Computing Behavioral Relations for Probabilistic Concurrent Systems

Daniel Gebler[1], Vahid Hashemi[2,3], and Andrea Turrini[4]

[1] Department of Computer Science, VU University Amsterdam,
De Boelelaan 1081a, NL-1081 HV Amsterdam, The Netherlands

[2] Max Planck Institute for Informatics, 66123 Saarbrücken, Germany

[3] Department of Computer Science
Saarland University, 66123 Saarbrücken, Germany

[4] State Key Laboratory of Computer Science, Institute of Software,
Chinese Academy of Sciences, 100190 Beijing, China

Abstract. Behavioral equivalences and preorders are fundamental notions to formalize indistinguishability of transition systems and provide means to abstraction and refinement. We survey a collection of models used to represent concurrent probabilistic real systems, the behavioral equivalences and preorders they are equipped with and the corresponding decision algorithms. These algorithms follow the standard refinement approach and they improve their complexity by taking advantage of the efficient algorithms developed in the optimization community to solve optimization and flow problems.

1 Introduction

1.1 Probabilistic Systems

Probability, *time*, and *nondeterminism*. These are three main characteristics of several real-world applications. *Probability* occurs every time the behavior of the applications is not unique, either by construction or by physical properties. For example, distributed algorithms like the Zeroconf protocol or cryptographic protocols like SSL are based on random choices to break symmetry or to insert uncertainty in order to achieve their goals. Each time a message is transmitted on the network, in fact, transmission protocols have to manage the corruption of the messages, as well as their loss, as the effect of the interference with other concurrent transmissions or physical properties of the transmission medium. For instance, simultaneous transmissions on the same channel of a wireless network lead to the collision of the sent messages and their corruption.

Beside probabilities, these systems often have another source of uncertainty, namely *nondeterminism*, that appears whenever an event may occur with unpredictable behavior; for instance, the event of a host starting the transmission in a wireless network.

Time governs the evolution of the system: with the time passing, the system performs and reacts to actions and correspondingly changes its state, according to its goals. Time can be considered as a discrete component (e.g., a program running on a computer performs one operation at each tick of the digital clock) or as a continuum behavior (e.g., the arrival and service of customers at the information desk).

To study the properties of such real-world applications, several models have been proposed by researchers: the basic model in the discrete time domain is the discrete time Markov chains (*DTMC*s) model [26, 52], where the time is discrete (i.e., the system performs one operation per clock tick) and only probability determines the reached states. The continuous-time counterpart is known as the continuous-time Markov chains (*CTMC*s) [3, 5] model, where exponentially distributed sojourn times distributions control the evolution of the system.

*DTMC*s and *CTMC*s are purely probabilistic, and they have been extended with nondeterminism to permit different operations or behaviors from a specific state. This extension results to Markov decision processes (*MDP*s) [10, 31, 32, 44] and continuous-time Markov decision processes (*CTMDP*s) [7, 11, 31, 44, 54], respectively. These models, despite being widely used to represent and study real systems, are not fully compositional, that is, there is no guarantee that complex systems can be obtained by composing smaller components while preserving the intended behavior. This property is rather important as it is usually much easier to model and study (a set of) small systems and then combine them together rather than a single large system. Moreover, in the real world, usually applications and protocols involve several parties each one composed by modules working together in parallel. Two models have been proposed to achieve such compositional property: the probabilistic automata (*PA*s) model [47, 48] for discrete time systems and the interactive Markov chains (*IMC*s) [28] model for continuous-time systems. Recently one model has been proposed to unify and merge all such models in a single framework: the Markov automata (*MA*s) model [17, 22, 23]. This formalism is suitable for studying systems featuring continuous-time based behaviors as well as probabilistic and nondeterministic choices. Moreover, the Markov automata model provides the semantics to every generalized stochastic Petri net (GSPN) [19], a popular modelling formalism for performance and dependability analysis.

1.2 Comparing System Behaviors

Given a real world system we want to analyze, for instance by verifying whether it satisfies a set of properties, we can model it in several ways. This analysis is commonly known as *model checking*. In particular, we can decide to model it as a *DTMC* or as a *PA* whenever we are interested in its properties as a discrete time system; alternatively, if we want to study its behavior in continuous time, we can use *CTMC*s or *IMC*s. The choice of the model framework depends on the properties we are interested in and the details we want to consider.

Once the model framework has been chosen, the real system can be represented by several different models: for example, we can use different names for

the states, we can encode probabilistic choices as sequences of events or as single events, we can detail or abstract from particular details, and so on. It is clear that these choices affect the resulting model whose size may vary even if all these models represent the same real system.

A possible way to abstract away from this modelling details is to use the so called *simulation* and *bisimulation* relations that allow us to declare that two models are similar or equivalent whenever they are related, respectively. Intuitively, a system S_1 simulates a system S_2 if $S1$ is able to mimic whatever S_2 can do; the bisimulation requires that also S_2 simulates S_1. Usually, a simulation (or bisimulation) is defined as a binary relation over the states of the model and for each pair (s_1, s_2), if s_1 can perform a step, then s_2 has to match such step via its own steps in order to reach states that are related to the states reached from s_1. Depending on the steps s_2 is allowed to perform, simulation relations can be classified as *strong* (s_2 has to match with exactly one step) or as *weak* (s_2 is free to perform an arbitrary additional number of internal steps). Computing such simulation relations is rather easy by using classical refinement algorithms, provided that we have a procedure for deciding the existence of the matching step from s_2 given a step from s_1. As we will see in Section 6, such procedure is the only part that has to be changed in order to decide different simulations and it is also the bottleneck of the computation and the main source of the complexity of the decision procedure.

We are interested in systems related by a simulation relation since also the properties they satisfy are related, so we can check whether the real world system satisfies a given property by verifying it in one of the similar models: the theory ensures us that the evaluation of the property does not depend on the specific model we consider to represent the real world system. When we consider the bisimulation relation, among all possible bisimilar models there is a unique minimal model (up to isomorphism) that represents the original system [21]: the *quotient* model. The quotient is the model with the minimum number of states and transitions still behaving as the system we want to analyze; this minimality mitigates the state explosion problem of the model checking [8, 14, 34] as well as it helps in reducing the computational effort needed to verify whether the desired properties are fulfilled. Moreover, the computation of the quotient automaton is independent on the properties we want to check, thus even if it may be rather time consuming, the overall gain it provides to the following model checking phase may justify it.

1.3 Optimization Problems

Optimization or mathematical programming uses mathematical techniques to find the best solution among a set of given alternatives. More precisely, an optimization problem asks for maximizing or minimizing a real valued function for which the variables take values from a permissible set. It includes many diverse areas such as decision theory [42], flow network optimization [1] and so on. Flow network optimization is a subclass of linear programming that has application

in a number of domains such as computer science, logistics, transportation systems. Although flow network based models are not as wide as models that can be formulated mathematically using linear or integer programming, they can be solved very quickly which enables them to be a powerful tool for decision making [1].

1.4 Probabilistic Systems vs. Optimization

To a casual observer, flow and optimization problems seem rather unrelated to probabilistic concurrent systems. In fact, as we have seen, the former aim to optimize problems like resource allocation or goods transportation and distribution while the latter model systems that run in parallel where the behavior depends on probabilistic events as well like random failures, errors, and choices needed to break symmetry. To a careful observer, flow and optimization problems and probabilistic concurrent systems are not so unrelated, since the probability mass concentrated in the initial state can be seen as a liquid that flows and distributes in the network representing the possible evolution of the system. To highlight this connection, in this survey we consider a selection of papers [29,30,55,57] that, together with other works in concurrency literature such as [2,4,13,15,20,21,43,45], make use of flow and optimization problems to decide or solve efficiently the challenges of probabilistic concurrent systems.

Organization of the paper. After the mathematical preliminaries in Section 2, we present in Section 3 the discrete and continuous-time models, followed in Section 4 by the simulation and bisimulation relations defined on them. We recall in Section 5 the theory about networks and flow problems that are widely used in Section 6 to efficiently compute simulations and bisimulations. We conclude the paper in Section 7.

2 Mathematical Preliminaries

2.1 Functions and Relations

Given a set X and $\bot \notin X$, we denote by $X_\bot$ the set $X \cup \{\bot\}$.

Let X, Y be two finite sets, $f\colon X \to \mathbb{R}$ and $g\colon X \times Y \to \mathbb{R}$ be two functions. For $X' \subseteq X$, we denote by $f(X')$ the value $f(X') = \sum_{x \in X'} f(x)$; for $x \in X$ and $Y' \subseteq Y$, $g(x, Y') = \sum_{y \in Y'} g(x, y)$ and similarly, for $y \in Y$ and $X' \subseteq X$, $g(X', y) = \sum_{x \in X'} g(x, y)$. Finally, we define for each $x \in X$ and $y \in Y$ the functions $g(x, \cdot)\colon Y \to \mathbb{R}$ and $g(\cdot, y)\colon X \to \mathbb{R}$ as $g(x, \cdot)(y') = g(x, y')$ for each $y' \in Y$ and $g(\cdot, y)(x') = g(x', y)$ for each $x' \in X$, respectively. Given two functions $f, g\colon X \to \mathbb{R}$ and $p \in \mathbb{R}$, we denote by $p \cdot f\colon X \to \mathbb{R}$ the function $(p \cdot f)(x) = p \cdot f(x)$ for each $x \in X$ and $f + g\colon X \to \mathbb{R}$ the function $(f + g)(x) = f(x) + g(x)$ for each $x \in X$.

For a function $f\colon X \to \mathbb{R}^{\geq 0}$, we denote by $\mathrm{Supp}(f)$ the *support* set $\mathrm{Supp}(f) = \{\, x \in X \mid f(x) > 0 \,\}$.

Given a relation $\mathcal{R} \subseteq X \times Y$ and the sets $X' \subseteq X$ and $Y' \subseteq Y$, we define $\mathcal{R}(X') = \{ y \in Y \mid \exists x \in X'.x \mathcal{R} y \}$ and $\mathcal{R}^{-1}(Y') = \{ x \in X \mid \exists y \in Y'.x \mathcal{R} y \}$.

Given a relation $\mathcal{R} \subseteq X \times X$, we call $\mathcal{R} \cap \mathcal{R}^{-1}$ the *kernel* of $\mathcal{R}$ and we denote by $\mathcal{R}_\perp \subseteq X_\perp \times X_\perp$ the relation $\mathcal{R} \cup \{ (\perp, x) \mid x \in X_\perp \}$.

2.2 Probability Distributions

For a set X, denote by $\mathrm{Disc}(X)$ the set of discrete probability distributions over X, and by $\mathrm{SubDisc}(X)$ the set of discrete sub-probability distributions over X. Since a discrete sub-probability distribution $\rho \in \mathrm{SubDisc}(X)$ can be seen as a function $\rho \colon X \to [0,1]$, we adopt the same terminology and operations. Given $\rho \in \mathrm{SubDisc}(X)$, we denote by $\rho(\perp)$ the value $1 - \rho(X)$ where $\perp \notin X$, and by $|\rho|$ the size $|\mathrm{Supp}(\rho)|$. We extend ρ to a probability distribution $\rho_\perp \in \mathrm{Disc}(X_\perp)$ by defining $\rho_\perp(\perp) = 1 - \rho(X)$ and $\rho_\perp(x) = \rho(x)$ for each $x \in X$. We denote by δ_x, where $x \in X_\perp$, the *Dirac* distribution such that $\delta_x(y) = 1$ for $y = x$, 0 otherwise. For a sub-probability distribution ρ, we also write $\rho = \{ (x, p_x) \mid x \in X \}$ where p_x is the probability of x. We say that ρ is *stochastic* if $\rho(X) = 1$ and *absorbing* if $\rho(\perp) = \delta_\perp$. We sometimes refer to $\rho(X)$ as the *mass* of ρ.

The lifting $\mathcal{L}(\mathcal{R}) \subseteq \mathrm{Disc}(X) \times \mathrm{Disc}(X)$ [34] of a relation $\mathcal{R} \subseteq X \times X$ to distributions is defined as: for $\rho_1, \rho_2 \in \mathrm{Disc}(X)$, $\rho_1 \; \mathcal{L}(\mathcal{R}) \; \rho_2$ holds if there exists a *weighting function* $w \colon X \times X \to [0,1]$ such that

1. for each $(x_1, x_2) \in X \times X$, $w(x_1, x_2) > 0$ implies $x_1 \; \mathcal{R} \; x_2$,
2. for each $x_1 \in X$, $w(x_1, X) = \rho_1(x_1)$, and
3. for each $x_2 \in X$, $w(X, x_2) = \rho_2(x_2)$.

This definition of lifting has been proposed for discrete systems [34, 50] and it is indeed equivalent [55] to the definition based on $\mathcal{R}$-closure introduced by [18] for non-discrete systems: the lifting $\mathcal{L}(\mathcal{R}) \subseteq \mathrm{Disc}(X) \times \mathrm{Disc}(X)$ of a relation $\mathcal{R} \subseteq X \times X$ is defined as: for $\rho_1, \rho_2 \in \mathrm{Disc}(X)$, $\rho_1 \; \mathcal{L}(\mathcal{R}) \; \rho_2$ holds if for each $X' \subseteq X$, $\rho_1(X') \leq \rho_2(\mathcal{R}(X'))$.

Extending the lifting to sub-distributions is rather easy [57]: for $\rho_1, \rho_2 \in \mathrm{SubDisc}(X)$, $\rho_1 \; \mathcal{L}(\mathcal{R}) \; \rho_2$ holds if there exists a *weighting function* $w \colon X_\perp \times X_\perp \to [0,1]$ such that

1. for each $(x_1, x_2) \in X_\perp \times X_\perp$, $w(x_1, x_2) > 0$ implies $x_1 \; \mathcal{R}_\perp \; x_2$,
2. for each $x \in X_\perp$, $w(x, X_\perp) = \rho_1(x)$, and
3. for each $x \in X_\perp$, $w(X_\perp, x) = \rho_2(x)$.

3 The Models

We now introduce the formal models for probabilistic concurrent systems we consider in this survey paper. We first recall the discrete time models and then the continuous-time models. In this work we consider only finite models, i.e., systems such that states, actions, and transition relations are finite.

3.1 Discrete Time Models

The first model we consider is the labelled substochastic discrete time Markov chain model where each state enables only a transition that may reach several states, each one with a given probability. The status of the system is represented by a set AP of atomic propositions that are true in the given state.

Definition 1 (Substochastic discrete time Markov chain [8, 33]). *A labelled* substochastic Discrete Time Markov Chain *(sDTMC) $\mathcal{S}$ is a tuple $\mathcal{S} = (S, \bar{s}, \mathbf{P}, L)$ where S is a finite set of* states*, $\bar{s}$ is the* start state*, $\mathbf{P}\colon S \times S \to [0,1]$ is a* probability matrix *such that $\mathbf{P}(s, \cdot) \in \mathrm{SubDisc}(S)$ for all $s \in S$, and $L\colon S \to 2^{AP}$ is a* labeling function*.*

Given a state s and the associated distribution $\mu_s = \mathbf{P}(s, \cdot) \in \mathrm{SubDisc}(S)$, we call (s, μ_s) a *transition* and we say that (s, μ_s) is enabled by s and that μ_s is the target of (s, μ_s).

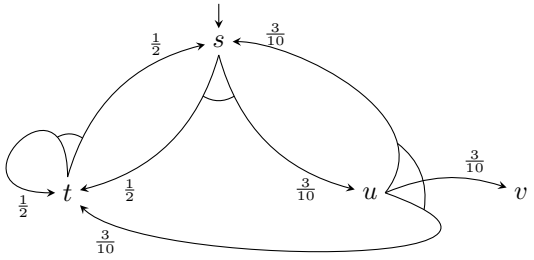


Figure 1: An example of substochastic discrete time Markov chain

Figure 1 shows an example of a *sDTMC*, where s is the initial state, denoted by the short incoming arrow. For each state, we represent the enabled transition by a set of arrows grouped by an arc and pointing to the target states, each one decorated with the corresponding probability. For example, the transition enabled by s reaches t and u with probability $\frac{1}{2}$ and $\frac{3}{10}$, respectively. As usual in this kind of representation of the model, to keep the picture clear we have omitted the arrows reaching states with probability 0. For instance, from s there should be also an arrow reaching v with probability 0. As labels of the states, we take $AP = S = \{s, t, u, v\}$ and we let $L(z) = z$ for each state $z \in S$. Note that the transitions from both s and u have as target a sub-probability distribution that is not a probability distribution. In fact, for the transition (s, μ_s) the mass of μ_s is $\frac{8}{10}$ and the transition (u, μ_u) the mass of μ_u is $\frac{9}{10}$.

We call a state s *stochastic* (*absorbing*) if the distribution $\mathbf{P}(s, \cdot)$ is stochastic (absorbing) respectively. For the *sDTMC* in Figure 1, t is stochastic, v is absorbing while both s and u are neither stochastic nor absorbing. If we restrict the states of a *sDTMC* to be either stochastic or absorbing, we obtain a discrete time Markov chain:

Definition 2 (Discrete time Markov chain [26, 52]). *A labelled* Discrete Time Markov Chain *(DTMC) $\mathcal{D}$ is a labelled sDTMC $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$ such that for each state $s \in S$, $\mathbf{P}(s, S) \in \{0, 1\}$.*

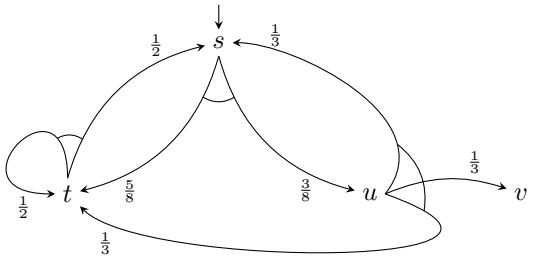


Figure 2: An example of discrete time Markov chain

Figure 2 shows an example of a *DTMC*. It is actually the *sDTMC* in Figure 1 where probability distributions have been normalized to have mass 1.

