Under consideration for publication in Formal Aspects of Computing

Deciding Probabilistic Automata Weak Bisimulation: Theory and Practice

Luis María Ferrer Fioriti[1], Vahid Hashemi[1,2], Holger Hermanns[1], and Andrea Turrini[3]

[1]Department of Computer Science, Saarland University, Saarbrücken, Germany
[2]Max Planck Institute for Informatics, Saarbrücken, Germany
[3]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

Abstract. Weak probabilistic bisimulation on probabilistic automata can be decided by an algorithm that needs to check a polynomial number of linear programming problems encoding weak transitions. It is hence of polynomial complexity. This paper discusses the specific complexity class of the weak probabilistic bisimulation problem, and it considers several practical algorithms and Linear Programming problem transformations that enable an efficient solution. We then discuss two different implementations of a probabilistic automata weak probabilistic bisimulation minimizer, one of them employing SAT modulo linear arithmetic as the solver technology. Empirical results demonstrate the effectiveness of the minimization approach on standard benchmarks, also highlighting the benefits of compositional minimization.

Keywords: Complexity; Compositional Analysis; Concurrency; Efficiency; Linear Programming; Probabilistic Automata; Satisfiability Modulo Theories; Weak Bisimulation

1. Introduction

Probability and nondeterminism are core aspects of concurrent systems. *Probability* for instance arises when a system, performing an action, is able to switch to more than one state and the likelihood of each of these states can be faithfully estimated. Probability can model both specific system choices (such as flipping a coin, commonly used in randomized distributed algorithms) and general system properties (such as message loss probabilities when sending a message over a wireless medium). *Nondeterminism* represents behaviours that we can not or do not want to attach a precise (possibly probabilistic) interpretation to. This might reflect the concurrent execution of several components at unknown (relative) speeds or behaviours we decide to keep undetermined for simplifying the system model or allowing for different implementations.

Several models have been proposed in the literature to study formally systems where a combination of

Correspondence and offprint requests to: Andrea Turrini. e-mail: turrini@ios.ac.cn probability and nondeterminism is considered: among others, there are Markov Decision Processes (*MDP*) [Der70], Labelled Concurrent Markov Chains (*LCMC*), Alternating Probabilistic Models [Var85, Han91, PLS00], and Probabilistic Automata (*PA*) [Seg95].

Probabilistic automata extend classical concurrency models in a simple yet conservative fashion. In probabilistic automata, there is no global notion of time, and concurrent processes may perform probabilistic experiments inside a transition. This is represented by transitions of the form $s \xrightarrow{a} \mu$, where s is a state, a is an action label, and μ is a probability measure on states. Labelled transition systems are instances of this model family, obtained by restricting to Dirac measures (assigning full probability to single states). Moreover, foundational concepts and results of standard concurrency theory are retained in full and extend smoothly to the *PA* model. Since the *PA* model is akin to the *MDP* model, its fundamental beauty can be paired with powerful model checking techniques, as implemented for instance in the PRISM tool [KNP11]. We refer the interested reader to [Seg06] for a survey on this and other models.

Given a real system, we can conceive several different probabilistic automata models to reflect its behavior. *Bisimulation relations* provide a powerful tool to check whether two models describe essentially the same system. They are then called bisimilar. The bisimilarity of two systems can be viewed in terms of a game played between a challenger and a defender. In each step of the possibly infinite bisimulation game, the challenger chooses one automaton, makes a step, and the defender matches it with a step of the other automaton. Depending on how we want to treat internal computations, this leads to *strong* and *weak* bisimulations: the former requires that each single step of the challenger automaton is matched by an equally labelled single step of the defender automaton, the latter allows the matching up to internal computation steps. On the other hand, depending on how nondeterminism is resolved, probabilistic bisimulations can be varied by allowing the defender to match the challenger's step by a convex combination of enabled probabilistic transitions. This results in a spectrum of four bisimulations: strong [Seg95, Han91, Var85], strong probabilistic [Seg95], weak [PLS00, Seg95, EHZ10a, EHZ10b], and weak probabilistic [Seg95, EHZ10b, EHZ10a] bisimulations. For a recent survey on behavioral equivalences and preorders, we refer the interested reader to [GHT14].

Besides comparing automata, bisimulation relations allow us to reduce the size of an automaton without changing its properties (i.e., with respect to logic formulae satisfied by it). This is particularly useful to alleviate the state explosion problem notoriously encountered in model checking. If the bisimulation is a congruence with respect to the operators of a process calculus used to build up the automata out of smaller ones, this can give rise to a compositional strategy to associate a small automaton model to a large system without intermediate state space explosion. In several related settings, this strategy has been proven very effective [CGM$^+$96, HK00, KKZJ07, BHH$^+$09, CHLS09]; it can speed up the overall model analysis or turn a too large problem into a tractable one. Both, strong and weak bisimilarity are used in practice, with weaker relations leading to greater reduction. However, this approach has thus far not been explored in the context of *MDP*s or probabilistic automata. A striking reason is that until recently no effective decision algorithm was at hand for weak probabilistic bisimilarity on *PA*. A polynomial time decision algorithm has been proposed only recently [TH15], based on linear programming problems. That algorithm can be embedded into a procedure to compress a given *PA* to its canonical minimal representative [EHS$^+$13]. Since weak probabilistic bisimilarity is a congruence for parallel composition and hiding operators on *PA*s (we refer the interested reader to [Seg95, SL95] for more details), this paves the way for compositional strategies to associate a small *PA* model to a large system without intermediate state space explosion.

The weak bisimilarity decision algorithm follows the standard partition refinement approach [KS90, PT87, PLS00, CS02], and thereby induces a polynomial number of linear programming problems that can be solved in polynomial time [Kar84, Kha79]. In this paper, we discuss the efficiency of solving the specific LP problems from both theoretical and practical viewpoints. We first consider the theoretical efficiency of solving the problem. We first look at rational *PA*s, i.e, *PA*s with only rational probability values, and study the complexity of the decision problem together with several optimizations. This entails reformulating the original LP problem [TH15] in order to simplify the construction of the dual LP problem [BT97] which is smaller in size than the original. By using a state-of-the-art preconditioned conjugate gradient (PCG) algorithm combined with a partial updating procedure [Ans99] that dual LP problem can be solved efficiently. On the other hand, taking advantage of the small-sized dual LP problem, we give an upper bound on the complexity of checking the feasibility of the original LP problem.

We also discuss how the efficiency of solving the decision problem can exploit the problem structure. In practice one would usually opt for the notoriously efficient simplex method [Sha87] to solve the LP problems. But a small modification of the underlying network [TH15] enables us to adapt the corresponding LP problem into a variant of a minimum cost flow problem [AMO93] with flow proportional sets. This is a special class of linear programming problems where the underlying network structure can be exploited, in particular if it is sparse. Sparsity is indeed frequently observed in practical applications of probabilistic automata. We therefore compare the simplex method with a very efficient state-of-the-art network simplex algorithm [BF12] specialized for the minimum cost flow problem with additional side constraints. This is known to outperform the simplex method [MSJ11, HK95, Cal02] when the number of nodes is an order of magnitude larger than the number of side constraints.

We furthermore discuss different implementations of the decision algorithm, focusing on effective minimization of *PA* with respect to weak probabilistic bisimilarity. One of the implementations exploits that the problem at hand can be encoded into SAT modulo linear arithmetic. We report on extensive empirical investigations in the context of concurrent probabilistic systems. It turns out that minimization can be applied effectively to standard *PA* benchmarks. Several techniques and heuristics are discussed to further reduce the actual execution time of the algorithm, by showing how an accurate management of transition computation and minimization helps in the reduction of large automata, in particular when they are the result of the composition of several automata. The problem of efficiently deciding bisimilarities for *PA*s and *MDP*s is of pivotal importance for compositional construction and minimization techniques for complex probabilistic models. Once in place, these techniques can be rolled out to operations research, automated planning, and decision support applications.

This article is a revised and extended version of [HHT13]. Implementation considerations, case studies, and empirical results have not been published before.

Organization of the paper After the preliminaries in Section 2, we present in Section 3 the probabilistic automata model and the weak probabilistic bisimulation. Then, in Section 4, we show how to compute the weak probabilistic bisimulation and how to minimize an automaton. We devote Section 5 to the LP problem construction and in Section 6 we focus on the efficiency of solving the LP problem. Section 7 presents implementation considerations together with several cases studies showing the effectiveness of the minimization in particular for compositional analysis. Section 8 concludes the paper.

2. Mathematical Preliminaries

We now recall the basic mathematical preliminaries together with the notational conventions we adhere to in this work.

Given two sets X and Y, denote by $X \uplus Y$ the *disjoint union* of X and Y.

A *σ-field* over a set X is a set $\mathcal{F} \subseteq 2^X$ that includes X and is closed under complement and countable union. A *measurable space* is a pair $(X, \mathcal{F})$ where X is a set, also called the *sample space*, and $\mathcal{F}$ is a σ-field over X. A measurable space $(X, \mathcal{F})$ is called *discrete* if $\mathcal{F} = 2^X$. A *measure* over a measurable space $(X, \mathcal{F})$ is a function $\rho \colon \mathcal{F} \to \mathbb{R}^{\geq 0}$ such that, for each countable collection $\{X_i\}_{i \in I}$ of pairwise disjoint elements of $\mathcal{F}$, $\rho(\cup_{i \in I} X_i) = \sum_{i \in I} \rho(X_i)$. A *probability measure* over a measurable space $(X, \mathcal{F})$ is a measure ρ over $(X, \mathcal{F})$ such that $\rho(X) = 1$. A *sub-probability measure* over $(X, \mathcal{F})$ is a measure over $(X, \mathcal{F})$ such that $\rho(X) \leq 1$. A measure over a discrete measurable space $(X, 2^X)$ is called a *discrete measure* over X. The *support* of a measure ρ over $(X, \mathcal{F})$, denoted by $\mathrm{Supp}(\mu)$, is the set $\{\, x \in X \mid \mu(x) > 0 \,\}$. To simplify the notation, we may write $\rho(x)$ instead of $\rho(\{x\})$, for $x \in X$.

Given a set X, denote by $\mathrm{Disc}(X)$ the set of discrete probability measures over X, and by $\mathrm{SubDisc}(X)$ the set of discrete sub-probability measures over X. Given a discrete sub-probability measure ρ of $\mathrm{SubDisc}(X)$, denote by $\rho(\perp)$ the value $1 - \rho(X)$. For a discrete sub-probability measure ρ, we also write $\rho = \{\, (x, p_x) \mid x \in X \,\}$ where p_x is the measure $\rho(x)$ of x. We call a discrete (sub-)probability measure $\rho \in \mathrm{SubDisc}(X)$ a *uniform* measure on a set $\emptyset \neq Y \subseteq X$, denoted by υ_Y, if $\upsilon_Y(y) = \frac{1}{|Y|}$ for each $y \in Y$. We call a discrete (sub-)probability measure a *Dirac* measure if it assigns measure 1 to exactly one object $x \in X$ (denote this measure by δ_x), that is, $\delta_x(y) = 1$ if $y = x$, 0 otherwise. We also call Dirac a discrete sub-probability measure that assigns measure 0 to all objects, and we denote it by $\delta_\perp$. Given $\rho \in \mathrm{SubDisc}(X)$, we denote by $\rho \backslash z$ the *z-conditional* sub-probability measure such that $\rho \backslash z(x) = 0$ if $x = z$ and $\rho \backslash z(x) = \frac{\rho(x)}{\rho(X \setminus \{z\})}$ otherwise, provided that $\rho(X \setminus \{z\}) \neq 0$. Given $\rho_x \in \mathrm{SubDisc}(X)$ and $\rho_y \in \mathrm{SubDisc}(Y)$, we denote by $\rho_x \times \rho_y$ the sub-probability measure over $X \times Y$ defined by $\rho_x \times \rho_y(u, v) = \rho_x(u) \cdot \rho_y(v)$ for each $(u, v) \in X \times Y$. Given a finite set I of indexes, a family $\{p_i \in \mathbb{R}^{>0}\}_{i \in I}$ such that $\sum_{i \in I} p_i = 1$, and a family $\{\rho_i \in \mathrm{SubDisc}(X)\}_{i \in I}$,

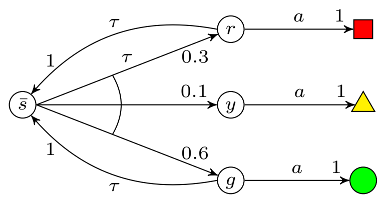


Figure 1. An example of *PA*s: the *PA* $\mathcal{E}$

we say that ρ is the *convex combination* of $\{\rho_i\}_{i \in I}$ according to $\{p_i\}_{i \in I}$, denoted by $\sum_{i \in I} p_i \cdot \rho_i$, if, for each $x \in X$, $\rho(x) = \sum_{i \in I} p_i \cdot \rho_i(x)$.

Given a relation $\mathcal{R} \subseteq X \times Y$ and $x \in X$, we denote by $\mathcal{R}(x)$ the set of elements of Y related to x, i.e., $\mathcal{R}(x) = \{\, y \in Y \mid x \mathrel{\mathcal{R}} y \,\}$ and we call $\mathcal{R}(x)$ the *relation set* of x.

Given an equivalence relation $\mathcal{R}$ on X, we denote by $X/\mathcal{R}$ the set of equivalence classes induced by $\mathcal{R}$ and, for $x \in X$, by $[x]_{\mathcal{R}}$ the class $\mathcal{C} \in X/\mathcal{R}$ such that $x \in \mathcal{C}$. We denote by $\mathcal{I}$ the *identity relation*, i.e., the equivalence relation having $[x]_{\mathcal{I}} = \{x\}$ for each $x \in X$.

The lifting [JL91] of a relation $\mathcal{R} \subseteq X \times Y$ to a relation $\mathcal{L}(\mathcal{R}) \subseteq \mathrm{Disc}(X) \times \mathrm{Disc}(Y)$ is defined as follows: for $\rho_X \in \mathrm{Disc}(X)$ and $\rho_Y \in \mathrm{Disc}(Y)$, $\rho_X \mathrel{\mathcal{L}(\mathcal{R})} \rho_Y$ holds if there exists a *weighting function* $\omega \colon X \times Y \to [0, 1]$ such that

- $\omega(x, y) > 0$ implies $x \mathrel{\mathcal{R}} y$,
- $\sum_{y \in Y} \omega(x, y) = \rho_X(x)$, and
- $\sum_{x \in X} \omega(x, y) = \rho_Y(y)$.

When $\mathcal{R}$ is an equivalence relation on a set X, $\rho_1 \mathrel{\mathcal{L}(\mathcal{R})} \rho_2$ holds if, for each $\mathcal{C} \in X/\mathcal{R}$, $\rho_1(\mathcal{C}) = \rho_2(\mathcal{C})$. In particular, when $\mathcal{R} = \mathcal{I}$, $\rho_1 \mathrel{\mathcal{L}(\mathcal{I})} \rho_2$ holds if and only if $\rho_1 = \rho_2$. This property can be generalized to: if $\mathcal{R} \cap \mathrm{Supp}(\rho_1) \times \mathrm{Supp}(\rho_2) \subseteq \mathcal{I}$, then $\rho_1 \mathrel{\mathcal{L}(\mathcal{R})} \rho_2$ holds if and only if $\rho_1 = \rho_2$.

3. Probabilistic Automata

We now recall the main parts of the probabilistic automata framework [Seg95] we use in this paper, following the notation of [Seg06]. Note that the probabilistic automata we use here correspond to the *simple* probabilistic automata of [Seg95].

Definition 1. A *probabilistic automaton* (*PA*) is a tuple $\mathcal{A} = (S, \bar{s}, \Sigma, T)$, where S is a set of *states*, $\bar{s} \in S$ is the *start* state, Σ is the set of *actions*, and $T \subseteq S \times \Sigma \times \mathrm{Disc}(S)$ is a *probabilistic transition relation.*

The start state is also called the *initial* state.

The set Σ is divided in two disjoint sets H and E of internal (hidden) and external actions, respectively; we let s, t, u, v, and their variants with indices range over S; a, b range over actions; and τ range over internal actions.

We denote the generic elements of a probabilistic automaton $\mathcal{A}$ by S, $\bar{s}$, Σ, H, E, T, and we propagate primes and indices when necessary. Thus, for example, the probabilistic automaton $\mathcal{A}'_i$ has states S'_i, start state $\bar{s}'_i$, actions Σ'_i, internal actions H'_i, external actions E'_i, and transition relation T'_i.

A transition $tr = (s, a, \mu) \in T$, also denoted by $s \xrightarrow{a} \mu$, is said to *leave* from state s, to be *labelled* by a, and to *lead* to the measure μ. We denote by $src(tr)$ the *source* state s, by $act(tr)$ the *action* a, and by $trg(tr)$ the *target* measure μ, also denoted by μ_{tr}. We also say that s enables the action a, that the action a is enabled from s, and that (s, a, μ) is enabled from s. We call a transition $s \xrightarrow{a} \mu$ *internal* or *external* whenever $a \in H$ or $a \in E$, respectively. Finally, we let $T(a) = \{\, tr \in T \mid act(tr) = a \,\}$ be the set of transitions with label a.

We say that a state s is a *deadlock* state if it enables no transitions, i.e., $\{\, tr \in T \mid src(tr) = s \,\} = \emptyset$.

Given a *PA* $\mathcal{A}$, we denote by $size(\mathcal{A}) = \max\{|S|, |T|\}$ the *size* of $\mathcal{A}$. For the purposes of this paper, we assume that $\mathcal{A}$ is finite, that is, both S and T are finite sets; moreover, we assume that each state of $\mathcal{A}$ can be reached from $\bar{s}$.

Example 1. An example of *PA* is the one shown in Figure 1: the set of states is $S = \{\bar{s}, r, y, g,$ 🟥, △, 🟢$\}$, the start state is $\bar{s}$, the set of actions Σ is the union of the set of external actions $E = \{a\}$ and of the set of internal actions $H = \{\tau\}$, and the transition relation T contains the following transitions: $\bar{s} \xrightarrow{\tau} \rho$ with $\rho = \{(r, 0.3), (y, 0.1), (g, 0.6)\}$, $r \xrightarrow{a} \delta_{\blacksquare}$, $y \xrightarrow{a} \delta_{\triangle}$, $g \xrightarrow{a} \delta_{\bullet}$, $r \xrightarrow{\tau} \delta_{\bar{s}}$, and $g \xrightarrow{\tau} \delta_{\bar{s}}$. 🟥, △, and 🟢 are deadlock states and the size of $\mathcal{E}$ is $\mathit{size}(\mathcal{E}) = 7$. ♦

3.1. Parallel Composition and Hiding

The following definition of parallel composition is an equivalent rewriting of the definition provided in [Seg06].

Definition 2. Given two *PA*s $\mathcal{A}_1$ and $\mathcal{A}_2$, we say that $\mathcal{A}_1$ and $\mathcal{A}_2$ are *compatible* if $\Sigma_1 \cap H_2 = \emptyset = H_1 \cap \Sigma_2$.

