & \ldots \arrow[r, "\tau_{m-1}, e_{m-1}"] & \langle l_m, \bar{x}_m\rangle
\end{tikzcd}$$

For every CMTS transition in Figure 3.2 there is an *Inner* edge in ς that follows the transition. There could be also *Boundary* edges, like e_2, that are triggered by the elapse of time. Therefore, the length m of ς could be greater than the length n of σ.

Syntax of CSL$^{\text{TA}}$.

State formulas in the stochastic temporal logic CSL$^{\text{TA}}$ have the following syntax:

Definition 22 (CSL$^{\text{TA}}$ syntax)**.** A *state formula* Φ in CSL$^{\text{TA}}$ is defined by:

$$\Phi \ ::= \ p \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \mathcal{S}_{\bowtie\lambda}(\Phi) \mid \mathcal{P}_{\bowtie\lambda}(\mathcal{A})$$

with $\mathcal{A}$ a DTA.

Definition 22 differs from the definition 15 of CSL in the probabilistic path operator $\mathcal{P}_{\bowtie\lambda}(\mathcal{A})$, which now describes the set of accepted paths through a DTA $\mathcal{A}$. A state s of $\mathcal{M}$ satisfies the CSL$^{\text{TA}}$ path operator $\mathcal{P}$ iff:

$$s \models \mathcal{P}_{\bowtie\lambda}(\mathcal{A}) \qquad \text{iff } \mathit{Prob}^{\mathcal{M}}(s, \mathcal{A}) \bowtie \lambda$$

where the probability measure $\mathit{Prob}^{\mathcal{M}}(s, \mathcal{A})$ denotes the probability of all paths in $\mathit{Cyl}(s)$ that are accepted by $\mathcal{A}$:

$$\mathit{Prob}^{\mathcal{M}}(s, \mathcal{A}) \overset{\text{def}}{=} \Pr\{\sigma \in \mathit{Cyl}(s) \mid \sigma \models \mathcal{A}\} \tag{3.12}$$

where $\sigma \models \mathcal{A}$ means that there exists a run ς in $\mathcal{A}$ that accepts σ.

3.4.2 Computation of CSL$^{\text{TA}}$ formulas

The computation of (3.12) cannot be done easily as in CSL, where for each of the two path operators $\mathcal{X}$ and $\mathcal{U}$ a fixed formula is given. The overall probability of accepting paths can be computed by defining a new stochastic process, denoted as $\mathcal{M}\times\mathcal{A}$, which describes the joint evolution of $\mathcal{M}$ with $\mathcal{A}$ from the initial state to the accepting states. The process $\mathcal{M}\times\mathcal{A}$ is enriched with two additional states: $\top$, which identifies the acceptance, and $\bot$, which identifies the rejection. Therefore:

- When the DTA is not able to follow $\mathcal{M}$, the process ends in $\bot$;
- When the DTA location is in L_F, the process ends in $\top$;

- In every other instant, the process has not yet identified if the path is accepted or rejected: therefore, the state is made by a triple $\langle s, l, \bar{x} \rangle$ which encodes the current CMTS state s, the current DTA location l and the current clock valuation $\bar{x}$;
- The computation of $\mathit{Prob}^{\mathcal{M}}(s, \mathcal{A})$ starts in state $\langle s, l_0, 0 \rangle$, with $l_0 \in L_0 \wedge s \models \Lambda(l_0)$. If there is no $l_0 \in L_0$ whose state proposition $\Lambda(l_0)$ is satisfied in state s, then the process $\mathcal{M} \times \mathcal{A}$ starts in $\perp$.

To simplify the description of the process, only states where the time may elapse are considered. Therefore, states where *Boundary* edges may trigger are removed with a transitive closure.

Definition 23 (Closure of tangible states)**.** Let $\langle s, l, \bar{x} \rangle \in \mathcal{S} \times L \times \mathbb{R}_{\geq 0}$ be a state of $\mathcal{M} \times \mathcal{A}$: then $\mathit{closure}(s, l, \bar{x})$ is defined as:

$$\frac{l \in L_F}{\mathit{closure}(s, l, \bar{x}) = \top} \ \text{CL–FINAL}$$

$$\frac{\exists!\, l \xrightarrow{x=\bar{x},\, \sharp,\, r} l' \text{ s.t. } s \models \Lambda(l') \wedge l \notin L_F}{\mathit{closure}(s, l, \bar{x}) = \mathit{closure}(s, l', \bar{x}[r := 0])} \ \text{CL–BOUNDARY}$$

$$\frac{\nexists\, l \xrightarrow{x=\bar{x},\, \sharp,\, r} l' \text{ s.t. } s \models \Lambda(l') \wedge l \notin L_F}{\mathit{closure}(s, l, \bar{x}) = \langle s, l, \bar{x} \rangle} \ \text{CL–TANGIBLE}$$

As long as there is a *Boundary* edge enabled, it is followed using the recursive rule CL–BOUNDARY. The state resulting from $\mathit{closure}(s, l, \bar{x})$ is either $\top$ if the location reached is final (by rule CL–FINAL), or is a *tangible* state with no *Boundary* edge enabled (by rule CL–TANGIBLE). Note that loops of *Boundary* edges are forbidden by definition 19, so the premise of rule CL–BOUNDARY can be satisfied by at most one *Boundary* edge.

The set of tangible states of $\mathcal{M} \times \mathcal{A}$ is constructed as a subset of:

$$\{\top, \perp\} \ \cup \ \{\langle s, l, \bar{x} \rangle \mid \langle s, l, \bar{x} \rangle = \mathit{closure}(s, l, \bar{x})\}$$

that is reachable from the initial state.

A remaining problem is that the state $\langle s, l, \bar{x} \rangle$ contains a continuous part $\bar{x}$, therefore the state space of $\mathcal{M} \times \mathcal{A}$ is not enumerable. A solution is provided in [DHS09, p. 7] using the method of *supplementary variables*[Cox55]. Let $K = \{k_0, \ldots, k_m\}$ be the set of constants that appear in all the clock guards of $\mathcal{A}$, with $k_0 = 0$, and let $\mathcal{K} = \{[k_i, k_{i+1}) \mid 0 \leq i \leq m\}$ be the set of intervals induced by K, with the last interval $[k_m, \infty)$. The $m + 1$ entries c_i of $\mathcal{K}$ partitions the set $\mathbb{R}_{\geq 0}$ into $m + 1$ *clock zones*. Given a clock zone $c_i \neq c_m$, let $\mathsf{next}(c_i)$ be the successive zone in the sequence $\mathcal{K}$, and let $\delta(c_i) = k_{i+1} - k_i$ be the *duration* of zone c_i.

The states (different from $\top$ and $\perp$) of $\mathcal{M} \times \mathcal{A}$ can be rewritten in terms of the clock zone $c_i = [k_i, k_{i+1})$ of $\mathcal{K}$ where $\bar{x}$ is encountered, generating an MRP. Let X_t be the continuous-time stochastic process of $\mathcal{M} \times \mathcal{A}$.

Theorem 3. *The process X_t is a Markov regenerative process.*

Proof. The proof for this theorem is given by showing that there exists a Markov renewal sequence embedded in $\mathcal{M} \times \mathcal{A}$. The process $\mathcal{M} \times \mathcal{A}$ can be rewritten with form $Z_t = \langle s_t, l_t, c_t, \bar{x}_t - c_t \rangle$, where $\bar{x}_t = \bar{x}_0 + t$ and c_t is the zone with interval $[k_i, k_{i+1})$ such that $\bar{x}_0 = k_i$, $\bar{x}_t \in [k_i, k_{i+1})$ and $t \in [0, \delta(c_t))$. Let $\{T_n \mid n \in \mathbb{N}\}$ be a sequence of increasing time points in the evolution of $\mathcal{M}\times\mathcal{A}$, defined as follows:

- $T_0 = 0$;
- In each point T_n the valuation $\bar{x}$ assumes a value $k \in K$;
- Given a time point T_n, the next time point T_{n+1} is taken when:
  - The valuation $\bar{x}$ enters a new clock zone $\mathsf{next}(c_t)$;
  - The valuation $\bar{x}$ is reset to 0;
  - The process $\mathcal{M}\times\mathcal{A}$ reaches the $\top$ or $\bot$ states.

Let $\{Y_n = Z_{T_n^+}\}$ be the state of Z_t right after the time point T_n. Then $\{\langle Y_n, T_n \rangle \mid n \in \mathbb{N}\}$ is a Markov renewal sequence by construction (see Def. 5), where the process Z_{T_n+t} depends only on t and on the state Y_{T_n}. □

Let $\mathbb{S}$ be the state space of Y_n. States of $\mathbb{S}$ have form $\langle s_{T_n^+}, l_{T_n^+}, c_{T_n^+} \rangle$, with $c_{T_n^+} \in \mathcal{K}$; in addition, $\mathbb{S}$ has the two terminal states $\top$ and $\bot$. Note that if Y_n reaches one of the two terminal states, the Markov renewal sequence becomes finite. Since Y_n is an MRP with *enabling restriction*, it can be characterized as an MRTS.

Construction of the MRTS of $\mathcal{M}\times\mathcal{A}$:

Let $\mathcal{R} = \langle \mathbb{S}, \mathrm{G}, \Gamma, Act, AP, lab, T, P, E, \boldsymbol{\alpha}_0 \rangle$ be the MRTS of Z_t with Y_n as its renewal sequence. The set AP is taken as that of the CMTS. The *lab* function assigns to each $\mathbb{s} \in \mathbb{S}$ the same atomic propositions as the corresponding CMTS state. The set of random variables G is defined as $\{g_i \sim \mathrm{Det}(\delta(c_i)) \mid 0 \leq i < m\}$, where each g_i has a *deterministic* duration of $\delta(c_i)$. There is a general event g_i in G assigned to each clock zone $c_i \in \mathcal{K}$, apart from the last clock zone c_m of infinite duration. The partition function Γ assigns the proper g_i variable to each state $\mathbb{s} \in \mathbb{S}$ that resides in the i-th clock zone, and assigns all other states (including $\top$ and $\bot$) in the exponential partition. The state space $\mathbb{S}$ is therefore divided into one subset $\mathbb{S}^g$ for each $g \in \mathrm{G}$, plus a subset of states $\mathbb{S}^{\mathrm{E}}$ where no general event is enabled.

The set of actions Act includes one action $a_{g_i} \in Act^{\mathrm{G}}$ for each clock zone, and a pair $a \in Act^{\mathrm{E}}, \bar{a} \in \overline{Act}^{\mathrm{E}}$ for each CMTS action a.

The state space $\mathbb{S}$ and the transition relation T (along with P and E) are constructed inductively. Let $\mathbb{S}_0$ be the set of initial states of the MRTS.

Definition 24 (Initial states of $\mathcal{M}\times\mathcal{A}$)**.** Let $\mathbb{S}_0 = \{\langle s, l_0, 0 \rangle \mid s \in \mathcal{S}, l_0 \in L_0, s \models \Lambda(l_0)\}$ be a subset of states of Y_n, one for each state $s \in \mathcal{S}$ of $\mathcal{M}$ that has an initial location $l_0 \in L_0$ that satisfies $s \models \Lambda(l_0)$. Let $init : \mathcal{S} \to \mathbb{S}_0 \cup \{\bot\}$ be a function that assigns to each CMTS state s the corresponding $\mathcal{M}\times\mathcal{A}$ state $\langle s, l_0, 0 \rangle$ if it exists, or $\bot$ otherwise.

The initial set of states $\mathbb{S}_0$ is constructed to contain all CMTS states that can be read by the DTA. In this way, given a CMTS state $s \in \mathcal{S}$, the set $\mathbb{S}_0$ contains the appropriate state $init(s)$. This construction also allows, with the method (3.14) given in the next section, to evaluate a CSL$^{\mathrm{TA}}$ formula for each initial state with a single numerical computation.

Given the initial states $\mathbb{S}_0$ and the final states $\{\top, \bot\}$, the other states of the process and the transition relation T are generated inductively by the following rules:

Definition 25 (Production rules of $\mathcal{M}{\times}\mathcal{A}$)**.** The state space $\mathbb{S}$ and the state transitions of $\mathcal{M}{\times}\mathcal{A}$ are constructed from $\mathbb{S}_0 \cup \{\top, \bot\}$ by following the rules:

$$\frac{s \xrightarrow{a} s' \sim_{\text{rate}} \mu,\ \exists!\, l \xrightarrow{\gamma, A, r} l' : s' \models \Lambda(l') \wedge a \in A \wedge c \models \gamma}{\langle s, l, c\rangle \xrightarrow{a/\bar{a}} \mathit{closure}(s', l', c[r := 0]) \ \sim_{\text{rate}}\ \mu}\ \text{M–MOVE}$$

$$\frac{s \xrightarrow{a} s' \sim_{\text{rate}} \mu,\ \nexists\, l \xrightarrow{\gamma, A, r} l' : s' \models \Lambda(l') \wedge a \in A \wedge c \models \gamma}{\langle s, l, c\rangle \xrightarrow{a/\bar{a}} \bot \ \sim_{\text{rate}}\ \mu}\ \text{M–KO}$$

$$\frac{c \neq c_m}{\langle s, l, c\rangle \xrightarrow{a_g} \mathit{closure}(s, l, \mathsf{next}(c)) \ \sim\ 1}\ \text{M–CLOCK}$$

The label $a/\bar{a}$ indicates that the CMTS transition with action name a becomes a non-preemptive Markovian transition in the MRTS (type $\mathbf{q}$) with name a, unless one of the following two conditions is met:

1. there is a reset of the clock variable x along the transition;
2. the process $\mathcal{M}{\times}\mathcal{A}$ enters the $\top$ or $\bot$ state (which is in the $\mathcal{S}^{\text{E}}$ subset) from a state with $c_i \neq c_m$ (which is in the $\mathcal{S}^{g_i}$ subset), thus preempting the general event g_i.

If one of these two conditions is satisfied, the MRTS transition is preemptive (type $\bar{\mathbf{q}}$) with action name $\bar{a}$.

Rule M–MOVE describes all Markovian transitions of $\mathcal{M}$ that are matched by an *Inner* edge of $\mathcal{A}$, such that the conditions on the state propositions, action sets and clock guards are satisfied. If the reset set of the DTA edge is $\{x\}$, the clock is reset (with $c[r := 0]$) and the closure relation triggers all the (possibly) enabled *Boundary* edges. The *closure* is also responsible of setting the target state to $\top$ when a final location is reached.

Rule M–KO describes a Markovian transition that is not matched by $\mathcal{A}$. In that case, the process moves to the $\bot$ state with rate μ.

Finally, rule M–CLOCK describes the elapse of time of the clock, and is used in every clock zone except the last. The *closure* relation is needed because the clock move could trigger *Boundary* edges or could reach a final location. Since there is no probabilistic choice, the branching probability has value 1. Clock transitions are labeled with the action a_{g_i} corresponding to the clock zone c_i with general distribution $g \in \mathrm{G}$.

The matrices $\mathbf{Q}$, $\bar{\mathbf{Q}}$ and $\boldsymbol{\Delta}$ are then constructed from the MRTS as described in section 2.4.2.

Computing the path probabilities $\mathit{Prob}^{\mathcal{M}}(s, \mathcal{A})$

Given that the embedded Markov chain can be constructed for the synchronized process $\mathcal{M}{\times}\mathcal{A}$, the probabilities of the set of paths starting from state $s \in \mathcal{S}$ of the CMTS that are matched by the DTA are equivalent to the probabilities of the set of paths starting from $\mathit{init}(s) \in \mathbb{S}$ and reaching the $\top$ state. Equation

(3.12) can be rewritten as:

$$\begin{aligned} \mathit{Prob}^{\mathcal{M}}(s,\mathcal{A}) &\stackrel{\text{def}}{=} \Pr\{\sigma \in \mathit{Cyl}^{\mathcal{S}}(s) \mid \sigma \models \mathcal{A}\} = \\ &= \Pr\{\varsigma \in \mathit{Cyl}^{\mathcal{M}\times\mathcal{A}}(\mathit{init}(s)) \mid \exists\, i : \varsigma[i] = \top\} = \\ &= \boldsymbol{\pi}^{\mathcal{M}\times\mathcal{A}}(\mathbf{i}_{\mathit{init}(s)}) \cdot \mathbf{i}_{\top} \end{aligned} \tag{3.13}$$

where $\mathbf{i}_{\mathit{init}(s)}$ and $\mathbf{i}_{\top}$ are the indicator vector for the $\mathit{init}(s)$ and the $\top$ states in $\mathbb{S}$. The computation of $\boldsymbol{\pi}^{\mathcal{M}\times\mathcal{A}}(\mathbf{i}_{\mathit{init}(s)})$ requires one steady-state solution of the embedded DTMC of $\mathcal{M}\times\mathcal{A}$.

Definition 26. Backward solution of CSL$^{\text{TA}}$ path probabilities Equation (3.13) can be rewritten using the forward-backward relation (2.26), (as done for CSL with (3.10)) leading to:

$$\underline{\mathit{Prob}}^{\mathcal{M}}(s,\mathcal{A}) \;=\; \boldsymbol{\xi}^{\mathcal{M}\times\mathcal{A}}(\mathbf{i}_{\top}) \tag{3.14}$$

Proof. The relation (3.14) can be proved for every state $s \in \mathcal{S}$ as follows:

$$\begin{aligned} \underline{\mathit{Prob}}^{\mathcal{M}}(s,\mathcal{A}) \cdot \mathbf{i}_{\mathit{init}(s)} &= \boldsymbol{\xi}^{\mathcal{M}\times\mathcal{A}}(\mathbf{i}_{\top}) \cdot \mathbf{i}_{\mathit{init}(s)} = \boldsymbol{\pi}^{\mathcal{M}\times\mathcal{A}}(\mathbf{i}_{\mathit{init}(s)}) \cdot \mathbf{i}_{\top} = \\ &= \mathit{Prob}^{\mathcal{M}}(s,\mathcal{A}) \end{aligned}$$

which completes the derivation. □

As before, formula (3.14) has the advantage of computing the path probability for each initial state of $\mathcal{M}\times\mathcal{A}$ in a single, backward computation. Therefore, the set $\mathit{Sat}(\mathcal{P}_{\bowtie\lambda}(\mathcal{A}))$ can be computed at once. This is useful for nested CSL$^{\text{TA}}$ formulas. CSL$^{\text{TA}}$ path properties are always computed as a steady state solution (either with (3.13) or (3.14)). This is different from CSL, where a mix of transient and steady-state solution is used.

The work in [Brá+11] extends automata-based stochastic logics with *semi-Markov Processes* (SMP), where non-competitive generally-distributed transitions are allowed. The work in [Che+11b] instead extends the concept of DTAs with a new logic, called *metric temporal logic* (MTL), which is claimed to be more expressive. In [Bal+11] the language of DTA is enriched toward hybrid automata, where multiple clocks (called *data variables*) are supported, along with linear constraints. However, that richer language is too hard to compute numerically, and can be evaluated only by means of simulations.

3.5 Specifying properties with Probes

A different approach to path-based measurement for CTMCs is given by directly assigning reward variables to paths. The work in [OIS98] introduced the idea of defining reward variables that capture the sequence of events and states of the modeled system. A *path-based reward structure* is a generalization of a state/action based reward structure (as defined in section 2.5). A *Path Automata* (PA) is an automaton that follows the CMTS and assign rewards to its states and actions. A PA is defined from a given reward structure, and the current state/action reward is made dependent on the PA location.

A more structured approach to path-based measures, outside of temporal logics, is provided by *Stochastic Probes*. Stochastic probes have been developed originally in [CG08] with the name XSP (for *eXtended Stochastic Probes*), and subsequently improved in [HBA11] with the name USP (for *Unified Stochastic Probes*). We will mainly refer to the latter. USP is a query mechanism for stochastic processes. It has been developed to be formalism-independent, although the original definition has been given around process algebra (PEPA) with immediate actions, state labels and action names. We provide a small summary of USP around the CMTS model formalism.

A USP query has a syntax similar to regular expressions: probe transitions are triggered by CMTS actions, while probe conditioned read CMTS state labels. The probe specifies a set of *source* state and a set of *target* states. The quantity measured is the time delay in reaching any target state from the set of start states. This is different from CSL (and partially from CSL^{TA}), because the measurement does not start immediately, but waits for a signal of the probe. A sample probe has syntax:

$$\textbf{probe} \ ::= \ \textit{initialise} : \mathsf{start}, \textit{completion} : \mathsf{stop} \tag{3.15}$$

which measures the time distribution from the *initialise* event to the *completion* event in the CMTS. The special labels start and stop indicates which probe transitions start and end the measurement.

A USP probe like (3.15) is then translated into one (or more) *Deterministic Finite Automaton* that run in parallel with the system model. Probe terms may also be *local* and follow specific parts of the system. When a term DFA first matches the stop-labeled action, the probe stops and registers the path duration. The probe starts in the initial state of the model, and follows each transition of it. When the probe reads the *initialize* action, it goes into a new location and starts measuring. Any other action read before the *initialize* action is silently ignored. When the *completion* action is read, the probe stops the measurement. After the stop event is reached, a probe may end or may be restarted: repetitive probes simulates a sort of steady-state measurement. However, since the probe started in the initial state of the model, it could be synchronized with it. Therefore, it is not an “exact” steady-state measurement. Consider, for instance, a model that generates repetitively the following path:

$$s_1 \xrightarrow{a} s_2 \xrightarrow{a} s_3 \xrightarrow{a} s_4 \xrightarrow{b} s_1 \xrightarrow{a} \ldots$$

where the sequence $s_1 \ldots s_4$ repeats forever, and consider a repetitive probe $\mathbf{p} = a : \mathsf{start}, b : \mathsf{stop}$. Since the probe $\mathbf{p}$ starts with the initial state s_1 of the CTMC, it will always read the sequence $s_1 \ldots s_4$ as a measured path, even if $s_2 \ldots s_4$ and $s_3 \ldots s_4$ are also steady-state paths that satisfies $\mathbf{p}$. This happens because the probe can maintain a form of synchronization on the verified model, that may hide these behaviors.

A formalism similar to stochastic probes is provided by *Probe Automata* (PrA), defined in [Amp+11] as an extension of XSP with a more regular steady-state semantic. A probe automaton $\mathcal{P}$ is specified as a DFA that follows the tangible execution of a GSPN model. The language of PrA allows for more control than XSP, allowing for pre and post conditions. The main peculiarity, however, is the way probe automaton defines the path measure. Instead of attaching the probe in the initial state, and observing the behavior in steady-state condition, a PrA automaton is attached at some random point in the stationary regime of the CTMC, and then the passage time is computed from that point. In this way, no residual of synchronization may happen, and the probe measures the expected mean stationary behavior.

3.5.1 Probe Automata

A PrA is a DFA with constraint-labeled edges. Given a CMTS $\mathcal{M}$ and a path $\sigma \in \mathit{Paths}_\omega^{\mathcal{M}}$, a PrA identifies a *passage subrun* in σ, that is used to compute the passage time distribution.

Definition 27 (Probe Automaton (PrA))**.** A probe automaton is a tuple $\mathcal{P} = \langle \Sigma, \mathit{Act}, L, L_0, \mathit{init}, L_P, L_F, E\rangle$ where:

- Σ is a finite set of *state propositions*;
- Act is a finite set of *action names*;
- L is a finite set of *locations*;
- $L_0 \subseteq L$ is the set of *initial locations*;
- $\mathit{init} : L \to \mathcal{B}(\Sigma)$ is the *initial location* constraint function;
- $L_P \subseteq L$ is the set of *passage locations*;
- $L_F \subseteq L$ is the set of *final locations*;
- $E \subseteq (L \setminus L_F) \times \mathcal{B}(\Sigma) \times 2^{\mathit{Act}} \times \mathcal{B}(\Sigma) \times L$ is the *edge relation*;

where $\mathcal{B}(\Sigma)$ is the language of boolean expressions over Σ. An edge $e = \langle l, \gamma_{\text{pre}}, A, \gamma_{\text{post}}, l'\rangle$ represents a transition from location l to location l', where γ_{pre} is the pre-condition, $A \subseteq 2^{\mathit{Act}}$ is the set of *activating actions*, and γ_{post} is the post-condition. For each edge e in E, it should hold that $l \in L_P \Rightarrow l' \in L_P$.

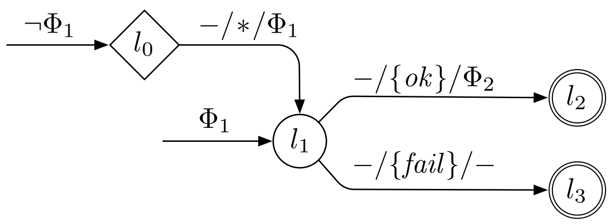
We restrict the set of probe automatons to that of *deterministic probe automatons*, where:

- For every value-assignment m over the set of state propositions Σ, there exists at most one initial location l_0 such that $\mathit{init}(l_0) \models m$.
- For every location $l \in L$, and for every action a, there exists at most one edge $e\langle l, \gamma_{\text{pre}}, A, \gamma_{\text{post}}, l'\rangle$ such that $a \in A$, γ_{pre} is satisfied in l and γ_{post} is satisfied in l'.

The state of the probe automaton $\mathcal{P}$ is given entirely by its current location $l \in L$. Final locations L_F select the states of the TRG that conclude the passage time computation, passage locations L_P select states of the TRG along which the passage time is computed, while initial locations in L_0 and *init* identify, for each TRG marking, the starting location of the probe. Once a passage location is reached, the probe may no longer visit non-passage locations until a final location is reached.

Figure 3.3 illustrates a simple PrA. Non-passage (l_0) and passage (l_1, l_2 and l_3) locations are drawn with diamonds and circles, and are labeled with a name. Final locations have a double border. Edges are labeled with constraints written as $\gamma_{\text{pre}}/A/\gamma_{\text{post}}$. Each L_0 location has an associated *init* constraint, shown as an entering arrow (l_0, l_1). The minus sign denotes the absence of a pre/post condition, and an asterisk denotes the entire *Act* set. For example, the edge from l_0 to l_1 is triggered by the firing of any action, as soon as the post-condition Φ_1 holds in the path. This probe automaton has two final locations and therefore accepts two languages. For instance, the language $\mathcal{L}_{l_2}(\mathcal{P})$ contains all the runs

Figure 3.3 Probe automaton example.



that first match the *ok* action when Φ_2 holds, after a state where Φ_1 holds has been visited.

As for DTAs of CSL$^{\text{TA}}$, probe automata are restricted to be *deterministic*, i.e. for each path σ in $\mathcal{M}$ there is a single run ς in $\mathcal{P}$ that follows σ up to a final location.

Probe automata describe a *passage subrun*, which is the subset of states in each path σ from where the probe location enter a L_P location, until an L_F location is reached. The time of each passage subrun is the time value measured by the probe over σ.

The evaluation of a PrA $\mathcal{P}$ against a model $\mathcal{M}$ works as follows. A path $\sigma \in \mathit{Cyl}(i)$ is chosen probabilistically according to an initial state $i_0 \sim \boldsymbol{\alpha}_0$. This is equivalent to starting the probe at any time when the process is in stationary regime. The probe starts in the initial location l_0 whose constraint $\mathit{init}(l_0) \models s$. A joint process $\langle l, i \rangle$ then describes the evolution from $\langle l_0, i_0 \rangle$. Let $\mathbb{S}_0$ be the set of all the possible initial $\langle l, i \rangle$ configurations. Let k be the state of $\mathcal{M}$ where the probe first enters a passage location, called the *start state*. The measurement starts in state k. Let $\mathbb{S}_E$ be the set of all possible *start states* that can be encountered starting from $\mathbb{S}_0$ states. The probe follows $\mathcal{M}$ until a state j where $\mathcal{P}$ is in a final location $l \in L_F$. Let $\mathbb{S}_{F(l)}$ be the set of all states with a final location $l \in L_F$. We now explain in more details the construction of a joint process $\mathcal{M} \times \mathcal{P}$ that incorporates the notion of these three barriers $\mathbb{S}_0$, $\mathbb{S}_E$ and $\mathbb{S}_{F(l)}$. From the joint process a CTMC can be derived, whose passage time values give the measured value of the probe.

Passage times are computed from a concatenation of two forward probabilities. First, the stationary distribution $\boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha})$ of $\mathcal{M}$ is computed. Then the set of states $\mathbb{S}_0$ of the joint process starts with the distribution $\boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha})$, and the probability of reaching the barrier $\mathbb{S}_E$ is computed. This is a forward steady-state non-ergodic computation. Finally, the passage time from the enter set $\mathbb{S}_E$ to the stop set $\mathbb{S}_{F(l)}$ is computed, which gives the measure value.

3.5.2 Computation of the passage time

A probe automaton $\mathcal{P}$ follows passively the actions of a CMTS $\mathcal{M}$. Each action is read by the probe, that has a chance of moving from its current location according to the conditions specified on the outgoing edges of l. This results in a joint process that can be constructed as a synchronized product of the probe with the state space S of $\mathcal{M}$.

The *initial states* set $\mathbb{S}_0$ of the product is a set of pairs made by a CMTS state and by a probe location, defined as:

$$\mathbb{S}_0 = \{\langle s, l\rangle \in S \times L_0 \mid s \models \mathit{init}(l)\}$$

$\mathbb{S}_0$ contains the initial configuration of the probe in each reachable state of $\mathcal{M}$. We call *product* of the probe $\mathcal{P}$ with $\mathcal{M}$ the union of all synchronized products of $\mathcal{P}$ starting from each initial state in $\mathbb{S}_0$.

Definition 28 (**Product $\mathcal{M}\times\mathcal{P}$ of a CMTS $\mathcal{M}$ with a probe $\mathcal{P}$**). A *Product* of a CMTS $\mathcal{M}$ with a probe automaton $\mathcal{P}$ is a tuple $\mathcal{M}\times\mathcal{P} = \langle \mathbb{S}, \mathbb{S}_0, \mathbb{S}_P, \mathbb{S}_F, \to, \hat{\rho}\rangle$, where:

- $\mathbb{S} \subseteq S \times L$ is a finite set of *states*, given by the cross product of CMTS state space S with the probe location set L;
- $\mathbb{S}_0 \subseteq \mathbb{S}$ is the set of *initial states*, defined before;
- $\mathbb{S}_P = (M \times L_P) \cap \mathbb{S}$ is the set of *passage states*;
- $\mathbb{S}_F = (M \times L_F) \cap \mathbb{S}$ is the set of *final states*;
- $T \subseteq \mathbb{S} \times T_{\mathcal{M}} \times \mathbb{S}$ is the *transition relation*;
- $\hat{\rho} : T \to \mathbb{R}_{\geq 0}$ is the *transition rate* function;

where $T_{\mathcal{M}}$ is the CMTS transition relation.

The state space $\mathbb{S}$ is defined inductively according to these rules:

1. The set of initial states belongs to $\mathbb{S}$: $(\mathbb{S} \cap \mathbb{S}_0) = \mathbb{S}_0$.
2. Transition rule "probe moves": let $\mathbb{s} = \langle s, l\rangle$ be a state in $\mathbb{S}$, $m = (s, a, s') \sim_{\text{rate}} \lambda$ a CMTS transition, and $e = (l, \gamma_{\mathsf{pre}}, X, \gamma_{\mathsf{post}}, l')$ a probe edge in E, such that $m \models e$. Then the state $\mathbb{s}' = \langle s', l'\rangle$ is also in $\mathbb{S}$ and the arc $t = (\mathbb{s}, m, \mathbb{s}')$ is in T, with its exponential transition rate $\hat{\rho}(t) = \lambda$.
3. Transition rule "probe stands still": let $\mathbb{s} = \langle s, l\rangle$ be a state in $\mathbb{S}$ with $l \notin L_F$, let $m = (s, a, s') \sim_{\text{rate}} \lambda$ be a CMTS transition, such that there is no probe edge $e \in E$ where $m \models e$ is true. Then the state $\mathbb{s}' = \langle s', l\rangle$ belongs to $\mathbb{S}$, and the arc $t = (\mathbb{s}, a, \mathbb{s}')$ is in T, with rate $\hat{\rho}(t) = \lambda$.

The product $\mathcal{M}\times\mathcal{P}$ is *deterministic* if and only if:

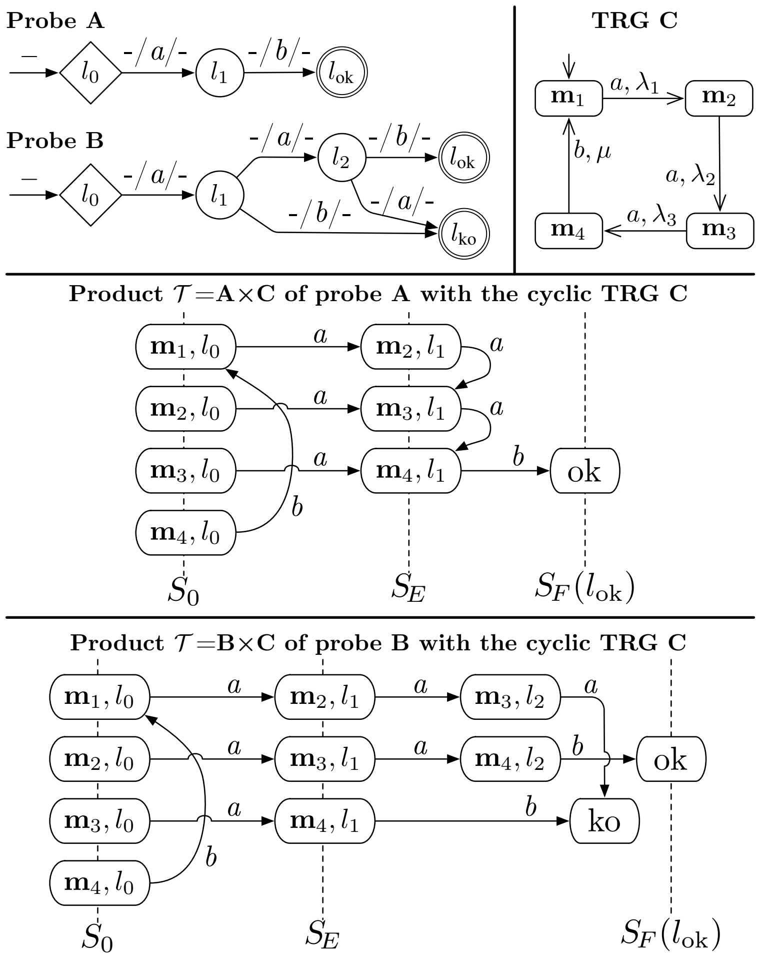
- *Initial location determinism*: for each CMTS state $s \in S$, there exists at most one initial location $l \in L_0$ such that $s \models \mathit{init}(l)$.
- *Probe edges determinism*: for each state $\mathbb{s} = \langle s, l\rangle$ and for each CMTS transition $m = (s, a, s')$ leaving s, there exists at most one location $l' \in L$ s.t. any probe edges $e \in E$ that satisfies the transition m, i.e. $m \models e$, reaches location l'.

When $\mathcal{M}\times\mathcal{P}$ is deterministic, each path τ in $\mathit{Paths}_\omega^{\mathcal{M}}$ is matched by a single run r of the probe. Therefore, the stochastic behavior of $\mathcal{M}\times\mathcal{P}$ is fully determined and can be described as a CTMC. Given a (infinite) path τ in $\mathit{Paths}_\omega^{\mathcal{M}}$ starting in s, there exists a unique path σ rooted in $\langle s, l\rangle$, with $s \models \mathit{init}(l)$, that follows each transition in τ until the probe reaches a final location. Such path σ is unique due to determinism constraints. Vice versa, the sequence of states of each path σ rooted in $\langle s, l\rangle$ corresponds to the (infinite) set $J(\sigma)$ of CTMC paths in $\mathit{Paths}_\omega^{\mathcal{M}}$, which share the common prefix. Since transition rates are "copied" by the construction rules of $\mathcal{M}\times\mathcal{P}$, the continuous-time behavior of each $\mathcal{M}\times\mathcal{P}$ path σ is the same behavior of the common prefix of each path in $J(\sigma)$.