These two models are suitable for systems exhibiting only probabilistic behaviors, that is, they are not able to represent systems where different transitions are available from the states. For instance, the system that is in a particular state may react differently to different stimuli and this can be modeled by performing different transitions leading to different distributions over the states of the system. We call this capacity nondeterminism that is encoded, together with probability, by the following two discrete time models: Markov decision processes and probabilistic automata. In order to have a uniform approach, for probabilistic automata we adopt the notation of [57] instead of the one used in [47, 48].

Definition 3 (Probabilistic automaton [47, 48]). *A* Probabilistic Automaton *(PA) $\mathcal{P}$ is a tuple $\mathcal{P} = (S, \bar{s}, \Sigma, \rightarrow, L)$ where S is a finite set of* states*, $\bar{s}$ is the* start state*, Σ is a finite set of* actions*, $\rightarrow \subseteq S \times \Sigma \times \mathrm{Disc}(S)$ is a finite* probabilistic transition relation*, and $L\colon S \rightarrow 2^{AP}$ is a* labeling function*.*

The set Σ is divided in two sets $\mathtt{H}$ and $\mathtt{E}$ of *internal* (*hidden*) and *external* actions, respectively. We remark that the definition of probabilistic automata we are presenting here is different from the original one given by Segala in [48] named *simple probabilistic automata*, but currently known as just probabilistic automata. In fact, in such work (simple) probabilistic automata are defined as follows (cf. [48, Section 3.1]): A Probabilistic Automaton (*PA*) $\mathcal{P}$ is a tuple $(S, \bar{s}, \Sigma, \rightarrow)$ where S is a countable set of *states*, $\bar{s} \in S$ is the *start state*, Σ is a countable set of *actions*, and $\rightarrow \subseteq S \times \Sigma \times \mathrm{Disc}(S)$ is a *probabilistic transition relation*. The main difference with Definition 3 is that in [48] there is no labeling function. This difference can be easily bridged by defining L as $L(s) = \emptyset$ for each $s \in S$. Figure 3 shows an example of a *PA* where $\mathtt{H} = \{\tau\}$ and $\mathtt{E} = \{a, b\}$.

In a probabilistic automaton $\mathcal{P}$ we can distinguish between two kinds of nondeterminism: external and internal nondeterminism. We say that a state s

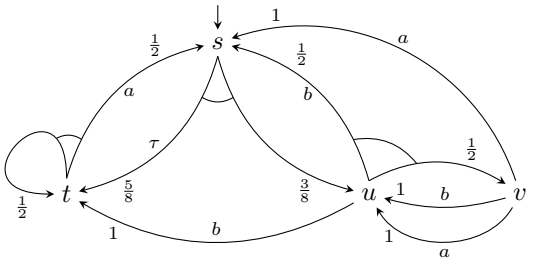


Figure 3: An example of probabilistic automaton

exhibits external nondeterminism if there exist two different actions a and b such that $(s, a, \mu_a) \in \rightarrow$ and $(s, b, \mu_b) \in \rightarrow$ for some $\mu_a, \mu_b \in \mathrm{Disc}(S)$. For instance, this is the case for the state v of the *PA* in Figure 3 since we have the two transitions (v, a, δ_s) and (v, b, δ_u). On the other hand, we say that a state s exhibits internal nondeterminism if there exist an action a and two different distributions $\mu_1, \mu_2 \in \mathrm{Disc}(S)$ such that $(s, a, \mu_1) \in \rightarrow$ and $(s, a, \mu_2) \in \rightarrow$. This happens for the state u that enables two different transitions both with action b. Note that a state may exhibit both internal and external nondeterminism (as happens for v) or none of them (see states s and t).

3.2 Continuous-Time Models

We now consider the continuous-time counterparts of the previous models, where state transitions are governed by the passing of the time. Essentially, they are defined as the discrete time models except for the probability distributions that are replaced by transition rates, i.e., the speed of transition firing.

The first model we recall is about continuous-time Markov chains that are just discrete time Markov chains where the probability matrix is replaced by the rate matrix.

Definition 4 (Continuous-time Markov chain [5, 44, 55]).
A labelled Continuous-Time Markov Chain *(CTMC)* $\mathcal{C}$ *is a tuple* $\mathcal{C} = (S, \bar{s}, \mathbf{R}, L)$ *such that* S*,* $\bar{s}$*, and* L *are defined as for DTMC and* $\mathbf{R}\colon S \times S \to \mathbb{R}^{\geq 0}$ *is the* rate matrix*.*

Note that the usual definition of CTMCs, such as the one in [3], requires that $\mathbf{R}\colon S \times S \to \mathbb{R}$ where for each $s \in S$, $\mathbf{R}(s, s') \geq 0$ for each $s' \neq s$ and $\mathbf{R}(s, s) = -\sum_{s' \neq s} \mathbf{R}(s, s')$. As remarked in [5], allowing self loops neither alters the transient nor the steady-state behavior of the CTMC, but it allows the usual interpretation of the linear-time CSL operators like next-step and until.

Figure 4 shows an example of a *CTMC*. Greek letters λ, κ, and ρ are the rates governing the speed of the firing of the transitions. So, for example, the λ on the transition from s to t means that $\mathbf{R}(s, t) = \lambda$. We omitted the transitions with rate 0 to keep the picture clear.

The probability of performing a transition and reaching a given state can be computed as follows: starting from the state s, the probability of performing a

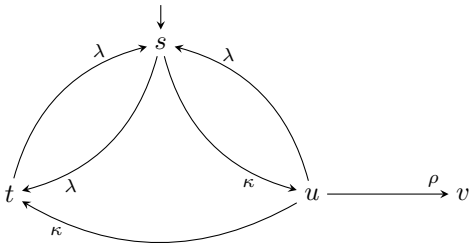


Figure 4: An example of continuous-time Markov chain

transition within time t is $1-e^{-\mathbf{R}(s,S)\cdot t}$ and the probability of reaching the state s' with this transition is $(1-e^{-\mathbf{R}(s,S)\cdot t})\cdot\frac{\mathbf{R}(s,s')}{\mathbf{R}(s,S)}$.

This allows us to consider the *DTMC* embedded into a *CTMC* that captures the system behavior after abstracting away the time:

Definition 5 (Embedded *DTMC* [5, 55, 57]). *Let $\mathcal{C}$ be a CTMC. The* embedded *DTMC $\mathcal{D}$ of $\mathcal{C}$ is defined by $emb(\mathcal{C}) = (S, \bar{s}, \mathbf{P}, L)$ where for each $s, s' \in S$, $\mathbf{P}(s,s')$ is defined as $\mathbf{P}(s,s') = \frac{\mathbf{R}(s,s')}{\mathbf{R}(s,S)}$ if $\mathbf{R}(s,S) > 0$, and $\mathbf{P}(s,s') = 0$ otherwise.*

Similarly to *CTMC*s and *DTMC*s, the continuous-time counterparts of *PA*s, called continuous-time probabilistic automata (*CTPA*), are obtained by replacing the transition relation with a rate matrix.

We call a function $r\colon S \to \mathbb{R}^{\geq 0}$ a *rate function* and we denote the set of all rate functions by $Rate(S)$. Given the rate function r, we call $r(S)$ the *exit rate*. Given $\mathbf{R}$ and a state s of a *CTMC* $\mathcal{C}$, we call $\mathbf{R}(s,\,\cdot\,)\colon S \to \mathbb{R}^{\geq 0}$ the *rate function* associated with s and we usually denote it by r_s.

Definition 6 (Continuous-time probabilistic automaton [11, 37, 44]). *A* Continuous Time Probabilistic Automaton *(CTPA) $\mathcal{CP}$ is a tuple $\mathcal{CP} = (S, \bar{s}, \Sigma, \mathbf{R}, L)$, where S is a finite set of* states*, $\bar{s}$ is the* start state*, Σ is a finite set of* actions*, $\mathbf{R} \subseteq S \times \Sigma \times Rate(S)$ is a finite* rate matrix*, and $L\colon S \to 2^{AP}$ is a* labeling function*.*

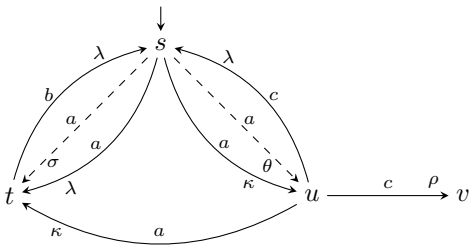


Figure 5: An example of continuous-time probabilistic automaton

Figure 5 shows an example of *CTPA*; arrows emanating from a state with the same label and shape belong to the same transition. For instance, state s enables

two transitions (s, a, r) and (s, a, r') with $r, r' \in \mathit{Rate}(S)$ such that $r(t) = \sigma$, $r(u) = \theta$, and $r(s) = r(v) = 0$ and $r'(t) = \lambda$, $r'(u) = \kappa$, and $r'(s) = r'(v) = 0$, respectively.

3.3 Mixed Discrete and Continuous-Time Models

We now present two models that merge continuous-time and discrete time behavior, the interactive Markov chains and the Markov automata. They exhibit continuous-time behavior like *CTMC*s, where transitions are fired by the passage of the time, as well as discrete time behavior like labelled transitions systems where transitions are fired by actions. These two models are especially suited for compositional reasoning over continuous-timed systems due to the separation of action and Markovian transitions and the maximal progress assumption, that is, if a state enables both timed transitions and internally labelled transitions, then the latter take precedence and the former are ignored.

Definition 7 (Markov automaton [17,22,23]). *A* Markov Automaton *(MA)* $\mathcal{MA}$ *is a tuple* $\mathcal{MA} = (S, \bar{s}, \Sigma, \rightarrow, \mathbf{R}, L)$ *where* S *is a finite set of* states*,* $\bar{s}$ *is the* start state*,* Σ *is a finite set of* actions*,* $\rightarrow \subseteq S \times \Sigma \times \mathrm{Disc}(S)$ *is a finite* probabilistic transition relation*,* $\mathbf{R} \subseteq S \times \mathbb{R}^{\geq 0} \times S$ *is a finite set of* timed transitions*, and* $L \colon S \to 2^{AP}$ *is a* labeling function*.*

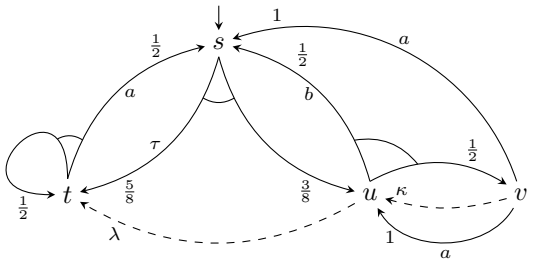


Figure 6: An example of Markov automaton

Figure 6 shows an example of a *MA*. As for the *CTMC* in Figure 4, we use Greek letters λ and κ for the rates governing the speed of the firing of the transitions that we represent by dashed arrows in order to distinguish them from probabilistic transitions.

An interactive Markov chain is an *MA* such that each probabilistic transition leads to a Dirac distribution, i.e., to a single state:

Definition 8 (Interactive Markov chain [28]). *An* Interactive Markov Chain *(IMC)* $\mathcal{I}$ *is a tuple* $\mathcal{I} = (S, \bar{s}, \Sigma, \rightarrow, \mathbf{R}, L)$ *where* S *is a finite set of* states*,* $\bar{s}$ *is the* start state*,* Σ *is a finite set of* actions*,* $\rightarrow \subseteq S \times \Sigma \times S$ *is an* interactive transition relation*,* $\mathbf{R} \subseteq S \times \mathbb{R}^{\geq 0} \times S$ *is a finite set of* timed transitions*, and* $L \colon S \to 2^{AP}$ *is a* labeling function*.*

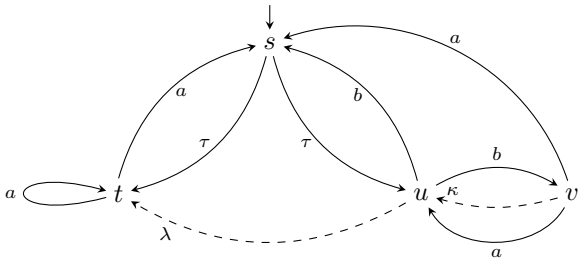


Figure 7: An example of interactive Markov chain

Figure 7 shows an example of an *IMC*. In particular, *IMC* can be seen as the merger of labelled transitions systems and *CTMC*s while *MA* can be seen as the merger of *PA*s and *CTMC*s. In fact, each model is an instance of the *MA* model with specific restrictions on $\rightarrow$ and $\mathbf{R}$ (cf. [22, Section 3]). As for probabilistic automata, the original definitions do not involve the labeling function L that we have added for uniformity. Again, the original model can be recovered by defining $L(s) = \emptyset$ for each $s \in S$.

3.4 Terminology and Notation

In the remaining of the paper we adopt the following terminology and notation, given in the context of probabilistic automata [27, 29, 47, 48, 57].

We refer to each instance of the discrete and continuous-time models as *automaton* and we denote it by $\mathcal{A}$, that is, we use the term (discrete time) automaton and $\mathcal{A}$ for the *sDTMC* $\mathcal{S}$, the *DTMC* $\mathcal{D}$, and the *PA* $\mathcal{P}$ as well as the term (continuous-time) automaton and $\mathcal{A}$ for the *CTMC* $\mathcal{C}$ and the *CTPA* $\mathcal{CP}$.

Given a *PA* $\mathcal{P}$, we let s,t,u,v, and their variants with indices range over S; a, b range over actions; and τ range over internal actions. A transition $tr = (s, a, \mu) \in \rightarrow$, also denoted by $s \xrightarrow{a} \mu$, is said to *leave* from state s, to be *labelled* by a, and to *lead* to the *target* distribution μ, also denoted by μ_{tr}. We denote by $src(tr)$ the *source* state s and by $act(tr)$ the *action* a. We also say that s enables action a, that action a is enabled from s, and that (s, a, μ) is enabled from s. Finally, we let $s \longrightarrow = \{\, tr \in \rightarrow \mid src(tr) = s \,\}$ be the set of transitions enabled by s and $\xrightarrow{a} = \{\, tr \in \rightarrow \mid act(tr) = a \,\}$ be the set of transitions with label a.

An *execution fragment* of a *PA* $\mathcal{P}$ is a finite or infinite sequence of alternating states and actions $\alpha = s_0 a_1 s_1 a_2 s_2 \ldots$ starting from a state s_0, also denoted by $first(\alpha)$, and, if the sequence is finite, ending with a state denoted by $last(\alpha)$, such that for each $i > 0$ there exists a transition $(s_{i-1}, a_i, \mu_i) \in \rightarrow$ such that $\mu_i(s_i) > 0$. The *length* of α, denoted by $len(\alpha)$, is the number of occurrences of actions in α. If α is infinite, then $len(\alpha) = \infty$. Denote by $frags(\mathcal{P})$ the set of execution fragments of $\mathcal{P}$ and by $frags^*(\mathcal{P})$ the set of finite execution fragments of $\mathcal{P}$. An execution fragment α is a *prefix* of an execution fragment α', denoted by $\alpha \leqslant \alpha'$, if the sequence α is a prefix of the sequence α'. The *trace* $trace(\alpha)$ of α is the sub-sequence of external actions of α; we denote by ε the empty trace and we define $trace(a) = a$ for $a \in \mathtt{E}$ and $trace(a) = \varepsilon$ for $a \in \mathtt{H}$.

We extend the above terminology to the other models introduced so far, when applicable; in particular, we use $\rightarrow$ to denote the transition relations $\mathbf{P}$ and $\mathbf{R}$ of *DTMC*s and *sDTMC*s, and of *CTMC*s and *CTPA*s, respectively. For instance, given a *DTMC* $\mathcal{D}$, a state s, and a probability distribution μ, we still call (s, τ, μ) a transition, denoted by $s \xrightarrow{\tau} \mu$, also written $(s, \tau, \mu) \in \mathbf{P}$, provided that $\mu = \mathbf{P}(s, \cdot)$. Note that here τ denotes just a step since it is not an actual action labeling the transition. Similarly, given a *CTPA* $\mathcal{CP}$, a state s, an action a, and a rate function r, we still write $(s, a, r) \in \rightarrow$ if $\mathbf{R}(s, a) = r$ and we call (s, a, r) a transition, denoted by $s \xrightarrow{a} r$ as well.

For a *CTPA* $\mathcal{CP}$ and a rate function r, we denote by $\mu_r \in \mathrm{SubDisc}(S)$ the induced sub-probability distribution defined by: if $r(S) > 0$, then for each $s \in S$, $\mu_r(s) = \frac{r(s)}{r(S)}$, and if $r(S) = 0$, then $\mu_r = \delta_\perp$.

We adopt a similar notation also for *CTMC*s: for a *CTMC* $\mathcal{C}$ and a state s, we denote by $\mu_{r_s} \in \mathrm{SubDisc}(S)$ the sub-probability distribution induced by the rate function $r_s = \mathbf{R}(s, \cdot)$, i.e., $\mu_{r_s} = \mathbf{P}(s, \cdot)$ for the embedded *DTMC* $emb(\mathcal{C})$.

Given an automaton $\mathcal{A}$ and a state s, we denote by $post(s)$ the set of successors of the state s, that is, $post(s) = \mathrm{Supp}(\mathbf{P}(s, \cdot))$ if $\mathcal{A}$ is a *DTMC* or a *sDTMC*, and $post(s) = \{\, s' \in S \mid \mathbf{R}(s, s') > 0 \,\}$ if $\mathcal{A}$ is a *CTMC*. For a *sDTMC* $\mathcal{S}$ and a state s, we denote by $post_\perp(s)$ the set $post_\perp(s) = \mathrm{Supp}(\mu_\perp)$ where $\mu = \mathbf{P}(s, \cdot)$. Similarly, we denote by $pre(s)$ the set of predecessors of the state s, that is, $pre(s) = \{\, s' \in S \mid \mathbf{P}(s', s) > 0 \,\}$ if $\mathcal{A}$ is a *DTMC* or a *sDTMC*, and $pre(s) = \{\, s' \in S \mid \mathbf{R}(s', s) > 0 \,\}$ if $\mathcal{A}$ is a *CTMC*. Finally, we denote by $reach(s)$ the states that are reachable with positive probability from s, that is, $reach(s) = \{\, t \in S \mid \exists \alpha \in frags^*(\mathcal{A}).last(\alpha) = t \,\}$.