Given two compatible *PA*s $\mathcal{A}_1$ and $\mathcal{A}_2$, the *parallel composition* of $\mathcal{A}_1$ and $\mathcal{A}_2$, denoted by $\mathcal{A}_1 \parallel \mathcal{A}_2$, is the probabilistic automaton $\mathcal{A} = (S, \bar{s}, \Sigma, T)$ where

- $S = S_1 \times S_2$,
- $\bar{s} = (\bar{s}_1, \bar{s}_2)$,
- $\Sigma = E \cup H$ where $E = E_1 \cup E_2$ and $H = H_1 \cup H_2$, and
- $((s_1, s_2), a, \mu_1 \times \mu_2) \in T$ if and only if
  - whenever $a \in \Sigma_1 \cap \Sigma_2$, $(s_1, a, \mu_1) \in T_1$ and $(s_2, a, \mu_2) \in T_2$,
  - whenever $a \in \Sigma_1 \setminus \Sigma_2$, $(s_1, a, \mu_1) \in T_1$ and $\mu_2 = \delta_{s_2}$, and
  - whenever $a \in \Sigma_2 \setminus \Sigma_1$, $(s_2, a, \mu_2) \in T_2$ and $\mu_1 = \delta_{s_1}$.

For $a \in \Sigma_1 \setminus \Sigma_2$, we denote by $(s_2, \nu_a, \delta_{s_2})$ the *apparent* internal transition corresponding to not performing any transition from s_2 in the combined transition, and similarly for $a \in \Sigma_2 \setminus \Sigma_1$.

For two compatible *PA*s $\mathcal{A}_1$ and $\mathcal{A}_2$ and their parallel composition $\mathcal{A}_1 \parallel \mathcal{A}_2$, we refer to $\mathcal{A}_1$ and $\mathcal{A}_2$ as the component automata and to $\mathcal{A}_1 \parallel \mathcal{A}_2$ as the composed automaton.

Definition 3. Given a *PA* $\mathcal{A}$ and a set A of actions, the *hiding* of A in $\mathcal{A}$, denoted by $\mathrm{Hide}_A(\mathcal{A})$, is the automaton $\mathcal{A}'$ that is the same as $\mathcal{A}$ except for $E' = E \setminus A$ and $H' = H \cup A$.

Remark 1. In the above definition of parallel composition between *PA*s, we require that they are compatible, i.e., the internal actions of one automaton can not be actions of the other automaton. This requirement seems to be never fulfilled when we consider the internal action τ.

In the Process Algebra world, usually τ is the only internal action available, and it is used by every process to denote an internal transition. In the Probabilistic Automata framework, τ is used as a symbol for referring to internal actions, but usually it is not an actual action of the automaton. This means that, for two automata $\mathcal{A}_1$ and $\mathcal{A}_2$, when we write $(s_1, \tau, \mu_1) \in T_1$ and $(s_2, \tau, \mu_2) \in T_2$, we are not requiring that the label is the same for both transitions, but we are just referring to $(s_1, a_1, \mu_1) \in T_1$ and $(s_2, a_2, \mu_2) \in T_2$ for some $a_i \in H_i$, $i \in \{1, 2\}$.

The role of τ as symbol for internal actions and not as actual action becomes clear from the definition of the hiding operator: for a given set A of actions to be hidden, instead of replacing each action in A with τ as happens in process algebra world, we simply move the actions in A from E to H; the actual actions remain unchanged.

Note that it is rather easy to transform two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ that are not compatible into compatible ones, by means of the *action renaming* operator [Seg95] that allows us to rename actions under the assumption that external actions remain external and internal actions remain internal. So, we can just rename the internal actions of both $\mathcal{A}_1$ and $\mathcal{A}_2$ with fresh (internal) actions and the resulting automata are then compatible.

3.2. Weak Transitions

In the setting of labelled transition systems, weak transitions are used to abstract from internal computations [Mil89]. Intuitively, an internal weak transition is formed by an arbitrary long sequence of internal transitions, and an external weak transition is formed by an external transition preceded and followed by arbitrary long sequences of internal transitions. Note that the empty sequence is a valid arbitrary long sequence of internal transitions. To lift this idea to the setting of probabilistic automata is a little intricate owed to the fact that transitions branch into probability measures, and one thus has to work with tree-like objects instead of sequences, as detailed in the sequel.

An *execution fragment* of a *PA* $\mathcal{A}$ is a finite or infinite sequence of alternating states and actions $\alpha = s_0 a_1 s_1 a_2 s_2 \dots$ starting from a state s_0, also denoted by $\mathit{first}(\alpha)$, and, if the sequence is finite, ending with a state denoted by $\mathit{last}(\alpha)$, such that for each $i > 0$ there exists a transition $(s_{i-1}, a_i, \mu_i) \in T$ such that $\mu_i(s_i) > 0$. The *length* of α, denoted by $|\alpha|$, is the number of occurrences of actions in α. If α is infinite, then $|\alpha| = \infty$. We denote by $\mathit{state}(\alpha, i)$ the state s_i and by $\mathit{action}(\alpha, j)$ the action a_j, provided that $0 \leq i \leq |\alpha|$ and $0 < j \leq |\alpha|$. Denote by $\mathit{frags}(\mathcal{A})$ the set of execution fragments of $\mathcal{A}$ and by $\mathit{frags}^*(\mathcal{A})$ the set of finite execution fragments of $\mathcal{A}$. An execution fragment α is a *prefix* of an execution fragment α', denoted by $\alpha \leqslant \alpha'$, if the sequence α is a prefix of the sequence α'. The *trace* of α, denoted by $\mathit{trace}(\alpha)$, is the sub-sequence of external actions of α; we denote by ε the empty trace and we extend $\mathit{trace}(\,\cdot\,)$ to actions by defining $\mathit{trace}(a) = a$ if $a \in E$ and $\mathit{trace}(a) = \varepsilon$ if $a \in H$.

A *scheduler* for a *PA* $\mathcal{A}$ is a function $\sigma \colon \mathit{frags}^*(\mathcal{A}) \to \mathrm{SubDisc}(T)$ such that for each $\alpha \in \mathit{frags}^*(\mathcal{A})$, $\sigma(\alpha) \in \mathrm{SubDisc}(\{\, tr \in T \mid \mathit{src}(tr) = \mathit{last}(\alpha) \,\})$ or, equivalently, $\mathrm{Supp}(\sigma(\alpha)) \subseteq \{\, tr \in T \mid \mathit{src}(tr) = \mathit{last}(\alpha) \,\}$. Given a scheduler σ and a finite execution fragment α, the measure $\sigma(\alpha)$ describes how transitions are chosen to move on from $\mathit{last}(\alpha)$. We call a scheduler *determinate* [CS02] if, for each $\alpha, \alpha' \in \mathit{frags}^*(\mathcal{A})$ such that $\mathit{trace}(\alpha) = \mathit{trace}(\alpha')$ and $\mathit{last}(\alpha) = \mathit{last}(\alpha')$, then $\sigma(\alpha) = \sigma(\alpha')$. Essentially, a determinate scheduler bases its choice only on the current state (as happens for history-independent schedulers, also known as stationary policies in the context of Markov decision processes) and on the past external actions. In other words, a determinate scheduler acts as a history-independent scheduler between one external action and the following external action (or the choice of stopping).

A scheduler σ and a state s induce a probability measure $\mu_{\sigma,s}$ over execution fragments as follows. The basic measurable events are the cones of finite execution fragments, where the cone of α, denoted by C_α, is the set $C_\alpha = \{\, \alpha' \in \mathit{frags}(\mathcal{A}) \mid \alpha \leqslant \alpha' \,\}$. The probability $\mu_{\sigma,s}$ of a cone C_α is defined recursively as follows:

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

Standard measure theoretical arguments ensure that $\mu_{\sigma,s}$ extends uniquely to the σ-field generated by cones. We call the resulting measure $\mu_{\sigma,s}$ a *probabilistic execution fragment* of $\mathcal{A}$ and we say that it is generated by σ from s. Given a finite execution fragment α, we define $\mu_{\sigma,s}(\alpha)$ as $\mu_{\sigma,s}(\alpha) = \mu_{\sigma,s}(C_\alpha) \cdot \sigma(\alpha)(\bot)$, where $\sigma(\alpha)(\bot)$ is the probability of choosing no transitions after α has occurred.

Definition 4. Given a *PA* $\mathcal{A}$, we say that there is a *weak combined transition* from $s \in S$ to $\mu \in \mathrm{Disc}(S)$ labelled by $a \in \Sigma$, denoted by $s \overset{a}{\Longrightarrow}_{\mathrm{c}} \mu$, if there exists a scheduler σ such that the following holds for the induced probabilistic execution fragment $\mu_{\sigma,s}$:

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

In this case, we say that the weak combined transition $s \overset{a}{\Longrightarrow}_{\mathrm{c}} \mu$ is induced by σ, that $s \overset{a}{\Longrightarrow}_{\mathrm{c}} \mu$ exists in $\mathcal{A}$, and that $\mathcal{A}$ enables $s \overset{a}{\Longrightarrow}_{\mathrm{c}} \mu$.

Albeit the definition of weak combined transitions is admittedly intricate, it is just the obvious extension of weak transitions on labelled transition systems to the setting with probabilities. We refer to Segala [Seg06] for more details on weak combined transitions.

Example 2. Consider the automaton $\mathcal{E}$ depicted in Figure 1 and let ρ be $\rho = \{(r, 0.3), (y, 0.1), (g, 0.6)\}$; $\mathcal{E}$ enables the weak combined transition $\bar{s} \overset{a}{\Longrightarrow}_{\mathrm{c}} \mu$ where $\mu = \{(\blacksquare, \frac{9}{50}), (\triangle, \frac{8}{50}), (\bullet, \frac{33}{50})\}$ via the scheduler σ

defined as follows:

$$\sigma(\alpha) = \begin{cases} \delta_{\bar{s} \xrightarrow{\tau} \rho} & \text{if } \mathit{last}(\alpha) = \bar{s}, \\ \delta_{r \xrightarrow{\tau} \delta_{\bar{s}}} & \text{if } \alpha = \bar{s}\tau r, \\ \delta_{r \xrightarrow{a} \delta_{\blacksquare}} & \text{if } \alpha \neq \bar{s}\tau r \text{ and } \mathit{last}(\alpha) = r, \\ \delta_{y \xrightarrow{a} \delta_{\blacktriangle}} & \text{if } \mathit{last}(\alpha) = y, \\ \{(g \xrightarrow{\tau} \delta_{\bar{s}}, 0.5), (g \xrightarrow{a} \delta_{\bullet}, 0.5)\} & \text{if } \alpha = \bar{s}\tau g, \\ \delta_{g \xrightarrow{a} \delta_{\bullet}} & \text{if } \alpha \neq \bar{s}\tau g \text{ and } \mathit{last}(\alpha) = g, \\ \delta_{\perp} & \text{otherwise.} \end{cases}$$

We now verify the three properties that $\mu_{\sigma,\bar{s}}$ has to satisfy in order to justify $\bar{s} \overset{a}{\Longrightarrow}_{\mathrm{c}} \mu$: we start from the third property, since the first two can be derived from it. Consider the state $\blacksquare$: it is reached with probability

$$\begin{aligned} &\mu_{\sigma,\bar{s}}(\{\, \alpha \in \mathit{frags}^*(\mathcal{E}) \mid \mathit{last}(\alpha) = \blacksquare \,\}) \\ &= \mu_{\sigma,\bar{s}}(\{\bar{s}\tau r\tau\bar{s}\tau r a\blacksquare\}) + \mu_{\sigma,\bar{s}}(\{\bar{s}\tau g\tau\bar{s}\tau r a\blacksquare\}) \\ &\quad + \mu_{\sigma,\bar{s}}(\{\, \alpha \in \mathit{frags}^*(\mathcal{E}) \mid \mathit{last}(\alpha) = \blacksquare \,\} \setminus \{\bar{s}\tau r\tau\bar{s}\tau r a\blacksquare, \bar{s}\tau g\tau\bar{s}\tau r a\blacksquare\}) \\ &= \mu_{\sigma,\bar{s}}(\{\bar{s}\tau r\tau\bar{s}\tau r a\blacksquare\}) + \mu_{\sigma,\bar{s}}(\{\bar{s}\tau g\tau\bar{s}\tau r a\blacksquare\}) + 0 \\ &= \left(\left(\left(((1) \cdot 1 \cdot \frac{3}{10}) \cdot 1 \cdot 1\right) \cdot 1 \cdot \frac{3}{10}\right) \cdot 1 \cdot 1\right) \cdot 1 + \left(\left(\left(((1) \cdot 1 \cdot \frac{6}{10}) \cdot \frac{1}{2} \cdot 1\right) \cdot 1 \cdot \frac{3}{10}\right) \cdot 1 \cdot 1\right) \cdot 1 \\ &= \frac{9}{100} + \frac{9}{100} = \frac{9}{50} = \mu(\blacksquare), \end{aligned}$$

as required. The fact that

$$\mu_{\sigma,\bar{s}}(\{\, \alpha \in \mathit{frags}^*(\mathcal{E}) \mid \mathit{last}(\alpha) = \blacksquare \,\} \setminus \{\bar{s}\tau r\tau\bar{s}\tau r a\blacksquare, \bar{s}\tau g\tau\bar{s}\tau r a\blacksquare\}) = 0$$

is justified as follows: let $\alpha \in \{\, \beta \in \mathit{frags}^*(\mathcal{E}) \mid \mathit{last}(\beta) = \blacksquare \,\}$ such that $\alpha \notin \{\bar{s}\tau r\tau\bar{s}\tau r a\blacksquare, \bar{s}\tau g\tau\bar{s}\tau r a\blacksquare\}$; if $\mathit{first}(\alpha) = t \neq \bar{s}$, then by the recursive definition of $\mu_{\sigma,\bar{s}}(C_\alpha)$ we have that the base case is $\mu_{\sigma,\bar{s}}(C_t) = 0$, hence $\mu_{\sigma,\bar{s}}(C_\alpha) = 0$ as well. Suppose that $\mathit{first}(\alpha) = \bar{s}$ and consider the case $\alpha = \bar{s}\tau r a\blacksquare$:

$$\begin{aligned} \mu_{\sigma,\bar{s}}(C_\alpha) &= \mu_{\sigma,\bar{s}}(C_{\bar{s}\tau r a\blacksquare}) \\ &= \mu_{\sigma,\bar{s}}(C_{\bar{s}\tau r}) \cdot \sum_{tr \in T(a)} \sigma(\bar{s}\tau r)(tr) \cdot \mu_{tr}(\blacksquare) \\ &= \mu_{\sigma,\bar{s}}(C_{\bar{s}\tau r}) \cdot (\sigma(\bar{s}\tau r)(r \xrightarrow{a} \delta_{\blacksquare}) \cdot \delta_{\blacksquare}(\blacksquare) \\ &\qquad + \sigma(\bar{s}\tau r)(y \xrightarrow{a} \delta_{\blacktriangle}) \cdot \delta_{\blacktriangle}(\blacksquare) \\ &\qquad + \sigma(\bar{s}\tau r)(g \xrightarrow{a} \delta_{\bullet}) \cdot \delta_{\bullet}(\blacksquare)) \\ &= \mu_{\sigma,\bar{s}}(C_{\bar{s}\tau r}) \cdot (0 \cdot 1 + 0 \cdot 0 + 0 \cdot 0) = 0. \end{aligned}$$

Finally, the remaining finite execution fragments are such that $\alpha \in C_{\bar{s}\tau r\tau\bar{s}\tau r\tau\bar{s}} \cup C_{\bar{s}\tau r\tau\bar{s}\tau g\tau\bar{s}} \cup C_{\bar{s}\tau g\tau\bar{s}\tau r\tau\bar{s}} \cup C_{\bar{s}\tau g\tau\bar{s}\tau g\tau\bar{s}}$. Consider the case $\alpha \in C_{\bar{s}\tau r\tau\bar{s}\tau r\tau\bar{s}}$: by the recursive definition of $\mu_{\sigma,\bar{s}}(C_\alpha)$ we have that $\mu_{\sigma,\bar{s}}(C_\alpha) = \mu_{\sigma,\bar{s}}(C_{\bar{s}\tau r\tau\bar{s}\tau r\tau\bar{s}}) \cdot p$ for some value $p \in \mathbb{R}^{\geq 0}$; now, consider $\mu_{\sigma,\bar{s}}(C_{\bar{s}\tau r\tau\bar{s}\tau r\tau\bar{s}})$:

$$\begin{aligned} \mu_{\sigma,\bar{s}}(C_{\bar{s}\tau r\tau\bar{s}\tau r\tau\bar{s}}) &= \mu_{\sigma,\bar{s}}(C_{\bar{s}\tau r\tau\bar{s}\tau r}) \cdot \sum_{tr \in T(\tau)} \sigma(\bar{s}\tau r\tau\bar{s}\tau r)(tr) \cdot \mu_{tr}(\bar{s}) \\ &= \mu_{\sigma,\bar{s}}(C_{\bar{s}\tau r\tau\bar{s}\tau r}) \cdot (\sigma(\bar{s}\tau r\tau\bar{s}\tau r)(r \xrightarrow{\tau} \delta_{\bar{s}}) \cdot \delta_{\bar{s}}(\bar{s}) \\ &\qquad + \sigma(\bar{s}\tau r\tau\bar{s}\tau r)(\bar{s} \xrightarrow{\tau} \rho) \cdot \rho(\bar{s}) \\ &\qquad + \sigma(\bar{s}\tau r\tau\bar{s}\tau r)(g \xrightarrow{\tau} \delta_{\bar{s}}) \cdot \delta_{\bar{s}}(\bar{s})) \\ &= \mu_{\sigma,\bar{s}}(C_{\bar{s}\tau r\tau\bar{s}\tau r}) \cdot (0 \cdot 1 + 0 \cdot 0 + 0 \cdot 1) = 0, \end{aligned}$$

and similarly for the remaining cases $\alpha \in C_{\bar{s}\tau r\tau\bar{s}\tau g\tau\bar{s}}$, $\alpha \in C_{\bar{s}\tau g\tau\bar{s}\tau r\tau\bar{s}}$, and $\alpha \in C_{\bar{s}\tau g\tau\bar{s}\tau g\tau\bar{s}}$. This completes the justification of

$$\mu_{\sigma,\bar{s}}(\{\, \alpha \in \mathit{frags}^*(\mathcal{E}) \mid \mathit{last}(\alpha) = \blacksquare \,\} \setminus \{\bar{s}\tau r\tau\bar{s}\tau r a\blacksquare, \bar{s}\tau g\tau\bar{s}\tau r a\blacksquare\}) = 0.$$

A similar analysis shows that $\mu_{\sigma,\bar{s}}(\{\, \alpha \in \mathit{frags}^*(\mathcal{E}) \mid \mathit{last}(\alpha) = $ △ $\}) = \mu($△$)$ and $\mu_{\sigma,\bar{s}}(\{\, \alpha \in \mathit{frags}^*(\mathcal{E}) \mid \mathit{last}(\alpha) = $ 🟢 $\}) = \mu($🟢$)$; for each remaining state $s \in \{\bar{s}, r, y, g\}$, it is easy to verify that $\mu_{\sigma,\bar{s}}(\{\, \alpha \in \mathit{frags}^*(\mathcal{E}) \mid \mathit{last}(\alpha) = s\,\}) = 0$. Regarding the first two properties of the definition of weak combined transition, we have that $\mu_{\sigma,\bar{s}}(\mathit{frags}^*(\mathcal{E})) = 1$ follows directly from the third condition, as well as the second property by considering the trace of the finite execution fragments occurring with non-zero probability. ♦

3.3. Weak Probabilistic Bisimulation

As said in the introduction, bisimulation relations constitute a powerful tool that allows us to verify whether two models describe essentially the same real system. Moreover, they allow us to compute the minimal automaton that is bisimilar to the given one [EHS+13]. We now recall the definition of weak probabilistic bisimulation [Seg95, Seg06], that is the relation that allows us to abstract away from internal computations while solving nondeterministic choices via convex combinations of the available transitions.