The state space of $\mathcal{M}\times\mathcal{P}$ has an asymptotical memory occupation of $O(|M| \cdot |L|)$, since the state space $\mathbb{S}$ is a subset of the Cartesian product of CMTS state space S with the location set L. A deterministic product $\mathcal{M}\times\mathcal{P}$ may be treated as a CMTS with reachable state space $\mathbb{S}$, transition set T and rate function $\hat{\rho}$. Unlike CSL$^{\text{TA}}$, probe automata produces a CTMC as a solution object, and not an MRP.

Figure 3.4 shows two simple probes (A and B), a cyclic CMTS C, and the products of both probes with C. The sets $\mathbb{S}_E$ and $\mathbb{S}_F(l_{\text{ok}})$ highlighted in the products will be explained later, and constitutes a sort of "barriers" used for the passage time computation.

Figure 3.4 Probes, CMTS and products.



Observe that we are considering all the possible initial states $\mathbb{S}_0$ for the product state space, since we are assuming that the probe may start "randomly" in any possible state of the model. The probe partitions the state space $\mathbb{S}$ into the passage set $\mathbb{S}_P$ (on the right of the $\mathbb{S}_E$ line) and the non-passage set (on the left). All the possible runs are indeed in the product CMTS $\mathcal{M}\times\mathcal{P}$.

From a CMTS $\mathcal{M}$, a CTMC is derived, as usual, in the domain space of

$\mathbb{S}$ by summing up the outgoing transition rates that leave each marking. Let $\{X(t) \mid t \in \mathbb{R}_{\geq 0}\}$ be the stochastic process that behaves according to the CTMC given by $\mathcal{M}{\times}\mathcal{P}$, and let $\mathbf{Q}$ be its infinitesimal generator. Entries of $\mathbf{Q}$ are:

$$\mathbf{Q}(i,j) \;=\; \begin{cases} \displaystyle\sum_{\substack{t=(i,a,j)\\ t\in T}} \rho(t) & i \neq j \\ \displaystyle -\sum_{k\neq i} \mathbf{Q}(i,k) & i = j \end{cases}$$

In order to compute the passage time on the underlying CTMC, we need the set of *initial* states $\mathbb{S}_0$ and two additional state subsets: the set of *start* states $\mathbb{S}_E$, where the measure time starts, and the *target* states $\mathbb{S}_{F(l)}$, for a given final location $l \in L_F$. Hence:

$$\mathbb{S}_E \overset{\text{def}}{=} \left\{ \mathbb{s}' \in \mathbb{S}_P \mid \mathbb{s}' \in \mathbb{S}_0 \;\vee\; \big(\exists \mathbb{s} \in (\mathbb{S} \setminus \mathbb{S}_P) \;\wedge\; \exists(\mathbb{s}, a, \mathbb{s}') \in T\big)\right\}$$

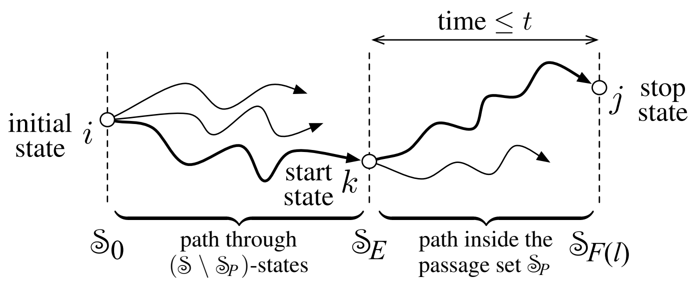
is the set of states where the probe enters the *passage set* for the first time, starting the *passage time* computation. Let:

$$\mathbb{S}_{F(l)} \overset{\text{def}}{=} \{\langle s, l'\rangle \in \mathbb{S} \mid l' = l\}, \qquad l \in L_F$$

be the set of states where the joint process accepts a run of $\mathcal{M}{\times}\mathcal{P}$ in the specified final location l, thus ending the passage time computation.

We consider that a probe may start at any time in the process described by the CMTS, in stationary conditions, so a marking s is chosen according to its probability in the steady state distribution $\boldsymbol{\pi}$ of $\mathcal{M}$. The probe starts in the initial location l_0 satisfied by the *init* constraint. Therefore, the joint process $\mathcal{M}{\times}\mathcal{P}$ begins in the state $i = \langle s, l_0\rangle$, as shown in Figure 3.5. The stochastic process $X(t)$ then continues its execution until an $\mathbb{S}_P$ state is reached, i.e. a state $k \in \mathbb{S}_E$ is first entered. If the initial state i is already an entering state (i.e. $i \in \mathbb{S}_E$), then $k = i$. At this point a clock starts, and the execution continues in the *passage subset* $\mathbb{S}_P$. The measure *Passage*(t, l) is the probability of reaching a state $j \in \mathbb{S}_{F(l)}$ before time t, starting at time 0 from any k.

Figure 3.5 Passage time for a path σ.



Passage time measure for a probe automaton $\mathcal{P}$ in $\mathcal{M}$ can be easily described by means of accepted runs. The measure *Passage* : $\mathbb{R}_{\geq 0} \times L_F \to \mathbb{R}_{[0,1]}$ describes the cumulative distribution function for the passage time that starts in an $\mathbb{S}_E$-state and ends in the specified $\mathbb{S}_{F(l)}$-state in less that $t \in \mathbb{R}_{\geq 0}$, conditioned on an initial probability distribution over $\mathbb{S}_0$-states. Let $\boldsymbol{\pi}$ be such distribution over $\mathbb{S}_0$-states in the product $\mathcal{M}\times\mathcal{P}$. Then the *Passage*$(t, l)$ at time up to t to a set of final states $\mathbb{S}_{F(l)} = \{\langle s, l'\rangle \in \mathbb{S}\}$ is:

$$Passage(t,l) = \underbrace{\sum_{i\in\mathbb{S}_0} \boldsymbol{\pi}(i)\cdot}_{(1)} \underbrace{\sum_{k\in\mathbb{S}_E} H^{\mathbb{S}\setminus\mathbb{S}_P}_{i,k}\cdot}_{(2)} \underbrace{\int_0^t \sum_{j\in\mathbb{S}_{F(l)}} H^{\mathbb{S}_P}_{k,j}(x)\,\mathrm{d}x}_{(3)}$$

where $H^{\mathbb{S}\setminus\mathbb{S}_P}_{i,k}$ is the long-run probability of going from state i to state k passing through $(\mathbb{S}\setminus\mathbb{S}_P)$ states only (note that $k \in \mathbb{S}_E$, which implies that $k \in \mathbb{S}_P$), and $H^{\mathbb{S}_P}_{k,j}(x)$ is the probability of going from state k to state j at time x passing only through $\mathbb{S}_P$-states.

The term $H^{\mathbb{S}'}_{i,j}(t) = Pr\{Z'(t) = j \mid Z'(0) = i\}$ is the *kernel matrix* of the stochastic process $\{X'(t) \mid t \geq 0\}$. Such process $X'(t)$ behaves like $X(t)$ restricted to the subset $\mathbb{S}' \subseteq \mathbb{S}$, which means that transitions may occur in $\mathbb{S}'$-states only. Therefore, the infinitesimal generator $\mathbf{Q}'$ of $X'(t)$ has non-zero rows only for $\mathbb{S}'$ states. The term $H^{\mathbb{S}'}_{i,j}$ is the limit $\lim\limits_{t\to\infty} H^{\mathbb{S}'}_{i,j}(t)$.

Remember that all $\mathbb{S}_F$ states are absorbing by construction, and $\mathbb{S}_{F(l)} \subseteq \mathbb{S}_F$. The integral (3) is the cumulative passage time distribution up to time t from an entering state $k \in \mathbb{S}_E$ to any final state in the target set $\mathbb{S}_{F(l)}$.

Definition 29 (Computation of *Passage*(t, l))**.** The passage time distribution computation at time t for a specified final location $l \in L_F$ can be computed with these steps:

1. Generate the cross-product process $\mathcal{M}\times\mathcal{P}$.
2. Starting from $\mathbb{S}_0$-states with initial steady state probability distribution $\boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha})$, compute the probability distribution of reaching the entering set $\mathbb{S}_E$, assuming that $\mathbb{S}_E$ states are made absorbing.
3. Compute the transient probability of reaching, from the $\mathbb{S}_E$ state subset of $\mathcal{M}\times\mathcal{P}$, the subset $\mathbb{S}_F(l)$ at time t.

As the reader may observe, the measure *Passage*(t, l) can be seen as a steady-state measure, in the sense that the accepted sequences do not depend on the initial probability distribution $\boldsymbol{\alpha}$ of the CMTS. Instead, it is more like inserting the probe in the CMTS in steady-state, and observing the probe behavior from that point on.

Tool support and experimentation.

A prototype implementation of probe automata exists as an add-on of GreatSPN [Baa+09]. It generates the CTMC derived by the product between the probe automaton and a GSPN, and exports it to HYDRA [DHK03] for the computation of the (first) passage time distribution (from $\mathbb{S}_E$ to $\mathbb{S}_{F(l)}$), while GreatSPN is used to compute the initial distribution of the entering states (up to $\mathbb{S}_E$). The CTMC generation and the product between the GSPN and the probe is based on Meddly [BM10], an existing open-source library for Decision Diagrams. The solution takes advantage (in terms of both memory utilization and execution time) of the efficacy of Multi-valued Decision Diagrams (MDDs) and of the saturation approach [CLS01] for state space generation. An empirical assessment of probe automata can be found in [Amp+11].

Final remarks

Temporal logics like CSL and CSL$^{\mathrm{TA}}$ provides a general framework for logic formulas that incorporates probabilistic path-based measurements. PrA and USP provide an explicit path-based measurement oriented to the computation of passage time measures. In general, it can be said that a path measure can be computed for a process $\{X_t\}$ by constructing a new process $\{X'_t\}$ that mimics X_t until the measure is recognized to be satisfied or not. This method has shown to be very flexible and efficient, and has been a large area of research for stochastic processes in the last 20-or-more years.

It can be said that path measures extend the possibility of expressing properties of Markov chains, and at the same time their solution still rely on the analytical foundation of Markov chains: computation of reward measures on forward and backward probabilities in transient and steady-state conditions.

Chapter 4

Numerical computation of measures

In chapter 2, the basic definitions for computing measures have been provided for DTMC, CTMC and MRP. In particular, the focus was about forward/backward measures computed in transient and steady state. The subject of MRP has been considered only in steady-state, by providing the solution of its embedded Markov process and by converting it back to the MRP solution. Finally, reward measures have been shown to be computable using the previously introduced formulas.

In chapter 3 the class of computable measures has been extended to path properties. Paths are expressed using formal temporal logics, which compute the probability of set of accepted paths. The computation of these probabilities for PCTL, CSL and CSL$^{\mathrm{TA}}$ reduces to a set of transient/steady-state computations on (modified) Markov processes. In particular, PCTL analyses DTMC, CSL uses CTMCs and CSL$^{\mathrm{TA}}$ uses an MRP.

Therefore, all the presented computations reduce to either a forward/backward analysis of a discrete-time or a continuous-time Markov chain. For MRPs, the DTMC is represented as the embedded Markov chain with equation (2.45).

This chapter focuses on the numerical analysis of transient and steady-state measures. The ultimate goal of the chapter is to introduce a set of techniques used to improve the performance of the model checking of CSL$^{\mathrm{TA}}$, which is the main contribution of the thesis.

4.1 Transient solution methods

Given a DTMC $\{Y_n \mid n \in \mathbb{N}\}$ with stochastic matrix $\mathbf{P}$, a discrete step $n \in \mathbb{N}$ and two boundary conditions $\boldsymbol{\alpha}, \boldsymbol{\rho} \in \mathit{Measure}(\mathcal{S})$ for Y_0 and Y_n, we have the two equations: (2.3) and (2.6):

$$\boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha}, n) \;=\; \boldsymbol{\alpha} \cdot \mathbf{P}^n, \qquad \boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}, n) \;=\; \mathbf{P}^n \cdot \boldsymbol{\rho}$$

which describe forward and backward transient probabilities. These two formulas are easily computed as-is, by multiplying n times the vector $\boldsymbol{\alpha}$ (or $\boldsymbol{\rho}$) with the stochastic matrix $\mathbf{P}$. Since $\mathbf{P}$ has no negative entries, the computation is numerically stable. It is also stable for reducible DTMC, although the formulas (2.12) provide a more efficient computation.

Transient computation is slightly different for CTMCs. Given a CTMC $\{X_t \mid t \in \mathbb{R}_{\geq 0}\}$ with infinitesimal generator $\mathbf{Q}$, a time step $t \in \mathbb{R}_{\geq 0}$ and two boundary conditions $\boldsymbol{\alpha}, \boldsymbol{\rho} \in \mathit{Measure}(\mathcal{S})$ for X_0 and X_t, we have the two forward and backward equations (2.21) and (2.25):

$$\boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha}, t) \;=\; \boldsymbol{\alpha} \cdot \mathrm{e}^{\mathbf{Q}t}, \qquad \boldsymbol{\xi}^{\mathcal{M}}(\boldsymbol{\rho}, t) \;=\; \mathrm{e}^{\mathbf{Q}t} \cdot \boldsymbol{\rho} \tag{4.1}$$

The computation of these two formulas require to compute a vector-product with a matrix exponential of $\mathbf{Q}t$. The work in [ML78] and in [Ste94] provide a survey of the numerical methods that can be used to compute such expressions.

In the case of deterministic general events, the two MRP matrices $\boldsymbol{\Omega}$ and $\boldsymbol{\Psi}$ are also defined as a matrix exponential $\mathrm{e}^{\mathbf{Q}^g\, t}$ and as an integral of a matrix exponential $\int_0^\infty \mathrm{e}^{\mathbf{Q}^g\, x}\,\mathrm{d}x$. These two qantities can be derived as follows.

A common technique is the *uniformisation* method (also called Jensen's method [TRS87]), which was already used in (2.28). Uniformisation exploits the Taylor approximation:

$$\mathrm{e}^{\mathbf{Q}t} \;=\; \sum_{k=0}^{\infty} \frac{(\mathbf{Q}t)^k}{k!}, \qquad \int_0^\infty \mathrm{e}^{\mathbf{Q}^g\, x}\,\mathrm{d}x \;=\; \sum_{k=0}^{\infty} \frac{1}{k!} \int_0^\infty (\mathbf{Q}x)^k\,\mathrm{d}x, \tag{4.2}$$

Since $\mathbf{Q}$ contains both positive and negative entries, the above formula is not numerically stable. A common transformation of (4.2) works by defining a DTMC process $\{Y_n \mid n \in \mathbb{N}\}$ that samples the CTMC X_t at identically-distributed time intervals of length $q \geq -\max_{i \in \mathcal{S}} \mathbf{Q}(i,i)$, so that $Y_n = X_{nq}$ for all $n \in \mathbb{N}$. The stochastic matrix of Y_n is given by $\mathbf{U} = {}^1\!/_q\mathbf{Q} + \mathbf{I}$, which can be used to rewrite (4.1) into (2.28).

The multiplications in (2.28) are done only with positive vector and matrixes, making these formulas numerically stable. Infinite summations can be approximated with a proper truncation interval $[R, L]$, leading to:

$$\begin{aligned} \mathrm{e}^{\mathbf{Q}t} \;&\approx\; \sum_{k=L}^{R} \left(\frac{e^{-qt}(qt)^k}{k!} \cdot \mathbf{U}^k \right) \;=\; \sum_{k=L}^{R} \beta(k, qt) \cdot \mathbf{U}^k \\ \int_0^\infty \mathrm{e}^{\mathbf{Q}^g\, x}\,\mathrm{d}x \;&\approx\; \sum_{k=L}^{R} \left(\int_0^\infty \frac{e^{-qx}(qx)^k}{k!}\,\mathrm{d}x \cdot \mathbf{U}^k \right) \;=\; \sum_{k=L}^{R} \hat{\beta}(k, qt) \cdot \mathbf{U}^k \end{aligned} \tag{4.3}$$

where $\beta(k, qt) = (k!)^{-1} e^{-qt} (qt)^k$ is the probability of having k renewals in a Poisson process with rate q at time t, and $\hat{\beta}(k, qt) = 1 - \sum_j = 0^{k-1} \beta(k, qt)$. The truncation points are chosen such that the modulus of the sum of the excluded entries is less than a small ϵ. The method of [FG88] can be used to compute L, R and all the $\beta(k, qt)$ and $\hat{\beta}(k, qt)$ coefficients for $k \in [L, R]$.

With (4.3) the forward and backward equations for CTMCs become:

$$\begin{aligned} \boldsymbol{\pi}^{\mathcal{M}}(\boldsymbol{\alpha}, t) \;&\approx\; \sum_{k=L}^{R} \left(\frac{e^{-qt}(qt)^k}{k!} \cdot \big(\boldsymbol{\alpha} \cdot \mathbf{U}^k\big) \right) \;=\; \sum_{k=L}^{R} \beta(k, qt) \cdot \big(\boldsymbol{\alpha} \cdot \mathbf{U}^k\big) \\ \boldsymbol{\xi}^{\mathcal{M}}(\boldsymbol{\rho}, t) \;&\approx\; \sum_{k=L}^{R} \left(\frac{e^{-qt}(qt)^k}{k!} \cdot \big(\mathbf{U}^k \cdot \boldsymbol{\rho}\big) \right) \;=\; \sum_{k=L}^{R} \beta(k, qt) \cdot \big(\mathbf{U}^k \cdot \boldsymbol{\rho}\big) \end{aligned} \tag{4.4}$$

These equations are numerically stable since they employs only products of non-negative vectors with a non-negative matrix $\mathbf{U}$.

4.2 Steady-state solution methods

Limiting forward and backward equations for DTMC and CTMC can be reduced to the solution of a linear equation system. The solution of the embedded DTMC of a MRP can be assimilated to the solution of a DTMC.

All linear equation systems share the standard form:

$$\mathbf{A}\,\mathbf{x} = \mathbf{b} \tag{4.5}$$

with $\mathbf{A}$ an $N \times N$ matrix, $\mathbf{b} \in \mathbb{R}^N$, and $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ is the solution to be found. We take back the DTMC and CTMC solutions to the standard form.

Given a DTMC $\{Y_n \mid n \in \mathbb{N}\}$ with stochastic matrix $\mathbf{P}$, the limiting forward and backward vectors $\boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha})$ and $\boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\alpha})$ are given by (2.8) and (2.9). Since $\boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha})$ does not depend on $\boldsymbol{\alpha}$ we drop it and write simply $\boldsymbol{\pi}^{\mathcal{D}}$.

A DTMC is in steady state if Y_{n+1} has the same distribution of Y_n. Let $\boldsymbol{\pi}^{\mathcal{D}}$ be that distribution. By equation (2.2) we derive that $\boldsymbol{\pi}^{\mathcal{D}} = \boldsymbol{\pi}^{\mathcal{D}}\mathbf{P}$, and therefore:

$$\big(\mathbf{P} - \mathbf{I}\big)^T \cdot \boldsymbol{\pi}^{\mathcal{D}} = 0 \qquad \Rightarrow \qquad \mathbf{A} = (\mathbf{P} - \mathbf{I})^T \text{ and } \mathbf{b} = 0 \tag{4.6}$$

with the additional condition that $\boldsymbol{\pi}^{\mathcal{D}}$ is a probability distribution, i.e. $\boldsymbol{\pi}^{\mathcal{D}}\mathbf{e} = 1$.

The backward solution defines the boundary condition: $\lim\limits_{n\to\infty} \boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}, n) = \boldsymbol{\rho}$. By equation (2.6) we know that:

$$\boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}) = \lim_{n\to\infty} \boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}, n) = \lim_{n\to\infty} \mathbf{P}^n \cdot \boldsymbol{\rho} = (\mathbf{I} - \mathbf{P})^{-1}\boldsymbol{\rho}$$

since $\lim\limits_{n\to\infty} \mathbf{P}^n = (\mathbf{I} - \mathbf{P})^{-1}$, so the linear equation system for $\boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho})$ is:

$$\big(\mathbf{P} - \mathbf{I}\big) \cdot \boldsymbol{\xi}^{\mathcal{D}}(\boldsymbol{\rho}) = \boldsymbol{\rho} \qquad \Rightarrow \qquad \mathbf{A} = (\mathbf{P} - \mathbf{I}) \text{ and } \mathbf{b} = \boldsymbol{\rho} \tag{4.7}$$

An almost equivalent derivation can be done for CTMCs. Given a CTMC $\{X_t \mid t \in \mathbb{R}_{\geq 0}\}$ with infinitesimal generator $\mathbf{Q}$, we may derive the forward and backward vectors $\boldsymbol{\pi}^{\mathcal{M}}$ and $\boldsymbol{\xi}^{\mathcal{M}}$ as the solutions of the two linear equation systems:

$$\mathbf{Q}^T \cdot \boldsymbol{\pi}^{\mathcal{M}} = 0 \qquad \Rightarrow \qquad \mathbf{A} = \mathbf{Q}^T \text{ and } \mathbf{b} = 0 \tag{4.8}$$

$$\mathbf{Q} \cdot \boldsymbol{\xi}^{\mathcal{M}}(\boldsymbol{\rho}) = \boldsymbol{\rho} \qquad \Rightarrow \qquad \mathbf{A} = \mathbf{Q} \text{ and } \mathbf{b} = \boldsymbol{\rho} \tag{4.9}$$

Equations (4.6)–(4.9) shows that both forward and backward steady-state vectors can be computed by solving a proper linear equation system in standard form (4.5).

The linear properties of a Markov chain are expressed by the *Perron-Frobenius* theorem, which gives the basic properties of stochastic matrices.

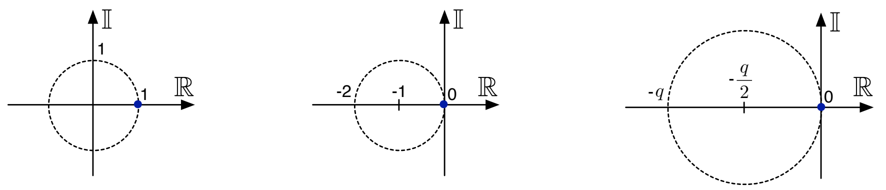
Definition 30 (*Perron-Frobenius* theorem)**.** Given a non-negative matrix $\mathbf{A}$ (i.e. $\mathbf{A}(i,j) \geq 0$ for all i,j), if $\mathbf{A}$ is *primitive*[1] and *irreducible*, then:

[1]A primitive Markov chain is sometimes called an *aperiodic* Markov chain.

- The maximum eigenvalue λ^* is positive and real;
- There is a single eigenvector $\mathbf{v}^*$ with eigenvalue λ^*;
- The eigenvector $\mathbf{v}^*$ is positive real, and has no imaginary parts;
- Every other eigenvector of $\mathbf{A}$ is negative;

The uniqueness of $\mathbf{v}^*$ depends on the primitivity (aka periodicity) of $\mathbf{P}$. If $\mathbf{A}$ is *irreducible* but not *primitive*, then there is a set of k eigenvectors $\mathbf{v}^*_1 \dots \mathbf{v}^*_k$, with the same eigenvalue λ^*.

Figure 4.1 Specters of $\mathbf{P}$, $(\mathbf{P} - \mathbf{I})$ and $\mathbf{Q}$, with $q = -\mathsf{max}\mathbf{Q}(i, i)$.



(a) Spectral radius of $\mathbf{P}$ **(b)** Spectral radius of $\mathbf{A} = \mathbf{P} - \mathbf{I}$ **(c)** Spectral radius of $\mathbf{A} = \mathbf{Q}$

Figure 4.1 clarifies the spectral properties of Markov chain matrices. The three circles show where eigenvalues of $\mathbf{P}$, of $(\mathbf{P} - \mathbf{I})$ and of $\mathbf{Q}$ lie in the complex plane. Note that $\mathbf{P} - \mathbf{I}$ is the equation system of (4.5). The Perron-Frobenius theorem refers to the diagram (a). For stochastic matrices, the eigenvalue λ^* has value 1, and every other eigenvalue lies in the unit disk with center in the origin. Since any eigenvector other than $\mathbf{v}^*$ has an eigenvalue that is less than 1, the product $\boldsymbol{\alpha} \lim_{n\to\infty} \mathbf{P}^n$ must be equal to $\mathbf{v}^*$, for any non-zero vector $\boldsymbol{\alpha}$. Therefore $\mathbf{v}^*$ is the solution vector $\mathbf{x}$ of $\mathbf{A}\,\mathbf{x} = \mathbf{b}$. For non-primitive matrices, there is a vector space made by the linear span $\{\mathbf{v}^*_1 \dots \mathbf{v}^*_k\}$ of eigenvectors with module λ^*, such that any vector $\mathbf{x}$ in the span is a solution of $\mathbf{A}\,\mathbf{x} = \mathbf{b}$. Spectral properties have a role in the interpretation of the effect of preconditioning, that will be explained later. In few words, the disposition of eigenvalues on the unit disk determines the conditioning, and thus the effectiveness and reliability of numerical solution methods.

Figure 4.2 Specters of $\mathbf{P}$ for increasingly ill-conditioned systems.

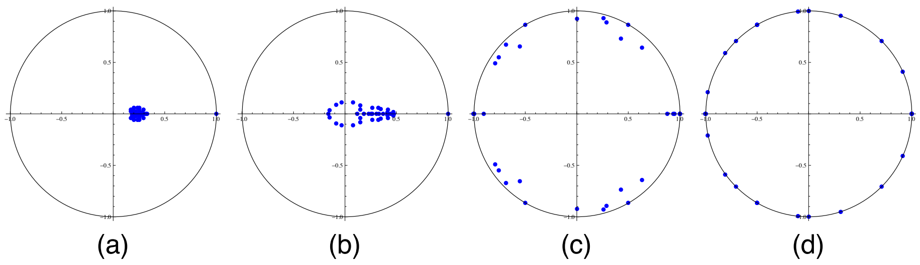


Figure 4.2 shows how the spectrum of a stochastic matrix appears, for increasingly ill-conditioned $\mathbf{P}$ matrices ((a) is well-conditioned, (d) is the most ill-conditioned). It can be said that the more the eigenvalues of a stochastic matrix are spread and near the disk perimeter (and have a norm near 1), the

more the system is ill-conditioned; non-unit eigenvalues with small norms and clustered around the zero means instead that the system is well-conditioned and favorable to a numerical solution.

Solving well-conditioned system is an important requirement, since ill-conditioned systems may break numerical solvers or could lead to potentially inaccurate solutions. In section 4.2.1 we look at a practical way of improving the conditioning of linear systems.

4.2.1 Preconditioning linear systems

Section 4.2 framed the problem of computing the forward and backward limiting behavior of Markov chains. So, the entire problem reduces to the solution of a system of linear equations: $\mathbf{A}\,\mathbf{x} = \mathbf{b}$. Before describing the numerical methods that can be used to compute a linear equation system, we consider the problem of transforming (4.5) in an easier problem.

Linear equation systems can be transformed to improve the accuracy and the reliability of the numerical methods. The main technique for doing this kind of transformations is *preconditioning*. The use of *preconditioning* usually improves the convergence rate and the reliability of a numerical solution method.

There are three general types of preconditioning [Auz11]:

1. *Left preconditioning* by a matrix $\mathbf{M}_L$:
$$\mathbf{M}_L^{-1}\mathbf{A}\,\mathbf{x} = \mathbf{M}_L^{-1}\mathbf{b} \tag{4.10}$$
2. *Right preconditioning* by a matrix $\mathbf{M}_R$:
$$\mathbf{A}\mathbf{M}_R^{-1}\,\mathbf{y} = \mathbf{b}, \qquad \mathbf{x} = \mathbf{M}_R^{-1}\,\mathbf{y} \tag{4.11}$$
which involves a substitution of $\mathbf{y}$ for the original variable $\mathbf{x}$.
3. *Split preconditioning* by a matrix $\mathbf{M} = \mathbf{M}_L\,\mathbf{M}_R$ multiplied on both sides:
$$\mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1}\,\mathbf{y} = \mathbf{M}_L^{-1}\mathbf{b}, \qquad \mathbf{x} = \mathbf{M}_R^{-1}\,\mathbf{y} \tag{4.12}$$

Split preconditioning extends both (4.10) and (4.11) by setting $\mathbf{M}_R = \mathbf{I}$ or $\mathbf{M}_L = \mathbf{I}$, respectively. The matrices $\mathbf{M}_L$ and $\mathbf{M}_R$ are called *preconditioners*. A *preconditioner* $\mathbf{M}$ is a matrix that transforms the coefficient matrix $\mathbf{A}$ into another system that is more favorable for an iterative solution. Usually, a preconditioned system has more non-unit eigenvectors clustered together around the zero.

Preconditioners are always evaluated as a product $\mathbf{M}^{-1}\mathbf{u}$ with some vector $\mathbf{u}$. An evaluation of a product $\mathbf{v} = \mathbf{M}^{-1}\,\mathbf{u}$ has to be intended as the solution of a linear equation system $\mathbf{M}\,\mathbf{v} = \mathbf{u}$. Of course, the convenience of preconditioning requires that such solution has to be cheap, in comparison to the solution of the original system.

The choice of left, right and split preconditioning is not particularly relevant. In fact, $\mathbf{M}_L^{-1}\mathbf{A}$, $\mathbf{A}\mathbf{M}_R^{-1}$ and $\mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1}$ have the same spectrum of eigenvalues. Of course, in floating point arithmetic there could be minor differences. In any case, it is not needed to form the actual system matrix $\mathbf{M}^{-1}\mathbf{A}$ explicitly (or $\mathbf{A}\mathbf{M}^{-1}$, $\mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1}$).

There is a wide literature on preconditioning methods [Ben02], either general purpose or tied to some specific problems. *Algebraic* preconditioners are built from a direct manipulation of the coefficient matrix $\mathbf{A}$. A set of classical algebraic preconditioners is given below.

Classical algebraic preconditioners.
Let say that $\mathbf{A}$ is given in split form: $\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U}$, where $\mathbf{D}$ is the diagonal part of $\mathbf{A}$, $\mathbf{L}$ is the strictly lower-diagonal part of $\mathbf{A}$, and $\mathbf{U}$ is the strictly upper-diagonal part of $\mathbf{A}$. Three different preconditioners can be constructed as:

$$\begin{aligned}
&\text{(Jacobi)} && \mathbf{M} = \mathbf{D} && (4.13)\\
&\text{(Gauss-Seidel)} && \mathbf{M} = \mathbf{D} + \mathbf{L} && (4.14)\\
&\text{(Symmetric SOR)} && \mathbf{M} = (\mathbf{D} + \mathbf{L})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}) && (4.15)
\end{aligned}$$

Each of these three algebraic preconditioners is then used either as left or right preconditioners. The evaluation of $\mathbf{M}^{-1}\mathbf{x}$ is easily done because $\mathbf{D}$ is easily invertible, and by forward and backward substitution for $\mathbf{L}$ and $\mathbf{U}$. The rationale for these preconditioners is that are in some sort an approximate of $\mathbf{A}$, taken by removing some entries of the original coefficient matrix.

Incomplete LU factorization preconditioners.
The *Incomplete LU factorization* (ILU) method [Saa94] is another widely used algebraic preconditioner. It has been shown in [Saa95] that *ILU with threshold* (ILUT) [Ste94, pag. 145] is generally a good choice for the solution of Markov chains. ILU factorizes matrix $\mathbf{A}$ as $\mathbf{A} = \widetilde{\mathbf{L}}\widetilde{\mathbf{U}} - \mathbf{E}$, where $\mathbf{E}$ contains the *remainder* of the incomplete factorization. The preconditioner is then computed as:

$$\mathbf{M} = \text{ILU}(\mathbf{A}) = \widetilde{\mathbf{L}}\widetilde{\mathbf{U}}, \qquad \mathbf{M}^{-1} = \widetilde{\mathbf{U}}^{-1}\widetilde{\mathbf{L}}^{-1} \tag{4.16}$$

This equation results in a non-singular matrix $\mathbf{M}$ when $\|\mathbf{E}\| \neq 0$. The choice of $\mathbf{M} = \widetilde{\mathbf{L}}\widetilde{\mathbf{U}}$ allows to evaluate efficiently $\mathbf{M}^{-1}\mathbf{x}$, since both $\widetilde{\mathbf{L}}$ and $\widetilde{\mathbf{U}}$ are triangular matrices: Therefore $\mathbf{y} = \widetilde{\mathbf{L}}^{-1}\mathbf{x}$ and $\mathbf{M}^{-1}\mathbf{x} = \widetilde{\mathbf{U}}^{-1}\mathbf{y}$ can be computed by forward and backward substitution [Auz11, p. 113].

The criteria for choosing the approximation $\widetilde{\mathbf{L}}\widetilde{\mathbf{U}} \approx \mathbf{LU}$ depend on the variant [CV94] of ILU factorization, and is usually one of the following:

- **ILU**(0): keep only the entries $\widetilde{\mathbf{L}}(i,j)$ and $\widetilde{\mathbf{U}}(i,j)$ if the entry $\mathbf{A}(i,j)$ is non-zero.
- **ILUT**(σ,ϵ): discard all entries in $\widetilde{\mathbf{L}}$ and in $\widetilde{\mathbf{U}}$ whose absolute value is below the threshold ϵ. Of the remaining entries, keep at most the σ largest entries per each row of $\widetilde{\mathbf{L}}$ and $\widetilde{\mathbf{U}}$.

Other class of preconditioners include *approximate inverses*, which try to construct directly a sparse approximate $\mathbf{M}^{-1}$ of $\mathbf{A}^{-1}$. A special class of preconditioners for MRPs will be shown later in section 4.3, and is part of the contribution of this thesis.

4.2.2 Numerical solution of linear systems

This section gives a brief description of the classes of numerical methods that can be used to compute the solution of (4.5), employing a preconditioner matrix when it is useful. We may classify numerical solution methods into three classes:

1. *Direct methods*, which manipulates directly the coefficient matrix $\mathbf{A}$ and produces the exact solution $\mathbf{x}$;

2. *Stationary iterative methods*, which construct a sequence of increasingly better approximates $\{\mathbf{x}_0, \mathbf{x}_1, \ldots\}$ of the solution vector $\mathbf{x}$, such that each $\mathbf{x}_k$ does not depend on the previous approximates in the sequence. Jacobi, Power method and Gauss-Seidel are example of methods in this class.
3. *Non-stationary iterative methods*, which are similar to stationary iterative methods, but where each $\mathbf{x}_k$ depends on the sequence of approximates $\{\mathbf{x}_0, \ldots, \mathbf{x}_{k-1}\}$. *Krylov-subspace* methods belong to this class.

Other more advanced classes of solution methods could be added, like *multilevel* methods. We now describe in more details these three classes.

Direct methods.
A direct method works by factorizing the coefficient matrix $\mathbf{A}$ into a form that is easily invertible. Almost every direct method is a variation of the *Gaussian elimination*. These methods are usually very stable and robust, and have the advantage of knowing in advance the total amount of time and space needed to compute the solution of (4.5).