4 Simulations and Bisimulations

We recall now the main behavioral preorders and equivalences that are used for the models presented in Section 3. These relations allow us to relate system that are syntactically different, for instance because they use different names for the states, but exhibit equivalent behaviors. Moreover, they allow to reduce the size of the automata without changing their properties. This is especially useful to mitigate the state space explosion problem that usually happens in model checking [8, 14, 34]. An empirical investigation to show the effectiveness of such behavioral relations minimization is performed in [36]. This study indicates that for traditional model checking, huge state space reductions (up to logarithmic) may be acquired. It is worthwhile to mention that the definition of such relations is based on a single automaton; however, as we will see, they are usually used to relate two automata $\mathcal{A}_1$ and $\mathcal{A}_2$. This technical problem is easily solved by taking the disjoint union of the two automata, that is, the automaton whose set of states is the disjoint union of the sets of states of $\mathcal{A}_1$ and $\mathcal{A}_2$, and whose other components are the union of the corresponding components of $\mathcal{A}_1$ and $\mathcal{A}_2$.

4.1 Strong Simulation and Bisimulation

The first relations we introduce are the strong simulation and bisimulation, that are the natural extension to probabilistic systems of the homonymous relations for labelled transition systems [39].

Definition 9 (Strong simulation for discrete time probabilistic automata [9, 50, 56, 57]). *Let $\mathcal{A}$ be a discrete time probabilistic automaton. A relation $\mathcal{R}$ on S is a* strong simulation *if, for each pair of states $s, t \in S$ such that $s\ \mathcal{R}\ t$,*

- *$L(s) = L(t)$ and*
- *if $s \xrightarrow{a} \mu_s$ for some probability distribution μ_s, then there exists μ_t such that $t \xrightarrow{a} \mu_t$ and $\mu_s\ \mathcal{L}(\mathcal{R})\ \mu_t$.*

We say that the discrete time automaton $\mathcal{A}_2$ strongly simulates *$\mathcal{A}_1$ if there exists a strong simulation $\mathcal{R}$ on the disjoint union $S_1 \uplus S_2$ such that $\bar{s}_1\ \mathcal{R}\ \bar{s}_2$ and we say that the state t strongly simulates the state s if there exists a strong simulation $\mathcal{R}$ such that $s\ \mathcal{R}\ t$. We denote the coarsest strong simulation, called strong similarity, by $\lesssim$.*

In the remaining of the paper and similarly for the following simulations, we refer to the second condition (*if $s \xrightarrow{a} \mu_s$ for some probability distribution μ_s, then there exists μ_t such that $t \xrightarrow{a} \mu_t$ and $\mu_s\ \mathcal{L}(\mathcal{R})\ \mu_t$*) as the *step condition* since it ensures that from two similar states s and t, each transition (or *step*) from s is matched by a transition/step from t and the reached states are still related according to the lifting of the reached distributions.

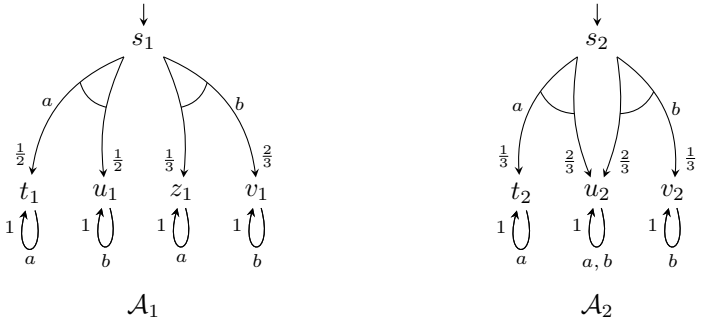


Figure 8: Two *PA*s with $L(x) = \emptyset$ for each state x such that $\mathcal{A}_1 \lesssim \mathcal{A}_2$. The single transition from u_2 with action a, b is just a compact form for the two transitions (u_2, a, δ_{u_2}) and (u_2, b, δ_{u_2})

Figure 8 shows two probabilistic automata $\mathcal{A}_1$ and $\mathcal{A}_2$ such that $\mathcal{A}_1 \lesssim \mathcal{A}_2$. In fact, consider the relation $\mathcal{R} = \{(s_1, s_2), (t_1, t_2), (u_1, u_2), (v_1, v_2), (z_1, u_2)\}$; it is rather easy to verify that $\mathcal{R}$ satisfies Definition 9: it is trivial to verify the step condition for the pairs (t_1, t_2), (u_1, u_2), (v_1, v_2), and (z_1, u_2). The only interesting case is the pair (s_1, s_2); the transition $s_1 \xrightarrow{a} \mu_{1a}$ with $\mu_{1a} = \{(t_1, \frac{1}{2}), (u_1, \frac{1}{2})\}$ is matched by s_2 via the transition $s_2 \xrightarrow{a} \mu_{2a}$ with $\mu_{2a} = \{(t_2, \frac{1}{3}), (u_2, \frac{2}{3})\}$ such

that $\mu_{1a}\ \mathcal{L}(\mathcal{R})\ \mu_{2a}$. The weighting function [34,56,57] w_a justifying $\mu_{1a}\ \mathcal{L}(\mathcal{R})\ \mu_{2a}$ is defined as follows:

$$w_a(x_1, x_2) = \begin{cases} \frac{1}{3} & \text{if } x_1 = t_1 \text{ and } x_2 = t_2, \\ \frac{1}{6} & \text{if } x_1 = t_1 \text{ and } x_2 = u_2, \\ \frac{1}{2} & \text{if } x_1 = u_1 \text{ and } x_2 = u_2, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Similarly, the transition $s_1 \xrightarrow{b} \mu_{1b}$ with $\mu_{1b} = \{(z_1, \frac{1}{3}), (v_1, \frac{2}{3})\}$ is matched by s_2 via the transition $s_2 \xrightarrow{b} \mu_{2b}$ with $\mu_{2b} = \{(u_2, \frac{2}{3}), (v_2, \frac{1}{3})\}$ such that $\mu_{1b}\ \mathcal{L}(\mathcal{R})\ \mu_{2b}$. The weighting function w_b justifying $\mu_{1b}\ \mathcal{L}(\mathcal{R})\ \mu_{2b}$ is:

$$w_b(x_1, x_2) = \begin{cases} \frac{1}{3} & \text{if } x_1 = z_1 \text{ and } x_2 = u_2, \\ \frac{1}{3} & \text{if } x_1 = v_1 \text{ and } x_2 = u_2, \\ \frac{1}{3} & \text{if } x_1 = v_1 \text{ and } x_2 = v_2, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

The definition of strong simulation for continuous-time automata is almost the same, except for the fact that we require that t can move stochastically *faster* than s, i.e., t has a rate higher than s:

Definition 10 (Strong simulation for continuous-time probabilistic automata [9,56,57]). *Let $\mathcal{A}$ be a continuous-time probabilistic automaton. A relation $\mathcal{R}$ on S is a* strong simulation *if, for each pair of states $s, t \in S$ such that $s\ \mathcal{R}\ t$,*

- *$L(s) = L(t)$ and*
- *if $s \xrightarrow{a} r_s$ for some rate function r_s, then there exists a rate function r_t such that $t \xrightarrow{a} r_t$, $\mu_{r_s}\ \mathcal{L}(\mathcal{R})\ \mu_{r_t}$, and $r_s(S) \leq r_t(S)$.*

We say that the continuous-time automaton $\mathcal{A}_2$ strongly simulates *$\mathcal{A}_1$ if there exists a strong simulation $\mathcal{R}$ on the disjoint union $S_1 \uplus S_2$ such that $\bar{s}_1\ \mathcal{R}\ \bar{s}_2$ and we say that the state t strongly simulates the state s if there exists a strong simulation $\mathcal{R}$ such that $s\ \mathcal{R}\ t$. We denote the coarsest strong simulation, called strong similarity, by $\lesssim$.*

Figure 9 shows two continuous time probabilistic automata $\mathcal{A}_1$ and $\mathcal{A}_2$ such that $\mathcal{A}_1 \lesssim \mathcal{A}_2$. The relation $\mathcal{R} = \{(s_1, s_2), (t_1, t_2), (u_1, u_2), (t_1, u_2), (u_1, t_2)\}$ indeed justifies $\mathcal{A}_1 \lesssim \mathcal{A}_2$: consider for instance the pair (t_1, t_2); the rate function r_{t_1} induces the probability distribution $\mu_{r_{t_1}} = \delta_{u_1}$ and the overall rate $r_{t_1}(S) = \lambda$. For t_2, we have the rate function r_{t_2} that induces the probability distribution $\mu_{r_{t_2}} = \delta_{u_2}$ and the overall rate $r_{t_1}(S) = 3\lambda$, thus $r_{t_1}(S) \leq r_{t_2}(S)$. Since $(u_1, u_2) \in \mathcal{R}$, then $\delta_{u_1}\ \mathcal{L}(\mathcal{R})\ \delta_{u_2}$ is trivially satisfied, hence the step condition is satisfied. A similar argument shows that the step condition is satisfied for the pairs (u_1, u_2), (t_1, u_2), and (u_1, t_2).

Now, consider the pair (s_1, s_2): we distinguish the case of the transitions with label b and c and the transition with label a, all from s_1. The transition from s_1

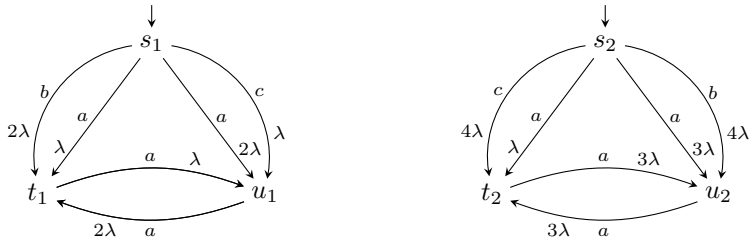


Figure 9: Two *CTPA*s with $L(s_1) = L(s_2) = \{s\}$ and $L(x) = \emptyset$ for each remaining state x such that $\mathcal{A}_1 \lesssim \mathcal{A}_2$

with label b induces the probability distribution $\mu_{r^b_{s_1}} = \delta_{t_1}$ and the overall rate $r^b_{s_1}(S) = 2\lambda$. For s_2, we have the rate function $r^b_{s_2}$ that induces the probability distribution $\mu_{r^b_{s_2}} = \delta_{u_2}$ and the overall rate $r^b_{s_2}(S) = 4\lambda$, thus $r^b_{s_1}(S) \leq r^b_{s_2}(S)$. Since $(t_1, u_2) \in \mathcal{R}$, then $\delta_{t_1}\ \mathcal{L}(\mathcal{R})\ \delta_{u_2}$ trivially holds, hence the step condition is satisfied. The case for the label c is similar.

The last step condition we have to check involves the transition with label a from s_1. The rate function $r^a_{s_1}$ induces the probability distribution $\mu_{r^a_{s_1}} = \{(t_1, \frac{\lambda}{3\lambda}), (u_1, \frac{2\lambda}{3\lambda})\}$ and the overall rate $r^a_{s_1}(S) = 3\lambda$. For s_2, we have the rate $r^a_{s_2}$ that induces the probability distribution $\mu_{r^a_{s_2}} = \{(t_2, \frac{\lambda}{4\lambda}), (u_2, \frac{3\lambda}{4\lambda})\}$ and the overall rate $r^a_{s_2}(S) = 4\lambda$. Obviously, $r^a_{s_1}(S) \leq r^a_{s_2}(S)$; $\mu_{r^a_{s_1}}\ \mathcal{L}(\mathcal{R})\ \mu_{r^a_{s_2}}$ is justified by the weighting function w defined as

$$w(x_1, x_2) = \begin{cases} \frac{1}{4} & \text{if } x_1 = t_1 \text{ and } x_2 = t_2, \\ \frac{1}{12} & \text{if } x_1 = t_1 \text{ and } x_2 = u_2, \\ \frac{2}{3} & \text{if } x_1 = u_1 \text{ and } x_2 = u_2, \text{ and} \\ 0 & \text{otherwise} \end{cases}$$

The definition of strong bisimulation and strong bisimilarity, denoted by $\sim$, is obtained by requiring $\mathcal{R}$ to be a symmetric relation.

Definition 11 (Strong bisimulation [38]). *Let $\mathcal{A}$ be a discrete time or a continuous-time probabilistic automaton. A relation $\mathcal{R}$ on S is a* strong bisimulation *if $\mathcal{R}$ is symmetric and a strong simulation.*

We denote the coarsest strong bisimulation, called strong bisimilarity, by $\sim$.

Other definitions of strong bisimulation require $\mathcal{R}$ to be an equivalence relation but it is easy to show that such definitions are equivalent to Definition 11.

Finally, only strong bisimulation on *IMC*s has been defined [28], and it is the expected merge of the bisimulation for *CTMC*s and labelled transition systems:

Definition 12 (Strong bisimulation for *IMC*s [28]). *Let $\mathcal{I}$ be a IMC. An equivalence relation $\mathcal{R}$ on S is a* strong bisimulation *if, for each pair of states $s, t \in S$ such that $s\ \mathcal{R}\ t$,*

- $L(s) = L(t)$,
- *if* $s \xrightarrow{a} s'$ *for some* $s' \in S$ *and* $a \in \Sigma$, *then there exists* t' *such that* $t \xrightarrow{a} t'$ *and* $s' \mathcal{R} t'$, *and*
- *if* s *does not enable a transition with label* τ, *then for each* $\mathcal{C} \in S/\mathcal{R}$, $\gamma(s, \mathcal{C}) = \gamma(t, \mathcal{C})$ *where* $\gamma(v, \mathcal{C}) = \sum_{\{\, \lambda \in \mathbb{R}^{\geq 0} \mid v \xrightarrow{\lambda} v', v' \in \mathcal{C} \,\}} \lambda$.

We say that the IMC $\mathcal{A}_2$ strongly bisimulates $\mathcal{A}_1$ *if there exists a strong bisimulation* $\mathcal{R}$ *on the disjoint union* $S_1 \uplus S_2$ *such that* $\bar{s}_1 \mathcal{R} \bar{s}_2$ *and we say that the state* t *strongly bisimulates the state* s *if there exists a strong bisimulation* $\mathcal{R}$ *such that* $s \mathcal{R} t$. *We denote the coarsest strong bisimulation, called strong bisimilarity, by* $\sim$.

A simulation (and a bisimulation) can be seen as a game where in each round the challenger, or attacker, s proposes a transition, or step, that has to be matched by the defender t. The two states s and t are strong (bi-)similar if the defender is always able to match the challenging transitions proposed by the attacker, that is, the game can be played forever.

4.2 Strong Probabilistic Simulation and Bisimulation

The fact that (continuous-time) probabilistic automata may exhibit internal nondeterminism, i.e., a state can enable different transitions with the same label, allows us to define the probabilistic counterpart of strong simulation and bisimulation where each transition proposed by the challenger is matched by some convex combination of the defender's enabled transitions.

Given a *PA* $\mathcal{P}$, a state $s \in S$, an action $a \in \Sigma$, and a distribution $\mu \in \mathrm{Disc}(S)$, we say that there exists a *combined transition* $s \xrightarrow{a}_{\mathrm{C}} \mu$ if there exists a finite set I of indexes, a family $\{p_i\}_{i \in I} \subseteq [0,1]$ such that $\sum_{i \in I} p_i = 1$, and a family $\{s \xrightarrow{a} \mu_i\}_{i \in I} \subseteq \rightarrow$ such that $\mu = \sum_{i \in I} p_i \cdot \mu_i$.

Definition 13 (Strong probabilistic simulation for *PA*s [49, 50]). *Let* $\mathcal{A}$ *be a PA. A relation* $\mathcal{R}$ *on* S *is a* strong probabilistic simulation *if, for each pair of states* $s, t \in S$ *such that* $s \mathcal{R} t$,

- $L(s) = L(t)$ *and*
- *if* $s \xrightarrow{a} \mu_s$ *for some probability distribution* μ_s, *then there exists* μ_t *such that* $t \xrightarrow{a}_{\mathrm{C}} \mu_t$ *and* $\mu_s \, \mathcal{L}(\mathcal{R}) \, \mu_t$.

We say that the PA $\mathcal{P}_2$ strongly probabilistically simulates $\mathcal{P}_1$ *if there exists a strong probabilistic simulation* $\mathcal{R}$ *on the disjoint union* $S_1 \uplus S_2$ *such that* $\bar{s}_1 \mathcal{R} \bar{s}_2$ *and we say that the state* t *strongly probabilistically simulates the state* s *if there exists a strong probabilistic simulation* $\mathcal{R}$ *such that* $s \mathcal{R} t$. *We denote the coarsest strong probabilistic simulation, called strong probabilistic similarity, by* $\lesssim_p$.

Figure 10 shows two *PA*s $\mathcal{A}_1$ and $\mathcal{A}_2$ such that $\mathcal{A}_1 \lesssim_p \mathcal{A}_2$. The relation justifying $\mathcal{A}_1 \lesssim_p \mathcal{A}_2$ is $\mathcal{R} = \{\, (x_1, x_2) \mid x \in \{s, t, u\} \,\}$. All cases are trivial, except for the pair (s_1, s_2) and the transition $s_1 \xrightarrow{a} \mu 1$ with $\mu_1 = \{(t_1, \frac{1}{2}), (u_1, \frac{1}{2})\}$.

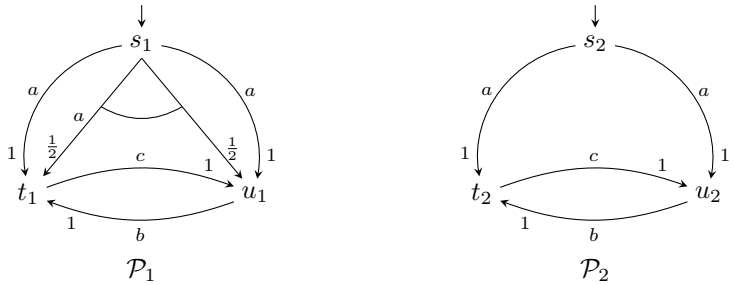


Figure 10: Two *PA*s with $L_i(x_i) = \{x\}$ for each state x_i, $i = 1, 2$ such that $\mathcal{A}_1 \lesssim_p \mathcal{A}_2$

This transition is matched by s_2 via the combined transition $s_2 \xrightarrow{a} \mu 2$ with $\mu_2 = \{(t_2, \frac{1}{2}), (u_2, \frac{1}{2})\}$. Such combined transition is obtained by taking transitions $s_2 \xrightarrow{a} \delta_{t_2}$ and $s_2 \xrightarrow{a} \delta_{u_2}$ both with probability $\frac{1}{2}$.