Definition 5. Given a *PA* $\mathcal{A}$, an equivalence relation $\mathcal{R}$ on S is a *weak probabilistic bisimulation* if, for each pair of states $s, t \in S$ such that $s \mathrel{\mathcal{R}} t$, if $s \xrightarrow{a} \mu_s$ for some probability measure μ_s, then there exists a probability measure μ_t such that $t \overset{a}{\Longrightarrow}_{\mathrm{c}} \mu_t$ and $\mu_s \mathrel{\mathcal{L}(\mathcal{R})} \mu_t$.

In the following, we may refer to the condition "there exists μ_t such that $t \overset{a}{\Longrightarrow}_{\mathrm{c}} \mu_t$ and $\mu_s \mathrel{\mathcal{L}(\mathcal{R})} \mu_t$" as the *step condition* of the bisimulation. Specially, when the bisimulation is seen as a two-player game between the two automata, the *step condition* is the condition on the weak transition (or weak step) performed by the *defender* state t while matching the transition (or step) performed by the *challenger* state s.

To check whether two *PA*s $\mathcal{A}_1$ and $\mathcal{A}_2$ are weak probabilistic bisimilar, we can either adapt the above definition to work with pairs of automata, or we can just consider the *PA* $\mathcal{A} = \mathcal{A}_1 \uplus \mathcal{A}_2$ such that $S = S_1 \uplus S_2$, $\bar{s} = \bar{s}_1$, $H = H_1 \cup H_2$, $E = E_1 \cup E_2$, $T = T_1 \uplus T_2$. Note that the choice $\bar{s} = \bar{s}_1$ is arbitrary, since it does not affect the weak probabilistic bisimulation; similarly, we can ignore the requirement $E \cap H = \emptyset$ since actions are taken into account by the step condition: if the same action is external for $\mathcal{A}_1$ and internal for $\mathcal{A}_2$, then $\mathcal{A}_1$ and $\mathcal{A}_2$ are not bisimilar since the external transition proposed by $\mathcal{A}_1$ can not be matched by $\mathcal{A}_2$. Deciding whether two automata are bisimilar then reduces to compute the bisimulation $\mathcal{R}$ on $\mathcal{A}$ and to check whether their start states are related by $\mathcal{R}$, i.e., whether $\bar{s}_1 \mathrel{\mathcal{R}} \bar{s}_2$.

Definition 6. Given two *PA*s $\mathcal{A}_1$ and $\mathcal{A}_2$, we say that $\mathcal{A}_1$ and $\mathcal{A}_2$ are *weakly probabilistic bisimilar* if there exists a weak probabilistic bisimulation $\mathcal{R}$ on $S_1 \uplus S_2$ such that $\bar{s}_1 \mathrel{\mathcal{R}} \bar{s}_2$. We denote the coarsest weak probabilistic bisimulation by $\approx$, and call it weak probabilistic bisimilarity.

Weak probabilistic bisimilarity is an equivalence relation preserved by standard process algebraic composition operators on *PA* [PS04], such as parallel composition, action hiding, action renaming, and action prefixing. As we will see in the next section, the complexity of deciding $\mathcal{A}_1 \approx \mathcal{A}_2$ strictly depends on finding the matching weak combined transition $t \overset{a}{\Longrightarrow}_{\mathrm{c}} \mu_t$ for which determinate schedulers suffice (cf. [CS02, Proposition 3]): in Section 5 we will show how to find them in polynomial time.

Remark 2. In this work we do not consider the weak bisimulation relation obtained by restricting to weak transitions $t \overset{a}{\Longrightarrow} \mu_t$ induced by a deterministic (or Dirac) scheduler, i.e., by a scheduler σ such that for each finite execution fragment α, either $\sigma(\alpha) = \delta_{tr}$ for some $tr \in T$, or $\sigma(\alpha) = \delta_{\perp}$. In fact, as shown in [Den05], the resulting bisimulation is not transitive and this makes the usual compositional minimization approach much more difficult to use. In such an approach a given automaton $\mathcal{A}_0$ is decomposed into multiple sub-automata running in parallel, i.e., $\mathcal{A}_0 = \mathcal{B}_1 \parallel \mathcal{B}_2 \parallel \ldots \parallel \mathcal{B}_n$; then one component $\mathcal{B}_i$ at a time is replaced by another component $\mathcal{B}'_i$ that is bisimilar to but smaller than $\mathcal{B}_i$. This gives rise to a sequence of automata $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_n$ such that for each $0 \leq i < n$, $\mathcal{A}_i$ and $\mathcal{A}_{i+1}$ are bisimilar. If the bisimulation relation is not transitive, then we can not derive that $\mathcal{A}_0$ and $\mathcal{A}_n$ are bisimilar. Instead, we have to provide a relation witnessing the bisimilarity of $\mathcal{A}_0$ and $\mathcal{A}_n$. Moreover, the construction we present in Section 5 to efficiently find a weak combined transition is not easily extendable to weak (non-combined) transitions; see Remark 4 for a more detailed explanation.

Since in this paper we consider only weak combined transitions and weak probabilistic bisimulation and bisimilarity, from now on we omit the adjectives "combined" and "probabilistic", respectively.

```
QUOTIENT($\mathcal{A}$)
1: $\mathcal{R} = \{S\}$;
2: $(s, a, \mu_s) = $ FINDSPLIT($\mathcal{R}$);
3: while $s \neq \perp$ do
4:     $\mathcal{R} = $ REFINE($\mathcal{R}, (s, a, \mu_s)$);
5:     $(s, a, \mu_s) = $ FINDSPLIT($\mathcal{R}$);
6: return $\mathcal{R}$
```

```
FINDSPLIT($\mathcal{R}$)
1: for all $s \in S$ do
2:     for all $(s, a, \mu_s) \in T$ do
3:         for all $t \in [s]_{\mathcal{R}}$ do
4:             if there does not exist $t \xrightarrow{a}_{\mathrm{c}} \mu_t$
               such that $\mu_s \; \mathcal{L}(\mathcal{R}) \; \mu_t$ then
5:                 return $(s, a, \mu_s)$
6: return $(\perp, \tau, \delta_{\bar{s}})$
```

```
REFINE($\mathcal{R}, (s, a, \mu_s)$)
1: $\mathcal{C}_s = \mathcal{C}_{\neg s} = \emptyset$
2: for all $t \in [s]_{\mathcal{R}}$ do
3:     if there exists $t \xrightarrow{a}_{\mathrm{c}} \mu_t$ such that $\mu_s \; \mathcal{L}(\mathcal{R}) \; \mu_t$ then
4:         $\mathcal{C}_s = \mathcal{C}_s \cup \{t\}$
5:     else
6:         $\mathcal{C}_{\neg s} = \mathcal{C}_{\neg s} \cup \{t\}$
7: return $\mathcal{R} \setminus \{[s]_{\mathcal{R}}\} \cup \{\mathcal{C}_s, \mathcal{C}_{\neg s}\}$
```

Figure 2. The decision procedure for the weak bisimilarity

4. Computing the Weak Bisimilarity for Minimizing Automata

In this section, we recast the decision procedure of [CS02] that decides whether two probabilistic automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are weak bisimilar by following the standard partition refinement approach [KS90, PT87, PLS00].

4.1. Deciding Weak Bisimilarity

We now study in detail the decision procedure for the weak bisimulation and then we analyze the complexity of the algorithm.

4.1.1. Weak Bisimilarity Decision Algorithm

The decision algorithm for the weak bisimulation is sketched in Figure 2; the procedure QUOTIENT iteratively constructs the set $S/{\approx}$, the set of equivalence classes of states S under $\approx$, starting with the partitioning $\mathcal{R} = \{S\}$ and refining it until $\mathcal{R}$ satisfies the definition of weak bisimulation and thus the resulting partitioning is the coarsest one, i.e., we compute the weak bisimilarity. In the following, we treat $\mathcal{R}$ both as a set of partitions and as an equivalence relation without further mentioning.

The partitioning is refined by procedure REFINE into a finer partitioning as long as there is a partition containing two states that violate the bisimulation condition, which is checked for in procedure FINDSPLIT. Procedure REFINE splits the partition $[s]_{\mathcal{R}}$ into two new partitions $\mathcal{C}_s$ and $\mathcal{C}_{\neg s}$ according to the discriminating information (s, a, μ_s) identified by FINDSPLIT before. More precisely, $\mathcal{C}_s$ contains all states belonging to $[s]_{\mathcal{R}}$ that are able to match (s, a, μ_s), while $\mathcal{C}_{\neg s}$ contains the remaining states in $[s]_{\mathcal{R}}$ that fail to match (s, a, μ_s). It is clear that at the termination of the **for** loop at line 2 of REFINE, both $\mathcal{C}_s$ and $\mathcal{C}_{\neg s}$ are not empty: $\mathcal{C}_s$ obviously contains the state s while $\mathcal{C}_{\neg s}$ contains for sure the state t that caused FINDSPLIT to return (s, a, μ_s) at line 4. So far, the procedure essentially agrees with the *DecideBisim*$(\mathcal{A}_1, \mathcal{A}_2)$ procedure of [CS02].

The real difference between the decision procedure we provide here and the one presented in [CS02] however appears inside the procedure FINDSPLIT, where we check directly the step condition by looking for a weak transition $t \xrightarrow{a}_{\mathrm{c}} \mu_t$ such that $\mu_s \; \mathcal{L}(\mathcal{R}) \; \mu_t$, instead of computing the information associated by a to s and t, i.e., the set with respect to $\mathcal{R}$ of the probability measures reached from s (and t) via a weak transition labelled by a.

Remark 3. In the context of model checking, the definition of bisimulation usually requires that two related states are labelled with identical sets of atomic propositions. The decision procedure presented in Figure 2 can be easily adapted to such a definition by modifying line 1 of QUOTIENT as follows: the initial partitioning $\mathcal{R}$ is such that for each class $\mathcal{C}$ of $\mathcal{R}$, $s, s' \in \mathcal{C}$ if and only if s and s' are labelled with identical sets of atomic propositions.

4.1.2. Complexity of the Decision Algorithm

Assume we are given the *PA* $\mathcal{A}$; let $N = \mathit{size}(\mathcal{A})$. The **for** loop at line 1 of the procedure FINDSPLIT cycles at most N times. Now, consider the **for** loop at line 2: since $T = \bigcup_{s \in S} \{\, tr \in T \mid \mathit{src}(tr) = s \,\}$ and $\{\, tr \in T \mid \mathit{src}(tr) = s \,\} \cap \{\, tr \in T \mid \mathit{src}(tr) = t \,\} = \emptyset$ for each $s, t \in S$ with $s \neq t$, it follows that the two **for** loops together cycle at most N times. In the worst case (that occurs when $[s]_{\mathcal{R}} = S$ and each state t satisfies the step condition), the **for** loop at line 3 cycles at most N times as well. This means that the existential check of $t \overset{a}{\Longrightarrow}_{\mathrm{c}} \mu_t$ such that $\mu_s \; \mathcal{L}(\mathcal{R}) \; \mu_t$ at line 4 is performed at most N^2 times. Let $W(N)$ be the complexity of such check; it is immediate to see that FINDSPLIT $\in \mathcal{O}(N^2 \cdot W(N))$.

The **for** loop in procedure REFINE can be performed at most N times; this happens when $[s]_{\mathcal{R}} = S$. In each loop, an instance of the existential check of $t \overset{a}{\Longrightarrow}_{\mathrm{c}} \mu_t$ such that $\mu_s \; \mathcal{L}(\mathcal{R}) \; \mu_t$ has to be computed, with complexity $W(N)$; the resulting complexity of REFINE is therefore $\mathcal{O}(N \cdot W(N))$.

The **while** loop in the procedure QUOTIENT can be performed at most N times; this happens when in each loop the procedure FINDSPLIT returns (s, a, μ_s) where $s \neq \bot$, that is, not every pair of states in $[s]_{\mathcal{R}}$ satisfies the step condition. Since in each loop the procedure REFINE replaces such class $[s]_{\mathcal{R}}$ with two non-empty classes $\mathcal{C}_s$ and $\mathcal{C}_{\neg s}$, after at most N loops every class contains a single state and the procedure FINDSPLIT returns $(\bot, \tau, \delta_{\bar{s}})$ since each transition $s \xrightarrow{a} \mu_s$ is obviously matched by s itself. Since REFINE has complexity $\mathcal{O}(N \cdot W(N))$ and FINDSPLIT $\mathcal{O}(N^2 \cdot W(N))$, it follows that the overall complexity of QUOTIENT is $\mathcal{O}(N \cdot (N^2 \cdot W(N) + N \cdot W(N))) = \mathcal{O}(N^3 \cdot W(N))$.

Proposition 1. Given two *PA*s $\mathcal{A}_1$ and $\mathcal{A}_2$, let $S = S_1 \uplus S_2$ and $N = \mathit{size}(\mathcal{A}_1) + \mathit{size}(\mathcal{A}_2)$; given a state $t \in S$, an action $a \in \Sigma$, the probability measures $\mu_s, \mu_t \in \mathrm{Disc}(S)$, and an equivalence relation $\mathcal{R}$ on S, let $W(N)$ be the complexity of checking the existence of $t \overset{a}{\Longrightarrow}_{\mathrm{c}} \mu_t$ such that $\mu_s \; \mathcal{L}(\mathcal{R}) \; \mu_t$. Checking $\mathcal{A}_1 \approx \mathcal{A}_2$ has complexity $\mathcal{O}(N^3 \cdot W(N))$.

Proof. Immediate by the previous analysis. □

4.2. Minimization and Parallel Composition

In this section, we explain in detail the practical steps that lead from a *PA* $\mathcal{A}$ to the minimal automaton $\mathcal{M}$ that is weak bisimilar to $\mathcal{A}$, as formalized in [EHS+13, HK00, CGM+96]: the first step extracts the reachable fragment $\mathcal{A}_{\circlearrowright}$ of $\mathcal{A}$, i.e., the states and the corresponding transitions that can be reached with non-zero probability from the start state. (Note that by our assumptions on the automata we do not need this initial step.) The second step generates the quotient automaton by computing the weak bisimilarity $\approx$. Once $\approx$ is at hand, the quotient automaton $[\mathcal{A}_{\circlearrowright}]_{\approx}$ is extracted in a third step: it has as set of states the set of the equivalences classes of $\approx$ and as the start state the class of $\bar{s}$; the sets of internal and external actions are the same as in $\mathcal{A}_{\circlearrowright}$ while the transition relation contains only the transitions $[s]_{\approx} \xrightarrow{a} \rho$ such that there exists $s \xrightarrow{a} \mu \in T_{\circlearrowright}$ where $\rho(\mathcal{C}) = \sum_{t \in \mathcal{C}} \mu(t)$ for each $\mathcal{C} \in [S_{\circlearrowright}]_{\approx}$. The fourth step of the minimization procedure removes from $[\mathcal{A}_{\circlearrowright}]_{\approx}$ the transitions that are redundant, i.e., the transitions that can be removed from the automaton since they can be weakly matched by the remaining transitions; the fifth and final step normalizes the internal transitions, i.e., each transition $s \xrightarrow{\tau} \mu$ is replaced by $s \xrightarrow{\tau} \mu \backslash s$. Note that the fourth step ensures that there are no transitions $s \xrightarrow{\tau} \delta_s$ since they are trivially redundant.

The correctness of the above construction is justified by the following properties of weak probabilistic bimulation: let A be a set of actions and $\mathcal{A}$, $\mathcal{A}'$, $\mathcal{A}''$, and $\mathcal{A}_e$ be four *PA*s such that $\mathcal{A}_e$ is compatible with both $\mathcal{A}$ and $\mathcal{A}$. Then the following holds:

- $\approx$ is transitive [Seg95]: if $\mathcal{A} \approx \mathcal{A}'$ and $\mathcal{A}' \approx \mathcal{A}''$, then $\mathcal{A} \approx \mathcal{A}''$;
- $\approx$ is preserved by parallel composition [Seg95]: if $\mathcal{A} \approx \mathcal{A}'$, then $\mathcal{A} \parallel \mathcal{A}_e \approx \mathcal{A}' \parallel \mathcal{A}_e$;
- $\approx$ is preserved by the hiding operator: if $\mathcal{A} \approx \mathcal{A}'$, then $\mathrm{Hide}_A(\mathcal{A}) \approx \mathrm{Hide}_A(\mathcal{A}')$;
- $\mathcal{A} \approx \mathcal{A}_{\circlearrowright}$ [EHS+13];

- $\mathcal{A} \approx [\mathcal{A}]_{\approx}$ [EHS+13];
- removing redundant transitions preserves weak bisimilarity [EHS+13]; and
- normalizing internal transitions preserves weak bisimilarity [EHS+13].

The main computational bottleneck of this overall minimization procedure applied to an automaton $\mathcal{A}$ is the second step, the weak bisimulation computation, that we have already seen by Proposition 1 to be $\mathcal{O}(N^3 \cdot W(N))$.

Therefore, this bottleneck has to be carefully considered, with respect to the size of the models to be processed by it: when we want to minimize a large automaton that is the result of the parallel composition of several smaller automata, according to the definition of parallel composition, the resulting state space is the Cartesian product of the single state spaces. This means that the state space of the composed automaton grows exponentially in the number of components, in particular when they are different instances of the same system, leading quickly to prohibitively large automata. However, it is quite common to in this way generate states and transitions that are actually useless since they are not reachable from the start state of the composed automaton, in particular when the resulting transition has as label an internal action. For instance, suppose that we have a transition $s_1 \xrightarrow{\tau} \mu_1 \in T_1$. According to the definition of parallel composition, for each $s_2 \in S_2$ we have to generate the transition $(s_1, s_2) \xrightarrow{\tau} \mu_1 \times \delta_{s_2}$, even when (s_1, s_2) can not be reached from $(\bar{s}_1, \bar{s}_2)$. To alleviate the fast growth of the parallel composition it is advisable to generate only the reachable fragment or adopt more advanced techniques [GSL96, KM00].