Unfortunately, Gaussian elimination has a space occupation that is $O(N^2)$ for a $N \times N$ matrix, which makes the method impractical for large matrices. Markov chain matrices may have very large state spaces, making these methods impractical.

Since Gaussian elimination works directly on the coefficients of $\mathbf{A}$, the matrix should be available. Therefore, preconditioning is rarely used, because it would require the explicit construction of the system $\mathbf{M}^{-1}\mathbf{A}$.

Stationary iterative methods.
Iterative methods work by constructing a sequence of vectors $\{\mathbf{x}_0, \mathbf{x}_1, \ldots\}$ which gradually approximates the solution $\mathbf{x}$. Usually, the iteration stops when the residual of the k-th vector in the sequence:

$$\mathbf{r}_k \;=\; \mathbf{A}\,\mathbf{x}_k - \mathbf{b} \tag{4.17}$$

has a magnitude that is less than a small ϵ. Iterative methods are the most common methods for large sparse systems, since they tend to require fewer storage then direct methods, in the order of $O(N)$ for a $N \times N$ matrix. Unfortunately, iterative methods are far less reliable than direct methods, since they are very dependent from the *conditioning* of the system matrix, which determines the number of iterations needed to achieve the accuracy ϵ. Preconditioning is usually used in iterative methods since they are easy to integrate and because they improve the convergence rate and accuracy.

One of the most important iterative methods is the *Richardson method*, whose non-preconditioned iterative relation is:

$$\mathbf{x}_{k+1} \;=\; \mathbf{x}_k + \omega(\mathbf{b} - \mathbf{A}\,\mathbf{x}_k) \tag{4.18}$$

with $\omega > 0$ the *relaxation factor*. The preconditioned Richardson method is:

$$\mathbf{y}_{k+1} \;=\; \mathbf{y}_k + \omega\mathbf{M}_L^{-1}(\mathbf{b} - \mathbf{A}\mathbf{M}_R^{-1}\,\mathbf{y}_k), \qquad \mathbf{x} = \mathbf{M}_R^{-1}\,\mathbf{y} \tag{4.19}$$

with and $\mathbf{M}_L$ and $\mathbf{M}_R$ the preconditioner matrices. The choice of a large enough ω value ensures the convergence of the sequence. It can be shown that, for

Markov chains, any value of ω in the range $(0, 1)$ can be used as relaxation coefficient. It is possible to show that an optimal ω value exists for each matrix $\mathbf{A}$; the computation of the optimum value is studied in [Rei66]. Equation (4.18) is also known as the *Power method.* The power method construct the sequence of $\mathbf{x}_k$ without any preconditioning.

The classical method of Jacobi, of Gauss-Seidel and SSOR are derived from the Richardson method using the appropriate preconditioner matrices (4.13), (4.14) or (4.15). Note that a specialized implementation of *Gauss-Seidel* (as in [Ste94]) produces the same sequence of the Richardson method, but is more efficient.

The pseudo-code of the Richardson method is given in Algorithm 4.1.

Algorithm 4.1 Preconditioned Richardson iterative method.

Given the linear system $\mathbf{A}\,\mathbf{x} = \mathbf{b}$, an initial guess $\mathbf{y}_0$ and the preconditioner $\mathbf{M} = \mathbf{M}_L\,\mathbf{M}_R$, do:

```
1. $\mathbf{y}_0 := $ an initial guess.
2. $\mathbf{x}_0 := \mathbf{M}_R^{-1}\,\mathbf{y}_0$                // solve: $\mathbf{M}_R\,\mathbf{x}_0 = \mathbf{y}_0$
3. $\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\,\mathbf{x}_0$
4. while $||\mathbf{r}_k|| > \epsilon$ do:
5.     $\mathbf{w}_k := \mathbf{M}_L^{-1}\,\mathbf{r}_k$            // solve: $\mathbf{M}_L\,\mathbf{w}_k = \mathbf{r}_k$
6.     $\mathbf{y}_{k+1} := \mathbf{y}_k + \omega\,\mathbf{w}_k$
7.     $\mathbf{x}_{k+1} := \mathbf{M}_R^{-1}\,\mathbf{y}_{k+1}$        // solve: $\mathbf{M}_R\,\mathbf{x}_{k+1} = \mathbf{y}_{k+1}$
8.     $\mathbf{r}_{k+1} := \mathbf{b} - \mathbf{A}\,\mathbf{x}_{k+1}$
9. return the last $\mathbf{x}_k$
```

Remark 1: The classical methods of Jacobi, Gauss-Seidel and SSOR are instances of the Richardson method with $\mathbf{M}$ defined as in (4.13), (4.14) and (4.15), respectively.

Algorithm 4.1 computes the sequence of $\mathbf{y}_k$ using the vectors $\mathbf{x}_k$, $\mathbf{r}_k$ and $\mathbf{w}_k$ to label the following parts of (4.19):

$$\mathbf{y}_{k+1} \;=\; \mathbf{y}_k + \omega\,\underbrace{\mathbf{M}_L^{-1}\,\overbrace{(\mathbf{b} - \mathbf{A}\,\underbrace{\mathbf{M}_R^{-1}\,\mathbf{y}_k}_{\mathbf{x}_k})}^{\mathbf{r}_k}}_{\mathbf{w}_k}, \qquad \mathbf{x}_{k+1} = \mathbf{M}_R^{-1}\,\mathbf{y}_{k+1}$$

The method proceeds until the magnitude of $\mathbf{r}_k$ is less than ϵ. Note that if the right preconditioner $\mathbf{M}_R$ is the identity matrix, then the sequence of $\mathbf{y}_k$ coincides with the sequence of $\mathbf{x}_k$. Analogously, the sequence of $\mathbf{w}_k$ coincides with the sequence of residuals $\mathbf{r}_k$ when the left preconditioner $\mathbf{M}_L$ is $\mathbf{I}$.

Non-stationary iterative methods (*Krylov-subspace* methods).
Non stationary iterative methods construct the solution $\mathbf{x}$ from the whole sequence of residual vectors $\{\mathbf{r}_0, \ldots, \mathbf{r}_m\}$, instead of using only the last residual. The *Generalized Minimum RESidual* (GMRES) method is one of these iterative methods. GMRES [SS86] was proposed by Saad and Schultz in 1986, and

works with large non-symmetric matrices. Other non-stationary methods for non-symmetric matrices exists, like BiCG-Stab [Vor92] or CGS [Son89], but are not a subject of this thesis.

A detailed description of the GMRES method is found in [SS86], and its rate of convergence is studied in [VV93]. In few words, the convergence rate of GMRES strictly depends on the spectrum of eigenvalues of the system matrix. The more the eigenvalues are clustered together and have a small norm, the more the system is well conditioned. On the opposite, a spectrum of eigenvalues that spreads over the perimeter of the unit disk characterizes an ill-conditioned system where GMRES will behave poorly. Thankfully, preconditioning is used to improve the clustering of eigenvalues around the zero.

GMRES supports left, right and split preconditioning. Unlike the Richardson method, if a preconditioner $\mathbf{M}$ is used, it cannot be changed during the iterative method. However, there exists a variant of GMRES, known as *Flexible GMRES* (FGMRES), that allows to change the right preconditioner at every iteration, with an additional memory cost. The variant of GMRES reported in this thesis is FGMRES with split preconditioning. Hereafter, with GMRES (or FGMRES) we mean the specific variant of the method reported in this thesis.

The pseudo-code of the preconditioned FGMRES(m) method is given in Algorithm 4.2. GMRES uses the *Arnoldi* method to constructs the *p–th* order *Krylov-subspace* $\mathcal{K}_p(\mathbf{A}, \mathbf{r}_0)$:

$$\mathcal{K}_p(\mathbf{A}, \mathbf{r}_0) \overset{\text{def}}{=} \left\{ \mathbf{r}_0,\, (\mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1})\mathbf{r}_0,\, \ldots,\, (\mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1})^{p-1}\mathbf{r}_0 \right\} \tag{4.20}$$

with $\mathbf{r}_0 = \mathbf{M}_L^{-1}(\mathbf{b} - \mathbf{A}\mathbf{M}_R^{-1}\mathbf{x}_0)$ and $p \leq m$. The parameter m of FGMRES(m) is the maximum number of dimensions of $\mathcal{K}_p(\mathbf{A}, \mathbf{r}_0)$. If the method does not reach the desired accuracy in m steps, it is restarted (line 17) using the last $\mathbf{x}_m$ as initial guess $\mathbf{x}_0$. The vector space $\mathcal{K}_p(\mathbf{A}, \mathbf{r}_0)$ is then orthonormalized using the *modified Gram-Schmidt* method to build a basis $\mathbf{V} = [\mathbf{v}_j]$ of unit vectors that are orthogonal to each other. The solution vector $\mathbf{x}_m$ is taken as a linear combination in the $\mathbf{V}$ basis as: $\mathbf{x}_m = \mathbf{V}\,\mathbf{q}$, where $\mathbf{q}$ is chosen to minimize the residual $\mathbf{r}_m$ of $\mathbf{x}_m$. In addition, the Arnoldi process produces an $(m+1) \times m$ matrix $\bar{\mathbf{H}}_m$ that has the property: $\mathbf{A}\mathbf{V}_m = \mathbf{V}_{m+1}\bar{\mathbf{H}}_m$. Because $\mathbf{V}_m$ is orthogonal, it is possible to derive the relation:

$$\|\mathbf{b} - \mathbf{A}\,\mathbf{x}_m\|_2 \;=\; \left\|\beta\mathbf{e}_1 - \bar{\mathbf{H}}_m\mathbf{q}\right\|_2$$

where $\mathbf{e}_1 = \langle 1, 0, 0, 0, \ldots, 0\rangle$ and β is the norm of the first residual $\mathbf{r}_0$. Finding a vector $\mathbf{x}_m$ in the vector space $\mathbf{V}_m$ that minimizes the above norm is an instance of the *linear least square* problem of size m. In the FGMRES variant, the basis is $\mathbf{Z}_m = \mathbf{M}_R^{-1}\mathbf{V}_m$ instead of $\mathbf{V}_m$, because each $\mathbf{z}_j$ vector may have been multiplied with a different $\mathbf{M}_R$ preconditioner at step 6.

The FGMRES method works as follows:

- Lines 1–4: Initialization.
- Lines 5–13: One step of the *Arnoldi* method that adds a new dimension to the basis $\mathbf{Z}_m$ (or $\mathbf{V}_m$ for basic GMRES).
- Lines 14–16: Compute the *linear least square* problem to find $\mathbf{x}_m$ as the best linear combination of $\mathbf{Z}_m$ (or $\mathbf{V}_m$) that minimizes the residual $\mathbf{r}_m$.
- Line 17: Restart if $\mathbf{x}_m$ is not sufficiently accurate.

Algorithm 4.2 Preconditioned FGMRES(m) iterative method

Given the linear system $\mathbf{A}\,\mathbf{x} = \mathbf{b}$, an initial guess $\mathbf{y}_0$, the dimension of the Krylov subspace m and the preconditioner $\mathbf{M} = \mathbf{M}_L\,\mathbf{M}_R$, do:

```
 1. $\bar{\mathbf{H}}_m := $ empty matrix of size $(m+1) \times m$
 2. $\mathbf{r}_0 := \mathbf{M}_L^{-1}(\mathbf{b} - \mathbf{A}\,\mathbf{x}_0)$
 3. $\beta = \|\mathbf{r}_0\|_2$
 4. $\mathbf{v}_1 := \mathbf{r}_0/\beta$
 5. for $j = 1 \dots m$ do:
 6.     $\mathbf{z}_j := \mathbf{M}_R^{-1}\,\mathbf{v}_j$
 7.     $\mathbf{w} := \mathbf{M}_L^{-1}(\mathbf{A}\,\mathbf{z}_j)$
 8.     for $i = 1 \dots j$ do:
 9.         $\bar{\mathbf{H}}_m(i,j) := \mathbf{w} \cdot \mathbf{v}_i$
10.         $\mathbf{w} := \mathbf{w} - \bar{\mathbf{H}}_m(i,j)\,\mathbf{v}_i$
11.     $\bar{\mathbf{H}}_m(j{+}1,j) := \|\mathbf{w}\|_2$
12.     if $\bar{\mathbf{H}}_m(j{+}1,j) = 0$ then $m = j$ and goto step 14
13.     $\mathbf{v}_{j+1} := \mathbf{w}/\bar{\mathbf{H}}_m(j{+}1,j)$
14. $\mathbf{Z}_m := $ matrix of $[\mathbf{z}_1, \dots, \mathbf{z}_m]$
15. Find $\mathbf{y}_m := \mathsf{argmin}\,\left\|\beta\mathbf{e}_1 - \bar{\mathbf{H}}_m\mathbf{q}\right\|_2$
16. $\mathbf{x}_m := \mathbf{x}_0 + \mathbf{Z}_m\,\mathbf{y}_m$
17. Return if accuracy is $< \epsilon$, otherwise restart with $\mathbf{x}_0 = \mathbf{x}_m$
```

Remark 1: GMRES is obtained by changing these lines:

```
14. $\mathbf{V}_m := $ matrix of $[\mathbf{v}_1, \dots, \mathbf{v}_m]$
16. $\mathbf{x}_m := \mathbf{x}_0 + \mathbf{M}_R^{-1}(\mathbf{V}_m\,\mathbf{y}_m)$
```

Remark 2: left preconditioning minimizes the residual norm: $\left\|\mathbf{M}_L^{-1}\mathbf{b} - \mathbf{A}\,\mathbf{x}\right\|_2$, while right preconditioning minimizes the original norm: $\|\mathbf{b} - \mathbf{A}\,\mathbf{x}\|_2$.

Remark 3: the right preconditioner matrix $\mathbf{M}_R$ of step 6 can change at any iteration, but this requires to store the $\mathbf{z}_j$ vectors to form the matrix $\mathbf{Z}_m$. If $\mathbf{M}_R$ does not change, then $\mathbf{Z}_m$ is not needed at step 16, and $\mathbf{x}_m$ can be computed with: $\mathbf{x}_m := \mathbf{x}_0 + \mathbf{M}_R^{-1}(\mathbf{V}_m\,\mathbf{y}_m)$.

Every iteration requires one matrix product with $\mathbf{A}$ (line 2 and 7) plus the products with the preconditioners (line 6 and 7), plus an additional product with the left preconditioner at the beginning. The space occupation of FGMRES is $O(m^2 + N)$. Usually the parameter m is a small number, in the order of tens.

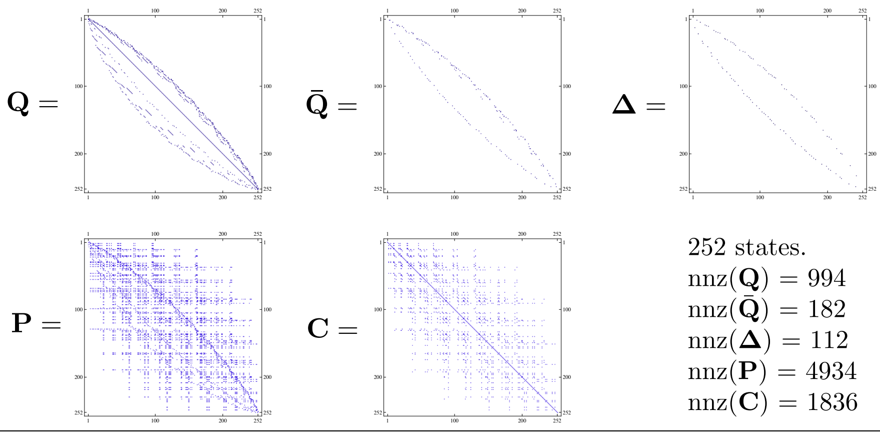
The ability of having a variable right preconditioner will be used in section 4.3, where a special class of preconditioners for MRP processes is described.

4.2.3 Matrix-free solution of MRPs

In section 4.2 the problem of computing the forward/backward steady-state distribution of both CTMCs and DTMCs has been reduced to the solution of a linear equation system. In theory, the same technique can be applied to the solution of the EMC matrix (2.45) of an MRP, since the EMC is a DTMC.

This powerful reuse of concepts hides, however, a severe problem. The formula (2.45) of $\mathbf{P}$ contains two matrix exponentials (2.40) and (2.41) for $\mathbf{\Omega}$ and $\mathbf{\Psi}$, respectively. A matrix exponential breaks the sparsity of its parameter: therefore, even if $\mathbf{Q}$, $\bar{\mathbf{Q}}$, $\mathbf{\Delta}$ are sparse (space occupation $O(N)$), the resulting EMC matrix will be almost dense (space occupation $O(N^2)$). Indeed the i-th row of $\mathbf{P}$ contains a non-zero entry for every state that is reachable from i following paths of exponential transitions; following these paths is like computing the transitive closure of the graph of the matrix that describes the exponential transitions. A similar situation holds for $\mathbf{C}$, since its formula (2.46) contains $\mathbf{\Psi}$. For large MRPs, the explicit construction of the EMC matrix may become unfeasible due to the large space requirement and the long time needed to compute the exponentials. The problem is illustrated in Figure 4.3, which shows the non-zero plots of the matrices of a small MRP. Figure 4.3 illustrates that the sparse generator matrixes may give rise to a dense EMC $\mathbf{P}$ and a dense conversion matrix $\mathbf{C}$.

Figure 4.3 Sample non-zero plots of $\mathbf{Q}$, $\bar{\mathbf{Q}}$, $\mathbf{\Delta}$ and of the resulting $\mathbf{P}$ and $\mathbf{C}$.



252 states.
$\text{nnz}(\mathbf{Q}) = 994$
$\text{nnz}(\bar{\mathbf{Q}}) = 182$
$\text{nnz}(\mathbf{\Delta}) = 112$
$\text{nnz}(\mathbf{P}) = 4934$
$\text{nnz}(\mathbf{C}) = 1836$

This is known as the *fill-in* problem of MRPs. Based on this problem, the MRP solution strategies present in the literature can be classified as:

- *Explicit methods:* the $\mathbf{P}$ and $\mathbf{C}$ matrices are computed once using the corresponding formulas, and stored in memory, as in [Lin98] and [AC87].
- *Implicit methods:* (also called *matrix-free*) the $\mathbf{P}$ and $\mathbf{C}$ matrices are never computed and stored in memory. In this thesis we use the terms *implicit* and *matrix-free* interchangeably.

An implicit method has been proposed in 2001 in [Ger01] (called "two-level iterative" by the author), that is based on the idea that, given a row vector $\mathbf{u}$, the $\mathbf{uP}$ and $\mathbf{uC}$ product operators can be rewritten in a way that does not require the computation and storage of $\mathbf{P}$ and $\mathbf{C}$. This leads to a *matrix-free* method that avoids the *fill-in*, with a space occupation that scales almost linearly with the model size and the sparsity of the generators. Note that the method is matrix-free for $\mathbf{P}$ and $\mathbf{C}$, but not for the three generator matrices, which are kept in memory.

Definition 31 (Implicit products)**.** Let $\mathbf{u}$ be a generic vector over $\mathcal{S}$. The product of $\mathbf{u}$ with $\mathbf{P}$ can be rewritten by expanding the Eq. (2.45), resulting in:

$$\begin{aligned}\mathbf{uP} &= \mathbf{u}\Big(\mathbf{I}^{\mathrm{E}} - \mathsf{diag}^{-1}(\mathbf{Q}^{\mathrm{E}})\mathbf{Q}^{\mathrm{E}} + \mathbf{\Omega\Delta} + \mathbf{\Psi\bar{Q}}\Big) = \\ &= \mathbf{u}\Big(\mathbf{I}^{\mathrm{E}} - \mathsf{diag}^{-1}(\mathbf{Q}^{\mathrm{E}})\mathbf{Q}^{\mathrm{E}}\Big) + \big(\mathbf{u\Omega}\big)\mathbf{\Delta} + \big(\mathbf{u\Psi}\big)\mathbf{\bar{Q}}\end{aligned} \tag{4.21}$$

The $\mathbf{u\Omega}$ and $\mathbf{u\Psi}$ parts of Eq. (4.21) are vector by matrix products that can be written by expanding the vector product into the Taylor series expansions of (2.40) and (2.41), resulting in:

$$\mathbf{u\Omega} = \sum_{g\in\mathrm{G}} \mathbf{u\Omega}^g, \qquad \mathbf{u\Psi} = \sum_{g\in\mathrm{G}} \mathbf{u\Psi}^g \tag{4.22}$$

where the $\mathbf{u\Omega}^g$ and $\mathbf{u\Psi}^g$ vector×matrix products are:

$$\mathbf{u\Omega}^g = \mathbf{uI}^g \mathrm{e}^{\mathbf{Q}^g\delta_g} = \sum_{k=0}^{R} \mathbf{uI}^g\Big(\mathbf{U}^g\Big)^k \beta(k, q_g\delta_g) \tag{4.23}$$

$$\mathbf{u\Psi}^g = \mathbf{uI}^g \int_0^{\delta_g} \mathrm{e}^{\mathbf{Q}^g t}\,\mathrm{d}t = \sum_{k=0}^{R} \mathbf{uI}^g\Big(\mathbf{U}^g\Big)^k \hat{\beta}(n, q_g\delta_g) \tag{4.24}$$

with $\mathbf{U}^g$, q, $\beta(k, qt)$ and $\hat{\beta}(k, qt)$ defined as in Eq. (4.3) for matrix $\mathbf{Q}^g$.
The product of $\mathbf{u}$ with $\mathbf{C}$ can be derived analogously from (2.46) as:

$$\begin{aligned}\mathbf{uC} &= \mathbf{u}\Big(-\mathsf{diag}^{-1}(\mathbf{Q}^{\mathrm{E}}) + \mathbf{\Psi}\Big) = \\ &= -\mathbf{u}\cdot\mathsf{diag}^{-1}(\mathbf{Q}^{\mathrm{E}}) + \mathbf{u\Psi}\end{aligned} \tag{4.25}$$

where the $\mathbf{u\Psi}$ can be expanded as in (4.24).
The transposed products $\mathbf{Pu}$ and $\mathbf{Cu}$ are derived analogously.

The asymptotical time cost of a vector×matrix product $\mathbf{uP}$ (or the transposed $\mathbf{Pu}$) is:

$$O\bigg(\eta(\mathbf{Q}^{\mathrm{E}}) + \sum_{g\in\mathrm{G}}\Big(\underbrace{\eta(\mathbf{Q}^g)\cdot R^g}_{\text{cost of } \mathbf{u\Omega} \text{ and } \mathbf{u\Psi}} + \underbrace{\eta(\mathbf{\Delta}^g) + \eta(\bar{\mathbf{Q}}^g)}_{\text{products with } \mathbf{\Delta} \text{ and } \bar{\mathbf{Q}}}\Big)\bigg) \tag{4.26}$$

which is dominated by the cost of the products $\mathbf{u\Omega}$ and $\mathbf{u\Psi}$. The space occupation is that of the uniformisation procedure, which is in the order of $O(N)$.

The advantage of the implicit method over the explicit one is that $\mathbf{P}$ does not need to be stored, but only the vectors resulting from the products with $\mathbf{Q}$, $\bar{\mathbf{Q}}$ and $\mathbf{\Delta}$. From a computational point of view, the computation of the product with $\mathbf{P}$, for each iteration of the chosen fixed-point solution method, requires at least a uniformisation step (for $\mathbf{u\Omega}$ and $\mathbf{u\Psi}$), plus few additional products with the (sparse) matrices $\mathbf{Q}$, $\bar{\mathbf{Q}}$ and $\mathbf{\Delta}$.

Similarly, the product with $\mathbf{C}$ has a cost:

$$O\bigg(\eta(\mathbf{Q}^{\mathrm{E}}) + \sum_{g\in\mathrm{G}} \underbrace{\eta(\mathbf{Q}^g)\cdot R^g}_{\text{cost of } \mathbf{u\Psi}}\bigg) \tag{4.27}$$

that is dominated by the computation of $\mathbf{u}\boldsymbol{\Psi}$. These costs should be compared with the much higher costs of building $\mathbf{P}$ and $\mathbf{C}$, which requires the cost of computing and storing two matrix exponentials.

In addition to consuming less memory, the implicit method may also be faster, depending on the ratio between the number of iterations required to reach the fixed point and the number of regenerative states. The work in [Ger01] has studied the advantage of the implicit method over the explicit one, in terms of both memory and time.

The implicit method can be used to compute both the transient and the steady-state solutions of the EMC, since equations (2.3), (2.6), (4.6), (4.6), (2.49) and (2.50) only require vector$\times$matrix products with $\mathbf{P}$ or $\mathbf{C}$.

4.3 Preconditioning MRP solutions

The steady-state solution of the embedded DTMC $\mathbf{P}$ can be computed with the implicit method (Def. 31) with both the Richardson iterative method and a Krylov-subspace method, since both require only the ability of computing residual vectors with (4.17). However, an important limitation of the implicit strategy is that the single entries of $\mathbf{P}$ and $\mathbf{C}$ are not available. Unfortunately, almost any preconditioner requires some algebraic manipulation of the system matrix $\mathbf{P}$, which is not available.

We now present a problem-specific preconditioner for MRP that is sparse and does not require the explicit knowledge of $\mathbf{P}$. This preconditioner has been introduced in [AD11] and is a variant of ILUT, described before at page 62.

The idea is to build the ILU preconditioner from a sparse estimate $\widetilde{\mathbf{A}}$ of $\mathbf{A}$, which in turn is derived from a sparse approximate $\widetilde{\mathbf{P}}$ of the EMC $\mathbf{P}$, chosen so to be fast to compute and sparse. This technique is based on the assumption that a preconditioner $\mathbf{M}^{-1}$ built from a sparse approximate matrix $\widetilde{\mathbf{A}}$ is still an effective preconditioner for the original matrix $\mathbf{A}$. We provide an empirical evaluation of this assumption in section 4.3.3. In what follows we consider the problem of generating $\widetilde{\mathbf{P}}$ and $\widetilde{\mathbf{A}}$ for a given MRP, and then we describe two ways of building the preconditioner $\mathbf{M}$ from $\widetilde{\mathbf{A}}$.

4.3.1 Construction of the approximate $\widetilde{\mathbf{P}}$ of the EMC $\mathbf{P}$.

Let drop be a *matrix function* that modifies a matrix by deleting certain entries according to a specified *dropping criteria*.

We consider two methods for building $\widetilde{\mathbf{P}}$:

1. Compute each row of the EMC $\mathbf{P}$ explicitly with equations (2.45), and then erase some entries to build an approximate $\widetilde{\mathbf{P}} = \mathsf{drop}(\mathbf{P})$.
2. Generate an approximate $\widetilde{\mathbf{P}}_{\text{sim}}$ of $\mathbf{P}$ through a row-by-row simulation: for each state i, the runs are performed, starting from i itself. A run ends when a regenerative state j is reached. The mean frequency of each reached state j gives the entries of $\widetilde{\mathbf{P}}_{\text{sim}}$. $\widetilde{\mathbf{P}}$ is then constructed as $\mathsf{drop}(\widetilde{\mathbf{P}}_{\text{sim}})$, to make $\widetilde{\mathbf{P}}$ a sparse matrix according to the dropping criteria.

In both cases, space occupation is not so problematic since eliminations can take place as soon as a row is computed. The first method is very slow to compute, for the reasons explained in section 4.2.3.

As dropping criteria, we use the same *dual threshold strategy* of ILUT(σ, ϵ): drop is defined so as to keep at most the σ largest entries of each row, while erasing all entries below a given tolerance value ϵ. Note that, due to eliminations, $\widetilde{\mathbf{P}}$ is substochastic: each row sum is in the $[0, 1]$ range.

In both methods, an approximate row i of $\mathbf{P}^{\mathrm{E}}$ is built directly with (2.42), since it is already a sparse, explicitly available matrix. In the simulation method, an approximate row i of $\mathbf{P}^{\mathrm{G}}$ is built by taking the relative frequencies of each regenerative state j reachable from i with some random simulations. The simulated matrix is the *subordinated Markov chain* of i. A pseudo-code of the simulation procedure for a given general state $i \in \mathcal{S}^{\mathrm{G}}$ is given in Algorithm 4.3:

Algorithm 4.3 Simulation of a path from state $i \in \mathcal{S}^g$

```
 1. $i \leftarrow$ initial state of the simulation
 2. $t \leftarrow 0$      // elapsed time counter
 3. $ft \leftarrow$ generate the firing time from $F_g$ randomly
 4. loop:
 5.   $t \leftarrow t +$ (random exp. duration with rate $-q_{i,i}$)
 6.   if $t \geq ft$ :
 7.       $i \leftarrow$ choose next state randomly from $\mathbf{\Delta}_i$
 8.       return $i$      // firing of $g$
 9.   $i \leftarrow$ choose next state randomly from $(\mathbf{Q}_i + \bar{\mathbf{Q}}_i)$
10.   if $i$ has been chosen from $\bar{\mathbf{Q}}_i$ :
11.       return $i$      // preemption of $g$
```

Algorithm 4.3 simulates a run inside the subordinated chain of $\mathbf{P}^{\mathrm{G}}$, starting from state i. First, the firing time of the general transition $g \in \mathrm{G}$ is randomly selected at line 2, according to the cumulative distribution function F_g of the transition g. Then the simulation "explores" the Markov chain until either the general transition fires (lines 6–8), or a preemptive exponential transition fires (lines 10–11). The simulation is repeated s_i times for each initial state $i \in \mathcal{S}^{\mathrm{G}}$, where s_i can either be a fixed constant or dynamically chosen according to a stopping criteria based on accuracy and confidence level. Let s_i be the total number of simulations run for state i, and $f_{i,j}$ be the number of times a simulation has hit state j starting from i. An approximate entry $\widetilde{\mathbf{P}}_{\mathrm{sim}}(i, j)$ is given by:

$$\widetilde{\mathbf{P}}_{\mathrm{sim}}(i,j) = \begin{cases} p^{\mathrm{E}}_{i,j} & i \in \mathcal{S}^{\mathrm{E}} \\ \dfrac{f_{i,j}}{s_i} & i \in \mathcal{S}^{\mathrm{G}} \end{cases} \tag{4.28}$$

The sparse approximate $\widetilde{\mathbf{P}}$ is derived as drop$(\widetilde{\mathbf{P}}_{\mathrm{sim}})$. The estimated matrix $\widetilde{\mathbf{A}}$ of $\mathbf{A}$ is then $(\widetilde{\mathbf{P}} - \mathbf{I})^T$. Observe that rows of $\widetilde{\mathbf{A}}$ may not sum up to zero, since $\widetilde{\mathbf{P}}$ is substochastic. If there is at least a row of $\widetilde{\mathbf{P}}$ that does not sum to 1, then $\widetilde{\mathbf{A}}$ is non-singular.

4.3.2 Preconditioning strategies

Once the approximate matrix $\widetilde{\mathbf{A}}$ has been constructed, a preconditioner $\mathbf{M}^{-1}$ can be derived. We consider two different approaches for building the preconditioner from $\widetilde{\mathbf{A}}$. In both approaches $\mathbf{M}$ is constructed as a non-singular sparse irreducible matrix. Its inverse $\mathbf{M}^{-1}$ is never built up explicitly, since it is known that inverses of sparse matrices do not preserve the sparsity. The effectiveness of the proposed preconditioning method (balance between the additional computational cost of building the preconditioner and its effectiveness) is assessed experimentally in section 4.3.3.

The two approaches are:

[Single preconditioning]: $\mathbf{M}_s$ is taken as the ILU factorization of $\widetilde{\mathbf{A}}$, so that $\widetilde{\mathbf{A}} = \widetilde{\mathbf{L}}\widetilde{\mathbf{U}} - \widetilde{\mathbf{E}}_s$ and $\mathbf{M}_s = \text{ILU}(\widetilde{\mathbf{A}}) = \widetilde{\mathbf{L}}\widetilde{\mathbf{U}}$. An evaluation of the preconditioner $\mathbf{M}_\text{s}^{-1}\mathbf{y}$ requires only a forward and a backward substitution, which have a linear time cost.

[Inner-outer preconditioning]: This is a two-level preconditioning approach. The preconditioner is chosen exactly as $\mathbf{M}_\text{out} = \widetilde{\mathbf{A}}$. Observe that $\mathbf{M}_\text{out}$ is not easily invertible. Therefore, each evaluation of the preconditioner $\mathbf{x} = \mathbf{M}_\text{out}^{-1}\mathbf{y}$ is computed as the solution of the linear equation system $\mathbf{M}_\text{out}\mathbf{x} = \mathbf{y}$. In practice, the preconditioner is an iterative solution of $\widetilde{\mathbf{A}}$, repeated for every outer iteration. This solution can also be preconditioned using a second *internal* preconditioner $\mathbf{M}_\text{in}$, built as the ILU factorization of $\mathbf{M}_\text{out}$.

Hereafter we use $\mathbf{M}_s$ and $\mathbf{M}_\text{out}$ when writing about the two specific strategies, and we use $\mathbf{M}$ to refer to the general preconditioning method that applies to both the single and inner-outer preconditioning.

In the inner-outer preconditioner case, the inner solution of $\mathbf{M}_\text{out}$ preconditioned with $\mathbf{M}_\text{in}^{-1}$ follows this schema:

$$\mathbf{r}_i = \mathbf{b} - \mathbf{M}_\text{out}(\mathbf{M}_\text{in}^{-1}\mathbf{v}_i) \quad \Rightarrow \quad \mathbf{r}_i = \mathbf{b} - \mathbf{M}_\text{out}\mathbf{z}_i, \quad \mathbf{z}_i = \mathbf{M}_\text{in}^{-1}\mathbf{v}_i \tag{4.29}$$

and since $\mathbf{M}_\text{in} = \text{ILU}(\mathbf{M}_\text{out})$, then the vector $\mathbf{z}_i$ is computed from $\mathbf{M}_\text{in}^{-1}\mathbf{v}_i$ with a forward and a backward substitution.

The inner iteration can be carried out with any algorithm, like GMRES. The inner method could also be less accurate than the outer method [ESG05], and stop when a fixed tolerance is reached or after a fixed number of inner iterations. Since the outer preconditioner is the result of an iterative solution, the product with $\mathbf{M}_\text{out}^{-1}$ is not a constant operation, a condition which is required for preconditioned GMRES to converge. Therefore, the outer iteration has to be carried out with a *flexible* method, like FGMRES (given in Algorithm 4.2). Remember that flexible methods allow the outer preconditioner to vary at every iteration. The two proposed preconditioners have both a space occupation of $O(pN)$, which is linear in the model size. Inner-outer preconditioning strategy uses approximately twice the memory as the single preconditioning strategy, due to the memory occupation of FGMRES and the need of two matrices $\mathbf{M}_\text{out}$ and $\mathbf{M}_\text{in}$.

In the literature a relatively similar idea of using matrix estimation to build a preconditioner for matrix-free problems has been pursued in [CT04]. In that article, however, the estimation is based on the assumption that the graph structure of **A** is known a priori and the estimated matrix is computed with numerous matrix-free products. We have not followed this approach since the graph structure is not known and, more importantly, matrix-free products for MRPs can be very time-consuming.

4.3.3 Experimental assessment of the MRP preconditioner

We summarize the results in [AD11] of the efficiency of the implicit MRP solution with the problem-specific preconditioner of section 4.3. The results are computed with DSPN-Tool, described in section 5.2. The implementation supports deterministic and exponential distributions for MRP transitions.

The experiments are organized into three sets. The first set shows the results of 6 benchmark models. The second set shows some additional insights on the behavior of the preconditioners w.r.t. an ill-conditioned (stiff) problem. The last set investigates the behavior of the methods for larger models.