The definition of combined transition for *CTPA*s requires to consider for the convex combination only transitions with the same exit rate, in order to obtain a combined transition that is still exponentially distributed (see [57, Example 2.17] for more details).

Given a *CTPA* $\mathcal{CP}$, a state $s \in S$, an action $a \in \Sigma$, and a rate function $r \colon S \to \mathbb{R}^{\geq 0}$, we say that there exists a *combined transition* $s \xrightarrow{a}_{\mathrm{C}} r$ if there exists a finite set I of indexes, a family $\{p_i\}_{i \in I} \subseteq [0, 1]$ such that $\sum_{i \in I} p_i = 1$, and a family $\{s \xrightarrow{a} r_i\}_{i \in I} \subseteq \mathbf{R}$ such that $r_i(S) = r_j(S)$ for each $i, j \in I$ and $r = \sum_{i \in I} p_i \cdot r_i$.

As before, the definition of strong probabilistic simulation for *CTPA*s is the obvious continuous-time counterpart of the definition for *PA*s:

Definition 14 (Strong probabilistic simulation for *CTPA*s [9, 28, 56, 57]). *Let $\mathcal{A}$ be a CTPA. A relation $\mathcal{R}$ on S is a* strong probabilistic simulation *if, for each pair of states $s, t \in S$ such that $s \mathcal{R} t$,*

- *$L(s) = L(t)$ and*
- *if $s \xrightarrow{a} r_s$ for some rate r_s, then there exists r_t such that $t \xrightarrow{a}_{\mathrm{C}} r_t$, μ_{r_s} $\mathcal{L}(\mathcal{R})$ μ_{r_t}, and $r_s(S) \leq r_t(S)$.*

We say that the CTPA $\mathcal{CP}_2$ strongly probabilistically simulates *$\mathcal{CP}_1$ if there exists a strong probabilistic simulation $\mathcal{R}$ on the disjoint union $S_1 \uplus S_2$ such that $\bar{s}_1 \mathcal{R} \bar{s}_2$ and we say that the state t strongly probabilistically simulates the state s if there exists a strong probabilistic simulation $\mathcal{R}$ such that $s \mathcal{R} t$. We denote the coarsest strong probabilistic simulation, called strong probabilistic similarity, by $\lesssim_p$.*

As for the strong case, the definition of strong probabilistic bisimulation and strong probabilistic bisimilarity, denoted by $\sim_p$, is obtained by requiring $\mathcal{R}$ to be a symmetric relation. Note that the two *PA*s in Figure 10 are actually strong probabilistic bisimilar, not just strongly probabilistic similar.

4.3 Weak Simulation and Bisimulation

Strong (probabilistic) simulations and bisimulations require that each transition proposed by the challenger is matched by the defender via a single (combined) transition. If we are not interested in internal computations, but just on the visible behavior, these relations are too restrictive. In order to abstract away internal steps, such relations have been relaxed to weak (probabilistic) simulations and bisimulations where the defender is able to match the challenging transition by performing several internal steps before and after having exhibited the same visible behavior, for instance, the same external action. The simplest example of weak transition is the one for labelled transition systems [39]: it is just the concatenation of arbitrarily many internal steps, the external transition (if we have to match an external challenging transition), and again arbitrarily many internal steps.

The definition of weak transition for probabilistic systems is not so easy as we have to take into account probabilistic choices. We first consider weak simulation and bisimulation for Markov chains and *sDTMC*s, and then for probabilistic automata. We are not aware of any definition of weak simulation and bisimulation for *CTPA*s where sequences of transitions are involved.

Markov Chains Before presenting the weak simulation and bisimulation for Markov chains, we need to introduce some additional definition [55, 57].

For a given pair of states (s_1, s_2) of the automaton $\mathcal{A}$ and functions $\gamma_i \colon S \to [0,1]$, we denote by U_i and V_i the sets $\{\, u \in \mathit{post}(s_i) \mid \gamma_i(u) > 0 \,\}$ and $\{\, v \in \mathit{post}(s_i) \mid \gamma_i(v) < 1 \,\}$, respectively. Essentially, U_i represents the states that can be reached with non-zero probability according to γ_i from s_i by performing one transition while V_i represents the states that cannot be reached with probability 1 according to γ_i from s_i by performing one transition. It is, however, worthwhile to mention that U_i and V_i are in general non-disjoint. The definition of weak simulation for *DTMC*s is not so immediate, because the "weak step" does not represent the fact that multiple transitions can be performed as in non-probabilistic settings like CCS and π-calculus [39,40] or in the other probabilistic models, as we will see later in the section, but that a single transition represents a *visible* or *stutter* step to a reached state z depending on whether z is in U or in V, respectively. More precisely we require for the visible steps (i.e., steps reaching states in U_i) that there exists a weighting function w for the conditional distributions $\frac{\mathbf{P}(s_1,\cdot)}{K_1}$ and $\frac{\mathbf{P}(s_2,\cdot)}{K_2}$ where K_i is essentially the probability to perform a visible step. The stutter steps (i.e., steps reaching states in V_i) must respect the weak bisimulations, that is, states in V_1 are weakly simulated by s_2 and s_1 is weakly simulated by all states in V_2, as depicted in Figure 11. Since a state t may belong to both U and V, the functions γ_i take care of distributing s_i over U_i and V_i. See [55, Section 4.3.1] for more details.

Definition 15 (Weak simulation for *DTMC*s [6, 9, 56, 57]). *Let $\mathcal{D}$ be a DTMC. A relation $\mathcal{R}$ on S is a* weak simulation *if, for each pair of states $s_1, s_2 \in S$ such that $s_1 \mathcal{R} s_2$,*

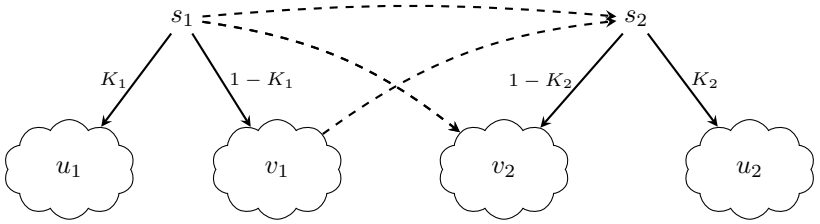


Figure 11: Splitting of successor states in weak simulations for *DTMC*s

- $L(s_1) = L(s_2)$ *and*
- *there exist functions* $\gamma_i \colon S \to [0,1]$ *for* $i \in \{1,2\}$ *such that*
  1. *(a)* $v_1 \mathcal{R}\, s_2$ *for each* $v_1 \in V_1$ *and (b)* $s_1 \mathcal{R}\, v_2$ *for each* $v_2 \in V_2$*;*
  2. *there exists a weighting function* $w \colon S \times S \to [0,1]$ *such that*
     (a) $w(u_1, u_2) > 0$ *implies* $u_1 \in U_1$*,* $u_2 \in U_2$*, and* $u_1 \mathcal{R}\, u_2$*,*
     (b) if $K_1 > 0$ *and* $K_2 > 0$*, then for all states* $t \in S$*,*

$$K_1 \cdot w(t, U_2) = \mathbf{P}(s_1, t) \cdot \gamma_1(t) \text{ and } K_2 \cdot w(U_1, t) = \mathbf{P}(s_2, t) \cdot \gamma_2(t)$$

     where $K_i = \sum_{u_i \in U_i} \mathbf{P}(s_i, u_i) \cdot \gamma_i(u_i)$ *for* $i \in \{1,2\}$*; and*
  3. *for* $u_1 \in U_1$ *there exist an execution fragment* $s_2 t_1 \dots t_n u_2$ *with positive probability such that* $n \in \mathbb{N}$*,* $s_1 \mathcal{R}\, t_j$ *for* $0 < j \leq n$*, and* $u_1 \mathcal{R}\, u_2$*.*

We say that the DTMC $\mathcal{D}_2$ weakly simulates $\mathcal{D}_1$ *if there exists a weak simulation* $\mathcal{R}$ *on the disjoint union* $S_1 \uplus S_2$ *such that* $\bar{s}_1 \mathcal{R}\, \bar{s}_2$ *and we say that the state* t *weakly simulates the state* s *if there exists a weak simulation* $\mathcal{R}$ *such that* $s \mathcal{R}\, t$*. We denote the coarsest weak simulation, called weak similarity, by* $\lessapprox$.

Figure 12 shows a *DTMC* for which $s_i \lessapprox t_j$ for $i, j \in \{1,2,3\}$. For each of these pairs we can select $U_1 = \emptyset$ and $V_2 = \emptyset$. Since $K_1 = 0$, we need to check only the Condition 1. However, since all the successor states of s_i are either empty or itself, this conditions holds trivially. It holds similarly that $v_1 \lessapprox v_2$.

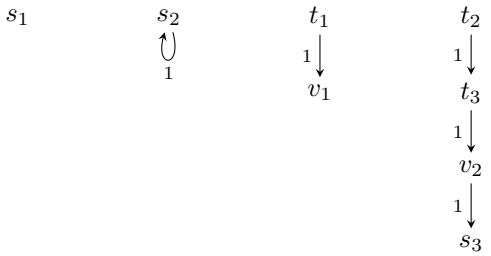


Figure 12: A *DTMC* with $L(v_1) = L(v_2) = \{v\}$ and $L(x) = \emptyset$ for each other state x.

The definition of weak simulation for *CTMC*s is similar, where condition (3) is replaced by $K_1 \cdot \mathbf{R}(s_1, S) \leq K_2 \cdot \mathbf{R}(s_2, S)$.

Similarly, the definition of weak simulation for *sDTMC*s is just a slight variation of the one for *DTMC*s, where we consider sub-distributions instead of distributions: for a given pair of states (s_1, s_2) of the *sDTMC* $\mathcal{S}$ and functions $\gamma_i \colon S_\perp \to [0,1]$, we change the definition of U_i and V_i as follows: U_i and V_i are the sets $\{\, u \in post_\perp(s_i) \mid \gamma_i(u) > 0 \,\}$ and $\{\, v \in post_\perp(s_i) \mid \gamma_i(v) < 1 \,\}$, respectively.

Definition 16 (Weak simulation for *sDTMC*s [6, 9, 56, 57]). *Let $\mathcal{S}$ be a sDTMC. A relation $\mathcal{R}$ on S is a* weak simulation *if, for each pair of states $s_1, s_2 \in S$ such that $s_1 \mathcal{R} s_2$,*

- *$L(s_1) = L(s_2)$ and*
- *there exist functions $\gamma_i \colon S_\perp \to [0,1]$ for $i \in \{1,2\}$ such that*
  1. *(a) $v_1 \mathcal{R} s_2$ for each $v_1 \in V_1 \setminus \{\perp\}$ and (b) $s_1 \mathcal{R} v_2$ for each $v_2 \in V_2 \setminus \{\perp\}$;*
  2. *there exists a function $w \colon S_\perp \times S_\perp \to [0,1]$ such that*
     (a) $w(u_1, u_2) > 0$ implies $u_1 \in U_1$, $u_2 \in U_2$, and $u_1 \mathcal{R}_\perp u_2$,
     (b) if $K_1 > 0$ and $K_2 > 0$, then for all states $t \in S$,

$$K_1 \cdot w(t, U_2) = \mathbf{P}(s_1, t) \cdot \gamma_1(t) \text{ and } K_2 \cdot w(U_1, t) = \mathbf{P}(s_2, t) \cdot \gamma_2(t)$$

     where $K_i = \sum_{u_i \in U_i} \mathbf{P}(s_i, u_i) \cdot \gamma_i(u_i)$ for $i \in \{1,2\}$; and
  3. *for $u_1 \in U_1 \setminus \{\perp\}$ there exist an execution fragment $s_2 t_1 \ldots t_n u_2$ with positive probability such that $n \in \mathbb{N}$, $s_1 \mathcal{R} t_j$ for $0 < j \leq n$, and $u_1 \mathcal{R} u_2$.*

We say that the sDTMC $\mathcal{S}_2$ weakly simulates *$\mathcal{S}_1$ if there exists a weak simulation $\mathcal{R}$ on the disjoint union $S_1 \uplus S_2$ such that $\bar{s}_1 \mathcal{R} \bar{s}_2$ and we say that the state t weakly simulates the state s if there exists a weak simulation $\mathcal{R}$ such that $s \mathcal{R} t$. We denote the coarsest weak simulation, called weak similarity, by $\lessapprox$.*

As for the strong bisimulation, the definition of weak bisimulation and weak bisimilarity, denoted by $\approx$, is obtained by requiring $\mathcal{R}$ to be a symmetric relation.

Remark 1. The definition of weak simulation for *sDTMC* we present here is neither sound nor complete for the liveness fragment of PCTL without the next operator [33]. To fix this problem, [33] proposes a new definition of weak simulation for *sDTMC* that is sound and conjectured to be complete. However, the associated technical report shows that completeness does not hold as well.

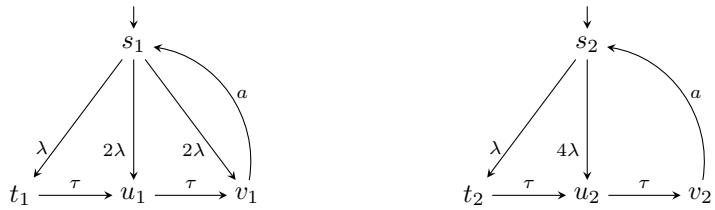
We have decided to maintain the definition from [55,57] instead of switching to the definition proposed in [33] because the latter currently lacks of a published decision algorithm while such algorithm is available for the former.

Interactive Markov Chains The definition of weak bisimulation for *IMC* is rather simple, since it is the obvious extension to the weak case of the strong bisimulation. Given an *IMC* $\mathcal{I}$, two state s and t, and an action a, we denote by $s \stackrel{a}{\Longrightarrow} t$ the sequence of transitions $s \stackrel{\tau}{\Longrightarrow} s' \stackrel{a}{\longrightarrow} t' \stackrel{\tau}{\Longrightarrow} t$ for some state s' and t' where $s \stackrel{\tau}{\Longrightarrow} s'$ is the reflexive and transitive closure of $\stackrel{\tau}{\longrightarrow}$, as defined for labelled transition systems [40]. For an *IMC* $\mathcal{I}$, we recall that $\gamma(v, \mathcal{C}) = \sum_{\{\, \lambda \in \mathbb{R}^{\geq 0} \mid v \stackrel{\lambda}{\longrightarrow} v', v' \in \mathcal{C} \,\}} \lambda$.

Definition 17 (Weak bisimulation for *IMC*s [28]). *Let $\mathcal{I}$ be a IMC. An equivalence relation $\mathcal{R}$ on S is a* weak bisimulation *if, for each pair of states $s, t \in S$ such that $s \mathcal{R} t$,*

- $L(s) = L(t)$,
- *if $s \stackrel{a}{\Longrightarrow} s'$ for some $s' \in S$ and $a \in \Sigma$, then there exists t' such that $t \stackrel{a}{\Longrightarrow} t'$ and $s' \mathcal{R} t'$, and*
- *if $s \stackrel{\tau}{\Longrightarrow} s'$ and s' does not enable a transition with label τ, then there exists t' such that t' does not enable a transition with label τ, $t \stackrel{\tau}{\Longrightarrow} t'$, and for each $\mathcal{C} \in S/\mathcal{R}$, $\gamma(s', \mathcal{C}^\tau) = \gamma(t', \mathcal{C}^\tau)$ where $\mathcal{C}^\tau = \{\, u \mid \exists v \in \mathcal{C}. u \stackrel{\tau}{\Longrightarrow} v \,\}$.*

We say that the IMC $\mathcal{A}_2$ weakly bisimulates *$\mathcal{A}_1$ if there exists a weak bisimulation $\mathcal{R}$ on the disjoint union $S_1 \uplus S_2$ such that $\bar{s}_1 \mathcal{R} \bar{s}_2$ and we say that the state t weakly bisimulates the state s if there exists a weak bisimulation $\mathcal{R}$ such that $s \mathcal{R} t$. We denote the coarsest weak bisimulation, called weak bisimilarity, by $\approx$.*



$\mathcal{I}_1$ $\mathcal{I}_2$

Figure 13: Two *IMC*s with $L_i(x) = \emptyset$ for each state x except for $L_i(t_i) = \{t\}$, $i = 1, 2$, such that $\mathcal{I}_1 \approx \mathcal{I}_2$

Figure 13 shows two *IMC*s that are weak bisimilar. This is justified by the equivalence relation whose classes are $\mathcal{C}_s = \{s_1, s_2\}$, $\mathcal{C}_t = \{t_1, t_2\}$, and $\mathcal{C}_o = \{u_1, u_2, v_1, v_2\}$. The first two conditions about labeling and interactive transitions are trivial for all pairs of related states; in particular, classes $\mathcal{C}_t$ and $\mathcal{C}_o$ (or $\mathcal{C}_s$) cannot be merged since labels are different: for instance, $L(t_1) = \{t\} \neq \emptyset = L(u_1)$, so the first condition would be violated. Consider the classes $\mathcal{C}_o$ and $\mathcal{C}_s$: they have the same labeling (each state in them has label $\emptyset$) but they cannot be merged since for instance the state $u_1 \in \mathcal{C}_o$ enables an a weak transition reaching s_1 that cannot be matched by the state $s_2 \in \mathcal{C}_s$, so the second condition would not be satisfied. The third condition about rates is obvious as well for pairs of states in the classes $\mathcal{C}_t$ and $\mathcal{C}_o$ since none of their states enables a timed transition, so $\gamma(x, \mathcal{C}^\tau)$ is 0 for each $x \in \mathcal{C}_t \cup \mathcal{C}_o$ and $\mathcal{C} \in \{\mathcal{C}_s, \mathcal{C}_t, \mathcal{C}_o\}$. The only non-trivial case is the pair (s_1, s_2) (the symmetric case is analogous). The only weak transitions with label τ enabled by s_1 and s_2 are $s_1 \stackrel{\tau}{\Longrightarrow} s_1$ and $s_2 \stackrel{\tau}{\Longrightarrow} s_2$, since neither s_1 nor s_2 enables a transition with label τ; $s_1 \mathcal{R} s_2$ trivially holds. $\gamma(s_1, \mathcal{C}_s^\tau) = 0 = \gamma(s_2, \mathcal{C}_s^\tau)$ since $\mathcal{C}_s^\tau = \mathcal{C}_s$ and there is no timed transition reaching

$\mathcal{C}_s$; $\gamma(s_1, \mathcal{C}_t^\tau) = \lambda = \gamma(s_2, \mathcal{C}_t^\tau)$ since $\mathcal{C}_t^\tau = \mathcal{C}_t$ and both s_1 and s_2 have a single timed transition with rate λ reaching $\mathcal{C}_t$; finally, $\gamma(s_1, \mathcal{C}_o^\tau) = 5\lambda = \gamma(s_2, \mathcal{C}_o^\tau)$ since $\mathcal{C}_o^\tau = \mathcal{C}_o \cup \mathcal{C}_t$.

