Radboud Repository

Radboud University Nijmegen

PDF hosted at the Radboud Repository of the Radboud University Nijmegen

The following full text is a publisher's version.

For additional information about this publication click this link.
http://hdl.handle.net/2066/34786

Please be advised that this information was generated on 2017-12-06 and may be subject to change.

FLOW FASTER: EFFICIENT DECISION ALGORITHMS FOR PROBABILISTIC SIMULATIONS *

LIJUN ZHANG [a], HOLGER HERMANNS [b], FRIEDRICH EISENBRAND [c], AND DAVID N. JANSEN [d]

[a,b] Department of Computer Science, Saarland University, Germany
e-mail address: {zhang,hermanns}@cs.uni-sb.de

[c] Department of Mathematics, EPFL, Switzerland
e-mail address: friedrich.eisenbrand@epfl.ch

[d] Software Modelling and Verification, RWTH Aachen University, Germany, and Model-Based System Development, Radboud University, Nijmegen, The Netherlands
e-mail address: D.Jansen@cs.ru.nl

Abstract. Strong and weak simulation relations have been proposed for Markov chains, while strong simulation and strong probabilistic simulation relations have been proposed for probabilistic automata. This paper investigates whether they can be used as effectively as their non-probabilistic counterparts. It presents drastically improved algorithms to decide whether some (discrete- or continuous-time) Markov chain strongly or weakly simulates another, or whether a probabilistic automaton strongly simulates another. The key innovation is the use of parametric maximum flow techniques to amortize computations. We also present a novel algorithm for deciding strong probabilistic simulation preorders on probabilistic automata, which has polynomial complexity via a reduction to an LP problem. When extending the algorithms for probabilistic automata to their continuous-time counterpart, we retain the same complexity for both strong and strong probabilistic simulations.

1. Introduction

Many verification methods have been introduced to prove the correctness of systems exploiting rigorous mathematical foundations. As one of the automatic verification techniques, model checking has successfully been applied to automatically find errors in complex systems. The power of model checking is limited by the state space explosion problem. Notably,

minimizing the system to the bisimulation [34, 35] quotient is a favorable approach. As a more aggressive attack to the problem, simulation relations [33] have been proposed for these models. In particular, they provide the principal ingredients to perform abstractions of the models, while preserving safe CTL properties (formulas with positive universal path-quantifiers only) [16].

Simulation relations are preorders on the state space such that whenever state s' simulates state s (written $s \precsim s'$) then s' can mimic all stepwise behaviour of s, but s' may perform steps that cannot be matched by s. One of the interesting aspects of simulation relations is that they allow a verification by "local" reasoning. Based on this, efficient algorithms for deciding simulation preorders have been proposed in [10, 23].

Randomisation has been employed widely for performance and dependability models, and consequently the study of verification techniques of probabilistic systems with and without nondeterminism has drawn a lot of attention in recent years. A variety of equivalence and preorder relations, including strong and weak simulation relations, have been introduced and widely considered for probabilistic models. In this paper we consider discrete-time Markov chains (DTMCs) and discrete-time probabilistic automata (PAs) [39]. PAs extend labelled transition systems (LTSs) with probabilistic selection, or, viewed differently, extend DTMCs with nondeterminism. They constitute a natural model of concurrent computation involving random phenomena. In a PA, a labelled transition leads to a probability distribution over the set of states, rather than a single state. The resulting model thus exhibits both non-deterministic choice (as in LTSs) and probabilistic choice (as in Markov chains).

Strong simulation relations have been introduced [26, 30] for probabilistic systems. For $s \precsim s'$ (s' strongly simulates s), it is required that every successor distribution of s via action α (called α-successor distribution) has a corresponding α-successor distribution at s'. Correspondence of distributions is naturally defined with the concept of weight functions [26]. In the context of model checking, strong simulation relations preserve safe PCTL formulas [40]. Probabilistic simulation [40] is a relaxation of strong simulation in the sense that it allows for convex combinations of multiple distributions belonging to equally labelled transitions. More concretely, it may happen that for an α-successor distribution μ of s, there is no α-successor distribution of s' which can be related to μ, yet there exists a so-called α-combined transition, a convex combination of several α-successor distributions of s'. Probabilistic simulation accounts for this and is thus coarser than strong simulation, but still preserves the same class of PCTL-properties as strong simulation does.

Apart from discrete time models, this paper considers continuous-time Markov chains (CTMCs) and continuous-time probabilistic automata (CPAs) [29, 42]. In CPAs, the transition delays are governed by exponential distributions. CPAs can be considered also as extensions of CTMCs with nondeterminism. CPAs are natural foundational models for various performance and dependability modelling formalisms including stochastic activity networks [37], generalised stochastic Petri nets [32] and interactive Markov chains [24]. Strong simulation and probabilistic simulation have been introduced for continuous-time models [9, 44]. For CTMCs, $s \precsim s'$ requires that $s \precsim s'$ holds in the embedded DTMC, and additionally, state s' must be "faster" than s which manifests itself by a higher exit rate. Both strong simulation and probabilistic simulation preserve safe CSL formulas [4], which is a continuous stochastic extension of PCTL, tailored to continuous-time models.

Weak simulation is proposed in [9] for Markov chains. In weak simulation, the successor states are split into visible and invisible parts, and the weight function conditions are only imposed on the transitions leading to the visible parts of the successor states. Weak

simulation is strictly coarser than the afore-mentioned strong simulation for Markov chains, thus allows further reduction of the state space. It preserves the safe PCTL- and CSL-properties without the next state formulas for DTMCs and CTMCs respectively [9].

Decision algorithms for strong and weak simulations over Markov chains, and for strong simulation over probabilistic automata are not efficient, which makes it as yet unclear whether they can be used as effectively as their non-probabilistic counterparts. In this paper we improve efficient decision algorithms, and devise new algorithms for deciding strong and strong probabilistic simulations for probabilistic automata. Given the simulation preorder, the simulation quotient automaton is in general smaller than the bisimulation quotient automaton. Then, for safety and liveness properties, model checking can be performed on this smaller quotient automata. The study of decision algorithms is also important for specification relations: The model satisfies the specification if the automaton for the specification simulates the automaton for the model. In many applications the specification cannot be easily expressed by logical formulas: it is rather a probabilistic model itself. Examples of this kind include various recent wireless network protocols, such as ZigBee [21], Firewire Zeroconf [11], or the novel IEEE 802.11e, where the central mechanism is selecting among different-sided dies, readily expressible as a probabilistic automaton [31].

The common strategy used by decision algorithms for simulations is as follows. The algorithm starts with a relation R which is guaranteed to be coarser than the simulation preorder $\precsim$. Then, the relation R is successively be refined. In each iteration of the refinement loop, pairs (s, s') are eliminated from the relation R if the corresponding simulation conditions are violated with respect to the current relation. In the context of labelled transitions systems, this happens if s has a successor state t, but we cannot find a successor state t' of s' such that (t, t') is also in the current relation R. For DTMCs, this correspondence is formulated by the existence of a weight function for distributions $(\mathbf{P}(s, \cdot), \mathbf{P}(s', \cdot))$ with respect to the current relation R. Checking this weight function condition amounts to checking whether there is a maximum flow over the network constructed out of $(\mathbf{P}(s, \cdot), \mathbf{P}(s', \cdot))$ and the current relation R. The complexity for one such check is however rather expensive, it has time complexity $\mathcal{O}(n^3/\log n)$. If the iterative algorithm reaches a fix-point, the strong simulation preorder is obtained. The number of iterations of the refinement loop is at most $\mathcal{O}(n^2)$, and the overall complexity [3] amounts to $\mathcal{O}(n^7/\log n)$ in time and $\mathcal{O}(n^2)$ in space.

Fixing a pair (s, s'), we observe that the networks for this pair across iterations of the refinement loop are very similar: They differ from iteration to iteration only by deletion of some edges induced by the successive clean up of R. We exploit this by adapting a parametric maximum flow algorithm [18] to solve the maximum flow problems for the arising sequences of similar networks, hence arriving at efficient simulation decision algorithms. The basic idea is that all computations concerning the pair (s, s') can be performed in an incremental way: after each iteration we save the current network together with maximum flow information. Then, in the next iteration, we update the network, and derive the maximum flow while using the previous maximum flow function. The maximum flow problems for the arising sequences of similar networks with respect to the pair (s, s') can be computed in time $\mathcal{O}(|V|^3)$ where $|V|$ is the number of nodes of the network. This leads to an overall time complexity $\mathcal{O}(m^2n)$ for deciding the simulation preorder. Because of the storage of the networks, the space complexity is increased to $\mathcal{O}(m^2)$. Especially in the very common case where the state fanout of a model is bounded by a constant g (and hence $m \leq gn$), our strong simulation algorithm has time and space complexity $\mathcal{O}(n^2)$. The algorithm can be extended easily to handle CTMCs with same time and space complexity. For weak

simulation on Markov chains, the parametric maximum flow technique cannot be applied directly. Nevertheless, we manage to incorporate the parametric maximum flow idea into a decision algorithm with time complexity $\mathcal{O}(m^2 n^3)$ and space complexity $\mathcal{O}(n^2)$. An earlier algorithm [6] uses LP problems [27, 38] as subroutines. The maximum flow problem is a special instance of an LP problem but can be solved much more efficiently [1].

We extend the algorithm to compute strong simulation preorder to also work on PAs. It takes the skeleton of the algorithm for Markov chains: It starts with a relation R which is coarser than $\precsim$, and then refines R until $\precsim$ is achieved. In the refinement loop, a pair (s, s') is eliminated if the corresponding simulation conditions are violated with respect to the current relation. For PAs, this means that there exists an α-successor distribution μ of s, such that for all α-successor distribution μ' of s', we cannot find a weight function for (μ, μ') with respect to the current relation R. Again, as for Markov chains, the existence of such weight functions can be reduced to maximum flow problems. Combining with the parametric maximum flow algorithm [18], we arrive at the same time complexity $\mathcal{O}(m^2 n)$ and space complexity $\mathcal{O}(m^2)$ as for Markov chains. The above maximum flow based procedure cannot be applied to deal with strong *probabilistic* simulation for PAs. The reason is that an α-combined transition of state s is a convex combination of several α-successor distributions of s, thus induces uncountable many such possible combined transitions. The computational complexity of deciding strong probabilistic simulation has not been investigated before. We show that it can be reduced to solving LP problems. The idea is that we introduce for each α-successor distribution a variable, and then reformulates the requirements concerning the combined transitions by linear constraints over these variables. This allows us to construct a set of LP problem such that whether a pair (s, s') should be thrown out of the current pair R is equivalent to whether each of the LP problem has a solution.

The algorithms for PAs are then extended to handle their continuous-time analogon, CPAs. In the algorithm, for each pair (s, s') in the refinement loop, an additional rate condition is ensured by an additional check via comparing the appropriate rates of s and s'. The resulting algorithm has the same time and space complexity.

Related Works. In the non-probabilistic setting, the most efficient algorithms for deciding simulation preorders have been proposed in [10, 23]. The complexity is $\mathcal{O}(mn)$ where n and m denote the number of states and transitions of the transition system respectively. For Markov chains, Derisavi *et al.* [17] presented an $\mathcal{O}(m \log n)$ algorithm for strong bisimulation. Weak bisimulation for DTMCs can be computed in $\mathcal{O}(n^3)$ time [5]. For strong simulation, Baier *et al.* [3] introduced a polynomial decision algorithm with complexity $\mathcal{O}(n^7 / \log n)$, by tailoring a network flow algorithm [20] to the problem, embedded into an iterative refinement loop. In [6], Baier *et al.* proved that weak simulation is decidable in polynomial time by reducing it to linear programming (LP) problems. For a subclass of PAs (reactive systems), Huynh and Tian [25] presented an $\mathcal{O}(m \log n)$ algorithm for computing strong bisimulation. Cattani and Segala [12] have presented decision algorithms for strong and *bi*simulation for PAs. They reduced the decision problems to LP problems. To compute the coarsest strong simulation for PAs, Baier *et al.* [3] presented an algorithm which reduces the query whether a state strongly simulates another to a maximum flow problem. Their algorithm has complexity $\mathcal{O}((mn^6 + m^2 n^3)/ \log n)$[1]. Recently, algorithm for computing simulation and bisimulation metrics for concurrent games [13] has been studied.

---

[1]The m used in paper [3] is slightly different from the m as we use it. A detailed comparison is provided later, in Remark 4.11 of Section 4.3.

Outline of The Paper. The paper proceeds by recalling the definition of the models and simulation relations in Section 2. In Section 3 we give a short interlude on maximum flow problems. In Section 4 we present a combinatorial method to decide strong simulations. In this section we also introduce new decision algorithms for deciding strong probabilistic simulations for PAs and CPAs. In Section 5 we focus on algorithms for weak simulations. Section 6 concludes the paper.

2. PRELIMINARIES

In Subsection 2.1, we recall the definitions of fully probabilistic systems, discrete- and continuous-time Markov chains [41], and the nondeterministic extensions of these discrete-time [40] and continuous-time models [36, 7]. In Subsection 2.2 we recall the definition of simulation relations.

2.1. **Markov Models.** Firstly, we introduce some general notations. Let X, Y be finite sets. For $f : X \to \mathbb{R}$, let $f(A)$ denote $\sum_{x \in A} f(x)$ for all $A \subseteq X$. For $f : X \times Y \to \mathbb{R}$, we let $f(x, A)$ denote $\sum_{y \in A} f(x, y)$ for all $x \in X$ and $A \subseteq Y$, and $f(A, y)$ is defined similarly. Let AP be a fixed, finite set of atomic propositions.

For a finite set S, a distribution μ over S is a function $\mu : S \to [0, 1]$ satisfying the condition $\mu(S) \leq 1$. The support of μ is defined by $Supp(\mu) = \{s \mid \mu(s) > 0\}$, and the size of μ is defined by $|\mu| = |Supp(\mu)|$. The distribution μ is called stochastic if $\mu(S) = 1$, absorbing if $\mu(S) = 0$. We sometimes use an auxiliary state (not a *real* state) $\bot \notin S$ and set $\mu(\bot) = 1 - \mu(S)$. If μ is not stochastic we have $\mu(\bot) > 0$. Further, let $S_\bot$ denote the set $S \cup \{\bot\}$, and let $Supp_\bot(\mu) = Supp(\mu) \cup \{\bot\}$ if $\mu(\bot) > 0$ and $Supp_\bot(\mu) = Supp(\mu)$ otherwise. We let $Dist(S)$ denote the set of distributions over the set S.

Definition 2.1. A labelled fully probabilistic system (FPS) is a tuple $\mathcal{M} = (S, \mathbf{P}, L)$ where S is a finite set of states, $\mathbf{P} : S \times S \to [0, 1]$ is a probability matrix such that $\mathbf{P}(s, \cdot) \in Dist(S)$ for all $s \in S$, and $L : S \to 2^{AP}$ is a labelling function.

A state s is called stochastic and absorbing if the distribution $\mathbf{P}(s, \cdot)$ is stochastic and absorbing respectively. For $s \in S$, let $post(s) = Supp(\mathbf{P}(s, \cdot))$, and let $post_\bot(s) = Supp_\bot(\mathbf{P}(s, \cdot))$.

Definition 2.2. A labelled discrete-time Markov chain (DTMC) is an FPS $\mathcal{M} = (S, \mathbf{P}, L)$ where s is either absorbing or stochastic for all $s \in S$.

FPSs and DTMCs are *time-abstract*, since the duration between triggering transitions is disregarded. We observe the state only at a discrete set of time points $0, 1, 2, \ldots$. We recall the definition of CTMCs which are *time-aware*:

Definition 2.3. A labelled continuous-time Markov chain (CTMC) is a tuple $\mathcal{M} = (S, \mathbf{R}, L)$ with S and L as before, and $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$ is a rate matrix.

For CTMC $\mathcal{M}$, let $post(s) = \{s' \in S \mid \mathbf{R}(s, s') > 0\}$ for all $s \in S$. The rates give the average delay of the corresponding transitions. Starting from state s, the probability that within time t a successor state is chosen is given by $1 - e^{-\mathbf{R}(s, S)t}$. The probability that a specific successor state s' is chosen within time t is thus given by $(1 - e^{-\mathbf{R}(s, S)t}) \cdot \mathbf{R}(s, s')/\mathbf{R}(s, S)$. A CTMC induces an embedded DTMC, which captures the time-abstract behaviour of it:

Definition 2.4. Let $\mathcal{M} = (S, \mathbf{R}, L)$ be a CTMC. The embedded DTMC of $\mathcal{M}$ is defined by $emb(\mathcal{M}) = (S, \mathbf{P}, L)$ with $\mathbf{P}(s, s') = \mathbf{R}(s, s')/\mathbf{R}(s, S)$ if $\mathbf{R}(s, S) > 0$ and 0 otherwise.

We will also use $\mathbf{P}$ for a CTMC directly, without referring to its embedded DTMC explicitly. If one is interested in time-abstract properties (e. g., the probability to reach a set of states) of a CTMC, it is sufficient to analyse its embedded DTMC.

For a given FPS, DTMC or CTMC, its *fanout* is defined by $\max_{s \in S} |post(s)|$. The number of states is defined by $n = |S|$, and the number of transitions is defined by $m = \sum_{s \in S} |post(s)|$. For $s \in S$, $reach(s)$ denotes the set of states that are reachable from s with positive probability. For a relation $R \subseteq S \times S$ and $s \in S$, let $R(s)$ denote the set $\{s' \in S \mid (s, s') \in R\}$. Similarly, for $s' \in S$, let $R^{-1}(s')$ denote the set $\{s \in S \mid (s, s') \in R\}$. If $(s, s') \in R$, we write also $s \, R \, s'$.

Markov chains are purely probabilistic. Now we consider extensions of Markov chains with nondeterminism. We first recall the definition of probabilistic automata, which can be considered as the *simple probabilistic automata* with transitions allowing deadlocks in [39].

Definition 2.5. A probabilistic automaton (PA) is a tuple $\mathcal{M} = (S, Act, \mathbf{P}, L)$ where S and L are defined as before, Act is a finite set of actions, $\mathbf{P} \subseteq S \times Act \times Dist(S)$ is a finite set, called the probabilistic transition matrix.

For $(s, \alpha, \mu) \in \mathbf{P}$, we use $s \xrightarrow{\alpha} \mu$ as a shorthand notation, and call μ an α-successor distribution of s. Let $Act(s) = \{\alpha \mid \exists \mu : s \xrightarrow{\alpha} \mu\}$ denote the set of actions enabled at s. For $s \in S$ and $\alpha \in Act(s)$, let $Steps_\alpha(s) = \{\mu \in Dist(S) \mid s \xrightarrow{\alpha} \mu\}$ and $Steps(s) = \bigcup_{\alpha \in Act(s)} Steps_\alpha(s)$. The fanout of a state s is defined by $fan(s) = \sum_{\alpha \in Act(s)} \sum_{\mu \in Steps_\alpha(s)} (|\mu| + 1)$. Intuitively, $fan(s)$ denotes the total sum of the sizes of outgoing distributions of state s plus their labelling. The fanout of $\mathcal{M}$ is defined by $\max_{s \in S} fan(s)$. Summing up over all states, we define the size of the transitions by $m = \sum_{s \in S} fan(s)$.

A *Markov decision process* (MDP) [36] arises from a PA $\mathcal{M}$ if for $s \in S$ and $\alpha \in Act$, there is at most one α-successor distribution μ of s, which must be stochastic.

We consider a continuous-time counterpart of PAs where the transitions are described by rates instead of probabilities. A rate function is simply a function $r : S \to \mathbb{R}_{\geq 0}$. Let $|r| = |\{s \mid r(s) > 0\}|$ denote the size of r. Let $Rate(S)$ denote the set of all rate functions.

Definition 2.6. A continuous-time PA (CPA) is a tuple $(S, Act, \mathbf{R}, L)$ where S, Act, L as defined for PAs, and $\mathbf{R} \subseteq S \times Act \times Rate(S)$ a finite set, called the rate matrix.

We write $s \xrightarrow{\alpha} r$ if $(s, \alpha, r) \in \mathbf{R}$, and call r an α-successor rate function of s. For transition $s \xrightarrow{\alpha} r$, the sum $r(S)$ is also called the exit rate of it. Given that the transition $s \xrightarrow{\alpha} r$ is chosen from state s, the probability that any successor state is chosen within time t is given by $1 - e^{-r(S)t}$, and a specific successor state s' is chosen within time t is given by $(1 - e^{-r(S)t}) \cdot \frac{r(s')}{r(S)}$. The notion of $Act(s)$, $Steps_\alpha(s)$, $Steps(s)$, fanout and size of transitions for PAs can be extended to CPAs by replacing occurrence of distribution μ by rate function r in an obvious way.

The model continuous-time Markov decision processes (CTMDPs) [36, 7] can be considered as special CPAs where for $s \in S$ and $\alpha \in Act$, there exists at most one rate function $r \in Rate(S)$ such that $s \xrightarrow{\alpha} r$. The model CTMDPs considered in paper [42] essentially agree with our CPAs.
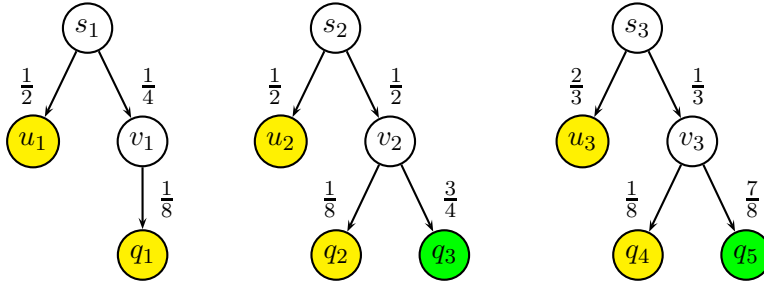
Figure 1: An FPS for illustrating the simulation relations[2].

2.2. **Strong and Weak Simulation Relations.** We first recall the notion of strong simulation on Markov chains [9], PAs [40], and CPAs [44]. Strong probabilistic simulation is defined in Subsection 2.2.2. Weak simulation for Markov chains will be given in Subsection 2.2.3. The notion of simulation up to R is introduced in Subsection 2.2.4.

2.2.1. *Strong Simulation.* Strong simulation requires that each successor distribution of one state has a corresponding successor distribution of the other state. The correspondence of distributions is naturally defined with the concept of *weight functions* [26], adapted to FPSs as in [9]. For a relation $R \subseteq S \times S$, we let $R_\perp$ denote the set $R \cup \{(\perp, s) \mid s \in S_\perp\}$.

Definition 2.7. Let $\mu, \mu' \in Dist(S)$ and $R \subseteq S \times S$. A weight function for (μ, μ') with respect to R is a function $\Delta : S_\perp \times S_\perp \to [0, 1]$ such that
(1) $\Delta(s, s') > 0$ implies $s\, R_\perp\, s'$,
(2) $\mu(s) = \Delta(s, S_\perp)$ for $s \in S_\perp$ and
(3) $\mu'(s') = \Delta(S_\perp, s')$ for $s' \in S_\perp$.
    We write $\mu \sqsubseteq_R \mu'$ if there exists a weight function for (μ, μ') with respect to R.

The first condition requires that only pairs (s, s') in the relation $R_\perp$ have a positive weight. In other words, for $s, s' \in S$ with $s' \notin R_\perp(s)$, it holds that $\Delta(s, s') = 0$. Strong simulation requires similar states to be related via weight functions on their distributions [26].

Definition 2.8. Let $\mathcal{M} = (S, \mathbf{P}, L)$ be an FPS, and let $R \subseteq S \times S$. The relation R is a strong simulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\, R\, s_2$: $L(s_1) = L(s_2)$ and $\mathbf{P}(s_1, \cdot) \sqsubseteq_R \mathbf{P}(s_2, \cdot)$.
    We say that s_2 strongly simulates s_1 in $\mathcal{M}$, denoted by $s_1 \precsim_{\mathcal{M}} s_2$, iff there exists a strong simulation R on $\mathcal{M}$ such that $s_1\, R\, s_2$.

By definition, it can be shown [9] that $\precsim_{\mathcal{M}}$ is reflexive and transitive, thus a *preorder*. Moreover, $\precsim_{\mathcal{M}}$ is the coarsest strong simulation relation for $\mathcal{M}$. If the model $\mathcal{M}$ is clear from the context, the subscript $\mathcal{M}$ may be omitted. Assume that $s_1 \precsim s_2$ and let Δ denote the corresponding weight function. If $\mathbf{P}(s_2, \perp) > 0$, we have that $\mathbf{P}(s_2, \perp) = \sum_{s \in S_\perp} \Delta(s, \perp) = \Delta(\perp, \perp)$. The second equality follows by the fact that $\perp$ can not strongly simulate any real state in S. Another observation is that if s is absorbing, then it can be strongly simulated by any other state s' with $L(s) = L(s')$.