Table 4.1 shows the results with six benchmark DSPN models: a Kanban system with 4 cards, a *Simultaneous Generalized Kanban Control System* (SGKCS, see [CD97]) with 15 cards, a CSMA/CD ethernet network with 120 connected clients (see [Ger00, pag. 327]), a Polling server with 12 clients (see [AD10d, sec. 4.1]), a stiff MRP of a semaphore model with n=4 jobs running in 2 cpus, and a *Gambler's ruin*(GR) model with 22 players, whose MRP is non-ergodic with 44 recurrence classes. All the model files are available in GreatSPN format at `http://www.di.unito.it/~amparore/NSMC/TestModels.zip`, for the interested readers, together with the solver sources. Solutions are computed on a 2.3Ghz x86/64 machine.

Table 4.1 Results of 6 MRP benchmarks.

			Kanban		**SGKCS**		**CSMA/CD**		**Polling**		**Semaphore**		**GR**	
	Petri Net params		n = 4		n = 15		n = 120		n = 12		n = 4		n = 22	
	States		12 985		38 976		14 641		106 496		252		63 250	
	Transitions		86 220		174 755		43 320		798 720		1 036		394 680	
	Regenerative states		9 859		34 880		7 382		106496		252		63 250	
	P entries		12 051 964		88 663 327		422 943		7 334 717		4 918		254 653 278	
	C entries		12 021 959		88 652 528		9 374 110		6 961 981		1 836		127 288 183	
	P + **C** build time		1 538s		2 491s		24 879s		2 445s		0.07s		10 113s	
	$\mathbf{P}^{\mathrm{sim}}$ build time		6.72s		3.69s		9.94s		3.19s		0.02s		8.52s	
	Recurrence classes		irreducible		irreducible		irreducible		irreducible		irreducible		44	
			xP	time	**xP**	time	**xP**	time	**xP**	time	**xP**	time	**xP**	time
①	Power method	ω=0.90	528	187.3	878	126.7	106	1 070.0	95	59.0	160 488	165.8	195	174.6
		ω=0.95	500	190.8	832	120.1	223	2 461.5	89	56.5	152 041	156.0	184	165.4
		ω=0.99	614	247.2	798	115.2	1155	7 873.2	85	53.9	145 898	150.3	175	193.5
②	GMRES(30)		122	48.6	306	40.9	21	223.5	53	35.2	762	0.82	286	244.2
	Bi-CG Stabilized		91	29.6	127	12.2	33	342.6	55	35.6	251	0.33	103	118.1
	CGS		106	33.9	132	12.6	28	277.6	-	-	180	0.25	106	95.0
③	$\mathbf{M}_s$=ILUT($\mathbf{A}$)		20	1 830	48	5 582	10	251 305	31	7 848	27	0.13	23	149 225
④	$\mathbf{M}_s$=ILUT($\tilde{\mathbf{A}}$)	σ=4	20	14.5	53	13.6	11	130.4	29	25.5	26	0.06	24	39.9
		σ=7	17	14.1	45	12.6	11	131.6	25	23.9	23	0.06	21	32.8
		σ=10	14	14.8	34	10.9	11	126.4	22	23.1	20	0.06	20	33.9
⑤	$\mathbf{M}_{\mathrm{out}}$=$\tilde{\mathbf{A}}$ $\mathbf{M}_{\mathrm{in}}$=ILUT($\tilde{\mathbf{A}}$)	σ=4	11 (17)	12.5	50 (7)	13.8	10 (7)	120.5	19 (15)	31.3	19 (15)	0.07	21 (16)	36.3
		σ=7	11 (14)	11.4	44 (7)	18.9	9 (7)	110.6	15 (16)	29.7	17 (14)	0.07	19 (13)	36.0
		σ=10	11 (12)	11.5	31 (6)	15.5	9 (7)	107.8	13 (16)	24.7	16 (12)	0.06	18 (11)	37.5

The first lines of Table 4.1 report the model parameters, the number of tangible states and transitions of the MRP, the number of regenerative states, the number of entries in the embedded DTMC **P** and in the conversion factors matrix $\mathbf{C}$, the total time needed to build $\mathbf{P}$ and $\mathbf{C}$ explicitly, the time needed to build the approximate matrix $\widetilde{\mathbf{P}}^{\text{sim}}$ through simulations (with s=200 simulations per state), and the recurrence structure. The problem of fill-in is evident from these examples: the number of entries in $\mathbf{P}$ and $\mathbf{C}$ grows almost quadratically in many cases, requiring a large storage space. For instance, in the GR model, the storage space of $\mathbf{P}$ and $\mathbf{C}$ is almost a thousand times bigger than the generator matrices. The time required to build the simulated EMC $\mathbf{P}^{\text{sim}}$ for the given value of s is small, compared to the time needed to build $\mathbf{P}$ and $\mathbf{C}$. Note that the matrices of Figure 4.3 are the one of the *Semaphore* model.

Remaining lines of Table 4.1 list the performance of iterative algorithms, expressed as the number of implicit $\mathbf{xP}$ products and the total solution time. A minus sign is used when the method does not converge. Accuracy is set to 10^{-10}. For the construction of $\widetilde{\mathbf{A}}$ the drop tolerance is $\epsilon = 10^{-4}$, and the maximum number of elements per row is σ (chosen between 4, 7 and 10). The ILU method used for the experiments is ILUT [Ste94; CV94] with drop threshold 10^{-4} and a maximum of 4 entries per row.

Five combinations of methods and preconditioners have been evaluated:

① Unpreconditioned Richardson method (i.e. Power method) with three under-relaxation coefficients ω, as described in [Ger01, sec. 3].

② Unpreconditioned Krylov methods: GMRES with m=30, Bi-CG Stabilized and CGS.

③ Single-preconditioned GMRES with the ILUT factorization of the *exact* dense matrix $\mathbf{A}$. It is shown only for comparative purposes.

④ Single-preconditioned GMRES with the ILUT factorization of the sparse approximate matrix $\widetilde{\mathbf{A}}$, for three values of σ (maximum number of entries per row in $\widetilde{\mathbf{A}}$).

⑤ FGMRES with inner-outer preconditioner $\mathbf{M}_{\text{out}}$, for three values of σ. Table 4.1 reports the iteration count as "$i_1(i_2)$", where i_1 is the number of outer FGMRES iterations, and i_2 is the mean number of inner GMRES iterations done to solve the system of equations (4.29) in $\mathbf{M}_{\text{out}}$ preconditioned with $\mathbf{M}_{\text{in}} = \text{ILUT}(\mathbf{M}_{\text{out}})$. Inner accuracy is set to 10^{-6}.

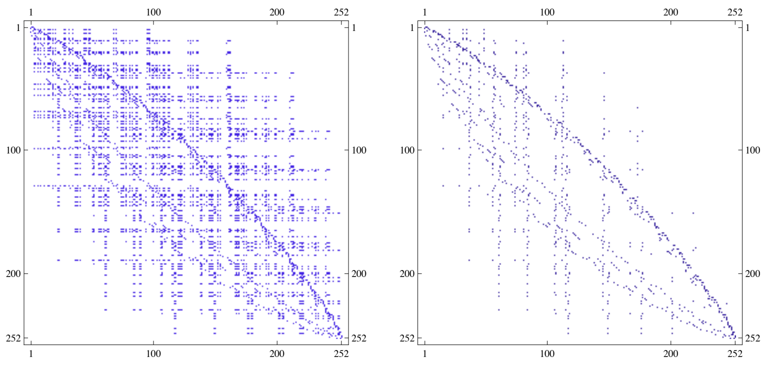
The worst case in time is always ③. Indeed this was expected as the exact matrix $\mathbf{A}$ (dense and slow to compute) is used for the ILUT preconditioner $\mathbf{M}_{\text{s}}$. This case is reported as a reference for cases ④ and ⑤.

Apart from ③, *power* method is the most expensive in terms of both $\mathbf{uP}$ products and solution times. The methods in ② show the advantage of using implicit Krylov methods instead of power method, with the exception of CGS over the polling model which does not converge. Preconditioned methods (③, ④ and ⑤) consistently reduce the number of iterations w.r.t. the unpreconditioned ones (①, ②). Among preconditioned methods, we cannot observe significant differences in the iteration counts.

The impact of the preconditioner clearly depends on the model. Comparing the two methods based on matrix estimation (④ and ⑤), we observe that inner-outer preconditioning with $\widetilde{\mathbf{A}}$ (⑤) is the most effective in reducing the iteration count, and it is slightly better than the *single* preconditioning (④) with ILUT factorization. However, this does not always result in a speed up, due to the additional cost of solving $\mathbf{M}_{\text{out}}$. The difference between single and inner-outer preconditioning could be due to the fact that ILUT adds another approximation to an already approximated matrix $\widetilde{\mathbf{A}}$, as observed in the semaphore model.

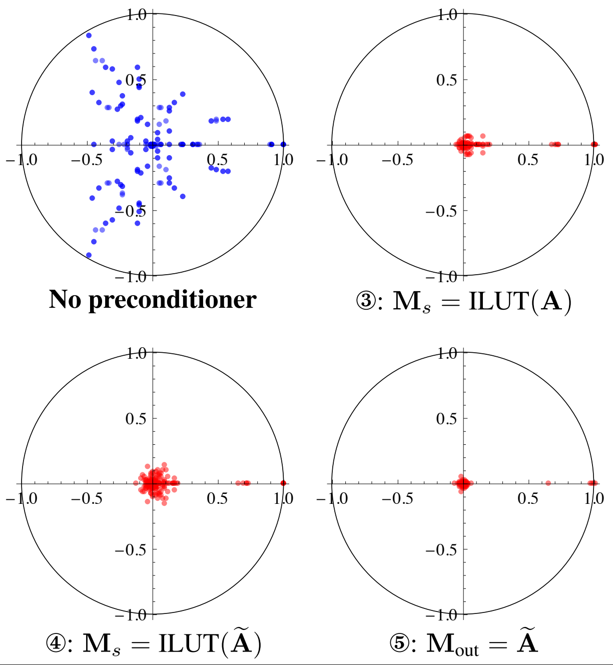
Figure 4.4 shows the matrix plots of $\mathbf{A}$ and of $\widetilde{\mathbf{A}}$ with $\sigma = 7$ of the semaphore model. The reader can observe how the sparse approximate matrix $\widetilde{\mathbf{A}}$ generated from $\widetilde{\mathbf{P}}^{\text{sim}}$ still "resembles" the dense matrix $\mathbf{A}$ (at least for what concerns the most significant entries). This is not surprising, since it is the expected behavior of simulations.

Figure 4.4 Matrix plots of $\mathbf{A}$ (left) and $\widetilde{\mathbf{A}}$ (right), with $\sigma = 7$.



The effect of the preconditioner is more evident when viewing the spectra of the matrix before and after the preconditioning. Figure 4.5 shows the eigenvalue distribution of $\mathbf{I} + \mathbf{A}$, the spectra of $\mathbf{I} + \mathbf{M}^{-1}\mathbf{A}$ for the ILUT factorization of $\mathbf{A}$, the spectra of $\mathbf{I} + \mathbf{M}_s^{-1}\mathbf{A}$ for the single preconditioner $\mathbf{M}_s = \text{ILUT}(\widetilde{\mathbf{A}})$, and the spectra of $\mathbf{I} + \mathbf{M}_{\text{out}}^{-1}\mathbf{A}$ for the inner-outer preconditioner $\mathbf{M}_{\text{out}} = \widetilde{\mathbf{A}}$.

Figure 4.5 Spectra of the "semaphore" model.



Even if the model is small (252 states), it requires an unreasonably large number of power method iterations to achieve convergence. The spectral analysis of $\mathbf{I}+\mathbf{A}$ reveals that almost one fifth of the eigenvalues are clustered around 0, while the others are spread away. The subdominant eigenvalue λ_2 is 0.999926, which explains the extremely slow convergence rate of the power method, since this rate is approximatively $\left|\frac{\lambda_2}{\lambda_1}\right|$. As expected, preconditioning clusters together the eigenvalues of $(\mathbf{I}+\mathbf{M}^{-1}\mathbf{A})$ around zero [Ste94, pag. 150]. However, since various approximations take place (eliminations of some estimated values of $\widetilde{\mathbf{A}}$ due to drop, eliminations in the ILUT factorization), the quality of the resulting preconditioner may vary. In the semaphore model inner-outer preconditioner produces a more compact cluster than the ILUT factorization, which could explain the slightly better behavior of GMRES. In practice, fast convergence of GMRES is often observed in the case of clustered spectra.

The last set of experiments is reported in Table 4.2. It was defined to investigates the preconditioner effect on the six DSPN models for increasing state space sizes. For this test we consider methods ②, ④ and ⑤. The approximated $\widetilde{\mathbf{A}}$ is built with σ=10 maximum entries per row. In the table we list the number of states, the number of transitions, the $\mathbf{uP}$ product counts and the solution times. In the GR model we also report the number of recurrence classes. Unpreconditioned GMRES (method ②) does not converge for the SGKCS model, and was the slowest to converge among the three. The *inner-outer* preconditioning strategy (method ⑤) shows a marginal advantage over method ④ in 4 out of 6 models. Note that for some of the larger models the solution time can be quite high, but this is not surprising considering that each matrix-free product requires the uniformization steps of equations (4.23) and (4.24).

4.4 Reducible MRPs: the component method

The matrix-free method for MRPs given in section 4.2.3 works under the restriction of irreducibility. Of course, it is possible to extend the matrix-free method as-is for reducible processes, using equations (2.14) to compute in isolation each recurrent class, as done in [AD11, ch. 5].

A different approach, known as the *component-based method* for MRPs, is provided in [AD10a] and [AD12a]. The component-based method computes the steady-state solution $\boldsymbol{\pi}^{\mathcal{D}}(\boldsymbol{\alpha})$ of the EMC. Instead of computing a single iterative solution of transient set $\mathbf{J}\cdot\mathbf{F}$ in (2.14), the transient set is divided into smaller pieces, called *components*, that can be solved in a sequence. Components taken in isolation can be further classified by their structure, which allows to select the proper solution method (which can span from a single vector-by-matrix multiplication to the implicit solution of a full MRP). Overall, the solution cost is at most that of the implicit method, but in many cases it was found to be much better.

Let's say that the EMC matrix $\mathbf{P}$ is reducible with the structure of (2.11), such that the set of states $\mathcal{S}$ is partitioned into the set of *transient* states $\mathcal{S}_{\mathrm{T}}$ and m sets of *recurrent* states $\mathcal{S}_{\mathrm{R}_i}$, with $1 \leq i \leq m$. Only recurrent states have a non-null probability. Each recurrent class $\mathcal{S}_{\mathrm{R}_i}$ can be solved in isolation as the solution of $\boldsymbol{\pi}_{\mathrm{R}_i} = \boldsymbol{\pi}_{\mathrm{R}_i}\mathbf{R}_i$.

Let's denote with $\boldsymbol{\mu}(\mathrm{T})$ the *outgoing probability vector* from the transient

Table 4.2 Benchmark of the implicit methods ②, ④ and ⑤.

N	States	Trns.	⑤: $\sigma = 10$		④: $\sigma = 10$		②	
			xP	time	**xP**	time	**xP**	time
4	12 985	86 220	11 (12)	12	14	14	122	48
5	45 472	330 897	12 (14)	116	16	137	172	1 004
6	134 064	1 042 720	15 (13)	377	16	438	150	2 026
7	347 034	2 839 536	14 (17)	1 336	20	2 311	171	8 722
8	811 305	6 908 760	15 (18)	4 589	23	4 909	268	33 659
9	1 747 240	15 367 605	17 (19)	11 193	25	13 507	304	104 908
10	3 517 228	31 772 664	18 (20)	27 727	27	33 539	369	282 423

kanban

N	States	Trns.	⑤: $\sigma = 10$		④: $\sigma = 10$		②	
			xP	time	**xP**	time	**xP**	time
20	99 666	456 955	45 (7)	56	49	43	-	-
25	211 081	981 755	55 (8)	157	60	104	-	-
30	394 971	1 855 655	63 (8)	552	85	264	-	-
35	676 836	3 203 905	90 (8)	971	125	889	-	-
40	1 085 926	5 170 505	194 (8)	3 043	182	1 460	-	-

SGKCS

N	States	Trns.	⑤: $\sigma = 10$		④: $\sigma = 10$		②	
			xP	time	**xP**	time	**xP**	time
140	19 881	58 940	9 (8)	221	11	267	21	473
160	25 921	76 960	9 (8)	479	12	495	21	1 013
180	32 761	97 380	9 (8)	866	12	847	21	1 581
200	40 401	120 200	9 (8)	1 043	12	1 431	21	2 336
220	48 841	145 420	9 (8)	1 234	12	2 077	21	2 882

CSMA/CD

N	States	Trns.	⑤: $\sigma = 10$		④: $\sigma = 10$		②	
			xP	time	**xP**	time	**xP**	time
14	491 520	4 177 920	14 (17)	232	25	245	56	413
16	2 228 224	21 168 128	15 (19)	1 197	27	1 156	60	1 609
18	9 961 472	104 595 456	16 (20)	5 252	29	4 919	66	8 992

polling

N	States	Trns.	⑤: $\sigma = 10$		④: $\sigma = 10$		②	
			xP	time	**xP**	time	**xP**	time
5	504	2 268	19 (15)	0.2	25	0.2	854	2
10	6 006	34 034	31 (14)	6	93	10	578	43
15	31 008	193 800	58 (16)	193	94	220	492	977
20	106 260	701 316	61 (20)	888	107	1 296	587	6 431
25	285 012	1 947 582	74 (21)	5 491	98	4 481	692	32 123
30	649 264	4 544 848	90 (22)	14 085	125	25 009	811	107 924

Semaphore

N	States	Trns.	R. C.	⑤: $\sigma = 10$		④: $\sigma = 10$		②	
				xP	time	**xP**	time	**xP**	time
20	44 275	272 580	40	18 (11)	26	19	32	222	144
25	102 375	649 350	50	19 (20)	100	21	106	216	643
30	204 600	1 324 320	60	20 (13)	252	21	113	338	2 342
35	369 075	2 424 240	70	20 (14)	392	22	525	460	6 789
40	617 050	4 098 360	80	21 (15)	758	23	1 004	507	11 964

GR

states, defined as:

$$\boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}) \overset{\text{def}}{=} \boldsymbol{\alpha}_{\mathrm{T}} \cdot \sum_{k=0}^{\infty} \mathbf{T}^{k} \cdot \mathbf{F} \;=\; \boldsymbol{\alpha}_{\mathrm{T}} \cdot (\mathbf{I} - \mathbf{T})^{-1} \cdot \mathbf{F}, \tag{4.30}$$

and let's denote with $\boldsymbol{\mu}^{\mathcal{D}}_{\mathrm{R}_i}(\mathrm{T})$ the probability of entering the i-th recurrent class in the long run from the transient states, defined as:

$$\boldsymbol{\mu}^{\mathcal{D}}_{\mathrm{R}_i}(\mathrm{T}) \overset{\text{def}}{=} \mathbf{I}_{\mathrm{R}_i} \cdot \boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}) \tag{4.31}$$

The filtered quantity $\boldsymbol{\mu}^{\mathcal{D}}_{\mathrm{R}_i}(\mathrm{T})$ is the probability mass that, in the long run, leaves transient states and goes into the i-th recurrent class.

The steady-state probability distribution in the i-th recurrent class can be rewritten using (4.31) as:

$$\boldsymbol{\pi}^{\mathcal{D}}_{\mathrm{R}_i}(\boldsymbol{\alpha}) \overset{\text{def}}{=} \left(\boldsymbol{\mu}^{\mathcal{D}}_{\mathrm{R}_i}(\mathrm{T}) \;+\; \boldsymbol{\alpha}_{\mathrm{R}_i}\right) \cdot \lim_{n\to\infty} \mathbf{R}_i^{n}, \tag{4.32}$$

This section proceeds by describing the component-based method step-by-step, by considering first the explicit and implicit approach for computing (4.32) in a case in which the transient states are treated as a single component. The section then proceeds to the more general case in which the transient set is divided into multiple, strongly connected, subsets. It is the exploitation of the structure of the transient set that leads to the definition of the component method. Finally, a heuristic improvement is described, that exploits a limited form of aggregation of these strongly connected subsets.

4.4.1 Explicit with single transient set

Let's assume for the moment that $\mathbf{P}$ is explicitly built with (2.45). Then the identification of the transient component matrix $\mathbf{T}$ and of the recurrent component matrices $\mathbf{R}_i$ of the EMC needed for the steady-state solution of the MRP can easily be identified from the structure of $\mathbf{P}$ using well-established methods like [Tar71]. Once the structure is known, the vector $\boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T})$ can be computed using Eq. (4.30) for DTMCs, and the probability of recurrent states can then be derived using Eq. (4.32).

4.4.2 Implicit with single transient set

The implicit solution raises two additional problems: the recurrence structure of $\mathbf{P}$ is not known, and the splitting of a matrix-free product $\mathbf{uP}$ into the sub-products $\mathbf{uT}$, $\mathbf{uF}$ and $\mathbf{uR}_i$ used in Eq. (4.30) requires some caution. For the time being, we shall assume that the recurrence structure of $\mathbf{P}$ is given (this matter will be addressed later). For the second problem, we would like to proceed as in the matrix-free case of (4.21): all products with the (implicit) matrices should be rewritten by expanding the matrix expressions. However Eq. (4.30) has the additional problem that it includes a product with $(\mathbf{I}-\mathbf{T})^{-1}$, which is not easily derivable in implicit form, but it can be rewritten to use only products with $\mathbf{F}$ and $\mathbf{T}$ as follows:

$$\boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}) = \boldsymbol{\sigma}_{\mathrm{T}} \cdot \mathbf{F}, \qquad \boldsymbol{\sigma}_{\mathrm{T}} = \boldsymbol{\alpha}_{\mathrm{T}} \cdot (\mathbf{I} - \mathbf{T})^{-1} \quad \Rightarrow \quad \boldsymbol{\sigma}_{\mathrm{T}} \cdot (\mathbf{I} - \mathbf{T}) = \boldsymbol{\alpha}_{\mathrm{T}}$$

where $\boldsymbol{\sigma}_{\mathrm{T}}$ is the solution of a linear equation system in $(\mathbf{I} - \mathbf{T})$ with column vector $\boldsymbol{\alpha}_{\mathrm{T}}$, which requires an iteration over $\mathbf{uT}$ products. A similar schema can be derived for $\boldsymbol{\pi}^{\mathcal{D}}_{\mathrm{R}_i}(\boldsymbol{\alpha})$ of Eq. (4.32), which requires just products with $\mathbf{R}_i$. With the above formulation, only vector$\times$matrix product with $\mathbf{T}$, $\mathbf{F}$ and $\mathbf{R}_i$ are needed.

The above implicit products should be done with a little care, since they may involve a state space based on an *augmented set*. The problem is well illustrated in Fig. 4.6(a), that shows a portion of MRTS in which there are two SCCs, namely $\{s_1\}$ and $\{s_2, s_3\}$. When the regenerative process enters state s_1, it may reach state s_2 with a $\mathbf{q}$ transition before the next regeneration event takes place (that is, before the firing of the $\boldsymbol{\delta}$ transition from s_1), so the state s_2 has to be considered in the computation of the subordinated behaviour.

This implies that implicit products with a subset of states $\mathcal{S}_A \subseteq \mathcal{S}$ should consider an *augmented* set of states $\mathcal{S}_{\widehat{A}}$ which includes all the states of $\mathcal{S}_A$, plus all other states that the process can reach from $\mathcal{S}_A$ before a new regeneration point is entered (firing/preemption of the general event). The added states are therefore those states reachable from $\mathcal{S}_A$ with a sequence of one or more $\mathbf{q}$ transitions: $j \in \mathcal{S}_{\widehat{A}}$ iff $j \in \mathcal{S}_A$ or $j \in \mathcal{S}^{\mathrm{G}}$ and there exists $i \in \mathcal{S}_A$ such that there is a path of $\mathbf{q}$ transitions from i to j. For instance, the augmented set of $\{s_1\}$ in Fig. 4.6(a) is $\{s_1, s_2\}$.

Given the augmented sets $\mathcal{S}_{\widehat{\mathrm{T}}}$ and $\mathcal{S}_{\widehat{\mathrm{R}_i}}$ of $\mathcal{S}_{\mathrm{T}}$ and $\mathcal{S}_{\mathrm{R}_i}$ and a generic vector $\mathbf{u}$, the products $\mathbf{uT}$, $\mathbf{uF}$ and $\mathbf{uR}_i$ in implicit form are:

$$
\begin{aligned}
\mathbf{uT} &= \mathbf{I}_{\mathrm{T}} \cdot \left[\mathbf{u}\left(\mathbf{I}^{\mathrm{E}}_{\mathrm{T}} - \mathsf{diag}^{-1}(\mathbf{Q}^{\mathrm{E}}_{\mathrm{T}})\, \mathbf{Q}^{\mathrm{E}}_{\mathrm{T}}\right) + \mathbf{u}\, \boldsymbol{\Omega}_{\widehat{\mathrm{T}}}\, \boldsymbol{\Delta}_{\widehat{\mathrm{T}}} + \mathbf{u}\, \boldsymbol{\Psi}_{\widehat{\mathrm{T}}}\, \bar{\mathbf{Q}}_{\widehat{\mathrm{T}}}\right] \\
\mathbf{uF} &= \mathbf{I}_{\mathrm{R}} \cdot \left[\mathbf{u}\left(\mathbf{I}^{\mathrm{E}}_{\mathrm{T}} - \mathsf{diag}^{-1}(\mathbf{Q}^{\mathrm{E}}_{\mathrm{T}})\, \mathbf{Q}^{\mathrm{E}}_{\mathrm{T}}\right) + \mathbf{u}\, \boldsymbol{\Omega}_{\widehat{\mathrm{T}}}\, \boldsymbol{\Delta}_{\widehat{\mathrm{T}}} + \mathbf{u}\, \boldsymbol{\Psi}_{\widehat{\mathrm{T}}}\, \bar{\mathbf{Q}}_{\widehat{\mathrm{T}}}\right] \qquad (4.33) \\
\mathbf{uR}_i &= \mathbf{I}_{\mathrm{R}_i} \cdot \left[\mathbf{u}\left(\mathbf{I}^{\mathrm{E}}_{\mathrm{R}_i} - \mathsf{diag}^{-1}(\mathbf{Q}^{\mathrm{E}}_{\mathrm{R}_i})\, \mathbf{Q}^{\mathrm{E}}_{\mathrm{R}_i}\right) + \mathbf{u}\, \boldsymbol{\Omega}_{\widehat{\mathrm{R}_i}}\, \boldsymbol{\Delta}_{\widehat{\mathrm{R}_i}} + \mathbf{u}\, \boldsymbol{\Psi}_{\widehat{\mathrm{R}_i}}\, \bar{\mathbf{Q}}_{\widehat{\mathrm{R}_i}}\right]
\end{aligned}
$$

where $\boldsymbol{\Omega}_{\widehat{\mathrm{T}}}$, $\boldsymbol{\Psi}_{\widehat{\mathrm{T}}}$, $\boldsymbol{\Omega}_{\widehat{\mathrm{R}_i}}$ and $\boldsymbol{\Psi}_{\widehat{\mathrm{R}_i}}$ are computed with the exponentials of $\mathbf{Q}^{\mathrm{G}}_{\widehat{\mathrm{T}}}$ and $\mathbf{Q}^{\mathrm{G}}_{\widehat{\mathrm{R}_i}}$. Recall that the exponential of $\mathbf{Q}^{\mathrm{G}}$ gives the reachability among regenerative states based on the subordinated Markov chains and should be based on the augmented set, for the reasons explained above. Clearly no augmented set is needed for components which are in $\mathcal{S}^{\mathrm{E}}$, since the firing of each exponential event corresponds to entering a new regeneration point.

4.4.3 Explicit and component-based

If the transient set $\mathcal{S}_{\mathrm{T}}$ does not constitute a single strongly connected component, we can devise an alternative solution that exploits the component structure of $\mathcal{S}_{\mathrm{T}}$ in a directed acyclic graph (DAG) of k SCCs, resulting in a partitioning $\mathcal{S}_{\mathrm{T}_1} \cdots \mathcal{S}_{\mathrm{T}_k}$ of the transient states. Without loss of generality we assume that the index of the transient classes gives a total order compatible with the partial order induced by the DAG. In this case, the stochastic matrix can be written as:

$$\mathbf{P} = \begin{bmatrix} \mathbf{T}_1 & \boxed{\quad\mathbf{F}_1\quad} & & & & \\ & \ddots & \ddots & & & \\ & & \mathbf{T}_k & \boxed{\quad\mathbf{F}_k\quad} & & \\ & & & \mathbf{R}_1 & & \\ & & & & \ddots & \\ & & & & & \mathbf{R}_m \end{bmatrix} \tag{4.34}$$

The idea of the component method is then to compute the probability of reaching each recurrent set by moving forward the probability from the transient sets taken for increasing value of their index, an order which respects the ordering of the $\mathcal{S}_{\mathrm{T}_i}$ sets.

Let $\boldsymbol{\mu}^{\mathcal{D}}(T_i)$ be the probability vector outgoing the states of $\mathcal{S}_{\mathrm{T}_i}$, and let $\boldsymbol{\mu}^{\mathcal{D}}_{\mathrm{T}_i}(\mathrm{T}_j) = \mathbf{I}_{\mathrm{T}_i} \cdot \boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_j)$ be the outgoing probability that reaches the set $\mathcal{S}_{\mathrm{T}_i}$ from the set $\mathcal{S}_{\mathrm{T}_j}$. Given the initial probability vector $\boldsymbol{\alpha}$ and an SCC $\mathcal{S}_{\mathrm{T}_i}$, we can compute $\boldsymbol{\mu}^{\mathcal{D}}(T_i)$ with:

$$\boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_i) \;=\; \Big(\boldsymbol{\alpha}_{\mathrm{T}_i} + \textstyle\sum_{j<i} \boldsymbol{\mu}^{\mathcal{D}}_{\mathrm{T}_i}(\mathrm{T}_j)\Big) \cdot (\mathbf{I} - \mathbf{T}_i)^{-1} \cdot \mathbf{F}_i \tag{4.35}$$

Given a recurrent set $\mathcal{S}_{\mathrm{R}_i}$, its stationary distribution conditioned to $\boldsymbol{\alpha}$ is:

$$\boldsymbol{\pi}^{\mathcal{D}}_{\mathrm{R}_i} \;=\; \Big(\boldsymbol{\alpha}_{\mathrm{R}_i} + \textstyle\sum_{j=1}^{k} \boldsymbol{\mu}^{\mathcal{D}}_{\mathrm{R}_i}(\mathrm{T}_j)\Big) \cdot \lim_{n\to\infty} \big(\mathbf{R}_i\big)^n \tag{4.36}$$

The outgoing probability $\boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_i)$ for set $\mathcal{S}_{\mathrm{T}_i}$ depends on all the previous $\boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_j)$, in topological order of the SCCs. Stationary recurrent probabilities depend on all the $\boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_i)$ vectors.

Observe that (4.35) is additive over the union operator for a target set of states C, so $\boldsymbol{\mu}^{\mathcal{D}}_C(A \cup B) = \boldsymbol{\mu}^{\mathcal{D}}_C(A) + \boldsymbol{\mu}^{\mathcal{D}}_C(B)$. The relation holds when A, B and C are mutually disjoint. Therefore $\boldsymbol{\mu}^{\mathcal{D}}_{\mathrm{R}_i}(\mathrm{T}) = \sum_{j=1}^{k} \boldsymbol{\mu}^{\mathcal{D}}_{\mathrm{R}_i}(\mathrm{T}_j)$. It follows that (4.36) is just a reformulation of (4.32), so the equation correctly computes the steady-state probability of recurrent states. The whole method can then be described in Algorithm 4.4:

Algorithm 4.4 Component-based solution with explicit knowledge of $\mathbf{P}$

1. Compute with (4.35) the outgoing probability vector $\boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_j)$ of every component $\mathcal{S}_{\mathrm{T}_i}$ following the total order.
2. The total outgoing probability $\boldsymbol{\mu}^{\mathcal{D}}_{\mathrm{R}_i}(\mathrm{T})$ from the transient set into each class R_i is given by: $\mathbf{I}_{\mathrm{R}_i} \cdot \sum_{j=1}^{k} \boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_j)$.
3. Compute the probability of every recurrent class with (4.36), weighted by the entering probability $|\,\boldsymbol{\mu}^{\mathcal{D}}_{\mathrm{R}_i}(\mathrm{T})\,|$.

4.4.4 Implicit and component-based

Assume now that we do not want to incur the time and storage cost of building the matrix $\mathbf{P}$. The computation of each $\boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_i)$ can be done entirely in implicit form, assuming that we can do a vector×matrix product with $\mathbf{T}_i$ and $\mathbf{F}_i$. As before, the implicit product formulas (4.34) can be adapted to compute the $\mathbf{uT}_i$ and $\mathbf{uF}_i$ products, using the augmented set $\mathcal{S}_{\widehat{\mathrm{T}}_i}$ of $\mathcal{S}_{\mathrm{T}_i}$, resulting in:

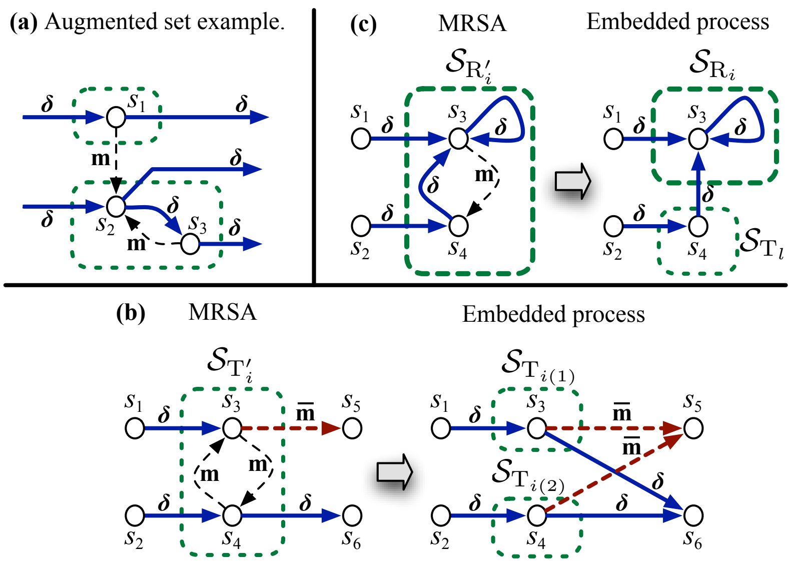
$$\begin{aligned} \mathbf{uT}_i &= \mathbf{I}_{\mathrm{T}_i} \cdot \left[\mathbf{u} \left(\mathbf{I}^{\mathrm{E}}_{\mathrm{T}_i} - \mathsf{diag}^{-1}(\mathbf{Q}^{\mathrm{E}}_{\mathrm{T}_i})\, \mathbf{Q}^{\mathrm{E}}_{\mathrm{T}_i} \right) + \mathbf{u}\,\boldsymbol{\Omega}_{\widehat{\mathrm{T}}_i}\, \boldsymbol{\Delta}_{\widehat{\mathrm{T}}_i} + \mathbf{u}\,\boldsymbol{\Psi}_{\widehat{\mathrm{T}}_i}\, \bar{\mathbf{Q}}_{\widehat{\mathrm{T}}_i} \right] \\ \mathbf{uF}_i &= (\mathbf{I} - \mathbf{I}_{\mathrm{T}_i}) \cdot \left[\mathbf{u} \left(\mathbf{I}^{\mathrm{E}}_{\mathrm{T}_i} - \mathsf{diag}^{-1}(\mathbf{Q}^{\mathrm{E}}_{\mathrm{T}_i})\, \mathbf{Q}^{\mathrm{E}}_{\mathrm{T}_i} \right) + \mathbf{u}\,\boldsymbol{\Omega}_{\widehat{\mathrm{T}}_i}\, \boldsymbol{\Delta}_{\widehat{\mathrm{T}}_i} + \mathbf{u}\,\boldsymbol{\Psi}_{\widehat{\mathrm{T}}_i}\, \bar{\mathbf{Q}}_{\widehat{\mathrm{T}}_i} \right] \end{aligned} \tag{4.37}$$

The only remaining problem is that without the explicit knowledge of $\mathbf{P}$, the component structure is not available. The work in [AD10a] uses instead the structure of the process X_t, which can be computed simply on the graph of the MRTS. This structure is different from that of the renewal sequence Y_n (and thus of the EMC $\mathbf{P}$). It is therefore important to understand if the structure of the MRTS can be used in place of the structure of the embedded process Y_n identified by $\mathbf{P}$.