Furthermore, consider the two *PA*s $\mathcal{A}_1$ and $\mathcal{A}_2$ such that their only transitions are $\{s \xrightarrow{\tau} \delta_t, t \xrightarrow{a} \delta_t\}$ and $\{x \xrightarrow{a} \delta_y, y \xrightarrow{a} \delta_y\}$, respectively: it is immediate to see that both automata are weak bisimilar to $\mathcal{A}_3$ whose only transition is $v \xrightarrow{a} \delta_v$ and that $\mathcal{A}_1 \parallel \mathcal{A}_2$ is weak bisimilar to $\mathcal{A}_3 \parallel \mathcal{A}_3$ whose only transition is $(v, v) \xrightarrow{a} \delta_{(v,v)}$. Such weak bisimilarity between $\mathcal{A}_1 \parallel \mathcal{A}_2$ and $\mathcal{A}_3 \parallel \mathcal{A}_3$ is not fortuitous but derives from the fact that the weak bisimulation is preserved by the parallel composition. In fact, for any pair of compatible *PA*s $\mathcal{A}_1$ and $\mathcal{A}_2$, we have that $\mathcal{A}_1 \parallel \mathcal{A}_2 \approx [\mathcal{A}_1]_{\approx} \parallel \mathcal{A}_2 \approx [\mathcal{A}_1]_{\approx} \parallel [\mathcal{A}_2]_{\approx}$. The first bisimulation is justified by taking $\mathcal{A}_2$ as context and the fact that $\mathcal{A}_1 \approx [\mathcal{A}_1]_{\approx}$, and similarly for the second bisimulation. The compatibility of the pair of automata we compose is ensured by the fact that an automaton and its quotient have the same sets of actions. In general $[\mathcal{A}_1]_{\approx} \parallel [\mathcal{A}_2]_{\approx}$ is not the minimal automaton that is weak bisimilar to $\mathcal{A}_1 \parallel \mathcal{A}_2$: in fact, the presence of internal transitions may lead to symmetric constructions that are identified and collapsed by computing the weak bisimulation. For instance, suppose that we have the states s_1 and s_2 enabling the transitions $s_1 \xrightarrow{\tau} \delta_{s'_1}$, $s_1 \xrightarrow{a} \mu_1$, and $s'_1 \xrightarrow{b} \mu'_1$ and $s_2 \xrightarrow{\tau} \delta_{s'_2}$, $s_2 \xrightarrow{a} \mu_2$, and $s'_2 \xrightarrow{b} \mu'_2$, respectively. In the parallel composition we obtain the four internal transitions $(s_1, s_2) \xrightarrow{\tau} \delta_{(s'_1,s_2)}$, $(s_1, s_2) \xrightarrow{\tau} \delta_{(s_1,s'_2)}$, $(s'_1, s_2) \xrightarrow{\tau} \delta_{(s'_1,s'_2)}$, and $(s_1, s'_2) \xrightarrow{\tau} \delta_{(s'_1,s'_2)}$ and the two external transitions $(s_1, s_2) \xrightarrow{a} \mu_1 \times \mu_2$ and $(s'_1, s'_2) \xrightarrow{b} \mu'_1 \times \mu'_2$. It is clear that the states (s'_1, s_2), (s_1, s'_2), and (s'_1, s'_2) are weak bisimilar, so they can be collapsed. Applying the hiding operator after a parallel composition increases this effect considerably.

5. Weak Transition Construction as a Linear Programming Problem

As discussed in the previous section, the main source of the worst case behaviour of the decision algorithms [TH15, CS02] for *PA* weak probabilistic bisimulation is the recurring need to check for the existence of the weak transition. This is solved with an exponential algorithm in [CS02] and a polynomial algorithm in [TH15]. The latter approach takes inspiration from network flow problems: a weak transition $t \overset{a}{\Longrightarrow}_{\mathrm{c}} \mu_t$ of a *PA* $\mathcal{A}$ is described as an enriched flow problem in which the initial probability mass δ_t splits along internal transitions, and precisely one external transition with label $a \neq \tau$ for every stream, in order to reach μ_t. The enriched flow problem is then translated into a Linear Programming (LP) problem extended with *balancing constraints* that encode the need to respect transition probability measure.

5.1. Network Construction

To describe the structure of the enriched LP problem, we first recall the definition of the network graph corresponding to a weak transition.

Definition 7 (cf. [TH15, Sect. 5.2]). Given a *PA* $\mathcal{A}$, a state t, an action a, a probability measure μ, and an equivalence relation $\mathcal{R}$ on S, the network graph $G(t, a, \mu, \mathcal{R}) = (V, E)$ relative to the weak transition $t \xRightarrow{a}_{\mathrm{c}} \mu_t$ is defined as follows. Given $v \in S$, $a \in E$, and $tr \in T$, let v_a, v^{tr}, and v_a^{tr} be three copies of v. For $a \in E$, the set V of vertices is

$$V = \{\vartriangle, \blacktriangledown\} \cup S \cup S^{tr} \cup S_a \cup S_a^{tr} \cup S/\mathcal{R}$$

where

$$\begin{aligned} S^{tr} &= \{\, v^{tr} \mid tr = v \xrightarrow{b} \rho \in T, b \in \{a\} \cup H \,\}, \\ S_a &= \{\, v_a \mid v \in S \,\}, \text{ and} \\ S_a^{tr} &= \{\, v_a^{tr} \mid v^{tr} \in S^{tr} \,\} \end{aligned}$$

and the set E of arcs is

$$E = \{(\vartriangle, t)\} \cup L_1 \cup L_a \cup L_2 \cup L_{\mathcal{R}}^a$$

where

$$\begin{aligned} L_1 &= \{\, (v, v^{tr}), (v^{tr}, v') \mid tr = v \xrightarrow{\tau} \rho \in T, v' \in \mathrm{Supp}(\rho) \,\}, \\ L_a &= \{\, (v, v_a^{tr}), (v_a^{tr}, v'_a) \mid tr = v \xrightarrow{a} \rho \in T, v' \in \mathrm{Supp}(\rho) \,\}, \\ L_2 &= \{\, (v_a, v_a^{tr}), (v_a^{tr}, v'_a) \mid tr = v \xrightarrow{\tau} \rho \in T, v' \in \mathrm{Supp}(\rho) \,\}, \text{ and} \\ L_{\mathcal{R}}^a &= \{\, (v_a, \mathcal{C}), (\mathcal{C}, \blacktriangledown) \mid \mathcal{C} \in S/\mathcal{R}, v \in \mathcal{C} \,\}. \end{aligned}$$

For $a \in H$ the definition is similar:

$$V = \{\vartriangle, \blacktriangledown\} \cup S \cup S^{tr} \cup S_{\mathcal{R}} \cup S/\mathcal{R}$$

and

$$E = \{(\vartriangle, t)\} \cup L_1 \cup L_\perp \cup L_{\mathcal{R}},$$

where $L_{\mathcal{R}} = \{\, (v, \mathcal{C}), (\mathcal{C}, \blacktriangledown) \mid \mathcal{C} \in S/\mathcal{R}, v \in \mathcal{C} \,\}$.

We refer to the elements of $S \cup S_a$ as state nodes, of $\mathcal{T} = S^{tr} \cup S_a^{tr}$ as transition nodes, and of $S/\mathcal{R}$ as class nodes.

Example 3. Consider again the *PA* $\mathcal{E}$ in Figure 1 and suppose that we want to check whether there exists a weak transition $\bar{s} \xRightarrow{a}_{\mathrm{c}} \rho$ such that $\rho \, \mathcal{L}(\mathcal{R}) \, \mu$ where $\mu = \{(🟥, \frac{9}{50}), (🔺, \frac{8}{50}), (🟢, \frac{33}{50})\}$ and $\mathcal{R} = \mathcal{I}$. Note that this implies that $\rho = \mu$. Denote as usual the transitions of $\mathcal{E}$ as follows: $tr_0 = \bar{s} \xrightarrow{\tau} \{(r, 0.3), (y, 0.1), (g, 0.6)\}$, $tr_1 = r \xrightarrow{a} \delta_{🟥}$, $tr_2 = y \xrightarrow{a} \delta_{🔺}$, $tr_3 = g \xrightarrow{a} \delta_{🟢}$, $tr_4 = r \xrightarrow{\tau} \delta_{\bar{s}}$, and $tr_5 = g \xrightarrow{\tau} \delta_{\bar{s}}$. The network $G(\bar{s}, a, \mu, \mathcal{R})$ is shown in Figure 3, where we omit the state vertices 🟥, 🔺, and 🟢 as well as the transition vertices r^{tr_1}, y^{tr_2}, and g^{tr_3} since they are not involved in any arc of the network.

It is worthwhile to note that for $a \in E$, each path in the network graph from $\vartriangle$ to $\blacktriangledown$ has to pass through a transition vertex v_a^{tr} where $act(tr) = a$, i.e., $r_a^{tr_1}$, $y_a^{tr_2}$, or $g_a^{tr_3}$. This construction ensures that the external action is performed with probability 1. ♦

5.2. LP Problem Construction

As pointed out in [TH15], the fact that the network admits a flow that respects the probability measure μ_t does by itself not imply the existence of a corresponding weak transition, because the flow may not respect probability ratios. To account for the latter, the network is converted into a linear programming problem for which the feasibility is shown to be equivalent to the existence of the desired weak transition. The idea is to convert the flow network into the canonical LP problem and then add the balancing constraints that force the "flow" to split according to transition probability measures.

Definition 8 (cf. [TH15, Definition 7]). Given a *PA* $\mathcal{A}$, a state $t \in S$, an action $a \in \Sigma$, a probability

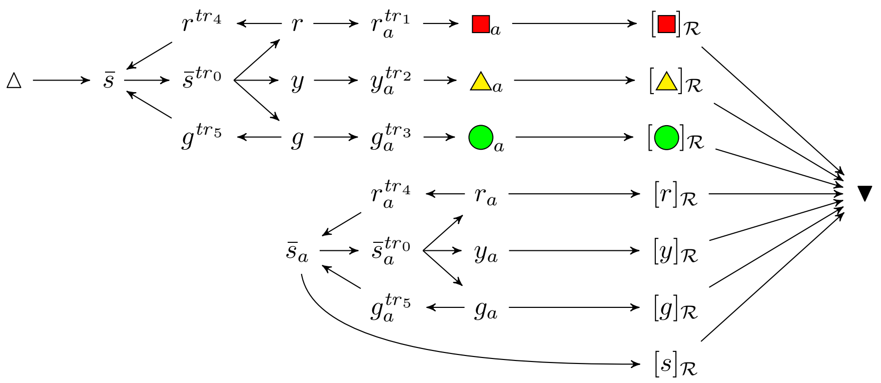


Figure 3. The network $G(\bar{s}, a, \mu, \mathcal{R})$ of Example 3

measure $\mu \in \mathrm{Disc}(S)$, and a binary relation $\mathcal{R}$ on S, for $a \in E$ we define the LP problem $LP(t, a, \mu, \mathcal{R})$ associated to the network graph $(V, E) = G(t, a, \mu, \mathcal{R})$ as follows.

$\max \sum_{(u,v)\in E} -f_{u,v}$	
subject to	
$f_{u,v} \geq 0$	for each $(u, v) \in E$
$f_{\triangle,t} = 1$	
$f_{\mathcal{C},\blacktriangledown} = \mu(\mathcal{C})$	for each $\mathcal{C} \in S/\mathcal{R}$
$\sum_{(u,v)\in E} f_{u,v} - \sum_{(v,w)\in E} f_{v,w} = 0$	for each $v \in V \setminus \{\triangle, \blacktriangledown\}$
$f_{v^{tr},v'} - \rho(v') \cdot f_{v,v^{tr}} = 0$	for each $tr = v \xrightarrow{\tau} \rho \in T$ and $v' \in \mathrm{Supp}(\rho)$
$f_{v_a^{tr},v'_a} - \rho(v') \cdot f_{v_a,v_a^{tr}} = 0$	for each $tr = v \xrightarrow{\tau} \rho \in T$ and $v' \in \mathrm{Supp}(\rho)$
$f_{v_a^{tr},v'_a} - \rho(v') \cdot f_{v,v_a^{tr}} = 0$	for each $tr = v \xrightarrow{a} \rho \in T$ and $v' \in \mathrm{Supp}(\rho)$

When $a \in H$, the LP problem $LP(t, a, \mu, \mathcal{R})$ associated to $G(t, a, \mu, \mathcal{R})$ is defined as above without the last two groups of constraints:

$\max \sum_{(u,v)\in E} -f_{u,v}$	
subject to	
$f_{u,v} \geq 0$	for each $(u, v) \in E$
$f_{\triangle,t} = 1$	
$f_{\mathcal{C},\blacktriangledown} = \mu(\mathcal{C})$	for each $\mathcal{C} \in S/\mathcal{R}$
$\sum_{(u,v)\in E} f_{u,v} - \sum_{(v,w)\in E} f_{v,w} = 0$	for each $v \in V \setminus \{\triangle, \blacktriangledown\}$
$f_{v^{tr},v'} - \rho(v') \cdot f_{v,v^{tr}} = 0$	for each $tr = v \xrightarrow{\tau} \rho \in T$ and $v' \in \mathrm{Supp}(\rho)$

Example 4. Consider again the automaton $\mathcal{E}$ from Example 1 (depicted in Figure 1) and a weak transition $\bar{s} \overset{a}{\Longrightarrow}_{\mathrm{c}} \rho$ such that $\rho\ \mathcal{L}(\mathcal{R})\ \mu$ where $\mu = \{($🟥$, \frac{9}{50}), ($▲$, \frac{8}{50}), ($🟢$, \frac{33}{50})\}$ and $\mathcal{R} = \mathcal{I}$. As in Example 3, since $\mathcal{I}$ is the identity relation, we have that $\rho = \mu$. Denote as usual the transitions of $\mathcal{E}$ as follows: $tr_0 = \bar{s} \xrightarrow{\tau} \{(r, 0.25), (y, 0.25), (g, 0.5)\}$, $tr_1 = r \xrightarrow{a} \delta_{\text{🟥}}$, $tr_2 = y \xrightarrow{a} \delta_{\text{▲}}$, $tr_3 = g \xrightarrow{a} \delta_{\text{🟢}}$, $tr_4 = r \xrightarrow{\tau} \delta_{\bar{s}}$, and $tr_5 = g \xrightarrow{\tau} \delta_{\bar{s}}$.

Besides the constraints for the non-negativity of the variables, the LP problem $LP(\bar{s}, a, \mu, \mathcal{R})$ has the following constraints:

- initial flow and challenging probabilities:

$f_{\triangle,\bar{s}} = 1$	$f_{[\text{🟥}]_\mathcal{R},\blacktriangledown} = 9/50$	$f_{[\text{▲}]_\mathcal{R},\blacktriangledown} = 8/50$
$f_{[\text{🟢}]_\mathcal{R},\blacktriangledown} = 33/50$	$f_{[\bar{s}]_\mathcal{R},\blacktriangledown} = 0$	$f_{[r]_\mathcal{R},\blacktriangledown} = 0$
$f_{[y]_\mathcal{R},\blacktriangledown} = 0$	$f_{[g]_\mathcal{R},\blacktriangledown} = 0$	

- conservation of the flow for vertices in S:

$f_{\triangle,\bar{s}} + f_{r^{tr_4},\bar{s}} + f_{g^{tr_5},\bar{s}} - f_{\bar{s},\bar{s}^{tr_0}} = 0$	$f_{\bar{s}^{tr_0},r} - f_{r,r_a^{tr_1}} - f_{r,r^{tr_4}} = 0$
$f_{\bar{s}^{tr_0},y} - f_{y,y_a^{tr_2}} = 0$	$f_{\bar{s}^{tr_0},g} - f_{g,g_a^{tr_3}} - f_{g,g^{tr_5}} = 0$

- conservation of the flow for vertices in S^{tr}:

$$\begin{array}{ll}
f_{\bar{s},\bar{s}^{tr_0}} - f_{\bar{s}^{tr_0},r} - f_{\bar{s}^{tr_0},y} - f_{\bar{s}^{tr_0},g} = 0 & f_{r,r^{tr_4}} - f_{r^{tr_4},\bar{s}} = 0 \\
f_{g,g^{tr_5}} - f_{g^{tr_5},\bar{s}} = 0 &
\end{array}$$

- conservation of the flow for vertices in S_a:

$$\begin{array}{ll}
f_{r_a^{tr_4},\bar{s}_a} + f_{g_a^{tr_5},\bar{s}_a} - f_{\bar{s}_a,\bar{s}_a^{tr_0}} - f_{\bar{s}_a,[\bar{s}]_{\mathcal{R}}} = 0 & f_{r_a^{tr_1},\blacksquare_a} - f_{\blacksquare_a,[\blacksquare]_{\mathcal{R}}} = 0 \\
f_{\bar{s}_a^{tr_0},r_a} - f_{r_a,r_a^{tr_4}} - f_{r_a,[r]_{\mathcal{R}}} = 0 & f_{y_a^{tr_2},\blacktriangle_a} - f_{\blacktriangle_a,[\blacktriangle]_{\mathcal{R}}} = 0 \\
f_{\bar{s}_a^{tr_0},y_a} - f_{y_a,[y]_{\mathcal{R}}} = 0 & f_{g_a^{tr_3},\bullet_a} - f_{\bullet_a,[\bullet]_{\mathcal{R}}} = 0 \\
f_{\bar{s}_a^{tr_0},g_a} - f_{g_a,g_a^{tr_5}} - f_{g_a,[g]_{\mathcal{R}}} = 0 &
\end{array}$$

- conservation of the flow for vertices in S_a^{tr}:

$$\begin{array}{ll}
f_{\bar{s}_a,\bar{s}_a^{tr_0}} - f_{\bar{s}_a^{tr_0},r_a} - f_{\bar{s}_a^{tr_0},y_a} - f_{\bar{s}_a^{tr_0},g_a} = 0 & f_{r,r_a^{tr_1}} - f_{r_a^{tr_1},\blacksquare_a} = 0 \\
f_{r_a,r_a^{tr_4}} - f_{r_a^{tr_4},\bar{s}_a} = 0 & f_{y,y_a^{tr_2}} - f_{y_a^{tr_2},\blacktriangle_a} = 0 \\
f_{g_a,g_a^{tr_5}} - f_{g_a^{tr_5},\bar{s}_a} = 0 & f_{g,g_a^{tr_3}} - f_{g_a^{tr_3},\bullet_a} = 0
\end{array}$$

- conservation of the flow for vertices in $S/\mathcal{R}$:

$$\begin{array}{ll}
f_{\bar{s}_a,[\bar{s}]_{\mathcal{R}}} - f_{[\bar{s}]_{\mathcal{R}},\blacktriangledown} = 0 & f_{\blacksquare_a,[\blacksquare]_{\mathcal{R}}} - f_{[\blacksquare]_{\mathcal{R}},\blacktriangledown} = 0 \\
f_{r_a,[r]_{\mathcal{R}}} - f_{[r]_{\mathcal{R}},\blacktriangledown} = 0 & f_{\blacktriangle_a,[\blacktriangle]_{\mathcal{R}}} - f_{[\blacktriangle]_{\mathcal{R}},\blacktriangledown} = 0 \\
f_{y_a,[y]_{\mathcal{R}}} - f_{[y]_{\mathcal{R}},\blacktriangledown} = 0 & f_{\bullet_a,[\bullet]_{\mathcal{R}}} - f_{[\bullet]_{\mathcal{R}},\blacktriangledown} = 0 \\
f_{g_a,[g]_{\mathcal{R}}} - f_{[g]_{\mathcal{R}},\blacktriangledown} = 0 &
\end{array}$$

- balancing constraints for τ-transitions generating L_1:

$$\begin{array}{ll}
f_{\bar{s}^{tr_0},r} - 0.3 \cdot f_{\bar{s},\bar{s}^{tr_0}} = 0 & f_{\bar{s}^{tr_0},y} - 0.1 \cdot f_{\bar{s},\bar{s}^{tr_0}} = 0 \\
f_{\bar{s}^{tr_0},g} - 0.6 \cdot f_{\bar{s},\bar{s}^{tr_0}} = 0 & f_{r^{tr_4},\bar{s}} - 1 \cdot f_{r,r^{tr_4}} = 0 \\
f_{g^{tr_5},\bar{s}} - 1 \cdot f_{g,g^{tr_5}} = 0 &
\end{array}$$

- balancing constraints for a-transitions generating L_a:

$$\begin{array}{ll}
f_{r_a^{tr_1},\blacksquare_a} - 1 \cdot f_{r,r_a^{tr_1}} = 0 & f_{y_a^{tr_2},\blacktriangle_a} - 1 \cdot f_{y,y_a^{tr_2}} = 0 \\
f_{g_a^{tr_3},\bullet_a} - 1 \cdot f_{g,g_a^{tr_3}} = 0 &
\end{array}$$