---

[2]Although this graph is not connected, it shows a single FPS. Similarly, later figures will show a single DTMC, CTMC etc.

Example 2.9. Consider the FPS depicted in Figure 1. Recall that labelling of states is indicated by colours in the states. Since the yellow (grey) states are absorbing, they strongly simulate each other. The same holds for the green (dark grey) states. We show now that $s_1 \precsim s_2$ but $s_2 \not\precsim s_3$.

Consider first the pair (s_1, s_2). Let $R = \{(s_1, s_2), (u_1, u_2), (v_1, v_2), (q_1, q_2)\}$. We show that R is a strong simulation relation. First observe that $L(s) = L(s')$ for all $(s, s') \in R$. Since states u_1, q_1 are absorbing, the conditions for the pairs (u_1, u_2) and (q_1, q_2) hold trivially. To show the conditions for (v_1, v_2), we consider the function Δ_1 defined by: $\Delta_1(q_1, q_2) = \frac{1}{8}$, $\Delta_1(\bot, q_3) = \frac{3}{4}$, $\Delta_1(\bot, \bot) = \frac{1}{8}$ and $\Delta_1(\cdot) = 0$ otherwise. It is easy to check that Δ_1 is a weight function for $(\mathbf{P}(v_1, \cdot), \mathbf{P}(v_2, \cdot))$ with respect to R. Now consider (s_1, s_2). The weight function Δ_2 for $(\mathbf{P}(s_1, \cdot), \mathbf{P}(s_2, \cdot))$ with respect to R is given by $\Delta_2(u_1, u_2) = \frac{1}{2}$ and $\Delta_2(v_1, v_2) = \Delta_2(\bot, v_2) = \frac{1}{4}$ and $\Delta_2(\cdot) = 0$ otherwise. Thus R is a strong simulation which implies that $s_1 \precsim s_2$.

Consider the pair (s_2, s_3). Since $\mathbf{P}(s_2, v_2) = \frac{1}{2}$, to establish the Condition 2 of Definition 2.7, we should have $\frac{1}{2} = \Delta(v_2, S_\bot)$. Observe that v_3 is the only successor state of s_3 which can strongly simulate v_2, thus $\Delta(v_2, S_\bot) = \Delta(v_2, v_3)$. However, for state v_3 we have $\mathbf{P}(s_3, v_3) < \Delta(v_2, v_3)$, which violates the Condition 3 of Definition 2.7, thus we cannot find such a weight function. Hence, $s_2 \not\precsim s_3$.

Since each DTMC is a special case of an FPS, Definition 2.8 applies directly for DTMCs. For CTMCs we say that s_2 strongly simulates s_1 if, in addition to the DTMC conditions, s_2 can move stochastically *faster* than s_1 [9], which manifests itself by a higher rate.

Definition 2.10. Let $\mathcal{M} = (S, \mathbf{R}, L)$ be a CTMC and let $R \subseteq S \times S$. The relation R is a strong simulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1 \ R \ s_2$: $L(s_1) = L(s_2)$, $\mathbf{P}(s_1, \cdot) \sqsubseteq_R \mathbf{P}(s_2, \cdot)$ and $\mathbf{R}(s_1, S) \le \mathbf{R}(s_2, S)$.

We say that s_2 strongly simulates s_1 in $\mathcal{M}$, denoted by $s_1 \precsim_\mathcal{M} s_2$, iff there exists a strong simulation R on $\mathcal{M}$ such that $s_1 \ R \ s_2$.

Thus, $s \precsim_\mathcal{M} s'$ holds if $s \precsim_{emb(\mathcal{M})} s'$, and s' is faster than s. By definition, it can be shown that $\precsim_\mathcal{M}$ is a preorder, and is the coarsest strong simulation relation for $\mathcal{M}$. For PAs, strong simulation requires that every α-successor distribution of s_1 is related to an α-successor distribution of s_2 via a weight function [40, 26]:

Definition 2.11. Let $\mathcal{M} = (S, Act, \mathbf{P}, L)$ be a PA and let $R \subseteq S \times S$. The relation R is a strong simulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1 \ R \ s_2$: $L(s_1) = L(s_2)$ and if $s_1 \xrightarrow{\alpha} \mu_1$ then there exists a transition $s_2 \xrightarrow{\alpha} \mu_2$ with $\mu_1 \sqsubseteq_R \mu_2$.

We say that s_2 strongly simulates s_1 in $\mathcal{M}$, denoted $s_1 \precsim_\mathcal{M} s_2$, iff there exists a strong simulation R on $\mathcal{M}$ such that $s_1 \ R \ s_2$.

Example 2.12. Consider the PA in Figure 2. Then, it is easy to check $s_1 \precsim s_2$: each α-successor distribution of s_1 has a corresponding α-successor distribution of s_2. However, s_1 does not strongly simulate s_2, as the middle α-successor distribution of s_2 can not be related by any α-successor distribution of s_1.

Now we consider CPAs. For a rate function r, we let $\mu(r) \in Dist(S)$ denote the induced distribution defined by: if $r(S) > 0$ then $\mu(r)(s)$ equals $r(s)/r(S)$ for all $s \in S$, and if $r(S) = 0$, then $\mu(r)(s) = 0$ for all $s \in S$. Now we introduce the notion of strong simulation for CPAs [44], which can be considered as an extension of the definition for CTMCs [9]:
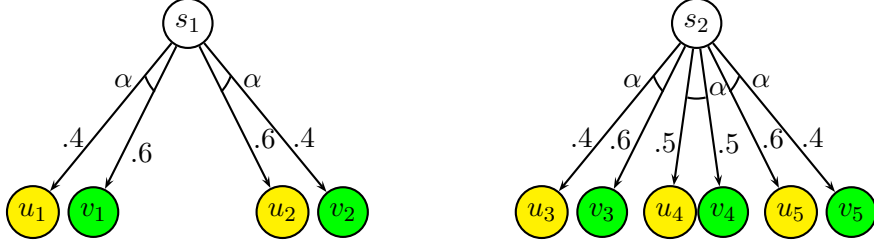
Figure 2: A PA for illustrating the simulation relations.

Definition 2.13. Let $\mathcal{M} = (S, Act, \mathbf{R}, L)$ be a CPA and let $R \subseteq S \times S$. The relation R is a strong simulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\, R\, s_2$: $L(s_1) = L(s_2)$ and if $s_1 \xrightarrow{\alpha} r_1$ then there exists a transition $s_2 \xrightarrow{\alpha} r_2$ with $\mu(r_1) \sqsubseteq_R \mu(r_2)$ and $r_1(S) \leq r_2(S)$.

　　We write $s_1 \precsim_{\mathcal{M}} s_2$ iff there exists a strong simulation R on $\mathcal{M}$ such that $s_1\, R\, s_2$.

Similar to CTMCs, the additional rate condition $r_1(S) \leq r_2(S)$ indicates that the transition $s_2 \xrightarrow{\alpha} r_2$ is faster than $s_1 \xrightarrow{\alpha} r_1$. As a shorthand notation, we use $r_1 \sqsubseteq_R r_2$ for the condition $\mu(r_1) \sqsubseteq_R \mu(r_2)$ and $r_1(S) \leq r_2(S)$. For both PAs and CPAs, $\precsim_{\mathcal{M}}$ is the coarsest strong simulation relation.

2.2.2. *Strong Probabilistic Simulations.* We recall the definition of strong probabilistic simulation, which is coarser than strong simulation, but still preserves the same class of PCTL-properties as strong simulation does. We first recall the notion of combined transition [39], a convex combination of several equally labelled transitions:

Definition 2.14. Let $\mathcal{M} = (S, Act, \mathbf{P}, L)$ be a PA. Let $s \in S$, $\alpha \in Act(s)$ and $k = |Steps_\alpha(s)|$. Assume that $Steps_\alpha(s) = \{\mu_1, \ldots, \mu_k\}$. The tuple (s, α, μ) is a combined transition, denoted by $s \xrightarrow{\alpha}_{\leadsto} \mu$, iff there exist constants $c_1, \ldots, c_k \in [0,1]$ with $\sum_{i=1}^{k} c_i = 1$ such that $\mu = \sum_{i=1}^{k} c_i \mu_i$.

The key difference to Definition 2.11 is the use of $\overset{\alpha}{\leadsto}$ instead of $\xrightarrow{\alpha}$:

Definition 2.15. Let $\mathcal{M} = (S, Act, \mathbf{P}, L)$ be a PA and let $R \subseteq S \times S$. The relation R is a strong probabilistic simulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\, R\, s_2$: $L(s_1) = L(s_2)$ and if $s_1 \xrightarrow{\alpha} \mu_1$ then there exists a combined transition $s_2 \overset{\alpha}{\leadsto} \mu_2$ with $\mu_1 \sqsubseteq_R \mu_2$.

　　We write $s_1 \precsim^p_{\mathcal{M}} s_2$ iff there exists a strong probabilistic simulation R on $\mathcal{M}$ such that $s_1\, R\, s_2$.

Strong probabilistic simulation is insensitive to combined transitions[3], thus, it is a relaxation of strong simulation. Similar to strong simulation, $\precsim^p_{\mathcal{M}}$ is the coarsest strong probabilistic simulation relation for $\mathcal{M}$. Since MDPs can be considered as special PAs, we obtain the notions of strong simulation and strong probabilistic simulation for MDPs. Moreover, these two relations coincide for MDPs as, by definition, for each state there is at most one successor distribution per action.

---

[3]The combined transition defined in [39] is more general in two dimensions: First, successor distributions are allowed to combine different actions. Second, $\sum_{i=1}^{k} c_i \leq 1$ is possible. The induced strong probabilistic probabilistic simulation preorder is, however, the same.
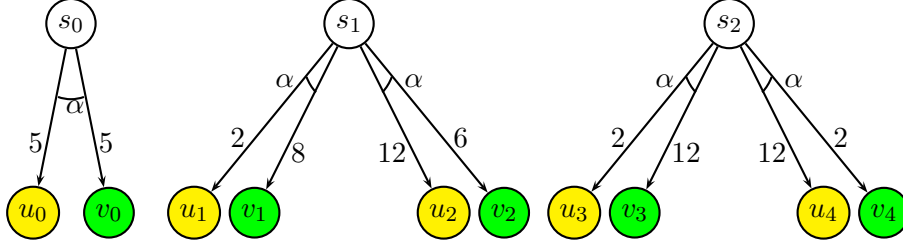
Figure 3: A Continuous-time Probabilistic Automaton.

Example 2.16. We consider again the PA depicted in Figure 2. From Example 2.12 we know that $s_2 \not\precsim s_1$. In comparison to state s_1, state s_2 has one additional α-successor distribution: to states u_4 and v_4 with equal probability 0.5. This successor distribution can be considered as a combined transition of the two successor distributions of s_1: each with constant 0.5. Hence, we have $s_2 \precsim^p s_1$.

We extend the notion of strong probabilistic simulation for PAs to CPAs. First, we introduce the notion of combined transitions for CPAs. In CPAs the probability that a transition occurs is exponentially distributed. The combined transition should also be exponentially distributed. The following example shows that a straightforward extension of Definition 2.14 does not work.

Example 2.17. For this purpose we consider the CPA in Figure 3. Let r_1 and r_2 denote left and the right α-successor rate functions out of state s_1. Obviously, they have different exit rates: $r_1(S) = 10$, $r_2(S) = 18$. Taking each with probability 0.5, we would get the combined transition $r = 0.5r_1 + 0.5r_2$: $r(\{u_1, u_2\}) = 7$ and $r(\{v_1, v_2\}) = 7$. However, r is hyper-exponentially distributed: the probability of reaching yellow (grey) states (u_1 or u_2) within time t under r is given by: $0.5 \cdot \frac{2}{10} \cdot (1 - e^{-10t}) + 0.5 \cdot \frac{12}{18} \cdot (1 - e^{-18t})$. Similarly, the probability of reaching green (dark grey) states within time t is given by: $0.5 \cdot \frac{8}{10} \cdot (1 - e^{-10t}) + 0.5 \cdot \frac{6}{18} \cdot (1 - e^{-18t})$.

From state s_2, the two α-successor rate functions have the same exit rate 14. Let r_1' and r_2' denote left and the right α-successor rate functions out of state s_2. In this case the combined transition $r' = 0.5r_1' + 0.5r_2'$ is also exponentially distributed with rate 14: the probability to reach yellow (grey) states (u_3 and u_4) within time t is $\frac{7}{14} \cdot (1 - e^{-14t})$, which is the same as the probability of reaching green (dark grey) states within time t.

Based on the above example, it is easy to see that to get a combined transition which is still exponentially distributed, we must consider rate functions with the same exit rate:

Definition 2.18. Let $\mathcal{M} = (S, Act, \mathbf{R}, L)$ be a CPA. Let $s \in S$, $\alpha \in Act(s)$ and let $\{r_1, \ldots, r_k\} \subseteq Steps_\alpha(s)$ where $r_i(S) = r_j(S)$ for $i, j \in \{1, \ldots, k\}$. The tuple (s, α, r) is a combined transition, denoted by $s \overset{\alpha}{\leadsto} r$, iff there exist constants $c_1, \ldots, c_k \in [0, 1]$ with $\sum_{i=1}^{k} c_i = 1$ such that $r = \sum_{i=1}^{k} c_i r_i$.

In the above definition, unlike for the PA case, only α-successor rate functions with the same exit rate are combined together. Similar to PAs, strong probabilistic simulation is insensitive to combined transitions, which is thus a relaxation of strong simulation:

Definition 2.19. Let $\mathcal{M} = (S, Act, \mathbf{R}, L)$ be a CPA and let $R \subseteq S \times S$. The relation R is a strong probabilistic simulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1 \mathrel{R} s_2$: $L(s_1) = L(s_2)$ and if $s_1 \overset{\alpha}{\rightarrow} r_1$ then there exists a combined transition $s_2 \overset{\alpha}{\leadsto} r_2$ with $r_1 \sqsubseteq_R r_2$.
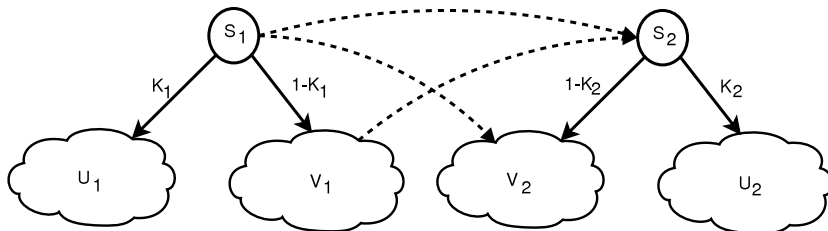
Figure 4: Splitting of successor states for weak simulations.

We write $s_1 \precsim^p_{\mathcal{M}} s_2$ iff there exists a strong simulation R on $\mathcal{M}$ such that $s_1 R s_2$.

Recall $r_1 \sqsubseteq_R r_2$ is a shorthand notation for $\mu(r_1) \sqsubseteq_R \mu(r_2)$ and $r_1(S) \leq r_2(S)$. By definition, the defined strong probabilistic simulation $\precsim^p_{\mathcal{M}}$ is the coarsest strong probabilistic simulation relation for $\mathcal{M}$.

Example 2.20. Reconsider the CPA in Figure 3. As discussed in Example 2.17, the two α-successor rate functions of s_1 cannot be combined together, thus the relation $s_0 \precsim^p s_1$ cannot be established. However, $s_0 \precsim^p s_2$ holds: denoting the left rate function of s_2 as r_1 and the right rate function as r_2, we choose as the combined rate function $r = 0.5r_1 + 0.5r_2$. Obviously, the conditions in Definition 2.19 are satisfied.

2.2.3. *Weak Simulations.* We now recall the notion of weak simulation [9] on Markov chains[4]. Intuitively, s_2 weakly simulates s_1 if they have the same labelling, and if their successor states can be grouped into sets U_i and V_i for $i = 1, 2$, satisfying certain conditions. Consider Figure 4. We can view steps to V_i as *stutter* steps while steps to U_i are *visible* steps. With respect to the visible steps, it is then required that there exists a weight function for the conditional distributions: $\mathbf{P}(s_1, \cdot)/K_1$ and $\mathbf{P}(s_2, \cdot)/K_2$ where K_i intuitively is the probability to perform a visible step from s_i. The stutter steps must respect the weak simulation relations: thus states in V_2 should weakly simulate s_1, and state s_2 should weakly simulate states in V_1. This is depicted by dashed arrows in the figure. For reasons we will explain later in Example 2.22, the definition needs to account for states which partially belong to U_i and partially to V_i. Technically, this is achieved by functions δ_i that distribute s_i over U_i and V_i in the definition below. For a given pair (s_1, s_2) and functions $\delta_i : S \to [0, 1]$, let $U_{\delta_i}, V_{\delta_i} \subseteq S$ (for $i = 1, 2$) denote the sets

$$U_{\delta_i} = \{u \in post(s_i) \mid \delta_i(u) > 0\}, \quad V_{\delta_i} = \{v \in post(s_i) \mid \delta_i(v) < 1\} \qquad (2.1)$$

If (s_1, s_2) and δ_i are clear from the context, we write U_i, V_i instead.

Definition 2.21. Let $\mathcal{M} = (S, \mathbf{P}, L)$ be a DTMC and let $R \subseteq S \times S$. The relation R is a weak simulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1 R s_2$: $L(s_1) = L(s_2)$ and there exist functions $\delta_i : S \to [0, 1]$ such that:

(1) (a) $v_1 R s_2$ for all $v_1 \in V_1$, and (b) $s_1 R v_2$ for all $v_2 \in V_2$
(2) there exists a function $\Delta : S \times S \to [0, 1]$ such that:
    (a) $\Delta(u_1, u_2) > 0$ implies $u_1 \in U_1, u_2 \in U_2$ and $u_1 R u_2$.

---

[4]In [45], we have also considered decision algorithm for weak simulation for FPSs, which is defined in [9]. However, as indicated in [43], the proposed weak simulation for FPSs contains a subtle flaw, which cannot be fixed in an obvious way. Thus, in this paper we restrict to weak simulation on DTMCs and CTMCs.
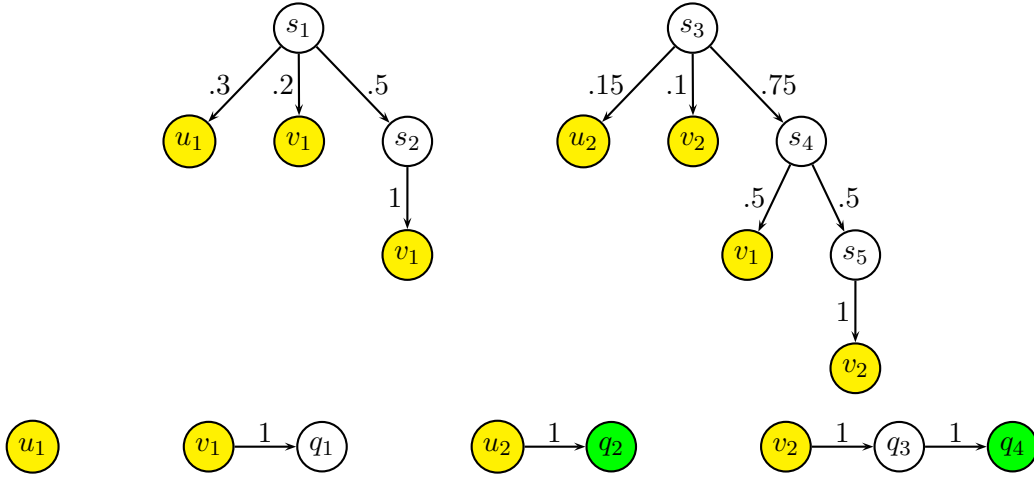
Figure 5: A DTMC where splitting states is necessary to establish the weak simulation. In the model some states are drawn more than once.

(b) if $K_1 > 0$ and $K_2 > 0$ then for all states $w \in S$:
$$K_1 \cdot \Delta(w, U_2) = \mathbf{P}(s_1, w)\delta_1(w), \quad K_2 \cdot \Delta(U_1, w) = \mathbf{P}(s_2, w)\delta_2(w)$$
where $K_i = \sum_{u_i \in U_i} \delta_i(u_i) \cdot \mathbf{P}(s_i, u_i)$ for $i = 1, 2$.

(3) for $u_1 \in U_1$ there exists a path fragment $s_2, w_1, \ldots, w_n, u_2$ with positive probability such that $n \geq 0$, $s_1 \, R \, w_j$ for $0 < j \leq n$, and $u_1 \, R \, u_2$.

We say that s_2 weakly simulates s_1 in $\mathcal{M}$, denoted $s_1 \precsim_{\mathcal{M}} s_2$, iff there exists a weak simulation R on $\mathcal{M}$ such that $s_1 \, R \, s_2$.

Note again that the sets U_i, V_i in the above definition are defined according to Equation 2.1 with respect to the pair (s_1, s_2) and the functions δ_i. The functions δ_i can be considered as a generalisation of the characteristic function of U_i in the sense that we may *split* the membership of a state to U_i and V_i into *fragments* which sum up to 1. For example, if $\delta_1(s) = \frac{1}{3}$, we say that $\frac{1}{3}$ fragment of the state s belongs to U_1, and $\frac{2}{3}$ fragment of s belongs to V_1. Hence, U_i and V_i are not necessarily disjoint. Observe that $U_i = \emptyset$ implies that $\delta_i(s) = 0$ for all $s \in S$. Similarly, $V_i = \emptyset$ implies that $\delta_i(s) = 1$ for all $s \in S$.

Condition 3 will in the sequel be called the *reachability condition*. If $K_1 > 0$ and $K_2 = 0$, which implies that $U_2 = \emptyset$ and $U_1 \neq \emptyset$, the reachability condition guarantees that for any visible step $s_1 \to u_1$ with $u_1 \in U_1$, s_2 can reach a state u_2 which simulates u_1 while passing only through states simulating s_1. Assume that we have $S = \{s_1, s_2, u_1\}$ where $L(s_1) = L(s_2)$ and u_1 has a different labelling. There is only one transition $\mathbf{P}(s_1, u_1) = 1$. Obviously $s_1 \not\precsim s_2$. Dropping Condition 3 would mean that $s_1 \precsim s_2$. We illustrate the use of fragments of states in the following example:

Example 2.22. Consider the DTMC depicted in Figure 5. For states u_1, u_2, v_1, v_2, obviously the following pairs $(u_1, u_2), (u_1, v_2), (v_1, v_2)$ are in the weak simulation relation. The state u_2 cannot weakly simulate v_1. Since v_2 weakly simulates v_1, it holds that $s_2 \precapprox s_5$. Similarly, from $u_1 \precapprox v_1$ we can easily show that $s_1 \precapprox s_4$. We observe also that $s_2 \not\precapprox s_3$: $K_1 > 0$ and $K_2 > 0$ since both s_2 and s_3 have yellow (grey) successor states, but the required function Δ cannot be established since u_2 cannot weakly simulate any successor state of s_2 (which is v_1). Thus $s_2 \not\precapprox s_3$.

Without considering fragments of states, we show that a weak simulation between s_1 and s_3 cannot be established. Since $s_2 \not\precsim s_3$, we must have $U_1 = \{u_1, v_1, s_2\}$ and $V_1 = \emptyset$. The function δ_1 is thus defined by $\delta_1(u_1) = \delta_1(v_1) = \delta(s_2) = 1$ which implies that $K_1 = 1$. Now consider the successor states of s_3. Obviously $\delta_2(u_2) = \delta_2(v_2) = 1$, which implies that $u_2, v_2 \in U_2$. We consider the following two cases:

- The case $\delta_2(s_4) = 1$. In this case we have that $K_2 = 1$. A function Δ must be defined satisfying Condition 2b in Definition 2.21. Taking $w = s_4$, the following must hold: $K_2 \cdot \Delta(U_1, s_4) = \mathbf{P}(s_3, s_4)\delta_2(s_4)$. As $K_2 = 1, \mathbf{P}(s_3, s_4) = 0.75$ and $\delta_2(s_4) = 1$, it follows that $\Delta(U_1, s_4) = 0.75$. The state s_2 is the only successor of s_1 that can be weakly simulated by s_4, so $\Delta(s_2, s_4) = 0.75$ must hold. However, the equation $K_1 \cdot \Delta(s_2, U_2) = \mathbf{P}(s_1, s_2)\delta_1(s_2)$ does not hold any more, as on the left side we have 0.75 but on the right side we have 0.5 instead.
- The case $\delta_2(s_4) = 0$. In this case we have still $K_2 > 0$. Similar to the previous case it is easy to see that the required function Δ cannot be defined: the equation $K_1 \cdot \Delta(s_2, U_2) = \mathbf{P}(s_1, s_2)\delta_1(s_2)$ does not hold since the left side is 0 (no states in U_2 can weakly simulate s_2) but the right side equals 0.5.