A first, general, observation is that this choice is adequate for the states in the exponential partition $\mathcal{S}^{\mathrm{E}}$, since these states have the same connectivity in the MRTS and in $\mathbf{P}$, i.e. each state in $\mathcal{S}^{\mathrm{E}}$ corresponds to a regeneration point. Instead a path $s_0 \xrightarrow{\mathbf{q}} \cdots \xrightarrow{\mathbf{q}} s_{n-1} \xrightarrow{\boldsymbol{\delta} \text{ or } \bar{\mathbf{q}}} s_n$ in the general state partition $\mathcal{S}^{\mathrm{G}}$ of the MRTS becomes a single state transition $s_0 \xrightarrow{\boldsymbol{\delta} \text{ or } \bar{\mathbf{q}}} s_n$ in the embedded process, so that the intermediate (non-regenerative) states reached by $\mathbf{q}$ transitions are not visible. This is a sort of transitive closure over the $\mathbf{q}$ transitions.

For a deeper understanding let's indicate with prime variables the sets computed on the MRTS, and with unprimed variables those computed on $\mathbf{P}$, and let $\mathcal{S}_{\mathrm{T}'_1} \dots \mathcal{S}_{\mathrm{T}'_{k'}}$ and $\mathcal{S}_{\mathrm{R}'_1} \dots \mathcal{S}_{\mathrm{R}'_{m'}}$ be the k' transient and m' recurrent SCCs of the structure of the MRTS. What are the relationships between primed and unprimed sets? Remember that the following considerations only apply to components in $\mathcal{S}^{\mathrm{G}}$.

Figure 4.6 Difference in the component structure of an MRP and of its embedded process.



Figs. 4.6(b) and (c) show on the left an MRTS and on the right the structure of the embedded process $\mathbf{P}$ and depict the differences of the two structures, which can be summarized as:

1. *Reachability in the MRTS is richer than in* $\mathbf{P}$. For the definition of embedded process, if a state j is reachable (in one or more steps) from a state i in the embedded process, described by $\mathbf{P}$, then it is reachable from i also in the MRTS. If instead a state j is reachable from a state i in the MRTS, then it may be the case that j is not reachable by i in $\mathbf{P}$. This is because the embedded process only observes transitions among regenerative states, and for general states these transitions are of $\overline{\mathbf{q}}$ and $\boldsymbol{\delta}$ type. Indeed in the MRP of Fig. 4.6(b) state s_4 is reachable from state s_3 in the MRTS but not in the embedded process.
2. *Recurrent components in* $\mathbf{P}$ *are subsets of those in the MRTS.* The number of recurrent classes in $\mathbf{P}$ and in the MRTS is the same ($m = m'$), but it holds that $\mathcal{S}_{\mathrm{R}_i} \subseteq \mathcal{S}_{\mathrm{R}'_i}$, and the inclusion may be strict. The case is well-illustrated by the example of Fig. 4.6(c): the state s_4 is recurrent in the MRTS, but it is visited only once by the embedded process, and therefore it is not part of a recurrent component, so the bottom SCC $\{s_3, s_4\}$ of the MRTS reduces to the single state s_3 in the embedded process, and the (sub)path $s_3 \xrightarrow{\overline{\mathbf{q}}} s_4 \xrightarrow{\boldsymbol{\delta}} s_3$ reduces to a self-loop on s_3
3. *Inclusion of transient components.* Each $\mathcal{S}_{\mathrm{T}_i}$ is contained either into a single $\mathcal{S}_{\mathrm{T}'_j}$ (as in Fig. 4.6(b)), or in a single $\mathcal{S}_{\mathrm{R}'_k}$ (as in Fig. 4.6(c)), and each $\mathcal{S}_{\mathrm{T}'_i}$ is a union of SCCs of $\mathbf{P}$, indicated by $\mathcal{S}_{\mathrm{T}_{i(1)}} \dots \mathcal{S}_{\mathrm{T}_{i(h)}}$.The number of transient classes in $\mathbf{P}$ and in the MRTS differs ($k' \neq k$). These inclusion relationships derive from the fact that the reachability relation of $\mathbf{P}$ is a subset of the reachability relation of the MRTS.

To use the SCCs $\{\mathcal{S}_{\mathrm{T}'_i}\}$ and $\{\mathcal{S}_{\mathrm{R}'_j}\}$ of an MRTS as the basis of the component method, we need to prove that they lead to the same steady-state probability of the states in each $\mathcal{S}_{\mathrm{R}_j}$. Equation (4.36), computed by Algorithm 4.4 using the MRTS structure, results in:

$$\boldsymbol{\pi}^{\mathcal{D}}_{\mathrm{R}'_i} \;=\; \left(\boldsymbol{\alpha}_{\mathrm{R}_i} + \boldsymbol{\alpha}_{\mathrm{R}'_i \setminus \mathrm{R}_i} + \textstyle\sum_{j=1}^{k'} \boldsymbol{\mu}^{\mathcal{D}}_{\mathrm{R}'_i}(\mathrm{T}'_j)\right) \cdot \lim_{n\to\infty} \left(\mathbf{R}'_i\right)^n \tag{4.38}$$

and we need to prove that, for every state $s \in \mathcal{S}_{\mathrm{R}_i}$ it holds that $\boldsymbol{\pi}^{\mathcal{D}}_{\mathrm{R}'_i}(s) = \boldsymbol{\pi}^{\mathcal{D}}_{\mathrm{R}_i}(s)$. Note that this equivalence is only for the regenerative states of the MRP, since this is what is computed by the component method. To show the equivalence we consider three different cases: 1) s is classified as transient in $\mathbf{P}$ and in the MRTS; 2) s is classified as transient in $\mathbf{P}$ and as recurrent in the MRTS; 3) s is classified as recurrent in both.

For the first case, the steady-state probability is obviously zero in both cases, and the issue is whether the probability of entering a transient state s of $\mathcal{S}_{\mathrm{T}'_j}$ from the components that precede $\mathcal{S}_{\mathrm{T}'_j}$ is the DAG. But this is indeed the case, due to the additivity of the $\boldsymbol{\mu}^{\mathcal{D}}(\mathcal{S}_{\mathrm{T}_i})$ vectors, as discussed before. For example in Fig. 4.6(b), the probability of entering s_5 from component $\mathcal{S}_{\mathrm{T}'_i}$ is the sum of the probabilities of entering s_5 respectively from $\mathcal{S}_{\mathrm{T}_{i(1)}}$ and $\mathcal{S}_{\mathrm{T}_{i(2)}}$.

For the second case, if s is classified as transient in $\mathbf{P}$, then it means that it is not a recurrent regenerative state also in the MRP, and its steady-state probability is zero. However, if the state is not recurrent regenerative, even the implicit computation based on the recurrent component of the MRTS will assign to it a zero probability. With reference to the example of Fig. 4.6(c), this is the case for state s_4, which, in the recurrent component $\mathcal{S}_{\mathrm{R}'_i}$, is always entered by a non-preemptive $\mathbf{q}$ arc, so s_4 is not regenerative recurrent. The issue is then whether the initial probability $\boldsymbol{\alpha}_{\mathrm{R}'_i}(s)$ assigned to s is correctly taken into account. The value of $\boldsymbol{\alpha}_{\mathrm{R}'_i}(s)$ is considered by Eq. (4.35) when using the SCCs of $\mathbf{P}$, while it is taken into account in the term $\boldsymbol{\alpha}_{\mathrm{R}'_i \setminus \mathrm{R}_i}$ by Eq. (4.38) when using the SCCs of the MRTS.

For the last case, if s is in a recurrent component $\mathcal{S}_{\mathrm{R}_i}$ of $\mathbf{P}$ then s belongs to the $\mathcal{S}_{\mathrm{R}'_i}$ of the MRTS. As explained before, any state in $\mathcal{S}_{\mathrm{R}'_i} \setminus \mathcal{S}_{\mathrm{R}_i}$ is not regenerative recurrent, and its initial probability is correctly taken into account in the steady-state solution equations, so the computation of the probability of s can be based on $\mathcal{S}_{\mathrm{R}'_i}$.

In summary, the component-based, implicit, solution for $\boldsymbol{\pi}^{\mathcal{D}}_{\mathrm{R}_i}(\boldsymbol{\alpha})$ can be done by exploiting the structure of the MRTS, without the need of having $\mathbf{P}$ entries.

Component classification But what are the advantages of solving a set of MRPs (one per SCC) instead of taking all the SCCs together? The main advantage is that the computation of the vector $\boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_i)$, for some component i may require a simpler solution technique than a full MRP implicit (or explicit) technique. For example, we can observe that when $\mathbf{T}_i = 0$ then the computation of Eq. (4.35) for $\boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_i)$ does not require a matrix inversion. This condition is particularly convenient when the matrix product is implicit. Another favorable condition is when the internal behavior of the component is fully Markovian, since the computation of the vector reduces to a CTMC solution.

Following the above criteria five types of components are identified: they are illustrated in Fig. 4.7, and defined in Table 4.3. The table lists the component type, the conditions on its state space $\mathcal{S}_{\mathrm{T}_i}$ and on the matrices $\mathbf{Q}, \bar{\mathbf{Q}}$, and $\boldsymbol{\Delta}$, the translation of the matrix requirements into conditions that can be checked on the MRTS structure, and what the $\mathbf{T}_i$ matrix would be for a component of that type. Note that we are still considering an implicit approach, so $\mathbf{T}_i$ is actually never built, and the computation of $\boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_i)$ is based on Eq. (4.37).

A component i of type ① has no internal behavior, and therefore will result in a $\mathbf{T}_i$ matrix made of zeroes. Indeed each transition from an $\mathcal{S}_{\mathrm{T}_i}$ state takes the MRP out of the component, so the computation of $\boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_i)$ is just a vector matrix multiplication with $\mathbf{F}_i$:

$$\begin{array}{c} ①\text{:} \;\; \boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_i) \;=\; \boldsymbol{\beta}_{\mathrm{T}_i} \cdot \Big(\mathbf{I}^{\mathrm{E}}_{\mathrm{T}_i} - \mathsf{diag}^{-1}(\mathbf{Q}^{\mathrm{E}}_{\mathrm{T}_i})\, \mathbf{Q}^{\mathrm{E}}_{\mathrm{T}_i} \Big) \\ \text{with cost: } O\big(\eta(\mathbf{Q}^{\mathrm{E}}_{\mathrm{T}_i})\big) \end{array} \tag{4.39}$$

where $\boldsymbol{\beta}_{\mathrm{T}_i} = \big(\boldsymbol{\alpha}_{\mathrm{T}_i} + \mathbf{I}_{\mathrm{T}_i} \cdot \sum_{j<i} \boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_j) \big)$ is the entering probability vector for $\mathcal{S}_{\mathrm{T}_i}$.

In case ② there are transitions inside the component, but they are all Markovian, as are Markovian the transitions out of $\mathcal{S}_{\mathrm{T}_i}$, so the computation of $\boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_i)$ requires the steady-state solution of a CTMC, with the usual cost of $O\big(M_{\mathrm{T}_i} \cdot \eta(\mathbf{Q}^{\mathrm{E}}_{\mathrm{T}_i})\big)$, assuming that M_{T_i} iterations are required to achieve convergence to the steady-state solution.

As for type ①, also components of type ③ will result in a $\mathbf{T}_i$ matrix made of zeroes, but now the component has also an internal behaviour, determined only by non-preemptive Markovian transitions, while transitions that leave $\mathcal{S}_{\mathrm{T}_i}$ are

Figure 4.7 Examples of the five kinds of MRTS components.

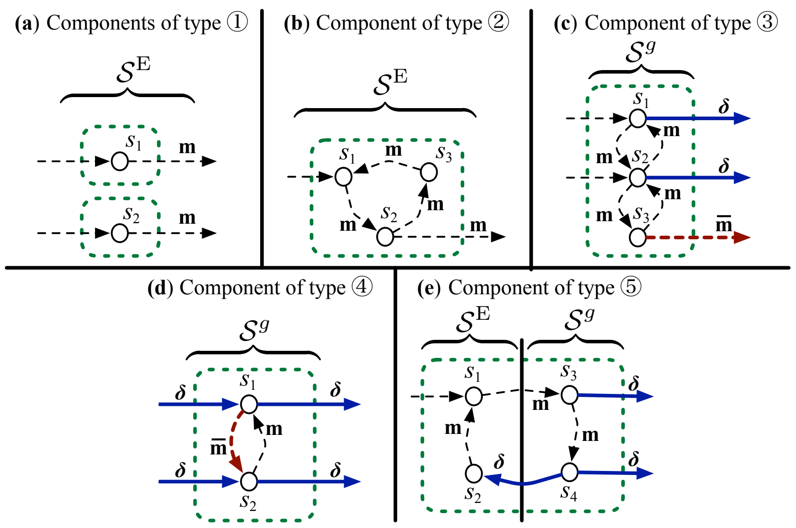


Table 4.3 Component classification.

	Conditions	*MRTS characteristics*
①	$\mathcal{S}_{\mathrm{T}_i} \subseteq \mathcal{S}^{\mathrm{E}}$ and $\mathbf{Q}^{\mathrm{E}}_{\mathrm{T}_i,\mathrm{T}_i} = 0$	The component is in $\mathcal{S}^{\mathrm{E}}$ and has no $\mathbf{q}$ transitions between $\mathcal{S}_{\mathrm{T}_i}$ states. The sub-matrix $\mathbf{T}_i$ is empty.
②	$\mathcal{S}_{\mathrm{T}_i} \subseteq \mathcal{S}^{\mathrm{E}}$ and $\mathbf{Q}^{\mathrm{E}}_{\mathrm{T}_i,\mathrm{T}_i} \neq 0$	The component is in $\mathcal{S}^{\mathrm{E}}$ with $\mathbf{q}$ transitions between $\mathcal{S}_{\mathrm{T}_i}$ states. The sub-matrix $\mathbf{T}_i$ is not empty.
③	$\mathcal{S}_{\mathrm{T}_i} \subseteq \mathcal{S}^{g}$ and $\bar{\mathbf{Q}}_{\mathrm{T}_i,\mathrm{T}_i} = 0$ and $\mathbf{\Delta}_{\mathrm{T}_i,\mathrm{T}_i} = 0$	The component is in $\mathcal{S}^{g}$ and has no $\bar{\mathbf{q}}$ and $\boldsymbol{\delta}$ transitions between $\mathcal{S}_{\mathrm{T}_i}$ states. The sub-matrix $\mathbf{T}_i$ is empty.
④	$\mathcal{S}_{\mathrm{T}_i} \subseteq \mathcal{S}^{g}$ and $(\bar{\mathbf{Q}}_{\mathrm{T}_i,\mathrm{T}_i} \neq 0$ or $\mathbf{\Delta}_{\mathrm{T}_i,\mathrm{T}_i} \neq 0)$	The component is in $\mathcal{S}^{g}$ with $\bar{\mathbf{q}}$ or $\boldsymbol{\delta}$ transitions between $\mathcal{S}_{\mathrm{T}_i}$ states. The sub-matrix $\mathbf{T}_i$ is not empty.
⑤	$\mathcal{S}_{\mathrm{T}_i}$ has multiple state partitions.	The component has different general transitions enabled in $\mathcal{S}_{\mathrm{T}_i}$. The sub-matrix $\mathbf{T}_i$ is not empty.

either general or preemptive Markovian. The computation of $\boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_i)$ amounts then to a transient analysis of $\mathbf{Q}^{g}_{\mathrm{T}_i}$ at the time of firing of g:

$$
\begin{aligned}
③\!:\ \boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_i) \ &= \ \boldsymbol{\beta}_{\mathrm{T}_i} \cdot \Big(\mathbf{\Omega}^{g}_{\widehat{\mathrm{T}}_i}\, \mathbf{\Delta}^{g}_{\widehat{\mathrm{T}}_i} + \mathbf{\Psi}^{g}_{\widehat{\mathrm{T}}_i}\, \bar{\mathbf{Q}}^{g}_{\widehat{\mathrm{T}}_i} \Big) \\
&\text{with cost: } O\Big(\eta(\mathbf{Q}^{g}_{\widehat{\mathrm{T}}_i}) \cdot R^{g}_{\widehat{\mathrm{T}}_i} + \eta(\mathbf{\Delta}^{g}_{\widehat{\mathrm{T}}_i}) + \eta(\bar{\mathbf{Q}}^{g}_{\widehat{\mathrm{T}}_i}) \Big)
\end{aligned}
\tag{4.40}
$$

In the other two cases (④ and ⑤) the component has the full behaviour of an MRP, so the computation requires a fixed-point solution of the MRP based on the following equations, illustrated before and rewritten here for convenience:

$$\boldsymbol{\sigma}_{\mathrm{T}_i} \;=\; \boldsymbol{\beta}_{\mathrm{T}_i} \cdot (\mathbf{I} - \mathbf{T}_i)^{-1} \quad\Rightarrow\quad \boldsymbol{\sigma}_{\mathrm{T}_i} \cdot (\mathbf{I} - \mathbf{T}_i) \;=\; \boldsymbol{\beta}_{\mathrm{T}_i} \tag{4.41}$$

The cost of ④ is then that of a fixed-point , with a cost per iteration that is as in (4.40), leading to:

$$\text{Cost of ④:}\;\; O\Big(M_{\mathrm{T}_i} \cdot \big(\eta(\mathbf{Q}^g_{\widehat{\mathrm{T}}_i}) \cdot R^g_{\widehat{\mathrm{T}}_i} + \eta(\boldsymbol{\Delta}^g_{\widehat{\mathrm{T}}_i}) + \eta(\bar{\mathbf{Q}}^g_{\widehat{\mathrm{T}}_i})\big)\Big) \tag{4.42}$$

The cost of ⑤ is again the cost of a fixed-point iterative method with a cost per iteration which is that of ② plus the cost of ④ for each transient component $\mathcal{S}_{\mathrm{T}_i}$, which results in:

$$\text{Cost of ⑤:}\;\; O\Big(M_{\mathrm{T}_i} \cdot \Big(\eta(\mathbf{Q}^{\mathrm{E}}_{\mathrm{T}_i}) + \textstyle\sum_{g\in \mathrm{G}} \big(\eta(\mathbf{Q}^g_{\widehat{\mathrm{T}}_i}) \cdot R^g_{\widehat{\mathrm{T}}_i} + \eta(\boldsymbol{\Delta}^g_{\widehat{\mathrm{T}}_i}) + \eta(\bar{\mathbf{Q}}^g_{\widehat{\mathrm{T}}_i})\big)\Big)\Big) \tag{4.43}$$

A component $\mathcal{S}_{\mathrm{T}_i}$ of the first three kinds is *simple*, *complex* otherwise.

Algorithm 4.4 can then be enriched to consider the above component classification, leading to Algorithm 4.5:

Algorithm 4.5 Implicit component solution without explicit knowledge of **P**

Derive the recurrent structure of the MRTS from $\mathbf{Q}$, $\bar{\mathbf{Q}}$ and $\boldsymbol{\Delta}$. Classify each transient component $\mathcal{S}_{\mathrm{T}_i}$ according to the five component kinds. Then proceed as Algorithm 4.4, using the appropriate implicit formula for the computation of each $\boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_j)$ (step 1). Steps 2 and 3 of Algorithm 4.4 remain unchanged.

Aggregation of transient SCCs into larger components There are some cases in which components can be aggregated together to improve the overall efficiency of the method. For instance, there could be many small components (as small as one state, as arising from type ①). Another situation that may lead to aggregation occurs for the components $\mathcal{S}_{\mathrm{T}_i}$ which are in a general state partition. In this case the solution is based on the augmented state space $\mathcal{S}_{\widehat{\mathrm{T}}_i}$, that could even be so large to cover other components: an aggregation could then avoid repeating the computation over the same set of states.

The correctness of an aggregation technique is ensured by the additivity of the computation of $\boldsymbol{\mu}^{\mathcal{D}}(\mathrm{T}_i)$, given that the aggregations performed do not violate the partial order of the SCCs. Indeed two components w and v of W can be aggregated into the new component $w{+}v$ given that there is no intermediate component $u \in W$ such that $w \leq u \leq v$, where two components v and v' are in relation $v \leq v'$ iff it is possible to reach a state of $\mathcal{S}_{v'}$ from a state of $\mathcal{S}_v$. The set of states of the aggregated component $w+v$ is defined as the union of the states of w and v. The set W of aggregated components can be built through successive component aggregations, starting from an initial set $W^{(0)}$ which contains the SCCs of the MRTS.

Equation (4.35) can then be rewritten for a generic aggregated component $w_i \in W$, assuming aggregated components are indexed with a total ordering that respects the poset $(W, \leq)$, as:

$$\boldsymbol{\mu}^{\mathcal{D}}(w_i) \;=\; \Big(\boldsymbol{\alpha}_{w_i} + \textstyle\sum_{j<i} \boldsymbol{\mu}^{\mathcal{D}}_{w_i}(w_j)\Big) \cdot (\mathbf{I} - \mathbf{T}_{w_i})^{-1} \cdot \mathbf{F}_{w_i} \tag{4.44}$$

where $\mathbf{T}_{w_i}$ is the square sub-stochastic matrix of the transitions between $\mathcal{S}_{w_i}$ states, $\mathbf{F}_{w_i}$ is the rectangular matrix of the outgoing transitions from $\mathcal{S}_{w_i}$ states to every other state of $\mathcal{S}$, and $\boldsymbol{\mu}^{\mathcal{D}}_{w_i}(w_j) = \mathbf{I}_{w_i} \cdot \boldsymbol{\mu}^{\mathcal{D}}(w_j)$ is the outgoing probability that reaches the set $\mathcal{S}_{w_i}$ from the set $\mathcal{S}_{w_j}$. The correctness of (4.44) is ensured by the additivity of $\boldsymbol{\mu}$ over the union of components. The implicit product formulas (4.37) can be modified: the $\mathbf{u}\mathbf{T}_{w_i}$ and $\mathbf{u}\mathbf{F}_{w_i}$ products are computed, using the sets $\mathcal{S}_{w_i}$ and $\mathcal{S}_{\widehat{w_i}}$, instead of $\mathcal{S}_{\mathrm{T}_i}$ and $\mathcal{S}_{\widehat{\mathrm{T}}_i}$.

Given that we can aggregate components, when is it convenient to do so? There is no absolute winner here, since joining components may have a detrimental impact on the number of uniformisation steps R_i of the transient solutions, and on the number of iterations M_i needed in fixed point computations. We have therefore devised a heuristic approach, driven by a few simple indications. The experimental evaluation of the heuristics is reported in Section 4.4.6.

A general indication is that, since simple components are less expensive to compute than complex ones, then aggregation should preserve simplicity. According to this principle, any aggregation of a simple component with a complex one should be avoided, since this will result in a larger complex component. Moreover, considering that the three types of simple components have different computational cost, it may be convenient to avoid aggregation among components of different kinds.

The heuristic can then be expressed as follows: given two components $w, v \in W$, for which it does not exist an intermediate component:

1. If w, v are complex components, then they are aggregated. This is justified by the fact that the complexity of type ④ and ⑤ components is very similar, and their aggregation could benefit from the sharing of the states in the augmented sets, since $\left|\mathcal{S}_{\widehat{w}}\right| + \left|\mathcal{S}_{\widehat{v}}\right| \geq \left|\mathcal{S}_{\widehat{w+v}}\right|$.
2. If w, v are simple, of the same type, then they are aggregated only if $(w{+}v)$ is of the same type. We observe that:
   (a) If w, v are of type ②, then $(w{+}v)$ has type ②.
   (b) If w, v are of type ①, the aggregation $(w+v)$ is of type ① only if there does not exist a $\mathbf{q}$ arc between the states of w and v, otherwise, $(w{+}v)$ becomes of type ② and no aggregation should be done.
   (c) If w, v are of type ③, the aggregation $(w+v)$ is of type ③ only if there does not exist a $\overline{\mathbf{q}}$ or a $\boldsymbol{\delta}$ arc between the states of w and v, otherwise, $(w{+}v)$ becomes of type ④, and no aggregation should be done.

The above heuristic is implemented in the following algorithm:

The cost of construction of $W^{(0)}$ is $O(K \log K)$, where K is the number of SCCs. The heuristic of algorithm 4.6 has also some disadvantages: the union of components with a large difference in the exponential transition rates increases the number of steps of the uniformization R and may also increase the number of fixed point iterations M.

Fig. 4.8 shows an example of aggregation: starting from an MRTS (a), the digraph of the SCC is built (b), which constitutes the poset $W^{(0)}$ which is the basis for the recurrence of Algorithm 4.6 that computes the digraph of the aggregated components shown in (c). For each node $\mathcal{S}_{\mathrm{T}_i}$ in (b) and $\mathcal{S}_{w_i}$ in (c), the figure reports the set of states, the state subset and the component type.

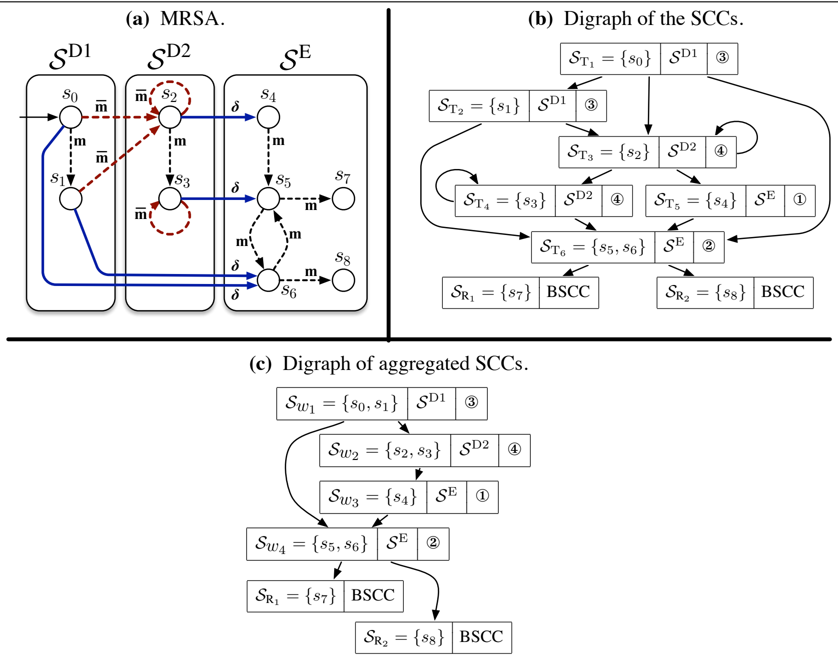
Algorithm 4.6 Heuristic for the choice of the poset W

The set of components W is the fixed-point of the following iteration:

$$
\begin{aligned}
W^{(0)} &= \text{recurrence structure of the MRTS, from } \mathbf{Q} + \bar{\mathbf{Q}} + \boldsymbol{\Delta} \\
W^{(i+1)} &= W^{(i)} \setminus \{w, v\} \cup \{w{+}v\} \text{ iff: } \nexists u \in W^{(i)} \;:\; w \leq u \leq v \text{ and} \\
&\quad \big(w \text{ and } v \text{ are simple, of the same type AND} \\
&\qquad w{+}v \text{ has the same type of } w \text{ and } v, \\
&\quad\; \text{OR } w \text{ and } v \text{ are complex components}\big)
\end{aligned}
$$

Note that W is not necessarily unique, since it depends on the order of evaluation of the w, v pairs. Typically, this depends on the choice of a topological sort of the elements in $W^{(i+1)}$.

Figure 4.8 An MRTS with its recurrence structure digraph.


(a) MRSA.
$\mathcal{S}^{\mathrm{D1}}$
$\mathcal{S}^{\mathrm{D2}}$
$\mathcal{S}^{\mathrm{E}}$
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8
$\mathbf{m}$ $\bar{\mathbf{m}}$ δ
(b) Digraph of the SCCs.
$\mathcal{S}_{\mathrm{T}_1} = \{s_0\}$ | $\mathcal{S}^{\mathrm{D1}}$ | ③
$\mathcal{S}_{\mathrm{T}_2} = \{s_1\}$ | $\mathcal{S}^{\mathrm{D1}}$ | ③
$\mathcal{S}_{\mathrm{T}_3} = \{s_2\}$ | $\mathcal{S}^{\mathrm{D2}}$ | ④
$\mathcal{S}_{\mathrm{T}_4} = \{s_3\}$ | $\mathcal{S}^{\mathrm{D2}}$ | ④
$\mathcal{S}_{\mathrm{T}_5} = \{s_4\}$ | $\mathcal{S}^{\mathrm{E}}$ | ①
$\mathcal{S}_{\mathrm{T}_6} = \{s_5, s_6\}$ | $\mathcal{S}^{\mathrm{E}}$ | ②
$\mathcal{S}_{\mathrm{R}_1} = \{s_7\}$ | BSCC
$\mathcal{S}_{\mathrm{R}_2} = \{s_8\}$ | BSCC
(c) Digraph of aggregated SCCs.
$\mathcal{S}_{w_1} = \{s_0, s_1\}$ | $\mathcal{S}^{\mathrm{D1}}$ | ③
$\mathcal{S}_{w_2} = \{s_2, s_3\}$ | $\mathcal{S}^{\mathrm{D2}}$ | ④
$\mathcal{S}_{w_3} = \{s_4\}$ | $\mathcal{S}^{\mathrm{E}}$ | ①
$\mathcal{S}_{w_4} = \{s_5, s_6\}$ | $\mathcal{S}^{\mathrm{E}}$ | ②
$\mathcal{S}_{\mathrm{R}_1} = \{s_7\}$ | BSCC
$\mathcal{S}_{\mathrm{R}_2} = \{s_8\}$ | BSCC

The initial poset $W^{(0)}$ and the aggregated poset W are:

$$
\begin{aligned}
W^{(0)} &= \Big\{ \{\mathcal{S}_{\mathrm{T}_1}\}, \{\mathcal{S}_{\mathrm{T}_2}\}, \{\mathcal{S}_{\mathrm{T}_3}\}, \{\mathcal{S}_{\mathrm{T}_4}\}, \{\mathcal{S}_{\mathrm{T}_5}\}, \{\mathcal{S}_{\mathrm{T}_6}\} \Big\}, \\
W &= \Big\{ \underbrace{\{\mathcal{S}_{\mathrm{T}_1}, \mathcal{S}_{\mathrm{T}_2}\}}_{w_1}, \underbrace{\{\mathcal{S}_{\mathrm{T}_3}, \mathcal{S}_{\mathrm{T}_4}\}}_{w_2}, \underbrace{\{\mathcal{S}_{\mathrm{T}_5}\}}_{w_3}, \underbrace{\{\mathcal{S}_{\mathrm{T}_6}\}}_{w_4} \Big\}
\end{aligned}
$$

and the aggregated components have the following types: w_1 is of type ③ and the outgoing probability vector $\boldsymbol{\mu}^{\mathcal{D}}(w_1)$ does not require the inversion of $\mathbf{T}_{w_1}$, moreover the augmented set of $\mathcal{S}_{\mathrm{T}_1}$ is $\{s_0, s_1\}$, so it is really convenient to aggregate $\mathcal{S}_{\mathrm{T}_1}$ and $\mathcal{S}_{\mathrm{T}_2}$. Component w_2 is of type ④, due to the self-loop of $\bar{\mathbf{q}}$ transitions over s_2 and s_3. The computation of $\boldsymbol{\mu}^{\mathcal{D}}(w_2)$ requires the solution of an MRP with the implicit method (which is iterative). Component w_3 is of type ① since there is no internal $\mathbf{q}$ transition, so the solution cost is that of a matrix-vector multiplication. Component w_4 is of type ②, which leads to the iterative solution of a CTMC. The two bottom SCCs $\mathcal{S}_{\mathrm{R}_1}$ and $\mathcal{S}_{\mathrm{R}_2}$ are composed of a single state, so no additional computation is required.

4.4.5 Backward component method

Section 4.4 has shown how to compute the forward steady state probability of the EMC $\mathbf{P}$ in implicit form with the component method, by computing a sequence of outgoing probability vectors $\boldsymbol{\mu}^{\mathcal{D}}(w)$. It remains to show that the same foundation can be used to compute the backward solution $\boldsymbol{\xi}^{\mathcal{D}}(\rho)$ for transient and recurrent states. The key variation is that the component sequence W computed with algorithm 4.6 has to be followed in reverse order, from the recurrent classes up the initial transient sets.

Given a target measure vector $\boldsymbol{\rho}$, and all the stationary backward vectors $\boldsymbol{\xi}^{\mathcal{D}}_{\mathrm{R}_i}(\boldsymbol{\rho})$ of all the recurrent classes, and the ordered sequence of components W, let $\boldsymbol{\nu}^{\mathcal{D}}(w_j)$ be the measure vector ingoing the states of the component $\mathcal{S}_{w_j}$, back from the recurrent classes, defined as:

$$\boldsymbol{\nu}^{\mathcal{D}}(w_j) \overset{\text{def}}{=} (\mathbf{I}-\mathbf{T}_{w_j})^{-1} \cdot \mathbf{F}_{w_j} \cdot \Big(\textstyle\sum_{i>j} \boldsymbol{\nu}^{\mathcal{D}}_{w_j}(w_i) + \boldsymbol{\xi}^{\mathcal{D}}_{\mathrm{R}}(\boldsymbol{\rho}) \Big) \tag{4.45}$$

Let $\boldsymbol{\nu}^{\mathcal{D}}_{w_j}(w_i) = \mathbf{I}_{w_j} \cdot \boldsymbol{\nu}^{\mathcal{D}}(w_i)$ be the ingoing quantity that reaches the component $\mathcal{S}_{w_j}$ from the component $\mathcal{S}_{w_i}$, with $i > j$. The backward vector $\boldsymbol{\xi}^{\mathcal{D}}_{\mathrm{R}}(\boldsymbol{\rho})$ can be computed, in isolation, with (2.14).

Each vector $\boldsymbol{\nu}^{\mathcal{D}}(w_j)$ obtained with (4.45) depends on all the successive $\boldsymbol{\nu}^{\mathcal{D}}_{w_j}(w_i)$, in topological order W. Therefore, the backward vector of the first transient component $\boldsymbol{\nu}^{\mathcal{D}}(w_1)$ requires the ingoing backward measure of all the components of the Markov process. As before, each component can be classified into one of the five kinds listed in Table 4.3. The class is a structural property of each component, and can be exploited for computing both the forward and the backward solution. The measure (4.45) has the property: $\boldsymbol{\nu}^{\mathcal{D}}_{C}(A \cup B) = \boldsymbol{\nu}^{\mathcal{D}}_{C}(A) + \boldsymbol{\nu}^{\mathcal{D}}_{C}(B)$, for mutually disjoint sets A, B and C.