- balancing constraints for τ-transitions generating L_2:

$$\begin{array}{ll}
f_{\bar{s}_a^{tr_0},r_a} - 0.3 \cdot f_{\bar{s}_a,\bar{s}_a^{tr_0}} = 0 & f_{\bar{s}_a^{tr_0},y_a} - 0.1 \cdot f_{\bar{s}_a,\bar{s}_a^{tr_0}} = 0 \\
f_{\bar{s}_a^{tr_0},g_a} - 0.6 \cdot f_{\bar{s}_a,\bar{s}_a^{tr_0}} = 0 & f_{r_a^{tr_4},\bar{s}_a} - 1 \cdot f_{r_a,r_a^{tr_4}} = 0 \\
f_{g_a^{tr_5},\bar{s}_a} - 1 \cdot f_{g_a,g_a^{tr_5}} = 0 &
\end{array}$$

A solution that maximizes the objective function sets all variables to the value 0 except for the following variables:

$f_{\triangle,\bar{s}}$	$= 50/50$	$f_{\bar{s},\bar{s}^{tr_0}}$	$= 80/50$	$f_{\bar{s}^{tr_0},r}$	$= 24/50$
$f_{\bar{s}^{tr_0},y}$	$= 8/50$	$f_{\bar{s}^{tr_0},g}$	$= 48/50$	$f_{r,r_a^{tr_1}}$	$= 9/50$
$f_{r,r^{tr_4}}$	$= 15/50$	$f_{y,y_a^{tr_2}}$	$= 8/50$	$f_{g,g_a^{tr_3}}$	$= 33/50$
$f_{g,g^{tr_5}}$	$= 15/50$	$f_{r^{tr_4},\bar{s}}$	$= 15/50$	$f_{g^{tr_5},\bar{s}}$	$= 15/50$
$f_{r_a^{tr_1},\blacksquare_a}$	$= 9/50$	$f_{y_a^{tr_2},\blacktriangle_a}$	$= 8/50$	$f_{g_a^{tr_3},\bullet_a}$	$= 33/50$
$f_{\blacksquare_a,[\blacksquare]_{\mathcal{R}}}$	$= 9/50$	$f_{\blacktriangle_a,[\blacktriangle]_{\mathcal{R}}}$	$= 8/50$	$f_{\bullet_a,[\bullet]_{\mathcal{R}}}$	$= 33/50$
$f_{[\blacksquare]_{\mathcal{R}},\blacktriangledown}$	$= 9/50$	$f_{[\blacktriangle]_{\mathcal{R}},\blacktriangledown}$	$= 8/50$	$f_{[\bullet]_{\mathcal{R}},\blacktriangledown}$	$= 33/50$

It is worthwhile to note the value 80/50 for the variable $f_{\bar{s},\bar{s}^{tr_0}}$: this is caused by the fact that the arc $(\bar{s}, \bar{s}^{tr_0})$ is part of a cycle and its flow value is greater than 1, confirming that 1, the maximum probability, in general is not a proper value for arc capacities, as discussed in [TH15]. ♦

In the LP problem described in Definition 8, the objective function maximizes the total sum of negated flow routed along the arcs of the network. In fact, the total flow is described as the sum of negated flow variables which are positive themselves. This prevents routing large amounts of flow over disconnected components of the network or over cycles that can be ignored. Furthermore, in the LP problem, there are two different sets of constraints. The first set is the ordinary set of flow conservation constraints which require the total flow incoming and outgoing a node of the network to be equal. The second set is the set of balancing constraints that require the entering amount of flow to a transition node to be distributed based on probabilities assigned to the outgoing arcs.

It is easy to observe that the $LP(t, a, \mu, \mathcal{R})$ LP problem has size that is quadratic in the size $N = \mathit{size}(\mathcal{A})$: the number of variables is at most $3N^2 + 5N + 1$ while the number of constraints is at most $6N^2 + 11N + 2$. Moreover, it is also worthwhile to spell out the number of transition, state, and class nodes of the network $G(t, a, \mu, \mathcal{R})$: there are at most $2|T|$ transition nodes, at most $2|S|$ state nodes, and at most $|S|$ class nodes.

The equivalence of the LP problem and the weak transition is formalized by Theorem 9 and Corollary 12(1) of [TH15]:

Proposition 2. A weak transition $t \overset{a}{\Longrightarrow}_{\mathrm{c}} \mu_t$ such that $\mu \ \mathcal{L}(\mathcal{R}) \ \mu_t$ exists if and only if the LP problem $LP(t, a, \mu, \mathcal{R})$ has a feasible solution.

Remark 4. The LP problem construction proposed in Definition 8 is not easily extendable to weak non-combined transitions induced by a Dirac scheduler. In fact, in order to obtain for such setting a result equivalent to Proposition 2, we should enforce that the flow leaving the nodes v and v_a is not split among several outgoing arcs, but it is routed completely to a single arc. To obtain such a situation, we should replace, for each $v \in S \cup S_a$, the flow conservation constraint in Definition 8

$$\textstyle\sum_{(u,v)\in E} f_{u,v} - \sum_{(v,w)\in E} f_{v,w} = 0 \qquad \text{for each } v \in V \setminus \{\vartriangle, \blacktriangledown\}$$

by the following set of constraints:

$$\begin{array}{ll}\sum_{(u,v)\in E} f_{u,v} - \sum_{(v,w)\in E} \alpha_{v,w} f_{v,w} = 0 & \text{for each } v \in V \setminus \{\vartriangle, \blacktriangledown\} \\ \sum_{(v,w)\in E} \alpha_{v,w} = 1 & \text{for each } v \in V \setminus \{\vartriangle, \blacktriangledown\} \\ \alpha_{v,w} \in \{0, 1\} & \text{for each } (v, w) \in E\end{array}$$

The latter ensures that the flow is sent through a single outgoing arc in its entirety. This change implies that the resulting problem is no longer a Linear Programming problem but a Mixed Integer Nonlinear Programming problem (MINLP), known to belong to the class of NP-complete problems [Sch03]. While it is rather easy to show that the problem of finding a weak transition induced by a Dirac scheduler is equivalent to the above MINLP problem (the proof is essentially the same of the one of Proposition 2, see [TH15, Lemmas 7 and 8]), such an equivalence is not sufficient to establish the NP-completeness of the problem. However, it is still possible to show such a result by a direct reduction from the 3-SAT problem.

First, we introduce some terminology about satisfiability of formulas. Given a set V of variables taking values in $\{\mathtt{t}, \mathtt{f}\}$, a *literal* l is either a variable v or the negation of a variable $\neg v$, where $v \in V$. A *clause* Cl is a disjunction of literals. A formula ϕ is written in conjunctive normal form with three variables per clause (*3-CNF*) if $\phi = \bigwedge_{i=1}^{n} \mathit{Cl}_i$ where each clause Cl_i is a disjunction of three literals. To simplify the presentation, we assume that each clause contains distinct literals. A formula ϕ is *satisfiable* if there exists a logical value assignment for the variables that makes the formula true. Given a formula ϕ, we denote by $\mathit{Var}(\phi)$ the set of variables occurring in ϕ, by $\mathit{Lit}(\phi)$ the set of literals occurring in ϕ, by $\mathit{Cl}(\phi)$ the set of clauses of ϕ, and, given a literal l, we denote by $\mathit{Cl}(\phi, l)$ the set of clauses of ϕ where l occurs.

Proposition 3. Given a *PA* $\mathcal{A}$, a state $s \in S$, an action $a \in \Sigma$, and a probability measure $\mu \in \mathrm{Disc}(S)$, checking whether there exists a Dirac scheduler inducing $s \overset{a}{\Longrightarrow} \mu$ is NP-complete.

Proof. To prove the claim, we have to show two results: the problem is NP-hard and belongs to NP.

The fact that the problem belongs to NP follows directly from the fact that the existence of a weak transition induced by a Dirac scheduler can be encoded as a MINLP problem, that is in NP.

For showing the NP-hardness, we provide a reduction from the 3-SAT problem. Let $\phi = \bigwedge_{i=1}^{n} \mathit{Cl}_i$ be a 3-CNF formula, $n = |\mathit{Cl}(\phi)|$, and $m = |\mathit{Var}(\phi)|$.

Consider the *PA* $\mathcal{A}_\phi$ whose set of states is $S = \{\phi, \triangledown\} \cup \mathit{Var}(\phi) \cup \{\, v^{\mathtt{f}}, v^{\mathtt{t}} \mid v \in \mathit{Var}(\phi)\,\} \cup \mathit{Cl}(\phi)$, whose start state is ϕ, whose set of actions is $\Sigma = \{\tau\}$, and whose transitions are:

$$\begin{aligned} T = {} & \{\phi \xrightarrow{\tau} \upsilon_{\mathit{Var}(\phi)}\} \cup \{\, v \xrightarrow{\tau} \delta_{v^{\mathtt{t}}}, v \xrightarrow{\tau} \delta_{v^{\mathtt{f}}} \mid v \in \mathit{Var}(\phi)\,\} \\ & \cup \{\, v^{\mathtt{t}} \xrightarrow{\tau} \rho_v \mid v \in \mathit{Lit}(\phi)\,\} \cup \{\, v^{\mathtt{f}} \xrightarrow{\tau} \rho_{\neg v} \mid \neg v \in \mathit{Lit}(\phi)\,\} \\ & \cup \{\, \mathit{Cl} \xrightarrow{\tau} \{(\mathit{Cl}, \frac{1}{k}), (\triangledown, \frac{k-1}{k})\} \mid \mathit{Cl} \in \mathit{Cl}(\phi), k \in \{1,2,3\}\,\}, \end{aligned}$$

where, for a literal l, ρ_l is defined as

$$\rho_l(t) = \begin{cases} \frac{1}{n} & \text{if } t \in \mathit{Cl}(\phi, l), \\ \frac{|\mathit{Cl}(\phi) \setminus \mathit{Cl}(\phi,l)|}{n} & \text{if } t = \triangledown, \\ 0 & \text{otherwise.} \end{cases}$$

We now prove that ϕ is satisfiable if and only if $\mathcal{A}_\phi$ exhibits the weak transition $\phi \overset{\tau}{\Longrightarrow} \mu$ where

$$\mu(t) = \begin{cases} \frac{1}{n \cdot m} & \text{if } t = \mathit{Cl} \text{ for some clause } \mathit{Cl} \in \mathit{Cl}(\phi), \\ 1 - \frac{1}{m} & \text{if } t = \triangledown, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Suppose that ϕ is satisfiable; this implies that there exists a truth value assignment for the variables occurring in ϕ that makes the formula true. Moreover, since ϕ is satisfiable, it follows that at least one literal of each clause $\mathit{Cl} \in \mathit{Cl}(\phi)$ has assignment $\mathtt{t}$. Let σ be the Dirac scheduler defined as follows:

$$\sigma(\alpha) = \begin{cases} \delta_{\phi \xrightarrow{\tau} v_{\mathit{Var}(\phi)}} & \text{if } \alpha = \phi, \\ \delta_{v \xrightarrow{\tau} \delta_{v^{\mathtt{t}}}} & \text{if } \alpha = \phi\tau v \text{ and } v \text{ is } \mathtt{t} \text{ in the assignment,} \\ \delta_{v \xrightarrow{\tau} \delta_{v^{\mathtt{f}}}} & \text{if } \alpha = \phi\tau v \text{ and } v \text{ is } \mathtt{f} \text{ in the assignment,} \\ \delta_{v^{\mathtt{t}} \xrightarrow{\tau} \rho_v} & \text{if } \alpha = \phi\tau v\tau v^{\mathtt{t}} \text{ and } v \in \mathit{Lit}(\phi), \\ \delta_{v^{\mathtt{f}} \xrightarrow{\tau} \rho_{\neg v}} & \text{if } \alpha = \phi\tau v\tau v^{\mathtt{f}} \text{ and } \neg v \in \mathit{Lit}(\phi), \\ \delta_{\mathit{Cl} \xrightarrow{\tau} \{(\mathit{Cl}, \frac{1}{k}), (\triangledown, \frac{k-1}{k})\}} & \text{if } \alpha \in \{\phi\tau v\tau v^{\mathtt{t}}\tau \mathit{Cl}, \phi\tau v\tau v^{\mathtt{f}}\tau \mathit{Cl}\} \text{ and exactly } k \text{ literals of } \mathit{Cl} \text{ are } \mathtt{t}, \\ \delta_\bot & \text{otherwise.} \end{cases}$$

It is rather easy to verify that σ actually induces the weak transition $\phi \overset{\tau}{\Longrightarrow} \mu$. Consider, for instance, a clause Cl; let $\mathit{Cl} = l_1 \vee l_2 \vee l_3$ and v_i be the variable associated to the literal l_i. The probability of reaching Cl is:

$$\begin{aligned} & \mu_{\sigma,\phi}(\{\, \alpha \in \mathit{frags}^*(\mathcal{A}_\phi) \mid \mathit{last}(\alpha) = \mathit{Cl} \,\}) \\ = {} & \mu_{\sigma,\phi}(\{\phi\tau v_1 \tau v_1^{\mathtt{v}} \tau \mathit{Cl} \tau \mathit{Cl}\}) && \text{if } l_1 = \mathtt{t} \text{ and } \mathtt{v} \text{ is the assignment of } v_1 \\ & + \mu_{\sigma,\phi}(\{\phi\tau v_2 \tau v_2^{\mathtt{v}} \tau \mathit{Cl} \tau \mathit{Cl}\}) && \text{if } l_2 = \mathtt{t} \text{ and } \mathtt{v} \text{ is the assignment of } v_2 \\ & + \mu_{\sigma,\phi}(\{\phi\tau v_3 \tau v_3^{\mathtt{v}} \tau \mathit{Cl} \tau \mathit{Cl}\}) && \text{if } l_3 = \mathtt{t} \text{ and } \mathtt{v} \text{ is the assignment of } v_3 \end{aligned}$$

For each $i \in \{1, 2, 3\}$ such that $l_i = \mathtt{t}$, we have that $\mu_{\sigma,\phi}(\{\phi\tau v_i \tau v_i^{\mathtt{v}} \tau \mathit{Cl} \tau \mathit{Cl}\}) = \frac{1}{m} \cdot \frac{1}{n} \cdot \frac{1}{k}$, where k is the number of literals of Cl that are $\mathtt{t}$. In fact, for each $i \in \{1, 2, 3\}$ such that $l_i = \mathtt{t}$,

$$\begin{aligned} & \mu_{\sigma,\phi}(\{\phi\tau v_i \tau v_i^{\mathtt{v}} \tau \mathit{Cl} \tau \mathit{Cl}\}) \\ = {} & \mu_{\sigma,\phi}(C_{\phi\tau v_i \tau v_i^{\mathtt{v}} \tau \mathit{Cl} \tau \mathit{Cl}}) \cdot \sigma(\phi\tau v_i \tau v_i^{\mathtt{v}} \tau \mathit{Cl} \tau \mathit{Cl})(\bot) \\ = {} & \mu_{\sigma,\phi}(C_\phi) \cdot \Big(\sum_{tr \in T(\tau)} \sigma(\phi)(tr) \cdot \mu_{tr}(v_i) \Big) \\ & \cdot \Big(\sum_{tr \in T(\tau)} \sigma(\phi\tau v_i)(tr) \cdot \mu_{tr}(v_i^{\mathtt{v}}) \Big) \\ & \cdot \Big(\sum_{tr \in T(\tau)} \sigma(\phi\tau v_i \tau v_i^{\mathtt{v}})(tr) \cdot \mu_{tr}(\mathit{Cl}) \Big) \\ & \cdot \Big(\sum_{tr \in T(\tau)} \sigma(\phi\tau v_i \tau v_i^{\mathtt{v}} \tau \mathit{Cl})(tr) \cdot \mu_{tr}(\mathit{Cl}) \Big) \\ & \cdot \sigma(\phi\tau v_i \tau v_i^{\mathtt{v}} \tau \mathit{Cl} \tau \mathit{Cl})(\bot) \end{aligned}$$

(In the following step, we omit the transitions chosen by σ with probability 0; for instance, $\phi \xrightarrow{\tau} \upsilon_{Var(\phi)}$ when $\alpha = \phi\tau v_i$. For improving readability, we write θ_{Cl} for the distribution $\{(Cl, \frac{1}{k}), (\nabla, \frac{k-1}{k})\}$.)

$$\begin{aligned}
&= \mu_{\sigma,\phi}(C_\phi) \cdot \Big(\sigma(\phi)(\phi \xrightarrow{\tau} \upsilon_{Var(\phi)}) \cdot \upsilon_{Var(\phi)}(v_i)\Big)\\
&\qquad \cdot \Big(\sigma(\phi\tau v_i)(v_i \xrightarrow{\tau} \delta_{v_i^{\mathtt{v}}}) \cdot \delta_{v_i^{\mathtt{v}}}(v_i^{\mathtt{v}})\Big)\\
&\qquad \cdot \Big(\sigma(\phi\tau v_i\tau v_i^{\mathtt{v}})(v_i^{\mathtt{v}} \xrightarrow{\tau} \rho_l) \cdot \rho_l(Cl)\Big)\\
&\qquad \cdot \Big(\sigma(\phi\tau v_i\tau v_i^{\mathtt{v}}\tau Cl)(Cl \xrightarrow{\tau} \theta_{Cl}) \cdot \theta_{Cl}(Cl)\Big)\\
&\qquad \cdot \sigma(\phi\tau v_i\tau v_i^{\mathtt{v}}\tau Cl\tau Cl)(\bot)\\
&= 1 \cdot \Big(\frac{1}{m}\Big) \cdot \Big(1\Big) \cdot \Big(\frac{1}{n}\Big) \cdot \Big(\frac{1}{k}\Big) \cdot 1 = \frac{1}{m} \cdot \frac{1}{n} \cdot \frac{1}{k}
\end{aligned}$$

Since $\mu_{\sigma,\phi}(\{\phi\tau v_i\tau v_i^{\mathtt{v}}\tau Cl\tau Cl\}) = \frac{1}{m} \cdot \frac{1}{n} \cdot \frac{1}{k}$ holds for each $i \in \{1, 2, 3\}$ such that $l_i = \mathtt{t}$, it follows that, for k literals being $\mathtt{t}$ in Cl, the overall probability assigned to the state Cl is $k \cdot \frac{1}{m} \cdot \frac{1}{n} \cdot \frac{1}{k} = \frac{1}{n \cdot m}$ as required. Since this probability is independent from the particular Cl, the overall probability assigned to $Cl(\phi)$ is $n \times \frac{1}{n \cdot m} = \frac{1}{m}$; the remaining probability value $1 - \frac{1}{m}$ is assigned to ∇, as required, as it can be easily checked in a similar way. The other properties the scheduler has to satisfy trivially follow from the previous one and the fact that the *PA* $\mathcal{A}_\phi$ has $E = \emptyset$, so σ actually induces the weak transition $\phi \stackrel{\tau}{\Longrightarrow} \mu$. This completes the proof that if ϕ is satisfiable, then there is a Dirac scheduler inducing $\phi \stackrel{\tau}{\Longrightarrow} \mu$.