Thus without splitting, s_3 does not weakly simulate s_1. We show it holds that $s_1 \precsim s_3$. It is sufficient to show that the relation $R = \{(s_1, s_3), (u_1, u_2), (v_1, v_2), (q_1, q_3), (s_1, s_4), (u_1, v_1), (v_1, v_1), (q_1, q_1), (s_2, s_5), (s_2, s_4)\}$ is a weak simulation relation. By the discussions above, it is easy to verify that every pair except (s_1, s_3) satisfies the conditions in Definition 2.21. We show now that the conditions hold also for the pair (s_1, s_3). The function δ_1 with $\delta_1(u_1) = \delta_1(v_1) = \delta_1(s_2) = 1$ is defined as above, also the sets $U_1 = \{u_1, v_1, s_2\}$, $V_1 = \emptyset$. The function δ_2 is defined by: $\delta_2(u_2) = \delta_2(v_2) = 1$ and $\delta_2(s_4) = \frac{1}{3}$, which implies that $U_2 = \{u_2, v_2, s_4\}$ and $V_2 = \{s_4\}$. Thus, we have $K_1 = 1$ and $K_2 = 0.5$. Since $s_1 \precsim s_4$, Condition 1 holds trivially as $(s_1, s_4) \in R$. The reachability condition also holds trivially. To show that Condition 2 holds, we define the function Δ as follows: $\Delta(u_1, u_2) = 0.3$, $\Delta(v_1, v_2) = 0.2$ and $\Delta(s_2, s_4) = 0.5$. We show that $K_2 \cdot \Delta(U_1, w) = \mathbf{P}(s_3, w)\delta_2(w)$ holds for all $w \in S$. It holds that $K_2 = 0.5$. First observe that for $w \notin U_2$ both sides of the equation equal 0. Let first $w = u_2$ for which we have that $\mathbf{P}(s_3, u_2)\delta_2(u_2) = 0.15$. Since $\Delta(U_1, u_2) = 0.3$, also the left side equals 0.15. The case $w = v_2$ can be shown in a similar way. Now consider $w = s_4$. Observe that $\Delta(U_1, s_4) = 0.5$ thus the left side equals 0.25. The right side equals $\mathbf{P}(s_3, s_4)\delta_2(s_4) = 0.75 \cdot \frac{1}{3} = 0.25$ thus the equation holds. The equation $K_1 \cdot \Delta(w, U_2) = \mathbf{P}(s_1, w)\delta_1(w)$ can be shown in a similar way. Thus Δ satisfies all the conditions, which implies that $s_1 \precsim s_3$.

Weak simulation for CTMCs is defined as follows.

Definition 2.23 ([9, 8]). Let $\mathcal{M} = (S, \mathbf{R}, L)$ be a CTMC and let $R \subseteq S \times S$. The relation R is a weak simulation on $\mathcal{M}$ iff for $s_1 \ R \ s_2$: $L(s_1) = L(s_2)$ and there exist functions $\delta_i : S \to [0,1]$ (for $i = 1, 2$) satisfying Equation 2.1 and Conditions 1 and 2 of Definition 2.21 and the *rate condition*:

$$(3') \qquad\qquad K_1 \cdot \mathbf{R}(s_1, S) \leq K_2 \cdot \mathbf{R}(s_2, S)$$

We say that s_2 weakly simulates s_1 in $\mathcal{M}$, denoted $s_1 \precsim_{\mathcal{M}} s_2$, iff there exists a weak simulation R on $\mathcal{M}$ such that $s_1 \ R \ s_2$.

In this definition, the rate condition $3'$ strengthens the reachability condition of the preceding definition. If $U_1 \neq \emptyset$, we have that $K_1 > 0$; the rate condition then requires that

Figure 6: A simple FPS for illustrating the simulation up to R.

$K_2 > 0$, which implies $U_2 \neq \emptyset$. For both DTMCs and CTMCs, the defined weak simulation $\precsim_{\approx}$ is a preorder [9], and is the coarsest weak simulation relation for $\mathcal{M}$.

2.2.4. *Simulation up to R.* For an arbitrary relation R on the state space S of an FPS with $s_1 \, R \, s_2$, we say that s_2 simulates s_1 strongly up to R, denoted $s_1 \precsim_R s_2$, if $L(s_1) = L(s_2)$ and $\mathbf{P}(s_1, \cdot) \sqsubseteq_R \mathbf{P}(s_2, \cdot)$. Otherwise we write $s_1 \not\precsim_R s_2$. Since only the first step is considered for $\precsim_R$, $s_1 \precsim_R s_2$ does not imply $s_1 \precsim_{\mathcal{M}} s_2$ unless R is a strong simulation. By definition, R is a strong simulation if and only if for all $s_1 \, R \, s_2$ it holds that $s_1 \precsim_R s_2$. Likewise, we say that s_2 simulates s_1 weakly up to R, denoted by $s_1 \precsim_{\approx R} s_2$, if there are functions δ_i and U_i, V_i, Δ as required by Definition 2.21 for this pair of states. Otherwise, we write $s_1 \not\precsim_{\approx R} s_2$. Similar to strong simulation up to R, $s_1 \precsim_{\approx R} s_2$ does not imply $s_1 \precsim_{\approx \mathcal{M}} s_2$, since no conditions are imposed on pairs in R different from (s_1, s_2). Again, R is a weak simulation if and only if for all $s_1 \, R \, s_2$ it holds that $s_1 \precsim_{\approx R} s_2$. These conventions extend to DTMCs, CTMCs, PAs and CPAs in an obvious way. For PAs and CPAs, strong probabilistic simulation up to R, denoted by $\precsim_R^p$, is also defined analogously.

Example 2.24. Consider the FPS in Figure 6. Let $R = \{(s_1, s_2), (w_1, w_2)\}$. Since $L(q_1) \neq L(q_2)$ we have that $w_1 \not\precsim w_2$. Thus, R is not a strong simulation. However, $s_1 \precsim_R s_2$, as the weight function is given by $\Delta(w_1, w_2) = 1$. Let $R' = \{(s_1, s_2)\}$, then, $s_1 \not\precsim_{R'} s_2$.

3. Maximum Flow Problems

Before introducing algorithms to decide the simulation preorder, we briefly recall the preflow algorithm [20] for finding the maximum flow over the network $\mathcal{N} = (V, E, c)$ where V is a finite set of vertices, $E \subseteq V \times V$ is a set of edges, and $c : E \to \mathbb{R}_{>0} \cup \{\infty\}$ is the capacity function. V contains a distinguished *source* vertex $\nearrow$ and a distinguished *sink* vertex $\searrow$. We extend the capacity function to all vertex pairs: $c(v, w) = 0$ if $(v, w) \notin E$. A *flow* f on $\mathcal{N}$ is a function $f : V \times V \to \mathbb{R}$ that satisfies:

(1) $f(v, w) \leq c(v, w)$ for all $(v, w) \in V \times V$ *capacity constraints*
(2) $f(v, w) = -f(w, v)$ for all $(v, w) \in V \times V$ *antisymmetry constraint*
(3) $f(V, v) = 0$ at vertices $v \in V \setminus \{\nearrow, \searrow\}$ *conservation rule*

The value of a flow function f is given by $f(\nearrow, V)$. A *maximum flow* is a flow of maximum value. A *preflow* is a function $f : V \times V \to \mathbb{R}$ satisfying Conditions 1 and 2 above, and the relaxation of Condition 3:

(3′) $f(V, v) \geq 0$ for all $v \in V \setminus \{\nearrow\}$.

The *excess* $e(v)$ of a vertex v is defined by $f(V, v)$. A vertex $v \notin \{\nearrow, \searrow\}$ is called *active* if $e(v) > 0$. Observe that if in a preflow function no vertex v is active for $v \in V \setminus \{\nearrow, \searrow\}$, it is then also a flow function. A pair (v, w) is a *residual edge* of f if $f(v, w) < c(v, w)$. The set of residual edges with respect to f is denoted by E_f. The *residual capacity* $c_f(v, w)$ of the residual edge (v, w) is defined by $c(v, w) - f(v, w)$. If (v, w) is not a residual edge, it is called *saturated*. A *valid distance function* (called *valid labelling* in [20]) d is a function

$V \to \mathbb{N} \cup \{\infty\}$ satisfying: $d(\nearrow) = |V|$, $d(\searrow) = 0$ and $d(v) \leq d(w) + 1$ for every residual edge (v, w). A residual edge (v, w) is *admissible* if $d(v) = d(w) + 1$.

Related to maximum flows are minimum cuts. A *cut* of a network $\mathcal{N} = (V, E, c)$ is a partition of V into two disjoint sets (X, X') such that $\nearrow \in X$ and $\searrow \in X'$. The *capacity* of (X, X') is the sum of all capacities of edges from X to X', i.e., $\sum_{v \in X, w \in X'} c(v, w)$. A *minimum cut* is a cut with minimal capacity. The *Maximum Flow Minimum Cut Theorem* [1] states that the capacity of a minimum cut is equal to the value of a maximum flow.

The Preflow Algorithm. We initialise the preflow f by: $f(v, w) = c(v, w)$ if $v = \nearrow$ and 0 otherwise. The distance function d is initialised by: $d(v) = |V|$ if $v = \nearrow$ and 0 otherwise. The preflow algorithm preserves the validity of the preflow f and the distance function d. If there is an active vertex v such that the residual edge (v, w) is admissible, we *push* $\delta := \min\{e(v), c_f(v, w)\}$ amount of flow from v toward the sink along the admissible edge (v, w) by increasing $f(v, w)$ (and decreasing $f(w, v)$) by δ. The excesses of v and w are then modified accordingly by: $e(v) = e(v) - \delta$ and $e(w) = e(w) + \delta$. If v is active but there are no admissible edges leaving it, one may *relabel* v by letting $d(v) := \min\{d(w) + 1 \mid (v, w) \in E_f\}$. Pushing and relabelling are repeated until there are no active vertices left. The algorithm terminates if no such operations apply. The resulting final preflow f is a maximum flow.

Feasible Flow Problem. Let $A \subseteq E$ be a subset of edges of the network $\mathcal{N} = (V, E, c)$, and define the lower bound function $l : A \to \mathbb{R}_{>0}$ which satisfies $l(e) \leq c(e)$ for all $e \in A$. We address the *feasible flow* problem which consists of finding a flow function f satisfying the condition: $f(e) \geq l(e)$ for all $e \in A$. We briefly show that this problem can be reduced to the maximum flow problem [1].

We can replace a minimum flow requirement on edge $v \to w$ by turning v into a demanding vertex (i.e., a vertex that consumes part of its inflow) and turning w into a supplying vertex (i.e., a vertex that creates some outflow *ex nihilo*). The capacity of edge $v \to w$ is then reduced accordingly.

Now, we are going to look for a flow-like function for the updated network. The function should satisfy the capacity constraints, and the difference between outflow and inflow in each vertex corresponds to its supply or demand, except for $\nearrow$ and $\searrow$. To remove that last exception, we add an edge from $\searrow$ to $\nearrow$ with capacity ∞.

We then apply another transformation to the updated network so that we can apply the maximum flow algorithm. We add new source and target vertices $\nearrow'$ and $\searrow'$. For each supplying vertex s, we add an edge $\nearrow' \to s$ with the same capacity as the supply of the vertex. For each demanding vertex d, we add an edge $d \to \searrow'$ with the same capacity as the demand of the vertex. In [1] it is shown that the original network has a feasible flow if and only if the transformed network has a flow h that saturates all edges from $\nearrow'$ and all edges to $\searrow'$. The flow h necessarily is a maximum flow, and if there is an h, each maximum flow satisfies the requirement; therefore it can be found by the maximum flow algorithm. An example will be given in Example 5.4 in Section 5.

4. Algorithms for Deciding Strong Simulations

We first recall the basic algorithm to compute the largest strong simulation relation $\precsim$ in Subsection 4.1. Then, we refine this algorithm to deal with strong simulation on Markov chains in Subsection 4.2, and extend it to deal with probabilistic automata in Subsection 4.3.

$\text{SimRel}_s(\mathcal{M})$
1.1: $R_1 \leftarrow \{(s_1, s_2) \in S \times S \mid L(s_1) = L(s_2)\}$ and $i \leftarrow 0$
1.2: **repeat**
1.3: $i \leftarrow i + 1$
1.4: $R_{i+1} \leftarrow \emptyset$
1.5:   **for all** $(s_1, s_2) \in R_i$ **do**
1.6:   **if** $s_1 \precsim_{R_i} s_2$ **then**
1.7: $R_{i+1} \leftarrow R_{i+1} \cup \{(s_1, s_2)\}$
1.8: **until** $R_{i+1} = R_i$
1.9: **return** R_i

Algorithm 1: Basic algorithm to decide strong simulation.

In Subsection 4.4 we present an algorithm for deciding strong probabilistic simulation for probabilistic automata.

4.1. **Basic Algorithm to Decide Strong Simulation.** The algorithm in [3], copied as SimRel_s in Algorithm 1, takes as a parameter a model, which, for now, is an FPS $\mathcal{M}$. The subscript 's' stands for strong simulation; a very similar algorithm, namely SimRel_w, will be used for weak simulation later. To calculate the strong simulation relation for $\mathcal{M}$, the algorithm starts with the initial relation $R_1 = \{(s_1, s_2) \in S \times S \mid L(s_1) = L(s_2)\}$ which is coarser than $\precsim_{\mathcal{M}}$. In iteration i, it generates R_{i+1} from R_i by deleting each pair (s_1, s_2) from R_i if s_2 cannot strongly simulate s_1 up to R_i, i.e., $s_1 \not\precsim_{R_i} s_2$. This proceeds until there is no such pair left, i.e., $R_{i+1} = R_i$. Invariantly throughout the loop it holds that R_i is coarser than $\precsim_{\mathcal{M}}$ (i.e., $\precsim_{\mathcal{M}}$ is a sub-relation of R_i). We obtain the strong simulation preorder $\precsim_{\mathcal{M}} = R_i$, once the algorithm terminates.

The decisive part of the algorithm is the check in Line 1.6, i.e., whether $s_1 \precsim_{R_i} s_2$. This can be answered via solving a maximum flow problem on a particular network $\mathcal{N}(\mathbf{P}(s_1, \cdot),$ $\mathbf{P}(s_2, \cdot), R_i)$ constructed from $\mathbf{P}(s_1, \cdot)$, $\mathbf{P}(s_2, \cdot)$ and R_i. This network is the relevant part of a graph containing two copies $t \in S_\perp$ and $\overline{t} \in \overline{S_\perp}$ of each state where $\overline{S_\perp} = \{\overline{t} \mid t \in S_\perp\}$ as follows: Let $\nearrow$ (the source) and $\searrow$ (the sink) be two additional vertices not contained in $S_\perp \cup \overline{S_\perp}$. For $\mu, \mu' \in Dist(S)$, and a relation $R \subseteq S \times S$ we define the network $\mathcal{N}(\mu, \mu', R) = (V, E, c)$ with the set of vertices $V = \{\nearrow, \searrow\} \cup Supp_\perp(\mu) \cup \overline{Supp_\perp(\mu')}$ and the set of edges E is defined by $E = \{(s, \overline{t}) \mid (s, t) \in R_\perp\} \cup \{(\nearrow, s)\} \cup \{(\overline{t}, \searrow)\}$ where $s \in Supp_\perp(\mu)$ and $t \in Supp_\perp(\mu')$. Recall the relation $R_\perp$ is defined by $R \cup \{(\perp, s) \mid s \in S_\perp\}$. The capacity function c is defined as follows: $c(\nearrow, s) = \mu(s)$ for all $s \in Supp_\perp(\mu)$, $c(\overline{t}, \searrow) = \mu'(t)$ for all $t \in Supp_\perp(\mu')$, and $c(s, \overline{t}) = \infty$ for all other $(s, \overline{t}) \in E$. This network is a bipartite network, since the vertices can be partitioned into two subsets $V_1 := Supp_\perp(\mu) \cup \{\searrow\}$ and $V_2 := \overline{Supp_\perp(\mu')} \cup \{\nearrow\}$ such that all edges have one endpoint in V_1 and another in V_2. Later, we will use two variations of this network: For $\gamma \in \mathbb{R}_{>0}$, we let $\mathcal{N}(\mu, \gamma\mu', R)$ denote the network obtained from $\mathcal{N}(\mu, \mu', R)$ by setting the capacities to the sink $\searrow$ to: $c(\overline{t}, \searrow) = \gamma\mu'(t)$. For two states s_1, s_2 of an FPS or a CTMC, we let $\mathcal{N}(s_1, s_2, R)$ denote the network $\mathcal{N}(\mathbf{P}(s_1, \cdot), \mathbf{P}(s_2, \cdot), R)$.

The following lemma expresses the crucial relationship between maximum flows and weight functions on which the algorithm is based. It is a direct extension of [3, Lemma 5.1]:

Lemma 4.1. *Let S be a finite set of states and R be a relation on S. Let $\mu, \mu' \in Dist(S)$. Then, $\mu \sqsubseteq_R \mu'$ iff the maximum flow of the network $\mathcal{N}(\mu, \mu', R)$ has value 1.*

Proof. As we introduced the auxiliary state $\bot$, μ and μ' are stochastic distributions in $Dist(S_\bot)$. The rest of the proof follows directly from [3, Lemma 5.1]. □

Thus we can decide $s_1 \precsim_{R_i} s_2$ by computing the maximum flow in $\mathcal{N}(s_1, s_2, R_i)$ and then check whether it has value 1. We recall the correctness and complexity of SimRel_s which will also be used later.

Theorem 4.2 ([3]). *If $\text{SimRel}_s(\mathcal{M})$ terminates, the returned relation equals $\precsim_\mathcal{M}$. Moreover, $\text{SimRel}_s(\mathcal{M})$ runs in time $\mathcal{O}(n^7/\log n)$ and in space $\mathcal{O}(n^2)$.*

Proof. First we show that after the last iteration (say iteration k), it holds that $\precsim$ is coarser than R_k: It holds that $R_{k+1} = R_k$, thus for all $(s_1, s_2) \in R_k$, we have that $s_1 \precsim_{R_k} s_2$. As for all $(s_1, s_2) \in R_k \subseteq R_1$, we have $L(s_1) = L(s_2)$, R_k is a strong simulation relation by Definition 2.8, thus $\precsim$ is coarser than R_k.

Now we show by induction that the loop of the algorithm invariantly ensures that R_i is coarser than $\precsim$. Assume $i = 1$. By definition of strong simulation, $s_1 \precsim s_2$ implies $L(s_1) = L(s_2)$. Thus, the initial relation R_1 is coarser than the simulation relation $\precsim$. Now assume that R_i is coarser than $\precsim$ for some $1 \le i < k$; we will show that also R_{i+1} is coarser than $\precsim$. Pick a pair $(s_1, s_2) \in \precsim$ arbitrarily. By Definition 2.8, $\mathbf{P}(s_1, \cdot) \sqsubseteq_\precsim \mathbf{P}(s_2, \cdot)$, so there exists a weight function for $(\mathbf{P}(s_1, \cdot), \mathbf{P}(s_2, \cdot))$ with respect to $\precsim$. Inspection of Definition 2.7 shows that the same function is also a weight function with respect to any set coarser than $\precsim$. As R_i is coarser than $\precsim$ by induction hypothesis, we conclude that $\mathbf{P}(s_1, \cdot) \sqsubseteq_{R_i} \mathbf{P}(s_2, \cdot)$, and from Subsection 2.2.4, $s_1 \precsim_{R_i} s_2$. This implies that $(s_1, s_2) \in R_{i+1}$ by line 1.6 for all $s_1 \precsim s_2$. Therefore, R_{i+1} is coarser than $\precsim$ for all $i = 1 \dots, k$.

Now we show the complexity. For one network $\mathcal{N}(s_1, s_2, R_i) = (V, E, c)$, the sizes of the vertices $|V|$ and edges $|E|$ are bounded by $2n + 4$ and $(n + 1)^2 + 2n$, respectively. The number of edges meets the worst case bound $\mathcal{O}(n^2)$. To the best of our knowledge, the best complexity of the flow computation for the network G is $\mathcal{O}(|V|^3/\log|V|) = \mathcal{O}(n^3/\log n)$ [14, 19]. In the algorithm SimRel_s, only one pair, in the worst case, is removed from R_i in iteration i, which indicates that the test whether $s_1 \precsim_{R_i} s_2$ is called $|R_1|$ times, $|R_1| - 1$ times and so on. Altogether it is bounded by $\sum_{i=1}^{|R_1|} i \le \sum_{i=1}^{n^2} i \in \mathcal{O}(n^4)$. Hence, the overall time complexity amounts to $\mathcal{O}(n^7/\log n)$. The space complexity is $\mathcal{O}(n^2)$ because of the representation of the transitions in $\mathcal{N}(s_1, s_2, R_i)$. □

4.2. An Improved Algorithm for FPSs.

We first analyse the behaviour of SimRel_s in more detail. For this, we consider an arbitrary pair (s_1, s_2), and assume that (s_1, s_2) stays in relation $R_1, \dots, R_k$ throughout the iterations $i = 1, \dots, k$, until the pair is either found not to satisfy $s_1 \precsim_{R_k} s_2$ or the algorithm terminates with a fix-point after iteration k. Then altogether the maximum flow algorithms are run k-times for this pair. However, the networks $\mathcal{N}(s_1, s_2, R_i)$ constructed in successive iterations are very similar, and may often be identical across iterations: They differ from iteration to iteration only by deletion of some edges induced by the successive cleanup of R_i. For our particular pair (s_1, s_2) the network might not change at all in some iterations, because the deletions from R_i do not affect their direct successors. We are going to exploit this observation by an algorithm that reuses the already computed maximum flow, in a way that whatever happens is good: If no

$\textsc{Smf}(i, \mathcal{N}(s_1, s_2, R_{i-1}), f_{i-1}, d_{i-1}, D_{i-1})$

2.1: $\mathcal{N}(s_1, s_2, R_i) \leftarrow \mathcal{N}(s_1, s_2, R_{i-1} \setminus D_{i-1})$ and $f_i \leftarrow f_{i-1}$ and $d_i \leftarrow d_{i-1}$

2.2: **for all** $(u_1, u_2) \in D_{i-1}$ **do**

2.3: $\quad f_i(\overline{u_2}, \searrow) \leftarrow f_i(\overline{u_2}, \searrow) - f_i(u_1, \overline{u_2})$

2.4: $\quad f_i(u_1, \overline{u_2}) \leftarrow 0$

2.5: Apply the preflow algorithm to calculate the maximum flow for $\mathcal{N}(s_1, s_2, R_i)$, but initialise the preflow to f_i and the distance function to d_i.

2.6: **return** $(|f_i| = 1, \mathcal{N}(s_1, s_2, R_i), f_i, d_i)$

$\textsc{Smf}_{init}(i, s_1, s_2, R_i)$

2.11: Initialise the network $\mathcal{N}(s_1, s_2, R_i)$.

2.12: Apply the preflow algorithm to calculate the maximum flow for $\mathcal{N}(s_1, s_2, R_i)$.

2.13: **return** $(|f_i| = 1, \mathcal{N}(s_1, s_2, R_i), f_i, d_i)$

Algorithm 2: Algorithm for a sequence of maximum flows.

changes occur from $\mathcal{N}(s_1, s_2, R_{i-1})$ to $\mathcal{N}(s_1, s_2, R_i)$, then the maximum flow is the same as the one in the previous iteration. If changes do occur, the preflow algorithm can be applied to get the new maximum flow very fast, using the maximum flow and distance function constructed in the previous iteration as a starting point.

To understand the algorithm, we look at the network $\mathcal{N}(s_1, s_2, R_1)$. Let $D_1, \ldots, D_k$ be pairwise disjoint subsets of R_1, which correspond to the pairs deleted from R_1 in iteration i, so $R_{i+1} = R_i \setminus D_i$ for $1 \le i \le k$. Let $f_i^{(s_1, s_2)}$ denote the maximum flow of the network $\mathcal{N}(s_1, s_2, R_i)$ for $1 \le i \le k$. We sometimes omit the superscript (s_1, s_2) in the parameters if the pair (s_1, s_2) is clear from the context. We address the problem of checking $|f_i| = 1$ for all $i = 1, \ldots, k$. Our algorithm *sequence of maximum flows* $\textsc{Smf}(i, \mathcal{N}(s_1, s_2, R_{i-1}), f_{i-1}, d_{i-1}, D_{i-1})$ is shown as Algorithm 2. It executes iteration i of a parametric flow algorithm, where $\mathcal{N}(s_1, s_2, R_{i-1})$ is the network for (s_1, s_2) and f_{i-1} and d_{i-1} are the flow and the distance function resulting from the previous iteration $i-1$; and D_{i-1} is a set of edges that have to be deleted from $\mathcal{N}(s_1, s_2, R_{i-1})$ to get the current network. The algorithm returns a tuple, in which the first component is a boolean that tells whether $|f_i| = 1$; it also returns the new network $\mathcal{N}(s_1, s_2, R_i)$, flow f_i and distance function d_i to be reused in the next iteration. $\textsc{Smf}$ is inspired by the *parametric maximum algorithm* in [18]. A variant of $\textsc{Smf}$ is used in the first iteration, shown in lines 2.11–2.13.