Probabilistic Automata Before introducing the weak (combined) transition for probabilistic automata, we need some preliminary definition.

A *scheduler* for a *PA* $\mathcal{P}$ is a function $\sigma \colon \mathit{frags}^*(\mathcal{P}) \to \mathrm{SubDisc}(\to)$ such that for each $\alpha \in \mathit{frags}^*(\mathcal{P})$, $\sigma(\alpha) \in \mathrm{SubDisc}(\{\, tr \in \to \mid \mathit{src}(tr) = \mathit{last}(\alpha) \,\})$. Given a scheduler σ and a finite execution fragment α, the distribution $\sigma(\alpha)$ describes how transitions are chosen to move on from $\mathit{last}(\alpha)$. We say that a scheduler σ is a *Dirac scheduler* if for each $\alpha \in \mathit{frags}^*(\mathcal{P})$, $\sigma(\alpha)$ is a Dirac distribution and we say that σ is a *determinate scheduler* if for each $\alpha, \alpha' \in \mathit{frags}^*(\mathcal{P})$, if $\mathit{trace}(\alpha) = \mathit{trace}(\alpha')$ and $\mathit{last}(\alpha) = \mathit{last}(\alpha')$, then $\sigma(\alpha) = \sigma(\alpha')$. A scheduler σ and a state s induce a probability distribution $\mu_{\sigma,s}$ over execution fragments as follows. The basic measurable events are the cones of finite execution fragments, where the cone of α, denoted by C_α, is the set $\{\, \alpha' \in \mathit{frags}(\mathcal{P}) \mid \alpha \leqslant \alpha' \,\}$. The probability $\mu_{\sigma,s}$ of a cone C_α is defined recursively as follows:

$$\mu_{\sigma,s}(C_\alpha) = \begin{cases} 0 & \text{if } \alpha = t \text{ for a state } t \neq s, \\ 1 & \text{if } \alpha = s, \\ \mu_{\sigma,s}(C_{\alpha'}) \cdot \sum_{tr \in \xrightarrow{a}} \sigma(\alpha')(tr) \cdot \mu_{tr}(t) & \text{if } \alpha = \alpha' a t. \end{cases}$$

Standard measure theoretical arguments ensure that $\mu_{\sigma,s}$ extends uniquely to the σ-field generated by cones. We call the resulting measure $\mu_{\sigma,s}$ a *probabilistic execution fragment* of $\mathcal{P}$ and we say that it is generated by σ from s. Given a finite execution fragment α, we define $\mu_{\sigma,s}(\alpha)$ as $\mu_{\sigma,s}(\alpha) = \mu_{\sigma,s}(C_\alpha) \cdot \sigma(\alpha)(\bot)$, where $\sigma(\alpha)(\bot)$ is the probability of terminating the computation after α has occurred.

We say that there is a *weak combined transition* from $s \in S$ to $\mu \in \mathrm{Disc}(S)$ labelled by $a \in \Sigma$, denoted by $s \stackrel{a}{\Longrightarrow}_{\mathrm{C}} \mu$, if there exists a scheduler σ such that the following holds for the induced probabilistic execution fragment $\mu_{\sigma,s}$:

1. $\mu_{\sigma,s}(\mathit{frags}^*(\mathcal{P})) = 1$;
2. for each $\alpha \in \mathit{frags}^*(\mathcal{P})$, if $\mu_{\sigma,s}(\alpha) > 0$ then $\mathit{trace}(\alpha) = \mathit{trace}(a)$;
3. for each state t, $\mu_{\sigma,s}(\{\, \alpha \in \mathit{frags}^*(\mathcal{P}) \mid \mathit{last}(\alpha) = t \,\}) = \mu(t)$.

In this case, we say that the weak combined transition $s \stackrel{a}{\Longrightarrow}_{\mathrm{C}} \mu$ is induced by σ. When σ is a Dirac scheduler, then we say that it induces a weak transition from $s \in S$ to $\mu \in \mathrm{Disc}(S)$ labelled by $a \in \Sigma$, denoted by $s \stackrel{a}{\Longrightarrow} \mu$.

Albeit the definition of weak (combined) transitions is somewhat intricate, this definition is just the obvious extension of weak transitions on labelled transition systems to the setting with probabilities. See [48] for more details on weak combined transitions.

As an example of weak combined transition, consider the *PA* in Figure 14. We now show that there exists a scheduler inducing the weak combined transition

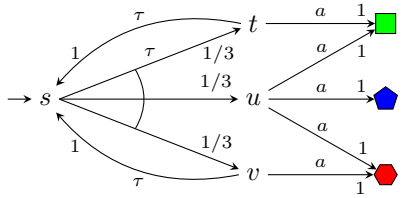


Figure 14: A probabilistic automaton

$s \stackrel{a}{\Longrightarrow}_{\mathrm{C}} \mu$ where $\mu = \{(\blacksquare, \frac{4}{18}), (\pentagon, \frac{7}{18}), (\hexagon, \frac{7}{18})\}$. Let μ_s be $\{(t, \frac{1}{3}), (u, \frac{1}{3}), (v, \frac{1}{3})\}$ and consider the scheduler σ defined as follows:

$$\sigma(\alpha) = \begin{cases} \delta_{s \xrightarrow{\tau} \mu_s} & \text{if } \mathit{last}(\alpha) = s, \\ \{(t \xrightarrow{\tau} \delta_s, \frac{1}{2}), (t \xrightarrow{a} \delta_{\blacksquare}, \frac{1}{2})\} & \text{if } \alpha = s\tau t, \\ \delta_{t \xrightarrow{a} \delta_{\blacksquare}} & \text{if } \alpha = s\tau t\tau s\tau t, \\ \delta_{u \xrightarrow{a} \delta_{\pentagon}} & \text{if } \mathit{last}(\alpha) = u, \\ \delta_{v \xrightarrow{a} \delta_{\hexagon}} & \text{if } \mathit{last}(\alpha) = v, \text{ and} \\ \delta_{\bot} & \text{otherwise.} \end{cases}$$

It is easy to show that indeed σ induces $s \stackrel{a}{\Longrightarrow}_{\mathrm{C}} \mu$. For instance, consider the state $\blacksquare$; in fact, $\mu_{\sigma,s}(\{\alpha \in \mathit{frags}^*(\mathcal{P}) \mid \mathit{last}(\alpha) = \blacksquare\}) = \mu_{\sigma,s}(\{s\tau ta\blacksquare, s\tau t\tau s\tau ta\blacksquare\}) + \mu_{\sigma,s}(\{\alpha \in \mathit{frags}^*(\mathcal{P}) \mid \mathit{last}(\alpha) = \blacksquare\} \setminus \{s\tau ta\blacksquare, s\tau t\tau s\tau ta\blacksquare\}) = \mu_{\sigma,s}(s\tau ta\blacksquare) + \mu_{\sigma,s}(s\tau t\tau s\tau ta\blacksquare) + 0 = 1 \cdot 1 \cdot \frac{1}{3} \cdot \frac{1}{2} \cdot 1 \cdot 1 + 1 \cdot 1 \cdot \frac{1}{3} \cdot \frac{1}{2} \cdot 1 \cdot 1 \cdot \frac{1}{3} \cdot 1 \cdot 1 \cdot 1 = \frac{4}{18} = \mu(\blacksquare)$.

Note that σ is neither Dirac nor determinate; moreover it is not the only scheduler inducing $s \stackrel{a}{\Longrightarrow}_{\mathrm{C}} \mu$: in fact, also the determinate scheduler σ' defined as follows induces $s \stackrel{a}{\Longrightarrow}_{\mathrm{C}} \mu$.

$$\sigma'(\alpha) = \begin{cases} \delta_{s \xrightarrow{\tau} \mu_s} & \text{if } \mathit{last}(\alpha) = s, \\ \{(t \xrightarrow{\tau} \delta_s, \frac{3}{7}), (t \xrightarrow{a} \delta_{\blacksquare}, \frac{4}{7})\} & \text{if } \mathit{last}(\alpha) = t, \\ \delta_{u \xrightarrow{a} \delta_{\pentagon}} & \text{if } \mathit{last}(\alpha) = u, \\ \delta_{v \xrightarrow{a} \delta_{\hexagon}} & \text{if } \mathit{last}(\alpha) = v, \text{ and} \\ \delta_{\bot} & \text{otherwise.} \end{cases}$$

Definition 18 (Weak (probabilistic) simulation on *PA*s [6, 9, 43, 51, 56, 57]). *Let $\mathcal{P}$ be a PA. A relation $\mathcal{R}$ on S is a* weak (probabilistic) simulation *if, for each pair of states $s, t \in S$ such that $s \mathcal{R} t$,*

- *$L(s) = L(t)$ and*
- *if $s \xrightarrow{a} \mu_s$ for some probability distribution μ_s, then there exists μ_t such that $t \stackrel{a}{\Longrightarrow} \mu_t$ ($t \stackrel{a}{\Longrightarrow}_{\mathrm{C}} \mu_t$) and $\mu_s \, \mathcal{L}(\mathcal{R}) \, \mu_t$.*

We say that the PA $\mathcal{P}_2$ weakly (probabilistically) simulates *$\mathcal{P}_1$ if there exists a weak (probabilistic) simulation $\mathcal{R}$ on the disjoint union $S_1 \uplus S_2$ such that $\bar{s}_1 \mathcal{R} \bar{s}_2$*

and we say that the state t weakly (probabilistically) simulates the state s if there exists a weak (probabilistic) simulation $\mathcal{R}$ such that $s\ \mathcal{R}\ t$. We denote the coarsest weak (probabilistic) simulation, called weak (probabilistic) similarity, by $\lessapprox$ ($\lessapprox_p$).

As usual, the weak (probabilistic) bisimulations [27,29,43,51], denoted by $\approx$ ($\approx_p$), are obtained by requiring $\mathcal{R}$ to be a symmetric relation.

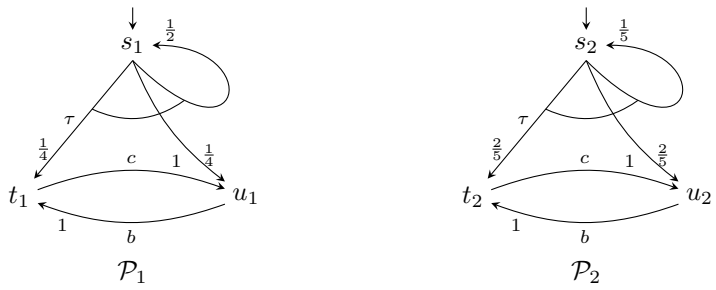


Figure 15: Two *PA*s with $L_i(x_i) = \{x\}$ for each state x_i, $i = 1, 2$ such that $\mathcal{P}_1 \approx_p \mathcal{P}_2$

Figure 15 shows two *PA*s that are weak probabilistic bisimilar. The relation justifying $\mathcal{P}_1 \approx_p \mathcal{P}_2$ is the equivalence relation whose classes are $\{s_1, s_2\}$, $\{t_1, t_2\}$, and $\{u_1, u_2\}$. Checking the pairs in $\{t_1, t_2\}$ and $\{u_1, u_2\}$ is trivial, so consider for instance the pair (s_1, s_2) and the transition $s_1 \xrightarrow{\tau} \mu_1$ where $\mu_1 = \{(s_1, \frac{1}{2}), (t_1, \frac{1}{4}), (u_1, \frac{1}{4})\}$. s_2 can match such transition via the weak combined transition $s_2 \stackrel{\tau}{\Longrightarrow}_{\mathrm{C}} \mu$ where $\mu = \{(s_2, \frac{1}{2}), (t_2, \frac{1}{4}), (u_2, \frac{1}{4})\}$ induced by the scheduler σ defined as

$$\sigma(\alpha) = \begin{cases} \{(s_2 \xrightarrow{\tau} \mu_2, \frac{5}{8}), (\bot, \frac{3}{8})\} & \text{if } \alpha = s_2 \text{ and} \\ \delta_\bot & \text{otherwise,} \end{cases}$$

where $\mu_2 = \{(s_2, \frac{1}{5}), (t_2, \frac{2}{5}), (u_2, \frac{2}{5})\}$. The transition $s_2 \xrightarrow{\tau} \mu_2$ can be matched by s_1 via the weak combined transition $s_1 \stackrel{\tau}{\Longrightarrow}_{\mathrm{C}} \mu'$ where μ' is the distribution $\{(s_1, \frac{1}{5}), (t_1, \frac{2}{5}), (u_1, \frac{2}{5})\}$, transition that is induced by the scheduler σ' defined as

$$\sigma'(\alpha) = \begin{cases} \delta_{s_1 \xrightarrow{\tau} \mu_1} & \text{if } \alpha = s_1 \text{ or } \alpha = s_1 \tau s_1, \\ \{(s_1 \xrightarrow{\tau} \mu_1, \frac{2}{5}), (\bot, \frac{3}{5})\} & \text{if } \alpha = s_1 \tau s_1 \tau s_1, \text{ and} \\ \delta_\bot & \text{otherwise.} \end{cases}$$

4.4 Markov Automata

Finally, we discuss simulations and bisimulations for Markov Automata. For the strong (probabilistic) simulations and bisimulations, they are just the merge of the corresponding definitions for *PA*s and *CTMC*s, where the timed transitions are considered only if the states do not enable internal transitions, as happens for *IMC*s.

For the weak (probabilistic) simulations and bisimulations, the approach is quite different from the previous definitions since they relate distributions instead of states. This makes the definition quite involved and out of the scope of this survey, also by considering that the exponential decision algorithms [20, 46] for these bisimulations just make use of the algorithm for *PA* weak combined transitions we will see in Section 6.2 as a black box.

We refer the interested reader to [20, 46] for the technical details and theoretical considerations that allow to define a state-based bisimulation that is equivalent to the distribution-based one as defined in [17, 22, 23].

5 Networks and Maximum Flow Problem

Given a set V, we say that (V, E) is a *directed graph* with vertices V and edges E if $E \subseteq V \times V$. A network $\mathcal{N}$ is a tuple $(V, E, \triangle, \blacktriangledown, c)$ where (V, E) is a finite directed graph, $\triangle$ and $\blacktriangledown$ are distinguished vertices called *source* and *sink*, respectively, and $c\colon E \to \mathbb{R}^{\geq 0} \cup \{\infty\}$ is a total function called edge *capacity function*. The capacity function, however, can be generalized to all pairs of vertexes by defining $c(u, v) = 0$ for each $(u, v) \notin E$.

Definition 19 (Flow [1, 12]). *A* flow f *on* $\mathcal{N}$ *is a function* $f\colon V \times V \to \mathbb{R}$ *such that:*

1. $f(u, v) \leq c(u, v)$ *for each* $(u, v) \in V \times V$ — capacity constraints
2. $f(u, v) = -f(v, u)$ *for each* $(u, v) \in V \times V$ — antisymmetry constraint
3. $f(V, v) = 0$ *for each* $v \in V \setminus \{\triangle, \blacktriangledown\}$ — conservation rule

The value of a flow function is computed as $f(\triangle, V)$, also denoted by $|f|$. A flow of maximum value is called a *maximum flow*.

5.1 Computing the Maximum Flow

A *preflow* [1] is a function $f\colon V \times V \to \mathbb{R}$ that satisfies the first two conditions above and the following relaxation of the last condition: $f(V, v) \geq 0$ for each $v \in V \setminus \{\triangle\}$.

For each vertex v, its *excess* $e(v)$ is defined by $f(V, v)$. A vertex $v \in V \setminus \{\triangle, \blacktriangledown\}$ is called *active* if $e(v) > 0$. It is easy to check that when no vertex $v \in V \setminus \{\triangle, \blacktriangledown\}$ is active, the preflow function is actually a flow function. A pair (u, v) is said to be a *residual edge* of f if $f(u, v) < c(u, v)$. We denote the set of residual edges with regard to f by E_f. Corresponding to each residual edge (u, v) we define the *residual capacity* $c_f(u, v)$ as $c(u, v) - f(u, v)$. We say that the edge (u, v) is *saturated* if it is not a residual edge. A *valid distance function* d (also known as *valid labeling* [25]) is a function $d\colon V \to \mathbb{N} \cup \{\infty\}$ such that $d(\triangle) = |V|$, $d(\blacktriangledown) = 0$, and $d(u) \leq d(v) + 1$ for each residual edge (u, v). A residual edge (u, v) is called *admissible* if $d(u) = d(v) + 1$.