Now, suppose that there exists a Dirac scheduler inducing $\phi \stackrel{\tau}{\Longrightarrow} \mu$. We want to derive a logical value assigment such that the formula ϕ holds. For each variable $v \in Var(\phi)$, define the assignment $\theta(v)$ as follows:

$$\theta(v) = \begin{cases} \mathtt{t} & \text{if } \sigma(\phi\tau v) = \delta_{v \xrightarrow{\tau} \delta_{v^{\mathtt{t}}}}, \\ \mathtt{f} & \text{otherwise.} \end{cases}$$

Since by hypothesis each clause Cl is reached with probability $\frac{1}{n \cdot m}$, it means that there exists at least one finite execution fragment of the form $\phi\tau v\tau v^{\mathtt{v}}\tau Cl\tau Cl$ that occurs with non-zero probability. In particular,

$$\mathtt{v} = \begin{cases} \mathtt{t} & \text{if } \sigma(\phi\tau v) = \delta_{v \xrightarrow{\tau} \delta_{v^{\mathtt{t}}}}, \\ \mathtt{f} & \text{otherwise,} \end{cases}$$

i.e., v has truth value $\mathtt{v}$. Moreover, the existence of such execution fragment implies that the literal v occurs in Cl if $\mathtt{v} = \mathtt{t}$ or the literal $\neg v$ occurs in Cl if $\mathtt{v} = \mathtt{f}$. The former case implies that $Cl = v \vee l' \vee l''$ for some literal l' and l'' with $v = \mathtt{t}$, while the latter case implies that $Cl = \neg v \vee l' \vee l''$ for some literal l' and l'' with $v = \mathtt{f}$. In both cases the clause Cl is satisfied, hence ϕ is satisfied as well since Cl is a generic clause in $Cl(\phi)$. This concludes the proof that if there exists Dirac scheduler inducing the weak transition $\phi \stackrel{\tau}{\Longrightarrow} \mu$, then ϕ is satisfiable.

Since we have shown that ϕ is satisfiable if and only if $\mathcal{A}_\phi$ exhibits the weak transition $\phi \stackrel{\tau}{\Longrightarrow} \mu$, in order to complete the reduction we have to show that the reduction is polynomial in the size of the formula ϕ: this follows immediately by the construction of $\mathcal{A}_\phi$ whose number of states and transitions is linear in the number of variables and clauses of ϕ. $\square$

5.3. Complexity Analysis of Deciding Weak Bisimulation

Proposition 2 allows us to verify the existence of a weak transition $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}} \mu_t$ such that $\mu_s \; \mathcal{L}(\mathcal{R}) \; \mu_t$ at line 3 of FINDSPLIT efficiently: $W(N)$ is actually $p(N)$ for some polynomial p, hence the following result holds.

Theorem 1. *Given two PAs $\mathcal{A}_1$ and $\mathcal{A}_2$, let $N = size(\mathcal{A}_1) + size(\mathcal{A}_2)$. Checking $\mathcal{A}_1 \approx \mathcal{A}_2$ is polynomial in N.*

6. Efficiency of Solving the LP Problem

The analysis of the $LP(t, a, \mu, \mathcal{R})$ LP problem formalized in [TH15, Proposition 6] considers the theoretical complexity class the problem belongs to. It does not address how efficiently the LP problem can indeed be solved. Practical implementation aspects and empirical results will be presented in Section 7. To prepare for that, we first discuss abstract observations concerning the worst case running time needed to solve the LP problem. Then we recast the LP problem into a flow network problem and exploit the underlying network structure to arrive at an efficient LP problem solution approach harvesting an algorithm in the network optimization setting. We further discuss various alternative approaches to improve solution efficiency, including approximative methods.

6.1. Efficient Solution: Theory

Throughout this section, we define the dimension of an input to an algorithm as the number of data items in the input. The size of a rational number p/q is defined as the length of its binary description, i.e., $\mathit{size}(p/q) = \lceil \log_2(p + 1)\rceil + \lceil \log_2(q + 1)\rceil$, where $\lceil x\rceil$ denotes the smallest integer not less than x. The size of a rational vector or matrix is defined as the sum of the sizes of its entries.

Deciding the existence of a weak transition in a probabilistic automaton can be done in polynomial time [TH15, Proposition 6 and Theorem 8]. With the aim to refine this result, we discuss the problem in the context of the restricted class of rational probabilistic automata.

Rational *PA*s We start our analysis with the class of rational *PA*s.

Definition 9. Given a *PA* $\mathcal{A}$, we say that $\mathcal{A}$ is *rational* if for each $(s, a, \mu) \in T$ and $v \in \mathrm{Supp}(\mu)$, we have that $\mu(v) \in \mathbb{Q}$.

For this class of *PA*s, we look for a tighter worst case complexity bound of solving the LP problem $LP(t, a, \mu, \mathcal{R})$. We proceed via a reformulation that reduces the size of $LP(t, a, \mu, \mathcal{R})$. This size reduction directly reduces the solution effort needed for the LP problem, since the latter depends on the number of variables and constraints, and this will indeed provide a tighter worst case bound. To reach our goal, we modify the network provided in Definition 7 and reformulate the original LP problem on the basis of these changes.

Consider the network $G(t, a, \mu, \mathcal{R})$ and let $\mathcal{G}(t, a, \mu, \mathcal{R})$ be a directed network which is generated from the network $G(t, a, \mu, \mathcal{R})$ by removing the source node $\vartriangle$ and the sink node $\blacktriangledown$; let $\mathcal{V} = V \setminus \{\vartriangle, \blacktriangledown\}$ and $\mathcal{E} = E \setminus (\{(\vartriangle, t)\} \cup \{\, (\mathcal{C}, \blacktriangledown) \mid \mathcal{C} \in S/\mathcal{R}\,\})$ be the set of vertices and directed arcs of $\mathcal{G}(t, a, \mu, \mathcal{R})$, respectively. Moreover, let $\bar{\mathcal{E}} \subseteq \mathcal{E}$ be the set $\bar{\mathcal{E}} = \{\, (v^{tr}, v'), (v^{tr}_a, v'_a) \mid tr = v \xrightarrow{\tau} \rho \in T, v' \in \mathrm{Supp}(\rho)\,\} \cup \{\, (v^{tr}_a, v'_a) \mid tr = v \xrightarrow{a} \rho \in T, v' \in \mathrm{Supp}(\rho)\,\}$. Then, we define $\rho_{i,j} = \mu_{tr}(v')$ as the proportionality coefficient corresponding to the arc $(i, j) \in \bar{\mathcal{E}}$ where $(i, j) = (v^{tr}, v')$ or $(i, j) = (v^{tr}_a, v'_a)$. Since in both original and modified networks each arc in $\bar{\mathcal{E}}$ belongs to a single transition, the corresponding proportional coefficient is uniquely determined.

For each node $u \in \mathcal{V}$, let b_u be a supply/demand value, that is, if $b_u > 0$ the node u is a supply node and if $b_u < 0$ the node u is a demand node. For the network $\mathcal{G}(t, a, \mu, \mathcal{R})$, we define b_u for each node $u \in \mathcal{V}$ so as to take value 1 if $u = t$, value $-\mu(\mathcal{C})$ if $u = \mathcal{C} \in S/\mathcal{R}$ and 0 otherwise. It is immediate to see that $\sum_{u\in\mathcal{V}} b_u = 0$. This fact can be seen as a feasibility condition in the corresponding flow network [AMO93]. For $s \in \mathcal{T}$, assume A_s to be the set of all arcs in the node-arc incidence matrix A that should have proportional flow. We define $\tilde{A}$ to be the subset of arcs in A that do not belong to any set A_s for $s \in \mathcal{T}$. More precisely, $\tilde{A} = A \setminus \bigcup_{s\in\mathcal{T}} A_s$. Based on the definitions, the $LP(t, a, \mu, \mathcal{R})$ LP problem can be reformulated as follows:

$$\begin{array}{rll}
\text{LP1: min} & \sum_{(i,j)\in\mathcal{E}} f_{i,j} & \\
\text{s.t.} & \sum_{(i,j)\in\mathcal{E}} f_{i,j} - \sum_{(j,i)\in\mathcal{E}} f_{j,i} = b_i & \text{for each } i \in \mathcal{V} \\
& \frac{f_{i,j}}{\rho_{i,j}} \text{ are all equal} & s \in \mathcal{T}, (i, j) \in A_s \\
& f_{i,j} \geq 0 & \text{for each } (i, j) \in \mathcal{E}
\end{array}$$

Lemma 1. The $LP(t, a, \mu, \mathcal{R})$ LP problem and LP1 are equivalent.

Proof. The statement follows immediately by a simple manipulation of the balancing constraints: consider the transition $tr = v \xrightarrow{\tau} \rho$; it is encoded in the network as the transition node v^{tr} and the arcs (v, v^{tr}) and

(v^{tr}, v') for $v' \in \mathrm{Supp}(\rho)$. The corresponding balancing constraints are $f_{v^{tr},v'} - \rho(v') \cdot f_{v,v^{tr}} = 0$, that is, $\frac{f_{v^{tr},v'}}{\rho(v')} = f_{v,v^{tr}}$. Since $f_{v,v^{tr}}$ is independent on v', it follows that the ratio $\frac{f_{v^{tr},v'}}{\rho(v')}$ is equal for all $v' \in \mathrm{Supp}(\rho)$, as required.

The same holds for the transition nodes v^{tr}_a and $v^{tr'}_a$, the latter corresponding to the transition $tr' = v \xrightarrow{a} \gamma$. $\square$

By assuming the unit flow cost $c_{i,j} = 1$ for each arc $(i,j) \in \mathcal{E}$, the objective of this problem is to minimise the total cost of routing the flow on network arcs subject to the ordinary flow conservation constraints, the proportional flow constraints corresponding to the balancing constraints of the original LP problem, and the arc flow lower bounds.

It is worthwhile to note that there exists a proportional flow set for each transition node in the network and that each arc may belong to at most one proportional flow set. The flow on the arcs in each of these flow proportional sets can be regarded as a single decision variable. Using this intuition, let $a_{i,j}$ denote the column corresponding to the arc (i,j) in the node-arc incidence matrix of the network $\mathcal{G}(t, a, \mu, \mathcal{R})$ and let $a_s = \sum_{(i,j)\in A_s} \rho_{i,j} \cdot a_{i,j}$ for each $s \in \mathcal{T}$. We denote by a^k_s the k-th component of the vector a_s. Since the column vector $a_{i,j}$ in the node-arc incidence matrix includes only entities 0, +1 and -1 therefore, the k-th component of the vector a_s, i.e., a^k_s can be equivalently written as $a^k_s = \sum_{(k,j)\in A_s} \rho_{k,j} - \sum_{(j,k)\in A_s} \rho_{j,k}$. By using the new notations, LP1 can be reformulated as the following LP problem which in turn can be regarded as an adaptation of the LP considered in [BF12].

$$\begin{array}{lll}
\text{LP2: min} & \sum_{(i,j)\in\tilde{A}} f_{i,j} + \sum_{s\in\mathcal{T}} f_s & \\
\text{s.t.} & \sum_{(i,j)\in\tilde{A}} f_{i,j} - \sum_{(j,i)\in\tilde{A}} f_{j,i} + \sum_{s\in\mathcal{T}} a^i_s \cdot f_s = b_i & \text{for each } i \in \mathcal{V} \\
& f_{i,j} \geq 0 & \text{for each } (i,j) \in \tilde{A} \\
& f_s \geq 0 & \text{for each } s \in \mathcal{T}
\end{array}$$

Lemma 2. LP1 and LP2 are equivalent.

Proof. Let $f = \{ f_{i,j} \mid (i,j) \in \tilde{A} \} \cup \{ f_s \mid s \in \mathcal{T} \}$ be a feasible solution for LP2. Define flow $\tilde{f}$ as follows:

$$\tilde{f}_{i,j} = \begin{cases} f_{i,j} & \text{if } (i,j) \in \tilde{A} \\ \rho_{i,j} \cdot f_s & \text{if } s \in \mathcal{T} \text{ and } (i,j) \in A_s \\ 0 & \text{otherwise.} \end{cases}$$

We claim that the flow $\tilde{f}$ satisfies the LP1 constrains. To show this, in the first set of constraints in LP1 and for each $i \in \mathcal{V}$ we get the following equivalences (comments refer to the previous equivalence):

$$\begin{aligned}
\sum_{(i,j)\in\mathcal{E}} \tilde{f}_{i,j} - \sum_{(j,i)\in\mathcal{E}} \tilde{f}_{j,i} &= \left(\sum_{(i,j)\in\tilde{A}} \tilde{f}_{i,j} + \sum_{(i,j)\in\mathcal{E}\setminus\tilde{A}} \tilde{f}_{i,j} \right) - \left(\sum_{(j,i)\in\tilde{A}} \tilde{f}_{j,i} + \sum_{(j,i)\in\mathcal{E}\setminus\tilde{A}} \tilde{f}_{j,i} \right) \\
&= \sum_{(i,j)\in\tilde{A}} \tilde{f}_{i,j} - \sum_{(j,i)\in\tilde{A}} \tilde{f}_{j,i} + \sum_{(i,j)\in\mathcal{E}\setminus\tilde{A}} \tilde{f}_{i,j} - \sum_{(j,i)\in\mathcal{E}\setminus\tilde{A}} \tilde{f}_{j,i} \\
&= \sum_{(i,j)\in\tilde{A}} \tilde{f}_{i,j} - \sum_{(j,i)\in\tilde{A}} \tilde{f}_{j,i} + \sum_{s\in\mathcal{T}} \sum_{(i,j)\in A_s} \tilde{f}_{i,j} - \sum_{s\in\mathcal{T}} \sum_{(j,i)\in A_s} \tilde{f}_{j,i}
\end{aligned}$$

by definition of $\tilde{A}$

$$= \sum_{(i,j)\in\tilde{A}} f_{i,j} - \sum_{(j,i)\in\tilde{A}} f_{j,i} + \sum_{s\in\mathcal{T}} \sum_{(i,j)\in A_s} \rho_{i,j} \cdot f_s - \sum_{s\in\mathcal{T}} \sum_{(j,i)\in A_s} \rho_{j,i} \cdot f_s$$

by definition of $\tilde{f}$

$$= \sum_{(i,j)\in\tilde{A}} f_{i,j} - \sum_{(j,i)\in\tilde{A}} f_{j,i} + \sum_{s\in\mathcal{T}} f_s \cdot \left(\sum_{(i,j)\in A_s} \rho_{i,j} - \sum_{(j,i)\in A_s} \rho_{j,i} \right)$$

by simple term manipulation

$$= \sum_{(i,j)\in\tilde{A}} f_{i,j} - \sum_{(j,i)\in\tilde{A}} f_{j,i} + \sum_{s\in\mathcal{T}} a^i_s \cdot f_s$$

by definition of a^i_s

$$= b_i$$

by definition of LP2. Moreover, for each $s \in \mathcal{T}$ and $(i,j) \in A_s$, $\frac{\tilde{f}_{i,j}}{\rho_{i,j}} = \frac{\rho_{i,j}\cdot f_s}{\rho_{i,j}} = f_s$. This means that for each $s \in \mathcal{T}$ and for all $(i,j) \in A_s$, $\frac{\tilde{f}_{i,j}}{\rho_{i,j}}$ are all equal. Also, for each $(i,j) \in \tilde{A}$, $\tilde{f}_{i,j} = f_{i,j} \geq 0$ and for each $s \in \mathcal{T}$ and $(i,j) \in A_s$, $\tilde{f}_{i,j} = \rho_{i,j} \cdot f_s \geq 0$. Therefore, $\tilde{f}_{i,j}$ for $(i,j) \in \mathcal{E}$ is indeed a feasible solution for LP1. Next, consider the value of the objective function for LP1:

$$\begin{aligned}\sum_{(i,j)\in\mathcal{E}} \tilde{f}_{i,j} &= \sum_{(i,j)\in\tilde{A}} \tilde{f}_{i,j} + \sum_{(i,j)\in\mathcal{E}\setminus\tilde{A}} \tilde{f}_{i,j} \\ &= \sum_{(i,j)\in\tilde{A}} \tilde{f}_{i,j} + \sum_{s\in\mathcal{T}} \sum_{(i,j)\in A_s} \tilde{f}_{i,j}\end{aligned}$$

by definition of $\tilde{A}$

$$= \sum_{(i,j)\in\tilde{A}} f_{i,j} + \sum_{s\in\mathcal{T}} \sum_{(i,j)\in A_s} \rho_{i,j} \cdot f_s$$

by definition of $\tilde{f}$

$$= \sum_{(i,j)\in\tilde{A}} f_{i,j} + \sum_{s\in\mathcal{T}} f_s \cdot \overbrace{\sum_{(i,j)\in A_s} \rho_{i,j}}^{1}$$

by simple term manipulation and the fact that $\rho_{i,j} = \mu_{tr}(v')$ where $(i,j) = (v^{tr}, v')$ or $(i,j) = (v^{tr}_a, v'_a)$

$$= \sum_{(i,j)\in\tilde{A}} f_{i,j} + \sum_{s\in\mathcal{T}} f_s.$$

Therefore, corresponding to this feasible solution, the value of the objective function of both LP problems are the same. For the reverse side, assume $\bar{f} = \{\, \bar{f}_{i,j} \mid (i,j) \in \mathcal{E} \,\}$ is a feasible solution for LP1. Define the flow $\hat{f} = \{\, \hat{f}_{i,j} \mid (i,j) \in \tilde{A} \,\} \cup \{\, \hat{f}_s \mid s \in \mathcal{T} \,\}$ where $\hat{f}_{i,j} = \bar{f}_{i,j}$ for $(i,j) \in \tilde{A}$ and $\hat{f}_s = \frac{\bar{f}_{i,j}}{\rho_{i,j}}$ for each $s \in \mathcal{T}$ where $(i,j) \in A_s$. In the following we show that $\hat{f}$ is a feasible solution for LP2. For each $i \in \mathcal{V}$, it holds:

$$\begin{aligned}&\sum_{(i,j)\in\tilde{A}} \hat{f}_{i,j} - \sum_{(j,i)\in\tilde{A}} \hat{f}_{j,i} + \sum_{s\in\mathcal{T}} a^i_s \cdot \hat{f}_s \\ &= \sum_{(i,j)\in\tilde{A}} \hat{f}_{i,j} - \sum_{(j,i)\in\tilde{A}} \hat{f}_{j,i} + \sum_{s\in\mathcal{T}} \left(\sum_{(i,j)\in A_s} \rho_{i,j} - \sum_{(j,i)\in A_s} \rho_{j,i} \right) \cdot \hat{f}_s\end{aligned}$$

by definition of a^i_s

$$= \sum_{(i,j)\in\tilde{A}} \hat{f}_{i,j} - \sum_{(j,i)\in\tilde{A}} \hat{f}_{j,i} + \sum_{s\in\mathcal{T}} \sum_{(i,j)\in A_s} \rho_{i,j} \cdot \hat{f}_s - \sum_{s\in\mathcal{T}} \sum_{(j,i)\in A_s} \rho_{j,i} \cdot \hat{f}_s$$

by simple term manipulation

$$= \sum_{(i,j)\in\tilde{A}} \bar{f}_{i,j} - \sum_{(j,i)\in\tilde{A}} \bar{f}_{j,i} + \sum_{s\in\mathcal{T}} \sum_{(i,j)\in A_s} \bar{f}_{i,j} - \sum_{s\in\mathcal{T}} \sum_{(j,i)\in A_s} \bar{f}_{j,i}$$

by definition of $\hat{f}$

$$= \sum_{(i,j)\in\tilde{A}} \bar{f}_{i,j} + \sum_{s\in\mathcal{T}} \sum_{(i,j)\in A_s} \bar{f}_{i,j} - \left(\sum_{(j,i)\in\tilde{A}} \bar{f}_{j,i} + \sum_{s\in\mathcal{T}} \sum_{(j,i)\in A_s} \bar{f}_{j,i} \right)$$

by simple term manipulation

$$= \sum_{(i,j)\in\tilde{A}} \bar{f}_{i,j} + \sum_{(i,j)\in\mathcal{E}\setminus\tilde{A}} \bar{f}_{i,j} - \left(\sum_{(j,i)\in\tilde{A}} \bar{f}_{j,i} + \sum_{(i,j)\in\mathcal{E}\setminus\tilde{A}} \bar{f}_{j,i} \right)$$

by definition of $\tilde{A}$

$$= \sum_{(i,j)\in\mathcal{E}} \bar{f}_{i,j} - \sum_{(j,i)\in\mathcal{E}} \bar{f}_{j,i}$$
$$= b_i$$

by definition of LP1.