This algorithm for sequence of maximum flow problems is called in an improved version of $\textsc{SimRel}_s$ shown as Algorithm 3. Lines 3.2–3.7 contain the first iteration, very similar to the first iteration of Algorithm 1 (lines 1.4–1.7). At line 3.4 we prepare for later iterations the set

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

where $pre(s) = \{t \in S \mid \mathbf{P}(t, s) > 0\}$. This set contains all pairs (u_1, u_2) such that the network $\mathcal{N}(u_1, u_2, R_1)$ contains the edge $(s_1, \overline{s_2})$. Iteration i (for $i > 1$) of the loop (lines 3.10–3.18) calculates R_{i+1} from R_i. In lines 3.11–3.14, we collect edges that should be removed from $\mathcal{N}(u_1, u_2, R_{i-1})$ in the sets $D_{i-1}^{(u_1, u_2)}$. At line 3.16, the algorithm $\textsc{Smf}$ constructs the maximum flow for parameters using information from iteration $i-1$. It uses the set $D_{i-1}^{(s_1, s_2)}$ to update the network $\mathcal{N}(s_1, s_2, R_{i-1})$, flow f_{i-1}, a distance function d_{i-1};

$\text{SIMREL}_s^{\text{FPS}}(\mathcal{M})$

3.1: $R_1 \leftarrow \{(s_1, s_2) \in S \times S \mid L(s_1) = L(s_2)\}$ and $i \leftarrow 1$

3.2: $R_2 \leftarrow \emptyset$

3.3: **for all** $(s_1, s_2) \in R_1$ **do**

3.4: $\quad Listener_{(s_1,s_2)} \leftarrow \{(u_1, u_2) \mid u_1 \in pre(s_1) \wedge u_2 \in pre(s_2) \wedge L(u_1) = L(u_2)\}$

3.5: $\quad (match, \mathcal{N}(s_1, s_2, R_1), f_1^{(s_1,s_2)}, d_1^{(s_1,s_2)}) \leftarrow \text{SMF}_{init}(1, s_1, s_2, R_1)$

3.6: $\quad$ **if** $match$ **then**

3.7: $\qquad R_2 \leftarrow R_2 \cup \{(s_1, s_2)\}$

3.8: **while** $R_{i+1} \neq R_i$ **do**

3.9: $\quad i \leftarrow i + 1$

3.10: $\quad R_{i+1} \leftarrow \emptyset$ and $D_{i-1} \leftarrow R_{i-1} \setminus R_i$

3.11: $\quad$ **for all** $(s_1, s_2) \in R_i$ **do**

3.12: $\qquad D_{i-1}^{(s_1,s_2)} \leftarrow \emptyset$

3.13: $\quad$ **for all** $(s_1, s_2) \in D_{i-1}, \quad (u_1, u_2) \in Listener_{(s_1,s_2)} \cap R_{i-1}$ **do**

3.14: $\qquad D_{i-1}^{(u_1,u_2)} \leftarrow D_{i-1}^{(u_1,u_2)} \cup \{(s_1, s_2)\}$

3.15: $\quad$ **for all** $(s_1, s_2) \in R_i$ **do**

3.16: $\qquad (match, \mathcal{N}(s_1, s_2, R_i), f_i^{(s_1,s_2)}, d_i^{(s_1,s_2)})$
$\qquad\qquad\qquad \leftarrow \text{SMF}(i, \mathcal{N}(s_1, s_2, R_{i-1}), f_{i-1}^{(s_1,s_2)}, d_{i-1}^{(s_1,s_2)}, D_{i-1}^{(s_1,s_2)})$

3.17: $\qquad$ **if** $match$ **then**

3.18: $\qquad\quad R_{i+1} \leftarrow R_{i+1} \cup \{(s_1, s_2)\}$

3.19: **return** R_i

Algorithm 3: Improved algorithm for deciding strong simulation for FPSs.

then it constructs the maximum flow f_i for the network $\mathcal{N}(s_1, s_2, R_i)$. If SMF returns true, (s_1, s_2) is inserted into R_{i+1} and survives this iteration (line 3.18).

Consider the algorithm SMF and assume that $i > 1$. At lines 2.1–2.4, we remove the edges D_{i-1} from the network $\mathcal{N}(s_1, s_2, R_{i-1})$ and generate the preflow f_i based on the flow f_{i-1}, which is the maximum flow of the network $\mathcal{N}(s_1, s_2, R_{i-1})$, by

- setting $f_i(u_1, \overline{u_2}) = 0$ for all deleted edges $(u_1, u_2) \in D_{i-1}$, and
- reducing $f_i(\overline{u_2}, \searrow)$ such that the preflow f_i becomes consistent with the (relaxed) flow conservation rule.

The excess $e(v)$ is increased if there exists $(v, w) \in D_{i-1}$ such that $f_{i-1}(v, w) > 0$, and unchanged otherwise. Hence, f_i after line 2.4 is a preflow. The distance function $d_{i-1} = d_i$ is still valid for this preflow, since removing the set of edges D_{i-1} does not introduce new residual edges. This guarantees that, at line 2.5, the *preflow algorithm* finds a maximum flow over the network $\mathcal{N}(s_1, s_2, R_i)$. In line 2.6, SMF returns whether the flow has value 1 together with information to be reused in the next iteration. (If $|f_k| < 1$ at some iteration k, then $|f_j| < 1$ for all iterations $j \geq k$ because deleting edges does not increase the maximum flow. In that case, it would be sufficient to return **false**.) We prove the correctness and complexity of the algorithm SMF:

Lemma 4.3. *Let* $(s_1, s_2) \in R_1$. *Then,* SMF_{init} *returns true iff* $s_1 \precsim_{R_1} s_2$. *For some* $i > 1$, *let* $\mathcal{N}(s_1, s_2, R_{i-1})$, f_{i-1}, *and* d_{i-1} *be as returned by some earlier call to* SMF *or* SMF_{init}. *Let* $D_{i-1} = (R_{i-1} \setminus R_i) \cap (post(s_1) \times post(s_2))$ *be the set of edges that will be removed from*

the network $\mathcal{N}(s_1, s_2, R_{i-1})$ *during the* $(i-1)$*th call of* SMF*. Then, the* $(i-1)$*th call of* SMF *returns true iff* $s_1 \precsim_{R_i} s_2$*.*

Proof. By Lemma 4.1, SMF$_{init}$ returns true iff $|f_1| = 1$, which is equivalent to $s_1 \precsim_{R_1} s_2$. Let $i > 1$. As discussed, at the beginning of line 2.5, the function f_{i-1} is a flow (thus a preflow) with value 1, and the distance function d_{i-1} is a valid distance function. It follows directly from the correctness of the preflow algorithm [2] that after line 2.5, f_i is a maximum flow for $\mathcal{N}(s_1, s_2, R_i)$. Thus, SMF returns true (i.e. $|f_i| = 1$) which is equivalent to $s_1 \precsim_{R_i} s_2$. $\square$

Lemma 4.4. *Consider the pair* (s_1, s_2) *and assume that* $|post(s_1)| \leq |post(s_2)|$*. All calls to* SMF$(i, \mathcal{N}(s_1, s_2, \cdot), \cdots)$ *related to* (s_1, s_2) *together run in time* $\mathcal{O}(|post(s_1)| \, |post(s_2)|^2)$*.*

Proof. In the bipartite network $\mathcal{N}(s_1, s_2, R_1)$, the set of vertices are partitioned into subsets $V_1 = post(s_1) \cup \{\searrow\}$ and $V_2 = post(s_2) \cup \{\nearrow\}$ as described in Section 4.1. Generating the initial network (line 2.11) takes time in $\mathcal{O}(|V_1| |V_2|)$. In our sequence of maximum flow problems, the number of (nontrivial) iterations, denoted by k, is bounded by the number of edges, i.e., $k \leq |E| \leq |V_1| |V_2| - 1$. We split the work being done by all calls to SMF$(i, \mathcal{N}(s_1, s_2, \cdot), \dots)$ together with the initial call to the preflow algorithm (line 2.12 and line 2.5) into edge deletions, relabels, non-saturating pushes, saturating pushes. (A non-saturating push along an edge (u, v) moves all excess at u to v; by such a push, the number of active nodes never increases.)

All edge deletions take time proportional to $\sum_{i=1}^{k} |D_i|$, which is less than the number of edges in the network. Therefore, edge deletions take time $\mathcal{O}(|V_1||V_2|)$. For all $v \in V$, it holds that $d_{i+1}(v) = d_i(v)$, i.e., the labelling function at the beginning of iteration $i + 1$ is the same as the labelling function at the end of iteration i.

We discuss the time for relabelling and saturating pushes [2]. For a bipartite network, the distance of the source can be initialised to $d(\nearrow) = 2|V_1|$ instead of $|V|$, and $d(v)$ never grows above $4|V_1|$ for all $v \in V$. For $v \in V$, let $I(v)$ denote the set of nodes containing w such that either $(v, w) \in E$ or $(w, v) \in E$. Intuitively, it represents edges which could be admissible leaving v. The time for relabel operations with respect to node v is thus $(4|V_1|)|I(v)|$. Altogether, this gives the time for all relabel operations: $\sum_{v \in V}((4|V_1|)|I(v)|) \in \mathcal{O}(|V_1||E|)$. Between two consecutive saturating pushes on (v, w), the distances $d(v)$ and $d(w)$ must increase by 2. Thus, the number of saturating pushes on edge (v, w) is bounded by $4|V_1|$. Summing over all edges, the work for saturating pushes is bounded by $\mathcal{O}(|V_1||E|)$.

Now we discuss the analysis of the number of non-saturating pushes, which is very similar to the proof of Theorem 2.2 in [22] where Max-d version of the algorithm is used. Assume that in iteration $l \leq k$ of SMF, the last relabelling action occurs. In the Max-d version [22], always the active node with the highest label is selected, and once an active node is selected, the excess of this node is pushed until it becomes 0. This implies that, between any two relabel operations, there are at most n active nodes processed (otherwise the algorithm terminates and we get the maximum flow). Also observe that at each time an active node is selected, at most one non-saturating push can occur, which implies that there are at most n non-saturating pushes between node label increases. Since $d_i(v)$ is bounded by $4|V_1|$, the number of relabels altogether is bounded by $\mathcal{O}(|V_1||V|)$. Thus, the number of non-saturating pushes before the iteration l is bounded by $\mathcal{O}(|V_1||V|^2)$. Since the distance function does not change after iteration l any more, inside any of the iterations $l' \geq l$, there are again at most $n - 1$ non-saturating pushes. Hence, the number of non-saturating pushes is bounded by $|V_1||V|^2 + (k + 1 - l)(|V| - 1) \in \mathcal{O}(|V_1||V|^2 + k|V|)$.

Since $k \leq |V_1||V_2| - 1$, and $|V| \leq 2|V_2|$, thus, the overall time complexity amounts to $\mathcal{O}(|V_1||V_2|^2) = \mathcal{O}(|post(s_1)|\,|post(s_2)|^2)$ as required. $\qquad\square$

Now we give the correctness and complexity of the algorithm SimRel for FPSs:

Theorem 4.5. *If* $\mathrm{SimRel}_s^{\mathrm{FPS}}(\mathcal{M})$ *terminates, the returned relation equals* $\precsim_{\mathcal{M}}$.

Proof. By Lemma 4.3, $\mathrm{SMF}_{init}(i, s_1, s_2, R_1)$ returns true in iteration $i = 1$ iff $s_1 \precsim_{R_1} s_2$; $\mathrm{SMF}(i, \mathcal{N}(s_1, s_2, R_{i-1}), \dots)$ returns true in iteration $i > 1$ iff $s_1 \precsim_{R_i} s_2$. The rest of the correctness proof is the same as the proof of Theorem 4.2. $\qquad\square$

Theorem 4.6. *The algorithm* $\mathrm{SimRel}_s^{\mathrm{FPS}}(\mathcal{M})$ *runs in time* $\mathcal{O}(m^2 n)$ *and in space* $\mathcal{O}(m^2)$. *If the fanout is bounded by a constant, it has complexity* $\mathcal{O}(n^2)$, *both in time and space.*

Proof. We first show the space complexity. In most cases, it is enough to store information from the previous iteration until the corresponding structure for the current iteration is calculated. The size of the set $Listener_{(s_1, s_2)}$ is bounded by $|pre(s_1)|\,|pre(s_2)|$ where $pre(s) = \{t \in S \mid \mathbf{P}(t, s) > 0\}$. Summing over all (s_1, s_2), we get $\sum_{s_1 \in S} \sum_{s_2 \in S} |pre(s_1)|\,|pre(s_2)| = m^2$. Assume we run iteration i. For every pair (s_1, s_2), we generate the set $D_{i-1}^{(s_1, s_2)}$ and the network $\mathcal{N}(s_1, s_2, R_i)$ together with f_i and d_i. Obviously, the size of $D_{i-1}^{(s_1, s_2)}$ is bounded by $|post(s_1)|\,|post(s_2)|$. Summing over all (s_1, s_2), we get the bound $\mathcal{O}(m^2)$. The number of edges of the network $\mathcal{N}(s_1, s_2, R_1)$ (together with f_i and d_i) is in $\mathcal{O}(|post(s_1)|\,|post(s_2)|)$. Summing over all (s_1, s_2) yields a memory consumption in $\mathcal{O}(m^2)$ again. Hence, the overall space complexity is $\mathcal{O}(m^2)$.

Now we show the time complexity. We observe that a pair (s_1, s_2) belongs to D_i in at most one iteration. Therefore, the time needed in lines 3.11–3.14 in all iterations together is bounded by the size of all sets $Listener_{(s_1, s_2)}$, which is $\mathcal{O}(m^2)$. We analyse the time needed for all calls to the algorithm SMF. Recall that the fanout g equals $\max_{s \in S} |post(s)|$, and therefore $|post(s_i)| \leq g$ for $i = 1, 2$. By Lemma 4.4, the complexity attributed to the pair (s_1, s_2) is bounded by $\mathcal{O}(g\,|post(s_1)|\,|post(s_2)|)$. Taking the sum over all possible pairs, we get the bound $gm^2 \in \mathcal{O}(m^2 n)$. If g is bounded by a constant, we have $m \leq gn$, and the time complexity is $gm^2 \leq g^3 n^2 \in \mathcal{O}(n^2)$. In this case the space complexity is also $\mathcal{O}(n^2)$. $\square$

Strong Simulation for Markov Chains. We now consider DTMCs and CTMCs. Since each DTMC is a special case of an FPS the algorithm $\mathrm{SimRel}_s^{\mathrm{FPS}}$ applies directly.

Let $\mathcal{M} = (S, \mathbf{R}, L)$ be a CTMC. Recall that $s \precsim_{\mathcal{M}} s'$ holds if $s \precsim_{emb(\mathcal{M})} s'$ in the embedded DTMC, and s' is faster than s. We can ensure the additional rate condition by incorporating it into the initial relation R. More precisely, initially R contains only those pair (s, s') such that $L(s) = L(s')$, and that the state s' is faster than s, i.e., we replace line 3.1 of the algorithm by

$$R_1 \leftarrow \{(s_1, s_2) \in S \times S \mid L(s_1) = L(s_2) \wedge \mathbf{R}(s_1, S) \leq \mathbf{R}(s_2, S)\}$$

to ensure the additional rate condition of Definition 2.10. In the refinement steps afterwards, only the weight function conditions need to be checked with respect to the current relation in the embedded DTMC. Thus, we arrive at an algorithm for CTMCs with the same time and space complexity as for FPSs.

Example 4.7. Consider the CTMC in the left part of Figure 7 (it has 10 states). Consider the pair $(s_1, s_2) \in R_1$. The network $\mathcal{N}(s_1, s_2, R_1)$ is depicted on the right of the figure.
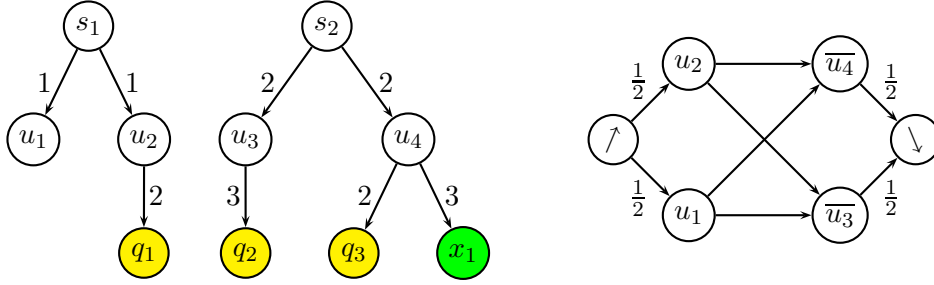
Figure 7: A CTMC example and its network $\mathcal{N}(s_1, s_2, R_1)$.

Assume that we get the maximum flow f_1 which sends $\frac{1}{2}$ amount of flow along the path $\nearrow, u_2, \overline{u_4}, \searrow$ and $\frac{1}{2}$ amount of flow along $\nearrow, u_1, \overline{u_3}, \searrow$. Hence, the check for (s_1, s_2) is successful in the first iteration. The checks for the pairs (u_1, u_3), (u_1, u_4) and (u_2, u_3) are also successful in the first iteration. However, the check for the pair (u_2, u_4) fails, as the probability to go from u_4 to q_3 in the embedded DTMC is $\frac{2}{5}$, while the probability to go from u_2 to q_1 in the embedded DTMC is 1.

In the second iteration, the network $\mathcal{N}(s_1, s_2, R_2)$ is obtained from $\mathcal{N}(s_1, s_2, R_1)$ by deleting the edge $(u_2, \overline{u_4})$. In $\mathcal{N}(s_1, s_2, R_2)$, the flows on $(u_2, \overline{u_4})$ and on $(\overline{u_4}, \searrow)$ are set to 0, and the vertex u_2 has a positive excess $\frac{1}{2}$. Applying the preflow algorithm, we push the excess from u_2, along $\overline{u_3}, u_1, \overline{u_4}$ to $\searrow$. We get a maximum flow f_2 for $\mathcal{N}(s_1, s_2, R_2)$ which sends $\frac{1}{2}$ amount of flow along the path $\nearrow, u_2, \overline{u_3}, \searrow$ and $\frac{1}{2}$ amount of flow along $\nearrow, u_1, \overline{u_4}, \searrow$. Hence, the check for (s_1, s_2) is also successful in the second iteration. Once the fix-point is reached, R still contains (s_1, s_2).

4.3. **Strong Simulation for Probabilistic Automata.** In this subsection we present algorithms for deciding strong simulations for PAs and CPAs. It takes the skeleton of the algorithm for FPSs: it starts with a relation R which is coarser than $\precsim$, and then refines R until $\precsim$ is achieved. In the refinement loop, a pair (s, s') is eliminated from the relation R if the corresponding strong simulation conditions are violated with respect to the current relation. For PAs, this means that there exists an α-successor distribution μ of s, such that for all α-successor distribution μ' of s', we cannot find a weight function for (μ, μ') with respect to the current relation R.

Let $\mathcal{M} = (S, Act, \mathbf{P}, L)$ be a PA. We aim to extend Algorithm 3 to determine the strong simulation on PAs. For a pair (s_1, s_2), assume that $L(s_1) = L(s_2)$ and that $Act(s_1) \subseteq Act(s_2)$, which is guaranteed by the initialisation. We consider line 3.17, which checks the condition $\mathbf{P}(s_1, \cdot) \sqsubseteq_{R_i} \mathbf{P}(s_2, \cdot)$ using SMF. By Definition 2.11 of strong simulation for PAs, we should instead check the condition

$$\forall \alpha \in Act. \ \forall s_1 \xrightarrow{\alpha} \mu_1. \ \exists s_2 \xrightarrow{\alpha} \mu_2 \text{ with } \mu_1 \sqsubseteq_{R_i} \mu_2 \qquad (4.1)$$

Recall the condition $\mu_1 \sqsubseteq_{R_i} \mu_2$ is true iff the maximum flow of the network $\mathcal{N}(\mu_1, \mu_2, R_i)$ has value one. Sometimes, we write $\mathcal{N}(s_1, \alpha, \mu_1, s_2, \mu_2, R_i)$ to denote the network $\mathcal{N}(\mu_1, \mu_2, R_i)$ associated with the pair (s_1, s_2) with respect to action α.

Our first goal is to extend SMF to check Condition 4.1 for a fixed action α and α-successor distribution μ_1 of s_1. To this end, we introduce a list $Sim^{(s_1, \alpha, \mu_1, s_2)}$ that contains all potential candidates of α-successor distributions of s_2 which could be used to establish

$\textsc{ActSmf}(i, Sim_{i-1}, \mathcal{N}(\mu_1, \mu_2, R_{i-1}), f_{i-1}, d_{i-1}, D_{i-1})$

4.1: $Sim_i \leftarrow Sim_{i-1}$
4.2: $(match, \mathcal{N}(\mu_1, \mu_2, R_i), f_i, d_i) \leftarrow \textsc{Smf}(i, \mathcal{N}(\mu_1, \mu_2, R_{i-1}), f_{i-1}, d_{i-1}, D_{i-1})$
4.3: **if** $match$ **then**
4.4:   **return** $(\mathbf{true}, Sim_i, \mathcal{N}(\mu_1, \mu_2, R_i), f_i, d_i)$
4.5: $Sim_i \leftarrow \mathbf{tail}(Sim_i)$
4.6: **while** $\neg\mathbf{empty}(Sim_i)$ **do**
4.7: $\mu_2 \leftarrow \mathbf{head}(Sim_i)$
4.8: $(match, \mathcal{N}(\mu_1, \mu_2, R_i), f_i, d_i) \leftarrow \textsc{Smf}_{init}(i, \mu_1, \mu_2, R_i)$
4.9:   **if** $match$ **then**
4.10:   **return** $(\mathbf{true}, Sim_i, \mathcal{N}(\mu_1, \mu_2, R_i), f_i, d_i)$
4.11: $Sim_i \leftarrow \mathbf{tail}(Sim_i)$
4.12: **return** $(\mathbf{false}, \emptyset, NIL, NIL, NIL)$

$\textsc{ActSmf}_{init}(i, \mu_1, Sim_i, R_i)$
          goto line 4.6

Algorithm 4: Subroutine to calculate whether $s_1 \precsim_{R_i} s_2$, as far as $s_1 \xrightarrow{\alpha} \mu_1$ is concerned. The parameter Sim denotes the subsets of α-successor distributions of s_2 serving as candidates for possible μ_2.

the condition $\mu_1 \sqsubseteq_R \mu_2$ for the relation R considered. The set $Sim^{(s_1, \alpha, \mu_1, s_2)}$ is represented as a list. This and some subsequent notations are similar to those used by Baier *et al.* in [3]. We use the function $\mathbf{head}(\cdot)$ to read the first element of a list; $\mathbf{tail}(\cdot)$ to read all but the first element of a list; and $\mathbf{empty}(\cdot)$ to check whether a list is empty. As long as the network for a fixed candidate $\mu_2 = \mathbf{head}(Sim^{(s_1, \alpha, \mu_1, s_2)})$ allows a flow of value 1 over the iterations, we stick to it, and we can reuse the flow and distance function from previous iterations. If by deleting some edges from $\mathcal{N}(\mu_1, \mu_2, R)$, its flow value falls below 1, we delete μ_2 from $Sim^{(s_1, \alpha, \mu_1, s_2)}$ and pick the next candidate.

     The algorithm $\textsc{ActSmf}$, shown as Algorithm 4, implements this. It has to be called for each pair (s_1, s_2) and each successor distribution $s_1 \xrightarrow{\alpha} \mu_1$ of s_1. It takes as input the list of remaining candidates $Sim_{i-1}^{(s_1, \alpha, \mu_1, s_2)}$, the information from the previous iteration (the network $\mathcal{N}(\mu_1, \mu_2, R_{i-1})$, flow f_{i-1}, and distance function d_{i-1}), and the set of edges that have to be deleted from the old network D_{i-1}.