The backward solution vector $\boldsymbol{\xi}^{\mathcal{D}}_{\mathrm{T}}$ can then be obtained from the backward component method as:

$$\boldsymbol{\xi}^{\mathcal{D}}_{\mathrm{T}} = \sum_{w \in W} \Big(\mathbf{I}_w \cdot \boldsymbol{\nu}^{\mathcal{D}}(w) \Big) \tag{4.46}$$

The backward solution with the implicit component method is then a variation of Algorithm 4.5 which is summarized in Algorithm 4.7.

4.4.6 Experimental assessment of the component method

In this section the costs of the component method and of the matrix-free MRP methods are compared. Firstly, the asymptotical costs are provided, to show what are the expected costs. Secondly, the actual cost of solving a set of reducible DSPNs is reported.

Algorithm 4.7 Backward component-based solution of $\mathbf{P}$

```
1. Derive the recurrent structure of $\mathbf{P}$ from $\mathbf{Q}, \bar{\mathbf{Q}}$ and $\boldsymbol{\Delta}$.
2. With algorithm 4.6 obtain the component sequence $W$.
3. Compute the backward solution $\boldsymbol{\xi}^{\mathcal{D}}_{\mathrm{R}_i}(\boldsymbol{\rho})$ of each recurrent component $\mathcal{S}_{\mathrm{R}_i}$
   with (2.14).
4. Compute each ingoing measure $\boldsymbol{\nu}^{\mathcal{D}}(w)$ with (4.45) in reverse order of $W$.
   Each component can be classified with the taxonomy of Table 4.3 and
   computed with the appropriate optimized method.
5. Compute the backward solution vector of the transient set with (4.46).
```

Asymptotical costs

Let us assume that the MRP being solved has K components, and that $W_{(1)}$ to $W_{(5)}$ are the sets of the five different types of *simple* and *complex* components. As before, let N and η be the number of states and transitions of the MRTS (N_i, η_i for a component i), R (R_i) the truncation point used in the uniformisation procedure, and M (M_i) the number of iterations required for convergence when an iteration method is used (for example for steady-state analysis of the recurrent components). All methods share the cost for storing the MRTS, which is in the order of the non-null entries of $\mathbf{Q} + \bar{\mathbf{Q}} + \boldsymbol{\Delta} = O(\eta)$, and (at least) one probability vector (of size N). Similarly, all methods require an SCC decomposition which can be done with Tarjan's algorithm [Tar71] at the time cost of a traversal of the matrix digraph of the MRTS, which is $O(\eta + N)$, with an additional space cost of K (the number of SCCs). The aggregation heuristic required by the component-based method costs $O(K \cdot \log K)$.

Table 4.4 lists the time and space complexities of the explicit, implicit and component-based methods, all for the non-ergodic case. For "component method" we mean both the forward and the backward variant, which have the same asymptotical complexity. Let us consider them one by one.

The explicit method requires the computation of the EMC matrix $\mathbf{P}$, which is a (possibly) dense matrix with a space occupation of $O(N^2)$. The construction of $\mathbf{P}$ is dominated by the computation of the matrix exponentials $\boldsymbol{\Omega}$ and $\boldsymbol{\Psi}$, which have a cost of $O(N\eta R)$. If we indicate with a subscript T and R_i the quantities that refer to the transient and to the recurrent sets, we can state that the explicit solution of $\mathbf{P}$ has the cost of a non-ergodic DTMC solution, based on an iterative method, leading to $O(M_\mathrm{T} N_\mathrm{T}^2 + \sum_{\mathrm{R}_i} M_{\mathrm{R}_i} N_{\mathrm{R}_i}^2)$, where the vector by matrix multiplication cost is equal to the number of non-null elements in $\mathbf{P}$ (in the order of N^2), and the M_i may vary significantly depending on the numerical solution used (Richardson 4.1, FGMRES 4.2), on the preconditioning and on the conditioning of the coefficient matrix $\mathbf{P}$.

The time complexity of the implicit method is described in [Ger01] and has been reported in this thesis in (4.26); for what concerns the non-ergodic case the method computes separately the $\mathbf{u}\boldsymbol{\Omega}$ and $\mathbf{u}\boldsymbol{\Psi}$ products for the transient and the recurrent parts, the cost of which is dominated by the cost of the uniformization on the $\mathcal{S}^\mathrm{G}$ state subset. The total solution cost is therefore $O(M_\mathrm{T}\eta_\mathrm{T} R_\mathrm{T} + \sum_{\mathrm{R}_i} M_{\mathrm{R}_i}\eta_{\mathrm{R}_i} R_{\mathrm{R}_i})$, and the space cost is that of the $\mathbf{u}$ vector.

The component-based method has a cost that is highly dependent on the number of simple or complex components of the five different types. The cost of the solution of each component has been defined in the previous section, and it differs greatly depending on the component type.

Table 4.4 Time and space complexity of the three steady-state MRP methods.

Algorithm	Time complexity	Space
SCC for $W^{(0)}$	$O(\eta + N)$	$O(N{+}K)$
Aggreg. heuristic W	$O(K \log K)$	$O(K)$
Uniformization	$O(\eta R)$	$O(N)$
Explicit MRP solution	$O(N\eta R + M_\mathrm{T} N_\mathrm{T}^2 + \sum_{\mathrm{R}_i} M_{\mathrm{R}_i} N_{\mathrm{R}_i}^2)$	$O(N^2)$
Implicit MRP solution	$O(M_\mathrm{T}\eta_\mathrm{T} R_\mathrm{T} + \sum_{\mathrm{R}_i} M_{\mathrm{R}_i}\eta_{\mathrm{R}_i} R_{\mathrm{R}_i})$	$O(N)$
Component method	$O\Big(\sum_{i\in W_{(1)}} \eta_{\mathrm{T}_i} + \sum_{i\in W_{(2)}} M_{\mathrm{T}_i}\eta_{\mathrm{T}_i} + \sum_{i\in W_{(3)}} \eta_{\mathrm{T}_i} R_{\mathrm{T}_i} + + \sum_{i\in W_{(4,5)}} M_{\mathrm{T}_i}\eta_{\mathrm{T}_i} R_{\mathrm{T}_i} + \sum_{\mathrm{R}_i} M_{\mathrm{R}_i}\eta_{\mathrm{R}_i} R_{\mathrm{R}_i}\Big)$	$O(N{+}K)$

A comparison of the time costs of the three methods shows very clearly the high cost of the explicit method (a quadratic cost N^2 induced by the explicit construction of $\mathbf{P}$). It also shows that all methods depends (although in different ways) on the number of iterations M_i. Since this value cannot be determined a priori, any solution cost that depends on M_i cannot be anticipated. A second observation is the distinction between the implicit method and the implicit component-based method: the difference in cost is clearly on the transient portion (the recurrent components are solved in the same way). If only simple ① and ③ components are present then the method does not need an iterative solution for the transient portion of the process, which may significantly decrease the solution cost. On the opposite side, if all components are complex (type ④ and ⑤) then the two methods collapse to the same one (if all complex components are aggregated into a single, large component equivalent to $\mathcal{S}_\mathrm{T}$).

Empirical assessment

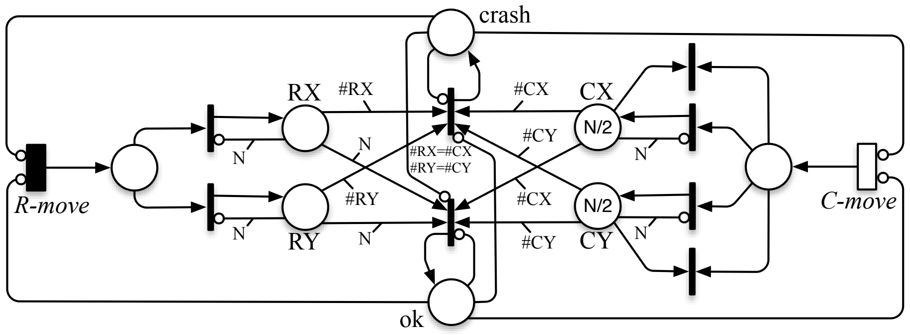
We report the empirical test of [AD12a]. The tests concerns three DSPN models: the first two are application-oriented (a robotic grid world and a manufacturing system), and the third one is built artificially to experiment the impact of rates on component aggregation.

All solutions have been computed using the DSPN-Tool [AD10b], running on an Intel 2Ghz Core Duo machine. The tool implements the implicit, the explicit and the component-based methods in a fairly optimized way.

Grid World model

This first example aims at assessing the impact of simple and complex components. The two types of MRTSs considered are generated from a DSPN model of a robot that moves in a $N \times N$ grid. The robot starts from the bottom-left corner of the grid and has to reach the opposite corner. The robot moves one cell at a time, only to the right or upward, hence, the goal cell is reached in $(2N-1)$ steps. Inside the grid world there is also a chaser, which moves randomly in each direction. The system evolution ends when either the chaser reaches the robot, or the robot arrives at its goal cell. Two different settings are considered: the delay of a robot move is deterministic and the delay of a chaser move is exponentially distributed (results in Table 4.5), and the opposite case (Table 4.6). Figure 4.9 shows the DSPN of this test for the first setting. Places RX,RY and CX,CY contains the discrete coordinates of the robot and of the chaser. The two transitions R-move and C-move trigger one step of the robot and of the chaser, respectively, while the two places "ok" and "crash" get a token when either the robot reaches the final cell or when the chaser reaches the robot.

Figure 4.9 The GridWorld DSPN.



Results are reported for a variable size N of the grid. In the first setting the generated MRTSs have no complex components, while in the second one all components but one (the unique BSCC) are complex.

Table 4.5 Explicit, implicit and component methods for the reducible Grid-World model with deterministic robot moves.

MRSA			Explicit			Implicit (Power Method)		Implicit (GMRES)		Implicit (preconditioned)		Comp. based (aggregate S)			Comp. based (no aggregate)		
N	states	trns	EMC	Time	Iter.	Time	Iter.	Time	Iter.	Time	Iter.	Time	S	C	Time	S	C
4	227	1 036	7 440	0.10	7	0.03	12	0.02	7	0.06	8	0.007	7	0	0.009	17	0
5	578	2 804	33 333	0.70	8	0.09	16	0.06	9	0.18	9	0.02	9	0	0.02	26	0
6	1 227	6 184	108 772	3.00	8	0.22	18	0.15	11	0.40	9	0.04	11	0	0.05	37	0
7	2 306	11 932	287 323	10.74	9	0.47	21	0.33	13	0.81	10	0.07	13	0	0.11	50	0
8	3 971	20 948	646 856	31.31	9	0.89	23	0.64	15	1.50	10	0.13	15	0	0.23	65	0
9	6 402	34 276	1 286 586	80.45	10	1.55	25	1.16	17	2.58	11	0.21	17	0	0.43	82	0
10	9 803	53 104	2 319 724	187.02	10	2.68	28	1.95	19	4.03	11	0.35	19	0	0.76	101	0
11	14 402	78 764	3 864 764	402.06	10	4.23	30	3.15	21	6.06	11	0.53	21	0	1.28	122	0
12	20 451	112 732	6 051 704	809.76	11	6.37	32	4.86	23	9.03	12	0.79	23	0	2.07	145	0
15	82 946	468 932						50.01	62			3.47	33	0	14.85	290	0
20	233 291	1 337 032						210.50	94			9.65	43	0	66.01	485	0
25	529 986	3 063 532						808.49	161			25.00	53	0	219.51	730	0

Table 4.5 reports the cost for the computation in terms of: size N of the grid, size of the MRTS (number of states and transitions), performance of the explicit method (size of the EMC, total execution time and number of iterations of the numerical solver), performance of the implicit method in three variations (Power method, GMRES and preconditioned GMRES) in terms of total execution time and number of iterations, performance of the component method (Algorithm 4.5) in terms of total execution time and number of simple/complex component,

Table 4.6 Explicit, implicit and component methods for the reducible GridWorld model with deterministic chaser moves.

MRSA			**Explicit**			**Implicit** (Power Method)		**Implicit** (GMRES)		**Implicit** (preconditioned)		**Comp. based** (aggregate C)			**Comp. based** (no aggregate)		
N	**states**	**trns**	**EMC**	**Time**	**Iter.**	**Time**	**Iter.**	**Time**	**Iter.**	**Time**	**Iter.**	**Time**	**S**	**C**	**Time**	**S**	**C**
4	227	1036	4760	0.09	12	0.05	27	0.03	16	0.08	9	0.04	1	1	0.15	1	15
5	578	2804	19278	0.58	13	0.15	31	0.09	18	0.22	10	0.11	1	1	0.62	1	24
6	1227	6184	59317	2.45	12	0.34	34	0.22	20	0.50	10	0.25	1	1	1.95	1	35
7	2306	11932	151501	8.64	15	0.72	38	0.45	22	1.00	11	0.50	1	1	5.18	1	48
8	3971	20948	334189	25.43	16	1.33	41	0.85	24	1.84	12	0.93	1	1	12.18	1	63
9	6402	34276	656665	65.03	17	2.30	44	1.42	25	3.02	12	1.55	1	1	25.59	1	80
10	9803	53104	1176807	150.17	17	3.85	47	2.33	27	4.89	13	2.54	1	1	49.90	1	99
11	14402	78764	1957237	322.65	18	5.99	50	3.69	29	7.31	13	3.98	1	1	99.83	1	120
12	20451	112732	3061005	648.06	19	8.98	53	5.42	30	10.88	14	5.84	1	1	156.65	1	143
15	82946	468932						32.40	45			32.69	1	1			
20	233291	1337032						138.35	70			138.45	1	1			
25	529986	3063532						523.31	119			523.63	1	1			

with and without the aggregation of components of Algorithm 4.6. The iterative solution required by components of type ②, ④ and ⑤ is based on GMRES.

Since the robot moves are deterministic, the number of components is equal to the length $(2N-1)$ of the paths in the grid from the initial to the goal cell. All components are simple since the firing of a deterministic event (robot move) always lead to the next component (robot one step closer to the goal, or robot chased). It is clear from the results reported in the table that in this case (but also in all other cases that we have experimented), the explicit construction and solution of the embedded Markov chain has a very detrimental impact on the performance, both in terms of space and time, making impractical experiments for $N \geq 13$. The three variations of the implicit solution show a quite consistent behavior. The preconditioner $\mathbf{M}$ used is the problem-specific preconditioner for MRPs described in section 4.3. While preconditioning reduces the number of iterations, the overall cost of building the preconditioner matrix makes this method unattractive for this specific case.

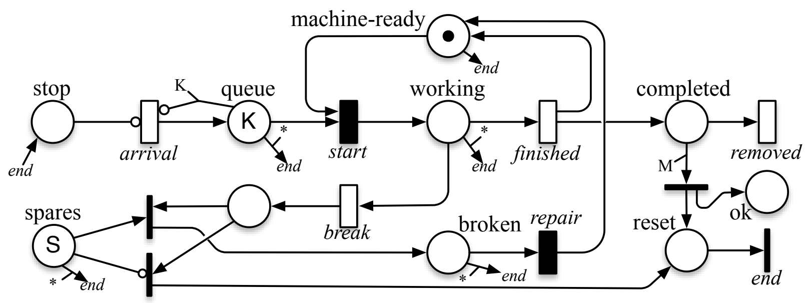
The last three lines of the table show the direct comparison of the two fastest methods for values of N larger than 13, showing the different growing rates of both, which indicates a clear advantage of the component method in this setting. It is important to remember that the implicit method goes through a fixed point iteration (around Eq. 4.21), while the component method, in this specific case in which all components are simple of type ③, executes a fixed number of transient solutions with the uniformisation technique (which is not fixed point).

Table 4.6 shows the results of the GridWorld model when the chaser moves are deterministic, and the robot moves are exponential: in this case there are (N^2-1) complex components. Implicit and explicit methods behave roughly as in the first setting. The last two columns show the advantage of aggregating complex components: in this case there are (N^2-1) complex components and without aggregation the method performs much worse than the implicit one. This happens because the augmented sets include all states until an absorbing state is reached: indeed from each state there is a path made only of $\mathbf{q}$ transitions (the robot moves) that ends into either the goal state or the failure state. As a consequence, the augmented state space of the first component includes the states of the remaining (N^2-2) components: this is the worst setting for the component method without aggregation, since the cost for the solution of a single component is equal to the cost of the whole transient set.

Production machine model

The second example experiments with a model characterized by simple and complex components, structured in such a way that Algorithm 4.6 performs no aggregation. The system is a production line with a single machine that may occasionally break down, modelled by the DSPN of Fig. 4.10. The goal is to compute the probability of completing M pieces, without having to interrupt the production, knowing that the factory has at most S spare parts of the machine and that pieces completed by the machine may be randomly removed. A piece is generated by transition *arrival* into the K-limited buffer modeled by place *queue*, the piece is then moved into a working area by the deterministic transition *start*, given that the machine is ready. The piece is worked by the machine while in the *working* place, and it is then moved to an output buffer, modeled by place *completed*. Pieces may be randomly removed from the output buffer by transition *removed*, while as soon as there are M pieces in the buffer (success condition), the place *ok* is marked, and a reset procedure (place *reset* marked) is activated, as the firing of transition *end* takes the net to a state in which no transition is enabled, and therefore no evolution is possible. Note that each arc entering transition *end* is marked with a $*$ to mean that all tokens in the input places should be removed when the transition fires. The machine may break down while working (exponential transition *break*) and can be repaired in within a deterministic delay (transition *repair*), by using a spare part. If all spares have been consumed (place *spares* empty) a token is put in place *reset*, and the reset procedure of transition *end* is activated.

Figure 4.10 Petri Net of the Production Machine model.



Since we have at most S spare parts, the number of *repair* actions is limited. However, an unbounded number of firings of the *start* transition is still possible. The component structure of this MRTS follows that of the consumption of the spare parts. Each repair activity moves the Petri Net in a new component of the state space with s spares, where any state with s+1, or more, spares will never be reachable again. These components are complex, since the deterministic *start* transition may fire several times before the number of parts decreases. Note that each pair of complex components (for s and $s+1$ spares) is separated by a simple component, delimited by the firing of transition *repair*, so no aggregation is possible.

The experiments were conducted for $S = 4$ and $S = 8$, for varying values of M and K, with $M = K$. Results are reported in Table 4.7. The overall system happens to be slightly ill-conditioned, resulting in a large number of GMRES iterations needed to achieve the requested accuracy of 10^{-7}. The table reports also the mean number of iterations for the convergence of the solution of each complex component. To investigate the behavior of convergence rates, both methods are tested with and without the preconditioner.

Table 4.7 Tests with 4 and 8 spare parts and various production goals M.

System			**Implicit**				**Component based**					
(Spares = 4)			(GMRES)		(precond.)		(GMRES)				(precond.)	
M/K	**states**	**trns**	**Time**	**It**	**Time**	**It**	**Time**	**It**	**S**	**C**	**Time**	**It**
5	336	973	0.44	50	0.12	7	0.07	23	4	4	0.07	7
10	1 221	3 808	3.66	115	0.59	8	0.55	63	4	4	0.30	7
15	2 656	8 493	14.03	203	1.46	8	2.78	149	4	4	0.76	7
20	4 641	15 028	30.94	256	2.71	8	5.66	155	4	4	1.48	8
25	7 176	23 413	53.47	286	4.35	8	10.90	245	4	4	2.64	8
30	10 261	33 648	72.23	270	6.39	8	16.94	243	4	4	4.18	8
35	13 896	45 733	127.85	353	9.34	9	25.12	249	4	4	6.23	8
40	18 081	59 668	190.42	404	12.34	9	38.29	278	4	4	8.79	8
45	22 816	75 453	240.41	404	15.83	9	48.28	261	4	4	12.32	9
50	28 101	93 088	338.65	462	19.72	9	69.52	299	4	4	16.59	9

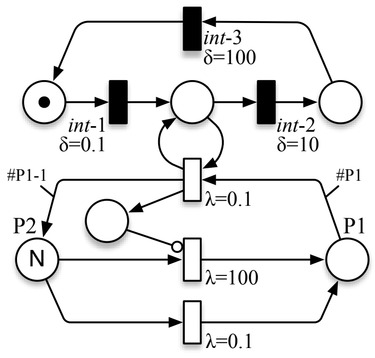
System			**Implicit**				**Component based**					
(Spares = 8)			(GMRES)		(precond.)		(GMRES)				(precond.)	
M/K	**states**	**trns**	**Time**	**It**	**Time**	**It**	**Time**	**It**	**S**	**C**	**Time**	**It**
5	696	2 021	1.05	57	0.26	7	0.15	23	8	8	0.13	7
10	2 541	7 916	7.35	110	1.30	8	1.15	63	8	8	0.62	7
15	5 536	17 661	30.24	208	3.17	8	6.25	172	8	8	1.57	7
20	9 681	31 256	64.38	253	5.90	8	12.01	171	8	8	3.06	7
25	14 976	48 701	122.25	309	9.54	8	22.53	245	8	8	5.47	8
30	21 421	69 996	161.03	285	14.19	8	35.77	278	8	8	8.78	8
35	29 016	95 141	260.68	341	19.45	8	52.37	340	8	8	13.18	8
40	37 761	124 136	401.44	403	27.11	9	74.92	297	8	8	18.85	8
45	47 656	156 981	500.52	397	34.91	9	97.74	261	8	8	26.38	8
50	58 701	193 676	676.09	434	43.55	9	132.46	299	8	8	35.51	8

In this model, each component-based solution requires S simple and S complex components. Note that also in this case the component method performs better than the implicit one, but the advantage is less significant than in the Grid case, since the cost is dominated by the solution of the S complex components, which cannot be aggregated into a single larger one.

Alternating timed model

To investigate whether there is any disadvantage in the aggregation, we have built a very simple DSPN (Fig. 4.8, left) where N tokens flow from place P2 to place P1. A cyclic set of 3 phases enables or disables the transition that moves back, from P1 to P2, all tokens except one, which is lost in the process, leaving a token in an intermediate place.

Table 4.8 A simple alternating timed model, with 3 time phases.



MRSA			Comp. based (aggregate C)			Comp. based (no aggregate)		
N	states	trns	Time	S	C	Time	S	C
2	18	36	1.44	0	1	0.98	0	7
4	45	97	10.18	0	1	3.98	0	15
6	84	186	32.55	0	1	9.91	0	28
8	135	303	77.64	0	1	19.73	0	45
10	198	448	156.80	0	1	34.46	0	66

This net has a significant difference of time scales between its transitions (four orders of magnitude), and indeed the results of the table in Fig. 4.8 suggest that in this case it is better not to aggregate components, to avoid the creation of components with very different transition rates, which may increase significantly the number of uniformisation steps R at each iteration, leading to a degradation of the performance. This could lead to the inclusion in Algorithm 4.6 of a more sophisticated criteria (based, for example, on the min/max rates of the components and on the covering of the augmented sets) to prevent aggregation from taking place.

Chapter 5

Improving the Model Checking of CSL$^{\mathrm{TA}}$

The previous chapters provide the foundations of path-based rewards, described with temporal logics, and the numerical methods used to evaluate these logics. We now describe some of the problems that affect the model checking procedure of CSL$^{\mathrm{TA}}$, described in section 3.4.2. In particular, the focus will be on the efficiency of CSL$^{\mathrm{TA}}$ when the path formulas modeled with DTAs are that of CSL (Until and Next). Since CSL$^{\mathrm{TA}}$ constructs an MRP as a solution process, there is an additional complexity in the model checking procedure, in contrast with the much more optimized solution method of CSL (see section 3.3.2). We show that this problem can be tackled efficiently with the component method. Section 5.1.1 illustrates this solution. Finally, some thoughts on the space occupation of CSL$^{\mathrm{TA}}$ are presented in section 5.1.3, along with a possible solution.

These two improvements represent the main contribution of this thesis for CSL$^{\mathrm{TA}}$. The usage of the component method has been also published in [AD10a] and revised in [AD12a].

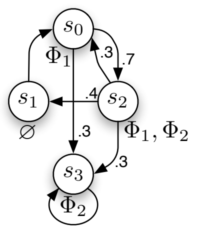
5.1 Performance mismatch of CSL and CSL$^{\mathrm{TA}}$

This section describes why CSL$^{\mathrm{TA}}$ and CSL have a discrepancy in the way they model check the same formula, and what can be done to improve the former. Recall that in CSL$^{\mathrm{TA}}$ a path formula is expressed using a Timed Automaton with a single clock [DHS09], described in Def. 19, and that the model checking of CSL$^{\mathrm{TA}}$ formulas requires the steady-state solution of a reducible MRP. CSL instead has only predefined path operators (Until and NeXt), and the model checking of these formulae can be done with a predefined sequence of CTMC functions. We shall consider only the Until formula, which is the most expensive CSL formula to check. Until formula requires the transient/stationary solution of two CTMCs [Bai+03], given in (3.10) and (3.11), in which the solution of the first one is used as initial distribution for the second one. This results in a substantial difference in the model checking times for the two cases.

Consider the CMTS $\mathcal{M}$ given in Figure 5.1, and the CSL Until formula:

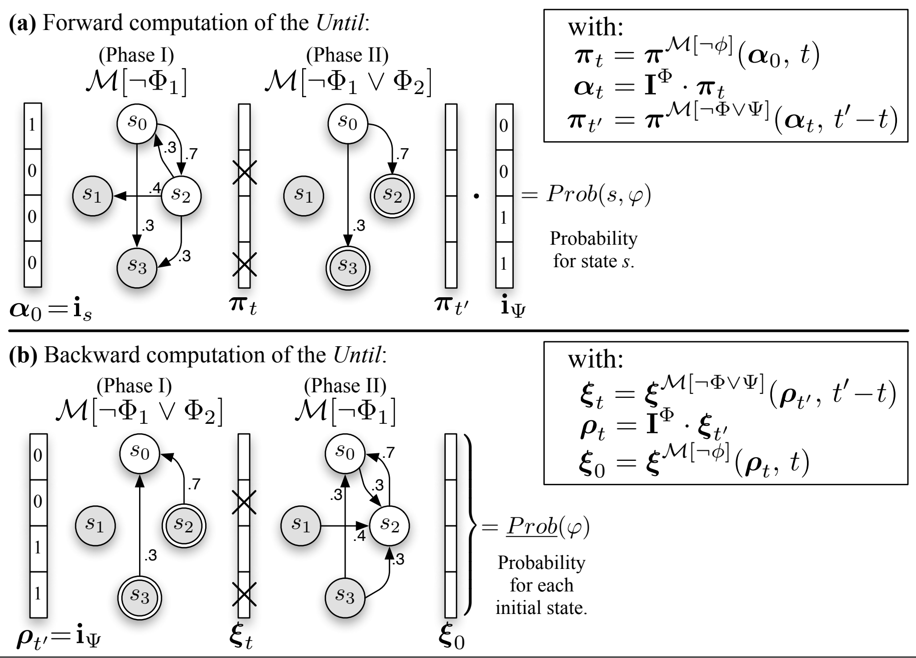
$$\Theta \ ::= \ \mathcal{P}_{\bowtie\lambda}(\Phi_1 \, \mathcal{U}^{[t,t']} \, \Phi_2)$$

Figure 5.1 Example of CMTS model for a CSL/CSL$^{\mathrm{TA}}$ comparison.



The computation of Θ can be done with forward and backward formulas. Figure 5.2 illustrates the various phases of the formulas (3.10) and (3.11) when applied to the model $\mathcal{M}$.

Figure 5.2 Forward and backward computations of the CSL formula Θ.


(a) Forward computation of the *Until*:
(Phase I)
$\mathcal{M}[\neg\Phi_1]$
(Phase II)
$\mathcal{M}[\neg\Phi_1 \vee \Phi_2]$
1
0
0
0
s_0
s_1
s_2
s_3
.3
.7
.4
.3
.3
0
0
1
1
$= Prob(s, \varphi)$
Probability for state s.
$\boldsymbol{\alpha}_0 = \mathbf{i}_s$
$\boldsymbol{\pi}_t$
$\boldsymbol{\pi}_{t'}$
$\mathbf{i}_\Psi$
with:
$\boldsymbol{\pi}_t = \boldsymbol{\pi}^{\mathcal{M}[\neg\phi]}(\boldsymbol{\alpha}_0, t)$
$\boldsymbol{\alpha}_t = \mathbf{I}^{\Phi} \cdot \boldsymbol{\pi}_t$
$\boldsymbol{\pi}_{t'} = \boldsymbol{\pi}^{\mathcal{M}[\neg\Phi\vee\Psi]}(\boldsymbol{\alpha}_t, t'-t)$
(b) Backward computation of the *Until*:
(Phase I)
$\mathcal{M}[\neg\Phi_1 \vee \Phi_2]$
(Phase II)
$\mathcal{M}[\neg\Phi_1]$
0
0
1
1
$= \underline{Prob}(\varphi)$
Probability for each initial state.
$\boldsymbol{\rho}_{t'} = \mathbf{i}_\Psi$
$\boldsymbol{\xi}_t$
$\boldsymbol{\xi}_0$
with:
$\boldsymbol{\xi}_t = \boldsymbol{\xi}^{\mathcal{M}[\neg\Phi\vee\Psi]}(\boldsymbol{\rho}_{t'}, t'-t)$
$\boldsymbol{\rho}_t = \mathbf{I}^{\Phi} \cdot \boldsymbol{\xi}_{t'}$
$\boldsymbol{\xi}_0 = \boldsymbol{\xi}^{\mathcal{M}[\neg\phi]}(\boldsymbol{\rho}_t, t)$

The boxes on the right of Figure 5.2 are the fragments of the forward and backward versions of: $Prob^{\mathcal{M}}(s, \Phi_1\ \mathcal{U}^{[t,t']}\ \Phi_2)$. Note that each phase computes a transient measure on a *modified* CTMC which is shown by making gray and absorbing the states that do not satisfy the specified state proposition expression. In the backward case, the transition arrows are drawn reversed, since the computation is done with the transpose of $\mathbf{Q}$.

For instance, the forward case **(a)** requires the transient solution of two modified CTMCs: at time t for the chain $\boldsymbol{\pi}^{\mathcal{M}[\neg\Phi_1]}$, assuming we start in s at time 0, and at time $t' - t$ for the chain $\boldsymbol{\pi}^{\mathcal{M}[\neg\Phi_1\vee\Phi_2]}$, assuming we start with a probability vector resulting from the previous computation. Note that the result of the first computation is filtered out using the $\mathbf{I}^{\Phi_1}$ vector, to put to zero the probability of all states which are not $Sat(\Phi_1)$ states at time t: as a consequence the second transient analysis starts from an initial vector that does not sum up to one. The elements of the second transient solution vector, $\boldsymbol{\pi}_{t'}$ that satisfy Φ_2 are then summed-up to obtain $Prob(s, \varphi)$.

The CSL query Θ can be formulated using the temporal logic CSL$^{\text{TA}}$:

$$\Theta' \ ::= \ \mathcal{P}_{\bowtie\lambda}\big(\mathit{Until}[\Phi_1, \Phi_2, t, t']\big)$$

where *Until* is the name of the DTA given in Figure 3.1(d), and Φ_1, Φ_2, t and t' are the DTA parameters.

Figure 5.3 State space construction of the CSL$^{\text{TA}}$ formula Θ'.

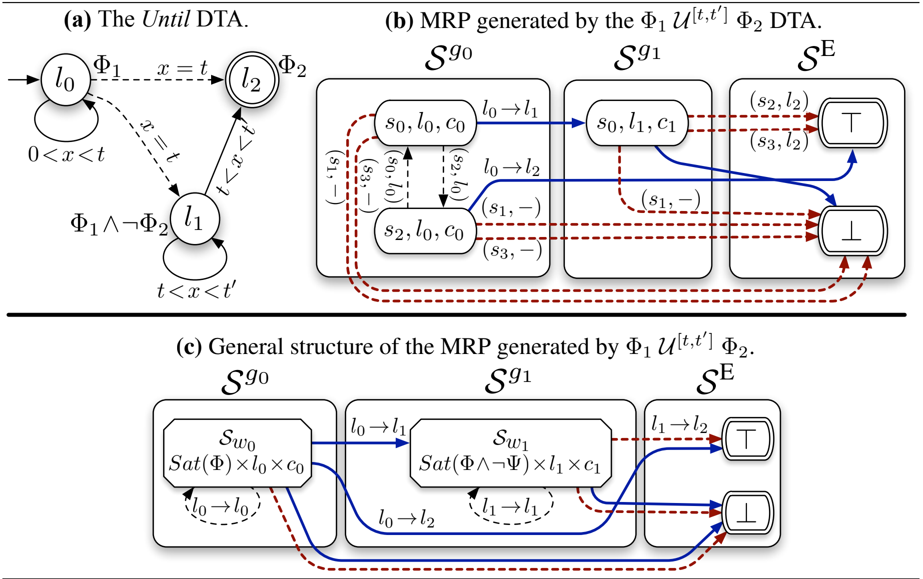


Figure 5.3 shows the *Until* DTA and the resulting MRTS process $\mathcal{M}\times\mathcal{A}$ with the model of Figure 5.1. It also shows the general structure of the $\mathcal{M}\times\mathcal{A}$ Until process for any CMTS model. The clock zones of the DTA are identified as: $\delta_{c_0} = [0, t)$, $\delta_{c_1} = [t, t')$, $\delta_{c_2} = [t', \infty)$. The MRP construction then associates to each finite zone (δ_{c_0} and δ_{c_1} in this case) a deterministic event in the MRP, called g_0 and g_1, of duration $(t - 0)$ and $(t' - t)$ respectively. The forward/backward computation of the probability of the set of accepted paths is then reduced to the computation of the probability of reaching the absorbing state $\top$ from the initial states with (3.13) or (3.14).

Comparing the CSL model checking process for the Until summarized by Fig. 5.2 and that for CSL$^{\text{TA}}$ on the same formula summarized in Fig. 5.3, identifies very clearly why model checking an Until with CSL$^{\text{TA}}$ is different from that of CSL. Despite the fact that the paths that satisfy the two formulas are the same, the computational cost in CSL is lower than in CSL$^{\text{TA}}$, since CSL$^{\text{TA}}$ builds a (real) MRP process, which is slow to solve in steady-state. CSL instead just require one or two forward/backward solution, which are usually transient solutions. Of course, CSL model checkers have only a few types of path operators to verify, and can therefore build a specialized solution schema for each of them. A CSL$^{\text{TA}}$ model checker has to deal with any type of DTA and it applies a single general procedure, which computes the steady-state solution of an MRP of a size which is of the order of the size of a cross-product. This size is upper bounded by the number of states of the CMTS $\mathcal{M}$ being checked, multiplied by the number of clock zones (three for the Until DTA), multiplied by the number of locations of the DTA $\mathcal{A}$ (three). Moreover an MRP solution in general requires many transient CTMC solutions (one per subordinated chain if an explicit method is used, and two per iteration if an implicit method is preferred).