The maximal flow can be computed by means of preflow as follows: the algorithm initializes the preflow f by defining $f(u, v) = 0$ for each $(u, v) \in$

$V \times V$ except for $f(\triangle, v) = c(\triangle, v)$ for each $v \in V$. The distance function d has initial values $d(\triangle) = |V|$ and $d(v) = 0$ for each other vertex $v \in V$. In order to maintain the validity of the preflow f and of the distance function d, the algorithm looks for active vertices in the network. If there exists an active vertex v and a residual edge (v, u) that is admissible, then, we *push* through (v, u) the amount of flow $\chi = \min\{e(v), c_f(v, u)\}$. This is done by increasing $f(v, u)$ (and decreasing $f(u, v)$) by χ and similarly, the excesses of v and u are updated by setting $e(v) = e(v) - \chi$ and $e(u) = e(u) + \chi$. If v is active but there is no admissible edge leaving it, the algorithm *relabels* v by defining $d(v) = \min\{\, d(u) + 1 \mid (v, u) \in E_f \,\}$. Pushing and relabeling are repeated until all vertexes are not active. It can be proved that the resulting ultimate flow f is a maximum flow [1, 25]. The generic preflow-push algorithm terminates after $\mathcal{O}(n^2 m)$ iterations where n and m are the number of nodes and the number of arcs of the network G, respectively.

5.2 Relation between Lifting and Maximum Flow

As we have seen in Section 2, the lifting $\mathcal{L}(\mathcal{R}) \subseteq \mathrm{Disc}(X) \times \mathrm{Disc}(X)$ of a relation $\mathcal{R} \subseteq X \times X$ has two different characterizations: via weighting functions and via $\mathcal{R}$-closure. It can be indeed characterized also via the maximum flow in a network as follows. First, we construct the network induced by the relation $\mathcal{R}$ and the two (sub-)probability distributions ρ_1 and ρ_2 and then we compute the maximum flow for such network. Given a set X, let $\overline{X}$ be the set $\overline{X} = \{\, \overline{x} \mid x \in X \,\}$.

Definition 20 ([4, 57]). *Let $\rho_1, \rho_2 \in \mathrm{Disc}(X)$ and $\mathcal{R} \subseteq X \times X$. The induced network $\mathcal{N}(\mathcal{R}, \rho_1, \rho_2) = (V, E, \triangle, \blacktriangledown, c)$ is defined by*

- $V = X \cup \overline{X} \cup \{\triangle, \blacktriangledown\}$,
- $E = \{\, (x, \overline{y}) \mid (x, y) \in \mathcal{R} \,\} \cup \{\, (\triangle, x) \mid x \in X \,\} \cup \{\, (\overline{y}, \blacktriangledown) \mid y \in X \,\}$, *and*
- $c(\triangle, x) = \rho_1(x)$, $c(\overline{y}, \blacktriangledown) = \rho_2(y)$, *and* $c(x, \overline{y}) = 1$ *for all* $x, y \in X$.

As shown in [55], $\rho_1 \; \mathcal{L}(\mathcal{R}) \; \rho_2$ if and only if the maximum flow of the induced network $\mathcal{N}(\mathcal{R}, \rho_1, \rho_2)$ is 1.

It is worthwhile to note that for each $x \notin \mathrm{Supp}(\rho_1)$, $c(\triangle, x) = 0$ (and similarly, for each $y \notin \mathrm{Supp}(\rho_2)$, $c(y, \blacktriangledown) = 0$), thus the flow along the edge $(\triangle, x)$ is always 0. Therefore the induced network can be equivalently simplified as follows: let $\rho_1, \rho_2 \in \mathrm{Disc}(X)$ and $\mathcal{R} \subseteq X \times X$. The induced network $\mathcal{N}(\mathcal{R}, \rho_1, \rho_2) = (V, E, \triangle, \blacktriangledown, c)$ is defined by

- $V = \mathrm{Supp}(\rho_1) \cup \overline{\mathrm{Supp}(\rho_2)} \cup \{\triangle, \blacktriangledown\}$,
- $E = \{\, (x, \overline{y}) \mid (x, y) \in \mathcal{R}, x \in \mathrm{Supp}(\rho_1), y \in \mathrm{Supp}(\rho_2) \,\} \cup \{\, (\triangle, x) \mid x \in \mathrm{Supp}(\rho_1) \,\} \cup \{\, (\overline{y}, \blacktriangledown) \mid y \in \mathrm{Supp}(\rho_2) \,\}$, and
- $c(\triangle, x) = \rho_1(x)$, $c(\overline{y}, \blacktriangledown) = \rho_2(y)$, and $c(x, \overline{y}) = 1$ for all $x, y \in X$.

In the remaining of the paper we use the more appropriate definition of induced network without further mentioning which one we are considering.

SIM($\preccurlyeq, \mathcal{A}$)

1: $i \leftarrow 0$; $\mathcal{R}_i \leftarrow \{ (s_1, s_2) \in S \times S \mid L(s_1) = L(s_2) \}$;
2: **repeat**
3: $\quad \mathcal{R}_{i+1} \leftarrow \mathcal{R}_i$;
4: $\quad$ **for all** $s_1 \xrightarrow{a} \rho_1 \in s_1 \longrightarrow$ **do**
5: $\quad\quad$ **for all** $s_2 \in S$ such that $s_1 \mathcal{R}_i s_2$ **do**
6: $\quad\quad\quad$ **if** there does not exist $s_2 \xrightarrow{a}_{\preccurlyeq} \rho_2$ satisfying the step condition
7: $\quad\quad\quad\quad \mathcal{R}_{i+1} \leftarrow \mathcal{R}_{i+1} \setminus \{(s_1, s_2)\}$;
8: $\quad i \leftarrow i + 1$;
9: **until** $\mathcal{R}_i = \mathcal{R}_{i-1}$;
10: **return** $\mathcal{R}_i$;

Figure 16: Algorithm for computing simulation

6 The Algorithms

We now consider the algorithms and their complexity that are used to decide the simulations and the bisimulations introduced in Section 4. In the following, we denote by $s \xrightarrow{a}_{\preccurlyeq} \rho$ the matching transition involved in the step condition of the relation $\preccurlyeq$. For instance, when $\preccurlyeq$ is $\lesssim$, then $s \xrightarrow{a}_{\preccurlyeq} \rho$ stands for $s \xrightarrow{a} \rho$ while when $\preccurlyeq$ is $\lesssim_p$, then $s \xrightarrow{a}_{\preccurlyeq} \rho$ stands for $s \stackrel{a}{\Longrightarrow}_{\mathrm{C}} \rho$.

6.1 The General Algorithms for Simulations and Bisimulations

Simulation algorithm Figure 16 depicts the algorithm SIM, proposed for instance in [4], that computes the simulation $\preccurlyeq$ for the automaton $\mathcal{A}$. The procedure begins with the initial relation $\mathcal{R}_0 = \{ (s_1, s_2) \in S \times S \mid L(s_1) = L(s_2) \}$ which is coarser than $\preccurlyeq$. In each iteration i of the main loop, the relation $\mathcal{R}_i$, initialized with $\mathcal{R}_{i-1}$, is refined by deleting each pair (s_1, s_2) such that s_2 is not able to exhibit the transition $s_2 \xrightarrow{a}_{\preccurlyeq} \rho_2$ such that $\rho_1 \; \mathcal{L}(\mathcal{R}_{i-1}) \; \rho_2$ required by the step condition of $\preccurlyeq$. The main loop terminates when all pairs of $\mathcal{R}_i$ satisfy the step condition, that is, when $\mathcal{R}_i = \mathcal{R}_{i-1}$. The resulting similarity $\preccurlyeq$ is then $\mathcal{R}_i$.

It is immediate to see that the core of this algorithm is the check for the existence of the step condition for s_1 and s_2, and that this is also the main source of the complexity of the algorithm. In fact, if we denote by N the size of the automaton, i.e., $N = \max\{|S|, |\rightarrow|\}$, it is easy to derive that the complexity of the algorithm is $\mathcal{O}(N^4 \cdot C)$ where C is the complexity of deciding the existence of the matching transition $s_2 \xrightarrow{a}_{\preccurlyeq} \rho_2$.

Bisimulation algorithm In order to compute the bisimulation $\asymp$ for the automaton $\mathcal{A}$ we can follow the standard partition refinement approach [13, 20, 29, 35, 41, 43] depicted in Figure 17: the procedure BISIM takes as parameter the bisimulation $\asymp$ and the automaton $\mathcal{A}$ and iteratively constructs the set $S/\asymp$, the set of equivalence classes of states S under $\asymp$, starting with the partitioning $\mathcal{W} = \{ (s_1, s_2) \in S \times S \mid L(s_1) = L(s_2) \}$ and refining it until $\mathcal{W}$ satisfies the

BISIM($\asymp, \mathcal{A}$)

1: $\mathcal{W} \leftarrow \{ (s_1, s_2) \in S \times S \mid L(s_1) = L(s_2) \}$;
2: **repeat**
3: $\quad \mathcal{W}' \leftarrow \mathcal{W}$;
4: $\quad (\mathcal{C}, a, \rho) \leftarrow$ FINDSPLIT($\mathcal{A}, \mathcal{W}$);
5: $\quad \mathcal{W} \leftarrow$ REFINE($\mathcal{C}, a, \rho$);
6: **until** $\mathcal{W} \leftarrow \mathcal{W}'$
7: **return** $\mathcal{W}$

FINDSPLIT($\mathcal{A}, \mathcal{W}$)

1: **for all** $(s_1, a, \rho_1) \in \rightarrow$ **do**
2: $\quad$ **for all** $s_2 \in [s_1]_{\mathcal{W}}$ **do**
3: $\quad\quad$ **if** there does not exist $s_2 \xrightarrow{a}_{\asymp} \rho_2$ satisfying the step condition
4: $\quad\quad\quad$ **return** $([s_1]_{\mathcal{W}}, a, \rho_1)$
5: **return** $(\emptyset, \tau, \delta_\perp)$

Figure 17: Algorithm for computing bisimulation

definition of $\asymp$ and thus the resulting partitioning is the coarsest one, i.e., the algorithm computes $\asymp$. In the refinement phase, the algorithm checks for each partition whether all pairs respect the step condition; if a pair (s_1, s_2) fails, then such partition is split in two partitions: one containing s_1 and all states satisfying the step condition with respect to transitions from s, the other containing the remaining states, including s_2. On termination, the partitioning $\mathcal{W}$ is the bisimilarity $\asymp$.

As happens for SIM, it is immediate to see that the core of this algorithm is the check whether the step condition for s_1 and s_2 holds, and that this is also the main source of the complexity of the algorithm. In fact, as done for the simulation procedure, if we denote by N the size of the automaton, i.e., $N = \max\{|S|, |\rightarrow|\}$, it is easy to derive that the complexity of the algorithm is $\mathcal{O}(N^3 \cdot C)$ where C is the complexity of deciding the existence of the matching transition $s_2 \xrightarrow{a}_{\asymp} \rho_2$.

6.2 The Specialized Algorithms

Strong Simulation We now present an improved algorithm [57] for the strong simulation on *sDTMC*s, *DTMC*s, and *CTMC*s, respectively, based on the properties of the network flow setting.

Let $\mathcal{A}$ be either a *sDTMC* or a *DTMC*. Since for *sDTMC*s and *DTMC*s every state s enables a single transition, checking the existence of the matching transition $s_2 \xrightarrow{\tau}_{\lesssim} \mu_2$ reduces to check whether $\mu_1 \; \mathcal{L}(\mathcal{R}_i) \; \mu_2$ where $\mu_1 = \mathbf{P}(s_1, \cdot)$ and $\mu_2 = \mathbf{P}(s_2, \cdot)$. As we have seen in Section 5.2, $\mu_1 \; \mathcal{L}(\mathcal{R}_i) \; \mu_2$ is equivalent to check whether the induced network $\mathcal{N}(\mathcal{R}_i, \mu_1, \mu_2)$ has 1 as maximum flow. Since finding the maximum flow has complexity $\mathcal{O}(\frac{N^3}{\log N})$, thus the resulting complexity of SIM($\lesssim, \mathcal{A}$) is $\mathcal{O}(\frac{N^7}{\log N})$ [4, 57].

Improved algorithm for sDTMCs Consider the $\textsc{Sim}(\lesssim, \mathcal{A})$ procedure and a pair of states $(s_1, s_2) \in \mathcal{R}_1$. Let $\mu_1 = \mathbf{P}(s_1, \cdot)$ and $\mu_2 = \mathbf{P}(s_2, \cdot)$ and suppose that (s_1, s_2) belongs to $\mathcal{R}_1, \ldots, \mathcal{R}_k$ during the whole of the iterations $i = 1, \ldots, k$ until the pair either violates the step condition with respect to $\mathcal{R}_k$ or the algorithm terminates after iteration k. This means that the maximum flow algorithm is used k times for this pair. As a matter of fact, the induced networks $\mathcal{N}(\mathcal{R}_i, \mu_1, \mu_2)$ built in successive iterations are very similar, and may often be the same across iterations. From iteration to iteration, in fact, they differ only for the removal of some edge $(t_1, \overline{t_2})$ induced by $\mathcal{R}_i \leftarrow \mathcal{R}_i \setminus \{(t_1, t_2)\}$ but this does not change the network when $t_1 \notin \text{Supp}(\mu_1)$ or $t_2 \notin \text{Supp}(\mu_2)$. This observation, inspired by [24], is the key point of [57] for improving the basic algorithm. In fact, the authors reuse the previous computed maximum flow in the sense that whatever happens to the network is good: if the network $\mathcal{N}(\mathcal{R}_i, \mu_1, \mu_2)$ is equal to $\mathcal{N}(\mathcal{R}_{i-1}, \mu_1, \mu_2)$, then the maximum flow is the same as the one in the previous iteration. On the other hand, if the two networks are different, then the preflow algorithm can be adapted to compute the new maximum flow using the previous maximum flow and distance function as a starting point.

$\textsc{Smf}_{init}(i, \mathcal{R}_i, \mu_1, \mu_2)$

1: Initialize the network $\mathcal{N}(\mathcal{R}_i, \mu_1, \mu_2)$;
2: Apply the preflow algorithm to compute the maximum flow for $\mathcal{N}(\mathcal{R}_i, \mu_1, \mu_2)$;
3: **return** $(|f_i| = 1, \mathcal{N}(\mathcal{R}_i, \mu_1, \mu_2), f_i, d_i)$;

$\textsc{Smf}(i, \mathcal{N}(\mathcal{R}_{i-1}, \mu_1, \mu_2), f_{i-1}, d_{i-1}, D_{i-1})$

1: $\mathcal{N}(\mathcal{R}_i, \mu_1, \mu_2) \leftarrow \mathcal{N}(\mathcal{R}_{i-1} \setminus D_{i-1}, \mu_1, \mu_2)$; $f_i \leftarrow f_{i-1}$; $d_i \leftarrow d_{i-1}$;
2: **for all** $(v_1, v_2) \in D_{i-1}$ **do**
3: $\quad f_i(\overline{v_2}, \blacktriangledown) \leftarrow f_i(\overline{v_2}, \blacktriangledown) - f_i(v_1, \overline{v_2})$;
4: $\quad f_i(v_1, v_2) \leftarrow 0$;
5: Apply the preflow algorithm initialized with f_i and d_i to compute the maximum flow for $\mathcal{N}(\mathcal{R}_i, \mu_1, \mu_2)$;
6: **return** $(|f_i| = 1, \mathcal{N}(\mathcal{R}_i, \mu_1, \mu_2), f_i, d_i)$;

Figure 18: Algorithm for computing a sequence of maximum flows

To explain this approach in more detail, consider the network $\mathcal{N}(\mathcal{R}_1, \mu_1, \mu_2)$ and let $D_1, \ldots, D_k$ be pairwise disjoint subsets of $\mathcal{R}_1$ that correspond to the pairs deleted from $\mathcal{R}_1$ in iteration i, i.e., $\mathcal{R}_{i+1} = \mathcal{R}_i \setminus D_i$ for $1 \leq i \leq k$. Let $f_i^{(s_1,s_2)}$ be the flow and $d_i^{(s_1,s_2)}$ the distance function of the network $\mathcal{N}(\mathcal{R}_i, \mu_1, \mu_2)$ where $0 \leq i \leq k$, respectively. The algorithm for updating the sequences of maximum flows and distances of the network $\mathcal{N}(\mathcal{R}_i, \mu_1, \mu_2)$ where $1 \leq i \leq k$ is depicted in Figure 18 and it works as follows: starting from the network $\mathcal{N}(\mathcal{R}_{i-1} \setminus D_{i-1}, \mu_1, \mu_2)$ with flow f_{i-1} and distance d_{i-1}, it computes for each pair $(v_1, v_2) \in D_{i-1}$ the flow $f_i(\overline{v_2}, \blacktriangledown)$ by decreasing the previous value $f_{i-1}(\overline{v_2}, \blacktriangledown)$ by the value of the flow $f_{i-1}(v_1, \overline{v_2})$ and then forces $f_i(v_1, v_2)$ to be 0. Then it calls

the preflow algorithm initialized with the updated flow and distance function to compute the maximum flow for the new network and returns the network, the updated flow and distance functions, and a boolean representing whether the maximum flow is 1.

The $\textsc{Smf}(i, \mathcal{N}(\mathcal{R}_{i-1}, \mu_1, \mu_2), f_{i-1}, d_{i-1}, D_{i-1})$ algorithm is the building block for the improved algorithm $\textsc{Sim}_{sDTMC}(\lesssim, \mathcal{S})$ that computes the strong simulation on *sDTMC*s, depicted in Figure 19.