Moreover, for each $(i,j) \in \tilde{A}$, $\hat{f}_{i,j} = \bar{f}_{i,j} \geq 0$ and also for each $s \in \mathcal{T}$, $\hat{f}_s = \frac{\bar{f}_{i,j}}{\rho_{i,j}} \geq 0$. Therefore, $\hat{f}$ is a feasible solution for the LP2. The amount of the objective function of LP2 corresponding to this feasible solution is:

$$\sum_{(i,j)\in\tilde{A}} \hat{f}_{i,j} + \sum_{s\in\mathcal{T}} \hat{f}_s = \sum_{(i,j)\in\tilde{A}} \hat{f}_{i,j} + \sum_{s\in\mathcal{T}} 1 \cdot \hat{f}_s$$

$$= \sum_{(i,j)\in\tilde{A}} \hat{f}_{i,j} + \sum_{s\in\mathcal{T}} \left(\sum_{(i,j)\in A_s} \rho_{i,j} \right) \cdot \hat{f}_s$$

by the fact that $\rho_{i,j} = \mu_{tr}(v')$ where $(i,j) = (v^{tr}, v')$ or $(i,j) = (v^{tr}_a, v'_a)$ and that $\sum_{(i,j)\in A_s} \rho_{i,j} = 1$

$$= \sum_{(i,j)\in\tilde{A}} \hat{f}_{i,j} + \sum_{s\in\mathcal{T}} \sum_{(i,j)\in A_s} \rho_{i,j} \cdot \hat{f}_s$$

by simple term manipulation

$$= \sum_{(i,j)\in\tilde{A}} \bar{f}_{i,j} + \sum_{s\in\mathcal{T}} \sum_{(i,j)\in A_s} \bar{f}_{i,j}$$

by definition of $\hat{f}_s$

$$= \sum_{(i,j)\in\tilde{A}} \bar{f}_{i,j} + \sum_{(i,j)\in\mathcal{E}\setminus\tilde{A}} \bar{f}_{i,j}$$

by definition of $\tilde{A}$

$$= \sum_{(i,j)\in\mathcal{E}} \bar{f}_{i,j}.$$

As a consequence, since every feasible solution for LP1 is a feasible solution for LP2 and vice versa, and the value of the objective functions is the same, we have that LP1 and LP2 are equivalent. $\square$

Since both LP1 and LP2 are equivalent to the $LP(t, a, \mu, \mathcal{R})$ LP problem, we exploit the structure of LP2 to improve the efficiency of checking for a solution of $LP(t, a, \mu, \mathcal{R})$. Simultaneously, we also improve the complexity of deciding weak bisimulation. Amongst all available versions of polynomial algorithms for solving a linear programming problem, we resort to a state-of-the-art polynomial interior point method [Ans99] which, to the best of our knowledge, is equipped with the tightest known worst case complexity.

Theorem 2. Consider a rational *PA* $\mathcal{A}$, the action a, the probability measure $\mu \in \mathrm{Disc}(S)$, the equivalence relation $\mathcal{R}$ on S and a state $t \in S$. Let $N = \mathit{size}(\mathcal{A})$. Then, checking the feasibility of the $LP(t, a, \mu, \mathcal{R})$ LP problem can be done in $\mathcal{O}(\frac{N^3}{\ln N} \cdot L)$ where L is the bit size of the problem.

Proof. By Lemmas 1 and 2, $LP(t, a, \mu, \mathcal{R})$ is feasible if and only if LP2 is feasible. Now, consider the dual of LP2; by assigning the dual variables π_s for each $s \in \mathcal{V}$, hence $\mathcal{O}(N)$ variables, we get the following dual LP problem:

$$\begin{array}{rll} \text{DLP2: max} & \sum_{s \in \mathcal{V}} b_s \cdot \pi_s & \\ \text{s.t.} & \pi_i - \pi_j \leq 1 & \text{for each } (i, j) \in \tilde{A} \\ & \sum_{t \in \mathcal{V}} a_s^t \cdot \pi_t \leq 1 & \text{for each } s \in \mathcal{T}. \end{array}$$

By using a state-of-the-art preconditioned conjugate gradient (PCG) method with a partial updating procedure [Ans99], this LP problem can be solved optimally in $\mathcal{O}(\frac{N^3}{\ln N} \cdot L)$ where L is the bit size of the problem. At termination of the algorithm, we have two possible cases:

1. The dual LP problem has a finite optimal objective value: by the strong duality theorem [BT97], the original LP2 is feasible and also has a finite optimal objective value.
2. The dual LP problem is unbounded: by the strong duality theorem the original LP2 is infeasible.

Thus, by solving the dual LP problem efficiently, we can verify the existence of a weak combined transition for the given *PA*. $\square$

Notably, if we were to use the interior point method directly on the original LP problem instead of LP2, we would face an extra factor N in the complexity bound. This is because the running time of the method depends on the number of variables: The number of variables occuring in $LP(t, a, \mu, \mathcal{R})$ is $\mathcal{O}(N^2)$ while the number of variables in LP2 is $\mathcal{O}(N)$. This reduction directly translates into a reduced worst case complexity, and this especially appreciable if working with large probabilistic automata.

Corollary 1. Given two *PA*s $\mathcal{A}_1$ and $\mathcal{A}_2$, let $N = \mathit{size}(\mathcal{A}_1) + \mathit{size}(\mathcal{A}_2)$. Checking $\mathcal{A}_1 \approx \mathcal{A}_2$ can be done in time $\mathcal{O}(\frac{N^6}{\ln N} \cdot L)$ where L is the maximum bit size of the LP problems solved in FindSplit and Refine.

Proof. Immediate by Proposition 1 and Theorem 2. $\square$

Since the worst case runtime bound essentially depends on the type of the polynomial algorithm used to solve the LP problem, any advancement in LP problem solution complexity directly improves the complexity of the weak bisimulation decision problem.

Remark 5. If considering the structure of the $LP(t, a, \mu, \mathcal{R})$ LP problem, one might observe that it is in essence a system of linear equations with non-negativity constraints. So, we may consider instead to use elimination techniques (inspired by Gaussian elimination) to reduce the number of variables and constraints we have in the LP problem:

1. take one of the linear equations, say $f_{v,v^{tr}} - \sum_{(v^{tr},u) \in E} f_{v^{tr},u} = 0$ and one variable occurring in it, say $f_{v,v^{tr}}$;
2. express the variable as linear combination of the other variables, i.e., $f_{v,v^{tr}} = \sum_{(v^{tr},u) \in E} f_{v^{tr},u}$;
3. replace each occurrence of the variable with such combination, i.e., $f_{v,v^{tr}}$ by $\sum_{(v^{tr},u) \in E} f_{v^{tr},u}$.

If we iterate this process until no more variables can be isolated at step 2, we obtain another LP problem that is equivalent to the original one.

Now, since we are not interested in the actual value of the variables, but only on whether the problem is feasible, we can eliminate the equations we considered at step 1 and the corresponding variables at step 2. This results in an LP problem no more equivalent to the original one, but it is easy to show that the latter is feasible if and only if the original problem is.

As an example, consider the following LP problem:

$$\begin{array}{llll}
f_0 - f_1 - f_2 - f_3 = 0 & f_0 = 1 & f_0 \geq 0 & f_1 \geq 0 \\
f_1 + f_2 - f_4 = 0 & f_4 = 0.5 & f_4 \geq 0 & f_2 \geq 0 \\
f_3 - f_5 = 0 & f_5 = 0.5 & f_5 \geq 0 & f_3 \geq 0
\end{array}$$

In a single time, if we replace f_0, f_4, and f_5 with their respective values, we obtain:

$$\begin{array}{llll}
1 - f_1 - f_2 - f_3 = 0 & f_0 = 1 & 1 \geq 0 & f_1 \geq 0 \\
f_1 + f_2 - 0.5 = 0 & f_4 = 0.5 & 0.5 \geq 0 & f_2 \geq 0 \\
f_3 - 0.5 = 0 & f_5 = 0.5 & 0.5 \geq 0 & f_3 \geq 0
\end{array}$$

Now, by replacing f_3 with 0.5, the system becomes:

$$\begin{array}{llll}
1 - f_1 - f_2 - 0.5 = 0 & f_0 = 1 & 1 \geq 0 & f_1 \geq 0 \\
f_1 + f_2 - 0.5 = 0 & f_4 = 0.5 & 0.5 \geq 0 & f_2 \geq 0 \\
f_3 - 0.5 = 0 & f_5 = 0.5 & 0.5 \geq 0 & 0.5 \geq 0
\end{array}$$

and by substituting f_1 with $0.5 - f_2$:

$$\begin{array}{llll}
1 - 0.5 + f_2 - f_2 - 0.5 = 0 & f_0 = 1 & 1 \geq 0 & 0.5 - f_2 \geq 0 \\
f_1 = 0.5 - f_2 & f_4 = 0.5 & 0.5 \geq 0 & f_2 \geq 0 \\
f_3 = 0.5 & f_5 = 0.5 & 0.5 \geq 0 & 0.5 \geq 0
\end{array}$$

that is,

$$\begin{array}{llll}
0 = 0 & f_0 = 1 & 1 \geq 0 & 0.5 - f_2 \geq 0 \\
f_1 = 0.5 - f_2 & f_4 = 0.5 & 0.5 \geq 0 & f_2 \geq 0 \\
f_3 = 0.5 & f_5 = 0.5 & 0.5 \geq 0 & 0.5 \geq 0
\end{array}$$

This system is feasible and it has a solution for each $0 \leq f_2 \leq 0.5$.

This approach looks promising, but in fact is much more expensive than the result achieved by Theorem 2: if we ignore the bit size of the problem, for an $n \times n$ matrix, the Gaussian elimination has complexity $\mathcal{O}(n^3)$ where n is the number of variables in the system of equations (corresponding to the number of columns of the matrix). In our setting, we have an $m \times n$ matrix with $m > n$, thus the actual complexity is larger than $\mathcal{O}(n^3)$. If we now express the complexity of the Gaussian elimination approach in terms of $N = \mathit{size}(\mathcal{A})$, since we have $\mathcal{O}(N^2)$ variables, the resulting complexity is at least $\mathcal{O}(N^6)$, without considering the complexity of solving the remaining LP problems.

Non-rational automata The class of rational probabilistic automata, as far as the authors know, encompasses all *PA*s that have appeared in practical applications. One may nevertheless consider relevant also the analysis of *PA*s with real valued probabilities.

One possible way to represent LP problems with real data is to use a model of computation that can perform any elementary arithmetic operation in constant time, regardless of the type of the operand. Another option is to encode reals as finite precision rationals. For a survey on the theory of computation over real numbers we refer the reader to [BSS89, Bel01].

When using finite precision rationals, the representation of the *PA* must become approximate, and still the size needed for this can no longer be guaranteed to be bounded by a polynomial. If assuming the rational approximation scheme being employed by the user, we are back to the rational setting for the LP problem solution process, and it is left to the user to interpret the outcome on the real valued *PA*. If instead the algorithm performs the approximation prior to solving the induced LP problem, the user may in general lack knowledge on how to transfer the result back to the original real valued *PA*.

6.2. Efficient Solution: Exploiting Structure

We now consider the practical efficiency of deciding probabilistic automata weak bisimulation. We first discuss available algorithms that can be employed. We show that the underlying structure of the problem enables us to check feasibility of the LP problem more efficiently than by just resorting to a general purpose LP solver implicitely finding the optimal solution. Afterwards we discuss other methods that are known to more efficient in general but turn out to be unsuitable for solving the $LP(t, a, \mu, \mathcal{R})$ LP problem.

Working with a linear programming problem allows practitioners to use the omnipresent simplex method as an extremely efficient computational tool. It is worthwhile to note that the efficiency of the simplex method is measured as the number of pivots needed to solve the LP problem. Moreover, practical experiments show that although this method is highly efficient, there exist problems that require an exponential number of pivots. This means that the worst case theoretical complexity of the simplex method is exponential time [KM72]. However, computational experience on thousands of real-world problems reveals that the number of pivots is usually polynomial in the number of variables and of constraints.For a comprehensive survey on the efficiency of the simplex method, we refer the interested reader to [Sha87].

Since the LP1 problem is a minimum cost flow problem on the network $\mathcal{G}(t, a, \mu, \mathcal{R})$ extended with an additional set of proportional flow constraints, we consider the usage of efficient algorithms that solve the problem directly on the flow network itself. One such algorithm is the network simplex algorithm [BF12] for the minimum cost proportional flow problem that improves the per iteration running time considerably with respect to the simplex method, as long as the number of nodes in the network is at least an order of magnitude larger than the number of side constraints in the LP problem [Cal02, MSJ11, BF12, MSJ13]. So, the network simplex algorithm is a candidate for improving the running time required to solve LP1. However, the number of side constraints coincides with the number of transition nodes in the LP1 problem. Since the number of transitions in the automaton is usually larger than the number of states, we have that the number of side constraints is linear in the number of nodes, and thus the above assumption is not satisfied. Still, a more accurate analysis tells us that, in our setting, the resulting per iteration running time of both methods is in the same complexity class, as shown in Table 1. Since it is known that the network simplex algorithm without side constraints performs better than the simplex method [AMO93], it is still worthwhile to consider its usage in an implementation.

Up to now, we have discussed that the simplex method and the network simplex algorithm [BF12] appear quite competitive in solving the LP1, and that the flow network structure underlying LP1 motivates the use of the network simplex algorithm. On the other hand, we can take the dual of the equivalent LP2. This allows us to deal with a smaller sized LP problem which is still close to a well known combinatorial problem by itself. To clarify the point, consider the dual DLP2 of the LP2 problem:

$$\begin{aligned}
\text{DLP2: max } & \sum_{s \in \mathcal{V}} b_s \cdot \pi_s \\
\text{s.t. } & \pi_i - \pi_j \leq 1 \quad \text{for each } (i,j) \in \tilde{A} && (1) \\
& \sum_{t \in \mathcal{V}} a_s^t \cdot \pi_t \leq 1 \quad \text{for each } s \in \mathcal{T} && (2)
\end{aligned}$$

The number of constraints in DLP2 is $\mathcal{O}(N)$, just as for LP1. The number of variables in DLP2 is $\mathcal{O}(N)$ which compares favorably with $\mathcal{O}(N^2)$, the number of variables in the original LP1. This observation is particularly important whenever the number of transitions is considerably larger than the number of states in the network. The dual LP problem can again be solved very efficiently using a state-of-the-art variant of the interior point method [Ans99]. This algorithm is a preconditioned conjugate gradient (PCG) method with a partial updating procedure which works excellent in practice as well. The algorithm is available in the software tools like CPLEX and LOQO. Furthermore, DLP2 has itself a combinatorial structure, i.e., it is the dual of the well known shortest path problem although with additional side constraints. Taking the advantage of this combinatorial property may help in the design of a more efficient algorithm to solve the problem.

Table 1 summarizes the size of the proposed LP problems and the per-iteration complexity of the simplex method and of the network simplex algorithm. Since each variable in the LP problem corresponds to an arc in the network, we identify by n both variables and arcs; on networks, each arc either belongs to a proportional flow set or is a free arc. The computational comparison of three LP problems is described based on N which is the size of the automaton $\mathcal{A}$. It is immediate to see that LP2 and DLP2 are the smallest problems that are at least one degree smaller than the other LP problems, making them more suitable as input for the LP solvers.

Table 1. Complexity comparison

		$LP(t,a,\mu,\mathcal{R})$	LP1	LP2	DLP2
Variables/Arcs	n	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$	$\mathcal{O}(N)$	$\mathcal{O}(N)$
Constraints	m	$\mathcal{O}(N^2)$	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(N)$
Proportional Flow Sets	p	not applicable	$\mathcal{O}(N)$	not applicable	not applicable
Free Arcs	n'	$\mathcal{O}(N^2)$	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(N)$
Simplex Method	$\mathcal{O}(nm)$	$\mathcal{O}(N^4)$	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$
Network Simplex Algorithm [MSJ13]	$\mathcal{O}(n' + mp + p^3)$	not applicable	$\mathcal{O}(N^3)$	not applicable	not applicable

6.3. Efficient Solution: Unsuitable Approaches

As we have seen, the $LP(t,a,\mu,\mathcal{R})$ LP problem can be solved efficiently using the simplex method or the network simplex algorithm.

Several other solutions have been proposed in the literature to solve variations of LP problems more efficiently: among others, there are approximated algorithms [Vaz04], electrical flow representation [CKMS11], network decomposition [Pul89], and Lagrangian relaxation [BT97]. As we will see in the remainder of this section, all these approaches are not suitable for solving the $LP(t,a,\mu,\mathcal{R})$ LP problem and its equivalent reformulations, but for different reasons: either because the corresponding model does not enable an encoding of the $LP(t,a,\mu,\mathcal{R})$ LP problem at hand, or the answer provided by the algorithm can not always be mapped into an answer to our problem, or the algorithm is prohibitively expensive in case of a positive answer.

Despite being unsuitable, we review our findings concerning these methods for two main motivations: the first is to clearly identify the specific characteristics of the problem faced, the second motivation is to propose a new challenging problem to both the optimization and the probabilistic automata world.

6.3.1. Approximation Algorithms

As a first approach, we consider the use of an approximation algorithm to check the feasibility of the original LP problem by dropping the proportional flow constraints and then solving the remaining linear programming problem efficiently. This exploits that the remaining LP problem is actually a minimum cost flow problem on the network $\mathcal{G}(t,a,\mu,\mathcal{R})$ and the general network simplex algorithm can solve it extremely efficiently. If the relaxed problem is infeasible, so it is the original LP problem; otherwise, we get a feasible solution that may not be feasible for the original one. In this case, we assign fixed weights to each proportional flow constraint and increase the weight for the violated side constraints as a penalization. This procedure, known as the multiplicative weight update method [AHK12], is repeated until a feasible solution which is near optimal is found. The advantage of this approach is that in each iteration we deal with a well structured LP problem that can be solved efficiently.

The main problem of this approach is that in general a positive result does not imply the existence of the corresponding weak transition. Consider, for example, the automaton whose only transitions are $s \xrightarrow{a} \mu$ and $t \xrightarrow{a} \delta_v$ where $\mu = \{(v,p),(u,1-p)\}$ for some $p \in [0,1]$; suppose that $u \not\mathcal{R}\, v$. It is easy to verify that $\mu \not\mathcal{L}(\mathcal{R})\, \delta_v$ for each $p \neq 1$, so there does not exist any weak transition $t \xRightarrow{a}_{\mathrm{c}} \rho$ such that $\mu \,\mathcal{L}(\mathcal{R})\, \rho$ since the only possible weak transition enabled by t labelled by a is $t \xRightarrow{a}_{\mathrm{c}} \delta_v$. However the approximation algorithm gives us a positive answer whenever p is close enough to 1, so a positive answer does not ensure the existence of the weak transition, unless we force the gap between optimal and near optimal objective values to be 0. But this may make the overall algorithm very expensive. However, for practical purposes, approximated algorithms can be used to refute the existence of a weak transition.