Lemma 4.8. *Let* $(s_1, s_2) \in R_1$, $\alpha \in Act(s_1)$, *and* μ_1 *such that* $s_1 \xrightarrow{\alpha} \mu_1$. *Let* $Sim_1 = Steps_\alpha(s_2)$. *Then* $\textsc{ActSmf}_{init}$ *returns true iff* $\exists \mu_2$ *with* $s_2 \xrightarrow{\alpha} \mu_2 \wedge \mu_1 \sqsubseteq_{R_1} \mu_2$. *For some* $i > 1$, *let* Sim_{i-1}, $\mathcal{N}(\mu_1, \mu_2, R_{i-1})$, f_{i-1} *and* d_{i-1} *be as returned by some earlier call to* $\textsc{ActSmf}$ *or* $\textsc{ActSmf}_{init}$. *Let* $D_{i-1} = (R_{i-1} \setminus R_i) \cap (Supp(\mu_1) \times Supp(\mu_2))$ *be the set of edges that will be removed from the network during the* $(i-1)th$ *call of* $\textsc{ActSmf}$. *Then, the* $(i-1)th$ *call of algorithm* $\textsc{ActSmf}$ *returns true iff:* $\exists \mu_2$ *with* $s_2 \xrightarrow{\alpha} \mu_2 \wedge \mu_1 \sqsubseteq_{R_i} \mu_2$.

Proof. Once $\textsc{Smf}$ returns false because the maximum flow for the current candidate μ_2 has value < 1, it will never become a candidate again, as edge deletions cannot lead to increased flow. The correctness proof is then the same as the proof of Lemma 4.3. $\square$

$\text{SIMREL}_s^{\text{PA}}(\mathcal{M})$

5.1: $R_1 \leftarrow \{(s_1, s_2) \in S \times S \mid L(s_1) = L(s_2) \wedge Act(s_1) \subseteq Act(s_2)\}$ and $i \leftarrow 1$

5.2: $R_2 \leftarrow \emptyset$

5.3: **for all** $(s_1, s_2) \in R_1$ **do**

5.4: $\qquad Listener_{(s_1, s_2)} \leftarrow \{(u_1, \alpha, \mu_1, u_2, \mu_2) \mid L(u_1) = L(u_2) \wedge u_1 \xrightarrow{\alpha} \mu_1 \wedge u_2 \xrightarrow{\alpha} \mu_2$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge \mu_1(s_1) > 0 \wedge \mu_2(s_2) > 0\}$

5.5: $\qquad$ **for all** $\alpha \in Act(s_1), \quad \mu_1 \in Steps_\alpha(s_1)$ **do**

5.6: $\qquad\qquad Sim_1^{(s_1, \alpha, \mu_1, s_2)} \leftarrow Steps_\alpha(s_2)$

5.7: $\qquad\qquad (match_{\alpha, \mu_1}, Sim_1^{(s_1, \alpha, \mu_1, s_2)}, \mathcal{N}(s_1, \alpha, \mu_1, s_2, \mu_2, R_1), f_1^{arc}, d_1^{arc})$
$\qquad\qquad\qquad\qquad \leftarrow \text{ACTSMF}_{init}(1, \mu_1, Sim_1^{(s_1, \alpha, \mu_1, s_2)}, R_1)$

5.8: $\qquad$ **if** $\bigwedge_{\alpha \in Act(s_1)} \bigwedge_{\mu_1 \in Steps_\alpha(s_1)} match_{\alpha, \mu_1}$ **then**

5.9: $\qquad\qquad R_2 \leftarrow R_2 \cup \{(s_1, s_2)\}$

5.10: **while** $R_{i+1} \neq R_i$ **do**

5.11: $\qquad i \leftarrow i + 1$

5.12: $\qquad R_{i+1} \leftarrow \emptyset$ and $D_{i-1} \leftarrow R_{i-1} \setminus R_i$

5.13: $\qquad$ **for all** $(s_1, s_2) \in R, \quad \alpha \in Act(s_1), \quad \mu_1 \in Steps_\alpha(s_1), \quad \mu_2 \in Steps_\alpha(s_2)$ **do**

5.14: $\qquad\qquad D_{i-1}^{(s_1, \alpha, \mu_1, s_2, \mu_2)} \leftarrow \emptyset$

5.15: $\qquad$ **for all** $(s_1, s_2) \in D_{i-1}, \quad (u_1, \alpha, \mu_1, u_2, \mu_2) \in Listener_{(s_1, s_2)}$ **do**

5.16: $\qquad\qquad$ **if** $(u_1, u_2) \in R_{i-1}$ **then**

5.17: $\qquad\qquad\qquad D_{i-1}^{(u_1, \alpha, \mu_1, u_2, \mu_2)} \leftarrow D_{i-1}^{(u_1, \alpha, \mu_1, u_2, \mu_2)} \cup \{(s_1, s_2)\}$

5.18: $\qquad$ **for all** $(s_1, s_2) \in R$ **do**

5.19: $\qquad\qquad$ **for all** $\alpha \in Act(s_1), \quad \mu_1 \in Steps_\alpha(s_1)$ **do**

5.20: $\qquad\qquad\qquad (match_{\alpha, \mu_1}, Sim_i^{(s_1, \alpha, \mu_1, s_2)}, \mathcal{N}(s_1, \alpha, \mu_1, s_2, \mu_2, R_i), f_i^{arc}, d_i^{arc})$
$\qquad\qquad\qquad\qquad \leftarrow \text{ACTSMF}(i, Sim_{i-1}^{(s_1, \alpha, \mu_1, s_2)}, \mathcal{N}(s_1, \alpha, \mu_1, s_2, \mu_2, R_{i-1}),$
$\qquad\qquad\qquad\qquad\qquad\qquad f_{i-1}^{arc}, d_{i-1}^{arc}, D_{i-1}^{arc})$

5.21: $\qquad\qquad$ **if** $\bigwedge_{\alpha \in Act(s_1)} \bigwedge_{\mu_1 \in Steps_\alpha(s_1)} match_{\alpha, \mu_1}$ **then**

5.22: $\qquad\qquad\qquad R_{i+1} \leftarrow R_{i+1} \cup \{(s_1, s_2)\}$

5.23: **return** R_i

Algorithm 5: Algorithm for deciding strong simulation for PAs, where arc denotes the associated parameter $(s_1, \alpha, \mu_1, s_2, \mu_2)$.

The algorithm $\text{SIMREL}_s^{\text{PA}}$ for deciding strong simulation for PAs is presented as Algorithm 5. During the initialisation (lines 5.1–5.6, intermixed with iteration 1 in lines 5.7–5.9), for $(s_1, s_2) \in R_1$ and $s_1 \xrightarrow{\alpha} \mu_1$, the list $Sim_1^{(s_1, \alpha, \mu_1, s_2)}$ is initialised to $Steps_\alpha(s_2)$ (line 5.6), as no α-successor distribution can be excluded as a candidate a priori. As in $\text{SIMREL}_s^{\text{FPS}}$, the set $Listener_{(s_1, s_2)}$ for (s_1, s_2) is introduced which contains tuples $(u_1, \alpha, \mu_1, u_2, \mu_2)$ such that the network $\mathcal{N}(u_1, \alpha, \mu_1, u_2, \mu_2, R_1)$ contains the edge $(s_1, \overline{s_2})$.

The main iteration starts with generating the sets $D_{i-1}^{(u_1, \alpha, \mu_1, u_2, \mu_2)}$ in lines 5.13–5.17 in a similar way as $\text{SIMREL}_s^{\text{FPS}}$. Lines 5.19–5.22 check Condition 4.1 by calling ACTSMF for each action α and each α-successor distribution μ_1 of s_1. The condition is true if and only if $match_{\alpha, \mu_1}$ is true for all $\alpha \in Act(s_1)$ and $\mu_1 \in Steps_\alpha(s_1)$. In this case we insert the pair (s_1, s_2) into R_{i+1} (line 5.22). We give the correctness of the algorithm:

Theorem 4.9. *When* $\text{SIMREL}_s^{\text{PA}}(\mathcal{M})$ *terminates, the returned relation equals* $\precsim_{\mathcal{M}}$.

Proof. The proof follows the same lines as the proof of the correctness of $\textsc{SimRel}_s^{\text{FPS}}$ in Theorem 4.5. The only new element is that we now have to quantify over the actions and successor distributions as prescribed by Definition 2.11. This translates to a conjunction in lines 5.8 and 5.21 of the algorithm. Exploiting Lemma 4.8 we get the correctness. $\square$

Now we give the complexity of the algorithm:

Theorem 4.10. *The algorithm $\textsc{SimRel}_s^{\text{PA}}(\mathcal{M})$ runs in time $\mathcal{O}(m^2 n)$ and in space $\mathcal{O}(m^2)$. If the fanout of $\mathcal{M}$ is bounded by a constant, it has complexity $\mathcal{O}(n^2)$, both in time and space.*

Proof. We first consider the space complexity. We save the sets $D_i^{(s_1,\alpha,\mu_1,s_2,\mu_2)}$, the networks $\mathcal{N}(s_1,\alpha,\mu_1,s_2,\mu_2,R_i)$ which are updated in every iteration, $Listener_{(s_1,s_2)}$ and the sets $Sim_i^{(s_1,\alpha,\mu_1,s_2)}$. The size of the set $D_i^{(s_1,\alpha,\mu_1,s_2,\mu_2)}$ is in $\mathcal{O}(|\mu_1||\mu_2|)$, which is the maximal number of edges of $\mathcal{N}(s_1,\alpha,\mu_1,s_2,\mu_2,R_i)$. Summing over all $(s_1,\alpha,\mu_1,s_2,\mu_2)$, we get:

$$\sum_{s_1 \in S} \sum_{\alpha \in Act(s_1)} \sum_{\mu_1 \in Steps_\alpha(s_1)} \sum_{s_2 \in S} \sum_{\mu_2 \in Steps_\alpha(s_2)} |\mu_1||\mu_2| \leq m^2 \qquad (4.2)$$

Similarly, the memory needed for saving the networks has the same bound $\mathcal{O}(m^2)$. Now we consider the set $Listener_{(s_1,s_2)}$ for the pair $(s_1,s_2) \in R_1$. Let $(u_1,\alpha,\mu_1,u_2,\mu_2) \in Listener_{(s_1,s_2)}$. Then, it holds that $s_1 \in Supp(\mu_1)$ and $s_2 \in Supp(\mu_2)$. Hence, the tuple $(u_1,\alpha,\mu_1,u_2,\mu_2)$ can be an element of $Listener_{(s_1,s_2)}$ of some arbitrary pair (s_1,s_2) at most $|\mu_1||\mu_2|$ times, which corresponds to the maximal number of edges between the set of nodes $Supp(\mu_1)$ and $\overline{Supp(\mu_2)}$ in $\mathcal{N}(s_1,\alpha,\mu_1,s_2,\mu_2,R_1)$. Summing over all $(s_1,\alpha,\mu_1,s_2,\mu_2)$, we get that memory needed for the set $Listener$ is also bounded by $\mathcal{O}(m^2)$. For each pair (s_1,s_2) and $s_1 \xrightarrow{\alpha} \mu_1$, the set $Sim_1^{(s_1,\alpha,\mu_1,s_2)}$ has size $|Steps_\alpha(s_2)|$. Summing up, this is smaller than or equal to m^2 according to Inequality 4.2. Hence, the overall space complexity amounts to $\mathcal{O}(m^2)$.

Now we consider the time complexity. All initialisations (lines 5.1–5.6 of $\textsc{SimRel}_s^{\text{PA}}$ and the initialisations in $\textsc{ActSmf}_{init}$, which calls $\textsc{Smf}_{init}$) take $\mathcal{O}(m^2)$ time. We observe that a pair (s_1,s_2) belongs to D_i during at most one iteration. Because of the Inequality 4.2, the time needed in lines 5.13–5.17 is in $\mathcal{O}(m^2)$. The rest of the algorithm is dominated by the time needed for calling $\textsc{Smf}$ in line 4.2 of $\textsc{ActSmf}$. By Lemma 4.4, the time complexity for successful and unsuccessful checks concerning the tuple $(s_1,\alpha,\mu_1,s_2,\mu_2)$ is bounded by $\mathcal{O}(g|\mu_1||\mu_2|)$. Taking the sum over all possible tuples $(s_1,\alpha,\mu_1,s_2,\mu_2)$ we get the bound gm^2 according to Inequality 4.2. Hence, the complexity is $\mathcal{O}(m^2 n)$. If the fanout g is bounded by a constant, we have $m \leq gn$. Thus, the time complexity is in the order of $\mathcal{O}(n^2)$. In this case the space complexity is also $\mathcal{O}(n^2)$. $\square$

Remark 4.11. Let $\mathcal{M} = (S, Act, \mathbf{P}, L)$ be a PA, and let $k = \sum_{s \in S} \sum_{\alpha \in Act(s)} |Steps_\alpha(s)|$, called the number of transitions in [3], denote the number of all distributions in $\mathcal{M}$. The algorithm for deciding strong simulation introduced by Baier *et al.* has time complexity $\mathcal{O}((kn^6 + k^2 n^3)/\log n)$, and space complexity $\mathcal{O}(k^2)$. The number of distributions k and the size of transitions m are related by $k \leq m \leq nk$. The left equality is established if $|\mu| = 1$ for all distributions, and the right equality is established if $|\mu| = n$ for all distributions.

The decision algorithm for strong simulation for CPAs can be adapted from $\textsc{SimRel}_s^{\text{PA}}$ in Algorithm 5 easily: Notations are extended with respect to rate functions instead of

distributions in an obvious way. To guarantee the additional rate condition, we rule out successor rate functions of s_2 that violate it by replacing line 5.6 by:

$$Sim_1^{(s_1,\alpha,r_1,s_2)} \leftarrow \{r_2 \in Steps_\alpha(s_2) \mid r_1(S) \le r_2(S)\}.$$

For each pair (s_1, s_2), and successor rate functions $r_i \in Steps_\alpha(s_i)$ $(i = 1, 2)$, the subroutine for checking whether $r_1 \sqsubseteq_{R_i} r_2$ is then performed in the network $\mathcal{N}(\mu(r_1), \mu(r_2), R_i)$. Obviously, the so obtained algorithm for CPAs has the same complexity $\mathcal{O}(m^2 n)$.

4.4. **Strong Probabilistic Simulation.** The problem of deciding strong probabilistic simulation for PAs has not been tackled yet. We show that it can be computed by solving LP problems which are decidable in polynomial time [27]. In Subsection 4.4.1, we first present an algorithm for PAs. We extend the algorithm to deal with CPAs in Subsection 4.4.2.

4.4.1. *Probabilistic Automata.* Recall that strong probabilistic simulation is a relaxation of strong simulation in the sense that it allows combined transitions, which are convex combinations of multiple distributions belonging to equally labelled transitions. Again, the most important part is to check whether $s_1 \precsim_R^p s_2$ where R is the current relation. By Definition 2.15, it suffices to check $L(s_1) = L(s_2)$ and the condition:

$$\forall \alpha \in Act. \ \forall s_1 \xrightarrow{\alpha} \mu_1. \ \exists s_2 \overset{\alpha}{\leadsto} \mu_2 \text{ with } \mu_1 \sqsubseteq_R \mu_2 \tag{4.3}$$

However, since the combined transition involves the quantification of the constants $c_i \in [0, 1]$, there are possibly infinitely many such μ_2. Thus, one cannot check $\mu_1 \sqsubseteq_R \mu_2$ for each possible candidate μ_2. The following lemma shows that this condition can be checked by solving LP problems which are decidable in polynomial time [27, 38].

Lemma 4.12. *Let $\mathcal{M} = (S, Act, \mathbf{P}, L)$ be a given PA, and let $R \subseteq S \times S$. Let $(s_1, s_2) \in R$ with $L(s_1) = L(s_2)$ and $Act(s_1) \subseteq Act(s_2)$. Then, $s_1 \precsim_R^p s_2$ iff for each transition $s_1 \xrightarrow{\alpha} \mu$, the following LP has a feasible solution:*

$$\sum_{i=1}^{k} c_i = 1 \tag{4.4}$$

$$0 \le c_i \le 1 \qquad \forall \ i = 1, \dots, k \tag{4.5}$$

$$0 \le f_{(s,t)} \le 1 \qquad \forall (s,t) \in R_\perp \tag{4.6}$$

$$\mu(s) = \sum_{t \in R_\perp(s)} f_{(s,t)} \qquad \forall s \in S_\perp \tag{4.7}$$

$$\sum_{s \in R_\perp^{-1}(t)} f_{(s,t)} = \sum_{i=1}^{k} c_i \mu_i(t) \qquad \forall t \in S_\perp \tag{4.8}$$

where $k = |Steps_\alpha(s_2)| > 0$ and $Steps_\alpha(s_2) = \{\mu_1, \dots, \mu_k\}$.

Proof. First assume that $s_1 \precsim_R^p s_2$. Let $s_1 \xrightarrow{\alpha} \mu$. By the definition of simulation up to R for strong probabilistic simulation, there exists a combined transition $s_2 \overset{\alpha}{\leadsto} \mu_c$ with $\mu \sqsubseteq_R \mu_c$. Let $Steps_\alpha(s_2) = \{\mu_1, \dots, \mu_k\}$ where $k = |Steps_\alpha(s_2)|$. Now $Act(s_1) \subseteq Act(s_2)$ implies $k > 0$. By definition of combined transition (Definition 2.14), there exist constants $c_1, \dots, c_k \in [0, 1]$ with $\sum_{i=1}^{k} c_i = 1$ such that $\mu_c = \sum_{i=1}^{k} c_i \mu_i$. Thus Constraints 4.4 and 4.5 hold. Since $\mu \sqsubseteq_R \mu_c$, there exists a weight function $\Delta : S_\perp \times S_\perp \to [0, 1]$ for (μ, μ_c) with

$\text{SimRel}_s^{\text{PA},p}(\mathcal{M})$

6.1: $R_1 \leftarrow \{(s_1, s_2) \in S \times S \mid L(s_1) = L(s_2) \wedge Act(s_1) \subseteq Act(s_2)\}$ and $i \leftarrow 0$
6.2: **repeat**
6.3: $i \leftarrow i + 1$
6.4: $R_{i+1} \leftarrow \emptyset$
6.5:   **for all** $(s_1, s_2) \in R_i$ **do**
6.6:   **for all** $\alpha \in Act(s_1), \quad \mu_1 \in Steps_\alpha(s_1)$ **do**
6.7: $match_{\alpha,\mu_1} \leftarrow LP(s_1, \alpha, \mu_1, s_2)$
6.8:   **if** $\bigwedge_{\alpha \in Act(s_1)} \bigwedge_{\mu_1 \in Steps_\alpha(s_1)} match_{\alpha,\mu_1}$ **then**
6.9: $R_{i+1} \leftarrow R_{i+1} \cup \{(s_1, s_2)\}$
6.10: **until** $R_{i+1} = R_i$
6.11: **return** R_i

Algorithm 6: Algorithm for deciding strong probabilistic simulation for PAs.

respect to R. For every pair $(s,t) \in R_\perp$, let $f_{(s,t)} := \Delta(s,t)$. Thus, Constraint 4.6 holds trivially. By Definition 2.7 of weight functions, it holds that (i) $\Delta(s,t) > 0$ implies that $(s,t) \in R_\perp$, (ii) $\mu(s) = \sum_{t \in S_\perp} \Delta(s,t)$ for $s \in S_\perp$, and (iii) $\mu_c(t) = \sum_{s \in S_\perp} \Delta(s,t)$ for all $t \in S_\perp$. Observe that (i) implies that for all $(s,t) \notin R_\perp$, we have that $\Delta(s,t) = 0$. Thus, (ii) and (iii) imply Equations 4.7 and 4.8 respectively.

Now we show the other direction. Let $k = |Steps_\alpha(s_2)|$ and $Steps_\alpha(s_2) = \{\mu_1, \ldots, \mu_k\}$. By assumption, for each $s_1 \xrightarrow{\alpha} \mu$, we have a feasible solution $c_1, \ldots, c_k$ and $f_{(s,t)}$ for all $(s,t) \in R_\perp$ which satisfies all of the constraints. We define $\mu_c = \sum_{i=1}^k c_i \mu_i$. By Definition 2.14, μ_c is a combined transition, thus $s_2 \xrightarrow{\alpha} \mu_c$. It remains to show that $\mu \sqsubseteq_R \mu_c$. We define a function Δ as follows: $\Delta(s,t)$ equals $f_{(s,t)}$ if $(s,t) \in R_\perp$ and 0 otherwise. With the help of Constraints 4.6, 4.7 and 4.8 we have that Δ is a weight function for (μ, μ_c) with respect to R, thus $\mu \sqsubseteq_R \mu_c$. $\square$

Now we are able to check Condition 4.3 by solving LP problems. For a PA $\mathcal{M} = (S, Act, \mathbf{P}, L)$, and a relation $R \subseteq S \times S$, let $(s_1, s_2) \in R$ with $L(s_1) = L(s_2)$ and $Act(s_1) \subseteq Act(s_2)$. For $s_1 \xrightarrow{\alpha} \mu_1$, we introduce a predicate $LP(s_1, \alpha, \mu, s_2)$ which is true iff the LP problem described as in Lemma 4.12 has a solution. Then, $s_1 \precsim_R^p s_2$ iff the conjunction $\bigwedge_{\alpha \in Act(s_1)} \bigwedge_{\mu_1 \in Steps_\alpha(s_1)} LP(s_1, \alpha, \mu_1, s_2)$ is true. The algorithm, which is denoted by $\text{SimRel}_s^{\text{PA},p}(\mathcal{M})$, is depicted in Algorithm 6. It takes the skeleton of $\text{SimRel}_s(\mathcal{M})$. The key difference is that we incorporate the predicate $LP(s_1, \alpha, \mu_1, s_2)$ in line 6.7. The correctness of the algorithm $\text{SimRel}_s^{\text{PA},p}(\mathcal{M})$ can be obtained from the one of $\text{SimRel}_s(\mathcal{M})$ together with Lemma 4.12. We discuss briefly the complexity. The number of variables in the LP problem in Lemma 4.12 is $k + |R|$, and the number of constraints is $1 + k + |R| + 2|S| \in \mathcal{O}(|R|)$. In iteration i of $\text{SimRel}_s^{\text{PA},p}(\mathcal{M})$, for $(s_1, s_2) \in R_i$ and $s_1 \xrightarrow{\alpha} \mu_1$, the corresponding LP problem is queried once. The number of iterations is in $\mathcal{O}(n^2)$. Therefore, in the worst case, one has to solve $n^2 \sum_{s \in S} \sum_{\alpha \in Act(s)} \sum_{\mu \in Steps_\alpha(s)} 1 \in \mathcal{O}(n^2 m)$ many such LP problems and each of them has at most $\mathcal{O}(n^2)$ constraints.

4.4.2. *Continuous-time Probabilistic Automata.* Now we discuss how to extend the algorithm to handle CPAs. Let $\mathcal{M} = (S, Act, \mathbf{R}, L)$ be a CPA. Similar to PAs, the most important part is to check the condition $s_1 \precsim_R^p s_2$ for some relation $R \subseteq S \times S$. By Definition 2.19,

$\text{SIMREL}_s^{\text{CPA},p}(\mathcal{M})$

7.1: $R_1 \leftarrow \{(s_1, s_2) \in S \times S \mid L(s_1) = L(s_2) \wedge Act(s_1) \subseteq Act(s_2)\}$ and $i \leftarrow 0$
7.2: **repeat**
7.3: $i \leftarrow i + 1$
7.4: $R_{i+1} \leftarrow \emptyset$
7.5:   **for all** $(s_1, s_2) \in R_i$ **do**
7.6:   **for all** $\alpha \in Act(s_1), \quad r_1 \in Steps_\alpha(s_1), \quad E \in E(s_2)$ **do**
7.7: $match_{\alpha, r_1, E} \leftarrow LP'(s_1, \alpha, r_1, s_2, E)$
7.8:   **if** $\bigwedge_{\alpha \in Act(s_1)} \bigwedge_{r_1 \in Steps_\alpha(s_1)} \bigwedge_{E \in E(s_2)} match_{\alpha, r_1, E}$ **then**
7.9: $R_{i+1} \leftarrow R_{i+1} \cup \{(s_1, s_2)\}$
7.10: **until** $R_{i+1} = R_i$
7.11: **return** R_i

Algorithm 7: Algorithm for deciding strong probabilistic simulation for CPAs.

it suffices to check $L(s_1) = L(s_2)$ and the condition:

$$\forall \alpha \in Act. \; \forall s_1 \xrightarrow{\alpha} r_1. \; \exists s_2 \overset{\alpha}{\rightsquigarrow} r_2 \text{ with } \mu(r_1) \sqsubseteq_R \mu(r_2) \wedge r_1(S) \leq r_2(S) \tag{4.9}$$

Recall that for CPAs only successor rate functions with the same exit rate can be combined together. For state $s \in S$, we let $E(s) := \{r(S) \mid s \xrightarrow{\alpha} r\}$ denote the set of all possible exit rates of α-successor rate functions of s. For $E \in E(s)$ and $\alpha \in Act(s)$, we let $Steps_\alpha^E(s) = \{r \in Steps_\alpha(s) \mid r(S) = E\}$ denote the set of α-successor rate functions of s with the same exit rate E. As for PAs, to check the condition $s_1 \precsim_R^p s_2$ we resort to a reduction to LP problems.