$\textsc{Sim}_{sDTMC}(\lesssim, sDTMC)$

1: $\mathcal{R}_1 \leftarrow \{ (s_1, s_2) \in S \times S \mid L(s_1) = L(s_2) \}$; $i \leftarrow 1$;
2: $\mathcal{R}_2 \leftarrow \emptyset$;
3: **for all** $(s_1, s_2) \in \mathcal{R}_1$ **do**
4: $\quad Listener(s_1, s_2) \leftarrow \{ (u_1, u_2) \mid u_1 \in pre(s_1), u_2 \in pre(s_2), L(u_1) = L(u_2) \}$;
5: $\quad (match, \mathcal{N}(\mathcal{R}_1, \mu_1, \mu_2), f_1^{(s_1,s_2)}, d_1^{(s_1,s_2)}) \leftarrow \textsc{Smf}_{init}(1, \mathcal{R}_1, \mu_1, \mu_2)$
6: $\quad$ **if** *match*
7: $\quad\quad \mathcal{R}_2 \leftarrow \mathcal{R}_2 \cup \{(s_1, s_2)\}$;
8: **while** $\mathcal{R}_{i+1} \neq \mathcal{R}_i$ **do**
9: $\quad i \leftarrow i + 1$; $\mathcal{R}_{i+1} \leftarrow \emptyset$; $D_{i-1} \leftarrow \mathcal{R}_{i-1} \setminus \mathcal{R}_i$;
10: $\quad$ **for all** $(s_1, s_2) \in \mathcal{R}_i$ **do**
11: $\quad\quad D_{i-1}^{(s_1,s_2)} \leftarrow \emptyset$;
12: $\quad$ **for all** $(s_1, s_2) \in D_{i-1}$, $(u_1, u_2) \in Listener(s_1, s_2) \cap \mathcal{R}_{i-1}$ **do**
13: $\quad\quad D_{i-1}^{(u_1,u_2)} \leftarrow D_{i-1}^{(u_1,u_2)} \cup \{(s_1, s_2)\}$;
14: $\quad$ **for all** $(s_1, s_2) \in \mathcal{R}_i$ **do**
15: $\quad\quad (match, \mathcal{N}(\mathcal{R}_i, \mu_1, \mu_2), f_i^{(s_1,s_2)}, d_i^{(s_1,s_2)})$
$\quad\quad\quad\quad \leftarrow \textsc{Smf}(i, \mathcal{N}(\mathcal{R}_{i-1}, \mu_1, \mu_2), f_{i-1}^{(s_1,s_2)}, d_{i-1}^{(s_1,s_2)}, D_{i-1}^{(s_1,s_2)})$;
16: $\quad\quad$ **if** *match*
17: $\quad\quad\quad \mathcal{R}_{i+1} \leftarrow \mathcal{R}_{i+1} \cup \{(s_1, s_2)\}$;
18: **return** $\mathcal{R}_i$;

where $\mu_1 = \mathbf{P}(s_1, \cdot)$ and $\mu_2 = \mathbf{P}(s_2, \cdot)$

Figure 19: Improved algorithm for deciding strong simulation for *sDTMC*s.

The first part of the algorithm, from line 1 to 7 is essentially the same as in $\textsc{Sim}(\lesssim, sDTMC)$, except for line 4 where the set

$$Listener(s_1, s_2) = \{ (u_1, u_2) \mid u_1 \in pre(s_1), u_2 \in pre(s_2), L(u_1) = L(u_2) \}$$

is computed for the remainder of the procedure. In particular, this set contains all pairs (u_1, u_2) such that $(s_1, \overline{s_2})$ is an edge of $\mathcal{N}(\mathcal{R}_0, \mathbf{P}(u_1, \cdot), \mathbf{P}(u_2, \cdot))$.

In each iteration i of the loop at lines 8–17, the procedure generates $\mathcal{R}_{i+1}$ from $\mathcal{R}_i$ by performing several steps: first, with the loop at line 12, it collects in $D_{i-1}^{(u_1,u_2)}$ the edges that should be removed from $\mathcal{N}(\mathcal{R}_{i-1}, \mathbf{P}(u_1, \cdot), \mathbf{P}(u_2, \cdot))$. Then, at line 15, the algorithm Smf builds the maximum flow by using information from the previous iteration $i - 1$. Basically, $\mathcal{N}(\mathcal{R}_{i-1}, \mu_1, \mu_2)$, $f_{i-1}^{(s_1,s_2)}$, and

$d_{i-1}^{(s_1,s_2)}$ are updated according to the set $D_{i-1}^{(s_1,s_2)}$; this generates the new maximum flow $f_i^{(s_1,s_2)}$ for the network $\mathcal{N}(\mathcal{R}_i, \mu_1, \mu_2)$ and if such flow is 1 (i.e., *match* is true), then (s_1, s_2) is added to $\mathcal{R}_{i+1}$ and survives this iteration. Eventually the while loop terminates and the last candidate simulation $\mathcal{R}_i$ is actually a strong bisimilarity.

The correctness and time complexity of this algorithm is stated in [57, Theorem 4.5 and 4.6], respectively. In particular, the time complexity is $\mathcal{O}(m^2 \cdot N + N^2)$, where $m = \sum_{s \in S} |post(s)|$, that is significantly smaller than $\mathcal{O}(\frac{N^7}{\log N})$ of the general algorithm $\textsc{Sim}(\lesssim, sDTMC)$.

Strong Simulation for DTMCs and CTMCs We now take into account *DTMC*s and *CTMC*s. Since every *DTMC* $\mathcal{D}$ is a *sDTMC*, we can use directly the algorithm $\textsc{Sim}_{sDTMC}(\lesssim, \mathcal{D})$. For *CTMC*s, we have to take care of the rate condition; this is easily obtained by replacing the assignment to $\mathcal{R}_1$ at the line 1 of the algorithm with

$$\mathcal{R}_1 \leftarrow \{\, (s_1, s_2) \in S \times S \mid L(s_1) = L(s_2), \mathbf{R}(s_1, S) \leq \mathbf{R}(s_2, S) \,\}$$

It is immediate to see that this change does not increase the complexity of the algorithm; in particular, the time complexity may be reduced, since there may be fewer pairs satisfying the rate condition as well.

Strong Probabilistic Simulation and Bisimulation Strong probabilistic simulation and bisimulation are defined only for *PA*s and *CTPA*s since they are the only models that exhibit internal nondeterminism, thus they allow to combine transitions with the same label (and the same rate, for *CTPA*s). Checking the step condition thus requires to find such combined transition. One possibility is to check, for every possible combined transition, whether it satisfies the step condition; however this approach is not practical since given two transitions, there are uncountable many different convex combinations of them. The other possibility is to check whether there exists a choice for the coefficients of the convex combination by solving a linear programming problem encoding convex combination and lifting [57].

For a *PA* $\mathcal{P}$, the *LP* problem relative to relation $\mathcal{R}$, transition $s_1 \xrightarrow{a} \mu$ and state s_2 is:

$$\begin{array}{ll}
\sum_{i=1}^{k} c_i = 1 & \\
0 \leq c_i \leq 1 & \text{for } 1 \leq i \leq k \\
0 \leq f_{u,v} \leq 1 & \text{for each } (u, v) \in \mathcal{R}_\perp \\
\mu(s) = \sum_{t \in \mathcal{R}_\perp(s)} f_{s,t} & \text{for each } s \in S_\perp \\
\sum_{s \in \mathcal{R}_\perp^{-1}(t)} f_{s,t} = \sum_{i=1}^{k} c_i \rho_i(t) & \text{for each } t \in S_\perp
\end{array}$$

where $\{\rho_1, \ldots, \rho_k\} = \{\, \rho \mid (s_2, a, \rho) \in \rightarrow \,\}$.

For a *CTPA* $\mathcal{CP}$, the *LP* problem relative to relation $\mathcal{R}$, transition $s_1 \xrightarrow{a} r$ and state s_2 is similar:

$$
\begin{array}{ll}
\sum_{i=1}^{k} c_i = 1 & \\
0 \leq c_i \leq 1 & \text{for } 1 \leq i \leq k \\
0 \leq f_{u,v} \leq 1 & \text{for each } (u,v) \in \mathcal{R}_{\perp} \\
r(s) = r(S) \cdot \sum_{t \in \mathcal{R}_{\perp}(s)} f_{s,t} & \text{for each } s \in S_{\perp} \\
E \cdot \sum_{s \in \mathcal{R}_{\perp}^{-1}(t)} f_{s,t} = \sum_{i=1}^{k} c_i r_i(t) & \text{for each } t \in S_{\perp}
\end{array}
$$

for some $E \in \{\, r'(S) \mid (s_2, a, r') \in \mathbf{R} \,\}$, $E \geq r(S)$ where $\{r_1, \ldots, r_k\} = \{\, r' \mid (s_2, a, r') \in \mathbf{R}, r'(S) = E \,\}$.

The complexity of the $\textsc{Sim}(\lesssim_p, \mathcal{A})$ and $\textsc{Bisim}(\sim_p, \mathcal{A})$ algorithms is then polynomial and directly depends on the polynomial complexity [53] of solving the above *LP* problems, each one with at most $\mathcal{O}(N^2)$ constraints.

It is worthwhile to note that by combining a preflow approach, as the one adopted for SMF, and abstract interpretation techniques, the complexity can be reduced to $\mathcal{O}(N^3)$ for simulation and $\mathcal{O}(N^2 \cdot \log N)$ for bisimulation [15].

Weak Simulation and Bisimulation for *DTMC*s and *CTMC*s Now, we focus our attention to weak simulations. As it was the case for strong simulations, the core of the algorithm is to check the step condition with respect to the current relation $\mathcal{R}$. Based on the definition of weak simulation, for fixed characteristic functions γ_i for $i = 1, 2$, maximum flow algorithms can be used in order to check condition (2) in definition 15. In order to improve this check, we can make use of the parametric maximum flow algorithm in order to determine whether functions γ_i exist, with the aid of *breakpoints*, as we will see in the following.

As shown in [57], checking the step condition of the weak simulation for the pair of states (s_1, s_2) for both *DTMC*s and *CTMC*s is equivalent to finding the parameter ψ that makes a parametric network valid. In particular, the considered parametric network is $\mathcal{N}_{\psi}(\mathcal{R}, \mu_1, \mu_2)$ that is defined as $\mathcal{N}(\mathcal{R}, \mu_1, \psi \cdot \mu_2)$; this means that $\mathcal{N}(\mathcal{R}, \mu_1, \psi \cdot \mu_2)$ is the network $\mathcal{N}(\mathcal{R}, \mu_1, \mu_2)$ where the capacities for the edges leading to the sink are $c(\overline{t}, \blacktriangledown) = \psi \cdot \mu_2(t)$. The network $\mathcal{N}_{\psi}(\mathcal{R}, \mu_1, \mu_2)$ is valid if there exists a flow f that saturates all edges $(\triangle, u_1)$ and $(\overline{u_2}, \blacktriangledown)$ where u_1 belongs to the set $MU_1 = post(s_1) \setminus PV_1$ with $PV_1 = post(s_1) \cap \mathcal{R}^{-1}(s_2)$ and u_2 belongs to the set $MU_2 = post(s_2) \setminus PV_2$ with $PV_2 = post(s_2) \cap \mathcal{R}(s_1)$.

Sets MU_i and PV_i are strictly related to the sets U_i and V_i used for the strong simulation algorithm. Indeed, MU_i stands for "must be in U_i" while PV_i stands for "potentially in V_i". Functions γ_i are extended as expected by $\gamma_i(u) = 1$ for $u \in MU_i$, $i \in \{1, 2\}$.

If we fix $\psi \in \mathbb{R}^{\geq 0}$, then checking whether $\mathcal{N}_{\psi}(\mathcal{R}, \mu_1, \mu_2)$ is valid reduces to verify the feasibility of a flow problem (f has to saturate edges to MU_1 and from MU_2); this can be done by applying a simple transformation to the graph (in time $\mathcal{O}(|MU_1| + |MU_2|)$), solving the maximum flow problem for the transformed graph, and checking whether the flow saturates all edges from the new source [1]. So now the problem is to find a good ψ that makes $\mathcal{N}_{\psi}(\mathcal{R}, \mu_1, \mu_2)$

StepCondition$_{\lessapprox}(\mathcal{D}, \mathcal{R}, s_1, s_2)$

1: **if** $post(s_1) \subseteq \mathcal{R}^{-1}(s_2)$
2: **return true**
3: **if** $post(s_2) \subseteq \mathcal{R}(s_1)$
4: $U_1 \leftarrow \{\, s_1' \in post(s_1) \mid s_1' \notin \mathcal{R}^{-1}(s_2) \,\}$
5: **return** $\forall u_1 \in U_1.\exists s \in post(reach(s_2)) \cap \mathcal{R}(s_1).s \in \mathcal{R}(u_1)$
6: Compute all of the breakpoints $\psi_1 < \psi_2 < \cdots < \psi_j$ of $\mathcal{N}_\psi(\mathcal{R}, \mu_1, \mu_2)$
7: **return** $\exists i \in \{1, \ldots, j\}.\psi_i$ is valid for $\mathcal{N}_{\psi_j}(\mathcal{R}, \mu_1, \mu_2)$

where $\mu_1 = \mathbf{P}(s_1, \cdot)$ and $\mu_2 = \mathbf{P}(s_2, \cdot)$

Figure 20: Algorithm to check whether s_2 weakly simulates s_1 with respect to $\mathcal{R}$

valid, but there are uncountably many of such ψ we may check for. However, the candidates that really matter are finite, not uncountably many, and are called *breakpoints*. In particular, breakpoints can be identified by solving one more parametric maximum flow problem: Let $\kappa(\psi)$ be the *minimum cut capacity function* for the parameter ψ, that is, the capacity of a minimum cut of $\mathcal{N}_\psi(\mathcal{R}, \mu_1, \mu_2)$ as a function of ψ. Based on the Max flow Min cut theorem [1], the capacity of a minimum cut equals the value of a maximum flow. On the other hand, if the edge capacities in the network are linear functions of ψ, $\kappa(\psi)$ is a piecewise linear concave function with at most $|V| - 2$ breakpoints [24]. In particular, $|V| - 1$ or fewer line segments of the graph of $\kappa(\psi)$ are equivalent to $|V| - 1$ or fewer distinct minimal cuts. For some ψ^*, the capacity of a minimum cut gives an equation that leads to a line segment to the function $\kappa(\psi)$ at $\psi = \psi^*$. Furthermore, this line segment attaches the two points $(\psi_1, \kappa(\psi_1))$ and $(\psi_2, \kappa(\psi_2))$, where ψ_1, ψ_2 are the nearest breakpoints to the left and right, respectively. Therefore, as it would be expected, it is enough to examine only the breakpoints of $\mathcal{N}_\psi(\mathcal{R}, \mu_1, \mu_2)$: there exists a valid ψ for $\mathcal{N}_\psi(\mathcal{R}, \mu_1, \mu_2)$ if and only if one of the breakpoints of $\mathcal{N}_\psi(\mathcal{R}, \mu_1, \mu_2)$ is valid.

For a fixed breakpoint, it is adequate to solve one feasible flow problem to check if it is valid. In the network $\mathcal{N}_\psi(\mathcal{R}, \mu_1, \mu_2)$ the capacities of the edges going to the sink are increasing functions of the real-valued parameter ψ. If $\mathcal{N}_\psi(\mathcal{R}, \mu_1, \mu_2)$ is reversed, a parametric network that satisfies the conditions in [24] can be derived: the capacities emanating from $\triangle$ are non-decreasing functions of ψ. Therefore, the *breakpoint algorithm* [24] can be applied to compute the breakpoints of $\mathcal{N}_\psi(\mathcal{R}, \mu_1, \mu_2)$.

The Algorithm for DTMCs We are now able to provide the decision algorithm for the *DTMC* weak simulation: we just consider the Sim$(\lessapprox, \mathcal{D})$ where the step condition is verified by invoking the algorithm in Figure 20 that, given two states s_1 and s_2, it actually computes whether $s_1 \lessapprox s_2$.

By using this approach, the resulting complexity of the algorithm that computes the weak simulation for *DTMC*s is $\mathcal{O}(N^5)$. This complexity can be improved in practice by exploiting the network $\mathcal{N}_\psi(\mathcal{R}, \mu_1, \mu_2)$ whenever it can be

parted into sub-networks. We refer the reader interested in this approach to [57, Section 5.1.4]

An Algorithm for CTMCs The algorithm for computing weak simulation on *CTMC*s is very close to the one for *DTMC*s since the only difference is the last requirement of the step condition: "$K_1 \cdot \mathbf{R}(s_1, S) \leq K_2 \cdot \mathbf{R}(s_2, S)$" instead of "for $u_1 \in U_1$ there exist an execution fragment $s_2 t_1 \dots t_n u_2$ with positive probability such that $n \in \mathbb{N}$, $s_1 \mathcal{R} t_j$ for $0 < j \leq n$, and $u_1 \mathcal{R} u_2$".

This makes the algorithm for $\mathcal{C}$ simpler: if $K_1 > 0$ and $K_2 = 0$, then $s_1 \not\mathcal{R} s_2$ for the rate condition. Therefore, the reachability condition does not need to be checked and the lines 3–5 of the algorithm $\textsc{StepCondition}_{\lessapprox}(\mathcal{D}, \mathcal{R}, s_1, s_2)$ can be omitted. In general, the rate condition can be verified by checking the validity of the network $\mathcal{N}_\psi(\mathcal{R}, \mu_1, \mu_2)$ induced in the embedded *DTMC* $emb(\mathcal{C})$. In particular, the step condition holds if and only if there exists $\psi \leq \frac{\mathbf{R}(s_2,S)}{\mathbf{R}(s_1,S)}$ such that ψ is valid for $\mathcal{N}_\psi(\mathcal{R}, \mu_1, \mu_2)$. This means that we can replace the returned value

$$\exists i \in \{1, \dots, j\}.\psi_i \text{ is valid for } \mathcal{N}_{\psi_j}(\mathcal{R}, \mu_1, \mu_2)$$

of line 7 of $\textsc{StepCondition}_{\lessapprox}(\mathcal{D}, \mathcal{R}, s_1, s_2)$ with

$$\exists i \in \{1, \dots, j\}.\psi_i \leq \frac{\mathbf{R}(s_2, S)}{\mathbf{R}(s_1, S)} \wedge \psi_i \text{ is valid for } \mathcal{N}_{\psi_j}(\mathcal{R}, \mu_1, \mu_2).$$

These improvements do not change the worst case complexity of the algorithm, but they improve it in practice, in particular when merged with the improved algorithm for *DTMC*s.

Weak Probabilistic Simulation and Bisimulation for *PA*s To complete the survey on the simulations and bisimulations defined on *PA*s, we consider the weak probabilistic (bi)simulation and the weak (bi)simulation. The latter relation is a restriction of the former where the step condition for the pair (s, t) requires that t matches the challenging transition proposed by s via a weak transition instead of a weak combined transition. By using the *PA*s proposed by [16], it is possible to show that both weak simulation and bisimulation are not transitive, so we omit them. On the contrary, both weak probabilistic simulation and bisimulation are transitive [47] and they can be used whenever we want to abstract away from the internal computation of a probabilistic automaton.