5.1.1 CSL$^{\text{TA}}$ with the component method

The MRP of the cross-product $\mathcal{M}\times\mathcal{A}$ has many interesting properties. It is reducible, with a small number of zones, and with a regular structure. This is exactly the class of MRP for which the component method described in section 4.4 is convenient. To assess the applicability of the component method, we can observe that $\overline{\mathbf{q}}$ and $\boldsymbol{\delta}$ arcs always provoke a change of partition, since in the DTA there is no clock reset nor self loops with condition $x = t$. Therefore the SCCs contained in $\mathcal{S}^{g_0}$ ($\mathcal{S}^{g_1}$) are all of type ③ and the Algorithm 4.6 aggregates them into the single component $\mathcal{S}_{w_0}$ ($\mathcal{S}_{w_1}$) of type ③. $\mathcal{S}_{w_0}$ and $\mathcal{S}_{w_1}$ are not aggregated any further since they would result in a component of type ⑤.

To understand the relationships between the component matrices and the matrices used by the CSL solution approach, we can also reason on the structure of the embedded Markov chain $\mathbf{P}$ and the generator matrices $\mathbf{Q}$, $\bar{\mathbf{Q}}$ and $\boldsymbol{\Delta}$ of the MRSA in Fig. 5.3, which have the block structure reported in Fig. 5.4, after an appropriate renumbering of the states. Different portions of $\mathbf{Q}$, $\bar{\mathbf{Q}}$ and $\boldsymbol{\Delta}$

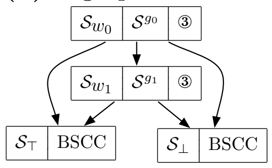
Figure 5.4 Structure of the MRP matrices for a CSL$^{\text{TA}}$ *Until*

(a) Generator matrices of the MRP:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_{l_0,l_0} & 0 & 0 & 0 \\ 0 & \mathbf{Q}_{l_1,l_1} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad \bar{\mathbf{Q}} = \begin{bmatrix} 0 & 0 & 0 & \bar{\mathbf{Q}}_{l_0,\perp} \\ 0 & 0 & \bar{\mathbf{Q}}_{l_1,\top} & \bar{\mathbf{Q}}_{l_1,\perp} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{\Delta} = \begin{bmatrix} 0 & \boldsymbol{\Delta}_{l_0,l_1} & \boldsymbol{\Delta}_{l_0,\top} & 0 \\ 0 & 0 & 0 & \boldsymbol{\Delta}_{l_1,\perp} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(b) Digraph of W:



$\mathcal{S}_{w_0}$ | $\mathcal{S}^{g_0}$ | ③
$\mathcal{S}_{w_1}$ | $\mathcal{S}^{g_1}$ | ③
$\mathcal{S}_{\top}$ | BSCC
$\mathcal{S}_{\perp}$ | BSCC

(c) Embedded Markov Chain $\mathbf{P}$ of the MRP:

$$\mathbf{P} = \begin{bmatrix} 0 & (\boldsymbol{\Omega}_{l_0,l_0}\cdot\boldsymbol{\Delta}_{l_0,l_1}) & (\boldsymbol{\Omega}_{l_0,l_0}\cdot\boldsymbol{\Delta}_{l_0,\top}) & (\boldsymbol{\Psi}_{l_0,l_0}\cdot\boldsymbol{\Delta}_{l_0,\perp}) \\ 0 & 0 & (\boldsymbol{\Omega}_{l_1,l_1}\cdot\boldsymbol{\Delta}_{l_1,\top}) & (\boldsymbol{\Psi}_{l_1,l_1}\cdot\boldsymbol{\Delta}_{l_1,\perp}) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{array}{l} \leftarrow \mathcal{S}^{g0} \\ \leftarrow \mathcal{S}^{g1} \\ \left.\begin{array}{l} \leftarrow \top \\ \leftarrow \perp \end{array}\right\} \mathcal{S}^{\text{E}} \end{array}$$

(and thus of $\mathbf{P}$) are generated by different edges of the DTA, which are written as subscript. For instance, the $\mathbf{Q}_{l_0,l_0}$ block is generated by the DTA edge $l_0 \to l_0$ with the CTMC $\mathcal{M}$ in the first clock zone δ_{c_0}. The bottom-right image of Fig. 5.4 shows the component digraph W after the aggregation. Note that the block diagonal matrices in $\mathbf{P}$ for $\mathcal{S}^{g_0}$ and $\mathcal{S}^{g_1}$ are made of zeros, which confirm the observation that the two components w_0 and w_1 are simple and of type ③.

With this structure, the MRP is well suitable for an efficient solution with the *component-based* method, since the entering probability is entirely in the $\top$ or $\bot$ state after one single firing of g_0 and one of g_1. This intuitive idea is formalized in the following theorem:

Theorem 4 (**Component-based analysis of CSL$^{\text{TA}}$ is almost as efficient as CSL**). *Let $\mathcal{M}$ be a CMTS, let $\Theta = \mathcal{P}_{\bowtie\lambda}(\Phi_1\ \mathcal{U}^I\ \Phi_2)$ be a* CSL *query, and let Θ' be the same query in* CSL$^{\text{TA}}$. *The computations of the $Prob(s, \Theta)$ (with the $\boldsymbol{\pi}^{\mathcal{M}}$ and $\boldsymbol{\xi}^{\mathcal{M}}$ operators) for* CSL *and (with the $\boldsymbol{\pi}^{\mathcal{M}\times\mathcal{A}}$ and $\boldsymbol{\xi}^{\mathcal{M}\times\mathcal{A}}$ operators) for* CSL$^{\text{TA}}$ *using the* component-based *analysis, have the same asymptotical costs.*

Proof. We prove this result for the *Until* path formula with time bound $I = [t, t']$, with t' finite, and for the forward case. The numerical analysis for CSL involves the three steps listed below, where $\boldsymbol{\pi}^{\mathcal{M}}\big(t, \mathbf{i}_s\big)$ is the solution at time t for the CTMC $\mathcal{M}$, assuming the CTMC starts in s. Let N_{Ia}, $\eta(\mathbf{Q}_{\text{Ia}})$ and R_{Ia} be the number of states, transitions and uniformisation steps of the CTMC $\mathcal{M}[\Phi_1]$. The quantity N_{IIa}, $\eta(\mathbf{Q}_{\text{IIa}})$ and R_{IIa} are the same for CTMC $\mathcal{M}[\neg\Phi_1\vee\Phi_2]$, and N_{IIIa} is the number of $Sat(\Phi_2)$-states.

	Step	Time	Space
Ia	Transient analysis at time t: $\mathbf{p}_1 \;=\; \boldsymbol{\pi}^{\mathcal{M}[\neg\Phi_1]}\big(t, \mathbf{i}_s\big)$	$O(\eta(\mathbf{Q}_{\text{Ia}}) \cdot R_{\text{Ia}})$	$O(N_{\text{Ia}})$
	Filtering $Sat(\neg\Phi_1)$-states: $\mathbf{p}'_1 \;=\; \mathbf{I}^{\Phi_1} \cdot \mathbf{p}_1$	$O(N_{\text{Ia}})$	$O(1)$
IIa	Transient analysis at time $t'-t$: $\mathbf{p}_2 \;=\; \boldsymbol{\pi}^{\mathcal{M}[\neg\Phi_1\vee\Phi_2]}\big(t'-t, \mathbf{p}'_1\big)$	$O(\eta(\mathbf{Q}_{\text{IIa}}) \cdot R_{\text{IIa}})$	$O(N_{\text{IIa}})$
IIIa	Summing up success probabilities: $Prob(s, \varphi) \;=\; \mathbf{i}_{\Phi_2} \cdot \mathbf{p}_2$	$O(N_{\text{IIIa}})$	$O(1)$

The overall time cost for a CSL $Until^{[t,t']}$ is then:

$$O\Big(\eta(\mathbf{Q}_{\text{Ia}}) \cdot R_{\text{Ia}} + N_{\text{Ia}} + \eta(\mathbf{Q}_{\text{IIa}}) \cdot R_{\text{IIa}} + N_{\text{IIIa}}\Big)$$

The component-based solution of CSL$^{\text{TA}}$ computes the outgoing probability vectors $\boldsymbol{\mu}^{\mathcal{D}}(w_0)$ and $\boldsymbol{\mu}^{\mathcal{D}}(w_1)$ from the two simple components $\mathcal{S}_{w_0}$ and $\mathcal{S}_{w_1}$ of type ③. The steps are the following:

	Time of the step	Space
0b	SCC decomposition: Cost: $O(\eta^{\mathcal{M}}\times\mathcal{A}+N^{\mathcal{M}}\times\mathcal{A}+K\log K)$	$O(N^{\mathcal{M}}\times\mathcal{A}+K)$
Ib	Outgoing probability $\boldsymbol{\mu}^{\mathcal{D}}(w_0)$ with Eq. (4.40): Cost: $O\big(\eta(\mathbf{Q}_{l_0,l_0}) \cdot R_{l_0} + \eta(\bar{\mathbf{Q}}_{l_0}) + \eta(\boldsymbol{\Delta}_{l_0})\big)$	$O(N_{l_0})$
IIb	Outgoing probability $\boldsymbol{\mu}^{\mathcal{D}}(w_1)$ with Eq. (4.40): Cost: $O\big(\eta(\mathbf{Q}_{l_1,l_1}) \cdot R_{l_1} + \eta(\bar{\mathbf{Q}}_{l_1}) + \eta(\boldsymbol{\Delta}_{l_1})\big)$	$O(N_{l_1})$
IIIb	$Prob(s, \varphi) = \mathbf{i}_{\top} \cdot \big(\boldsymbol{\mu}^{\mathcal{D}}_{\top}(w_0) + \boldsymbol{\mu}^{\mathcal{D}}_{\top}(w_1)\big)$: Cost: $O(1)$	$O(1)$

where the quantities η, N and R are defined as before but for the matrices of the two components w_0 and w_1 of the MRSA. Steps Ib and IIb rewrites to the computation of $\boldsymbol{\mu}^{\mathcal{D}}(w_0) = \mathbf{i}_s \cdot \big(\boldsymbol{\Omega}^{g0}_{\widehat{w_0}}\,\boldsymbol{\Delta}^{g0}_{\widehat{w_0}} + \boldsymbol{\Psi}^{g0}_{\widehat{w_0}}\,\bar{\mathbf{Q}}^{g0}_{\widehat{w_0}}\big)$ and $\boldsymbol{\mu}^{\mathcal{D}}(w_1) = \boldsymbol{\mu}^{\mathcal{D}}_{w_1}(w_0) \cdot \big(\boldsymbol{\Omega}^{g1}_{\widehat{w_1}}\,\boldsymbol{\Delta}^{g1}_{\widehat{w_1}} + \boldsymbol{\Psi}^{g1}_{\widehat{w_1}}\,\bar{\mathbf{Q}}^{g1}_{\widehat{w_1}}\big)$.

From the construction of $\mathcal{M}\times\mathcal{A}$ shown in Fig. 5.3 we can derive:

$$\begin{aligned}
N_{l_0} &= \big|S \times l_0 \times c_0\big| = \big|Sat(\Phi_1)\big| = N_{\text{Ia}}\\
N_{l_1} &= \big|S \times l_1 \times c_1\big| = \big|Sat(\neg\Phi_1 \vee \Phi_2)\big| = N_{\text{IIa}}\\
\eta(\mathbf{Q}_{\text{Ia}}) &= \eta(\mathbf{Q}_{l_0,l_0}) + \eta(\bar{\mathbf{Q}}_{l_0})\\
\eta(\mathbf{Q}_{\text{IIa}}) &= \eta(\mathbf{Q}_{l_1,l_1}) + \eta(\bar{\mathbf{Q}}_{l_1})\\
\eta(\boldsymbol{\Delta}_{l_0}) &= N_{l_0} = N_{\text{Ia}}\\
\eta(\boldsymbol{\Delta}_{l_1}) &= N_{l_1} = N_{\text{IIa}}
\end{aligned}$$

The R values are the same, since the exponential transitions involved in the uniformisations of Ia and Ib (IIa and IIb) are the same, so the computational costs for CSL and CSL$^{\text{TA}}$ are the same. Nevertheless, the space occupation is different, due to the necessity of step 0b of the entire product state space for the SCC analysis. More precisely, the CSL state space is S, the set of CTMC states, while for CSL$^{\text{TA}}$ the state space $\mathcal{S}$ is that of the MRSA of $\mathcal{M}\times\mathcal{A}$, and $\big|\mathcal{S}\big| \leq \big|S \times L \times C\big|$. In the *until* case $|L \times C| = 3$, so $\mathcal{S}$ could be up to 3 times larger than S. □

The proofs for the time intervals $[0,t]$, $[t,t]$, and for the backward formulas, are similar.

5.1.2 Experimental comparison of CSL and CSL$^{\text{TA}}$

In this section we shall test numerically the statement of theorem 4 with some examples to compare the performance of existing CSL and CSL$^{\text{TA}}$ model checkers on Until formulas. For CSL we have used the model checker Prism [KNP09], in its simpler form (sparse matrix representation, without decision diagrams). For CSL$^{\text{TA}}$ we have used CoDeMoC [Bar+10] and MC4CSL$^{\text{TA}}$ [AD10c]. All results are computed on a 2Ghz Intel Dual-core machine with 1GB of system memory. No multi-core parallelism is exploited.

The experiments are performed on a simple M/M/1/N queue with failure. The model has been written in the Prism process algebra language; the CTMC built by Prism have then been exported to the different input formats of CoDeMoC and MC4CSL$^{\text{TA}}$. The comparison is made on the CSL *Until* query: *true* $\mathcal{U}^I$ *failure*, with two time intervals, $I_1 = [0, 75]$ and $I_2 = [5, 75]$, appropriately translated into a CSL$^{\text{TA}}$ DTA.

To make the comparison more focused on the topic of the paper we only consider the time devoted to the numerical solution, once the CTMC is available (for CSL) or the MRP is available (for CSL$^{\text{TA}}$). Indeed the overall behavior of the three tools is somewhat different and not easily comparable. For instance, Prism builds the model from the process algebra description, and this requires a computational time that in the reported experiments is much larger than the actual solution time, while MC4CSL$^{\text{TA}}$ uses an intermediate conversion schema to a DSPN format that adds another cost to the overall solution time: this is a mere implementation choice to implement at low cost the cross product $\mathcal{M}\times\mathcal{A}$.

Table 5.1 Tool comparison for a CSL *Until*$^{[0,75]}$ query.

CTMC			Prism		MC4CSL$^{\text{TA}}$									CoDeMoC	
N	states	trns	Total time	Solut. time	$\mathcal{M}\times\mathcal{A}$ states	$\mathbf{Q}+\overline{\mathbf{Q}}+\mathbf{\Delta}$	**P**	Expl	*It*	Impl	*It*	Cmp	*S*	Prod. states	Solut. time
250	252	753	0.59	0.006	504	1 753	606	0.02	7	0.22	8	0.02	2	504	0.03
500	502	1 503	0.75	0.010	1 004	3 503	1 106	0.04	7	0.44	8	0.04	2	1 004	0.05
1 000	1 002	3 003	1.15	0.016	2 004	7 003	2 106	0.08	7	0.88	8	0.07	2	2 004	0.10
2 500	2 502	7 503	4.18	0.037	5 004	17 503	5 106	0.19	7	2.21	8	0.20	2	5 004	0.25
5 000	5 002	15 003	11.26	0.071	10 004	35 003	10 106	0.40	7	4.43	8	0.39	2	10 004	0.53
10 000	10 002	30 003	42.56	0.138	20 004	70 003	20 106	0.81	7	8.88	8	0.83	2	20 004	0.98
15 000	15 002	45 003	93.35	0.206	30 004	105 003	30 106	1.23	7	13.33	8	1.28	2	30 004	1.47

Results are reported in Table 5.1 for an increasing queue size N. The columns list the number of states and transitions of the CTMC being checked, followed by Prism results (total time and solution time), by MC4CSL$^{\text{TA}}$ results (states of the $\mathcal{M}\times\mathcal{A}$ MRP, transitions in the MRSA, size of the embedded DTMC $\mathbf{P}$, solution cost using explicit, implicit, and component methods) and by CoDeMoC results (state space size and solution time).

The first striking difference is in the size of the state spaces generated by the three tools (the one of Prism is not listed, but it is never larger than the CTMC being checked). A second observation is that Prism always performs better, which is not surprising since Prism is dedicated to CSL, but it shows an increasing slope that is similar to that of the two CSL$^{\text{TA}}$ tools. Since the time-bounded Until is computed with a single transient analysis, there are no significant differences from a finely-tuned CSL method and an automata based solution method. For the same reason there is no striking difference between explicit, implicit and component based methods, since, in this specific case, $\mathbf{P}$ is very sparse.

Table 5.2 Tool comparison for a CSL *Until*$^{[5,75]}$ query.

CTMC			Prism		MC4CSL$^{\text{TA}}$									CoDeMoC	
N	states	trns	Total time	Solut. time	$\mathcal{M}\times\mathcal{A}$ states	$\mathbf{Q}+\overline{\mathbf{Q}}+\mathbf{\Delta}$	**P**	Expl	*It*	Impl	*It*	Cmp	*S*	Prod. states	Solut. time
250	252	753	1.78	0.010	756	2 758	40 592	1.51	7	0.36	9	0.03	3	504	4.72
500	502	1 503	1.88	0.012	1 506	5 508	90 592	5.69	7	0.72	9	0.07	3	1 004	17.52
1000	1 002	3 003	1.92	0.020	3 006	11 008	190 592	22.38	7	1.45	9	0.13	3	2 004	69.14
2500	2 502	7 503	4.33	0.057	7 506	27 508	490 592	144.44	7	3.61	9	0.33	3	5 004	461.50
5000	5 002	15 003	11.88	0.086	15 006	55 008	990 592	545.16	7	7.22	9	0.67	3	10 004	1 894.18
10000	10 002	30 003	42.21	0.170	30 006	110 008	-	-	-	14.46	9	1.37	3	-	-
15000	15 002	45 003	92.16	0.260	45 006	165 008	-	-	-	21.70	9	2.07	3	-	-

Table 5.2 illustrates the results with interval $[t, t']$. In this case, the CSL solution requires two concatenated transient analysis. Prism is still the fastest tool. While slower, the cost of the component method of MC4CSL$^{\text{TA}}$ shows a similar growth as Prism. This type of Until shows a very different behaviour between explicit and implicit methods in MC4CSL$^{\text{TA}}$, but also between MC4CSL$^{\text{TA}}$ and CoDeMoC.

Indeed for single clock DTAs, CoDeMoC implements the solution schema described in [Che+11a, sec. 4.2]. This schema is a zone-based analysis, where for each zone the embedded Markov chain is constructed explicitly. The solution is always a global, iterative solution of all the zone EMCs: no component-based analysis is used. Therefore, it is not surprising that the performance of CoDeMoC is comparable to that of MC4CSL$^{\mathrm{TA}}$ with the explicit method. In fact we can observe the quadratic behaviour typical of the explicit methods due to the increasing size of the embedded matrix $\mathbf{P}$.

In our tests, Prism always outperforms MC4CSL$^{\mathrm{TA}}$ by a constant factor. This difference requires additional investigations, but could be due to a less optimized implementation of MC4CSL$^{\mathrm{TA}}$, or could be due to the fact that the state space of the $\mathcal{M}\times\mathcal{A}$ process is larger than the CTMC state space, since the state space of $\mathcal{M}\times\mathcal{A}$ used by MC4CSL$^{\mathrm{TA}}$ is three times larger than that of $\mathcal{M}$ used by Prism, even if in the component method each component has a size similar to $\mathcal{M}$.

5.1.3 On-the-fly state space construction

The result of section 5.1.1 has the limitation that the state space of the cross product $\mathcal{M}\times\mathcal{A}$ still has to be constructed before doing any numerical computation. Therefore, the CSL$^{\mathrm{TA}}$ solution with the component method still consumes more space than the solution done by a CSL model checker.

In some cases, the $\mathcal{M}\times\mathcal{A}$ process contains more states than it is needed. For instance, the *Until*$[t, t']$ of Table 5.2 clearly shows that the number of states in $\mathcal{M}\times\mathcal{A}$ is roughly three times the state space S of the CTMC. This happens because the $S \times L \times C$ process has three $L \times C$ configuration:

$$\langle l_0,\, (0{,}t)\rangle, \quad \langle l_1,\, (t{,}t')\rangle, \quad \langle l_1,\, (t'{,}\infty)\rangle$$

The last configuration $\langle l_1, (t'{,}\infty)\rangle$ underlines a problem of the Until DTA. When the CTMC in location l_1 after t' does a Markovian move, it will be rejected and the process goes to $\bot$. Yet the state exists and is reachable from the $(t{,}t')$ zone, since there is no way for the CSL$^{\mathrm{TA}}$ model checker for understanding that this state will inexorably lead to a reject.

A possible solution consists in expanding the DTA automaton into its *zoned transition system*, where each state has form $\langle l, c\rangle$. In this way, the reachability analysis can be anticipated. For brevity, we call it the *zoned DTA* $\mathcal{Z}(\mathcal{A})$ of $\mathcal{A}$.

Zoned Transition system of a DTA $\mathcal{A}$

Let $\mathcal{A} = \langle \Sigma, Act, L, L_0, L_F, \Lambda, \rightarrow\rangle$ be a DTA, as defined in Def. 19, and let $K = \{c_0, c_1, \ldots, c_m\}$ be its set of clock constants, with $c_0 = 0$ and $c_{i+1} > c_i$ $\forall i \in [0, m)$ and $c_m \neq \infty$. Let:

$$\dot{C} \overset{\text{def}}{=} \Big\{ [c] \mid c \in K \Big\} \qquad \text{and} \qquad \overline{C} \overset{\text{def}}{=} \Big\{ \big(c, \mathsf{next}(c)\big) \mid c \in K \Big\}$$

be the set of *immediate zones* $\dot{C}$ and the set of *timed zones* $\overline{C}$.

We want to derive a *zoned DTA*, where the clock zone information is paired with the DTA location. Let $\mathcal{Z}(\mathcal{A})$ be the *zoned DTA* of $\mathcal{A}$. The state space Z of $\mathcal{Z}(\mathcal{A})$ is made of pairs $\langle l, c\rangle$, where every location $l \in L$ is paired with either an immediate or a timed clock zone c.

Definition 32 (Zoned DTA $\mathcal{Z}(\mathcal{A})$ of $\mathcal{A}$)**.** The zoned DTA $\mathcal{Z}(\mathcal{A})$ of a DTA $\mathcal{A}$ is a tuple $\mathcal{Z}(\mathcal{A}) = \langle \mathcal{A}, C, Z, \dot{Z}_0, Z_F, \hookrightarrow \rangle$ where:

- $\mathcal{A} = \langle \Sigma, Act, L, L_0, L_F, \Lambda, \rightarrow \rangle$ is the DTA;
- $C = \dot{C} \cup \overline{C}$ is the set of immediate and timed *clock zones* of $\mathcal{A}$;
- $Z \subseteq L \times C$ is the finite set of *locations*, partitioned into the set of immediate locations $\dot{Z} \subseteq L \times \dot{C}$ and the set of timed locations $\overline{Z} \subseteq L \times \overline{C}$;
- $\dot{Z}_0 = L_0 \times \{\dot{c}_0\}$ is the set of *initial locations*;
- $Z_F = Z \cap (L_F \times C)$ is the set of *final locations*;
- $\hookrightarrow \subseteq \Big(\overline{Z} \cup \Big(\big(2^{Act} \times \{x\}\big) \cup \{\delta\}\Big) \times L\Big) \cup \Big(\dot{Z} \cup \Big(\big(\sharp \times \{x\}\big) \cup \{\chi\}\Big) \times L\Big)$ is the finite *edge* relation.

The edge relation of Def. 32 has four kinds of arcs between Z locations:

1. $\bar{z} \xhookrightarrow{A,r} z'$: an *Inner* edge of the DTA that reads the CMTS actions in A with reset set r;
2. $\bar{z} \xhookrightarrow{\delta} \dot{z}'$: time elapse move from a timed zone into a boundary zone;
3. $\dot{z} \xhookrightarrow{\sharp,r} \dot{z}'$: a *Boundary* edge of the DTA that may trigger with urgency, with reset set r;
4. $\dot{z} \xhookrightarrow{\chi} \bar{z}'$: end of a boundary zone;

The edge relation does not contain the clock zone of the destination location, which is deduced by the edge itself and its *reset set.* An arc of type 1 remains in the same timed clock zone, unless the clock is reset, in which case it goes directly to the immediate zone $[0]$. An arc of type 2 goes from a clock zone $\bar{c} = (k, k')$ into the immediate zone $\dot{c} = [k']$. An arc of type 3 remains in the same immediate zone $\dot{c}$ of the source location $\dot{z}$, unless the reset set is $\{x\}$, in which case it goes to the $[0]$ zone. Finally, an arc of type 4 exits the immediate zone $\dot{c} = [k]$ going into the timed zone $\bar{c} = (k, \mathsf{next}(k))$. The two special symbols δ and χ label those edges that enter and exit a boundary zone.

The production rules for these four kind of $\mathcal{Z}(\mathcal{A})$ edges may be written in inductive form. Locations from timed zones follow the two rules:

$$\frac{\bar{c} \in \overline{C},\ \exists\, l \xrightarrow{\gamma, A, r} l',\ \text{s.t. } \bar{c} \models \gamma}{\langle l, \bar{c} \rangle \xhookrightarrow{A,r} \langle l', \bar{c}[r := 0] \rangle}\ \text{ZM–INNER}$$

$$\frac{\bar{c} \in \overline{C},\ \bar{c} \neq (c_m, \infty)}{\langle l, \bar{c} \rangle \xhookrightarrow{\delta} \langle l, \mathsf{upper}(\bar{c}) \rangle}\ \text{ZM–ENTER–BOUNDARY}$$

Locations from immediate zones follow the two rules:

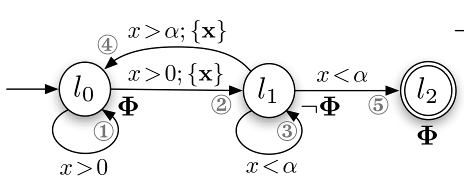
$$\frac{\dot{c} \in \dot{C},\ \exists\, l \xrightarrow{\gamma, \sharp, r} l',\ \text{s.t. } \dot{c} \models \gamma}{\langle l, \dot{c} \rangle \xhookrightarrow{\sharp,r} \langle l', \dot{c}[r := 0] \rangle}\ \text{ZM–BOUNDARY}$$

$$\frac{\dot{c} \in \dot{C},\ \nexists\, l \xrightarrow{\gamma, \sharp, r} l',\ \text{s.t. } \dot{c} \models \gamma}{\langle l, \dot{c} \rangle \xhookrightarrow{\chi} \langle l, (\dot{c}, \mathsf{next}(\dot{c})) \rangle}\ \text{ZM–EXIT–BOUNDARY}$$

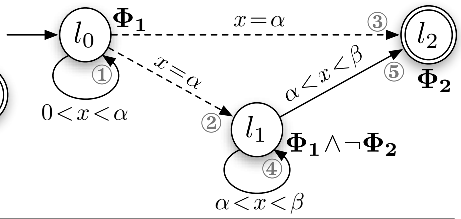
The $\mathsf{upper}(\bar{c})$ of a timed zone $\bar{c}$ and the $\mathsf{next}(\dot{c})$ of an immediate zone $\dot{c}$ are the upper time instant and the next clock constant in K, respectively. The next of the last clock constant c_m is ∞. The two rules ZM–INNER and ZM–BOUNDARY obey the *Inner* and *Boundary* edge semantics of the DTA $\mathcal{A}$.

Figure 5.5 Two sample DTAs with their associated zoned DTAs.

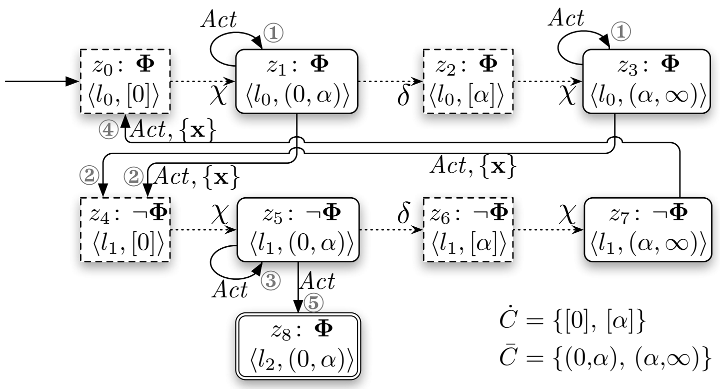
(a) Example DTA with loops of resets.



(b) The *Until*[α,β] DTA.

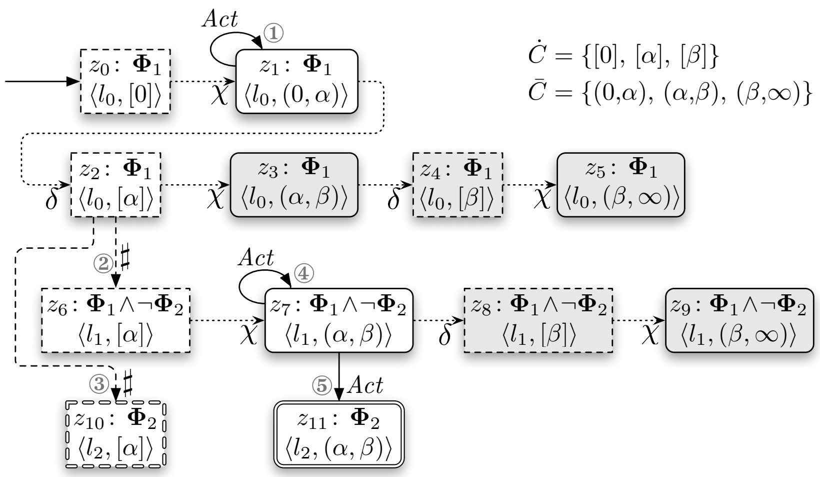


(c) Zoned DTA of the DTA (a).



$z_0: \mathbf{\Phi}$ $\langle l_0, [0]\rangle$
$z_1: \mathbf{\Phi}$ $\langle l_0, (0,\alpha)\rangle$
$z_2: \mathbf{\Phi}$ $\langle l_0, [\alpha]\rangle$
$z_3: \mathbf{\Phi}$ $\langle l_0, (\alpha,\infty)\rangle$
$z_4: \neg\mathbf{\Phi}$ $\langle l_1, [0]\rangle$
$z_5: \neg\mathbf{\Phi}$ $\langle l_1, (0,\alpha)\rangle$
$z_6: \neg\mathbf{\Phi}$ $\langle l_1, [\alpha]\rangle$
$z_7: \neg\mathbf{\Phi}$ $\langle l_1, (\alpha,\infty)\rangle$
$z_8: \mathbf{\Phi}$ $\langle l_2, (0,\alpha)\rangle$
Act
$Act, \{\mathbf{x}\}$
χ
δ
$\dot{C} = \{[0], [\alpha]\}$
$\bar{C} = \{(0,\alpha), (\alpha,\infty)\}$

(d) Zoned DTA of the DTA (b).



$z_0: \mathbf{\Phi}_1$ $\langle l_0, [0]\rangle$
$z_1: \mathbf{\Phi}_1$ $\langle l_0, (0,\alpha)\rangle$
$\dot{C} = \{[0], [\alpha], [\beta]\}$
$\bar{C} = \{(0,\alpha), (\alpha,\beta), (\beta,\infty)\}$
$z_2: \mathbf{\Phi}_1$ $\langle l_0, [\alpha]\rangle$
$z_3: \mathbf{\Phi}_1$ $\langle l_0, (\alpha,\beta)\rangle$
$z_4: \mathbf{\Phi}_1$ $\langle l_0, [\beta]\rangle$
$z_5: \mathbf{\Phi}_1$ $\langle l_0, (\beta,\infty)\rangle$
$z_6: \mathbf{\Phi}_1\wedge\neg\mathbf{\Phi}_2$ $\langle l_1, [\alpha]\rangle$
$z_7: \mathbf{\Phi}_1\wedge\neg\mathbf{\Phi}_2$ $\langle l_1, (\alpha,\beta)\rangle$
$z_8: \mathbf{\Phi}_1\wedge\neg\mathbf{\Phi}_2$ $\langle l_1, [\beta]\rangle$
$z_9: \mathbf{\Phi}_1\wedge\neg\mathbf{\Phi}_2$ $\langle l_1, (\beta,\infty)\rangle$
$z_{10}: \mathbf{\Phi}_2$ $\langle l_2, [\alpha]\rangle$
$z_{11}: \mathbf{\Phi}_2$ $\langle l_2, (\alpha,\beta)\rangle$
Act
$\sharp$
χ
δ

The other two rules realize the time sequence from timed to immediate zones, and back.

Figure 5.5 illustrates the zoned DTAs of two sample DTAs. Each location in (c) and (d) report the location $z \in Z$, the state proposition $\Lambda(l)$ that holds in $z = \langle l, c\rangle$, and on the second line the DTA location and the clock zone. The set of immediate and timed clock zones $\dot{C}$ and $\bar{C}$ are also drawn. Immediate and timed locations are drawn with a dotted and a solid border, respectively. Final locations have a double border. The set of locations that cannot reach a final location are grayed out.

The timed reachability of some locations (for instance z_8 and z_9 in (d)) represents an information that was not directly available in the DTA $\mathcal{A}$. These locations are irrelevant for the computation of the path probability, and can be discarded, since they will never reach a final location. Observe also that the correct construction of (d) does not build the edge $z_2 \xhookrightarrow{\delta} z_3$, since the *Boundary* edges ② and ③ in the DTA have priority and the process will take for sure one of the two edges, since the logic condition of remaining in l_0 in $[\alpha]$ is:

$$\begin{aligned}
\Phi_1 \wedge \neg\big(\Phi_2 \vee (\Phi_1 \wedge \neg\Phi_2)\big) &= \Phi_1 \wedge \Phi_2 \wedge \neg(\Phi_1 \wedge \neg\Phi_2) = \\
&= \Phi_1 \wedge \Phi_2 \wedge (\neg\Phi_1 \vee \Phi_2) = \\
&= (\Phi_1 \wedge \Phi_2 \wedge \neg\Phi_1) \vee (\Phi_1 \wedge \Phi_2 \wedge \Phi_2) = \\
&= \mathit{false} \vee \mathit{false}
\end{aligned}$$

which is never satisfied, for any given CTMC. Therefore, z_3 is unreachable, and also z_4 and z_5, which can be removed. This condition can be evaluated for any χ edge that is the result of a *Boundary* edge, which removes those locations that are logically unreachable.

Each location $\langle l, c\rangle$ of Fig. 5.5(c,d) is labeled with the state proposition expressions $\Lambda(l)$. Since these expressions are taken directly from the DTA labeling function Λ, they are are not part of the $\mathcal{Z}(\mathcal{A})$ definition.

Since *Boundary* edges may fire multiple times in boundary zones, it is more easy to define the concept of *tangible zoned DTA*, where only timed locations are kept, and boundary locations are collapsed with a transitive closure. The firing of a sequence of DTA *Boundary* edges $l_0 \xrightarrow{\gamma_1,\,\sharp,\,r_1} l_1 \xrightarrow{\gamma_2,\,\sharp,\,r_2} \ldots \xrightarrow{\gamma_n,\,\sharp,\,r_n} l_n$ may happen only if all the state proposition expressions $\Lambda(l_0), \Lambda(l_1), \ldots, \Lambda(l_n)$ are satisfied by the destination CTMC state s'. A transitive closure of *Boundary* firings is more easily expressed by moving the state proposition onto the edge.