Lemma 4.13. *Let $\mathcal{M} = (S, Act, \mathbf{R}, L)$ be a given CPA, and let $R \subseteq S \times S$. Let $(s_1, s_2) \in R$ with $L(s_1) = L(s_2)$ and that $Act(s_1) \subseteq Act(s_2)$. Then, $s_1 \precsim_R^p s_2$ iff for each transition $s_1 \xrightarrow{\alpha} r$ either $r(S) = 0$, or there exists $E \in E(s_2)$ with $E \geq r(S)$ such that the following LP has a feasible solution, which consists of Constraints 4.4, 4.5, 4.6 of Lemma 4.12, and additionally:*

$$r(s) = r(S) \sum_{t \in R_\perp(s)} f_{(s,t)} \qquad \forall s \in S_\perp \tag{4.10}$$

$$E \sum_{s \in R_\perp^{-1}(t)} f_{(s,t)} = \sum_{i=1}^{k} c_i r_i(t) \qquad \forall t \in S_\perp \tag{4.11}$$

where $k = \left| Steps_\alpha^E(s) \right|$ with $Steps_\alpha^E(s) = \{r_1, \ldots, r_k\}$.

Proof. The proof follows the same strategy as the proof of Lemma 4.12, in which the induced distribution of the corresponding rate function should be used. $\qquad\qquad\square$

Now we are able to check Condition 4.9 by solving LP problems. For a CPA $\mathcal{M} = (S, Act, \mathbf{R}, L)$, and a relation $R \subseteq S \times S$, let $(s_1, s_2) \in R$ with $L(s_1) = L(s_2)$ and $Act(s_1) \subseteq Act(s_2)$. For $s_1 \xrightarrow{\alpha} r_1$, and $E \in E(s_2)$, we introduce the predicate $LP'(s_1, \alpha, r_1, s_2, E)$ which is true iff $E \geq r_1(S)$ and the corresponding LP problem has a solution. Then, $s_1 \precsim_R^p s_2$ iff the conjunction $\bigwedge_{\alpha \in Act(s_1)} \bigwedge_{r_1 \in Steps_\alpha(s_2)} \bigwedge_{E \in E(s_2)} LP'(s_1, \alpha, r_1, s_2, E)$ is true. The decision algorithm is given in Algorithm 7. As complexity we have to solve $\mathcal{O}(n^2 m)$ LP problems and each of them has at most $\mathcal{O}(n^2)$ constraints.

5. Algorithms for Deciding Weak Simulations

We now turn our attention to weak simulations. Similar to strong simulations, the core of the algorithm is to check whether $s_1 \precsim_R s_2$, i.e., s_2 weakly simulates s_1 up to the current relation R. As for strong simulation up to R, $s_1 \precsim_R s_2$ does not imply $s_1 \precsim_{\mathcal{M}} s_2$, since no conditions are imposed on pairs in R different from (s_1, s_2). By the definition of weak simulation, for fixed characteristic functions δ_i ($i = 1, 2$), the weight function conditions can be checked by applying maximum flow algorithms. Unfortunately, δ_i-functions are not known a priori. Inspired by the parametric maximum flow algorithm, in this chapter, we show that one can determine whether such characteristic functions δ_i exist with the help of *breakpoints*, which can be computed by analysing a parametric network constructed out of $\mathbf{P}(s_1, \cdot), \mathbf{P}(s_2, \cdot)$ and R. We present dedicated algorithms for DTMCs in Subsection 5.1 and CTMCs in Subsection 5.2.

5.1. **An Algorithm for DTMCs.** Let $\mathcal{M} = (S, \mathbf{P}, L)$ be a DTMC. Let $R \subseteq S \times S$ be a relation and $s_1 \, R \, s_2$. Whether s_2 weakly simulates s_1 up to R is equivalent to whether there exist functions $\delta_i : S \to [0, 1]$ such that the conditions in Definition 2.21 are satisfied. Assume that we are given the U_i-characterising functions δ_i. In this case, $s_1 \precsim_R s_2$ can be checked as follows:

- Concerning Condition 1a we check whether for all $v \in S$ with $\delta_1(v) < 1$ it holds that $v \, R \, s_2$. Similarly, for Condition 1b, we check whether for all $v \in S$ with $\delta_2(v) < 1$ it holds that $s_1 \, R \, v$.
- The reachability condition can be checked by using standard graph algorithms. In more detail, for each u with $\delta_1(u) > 0$, the condition holds if a state in $R(u)$ is reachable from s_2 via $R(s_1)$ states.
- Finally consider Condition 2. From the given δ_i functions we can compute K_i. In case of that $K_1 > 0$ and $K_2 > 0$, we need to check whether there exists a weight function for the conditional distributions $\mathbf{P}(s_1, \cdot)/K_1$ and $\mathbf{P}(s_2, \cdot)/K_2$ with respect to the current relation R. From Lemma 4.1, this is equivalent to check whether the maximum flow for the network constructed from $(\mathbf{P}(s_1, \cdot)/K_1, \mathbf{P}(s_2, \cdot)/K_2)$ and R has value 1.

To check $s_1 \precsim_R s_2$, we want to check whether such δ_i functions exist. The difficulty is that there exist uncountably many possible δ_i functions. In this section, we first show that whether such δ_i exists can be characterised by analysing a parametric network in Subsection 5.1.1. Then, in Subsection 5.1.2, we recall the notion of breakpoints, and show that the breakpoints play a central role in the parametric networks considered: only these points need to be considered. Based on this, we present the algorithm for DTMCs in Subsection 5.1.3. An improvement of the algorithm for certain cases is reported in Subsection 5.1.4.

5.1.1. *The Parametric Network* $\mathcal{N}(\gamma)$. Let $\gamma \in \mathbb{R}_{\geq 0}$. Recall that $\mathcal{N}(\mathbf{P}(s_1, \cdot), \gamma\mathbf{P}(s_2, \cdot), R)$ is obtained from the network $\mathcal{N}(\mathbf{P}(s_1, \cdot), \mathbf{P}(s_2, \cdot), R)$ be setting the capacities to the sink $\searrow$ by: $c(\bar{t}, \searrow) = \gamma\mathbf{P}(s_2, t)$. If s_1, s_2, R are clear from the context, we use $\mathcal{N}(\gamma)$ to denote the network $\mathcal{N}(\mathbf{P}(s_1, \cdot), \gamma\mathbf{P}(s_2, \cdot), R)$ for arbitrary $\gamma \in \mathbb{R}_{\geq 0}$.

We introduce some notations. We focus on a particular pair $(s_1, s_2) \in R$, where R is the current relation. We partition the set $post(s_i)$ into MU_i (for: must be in U_i) and PV_i (for: potentially in V_i). The set PV_1 consists of those successors of s_1 which can be either put into U_1 or V_1 or both: $PV_1 = post(s_1) \cap R^{-1}(s_2)$. The set MU_1 equals $post(s_1) \backslash PV_1$, which
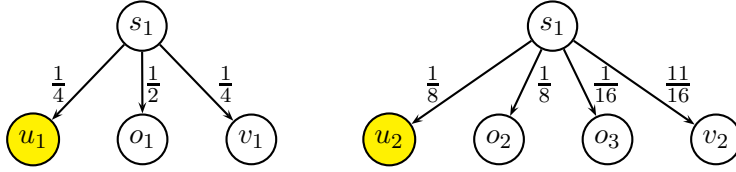
Figure 8: A simple DTMC.

consists of the successor states which can only be placed in U_1. The sets PV_2 and MU_2 are defined similarly by: $PV_2 = post(s_2) \cap R(s_1)$ and $MU_2 = post(s_2) \backslash PV_2$. Obviously, $\delta_i(u) = 1$ for $u \in MU_i$ for $i = 1, 2$.

Example 5.1. Consider the DTMC in Figure 8 and the relation $R = \{(s_1, s_2), (s_1, v_2), (v_1, s_2), (u_1, u_2), (o_1, o_2), (o_1, v_2), (v_1, o_3), (v_1, v_2), (o_2, o_1)\}$. We have $PV_1 = \{v_1\}$ and $PV_2 = \{v_2\}$. Thus, $MU_1 = \{u_1, o_1\}$, $MU_2 = \{u_2, o_2, o_3\}$.

We say a flow function f of $\mathcal{N}(\gamma)$ is valid for $\mathcal{N}(\gamma)$ iff f saturates all edges $(\nearrow, u_1)$ with $u_1 \in MU_1$ and all edges $(\overline{u_2}, \searrow)$ with $u_2 \in MU_2$. If there exists a valid flow f for $\mathcal{N}(\gamma)$, we say that γ is valid for $\mathcal{N}(\gamma)$. The following lemma considers the case in which both s_1 and s_2 have visible steps:

Lemma 5.2. Let $s_1 \, R \, s_2$. Assume that there exists a state $s_1' \in post(s_1)$ such that $s_1' \notin R^{-1}(s_2)$, and $s_2' \in post(s_2)$ such that $s_2' \notin R(s_1)$. Then, $s_1 \precsim_R s_2$ iff there exists a valid γ for $\mathcal{N}(\gamma)$.

Proof. By assumption, we have that $s_i' \in MU_i$ for $i = 1, 2$, thus $MU_i \neq \emptyset$, and it holds that $\delta_i(s_i') = 1$ for $i = 1, 2$.

We first show the *only if* direction. Assume $s_1 \precsim_R s_2$, and let $\delta_i, U_i, V_i, K_i, \Delta$ (for $i = 1, 2$) as described in Definition 2.21. Since $MU_i \neq \emptyset$ for $i = 1, 2$, both K_1 and K_2 are greater than 0. We let $\gamma = K_1/K_2$. For $s, s' \in S$, we define the function f for $\mathcal{N}(\gamma)$:

$$f(\nearrow, s) = \mathbf{P}(s_1, s)\delta_1(s), \qquad f(s, \overline{t}) = K_1\Delta(s, t), \qquad f(\overline{s}, \searrow) = \gamma\mathbf{P}(s_2, s)\delta_2(s)$$

Since $\delta_i(s) \leq 1$ ($i = 1, 2$) for $s \in S$, $f(\nearrow, s) \leq \mathbf{P}(s_1, s)$ and $f(\overline{s}, \searrow) \leq \gamma\mathbf{P}(s_2, s)$. Therefore, f satisfies the capacity constraints. f also satisfies the conservation rule:

$$f(s, \overline{S}) = K_1\Delta(s, S) = \mathbf{P}(s_1, s)\delta_1(s) = f(\nearrow, s)$$
$$f(S, \overline{s}) = K_1\Delta(S, s) = \gamma K_2\Delta(S, s) = \gamma\mathbf{P}(s_2, s)\delta_2(s) = f(\overline{s}, \searrow)$$

Hence, f is a flow function for $\mathcal{N}(\gamma)$. For $u_1 \in MU_1$, we have $\delta_1(u_1) = 1$, therefore, $f(\nearrow, u_1) = \mathbf{P}(s_1, u_1)$. Analogously, $f(\overline{u_2}, \searrow) = \gamma\mathbf{P}(s_2, u_2)$ for $u_2 \in MU_2$. Hence, f is valid for $\mathcal{N}(\gamma)$, implying that γ is valid for $\mathcal{N}(\gamma)$.

Now we show the *if* direction. Assume that there exists $\gamma > 0$ and a valid flow f for $\mathcal{N}(\gamma)$. The function δ_1 is defined by: $\delta_1(s)$ equals $f(\nearrow, s)/\mathbf{P}(s_1, s)$ if $s \in post(s_1)$ and 0 otherwise. The function δ_2 is defined similarly: $\delta_2(s)$ equals $f(\overline{s}, \searrow)/\gamma\mathbf{P}(s_2, s)$ if $s \in post(s_2)$ and 0 otherwise. Let the sets U_i and V_i be defined as required by Definition 2.21. It follows that

$$K_1 = \sum_{s \in U_1} \delta_1(s)\mathbf{P}(s_1, s) = \sum_{s \in U_1} f(\nearrow, s) = f(\nearrow, U_1)$$

$$K_2 = \sum_{s \in U_2} \delta_2(s)\mathbf{P}(s_2, s) = \sum_{s \in U_2} \frac{f(\overline{s}, \searrow)}{\gamma} = \frac{f(\overline{U_2}, \searrow)}{\gamma}$$
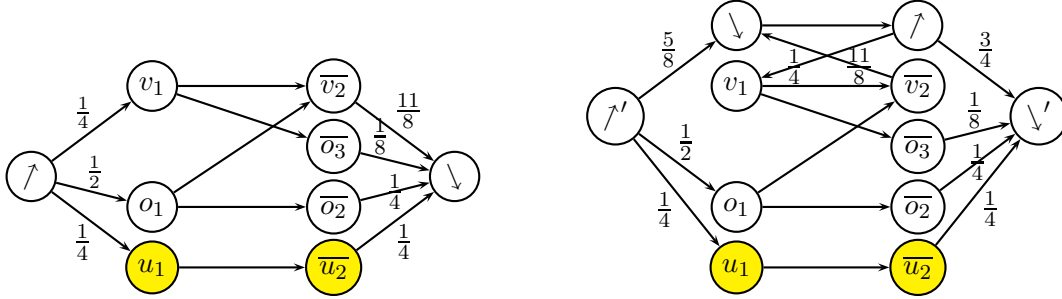
Figure 9: Left: The network $\mathcal{N}(2)$ of the DTMC in Figure 8; Right: The transformed network $\mathcal{N}_t(2)$ for $\mathcal{N}(2)$.

Since the amount of flow out of $\nearrow$ is the same as the amount of flow into $\searrow$, we have $K_1/K_2 = \gamma$. Since $\emptyset \neq MU_i \subseteq U_i$ for $i = 1, 2$, both of K_1 and K_2 are greater than 0. We show that the Conditions 1a and 1b of Definition 2.21 are satisfied. For $v_1 \in V_1$, we have that $\delta_1(v_1) < 1$ which implies that $f(\nearrow, v_1) < \mathbf{P}(s_1, v_1)$. Since f is valid for $\mathcal{N}(\gamma)$, and since the edge $(\nearrow, v_1)$ is not saturated by f, it must hold that $v_1 \in PV_1$. Therefore, $v_1 \ R \ s_2$. Similarly, we can prove that $s_1 \ R \ v_2$ for $v_2 \in V_2$.

We define $\Delta(w, w') = f(w, \overline{w'})/K_1$ for $w, w' \in S$. Assume that $\Delta(w, w') > 0$. Then, $f(w, \overline{w'}) > 0$, which implies that $(w, \overline{w'})$ is an edge of $\mathcal{N}(\gamma)$, therefore, $(w, w') \in R$. By the flow conservation rule, $f(\nearrow, w) \geq f(w, \overline{w'}) > 0$, implying that $\delta_1(w) > 0$. By the definition of U_1, we obtain that $w \in U_1$. Similarly, we can show that $w' \in U_2$. Hence, the Condition 2a is satisfied. To prove Condition 2b:

$$\Delta(w, U_2) = \sum_{u_2 \in U_2} \frac{f(w, \overline{u_2})}{K_1} = \frac{f(w, \overline{U_2})}{K_1} \overset{(*)}{=} \frac{f(\nearrow, w)}{K_1} = \frac{\delta_1(w)\mathbf{P}(s_1, w)}{K_1}$$

where equality $(*)$ follows from the flow conservation rule. Therefore, for $w \in S$ we have that $K_1\Delta(w, U_2) = \mathbf{P}(s_1, w)\delta_1(w)$. Similarly, we can show $K_2\Delta(U_1, w) = \mathbf{P}(s_2, w)\delta_2(w)$. Condition 2b is also satisfied. As $K_1 > 0$ and $K_2 > 0$, the reachability condition holds trivially, hence, $s_1 \precsim_R s_2$. $\square$

Example 5.3. Consider again Example 5.1 with the relation $R = \{(s_1, s_2), (s_1, v_2), (v_1, s_2), (u_1, u_2), (o_1, o_2), (o_1, v_2), (v_1, o_3), (v_1, v_2), (o_2, o_1)\}$. The network $\mathcal{N}(2)$ is depicted on the left part of Figure 9. Edges without numbers have capacity ∞. It is easy to see that 2 is valid for $\mathcal{N}(2)$: the corresponding flow sends $\frac{1}{4}$ amount of flow along the path $\nearrow, u_1, \overline{u_2}, \searrow$, $\frac{1}{4}$ amount of flow along the path $\nearrow, o_1, \overline{o_2}, \searrow$, $\frac{1}{4}$ amount of flow along the path $\nearrow, o_1, \overline{v_2}, \searrow$, and $\frac{1}{8}$ amount of flow along the path $\nearrow, v_1, \overline{o_3}, \searrow$.

For a fixed $\gamma \in \mathbb{R}_{>0}$, we now address the problem of checking whether there exists a valid flow f for $\mathcal{N}(\gamma)$. This is a feasible flow problem (f has to saturate edges to MU_1 and from MU_2). As we have discussed in Section 3, it can be solved 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.

Example 5.4. Consider the network $\mathcal{N}(2)$ on the left part of Figure 9. Applying the transformation for the feasible flow problem described in Section 3, we get the transformed network $\mathcal{N}_t(2)$ depicted on the right part of Figure 9. It is easy to see that the maximum
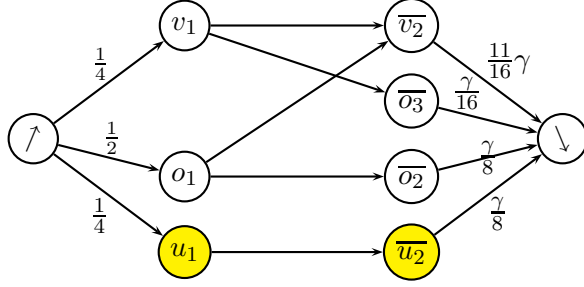
Figure 10: The network $\mathcal{N}(\gamma)$ of the DTMC in Figure 8 in Example 5.1.

flow h for $\mathcal{N}_t(2)$ has value $\frac{11}{8}$. Namely: It sends $\frac{1}{4}$ amount of flow along the path $\nearrow', u_1, \overline{u_2}, \searrow'$, $\frac{1}{4}$ amount of flow along the path $\nearrow', o_1, \overline{o_2}, \searrow'$, $\frac{1}{4}$ amount of flow along $\nearrow', o_1, \overline{v_2}, \searrow, \nearrow, \searrow'$, $\frac{1}{8}$ amount of flow along $\nearrow', \searrow, \nearrow, v_1, \overline{o_3}, \searrow'$, and $\frac{1}{2}$ amount of flow along $\nearrow', \searrow, \nearrow, \searrow'$. Thus, it uses all capacities of edges from $\nearrow'$. This implies that 2 is valid for the network $\mathcal{N}(2)$.

5.1.2. *Breakpoints.* Consider the pair $(s_1, s_2) \in R$. Assume the conditions of Lemma 5.2 are satisfied, thus, to check whether $s_1 \precsim_R s_2$ it is equivalent to check whether a valid γ for $\mathcal{N}(\gamma)$ exits. We show that only a finite possible γ, called breakpoints, need to be considered. The breakpoints can be computed using a variant of the parametric maximum flow algorithm. Then, $s_1 \precsim_R s_2$ if and only if for some breakpoint it holds that the maximum flow for the corresponding transformed network $\mathcal{N}_t(\gamma)$ has a large enough value.

Let $|V|$ denote the number of vertices of $\mathcal{N}(\gamma)$. Let $\kappa(\gamma)$ denote the *minimum cut capacity function* of the parameter γ, which is the capacity of a minimum cut of $\mathcal{N}(\gamma)$ as a function of γ. The capacity of a minimum cut equals the value of a maximum flow. If the edge capacities in the network are linear functions of γ, $\kappa(\gamma)$ is a piecewise-linear concave function with at most $|V| - 2$ breakpoints [18], i.e., points where the slope $\frac{d\kappa}{d\gamma}$ changes. The $|V| - 1$ or fewer line segments forming the graph of $\kappa(\gamma)$ correspond to $|V| - 1$ or fewer distinct minimal cuts. The minimum cut can be chosen as the same on a single linear piece of $\kappa(\gamma)$, and at breakpoints certain edges become saturated or unsaturated. The capacity of a minimum cut for some γ^* gives an equation that contributes a line segment to the function $\kappa(\gamma)$ at $\gamma = \gamma^*$. Moreover, this line segment connects the two points $(\gamma_1, \kappa(\gamma_1))$ and $(\gamma_2, \kappa(\gamma_2))$, where γ_1, γ_2 are the nearest breakpoints to the left and right, respectively.

Example 5.5. Consider the DTMC in Figure 8, together with the relation $R = \{(s_1, s_2), (s_1, v_2), (v_1, s_2), (u_1, u_2), (o_1, o_2), (o_1, v_2), (v_1, o_3), (v_1, v_2), (o_2, o_1)\}$. The network $\mathcal{N}(\gamma)$ for the pair (s_1, s_2) is depicted in Figure 10.

There are two breakpoints, namely $\frac{6}{7}$ and 2. For $\gamma \le \frac{6}{7}$, all edges leading to the sink can be saturated. This can be established by the following flow function f: sending $\frac{\gamma}{8}$ amount of flow along the path $\nearrow, u_1, \overline{u_2}, \searrow$, $\frac{\gamma}{8}$ amount of flow along the path $\nearrow, o_1, \overline{o_2}, \searrow$, $\frac{11\gamma}{24}$ amount of flow along the path $\nearrow, o_1, \overline{v_2}, \searrow$, $\frac{\gamma}{16}$ amount of flow over $\nearrow, v_1, \overline{o_3}, \searrow$, $\frac{11\gamma}{48}$ amount of flow along the path $\nearrow, v_1, \overline{v_2}, \searrow$. The amount of flow out of node o_1, denoted by $f(o_1, \overline{S})$, is $\frac{7\gamma}{12}$. Given that $\gamma \le \frac{6}{7}$, we have that $f(o_1, \overline{S}) \le \frac{1}{2}$. Similarly, consider the amount of flow out of node v_1, which is denoted by $f(v_1, \overline{S})$, is $\frac{7\gamma}{24}$ which implies that $f(v_1, \overline{S}) \le \frac{1}{4}$. The maximum flow thus has value $|f| = \frac{\gamma}{8} + \frac{\gamma}{8} + \frac{11\gamma}{24} + \frac{\gamma}{16} + \frac{11\gamma}{48} = \gamma$. Thus the value of the maximum flow, or equivalently the value of the minimum cut, for $\gamma \le \frac{6}{7}$ is $\kappa(\gamma) = \gamma$.
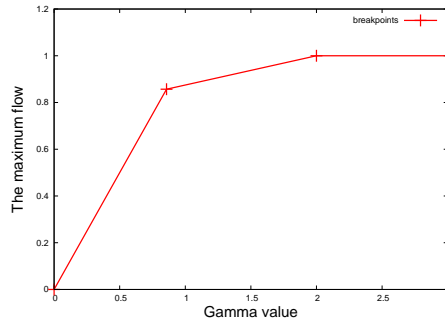
Figure 11: The value of the maximum flow, or equivalently the value of the minimum cut, as a function of γ for the network in Figure 10.

Observe that for $\gamma = \frac{6}{7}$, the edges to v_1 and o_1 are saturated, i.e., we have used full capacities of the edge $(\nearrow, v_1)$ and $(\nearrow, o_1)$. Thus, by a greater value of γ, although the capacities $c(\{\overline{v_2}, \overline{o_2}, \overline{o_3}\}, \searrow)$ increase (become greater than $\frac{3}{4}$), no additional flow can be sent through $\{v_1, o_1\}$. For the other breakpoint 2, we observe that for a value of $\gamma \leq 2$, we can still send $\frac{\gamma}{8}$ through the path $\nearrow, u_1, u_2, \searrow$, but for $\gamma > 2$, the edge to u_1 keeps saturated, thus the amount of flow sent through this path does not increase any more. Thus, for $\gamma \in [\frac{6}{7}, 2]$, the maximum value, or the value of the minimum cut, is $\frac{3}{4} + \frac{\gamma}{8}$. The first term $\frac{3}{4}$ corresponds to the amount of flow through v_1 and o_1. The breakpoint $\frac{6}{7}$ is not valid since the edge to u_1 can not be saturated. As discussed in Example 5.4, the breakpoint 2 is valid. The curve for $\kappa(\gamma)$ is depicted in Figure 11.

In the following lemma we show that if there is any valid γ, then at least one breakpoint is valid.