The decidability of weak probabilistic bisimulation has been stated in [13] and it is based on the standard partition refinement approach. The complexity of such algorithm is exponential in the number of transitions and only recently it has been improved to polynomial [29]. Indeed, [29] reduces the complexity to polynomial by constructing a flow network enriched with side constraints that admits a valid flow if and only if there exists a determinate scheduler that induces the desired weak combined transition.

With some inspiration from network flow problems, authors of [29, 30] were able to see a transition $t \stackrel{a}{\Longrightarrow}_{\mathrm{C}} \mu_t$ of the *PA* $\mathcal{P}$ as a *flow* where the initial probability mass δ_t flows and splits along internal transitions (and exactly one transition

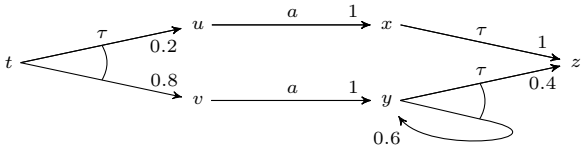


Figure 21: A probabilistic automaton

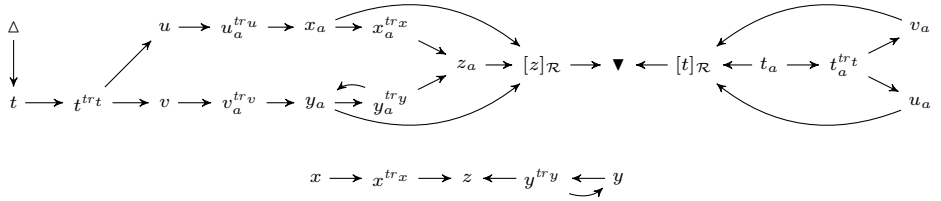


Figure 22: The network graph $G(t, a, \delta_z, \mathcal{R})$ for the automaton $\mathcal{A}$

with label a for each stream when $a \neq \tau$) according to the transition target distributions and the scheduler resolution of the nondeterminism. The resulting flow problem is then translated into an *LP* extended with *balancing constraints* that encode the need to respect transition probability distributions. To describe the structure of the *LP* problem, we first recall the original definition of the network graph corresponding to a weak combined transition.

Given a *PA* $\mathcal{P} = (S, \bar{s}, \Sigma, \rightarrow)$ and a relation $\mathcal{R} \subseteq S \times S$, for $a \in \mathtt{E}$, the network $G(t, a, \mu, \mathcal{R}) = (V, E, \triangle, \blacktriangledown, c)$ has the set of vertices $V = \{\triangle, \blacktriangledown\} \cup S \cup S^{tr} \cup S_a \cup S_a^{tr} \cup S_{\mathcal{R}}$ where

$$
\begin{aligned}
S^{tr} &= \{\, v^{tr} \mid tr = v \xrightarrow{b} \rho \in \rightarrow, b \in \{a, \tau\} \,\}, \\
S_a &= \{\, v_a \mid v \in S \,\}, \\
S_a^{tr} &= \{\, v_a^{tr} \mid v^{tr} \in S^{tr} \,\}, \text{ and} \\
S_{\mathcal{R}} &= \{\, s_{\mathcal{R}} \mid s \in S \,\}
\end{aligned}
$$

and the set of edges

$$
\begin{aligned}
E = {} & \{(\triangle, t)\} \cup \{\, (v_a, u_{\mathcal{R}}), (u_{\mathcal{R}}, \blacktriangledown) \mid u \in S, v \,\mathcal{R}\, u \,\} \\
& \cup \{\, (v, v^{tr}), (v^{tr}, v'), (v_a, v_a^{tr}), (v_a^{tr}, v_a') \mid tr = v \xrightarrow{\tau} \rho \in \rightarrow, v' \in \mathrm{Supp}(\rho) \,\} \\
& \cup \{\, (v, v_a^{tr}), (v_a^{tr}, v_a') \mid tr = v \xrightarrow{a} \rho \in \rightarrow, v' \in \mathrm{Supp}(\rho) \,\}.
\end{aligned}
$$

For each $(u, v) \in E$, $c(u, v) = \infty$. When $a \in \mathtt{H}$, the definition is similar: $V = \{\triangle, \blacktriangledown\} \cup S \cup S^{tr} \cup S_{\mathcal{R}}$ and $E = \{(\triangle, t)\} \cup \{\, (v, u_{\mathcal{R}}), (u_{\mathcal{R}}, \blacktriangledown) \mid u \in S, v \,\mathcal{R}\, u \,\} \cup \{\, (v, v^{tr}), (v^{tr}, v') \mid tr = v \xrightarrow{\tau} \rho \in \rightarrow, v' \in \mathrm{Supp}(\rho) \,\}$. As a concrete example to illustrate the construction of the flow network, consider the probabilistic automaton $\mathcal{P}$ given in Figure 21. The flow network $G(t, a, \delta_z, \mathcal{R})$ is depicted in Figure 22, where $\mathcal{R}$ is the equivalence relation inducing classes $\{t, u, v\}$ and $\{x, y, z\}$.

As pointed out in [29], the fact that the network admits a flow that respects the probability distribution μ_t does not imply the existence of a corresponding weak combined transition, because the flow may not respect probability ratios. Moreover, in order to define a flow problem, we need to define the capacity for each arc. There are several possibilities for doing this: the first possibility is to use as capacity for the arc (v^{tr}, u) corresponding to the transition $tr = s \xrightarrow{\tau} \rho$ with $u \in \mathrm{Supp}(\rho)$ the probability $\rho(u)$; the capacity of the remaining arcs is 1. As we will see, such capacity in general is not suitable for arcs that are part of cycles. Another possibility is to use as capacity the value $\frac{1}{1-\rho(u)}$ for arcs of the kind (v^{tr}, u), $\max\{ \frac{1}{1-\rho(u)} \mid u \in \mathrm{Supp}(\mu) \}$ for the arc (v, v^{tr}), and 1 for other arcs; in this case such capacity is suitable for the arcs involved in cycles, but still it does not force to respect probability ratios. Finally, arcs have infinite capacity; this is the simplest choice that has been adopted in [29]. Therefore, the network is converted into a linear programming problem for which the feasibility is shown to be equivalent to the existence of the desired weak combined transition. The idea is to convert the flow network into the canonical *LP* problem and then to add the balancing constraints that force the "flow" to split according to transition probability distributions.

Definition 21 (cf. [30, Definition 1]). *Given a PA $\mathcal{P}$, $\mathcal{R} \subseteq S \times S$, $\mu \in \mathrm{Disc}(S)$, and $t \in S$, for $a \in \mathtt{E}$ we define the $t \stackrel{a}{\Longrightarrow}_{\mathrm{C}} \circledast \mathcal{L}(\mathcal{R})\ \mu$ LP problem associated to the network graph $(V, E) = G(t, a, \mu, \mathcal{R})$ as follows:*

$\max \sum_{(x,y)\in E} -f_{x,y}$
under constraints

$f_{u,v} \geq 0$	*for each* $(u, v) \in E$
$f_{\triangle,t} = 1$	
$f_{v_{\mathcal{R}},\blacktriangledown} = \mu(v)$	*for each* $v \in S_{\mathcal{R}}$
$\sum_{u\in\{\, x\mid(x,v)\in E\,\}} f_{u,v} - \sum_{u\in\{\, y\mid(v,y)\in E\,\}} f_{v,u} = 0$	*for each* $v \in V \setminus \{\triangle, \blacktriangledown\}$
$f_{v^{tr},v'} - \rho(v') f_{v,v^{tr}} = 0$	*for each* $tr = v \xrightarrow{\tau} \rho \in \rightarrow$ *and* $v' \in \mathrm{Supp}(\rho)$
$f_{v_a^{tr},v'_a} - \rho(v') f_{v_a,v_a^{tr}} = 0$	*for each* $tr = v \xrightarrow{\tau} \rho \in \rightarrow$ *and* $v' \in \mathrm{Supp}(\rho)$
$f_{v_a^{tr},v'_a} - \rho(v') f_{v,v_a^{tr}} = 0$	*for each* $tr = v \xrightarrow{a} \rho \in \rightarrow$ *and* $v' \in \mathrm{Supp}(\rho)$

When a is τ, the *LP* problem $t \stackrel{\tau}{\Longrightarrow}_{\mathrm{C}} \circledast \mathcal{L}(\mathcal{R})\ \mu$ associated to $G(t, \tau, \mu, \mathcal{R})$ is defined as above without the last two groups of constraints.

Since it is possible to solve a linear programming problem in polynomial time [53], so it is to find a feasible solution for $t \stackrel{a}{\Longrightarrow}_{\mathrm{C}} \circledast \mathcal{L}(\mathcal{R})\ \mu$ (cf. [29, Theorem 7]), hence computing the weak probabilistic similarity and bisimilarity for probabilistic automata is polynomial as well. A comprehensive efficiency analysis about deciding weak probabilistic bisimulation on *PA*s is presented in [27].

6.3 The Algorithms for Mixed Time Models Relations

We do not present explicitly the algorithms for the simulations and bisimulations defined on *IMC*s and *MA*s: the former just makes use of the algorithms for *CTMC*s and graph visiting (as a depth first search) [28], while the latter just takes the *LP* problem for finding a weak transition in a *PA* as a blackbox [20,46].

7 Conclusion

In this survey we have presented several discrete and continuous time systems with external and/or internal nondeterminism and investigated the models of *CTMC*s, *DTMC*s, *PA*s, and *CTPA*s, and discussed *IMC*s and *MA*s. For these models, we have recalled the behavioral relations they are equipped with like simulations and bisimulations and we have described the corresponding decision algorithms. These procedures follow the standard refinement approach and they improve their complexity by using algorithms for optimization and flow network problems. We omitted some of the technical details but provided extensive references to the original literature.

Acknowledgements. The authors would like to thank Holger Hermanns for his invaluable assistance to motivate and improve the results of the paper. We would also like to thank the three anonymous reviewers for their many constructive comments and suggestions on the manuscript. This work has been supported by the DFG/NWO Bilateral Research Programme ROCKS, by the DFG as part of the SFB/TR 14 "Automatic Verification and Analysis of Complex Systems" (AVACS), and by the European Union Seventh Framework Programme under grant agreement no. 295261 (MEALS) and 318490 (SENSATION).

References

[1] Ahuja, R.K., Magnanti, T.J., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall (1993)
[2] Andova, S., Willemse, T.A.C.: Branching bisimulation for probabilistic systems: Characteristics and decidability. TCS 356(3), 325–255 (2006)
[3] Aziz, A., Sanwal, K., Singhal, V., Brayton, R.K.: Model-checking continuous-time Markov chains. ACM Transactions on Computational Logic 1(1), 162–170 (2000)
[4] Baier, C., Engelen, B., Majster-Cederbaum, M.: Deciding bisimilarity and similarity for probabilistic processes. J. Computer and Systems Science 60(1), 187–231 (2000)
[5] Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time Markov chains. IEEE Transactions on Software Engineering 29(6), 524–541 (2003)
[6] Baier, C., Hermanns, H.: Weak bisimulation for fully probabilistic processes. In: CAV. LNCS, vol. 1254, pp. 119–130 (1997)
[7] Baier, C., Hermanns, H., Katoen, J.P., Haverkort, B.R.: Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. TCS 345(1), 2–26 (2005)
[8] Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
[9] Baier, C., Katoen, J.P., Hermanns, H., Wolf, V.: Comparative branching-time semantics for Markov chains. I&C 200(2), 149–214 (2005)
[10] Bellman, R.: A Markovian decision process. Indiana University Mathematics Journal 6, 679–684 (1957)
[11] Bertsekas, D.P.: Dynamic Programming and Optimal Control. Athena Scientific (2005)
[12] Bertsimas, D., Tsitsiklis, J.N.: Introduction to Linear Optimization. Athena Scientific (1997)

[13] Cattani, S., Segala, R.: Decision algorithms for probabilistic bisimulation. In: CONCUR. LNCS, vol. 2421, pp. 371–385 (2002)
[14] Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Transactions on Programming Languages and Systems 16(5), 1512–1542 (1994)
[15] Crafa, S., Ranzato, F.: Probabilistic bisimulation and simulation algorithms by abstract interpretation. In: ICALP. LNCS, vol. 6756, pp. 295–306 (2011)
[16] Deng, Y.: Axiomatisations and Types for Probabilistic and Mobile Processes. Ph.D. thesis, École des Mines de Paris (2005)
[17] Deng, Y., Hennessy, M.: On the semantics of Markov automata. I&C 222, 139–168 (2012)
[18] Desharnais, J.: Labelled Markov Processes. Ph.D. thesis, McGill University (1999)
[19] Eisentraut, C., Hermanns, H., Katoen, J.P., Zhang, L.: A semantics for every GSPN. In: PETRI NETS. Lecture Notes in Computer Science, vol. 7927, pp. 90–109 (2013)
[20] Eisentraut, C., Hermanns, H., Krämer, J., Turrini, A., Zhang, L.: Deciding bisimilarities on distributions. In: QEST. LNCS, vol. 5084, pp. 72–88 (2013)
[21] Eisentraut, C., Hermanns, H., Schuster, J., Turrini, A., Zhang, L.: The quest for minimal quotients for probabilistic automata. In: TACAS. LNCS, vol. 7795, pp. 16–31 (2013)
[22] Eisentraut, C., Hermanns, H., Zhang, L.: Concurrency and composition in a stochastic world. In: CONCUR. LNCS, vol. 6269, pp. 21–39 (2010)
[23] Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: LICS. pp. 342–351 (2010)
[24] Gallo, G., Grigoriadis, M.D., Tarjan, R.E.: A fast parametric maximum flow algorithm and applications. SIAM J. Comp. 18(1), 30–55 (1989)
[25] Goldberg, A.V., Tarjan, R.E.: A new approach to the maximum-flow problem. J. ACM 35(4), 921–940 (1988)
[26] Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6(5), 512–535 (1994)
[27] Hashemi, V., Hermanns, H., Turrini, A.: On the efficiency of deciding probabilistic automata weak bisimulation. ECEASST 66 (2013)
[28] Hermanns, H.: Interactive Markov Chains: The Quest for Quantified Quality, LNCS, vol. 2428. Springer (2002)
[29] Hermanns, H., Turrini, A.: Deciding probabilistic automata weak bisimulation in polynomial time. In: FSTTCS. pp. 435–447 (2012)
[30] Hermanns, H., Turrini, A.: Cost preserving bisimulations for probabilistic automata. In: CONCUR. LNCS, vol. 8052, pp. 349–363 (2013)
[31] Howard, R.A.: Dynamic Programming and Markov Processes. John Wiley and Sons, Inc. (1960)
[32] Howard, R.A.: Dynamic Probabilistic Systems, Volume II: Semi-Markov and Decision Processes. Dover Publications (2007)
[33] Jansen, D.N., Song, L., Zhang, L.: Revisiting weak simulation for substochastic Markov chains. In: QEST. LNCS, vol. 8054, pp. 209–224 (2013)
[34] Jonsson, B., Larsen, K.G.: Specification and refinement of probabilistic processes. In: LICS. pp. 266–277 (1991)
[35] Kanellakis, P.C., Smolka, S.A.: CCS expressions, finite state processes, and three problems of equivalence. I&C 86(1), 43–68 (1990)
[36] Katoen, J.P., Kemna, T., Zapreev, I.S., Jansen, D.N.: Bisimulation minimisation mostly speeds up probabilistic model checking. In: TACAS. LNCS, vol. 4424, pp. 76–92 (2007)

[37] Knast, R.: Continuous-time probabilistic automata. Information and Control 15(4), 335–352 (1969)
[38] Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing (preliminary report). In: POPL. pp. 344–352 (1989)
[39] Milner, R.: Communication and Concurrency. Prentice-Hall International, Englewood Cleiffs (1989)
[40] Milner, R.: Communicating and Mobile Systems: the π-calculus. Cambridge University Press (1999)
[41] Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM J. on Computing 16(6), 973–989 (1987)
[42] Peterson, M.: An Introduction to Decision Theory. Cambridge University Press (2009)
[43] Philippou, A., Lee, I., Sokolsky, O.: Weak bisimulation for probabilistic systems. In: CONCUR. LNCS, vol. 1877, pp. 334–349 (2000)
[44] Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. No. 594 in Wiley Series in Probability and Statistics, John Wiley & Sons, Inc. (2005)
[45] Sack, J., Zhang, L.: A general framework for probabilistic characterizing formulae. In: VMCAI. pp. 396–411 (2012)
[46] Schuster, J., Siegle, M.: Markov automata: Deciding weak bisimulation by means of "non-naïvely" vanishing states. I&C (2014), to appear, available at `http://dx.doi.org/10.1016/j.ic.2014.02.001`
[47] Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. Ph.D. thesis, MIT (1995)
[48] Segala, R.: Probability and nondeterminism in operational models of concurrency. In: CONCUR. LNCS, vol. 4137, pp. 64–78 (2006)
[49] Segala, R., Lynch, N.A.: Probabilistic simulations for probabilistic processes. In: CONCUR. LNCS, vol. 836, pp. 481–496 (1994)
[50] Segala, R., Lynch, N.A.: Probabilistic simulations for probabilistic processes. Nordic J. Computing 2(2), 250–273 (1995)
[51] Segala, R., Turrini, A.: Comparative analysis of bisimulation relations on alternating and non-alternating probabilistic models. In: QEST. pp. 44–53 (2005)
[52] Stewart, W.J.: Introduction to the Numerical Solution of Markov Chains. Princeton University Press (1994)
[53] Todd, M.J.: The many facets of linear programming. Mathematical Programming 91(3), 417–436 (2002)
[54] Wolovick, N., Johr, S.: A characterization of meaningful schedulers for continuous-time Markov decision processes. In: Formal Modeling and Analysis of Timed Systems. LNCS, vol. 4202, pp. 352–367 (2006)
[55] Zhang, L.: Decision Algorithm for Probabilistic Simulations. Ph.D. thesis, Saarland University (2008)
[56] Zhang, L., Hermanns, H.: Deciding bisimulations on probabilistic automata. In: ATVA. LNCS, vol. 4762, pp. 207–222 (2007)
[57] Zhang, L., Hermanns, H., Eisenbrand, F., Jansen, D.N.: Flow faster: Efficient decision algorithms for probabilistic simulations. In: TACAS. LNCS, vol. 4424, pp. 155–169 (2007)