Definition 33 (Tangible Zoned DTA $\mathcal{T}(\mathcal{A})$ of $\mathcal{A}$)**.** The tangible zoned DTA $\mathcal{T}(\mathcal{A})$ of a DTA $\mathcal{A}$ is a tuple $\mathcal{T}(\mathcal{A}) = \langle \mathcal{A}, \overline{C}, \mathcal{Z}, \mathcal{Z}_0, \mathcal{Z}_F, \mathit{init}, \nrightarrow\!\!\!\rightarrow \rangle$ where:

- $\mathcal{A} = \langle \Sigma, \mathit{Act}, L, L_0, L_F, \Lambda, \rightarrow\rangle$ is the DTA;
- $\overline{C}$ is the set of timed *clock zones* of $\mathcal{A}$ with form (α, β), $\alpha < \beta$;
- $\mathcal{Z} \subseteq L \times \overline{C}$ is the finite set of timed *locations*;
- $\mathcal{Z}_0 = L_0 \times \{\bar{c}_0\}$ is the set of *initial locations*;
- $\mathcal{Z}_F = \mathcal{Z} \cap (L_F \times \overline{C})$ is the set of *final locations*;
- $\mathit{init} : \mathcal{Z}_0 \rightarrow \mathcal{B}(\Sigma)$ is the *initial location expression*;
- $\nrightarrow\!\!\!\rightarrow \;\subseteq (\mathcal{Z} \setminus \mathcal{Z}_F) \times \big(2^{\mathit{Act}} \cup \sharp\big) \times \mathcal{B}(\Sigma) \times \{x\} \times \mathcal{Z}$ is the finite *edge* relation, where $z \xrightarrow{A,\,\lambda,\,r} z'$ means that $\langle z, A, \lambda, r, z'\rangle \in \nrightarrow\!\!\!\rightarrow$.

Let $\sharp(\dot{z})$ be the set of locations $\dot{z}'$ that can be reached from the immediate location $\dot{z}$ of the ZDTA $\mathcal{Z}(\mathcal{A})$ with a boundary edge $\dot{z} \xhookrightarrow{\sharp,r} \dot{z}'$. The tangible closure of the boundary edge relation from the tangible location $\bar{z} \in Z$ is then given with the rule:

$$\frac{\bar{z} \in Z,\ \exists\, \bar{z} \xhookrightarrow{\delta,\,r_0} \dot{z}_1 \xhookrightarrow{\sharp,r_1} \ldots \xhookrightarrow{\sharp,r_n} \dot{z}_n \xhookrightarrow{\chi} \bar{z}',\ n \geq 0}{\lambda = \Lambda(\bar{z}') \wedge \big(\bigwedge_{i=1}^{n} \Lambda(\dot{z}_i)\big) \wedge \big(\bigwedge_{\dot{z}'' \in \sharp(\dot{z}_n)} \neg\Lambda(\dot{z}'')\big),\ \bar{z} \xrightarrow{\sharp,\,\lambda,\,\cup_{i=0}^{n} r_i} \bar{z}'} \ \text{TZ–CLOSURE-IMM}$$

Similarly, a Markovian action of the CTMC triggers an *Inner* DTA edge that may be followed by zero or more *Boundary* edges (when the clock x is reset to zero), leading to:

$$\frac{\bar z \in Z,\ \exists\, \bar z \xrightarrow{A,\, r_0} \dot z_1 \xrightarrow{\sharp, r_1} \dots \xrightarrow{\sharp, r_n} \dot z_n \xrightarrow{\chi} \bar z',\ n \geq 0}{\lambda = \Lambda(\bar z') \wedge \left(\bigwedge_{i=1}^{n} \Lambda(\dot z_i)\right) \wedge \left(\bigwedge_{\dot z'' \in \sharp(\dot z_n)} \neg\Lambda(\dot z'')\right),\ \bar z \xrightarrow{A,\, \lambda,\, \cup_{i=0}^{n} r_i} \bar z'} \ \text{TZ–CLOSURE-TM}$$

A TZDTA edge $z \xrightarrow{A, \lambda, r} z'$ has a logical condition λ which is the logical *and* of satisfying the destination location condition $\Lambda(z')$, all the intermediate location conditions $\Lambda(\dot z_i), 1 \leq i \leq n$, and in the last immediate location every other *Boundary* edge must not be satisfied. Given $\mathcal{Z}(\mathcal{A})$, the corresponding $\mathcal{T}(\mathcal{A})$ is constructed by taking all the timed locations and *Inner* edges, and by applying the closure rule on all *Boundary* edges. Also, there are neither δ nor χ transitions in $\mathcal{T}(\mathcal{A})$, which has only moves from tangible to tangible locations.

Figure 5.6 Tangible zoned DTA of the two DTAs of Fig. 5.5.

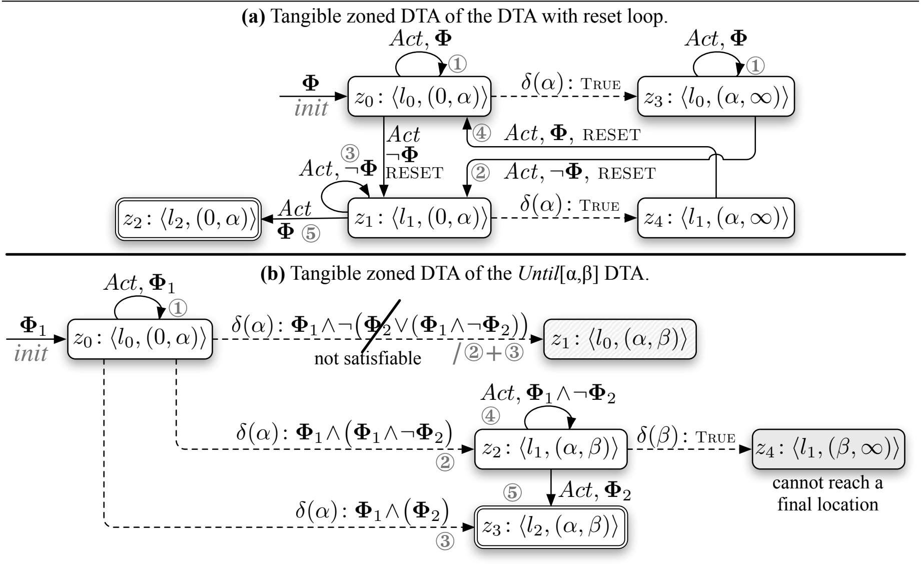
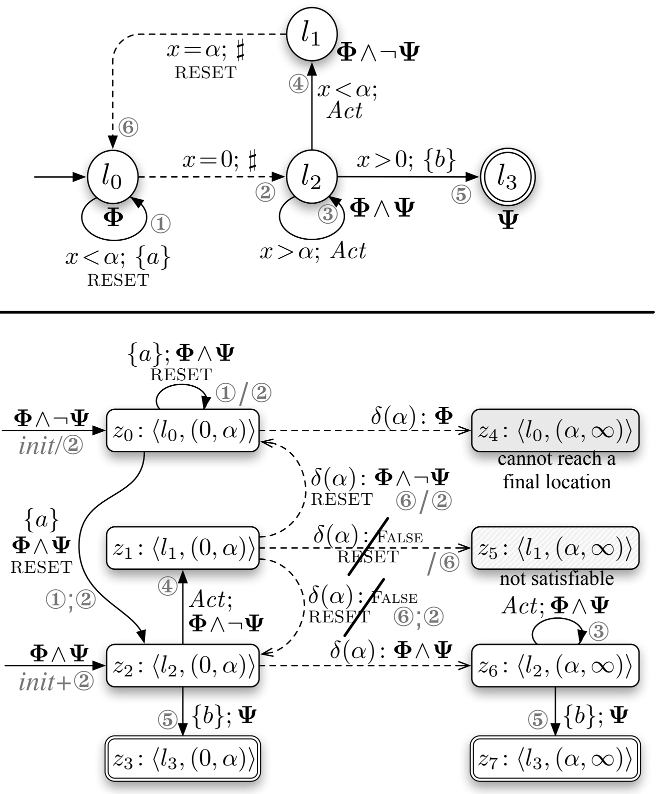


Figure 5.6 shows the *tangible* closure of the two $\mathcal{Z}(\mathcal{A})$ of Fig. 5.5. Boundary edges are all collapsed into $\boldsymbol{\delta}$ edges, which are labeled with a state proposition expression that is the transitive closure of all the s.p.e. that must be satisfied to follow that edge. The tangible ZDTA of the *Until*$[t, t']$ DTA is the most interesting. Location z_1 is unreachable because the $\boldsymbol{\delta}$ edge has an empty condition. Location z_4 is irrelevant, since any path that reaches this location is always rejected. The advantage of collapsing the state proposition expression of the closure of *Boundary* edges is that it becomes clear wether an edge has an unsatisfiable condition. Each edge is also labelled with the sequence of DTA edges that represents (with circled numbers), and which DTA edges are not satified by the transitive closure (written after a '/'). The structure of Fig. 5.6(b) shows that there are at most three tangible zones for an *Until*$[\alpha, \beta]$, while the other two zones can be discarded. This allows to optimize the $\mathcal{M}\times\mathcal{A}$ cross product, by removing irrelevant states in advance.

Figure 5.7 shows another DTA $\mathcal{A}$ with its tangible zoned DTA $\mathcal{T}(\mathcal{A})$. This DTA shows many characteristics of the tangible closure. Initial edges may be duplicated in $\mathcal{T}(\mathcal{A})$ with a complex conditions, due to *Boundary* edges enabled

Figure 5.7 Another sample DTA with its tangible zoned DTA.



at $x = 0$. Location z_4 is discarded since it cannot reach a final location. Location z_5 is unreachable, since the closure condition on $z_1 \xhookrightarrow{\sharp,\lambda,r} z_5$ is always false.

Component construction of the tangible state space

The structure of these tangible zoned DTAs shows how the path probability flows from the initial locations to the final locations. This structure can be used to improve the CSL$^{\text{TA}}$ model checking:

- The component structure can be identified directly on the $L \times C$ automaton, instead of using the $\mathcal{M} \times \mathcal{A}$ process.
- The construction of the $\mathcal{M} \times \mathcal{A}$ process and its numerical solution (the $\boldsymbol{\mu}^{\mathcal{D}}(w_i)$ function of (4.44)) can be done one component at a time, instead of keeping the entire $\mathcal{M} \times \mathcal{A}$ process in memory.

In fact, a DTA-based component method can be defined, where components are sets of tangible locations in $\mathbb{Z}$. Implicitly, each component contains all the CTMC states that may reach that component in the cross product $\mathcal{S} \times L \times C$. How to define these set of states will be described afterwards. For now, we assume that each location $z \in \mathbb{Z}$ has its entire CTMC state space $\mathcal{S}$.

Let $w \subseteq \mathcal{Z}$ be a *component*. The minimal unit of computation is the *augmented set* $\widehat{w}$ of w, which is:

$$\widehat{w} \overset{\text{def}}{=} w \cup \big\{ z \in \mathcal{Z} \mid z \text{ reachable from } w \text{ with one or more } z' \xrightarrow{A,\varnothing} z \text{ edges} \big\} \tag{5.1}$$

Similarly to the component method for MRPs, a sequence W of components can be derived for $\mathcal{T}(\mathcal{A})$. Since each location z is tangible, it is possible to derive:

- $\mathbf{Q}_z$ the matrix of Markovian actions without resets that go from $z = \langle l, c \rangle$ into all the other locations $z' = \langle l', c' \rangle$ with $c = c'$.
- $\bar{\mathbf{Q}}_z$ the matrix of Markovian actions with the reset of the clock that go from $z = \langle l, c \rangle$ into all the other locations $z' = \langle l', c_0 \rangle$ in the first clock zone.
- $\mathbf{\Delta}_z$ the matrix of branching probabilities at the clock boundary.

Future works

A complete analysis and implementation of an on-the-fly CSL$^{\mathrm{TA}}$ model checker guided by a structural analysis of the DTA is ongoing, at the time of writing. This methodology allows for a set of optimizations that are not possible if the model checker builds the $\mathcal{M} \times \mathcal{A}$ process directly. It becomes possible to construct only portions of the state space, since some zone are unneeded, for instance because the process will always be rejected once it enters a certain zone. Also, an optimized allocation of the state vectors for each zone allows to reduce the memory footprint of the model checker.

5.2 Tool support

The various experiments shown in this thesis has been computed with various existing tools, like Prism [KNP09] or MRMC [Kat+11]. However, some limitations of the implementations in these tools have created the necessity of building two additional tools, that will be presented briefly in this section.

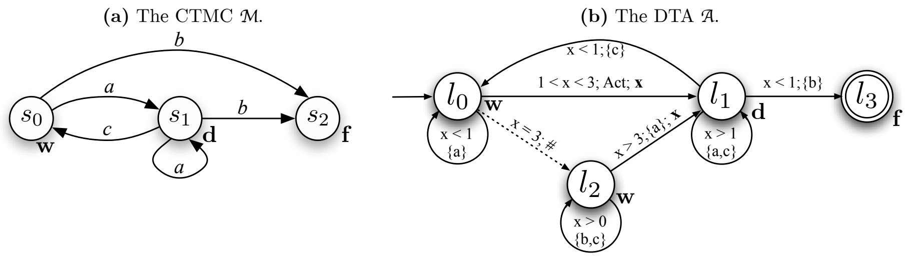
5.2.1 The MC4CSL$^{\mathrm{TA}}$ tool

The work in [AD10d] shows that the forward model checking of CSL$^{\mathrm{TA}}$ can also be reduced to the solution of a DSPN in steady-state. Such DSPN is generated as the parallel combination of the Petri net of the model, a Petri net for the DTA and a DSPN of the clock. The resulting system is a DSPN with at most one deterministic enabled at any time.

The tool MC4CSL$^{\mathrm{TA}}$ is a CSL$^{\mathrm{TA}}$ model checker. CTMCs and DTAs are read as files, and are evaluated in memory. The $\mathcal{P}_{\bowtie\lambda}(\mathcal{A})$ is implemented by translating the model $\mathcal{M}$ and the DTA $\mathcal{A}$ into a single DSPN. The DSPN is then evaluated numerically with an external DSPN tool. The tools *DSPNexpressNG* [Lin98] [CL93], *TimeNet* [Zim+00], *SPNica* [Ger00] and *DSPN-Tool* [AD10b] are used as backends, to do the actual numerical computation.

Figure 5.8 shows a sample CTMC and DTA. The CTMC represents a classical *working-degraded-failure* system with three states. The system may go back

Figure 5.8 A sample CTMC and DTA as input for the MC4CSL$^{\mathrm{TA}}$ tool.



and forth from the working state to the degraded state. The system may also break accidentally, going into a state of failure where it cannot recover. The DTA reads a complex path of the modeled system.

When given as input at the MC4CSL$^{\mathrm{TA}}$ tool, it generates the DSPN shown in Figure 5.9 and computes the steady-state solution of reaching the $\perp$ place. The DSPN is made by different parts that are synchronized together. synchronization arcs are drawn in grey. State proposition expressions and action sets are also represented as subnets, to simplify the translation into a Petri net model. The detailed description of the working structure of the tool is given in [AD10d].

5.2.2 The DSPN-Tool tool

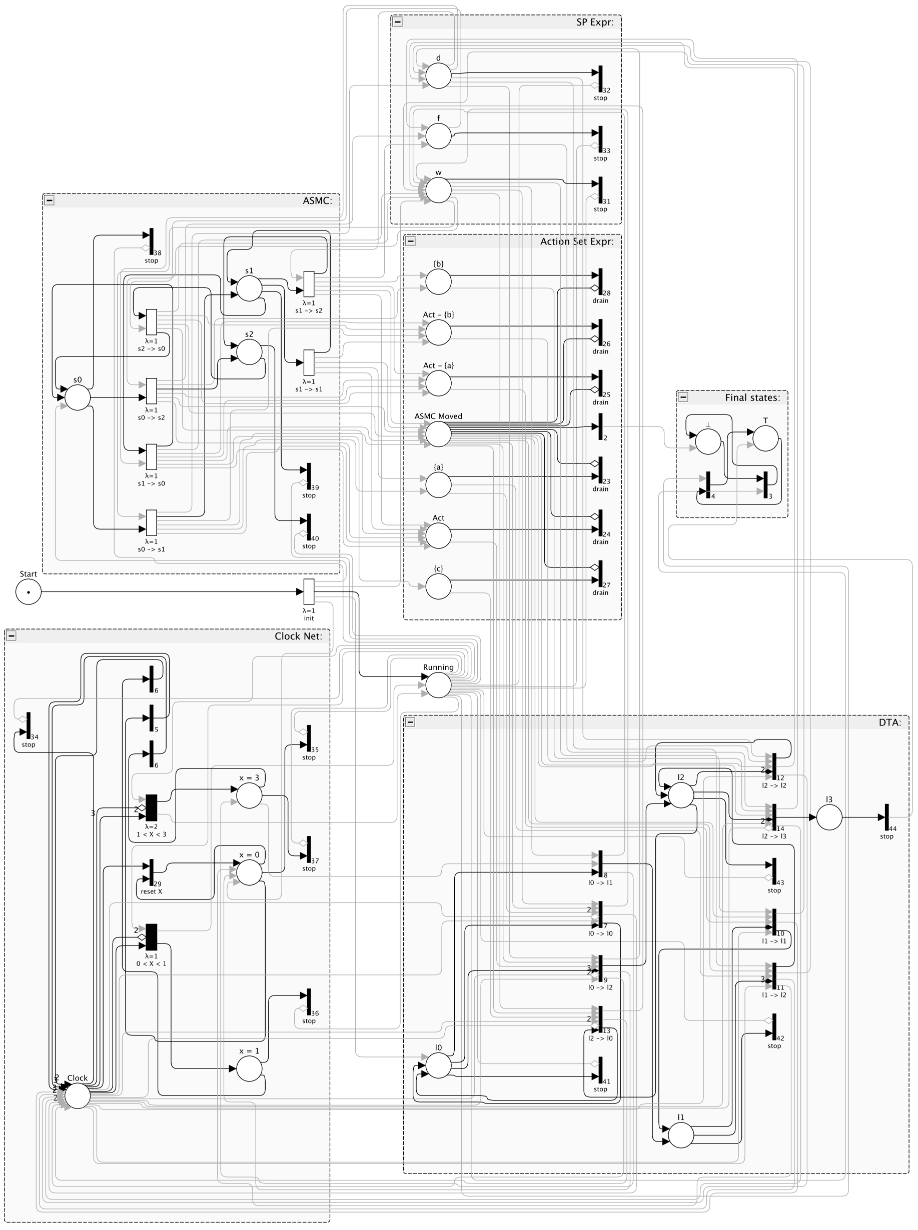
The DSPN-Tool is a Petri net solver for DSPN (*Deterministic Stochastic Petri Nets*). In a search for the best DSPN solver for solving the class of DSPN generated by MC4CSL$^{\mathrm{TA}}$ we tested many solvers and we found some complex limitations which were difficult to resolve. The particular DSPN produced require certain characteristics of the solvers, that for certain tools lead to different, and sometimes inappropriate, behaviors (described in [AD10d]). These characteristics include: (1) Large number of immediate transitions; (2) Large number of places and transitions (hundreds); (3) Multiple arcs: the Tangible Reachability Graph (TRG) can have more than one arc between any two states (4) Self-loop preemption: firing of an exponential transition that disables and re-enables a deterministic, coming back to the same state; (5) steady-state solver for non-ergodic MRPs.

Table 5.3 DSPN tools features overview

	SPNica	TimeNET	DSPN ExpressNG	DSPN Tool
1) Hundreds of immediate transitions	very slow	-	-	$\surd$
2) Hundreds of places	very slow	-	-	$\surd$
3) TRG with multiple arcs	$\surd$	-	-	$\surd$
4) TRG with preemptive self-loops	$\surd$	-	-	$\surd$
5) Steady-state for non ergodic	-	-	-	$\surd$
6) Fill-in avoidance	$\surd$	$\surd$	-	$\surd$
7) Exploitation of isomorphism	-	-	$\surd$	$\surd$
8) Data structures	full	sparse	sparse	sparse
9) Transient solution	$\surd$	$\surd$	$\surd$	-

Table 5.3 summarizes the results of the suitability analysis of SPNica, DSPNExpress and TimeNET for the problem at hand. The first 5 rows in the table

Figure 5.9 The DSPN constructed by the MC4CSL$^{\text{TA}}$ tool.



correspond to the characteristics listed above, row 6, 7, and 8 discriminate efficiency of the tools: “fill-in avoidance” refers to the use of the so-called “iterative solution” by German [Ger01], while “isomorphism” refers to the exploitation of subordinated Markov chains which are isomorphic [Lin98]. The last row accounts for the presence of a transient analysis solver: this is not required for our model-checking purposes, but we have added it for completeness.

The main features implemented in DSPN-Tool are:

- Supports exponential, immediate and deterministic transitions (but not general transitions), with marking-dependent delays and guards.

- Supports an almost unlimited number of places and transitions, and is explicitly optimized for huge nets (with sparse markings and various optimizations on the transition enabling detection).
- Implements steady-state and transient solution of CTMC, and steady state solution of Markov Regenerative Processes (MRP).
- It supports non-ergodic MRPs, with preemption arcs, preemption loops and branching probabilities.
- May write the (T)RG in Dot/Prism/Asmc formats.
- It may be used to convert GreatSPN net files into other Petri net file formats (SPNica, Cosmos).

The numerical analysis has the implementation of the matrix-free steady-state solution of MRP, the component method and the MRP preconditioner, described in chapter 4. The tool also supports transient and steady-state solutions of DTMCs and CTMCs. Steady state solutions are computed with a variety of methods: Jacobi, Gauss-Seidei, GMRES, BiCG-Stab, CGS. Preconditioning can be done with ILUTK, or with the input/output preconditioner,

Acknowledgements

DSPN-Tool and MC4CSL^{TA} are the result of a work done under the National Inter-University Consortium for Telecommunications (CNIT) research grant #022825/2009, as a part of a CSL^{TA} Stochastic Model Checker implementation under the CRUTIAL Research Project.

List of Publications

[AD10a] Elvio Gilberto Amparore and Susanna Donatelli. "A Component-based Solution Method for Non-Ergodic Markov Regenerative Processes". In: *EPEW*. Vol. 6342. Lecture Notes in Computer Science. Bertinoro, Italy: Springer, 2010, pp. 236–251.

[AD10b] Elvio Gilberto Amparore and Susanna Donatelli. "DSPN-Tool: a new DSPN and GSPN solver for GreatSPN". In: *International Conference on Quantitative Evaluation of Systems*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 79–80.

[AD10c] Elvio Gilberto Amparore and Susanna Donatelli. "MC4CSL$^{\mathrm{TA}}$: an efficient model checking tool for CSL$^{\mathrm{TA}}$". In: *International Conference on Quantitative Evaluation of Systems*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 153–154.

[AD10d] Elvio Gilberto Amparore and Susanna Donatelli. "Model Checking CSL$^{\mathrm{TA}}$ with Deterministic and Stochastic Petri Nets". In: *Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. DSN-PDS 2010. Chicago, USA: IEEE Computer Society Press, 2010.

[AD11] Elvio Gilberto Amparore and Susanna Donatelli. "Revisiting the Iterative Solution of Markov Regenerative Processes". In: *Numerical Linear Algebra with Applications, Special Issue on Numerical Solutions of Markov Chains* 18 (2011), pp. 1067–1083.

[AD12a] Elvio Gilberto Amparore and Susanna Donatelli. "A component-based solution for reducible Markov regenerative processes". Paper submitted at Performance Evaluation, Elsevier. 2012.

[AD12b] Elvio Gilberto Amparore and Susanna Donatelli. "Backward solution of Markov chains and Markov Renewal Processes: formalization and applications". In: *Sixth International Workshop on Practical Applications of Stochastic Modelling (PASM12)*. 2012.

[Amp+11] Elvio Gilberto Amparore, Marco Beccuti, Susanna Donatelli, and Giuliana Franceschinis. "Probe Automata for Passage Time Specification". In: *Proceedings of the 2011 Eighth International Conference on Quantitative Evaluation of SysTems*. QEST 2011. Washington, DC, USA: IEEE Computer Society, 2011, pp. 101–110.

Bibliography

[AC87] M. Ajmone Marsan and G. Chiola. “On Petri nets with deterministic and exponentially distributed firing times”. In: *Advances in Petri Nets*. Vol. 266/1987. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1987, pp. 132–145.

[Auz11] Dr. Winfried Auzinger. *General remarks on Preconditioning*. http://http://www.asc.tuwien.ac.at/~winfried/teaching/106.079/SS2011/downloads/script-p-106-122.pdf. [Online; accessed 28-Oct-2012]. 2011.

[Azi+00] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. “Model-checking continuous-time Markov chains”. In: *ACM Transactions on Computational Logic* 1.1 (2000), pp. 162–170.

[Baa+09] Souheib Baarir, Marco Beccuti, Davide Cerotti, Massimiliano De Pierro, Susanna Donatelli, and Giuliana Franceschinis. “The GreatSPN tool: recent enhancements”. In: *SIGMETRICS Performance Evaluation Review* 36.4 (2009), pp. 4–9.

[Bai+03] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. “Model-Checking Algorithms for Continuous-Time Markov Chains”. In: *IEEE Transactions on Software Engineering* 29.6 (2003), pp. 524–541.

[Bal+11] Paolo Ballarini, Hilal Djafri, Marie Duflot, Serge Haddad, and Nihal Pekergin. “HASL: An Expressive Language for Statistical Verification of Stochastic Models”. In: *Proceedings of the 5th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS’11)*. Cachan, France, May 2011, pp. 306–315.

[Bar+10] Benoit Barbot, Taolue Chen, Tingting Han, Joost-Pieter Katoen, and Alexandru Mereacre. *Efficient CTMC Model Checking of Linear Real-Time Objectives*. Tech. rep. RWTH Aachen University, 2010.

[Ben02] Michele Benzi. “Preconditioning techniques for large linear systems: a survey”. In: *Journal of Computational Physics* 182.2 (2002), pp. 418–477.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008, pp. I–XVII, 1–975.

[BM10] Junaid Babar and Andrew Miner. “Meddly: Multi-terminal and Edge-Valued Decision Diagram LibrarY”. In: *Quantitative Evaluation of Systems, International Conference on.* Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 195–196.

[Brá+11] Tomás Brázdil, Jan Krcál, Jan Kretínský, Antonín Kucera, and Vojtech Rehák. “Measuring Performance of Continuous-Time Stochastic Processes using Timed Automata”. In: *CoRR* abs/1101.4204 (2011).

[CD97] C. Chaouiya and Y. Dallery. “Petri Net Models of Pull Control Systems for Assembly Manufacturing Systems”. In: *the XVIII International Conference on Applications and Theory of Petri Nets.* 1997, p. 23.

[CG08] Allan Clark and Stephen Gilmore. “State-Aware Performance Analysis with eXtended Stochastic Probes”. In: *Proceedings of the 5th European Performance Engineering Workshop.* EPEW ’08. Palma de Mallorca, Spain: Springer-Verlag, 2008, pp. 125–140.

[Che+11a] Taolue Chen, Tingting Han, Joost-Pieter Katoen, and Alexandru Mereacre. “Model Checking of Continuous-Time Markov Chains Against Timed Automata Specifications”. In: *Logical Methods in Computer Science (2011)* 7.1 (2011).

[Che+11b] Taolue Chen, Marco Diciolla, Marta Kwiatkowska, and Alexandru Mereacre. “Time-bounded verification of CTMCs against real-time specifications”. In: *Proceedings of the 9th international conference on Formal modeling and analysis of timed systems.* FORMATS’11. Aalborg, Denmark: Springer-Verlag, 2011, pp. 26–42.

[CL06] Christos G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems.* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[CL93] Gianfranco Ciardo and Christoph Lindemann. “Analysis of Deterministic and Stochastic Petri Nets”. In: *Performance Evaluation.* IEEE Computer Society, 1993, pp. 160–169.

[CLS01] Gianfranco Ciardo, Gerald Lüttgen, and Radu Siminiceanu. “Saturation: An Efficient Iteration Strategy for Symbolic State-Space Generation.” In: *TACAS’01.* 2001, pp. 328–342.

[Cox55] D. R. Cox. “The analysis of non-Markovian stochastic processes by the inclusion of supplementary variables”. In: *Proceedings of the Cambridge Philosophical Society.* Vol. 51. Proceedings of the Cambridge Philosophical Society. July 1955, pp. 433–441.

[CT04] J. K. Cullum and M. Tuma. *Matrix-free preconditioning using partial matrix estimation.* Tech. rep. Los Alamos National Laboratory, 2004.

[CV94] Tony F. Chan and Henk A. van der Vorst. “Approximate And Incomplete Factorizations”. In: *ICASE/LARC interdisciplinary series in science and engineering.* 1994, pp. 167–202.

[DHK03] Nicholas J. Dingle, Peter G. Harrison, and William J. Knottenbelt. "HYDRA: HYpergraph-based Distributed Response-time Analyser". In: *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2003).* 2003, pp. 215–219.

[DHS09] Susanna Donatelli, Serge Haddad, and Jeremy Sproston. "Model Checking Timed and Stochastic Properties with CSL^{TA}". In: *IEEE Transactions on Software Engineering* 35.2 (2009), pp. 224–240.

[EC82] E.Allen Emerson and Edmund M. Clarke. "Using branching time temporal logic to synthesize synchronization skeletons". In: *Science of Computer Programming* 2.3 (1982), pp. 241 –266.

[EH86] E. Allen Emerson and Joseph Y. Halpern. ""Sometimes" and "Not Never" revisited: on branching versus linear time temporal logic". In: *J. ACM* 33.1 (1986), pp. 151–178.

[ESG05] Jasper van den Eshof, Gerard L. G. Sleijpen, and Martin B. van Gijzen. "Relaxation strategies for nested Krylov methods". In: *Journal of Computational and Applied Mathematics* 177 (2 2005), pp. 347–365.

[FG88] Bennett L. Fox and Peter W. Glynn. "Computing Poisson probabilities". In: *Communications of the ACM* 31.4 (1988), pp. 440–445.

[Ger00] Reinhard German. *Performance Analysis of Communication Systems with Non-Markovian Stochastic Petri Nets.* New York, NY, USA: John Wiley & Sons, Inc., 2000.

[Ger01] Reinhard German. "Iterative analysis of Markov regenerative models". In: *Performance Evaluation* 44 (1-4 2001), pp. 51–72.

[Han91] Hans A. Hansson. "Time and probability in formal design of distributed systems". Available as report DoCS 91/27. PhD thesis. docs, 1991.

[HBA11] Richard A. Hayden, Jeremy T. Bradley, and Clark Allan. "Performance specification and evaluation with Unified Stochastic Probes and fluid analysis". In: *IEEE Transactions on Software Engineering* (2011).

[Hor+11] Andras Horvath, Marco Paolieri, Lorenzo Ridi, and Enrico Vicario. "Bounded model checking of generalized semi-Markov processes using stochastic state classes". In: *International Conference on Quantitative Evaluation of Systems.* Los Alamitos, CA, USA: IEEE Computer Society, 2011.

[Kat+01] Joost-Pieter Katoen, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. "Faster and Symbolic CTMC Model Checking". In: *Proceedings of the Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification.* PAPM-PROBMIV '01. London, UK: Springer-Verlag, 2001, pp. 23–38.

[Kat+11] Joost-Pieter Katoen, Ivan S. Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David N. Jansen. "The ins and outs of the probabilistic model checker MRMC". In: *Performance Evaluation* 68 (2 2011), pp. 90–104.

[KNP09] M. Kwiatkowska, G. Norman, and D. Parker. “PRISM: Probabilistic Model Checking for Performance and Reliability Analysis”. In: *ACM SIGMETRICS Performance Evaluation Review* 36.4 (2009), pp. 40–45.

[Kul95] Vidyadhar G. Kulkarni. *Modeling and analysis of stochastic systems*. London, UK: Chapman & Hall Ltd., 1995.

[Kun06] Matthias Kuntz. “Symbolic semantics and verification of stochastic process algebras”. PhD thesis. 2006, pp. 1–241.

[Lin98] Christoph Lindemann. *Performance Modelling with Deterministic and Stochastic Petri Nets*. New York, NY, USA: John Wiley & Sons, Inc., 1998.

[MH06] Jose M. Martinez and Boudewijn R. Haverkort. “CSL Model Checking of Deterministic and Stochastic Petri Nets”. In: *Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB), 2006 13th GI/ITG Conference* (2006), pp. 1 –18.

[ML78] Cleve Moler and Charles Van Loan. “Nineteen Dubious Ways to Compute the Exponential of a Matrix”. In: *SIAM Review* 20 (1978), pp. 801–836.

[OIS98] W. Douglas Obal, II, and William H. Sanders. *State-Space Support for Path-Based Reward Variables*. 1998.

[Pnu77] Amir Pnueli. “The Temporal Logic of Programs”. In: *FOCS*. IEEE Computer Society, 1977, pp. 46–57.

[Pyk59] Ronald Pyke. *Markov Renewal Processes with Finitely Many States*. New York: Columbia University, 1959.

[Rei66] J. K. Reid. “A Method for Finding the Optimum Successive Over-Relaxation Parameter”. In: (1966).

[Saa94] Yousef Saad. “ILUT: A dual threshold incomplete LU factorization”. In: *Numerical Linear Algebra with Applications* 1.4 (1994), pp. 387–402.

[Saa95] Yousef Saad. “Preconditioned Krylov subspace methods for the numerical solution of Markov chains”. In: *Computations with Markov chains*. Kluwer Academic Publishers, 1995, pp. 49–64.

[Son89] Peter Sonneveld. “CGS, a fast Lanczos-type solver for nonsymmetric linear systems”. In: *SIAM Journal on Scientific and Statistical Computing* 10.1 (1989), pp. 36–52.

[SS86] Youcef Saad and Martin H Schultz. “GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems”. In: *SIAM Journal on Scientific and Statistical Computing* 7.3 (1986), pp. 856–869.

[Ste94] William J. Stewart. *Introduction to the numerical solution of Markov chains*. Princeton University Press, 1994.

[Tar71] Robert Tarjan. “Depth-first search and linear graph algorithms”. In: *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (SWAT 1971)*. Washington, DC, USA: IEEE Computer Society, 1971, pp. 114–121.

[Tri02] Kishor S. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. 605 Third Avenue, New York, USA: John Wiley and Sons, Ltd., 2002.

[TRS87] Kishor S. Trivedi, Andrew L. Reibman, and Roger Smith. “Transient Analysis of Markov and Markov Reward Models”. In: *Computer Performance and Reliability ’87*. 1987, pp. 535–545.

[Var85] Moshe Y. Vardi. “Automatic verification of probabilistic concurrent finite state programs”. In: *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*. SFCS ’85. Washington, DC, USA: IEEE Computer Society, 1985, pp. 327–338.

[Vor92] H. A. van der Vorst. “BI-CGSTAB: a fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems”. In: *SIAM Journal on Scientific and Statistical Computing* 13.2 (1992), pp. 631–644.

[VV93] H. A. Van der Vorst and C. Vuik. “The superlinear convergence behaviour of GMRES”. In: *J. Comput. Appl. Math.* 48.3 (1993), pp. 327–341.

[Zha+12] Lijun Zhang, David N. Jansen, Flemming Nielson, and Holger Hermanns. “Efficient CSL Model Checking Using Stratification”. In: *Logical Methods in Computer Science (2012)* 8 (2 2012). eprint: arXiv/1104.4983.

[Zim+00] Armin Zimmermann, Jörn Freiheit, Reinhard German, and Günter Hommel. “Petri Net Modelling and Performability Evaluation with TimeNET 3.0”. In: *TOOLS ’00: Proceedings of the 11th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools*. London, UK: Springer-Verlag, 2000, pp. 188–202.