Lemma 5.6. *Assume $\gamma^* \in (\gamma_1, \gamma_2)$ where γ_1, γ_2 are two subsequent breakpoints of $\kappa(\gamma)$, or $\gamma_1 = 0$ and γ_2 is the first breakpoint, or γ_1 is the last breakpoint and $\gamma_2 = \infty$. Assume γ^* is valid for $\mathcal{N}(\gamma^*)$, then, γ is valid for $\mathcal{N}(\gamma)$ for all $\gamma \in [\gamma_1, \gamma_2]$.*

Proof. Consider the network $\mathcal{N}(\gamma^*)$. Assume that the maximum flow f_{γ^*} is a valid maximum flow for $\mathcal{N}(\gamma^*)$.

Assume first $\gamma' \in (\gamma^*, \gamma_2]$. We use the augmenting path algorithm [1] to obtain a maximum flow f^* in the residual network $\mathcal{N}_{f_{\gamma^*}}(\gamma')$, requiring that the augmenting path contains no cycles, which is a harmless restriction. Then, $f_{\gamma'} := f_{\gamma^*} + f^*$ is a maximum flow in $\mathcal{N}(\gamma')$. Since f_{γ^*} saturates edges from $\nearrow$ to MU_1, $f_{\gamma'}$ saturates edges from $\nearrow$ to MU_1 as well , as flow along an augmenting path without cycles does not un-saturate edges to MU_1. We choose the minimum cut (X, X') for $\mathcal{N}(\gamma^*)$ with respect to f_{γ^*} such that $\overline{MU_2} \cap X' = \emptyset$, or equivalently $\overline{MU_2} \subseteq X$. This is possible since f_{γ^*} saturates all edges $(\overline{u_2}, \searrow)$ with $u_2 \in MU_2$. The minimum cut for $f_{\gamma'}$, then, can also be chosen as (X, X'), as $(\gamma', \kappa(\gamma'))$ lies on the same line segment as $(\gamma^*, \kappa(\gamma^*))$. Hence, $f_{\gamma'}$ saturates the edges from $\overline{MU_2}$ to $\searrow$, which indicates that $f_{\gamma'}$ is valid for $\mathcal{N}(\gamma')$. Therefore, γ' is valid for $\mathcal{N}(\gamma')$ for $\gamma' \in (\gamma^*, \gamma_2]$.

Now let $\gamma' \in [\gamma_1, \gamma^*)$. For the valid maximum flow f_{γ^*} we select the minimal cut (X, X') for $\mathcal{N}(\gamma^*)$ such that $MU_1 \cap X = \emptyset$. Let d denote a valid distance function corresponding to f_{γ^*}. We replace $f_{\gamma^*}(\overline{v}, \searrow)$ by $\min\{f_{\gamma^*}(\overline{v}, \searrow), c_{\gamma'}(\overline{v}, \searrow)\}$ where $c_{\gamma'}$ is the capacity function of $\mathcal{N}(\gamma')$. The modified flow is a preflow for the network $\mathcal{N}(\gamma')$. Moreover, d stays a valid
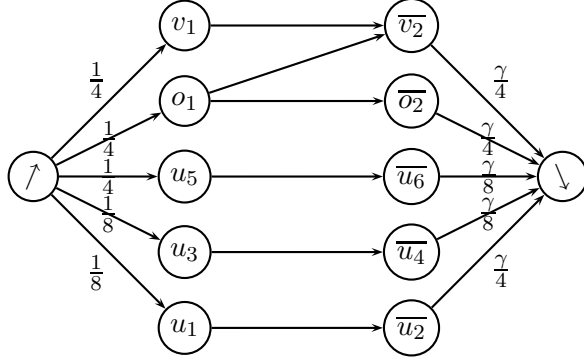
Figure 12: A network in which more than one breakpoint is valid.

distance function as no new residual edges are introduced. Then, we apply the preflow algorithm to get a maximum flow $f_{\gamma'}$ for the network $\mathcal{N}(\gamma')$. Since no flow is pushed back from the sink, edges from $\overline{MU_2}$ to $\searrow$ are kept saturated. Since $(\gamma^*, \kappa(\gamma^*))$ and $(\gamma', \kappa(\gamma'))$ are on the same line segment, the minimal cut for $f_{\gamma'}$ can also be chosen as (X, X'), which indicates that $f_{\gamma'}$ saturates all edges to MU_1. This implies that γ' is valid for $\mathcal{N}(\gamma')$ for $\gamma' \in [\gamma_1, \gamma^*)$. $\qquad\square$

In Example 5.5, only one breakpoint is valid. In the following example we show that it is in general possible that more than one breakpoint is valid.

Example 5.7. Consider the network depicted in Figure 12. By a similar analysis as Example 5.5, we can compute that there are three breakpoints $\frac{1}{2}$, 1 and 2. Assuming that $MU_1 = \{o_1\}$ and $MU_2 = \{o_2\}$, we show that all $\gamma \in [\frac{1}{2}, 1]$ are valid. We send $\frac{\gamma}{4}$ amount of flow along the path $\nearrow, o_1, \overline{o_2}, \searrow$, and $\frac{1}{4} - \frac{\gamma}{4}$ amount of flow along the path $\nearrow, o_1, \overline{v_2}, \searrow$. If $\gamma \in [\frac{1}{2}, 1]$, then $0 \leq \frac{1}{4} - \frac{\gamma}{4} \leq \frac{\gamma}{4}$ implying that the flow on edge $(\overline{v_2}, \searrow)$ satisfies the capacity constraints. Obviously this flow is feasible, and all $\gamma \in [\frac{1}{2}, 1]$ are valid for $\mathcal{N}(\gamma)$.

As we would expect now, it is sufficient to consider only the breakpoints of $\mathcal{N}(\gamma)$:

Lemma 5.8. *There exists a valid γ for $\mathcal{N}(\gamma)$ iff one of the breakpoints of $\mathcal{N}(\gamma)$ is valid.*

Proof. If there exists a valid γ for $\mathcal{N}(\gamma)$, Lemma 5.6 guarantees that one of the breakpoints of $\kappa(\gamma)$ is valid. The other direction is trivial. $\qquad\square$

For a given breakpoint, we need to solve one feasible flow problem to check whether it is valid. In the network $\mathcal{N}(\gamma)$ the capacities of the edges leading to the sink are increasing functions of a real-valued parameter γ. If we reverse $\mathcal{N}(\gamma)$, we get a parametric network that satisfies the conditions in [18]: The capacities emanating from $\nearrow$ are non-decreasing functions of γ. So we can apply the *breakpoint algorithm* [18] to obtain all of the breakpoints of $\mathcal{N}(\gamma)$.

5.1.3. *The Algorithm for DTMCs.* Let $\mathcal{M}$ be a DTMC and let $\text{SimRel}_w(\mathcal{M})$ denote the weak simulation algorithm, which is obtained by replacing line 1.6 of $\text{SimRel}_s(\mathcal{M})$ in Algorithm 1 by: **if** $s_1 \precsim_R s_2$. The condition $s_1 \precsim_R s_2$ is checked in $\text{Ws}(\mathcal{M}, s_1, s_2, R)$, shown as Algorithm 8.

The first part of the algorithm is the preprocessing part. Line 8.1 tests for the case that s_1 could perform only *stutter* steps with respect to the current relation R. If line 8.5

$\text{Ws}(\mathcal{M}, s_1, s_2, R)$

8.1: **if** $post(s_1) \subseteq R^{-1}(s_2)$ **then**
8.2:   **return true**
8.3: **if** $post(s_2) \subseteq R(s_1)$ **then**
8.4: $U_1 \leftarrow \{s'_1 \in post(s_1) \mid s'_1 \notin R^{-1}(s_2)\}$
8.5:   **return** $(\forall u_1 \in U_1. \exists s \in post(reach(s_2) \cap R(s_1)). s \in R(u_1))$
8.6: Compute all of the breakpoints $b_1 < b_2 < \ldots < b_j$ of $\mathcal{N}(\gamma)$
8.7: **return** $(\exists i \in \{1, \ldots, j\}. b_i$ is valid for $\mathcal{N}(b_i))$

Algorithm 8: Algorithm to check whether $s_1 \precsim_R s_2$.

is reached, s_1 has at least one *visible* step, and all successors of s_2 can simulate s_1 up to the current relation R. In this case we need to check the reachability Condition 3 of Definition 2.21, which is performed in line 8.5. Recall that $reach(s)$ denotes the set of states that are reachable from s with positive probability. If the algorithm does not terminate in the preprocessing part, the breakpoints of the network $\mathcal{N}(\gamma)$ are computed. Then, corresponding to Lemma 5.8, we check whether one of the breakpoints is valid. We show the correctness of the algorithm Ws:

Lemma 5.9. *The algorithm* $\text{Ws}(\mathcal{M}, s_1, s_2, R)$ *returns true iff* $s_1 \precsim_R s_2$.

Proof. We first show the *only if* direction. Assume that $\text{Ws}(\mathcal{M}, s_1, s_2, R)$ returns true. We consider three possible cases:

- The algorithm returns true at line 8.2. For this case we have that $post(s_1) \subseteq R^{-1}(s_2)$. We choose $U_1 = \emptyset, V_1 = post(s_1)$, $U_2 = post(s_2)$ and $V_2 = \emptyset$ to fulfill the conditions in Definition 2.21. Hence, $s_1 \precsim_R s_2$.
- The algorithm returns true at line 8.5. If the algorithm reaches line 8.5, the following conditions hold: there exists a state $s'_1 \in post(s_1)$ such that $s'_1 \notin R^{-1}(s_2)$ (line 8.1), and $post(s_2) \subseteq R(s_1)$ (line 8.3). Let $U_1 = \{s'_1 \in post(s_1) \mid s'_1 \notin R^{-1}(s_2)\}$, and define δ_i by: $\delta_1(s) = 1$ if $s \in U_1$, and 0 otherwise, $\delta_2(s) = 0$ for all $s \in S$. By construction, to show $s_1 \precsim_R s_2$ we only need to show the reachability condition. Since the algorithm returns true at line 8.5, it holds that for each $u_1 \in U_1$, there exists $s \in post(reach(s_2) \cap R(s_1))$ such that $s \in R(u_1)$. This is exactly the reachability condition required by weak simulation up to R, thus $s_1 \precsim_R s_2$.
- The algorithm returns true at line 8.7. Thus, there exists breakpoint b_i which is valid for $\mathcal{N}(b_i)$. Then, there exists a state $s'_1 \in post(s_1)$ such that $s'_1 \notin R^{-1}(s_2)$, and $s'_2 \in post(s_2)$ such that $s'_2 \notin R(s_1)$. By Lemma 5.2, we have that $s_1 \precsim_R s_2$.

Now we show the *if* direction. Assume that Ws returns false. It is sufficient to show that $s_1 \not\precsim_R s_2$. We consider two cases:

- The algorithm returns false at line 8.5. This implies that there exists a state $s'_1 \in post(s_1)$ such that $s'_1 \notin R^{-1}(s_2)$ (line 8.1), and $post(s_2) \subseteq R(s_1)$ (line 8.3). All states $s'_1 \in post(s_1)$ with $s'_1 \notin R^{-1}(s_2)$ must be put into U_1. However, since the algorithm returns false at line 8.5, it holds that there exists a state $u_1 \in U_1$, such that there does not exist $s \in post(reach(s_2) \cap R(s_1))$ with $s \in R(u_1)$. Thus the reachability condition of Definition 2.21 is violated which implies that $s_1 \not\precsim_R s_2$.
- The algorithm returns false at line 8.7. Then, there exists a state $s'_1 \in post(s_1)$ such that $s'_1 \notin R^{-1}(s_2)$, and $s'_2 \in post(s_2)$ such that $s'_2 \notin R(s_1)$. Moreover, for all breakpoints b of
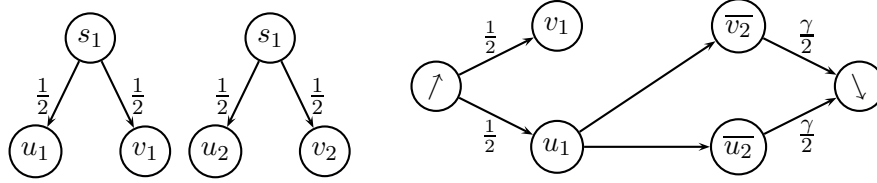
Figure 13: A simple DTMC for illustrating that not all maximum flows are valid.

$\mathcal{N}(\gamma)$, b is not valid for $\mathcal{N}(b)$. By Lemma 5.8, there does not exist a valid γ for $\mathcal{N}(\gamma)$. By Lemma 5.2, we have that $s_1 \not\precsim_R s_2$. $\square$

Now we state the correctness of the algorithm SIMREL_w for DTMCs:

Theorem 5.10. *If $\text{SIMREL}_w(\mathcal{M})$ terminates, the returned relation R equals $\precsim_{\mathcal{M}}$.*

Proof. The proof follows exactly the lines of the proof of Theorem 4.2. Let iteration k be the last iteration of Ws. Then, by Lemma 5.9, for each pair $(s_1, s_2) \in R_k$, it holds that s_2 weakly simulates s_1 up to R_k, so R_k is a weak simulation. On the other hand, one can prove by induction that each R_i is coarser than $\precsim$. $\square$

Complexity. For $(s_1, s_2) \in R$, we have shown that to check whether $s_1 \precsim_R s_2$ we could first compute the breakpoints of $\mathcal{N}(\gamma)$, then solve $\mathcal{O}(|V|)$ many maximum flow problems. To achieve a better bound, we first prove that applying a binary search method over the breakpoints, we only need to consider $\mathcal{O}(\log |V|)$ breakpoints, and thus solve $\mathcal{O}(\log |V|)$ maximum flow problems.

Assume that the sets MU_i and PV_i for $i = 1, 2$ are constructed as before for $\mathcal{N}(\gamma)$. Recall that a flow function f of $\mathcal{N}(\gamma)$ is valid for $\mathcal{N}(\gamma)$ iff f saturates all edges $(\nearrow, u_1)$ with $u_1 \in MU_1$ and all edges $(\overline{u_2}, \searrow)$ with $u_2 \in MU_2$. If f is also a maximum flow, we say that f is a valid maximum flow of $\mathcal{N}(\gamma)$. We first reformulate Lemma 5.2 using maximum flow.

Lemma 5.11. *There exists a valid flow f for $\mathcal{N}(\gamma)$ iff there exists a valid maximum flow f_m for $\mathcal{N}(\gamma)$.*

Proof. Assume there exists a valid flow f for $\mathcal{N}(\gamma)$. We let $\mathcal{N}_f(\gamma)$ denote the residual network. We use the augmenting path algorithm to get a maximum flow f' in the residual network $\mathcal{N}_f(\gamma)$. Assume that the augmenting path contains no cycles, which is a harmless restriction. Let $f_m = f + f'$. Obviously, f_m is a maximum flow for $\mathcal{N}(\gamma)$, and it saturates all of the edges saturated by f. Hence, f_m is valid for $\mathcal{N}(\gamma)$. The other direction is simple, since a valid maximum flow is also a valid flow for $\mathcal{N}(\gamma)$. $\square$

We first discuss how to get a valid maximum flow provided that γ is valid. Observe that even if γ is valid for $\mathcal{N}(\gamma)$, not all maximum flows for $\mathcal{N}(\gamma)$ are necessarily valid. Consider the DTMC on the left part of Figure 13. Assume that the relation R is given by $\{(s_1, s_2), (s_1, v_2), (v_1, s_2), (u_1, u_2), (u_1, v_2)\}$ and consider the pair (s_1, s_2). Thus, we have that $PV_1 = \{v_1\}, MU_1 = \{u_1\}, PV_2 = \{v_2\}, MU_2 = \{u_2\}$. The network $\mathcal{N}(\gamma)$ is depicted on the right part of Figure 13. The maximum flow f for $\mathcal{N}(1)$ has value 0.5. If f contains positive sub-flow along the path $\nearrow, u_1, \overline{v_2}, \searrow$, it does not saturate the edge $(\overline{u_2}, \searrow)$. On the contrary, the flow along the single path $\nearrow, u_1, \overline{u_2}, \searrow$ with value 0.5 would be a valid maximum flow. This example gives us the intuition to use the augmenting path through edges between MU_1 and $\overline{MU_2}$ as much as possible. For this purpose we define a cost function *cost* from

edges in $\mathcal{N}(\gamma)$ to real numbers as follows: $cost(u_1, \overline{u_2}) = 2$ for $u_1 \in MU_1$ and $u_2 \in MU_2$, $cost(u_1, \overline{v_2}) = 1$ for $u_1 \in MU_1$ and $v_2 \in PV_2$, $cost(v_1, \overline{u_2}) = 1$ for $v_1 \in PV_1$ and $u_2 \in MU_2$, $cost(s, s') = 0$ otherwise. The costs of edges starting from source, or ending at sink, or in $PV_1 \times \overline{PV_2}$ are 0. The cost of a flow f is defined by $cost(f) = \sum_{e \in E} f(e)cost(e)$. By definition of the cost function, we have the property $cost(f) = f(\nearrow, MU_1) + f(\overline{MU_2}, \searrow)$, i.e., the cost equals the sum of the amount of flow from $\nearrow$ into MU_1 and from $\overline{MU_2}$ into $\searrow$.

Lemma 5.12. *Assume that $\gamma > 0$ is valid for $\mathcal{N}(\gamma)$. Let f_γ denote a maximum flow over $\mathcal{N}(\gamma)$ with maximum cost. Then, f_γ is valid for $\mathcal{N}(\gamma)$.*

Proof. By Lemma 5.11, provided γ is valid for $\mathcal{N}(\gamma)$, there exists a valid maximum flow function g for $\mathcal{N}(\gamma)$. Since g saturates edges to MU_1 and from MU_2, obviously, $cost(g) = \mathbf{P}(s_1, MU_1) + \gamma \mathbf{P}(s_2, MU_2)$. Assume that f_γ is not valid, which indicates that f_γ does not saturate an edge $(\nearrow, u_1)$ with $u_1 \in MU_1$ or an edge $(\overline{u_2}, \searrow)$ with $u_2 \in \overline{MU_2}$. Then, $cost(f_\gamma) = f(\nearrow, MU_1) + f(\overline{MU_2}, \searrow) < \mathbf{P}(s_1, MU_1) + \gamma \mathbf{P}(s_2, MU_2) = cost(g)$. This contradicts the assumption that f_γ has maximum cost. $\square$

Let $\mathcal{N}_U(\gamma)$ denote the subnetwork of $\mathcal{N}(\gamma)$ where the set of vertices is restricted to $MU_1, \overline{MU_2}$ and $\{\nearrow, \searrow\}$. The following lemma discusses how to construct a maximum flow with maximum cost.

Lemma 5.13. *Assume that f^* is an arbitrary maximum flow of $\mathcal{N}_U(\gamma)$ and $\tilde{f}$ is an arbitrary maximum flow in the residual network $\mathcal{N}_{f^*}(\gamma)$ with the residual edges from MU_1 back to $\nearrow$ removed, as well as the residual edges from $\searrow$ back to $\overline{MU_2}$. Then $f_\gamma = f^* + \tilde{f}$ is a maximum flow over $\mathcal{N}(\gamma)$ with maximum cost.*

Proof. Recall that the cost of f_γ is equal to $cost(f_\gamma) = f_\gamma(\nearrow, MU_1) + f_\gamma(\overline{MU_2}, \searrow)$. Assume that $cost(f_\gamma)$ is not maximal for the sake of contradiction. Let f be a maximum flow such that $cost(f_\gamma) < cost(f)$. Without loss of generality, we assume that $f_\gamma(\nearrow, MU_1) < f(\nearrow, MU_1)$. It holds that $f_\gamma = f^* + \tilde{f}$ where f^* is a maximum flow of $\mathcal{N}_U(\gamma)$, and $\tilde{f}$ is a maximum flow in the residual network $\mathcal{N}_{f^*}(\gamma)$ with the residual edges from MU_1 back to $\nearrow$ removed, as well as the residual edges from $\searrow$ back to $\overline{MU_2}$. On the one hand, f^* sends as much flow as possible along MU_1 in $\mathcal{N}_U(\gamma)$. Since in the residual network $\mathcal{N}_{f^*}(\gamma)$ edges from MU_1 back to $\nearrow$ are removed, this guarantees that no flow can be sent back to $\nearrow$ from MU_1. On the other hand, $\tilde{f}$ sends as much flow as possible from MU_1 to $\overline{PV_2}$ (and also from PV_1 to $\overline{MU_2}$) in $\mathcal{N}_{f^*}(\gamma)$. Thus, $f_\gamma(\nearrow, MU_1)$ must be maximal which contradicts the assumption $f_\gamma(\nearrow, MU_1) < f(\nearrow, MU_1)$. $\square$

Assume that γ^* is not valid. The following lemma determines, provided a valid γ exists, whether it is greater or smaller than γ^*:

Lemma 5.14. *Let $\gamma^* \in [0, \infty)$, and let f be a maximum flow function with maximum cost for $\mathcal{N}(\gamma^*)$, as described in Lemma 5.13. Then,*

(1) *If f saturates all edges $(\nearrow, u_1)$ with $u_1 \in MU_1$ and $(\overline{u_2}, \searrow)$ with $u_2 \in MU_2$, γ^* is valid for $\mathcal{N}(\gamma^*)$.*

(2) *Assume that $\exists u_1 \in MU_1$ such that $(\nearrow, u_1)$ is not saturated by f, and all edges $(\overline{u_2}, \searrow)$ with $u_2 \in MU_2$ are saturated by f. Then, γ^* is not valid. If there exists a valid γ, $\gamma > \gamma^*$.*

(3) *Assume that all edges $(\nearrow, u_1)$ with $u_1 \in MU_1$ are saturated by f, and $\exists u_2 \in MU_2$ such that $(\overline{u_2}, \searrow)$ is not saturated by f. Then, γ^* is not valid. If there exists a valid γ, $\gamma < \gamma^*$.*

(4) *Assume that $\exists u_1 \in MU_1$ and $\exists u_2 \in MU_2$ such that $(\nearrow, u_1)$ and $(\overline{u_2}, \searrow)$ are not saturated by f. Then, there does not exist a valid γ.*

Proof. 1 : Follows directly from the definition. 2 : In this case, $f(\nearrow, u_1) < \mathbf{P}(s_1, u_1)$ for some $u_1 \in MU_1$. To saturate $(\nearrow, u_1)$, without un-saturating other edges from $\nearrow$ to MU_1, we have to increase the capacities of edges leading to $\searrow$, thus increase γ^*. 3 : Similar to the previous case. 4 : Combining 2 and 3.   □

According to the above lemma, we can use the binary search method over the breakpoints to check whether there exists a valid breakpoint for $\mathcal{N}(\gamma)$. Since there are at most $\mathcal{O}(|V|)$ breakpoints, we invoke the maximum flow algorithm at most $\mathcal{O}(\log |V|)$ times where $|V|$ is the number of vertices of $\mathcal{N}(\gamma)$.

Theorem 5.15. *The algorithm $\mathrm{SIMREL}_w(\mathcal{M})$ runs in time $\mathcal{O}(m^2 n^3)$ and in space $\mathcal{O}(n^2)$. If the fanout g is bounded by a constant, the time complexity is $\mathcal{O}(n^5)$.*

Proof. First, we consider a pair (s_1, s_2) out of the current relation R_i. Look at a single call of $\mathrm{WS}(\mathcal{M}, s_1, s_2, R_i)$. By saving the current relation sets R and R^{-1} in a two dimensional array, the conditions $s \in R(s')$ or $s \in R^{-1}(s')$ can be checked in constant time. Hence, line 8.1 takes time $|post(s_1)|$. To construct the set $reach(s)$ for a state s, BFS can be used, which has complexity $\mathcal{O}(m)$. The size of the set $reach(s)$ is bounded by n. Therefore, we need $\mathcal{O}(|post(s_1)| \, n)$ time at lines 8.3–8.5.

The algorithm computes all breakpoints of the network $\mathcal{N}(\gamma)$ (with respect to s_1, s_2 and R) using the breakpoint algorithm [18, p. 37–42]. Assume the set of vertices of $\mathcal{N}(\gamma)$ is partitioned into subsets V_1 and V_2 similar to the network described in Section 4.1. The number of edges of the network is at most $|E| := |V_1| |V_2| - 1$. Let $|V| := |V_1| + |V_2|$, and, without loss of generality, we assume that $|V_1| \leq |V_2|$. For our bipartite networks, the time complexity [18, p. 42] for computing the breakpoints is $\mathcal{O}(|V_1| |E| \log(\frac{|V_1|^2}{|E|} + 2))$ which can be simplified to $\mathcal{O}(|V_1|^2 |V_2|)$. Then, the binary search can be applied over all of the breakpoints to check whether at least one breakpoint is valid, for which we need to solve at most $\mathcal{O}(\log |V|)$ many maximum flow problems. For our network $\mathcal{N}(\gamma)$, the best known complexity[5] of the maximum flow problem is $\mathcal{O}(|V|^3 / \log |V|)$ [14]. As indicated in the proof of Lemma 4.4, the distance function is bounded by $4 |V_1|$ for our bipartite network. Applying this fact in the complexity analysis in [14], we get the corresponding complexity for computing maximum flow for bipartite networks $\mathcal{O}(|V_1| |V|^2 / \log |V|)$. Hence, the complexity for the $\mathcal{O}(\log |V|)$-invocations of the maximum flow algorithm is bounded by $\mathcal{O}(|V_1| |V|^2)$. As $|V| \leq 2 |V_2|$, the complexity is equal to $\mathcal{O}(|V_1| |V_2|^2)$. Summing over all (s_1, s_2) over all R_i, we get the overall complexity of $\mathrm{SIMREL}_w(\mathcal{M})$:

$$\sum_{i=1}^{k} \sum_{(s_1, s_2) \in R_i} (|post(s_1)| + m + |post(s_1)| \, n + |V_1| |V_2|^2) \leq 4knm^2 \qquad (5.1)$$

Recall that in the algorithm $\mathrm{SIMREL}_w(\mathcal{M})$, the number of iterations k is at most n^2. Hence, the time complexity amounts to $\mathcal{O}(m^2 n^3)$. The space complexity is $\mathcal{O}(n^2)$ because of the representation of R. If the fanout is bounded by a constant g, we have $m \leq gn$, and get the complexity $\mathcal{O}(n^5)$.   □

---

[5]For a network $G = (V, E)$ with small $|E|$, there are more efficient algorithms in [15] with complexity $\mathcal{O}(|V|^2 \sqrt{|E|})$, and in [28] with complexity $\mathcal{O}(|E| |V| + |V|^{2+\epsilon})$ where ϵ is an arbitrary constant. In our bipartite networks, however, $|E|$ is in the order of $|V|^2$. Hence, these complexities become $\mathcal{O}(|V|^3)$.

5.1.4. *An Improvement.* The algorithm $\mathrm{Ws}(\mathcal{M}, s_1, s_2, R)$ is dominated by the part in which all breakpoints ($\mathcal{O}(n)$ many) must be computed, and a binary search is applied to the breakpoints, with $\mathcal{O}(\log n)$ many feasible flow problems. In this section we discuss how to achieve an improved algorithm if the network $\mathcal{N}(\gamma)$ can be partitioned into sub-networks.

Let H denote the sub-relation $R \cap [(post(s_1) \cup \{s_1\}) \times (post(s_2) \cup \{s_2\})]$, which is the local fragment of the relation R. Now let $A_1, A_2, \ldots A_h$ enumerate the classes of the equivalence relation $(H \cup H^{-1})^*$ generated by H, where h denotes the number of classes. W. l. o. g., we assume in the following that A_h is the equivalence class containing s_1 and s_2, i. e., $s_1, s_2 \in A_h$. The following lemma gives some properties of the sets A_i provided that $s_1 \precsim_R s_2$:

Lemma 5.16. *For $(s_1, s_2) \in R$, assume that there exists a state $s_1' \in post(s_1)$ such that $s_1' \notin R^{-1}(s_2)$, and $s_2' \in post(s_2)$ such that $s_2' \notin R(s_1)$. Let $A_1, \ldots, A_h$ be the sets constructed for (s_1, s_2) as above. If $s_1 \precsim_R s_2$, the following hold:*

(1) $\mathbf{P}(s_1, A_i) > 0$ and $\mathbf{P}(s_2, A_i) > 0$ for all $i < h$,
(2) $\gamma_i = K_1/K_2$ for all $i < h$ where $\gamma_i = \mathbf{P}(s_1, A_i)/\mathbf{P}(s_2, A_i)$

Proof. Since $s_1 \precsim_R s_2$, we let $\delta_i, U_i, V_i, \Delta$ (for $I = 1, 2$) as described in Definition 2.21. Because of states s_1' and s_2', we have $K_1 > 0, K_2 > 0$. Let $post_i(s_j) = A_i \cap post(s_j)$ for $i = 1, \ldots, h$ and $j = 1, 2$. We prove the first part. For $i < h$, the set A_i does not contain s_1 nor s_2, but only (some of) their successors, so it is impossible that both $\mathbf{P}(s_1, A_i) = 0$ and $\mathbf{P}(s_2, A_i) = 0$. W. l. o. g., assume that $\mathbf{P}(s_1, A_i) > 0$. There exists $t \in post_i(s_1)$ such that $\mathbf{P}(s_1, t) > 0$. Obviously $\delta_1(t) = 1$, thus: $0 < \mathbf{P}(s_1, t) = (K_1 \Delta(t, U_2))/\delta_1(t) = K_1 \Delta(t, U_2)$ which implies that $\exists u_2 \in A_i$ with $\Delta(t, u_2) > 0$. Hence, $\mathbf{P}(s_2, u_2) = K_2 \Delta(U_1, u_2) \geq K_2 \Delta(t, u_2) > 0$. Now we prove the second part. It holds that:

$$\mathbf{P}(s_1, A_i) = \sum_{a_i \in A_i} \mathbf{P}(s_1, a_i) = \sum_{a_i \in post_i(s_1)} \mathbf{P}(s_1, a_i) \overset{(*)}{=} \sum_{a_i \in post_i(s_1)} \frac{K_1 \Delta(a_i, U_2)}{\delta_1(a_i)}$$

$$\overset{(!)}{=} K_1 \cdot \sum_{a_i \in post_i(s_1)} \Delta(a_i, U_2) \overset{(\dagger)}{=} K_1 \cdot \sum_{a_i \in post_i(s_1)} \Delta(a_i, A_i) = K_1 \cdot \sum_{a_i \in A_i} \Delta(a_i, A_i)$$

where $(*)$ follows from Condition 2b of Definition 2.21, $(!)$ follows from the equation $\delta_1(a_i) = 1$ for all $a_i \in post_i(a_1)$ with $i < n$, and $(\dagger)$ follows from the fact that if $a \in post_i(s_1)$, then $\Delta(a, b) = 0$ for $b \in U_2 \backslash post_i(s_2)$. In the same way, we get $\mathbf{P}(s_2, A_i) = K_2 \cdot \sum_{a_i \in A_i} \Delta(A_i, a_i)$. Therefore, $\gamma_i = K_1/K_2$ for $1 \leq i < h$. $\square$

For the case $h > 1$, the above lemma allows to check whether $s_1 \precsim_R s_2$ efficiently. For this case we replace lines 8.6–8.7 of Ws by the sub-algorithm WsImproved in Algorithm 9. The partition $A_1, \ldots, A_h$ is constructed in line 9.1. Lines 9.2–9.10 follow directly from Lemma 5.16: if $\gamma_i \neq \gamma_j$ for some $i, j < h$, we conclude from Lemma 5.16 that $s_1 \not\precsim_R s_2$. Line 9.11 follows from the following lemma, which is the counterpart of Lemma 5.2:

Lemma 5.17. *For $(s_1, s_2) \in R$, assume that there exists a state $s_1' \in post(s_1)$ such that $s_1' \notin R^{-1}(s_2)$, and $s_2' \in post(s_2)$ such that $s_2' \notin R(s_1)$. Assume that $h > 1$, and assume WsImproved$(\mathcal{M}, s_1, s_2, R)$ reaches line 9.11. Then, $s_1 \precsim_R s_2$ iff γ_1 is valid for $\mathcal{N}(\gamma_1)$.*

Proof. First, assume that $s_1 \precsim_R s_2$. According to Lemma 5.2, there exists a valid γ^* for $\mathcal{N}(\gamma^*)$. As in the proof of Lemma 5.2, $\gamma^* = K_1/K_2$ is valid for $\mathcal{N}(\gamma^*)$. If Ws reaches line 9.11, by Lemma 5.16, we have $\gamma_1 = K_1/K_2$, hence, γ_1 is valid for $\mathcal{N}(\gamma_1)$. The other direction follows directly from Lemma 5.2. $\square$

WSIMPROVED($\mathcal{M}, s_1, s_2, R$)

  9.1: Construct the partition $A_1, \ldots, A_h$ (* Assume that $h > 1$ *)
  9.2: **for all** $i \leftarrow 1, 2, \ldots h - 1$ **do**
  9.3:   **if** $\mathbf{P}(s_1, A_i) = \mathbf{P}(s_2, A_i) = 0$ **then**
  9.4:   **raise error**
  9.5:   **else if** $\mathbf{P}(s_1, A_i) = 0$ or $\mathbf{P}(s_2, A_i) = 0$ **then**
  9.6:   **return false**
  9.7:   **else**
  9.8: $\gamma_i \leftarrow \frac{\mathbf{P}(s_1, A_i)}{\mathbf{P}(s_2, A_i)}$
  9.9: **if** $\gamma_i \neq \gamma_j$ for some $i, j < h$ **then**
  9.10:   **return false**
  9.11: **return** (γ_1 is valid for $\mathcal{N}(\gamma_1)$)

Algorithm 9: Algorithm to check whether $s_1 \precsim_R s_2$ tailored to DTMCs.

Example 5.18. Consider the DTMC in Figure 8 together with the relation $R = \{(s_1, s_2), (s_1, v_2), (v_1, s_2), (u_1, u_2), (o_1, o_2), (o_1, v_2), (v_1, o_3), (v_1, v_2), (o_2, o_1)\}$. We obtain the relation $H = R \backslash \{(o_2, o_1)\}$. We get two partitions $A_1 = \{u_1, u_2\}$ and $A_2 = \{s_1, s_2, v_1, v_2, o_1, o_2, o_3\}$. In this case we have $h = 2$. Recall that $MU_1 = \{u_1, o_1\}$, $MU_2 = \{u_2, o_2, o_3\}$, $PV_1 = \{v_1\}$, $PV_2 = \{v_2\}$. We have $\mathbf{P}(s_1, A_1) = \frac{1}{4}$ and $\mathbf{P}(s_2, A_1) = \frac{1}{8}$. Hence, $\gamma_1 = \mathbf{P}(s_1, A_1)/\mathbf{P}(s_2, A_1) = 2$. As we have shown in Example 5.4, 2 is valid for the network $\mathcal{N}(2)$. Hence, $s_1 \precsim_R s_2$.

Assume that $(s_1, s_2) \in R_1$ such that $h > 1$ in the first iteration of SIMREL$_w$. We consider the set A_1 and let $\gamma_1 = \mathbf{P}(s_1, A_1)/\mathbf{P}(s_2, A_1)$. If A_1 is not split in the next iteration, γ_1 would not change, and hence, we can reuse the network constructed in the last iteration. Assume that in the next iteration A_1 is split into two sets A_1^a and A_1^b. There are two possibilities:

- either $\mathbf{P}(s_1, A_1^a)/\mathbf{P}(s_2, A_1^b) = \mathbf{P}(s_1, A_1^a)/\mathbf{P}(s_2, A_1^b)$. This implies that both of them are equal to γ_1. If all A_i are split like A_1, we just check whether γ_1 is valid for $\mathcal{N}(\gamma_1)$.
- or $\mathbf{P}(s_1, A_1^a)/\mathbf{P}(s_2, A_1^b) \neq \mathbf{P}(s_1, A_1^a)/\mathbf{P}(s_2, A_1^b)$. This case is simple, we conclude $s_1 \not\precsim_R s_2$ because of Lemma 5.16.

This indicates that once in the first iteration γ_1 is determined for (s_1, s_2), either it does not change throughout the iterations, or we conclude that $s_1 \not\precsim_R s_2$ directly. The above analysis can be generalised to the case in which A_1 is split into more than two sets. As the network $\mathcal{N}(\gamma_1)$ is fixed, we can apply an algorithm similar to SMF, which solves the maximum flow problems during all subsequent iterations using only one parametric maximum flow, as for strong simulation.

The above analysis implies that if $h > 1$ for all (s_1, s_2) in the initial R_1, we could even establish the time bound $\mathcal{O}(m^2 n)$, the same as for strong simulation. Since in the worst case it could be the case that $h = 1$ for all $(s_1, s_2) \in R$, the algorithm WSIMPROVED does not improve the worst case complexity.

Since the case that the network cannot be partitioned ($h = 1$) is the one that requires most of our attention, we suggest a heuristic approach that can reduce the number of occurrences of this case. We may choose to run some iterations incompletely (as long as the last iteration is run completely). If iteration i is incomplete, we first check for each pair $(s_1, s_2) \in R_i$ whether the corresponding h_i is greater than 1. If not, we skip the test and just add (s_1, s_2) to R_{i+1}. The intuition is that in the next complete iteration $i' > i$, for

each such pair (s_1, s_2) we hope to get $h_{i'} > 1$ because some other elements of R_i have been eliminated from it. We only perform the expensive computation if an incomplete iteration does no longer refine the relation.

5.2. **An Algorithm for CTMCs.** Let $\mathcal{M} = (S, \mathbf{R}, L)$ be a CTMC. We now discuss how to handle CTMCs. Recall that in Definition 2.23, we have the rate condition $3'$: $K_1 \mathbf{R}(s_1, S) \leq K_2 \mathbf{R}(s_2, S)$. To determine $\precsim_{\mathcal{M}}$, we simplify the algorithm for DTMCs. If $K_1 > 0$ and $K_2 = 0$, $s_1 \not\precsim_R s_2$ because of the rate condition. Hence, we do not need to check the reachability condition, and lines 8.3–8.5 of the algorithm $\mathrm{Ws}(\mathcal{M}, s_1, s_2, R)$ can be skipped. For states s_1, s_2 and relation R, we use $\mathcal{N}(\gamma)$ to denote the network defined in the embedded DTMC $emb(\mathcal{M})$. To check the additional rate condition we use the following lemma:

Lemma 5.19. *Let $s_1 \mathrel{R} s_2$. Assume that there exists $s_1' \in post(s_1)$ such that $s_1' \notin R^{-1}(s_2)$. Then, $s_1 \precsim_R s_2$ in $\mathcal{M}$ iff there exists $\gamma \leq \mathbf{R}(s_2, S)/\mathbf{R}(s_1, S)$ such that γ is valid for $\mathcal{N}(\gamma)$.*

Proof. Assume first $s_1 \precsim_R s_2$ in $\mathcal{M}$. Let $\delta_i, U_i, V_i, K_i, \Delta$ (for $i = 1, 2$) as described in Definition 2.23. Obviously, s_1' must be in U_1, implying that $K_1 > 0$. Because of the rate condition it holds that $K_1 \mathbf{R}(s_1, S) \leq K_2 \mathbf{R}(s_2, S)$, which implies that $K_2 > 0$. It is sufficient to show that $\gamma := K_1/K_2$ is valid for $\mathcal{N}(\gamma)$. Exactly as in the proof of Lemma 5.2 (the *only if* direction), we can construct a valid flow f for $\mathcal{N}(\gamma)$. Thus, γ is valid for $\mathcal{N}(\gamma)$ and $\gamma \leq \mathbf{R}(s_2, S)/\mathbf{R}(s_1, S)$.

Now we show the other direction. By assumption, γ is valid for $\mathcal{N}(\gamma)$. We may assume that there exists a valid flow function f for $\mathcal{N}(\gamma)$. We define $\delta_i, V_i, U_i, K_i, \Delta$ (for $i = 1, 2$) as in the proof (the *if* direction) of Lemma 5.2. Recall that s_1' must be in U_1, implying that $f(\nearrow, s_1') > 0$. Thus, there must be a node $\overline{s}$ in $\mathcal{N}(\gamma)$ with $f(\overline{s}, \searrow) > 0$, which implies that $s \in U_2$. Thus we have $K_2 > 0$. Using the proof (the *if* direction) of Lemma 5.2, it holds that $s_1 \precsim_R s_2$ in $emb(\mathcal{M})$, moreover, it holds that $\gamma = K_1/K_2$. By assumption it holds that $\gamma \leq \mathbf{R}(s_2, S)/\mathbf{R}(s_1, S)$ which is exactly the rate condition. $\qquad\square$

To check the rate condition for the case $h > 1$, we replace line 9.11 of the algorithm WsImproved by:

$$\textbf{return } (\gamma_1 \leq \gamma^* \wedge \gamma_1 \text{ is valid for } \mathcal{N}(\gamma_1))$$

where $\gamma^* = \mathbf{R}(s_2, S)/\mathbf{R}(s_1, S)$ can be computed directly. In case $h = 1$, we replace line 8.7 of Ws by:

$$\textbf{return } (\exists i \in \{1, \ldots, j\}.\ b_i \leq \gamma^* \wedge b_i \text{ is valid for } \mathcal{N}(b_i))$$

to check the rate condition. Or, equivalently, we can check whether the minimal valid breakpoint γ_m is smaller than or equal to γ^*. The binary search algorithm introduced for DTMCs can also be modified slightly to find the minimal valid breakpoint. The idea is that, if we find a valid breakpoint, we first save it, and then continue the binary search on the left side. If another breakpoint is valid, we save the smaller one. As the check for the reachability condition disappears for CTMCs, we get better bound for sparse CTMCs:

Theorem 5.20. *If the fanout g of $\mathcal{M}$ is bounded by a constant, the time complexity for CTMC is $\mathcal{O}(n^4)$.*

Proof. In the proof of Theorem 5.15 we have shown that Ws has complexity $\mathcal{O}(|V_1||V_2|^2)$. As we do not need to check the reachability condition, the overall complexity of the algorithm $\mathrm{SimRel}_w(\mathcal{M})$ (see Inequality 5.1) is $\sum_{s_1 \in S} \sum_{s_2 \in S} \sum_{i=1}^{l} \left(|post(s_1)| + |V_1||V_2|^2 \right)$

which is bounded by $2kgm^2$. Since k is bounded by n^2, the time complexity is bounded by $4gm^2n^2$. If g is a constant, we have $m \leq gn$, hence, the time complexity is $4g^3n^4 \in \mathcal{O}(n^4)$. $\square$

6. Conclusion and Future Work

In this paper we have proposed efficient algorithms deciding simulation on Markov models with complexity $\mathcal{O}(m^2n)$. For sparse models where the fanout is bounded by a constant, we achieve for strong simulation the complexity $\mathcal{O}(n^2)$ and for weak simulation $\mathcal{O}(n^4)$ on CTMCs and $\mathcal{O}(n^5)$ for DTMCs. We extended the algorithms for computing simulation preorders to handle PAs with the complexity $\mathcal{O}(m^2n)$ for strong simulation. For strong probabilistic simulation, we have shown that the preorder can be determined by solving LP problems. We also considered their continuous-time analogon, CPAs, and arrived at an algorithm with same complexities as for PAs.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[2] R. K. Ahuja, J. B. Orlin, C. Stein, and R. E. Tarjan. Improved algorithms for bipartite network flow. *SIAM J. Comput.*, 23(5):906–933, 1994.

[3] C. Baier, B. Engelen, and M. E. Majster-Cederbaum. Deciding bisimilarity and similarity for probabilistic processes. *J. Comput. Syst. Sci.*, 60(1):187–231, 2000.

[4] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Trans. Software Eng.*, 29(6):524–541, 2003.

[5] C. Baier and H. Hermanns. Weak bisimulation for fully probabilistic processes. In *Computer Aided Verification*, volume 1254 of *LNCS*, pages 119–130. Springer, 1997.

[6] C. Baier, H. Hermanns, and J.-P. Katoen. Probabilistic weak simulation is decidable in polynomial time. *Information processing letters*, 89(3):123–130, 2004.

[7] C. Baier, H. Hermanns, J.-P. Katoen, and B. R. Haverkort. Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. *Theor. Comput. Sci.*, 345(1):2–26, 2005.

[8] C. Baier, J.-P. Katoen, H. Hermanns, and B. Haverkort. Simulation for continuous-time Markov chains. In *Concurrency*, volume 2421 of *LNCS*, pages 338–354. Springer, 2002.

[9] C. Baier, J.-P. Katoen, H. Hermanns, and V. Wolf. Comparative branching-time semantics for Markov chains. *Information and computation*, 200(2):149–214, 2005.

[10] B. Bloom and R. Paige. Transformational design and implementation of a new efficient solution to the ready simulation problem. *Sci. Comput. Program.*, 24(3):189–220, 1995.

[11] H. C. Bohnenkamp, P. van der Stok, H. Hermanns, and F. W. Vaandrager. Cost-optimization of the IPv4 zeroconf protocol. In *Dependable Systems and Networks*, pages 531–540. IEEE Computer Society, 2003.

[12] S. Cattani and R. Segala. Decision algorithms for probabilistic bisimulation. In *Concurrency*, volume 2421 of *LNCS*, pages 371–385. Springer, 2002.

[13] K. Chatterjee, L. de Alfaro, R. Majumdar, and V. Raman. Algorithms for game metrics (full version). In *Foundations of Software Technology and Theoretical Computer Science*, 2008.

[14] J. Cheriyan, T. Hagerup, and K. Mehlhorn. Can a maximum flow be computed in $\mathcal{O}(nm)$ time? In *Automata, languages and programming*, volume 443 of *LNCS*, pages 235–248. Springer, 1990.

[15] J. Cheriyan and K. Mehlhorn. An analysis of the highest-level selection rule in the preflow-push maxflow. *Inf. Process. Lett.*, 69(5):239–242, 1999.

[16] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

[17] S. Derisavi, H. Hermanns, and W. H. Sanders. Optimal State-Space Lumping in Markov Chains. *Inf. Process. Lett.*, 87(6):309–315, 2003.

[18] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.*, 18(1):30–55, 1989.

[19] A. V. Goldberg. Recent developments in maximum flow algorithms. In *Scandinavian Workshop on Algorithm Theory*, volume 1432 of *LNCS*, pages 1–10. Springer, 1998.

[20] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, 1988.

[21] C. Groß, H. Hermanns, and R. Pulungan. Does clock precision influence zigbee's energy consumptions? In *International Conference on Principles of Distributed Systems*, volume 4878 of *LNCS*, pages 174–188. Springer, 2007.

[22] D. Gusfield and É. Tardos. A faster parametric minimum-cut algorithm. *Algorithmica*, 11(3):278–290, 1994.

[23] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *Foundations of Computer Science*, pages 453–462. IEEE Computer Society, 1995.

[24] H. Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*, volume 2428 of *LNCS*. Springer, 2002.

[25] D. T. Huynh and L. Tian. On some equivalence relations for probabilistic processes. *Fundam. Inform.*, 17(3):211–234, 1992.

[26] B. Jonsson and K. G. Larsen. Specification and refinement of probabilistic processes. In *Logic in Computer Science*, pages 266–277. IEEE Computer Society, 1991.

[27] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–396, 1984.

[28] V. King, S. Rao, and R. E. Tarjan. A faster deterministic maximum flow algorithm. *J. Algorithms*, 17(3):447–474, 1994.

[29] R. Knast. Continuous-time probabilistic automata. *Information and Control*, 15(4):335–352, 1969.

[30] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[31] S. Mangold, Z. Zhong, G. R. Hiertz, and B. Walke. IEEE 802.11e/802.11k wireless LAN: spectrum awareness for distributed resource sharing. *Wireless Communications and Mobile Computing*, 4(8):881–902, 2004.

[32] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. Modelling with generalized stochastic petri nets. *SIGMETRICS Performance Evaluation Review*, 26(2):2, 1998.

[33] R. Milner. An algebraic definition of simulation between programs. In *IJCAI*, pages 481–489, 1971.

[34] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.

[35] D. Park. Concurrency and automata on infinite sequences. In *Theor. Comput. Sci.*, pages 167–183, 1981.

[36] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York, 1994.

[37] W. H. Sanders and J. F. Meyer. Reduced base model construction methods for stochastic activity networks. *IEEE Journal on Selected Areas in Communications*, 9(1):25–36, 1991.

[38] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.

[39] R. Segala. *Modeling and Verification of Randomized Distributed Realtime Systems*. PhD thesis, MIT, 1995.

[40] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[41] W. J. Steward. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

[42] N. Wolovick and S. Johr. A characterization of meaningful schedulers for continuous-time Markov decision processes. In *Formal Modeling and Analysis of Timed Systems*, volume 4202 of *LNCS*, pages 352–367. Springer, 2006.

[43] L. Zhang. *Decision Algorithms for Probabilistic Simulations*. PhD thesis, Universität des Saarlandes, 2008.

[44] L. Zhang and H. Hermanns. Deciding simulations on probabilistic automata. In *Automated Technology for Verification and Analysis*, volume 4762 of *LNCS*, pages 207–222. Springer, 2007.

[45] L. Zhang, H. Hermanns, F. Eisenbrand, and D. N. Jansen. Flow faster: Efficient decision algorithms for probabilistic simulations. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *LNCS*, pages 155–169. Springer, 2007.