6.3.2. Electrical Flows

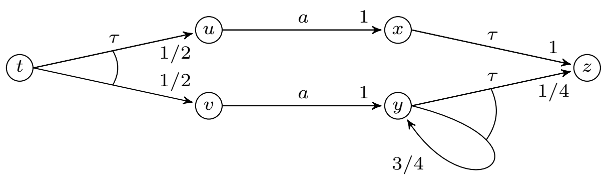
As a second approach we consider a physical metaphor for graphs by transforming the network $\mathcal{G}(t,a,\mu,\mathcal{R})$ as an electrical network. This comes with replacing the arcs of the network by resistors. Our goal is to arrive at a setting where we can use the state-of-the-art max flow algorithm [CKMS11] as an approximation algorithm to solve the original LP problem. The weakness of this approach is twofold: the approximate nature of the procedure has the same drawback as the previous approach, and furthermore the applicability of the transformation. Even if an efficient non-approximate algorithm was at hand, the transformation can not be applied, since it is restricted to undirected networks [CKMS11], while $G(t, a, \mu, \mathcal{R})$ is directed. Extending the results in [CKMS11] to directed networks is an open problem.

To make the network directed, we may represent each arc by a resistor and a diode that is a two-terminal component allowing the current to flow in a single direction. Even by using diodes to direct the network, we still need to solve two problems: cycles and nondeterminism. In an electrical network it is not possible to have the current going through a passive cycle since the overall potential difference in the cycle is zero, unless we use some fictitious voltage generator that breaks the cycle, while in probabilistic automata it is common to have internal cycles: consider for instance the transition $s \xrightarrow{\tau} \mu$ where $\mu = \{(t, 0.3), (s, 0.7)\}$; by using the determinate scheduler σ that stops in t and performs $s \xrightarrow{\tau} \mu$ in s, we obtain the weak transition $s \overset{\tau}{\Longrightarrow}_{\mathrm{c}} \delta_t$, that is, we eventually leave the self-loop with probability 1. In order to obtain a similar result in an electrical network, we have to add a fictitious voltage generator in the cycle corresponding to the self-loop that generates the correct potential difference. Finding such difference is essentially equivalent to defining the scheduler. The second problem is related to nondeterminism: suppose that we have two transitions $s \xrightarrow{\tau} \mu$ and $s \xrightarrow{\tau} \rho$ where $\mu = \{(u, 0.3), (v, 0.7)\}$ and $\rho = \{(u, 0.7), (v, 0.3)\}$, so we can reach v with different probabilities by using two different transitions. When we encode these two transitions in the electrical network, we obtain two parallel paths from s to v that are subject to the same voltage difference V_{sv}, so the current flows in both paths according to Ohm's law. However the scheduler can choose to perform only $s \xrightarrow{\tau} \mu$ thus in the network we should have a non null current from s to v in the path modelling such transition and a null current in the path corresponding to $s \xrightarrow{\tau} \rho$, that is, the former requires $V_{sv} > 0$ while the latter requires $V_{sv} = 0$ and this is clearly impossible.

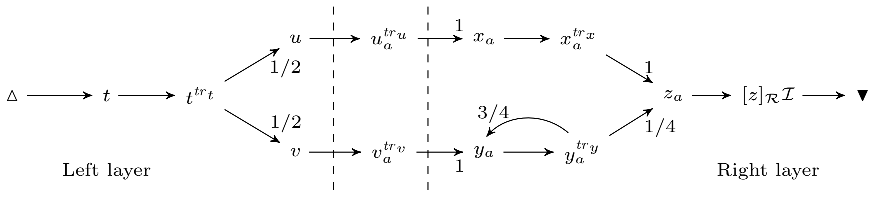
6.3.3. Network Decomposition

As a third approach, we consider a natural decomposition of the state space of the underlying network. We aim at designing a parallel algorithm that speeds up the check for feasibility of the LP problem when $a \in E$. The underlying network can be seen as a network of three layers (here considered in horizontal layout): the left hand side and the right hand side layers correspond to the internal transitions (sets L_1 and L_2, respectively) while the central layer to the external transitions (set L_a). Moreover it is possible to change layer only from left to right. Using this intuition, each layer can be treated independently so that the network simplex algorithm instantiations can find the minimum cost flow in the left and the right layers in parallel. Then, to connect these two layers via the central one is enough to solve a linear system of equations corresponding to the central arcs. This system can be solved in linear time.

However, this approach is not suitable as a negative answer does not imply the non-existence of the weak transition: consider the following *PA*.



It is immediate to see that $t \overset{a}{\Longrightarrow}_{\mathrm{c}} \delta_z$; now, consider the identity relation over states $\mathcal{I}$ and the part of the corresponding layered network between source and sink, where numbers attached to arcs indicate probabilities, and are not part of the graph.



The solution of the left layer is unique and it assigns outgoing flow 1/2 to both vertices u and v, while the optimal solution for the right layer assigns flow 1 to arcs $(x_a, x_a^{tr_x})$, $(x_a^{tr_x}, t_a)$ and flow 0 to arcs $(y_a, y_a^{tr_y})$, $(y_a^{tr_y}, t_a)$, and $(y_a^{tr_y}, y_a)$. By using these two optimal solutions, there is no way to obtain a solution for the central layer since there is only one path from u to x_a and flow requirements are different. However there exists a solution for the network as a whole that requires for the right layer the non-optimal feasible solution $f_{x_a, x_a^{tr_x}} = 1/2$, $f_{x_a^{tr_x}, z_a} = 1/2$, $f_{y_a, y_a^{tr_y}} = 4/2$, $f_{y_a^{tr_y}, z_a} = 1/2$, and $f_{y_a^{tr_y}, y_a} = 3/2$, thus a negative answer from the layered network decomposition does not imply that there does not exist a feasible solution for the whole network. The core reason is that the optimal solution of one layer may be not part of a feasible solution of the whole network: a feasible solution may only be induced by a sub-optimal layer solution.

6.3.4. Lagrangian Relaxation

As a last approach, we consider a Lagrangian relaxation [BT97] algorithm to solve the dual LP problem efficiently. Consider again the DLP2 problem; in order to form a Lagrangian relaxation of DLP2, we multiply the set of constraints (2) by non-negative Lagrangian multipliers λ_s, $s \in \mathcal{T}$, and we add them to the objective function, obtaining the following relaxed dual LP (RDLP) problem:

$$\begin{aligned} L(\lambda)\colon \max\ & \sum_{t \in \mathcal{V}} (b_t + \sum_{s \in \mathcal{T}} \lambda_s \cdot a_s^t) \cdot \pi_t - \sum_{s \in \mathcal{T}} \lambda_s \\ \text{s.t. } & \pi_i - \pi_j \leq 1 \qquad\qquad \text{for each } (i,j) \in \tilde{A} \end{aligned}$$

For a fixed vector λ, this LP problem can be efficiently solved using the simple and fast algorithm described in [HN94].

Consider the Lagrangian dual LP (LDLP) problem:

$$\begin{aligned} \text{LDLP: } \min\ & L(\lambda) \\ \text{s.t. } & \lambda \geq 0 \end{aligned}$$

The purpose of the LDLP problem is to find the tightest bound $L(\lambda)$ for the possible values of λ. Since RLDP is always feasible with solution $\pi_t = 0$ for each $t \in \mathcal{V}$, we can make no claims on the feasibility of the original DLP2 problem unless (1) actually finding a feasible solution for the original DLP2 problem, or (2) proving that the optimum solution for LDLP is bounded. This is the main point that induces the weakness of this approach as an efficient verification procedure.

7. Implementation of Minimization

The results presented thus far provide ample understanding for an implementation of a quotienting algorithm. This section presents and discusses such an implementation which is tailored to the computation of the minimal automaton that is weak probabilistic bisimilar to a given one [EHS+13]. In fact, some intricate problems remained to be overcome to make the approach effective and scalable. These problems are rooted in numerical aspects of the computations at hand as well as in the often excessive number of feasibility checks needed. Both these aspects are genuine to the setting considered and neither occur in the context of minimizing labelled transition systems, nor in other stochastic minimization contexts.

We here report on our second generation prototype minimizer, implemented in Java. It has a modular structure and it can delegate the feasibility checks either to an LP solver, or to an SMT solver. We use LpSolve [LpS] as LP solver, the GLP Kit [GLP] as exact arithmetic LP solver, and Z3 [dMB08] as SMT solver. We encode our SMT formulation according to the SMT-LIB format [BST10] allowing the use of other solvers. We can use an SMT solver instead of an LP solver since Proposition 2 relates the existence of the desired weak transition $t \overset{a}{\Longrightarrow} \mu_t$ such that $\mu\ \mathcal{L}(\mathcal{R})\ \mu_t$ with the feasibility of the $LP(t, a, \mu, \mathcal{R})$ problem, so we are not interested in the optimal solution, but just in a solution.

We perform the feasibility check directly on the original $LP(t, a, \mu, \mathcal{R})$ problem. This allows us to maintain an undisguised view on the structure of the problem, which we considered important to ensure the correctness of the prototype implementation, and also to assess the relative share of the different algorithmic steps to the overall runtime and space requirements. We plan to implement the module for the smaller LP2 problem to achieve better running times while preserving correctness.

7.1. Implementation Details

In our prototype we have implemented several heuristics in order to minimize the number of solver calls needed to compute the coarsest weak bisimilarity. We can classify these heuristics in two classes: the *pre-bisimulation reductions* and the *in-loop optimizations.* Finally, we consider the extraction of exact solutions from inexact solutions to improve the in-loop optimizations and the parallelization of the solver calls.

7.1.1. Pre-Bisimulation Reductions

We reduce the automaton before computing the weak probabilistic bisimulation by removing irrelevant transitions and collapsing states that are trivially bisimilar. In particular, we remove internal self loops (i.e., transitions as $s \xrightarrow{\tau} \delta_s$), we merge all deadlock states in a single one and each pair of states (s, t) in t such that the only transition enabled by s is $s \xrightarrow{\tau} \delta_t$. Moreover, we merge states s and t if the transitions they enable reach the same distributions via the same labels. These reductions are sound since it is easy to prove that all these merged states are weak bisimilar.

It is also possible to apply a preliminary bisimulation reduction based on strong (probabilistic) bisimulation [Seg06, Seg95] that would collapse some more state; note however that such reduction does not cover all above reductions; for instance, it does not remove self-loops as well as usually it does not collapse transitions like $s \xrightarrow{\tau} \delta_t$.

7.1.2. In-Loop Optimizations

We adopt several optimizations in order to reduce the number of weak combined transitions computed by calling the solver. In particular, we implement the *internal* optimization that allows us to skip the LP problem construction and solution whenever the challenging transition at line 2 of the FindSplit procedure is of the form $s \xrightarrow{\tau} \mu_s$ with $\mu_s([s]_{\mathcal{R}}) = 1$. Such transition is trivially matched by any $t \in [s]_{\mathcal{R}}$ by performing no transitions at all. Similarly, by using the *direct transition* optimization we save a solver call if the state t directly enables a transition $t \xrightarrow{a} \mu_t$ matching $s \xrightarrow{a} \mu_s$, i.e., we check whether there exists $(t, a, \mu_t) \in T$ such that $\mu_s \ \mathcal{L}(\mathcal{R}) \ \mu_t$. Finally, we maintain a *transition cache* containing for each state t, the list of computed transitions $t \overset{a}{\Longrightarrow}_{\mathrm{c}} \mu_t$ where the distribution μ_t has been generated by the solver on the $LP(t, a, \mu_s, \mathcal{R})$ problem for some challenging transition $s \xrightarrow{a} \mu_s$. This cache allows us to save a solver invocation whenever the cache contains a matching transition. Suppose that we have already used the LP or SMT solver in order to find a matching transition $t \overset{a}{\Longrightarrow}_{\mathrm{c}} \mu'_t$ for $s \xrightarrow{a} \mu'_s$ such that $\mu'_s \ \mathcal{L}(\mathcal{R}') \ \mu'_t$ for some partitioning $\mathcal{R}'$. As long as $\mu_s \ \mathcal{L}(\mathcal{R}) \ \mu'_t$ holds for the current partitioning $\mathcal{R}$, there is no need to call again the solver since it is going to give a positive answer, so we can save such call. In order to be effective, we need to keep the cache updated and this can be achieved easily since the only operation we need is to add entries to the cache, provided that we store the computed $t \overset{a}{\Longrightarrow}_{\mathrm{c}} \mu_t$ transitions.

7.1.3. Exact solutions from inexact solutions

The optimizations presented previously need to compare distributions, either when checking $\mu_s \ \mathcal{L}(\mathcal{R}) \ \mu_t$ or while searching for a cached weak combined transition. In order to be effective, rational numbers cannot be represented using floating-point numbers since small rounding errors may render floating-point comparisons to become incorrect. For the direct transition optimization we overcome the problem by using exact representations for probabilities, such as infinite precision integers. For the caching, we need to retrieve the optimal feasible solution from an LP or SMT solver. This is only possible if we use an SMT solver or an LP solver equipped with exact arithmetic, since floating-point based LP solvers only provide inexact solutions for the system of inequalities, making the cache rather useless. We solve this problem via the *inexact to exact* optimization, i.e., by finding the *exact* solution as long as the underlying rational number does not have a large denominator. Any number $p \in \mathbb{R}$ can uniquely be represented as a *simple continuous fraction* of the form:

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots}}$$

Therefore any number can be represented canonically by the sequence $a_0, a_1, \ldots$; note that such sequence is finite if and only if $p \in \mathbb{Q}$. The canonical representation can be obtained by using the following inductive definitions [JT80]:

$$a_i = \lfloor p_i \rfloor \qquad p_0 = p \qquad p_{i+1} = (p_i - \lfloor p_i \rfloor)^{-1}$$

Any number can be approximated by c_n, the continuous fraction obtained from the finite sequence $a_0, \dots, a_n$. In fact, c_n is the *best rational approximation*: any other rational number that is closer to p has a denominator larger than c_n. Moreover, c_{n+1} has a denominator larger than c_n, thus the sequence $\{c_i\}$ converges absolutely to p. We calculate the sequence of approximations $c_0, c_1, \dots$ until we find a value c_n such that $|p - c_n| < \varepsilon$ for a predefined $\varepsilon > 0$.

7.1.4. Parallel solvers

In FindSplit, for each state s, for the current partition $\mathcal{R}$ we have to check whether all its enabled transitions have a matching weak transition in all the states in $[s]_{\mathcal{R}}$. During such a loop the partition $\mathcal{R}$ does not change, therefore we can generate all the LP problems in advance, so as to solve them in parallel. We implemented a simple thread pool where each single task checks all the matching transitions from a given $t \in [s]_{\mathcal{R}}$, i.e., the first task manages t_1, the second task t_2, and so on.

7.2. Case Studies

We evaluated our prototypical implementation by applying it to several cases studies taken from the literature. Experiments were run on four AMD® Opteron® 8350 (Quad-core) 2GHz with 120GB of RAM. We only used 14 out of 16 cores with the memory usage restricted to 8GB. Time-outs correspond to experiments that took more than 6 hours to complete. The models we considered are IEEE 802.3 CSMA/CD protocol, dining cryptographers, IEEE 1394 FireWire root contention protocol, IEEE 802.11 Wireless LAN, and IPv4 Zeroconf protocol that we have taken from the PRISM benchmark suite [PRI] and only minor changes have been made to manage the shared variables for synchronization. More information on the case studies and the choice of parameters is directly available from the benchmark suite [PRI]. We hide the action `time` in the variant with suffix "-nt" and we unify similar actions in the "-sa" variant; for example, we rename `send1` and `send2` to `send`.

7.2.1. Additional Reductions

Given a *PA* $\mathcal{A}$, we denote by $\mathcal{A}_{\bowtie}$ the automaton resulting from the repeated application of the pre-bisimulation reductions until such reductions do not change the automaton anymore. The effectiveness of these reductions is shown in Table 2, where we report for several case studies the state and transitions space size of the original automaton $\mathcal{A}$, of the corresponding $\mathcal{A}_{\bowtie}$, and of the minimal automaton $[\mathcal{A}]_{\approx}$. The columns $t_{\mathcal{A}_{\bowtie}}$ and $t_{[\mathcal{A}]_{\approx}}$ report the time needed to reduce $\mathcal{A}$ to $\mathcal{A}_{\bowtie}$ and to generate $[\mathcal{A}_{\bowtie}]_{\approx}$ from $\mathcal{A}_{\bowtie}$ and the given $\approx$, including the time for the post-quotient reductions of the automaton.

More precisely, we consider in $[\mathcal{A}_{\bowtie}]_{\approx}$ also the time needed for removing redundant transitions (that requires further checks for the existence of the weak combined transitions) and for rescaling distributions (cf. Section 4.2). In particular, the former reduction requires to remove one transition at a time and check whether there exists a weak combined transition using only the remaining transitions that reaches the same distribution. The difference of time of the Wireless LAN case with respect to the other cases depends on the number of internal transitions still present in the quotient as well as the number of transitions leaving the state: if a state enables only one transition or only transitions with different external actions, then there is no need to try to remove such a transition, since we will obtain a negative answer for sure.

7.2.2. Quotient *Performance*

Tables 3, 4, and 5 show the effects of the different optimizations and the running time of the implementation of the Quotient procedure, where the solver used for checking $LP(t, a, \mu_s, \mathcal{R})$ is SMT, LP, and GLP, respectively.

After the columns with the problem and the number of transitions of $\mathcal{A}_{\bowtie}$, the column Δ shows the number of challenging transitions verified via the internal optimization. Note that this condition trivially holds for each internal transition in the first round of the outer cycle since the initial partition contains only

Table 2. Minimization overview

Problem	$\lvert S\rvert$	$\lvert T\rvert$	$\lvert S_{\bowtie}\rvert$	$\lvert T_{\bowtie}\rvert$	$t_{\mathcal{A}_{\bowtie}}$	$\lvert[S]_{\approx}\rvert$	$\lvert[T]_{\approx}\rvert$	$t_{[\mathcal{A}_{\bowtie}]_{\approx}}$
csma2	1038	1054	835	849	1s	449	459	1s
csma2-sa	1038	1054	621	630	7s	233	237	< 1s
csma2-sa-nt	1038	1054	91	98	< 1s	87	90	< 1s
dining4	2165	4540	161	300	< 1s	1	1	< 1s
firewire3	611	694	425	469	5s	425	469	5s
firewire3-nt	611	694	29	62	< 1s	4	4	< 1s
wlan_dl0dl6	97	148	63	94	< 1s	59	86	1s
wlan0col0	2954	3972	1097	1591	14s	798	1092	120s
zeroconf	670	827	341	433	< 1s	334	420	14s
zeroconf-nt	670	827	52	75	< 1s	41	52	< 1s

Table 3. Caching overview for SMT

Problem	$\lvert T_{\bowtie}\rvert$	Δ	DT	CH	TC	$t_{\approx}$	t_{Solver}	$\lvert\approx\rvert$
csma2	849	24297	82337	150	11700	288s	243s	449
csma2-sa	630	9111	66162	0	6067	97s	83s	233
csma2-sa-nt	98	1739	186	14	1118	12s	6s	87
dining4	300	45440	240	2175	145	6s	5s	1
firewire3	469	30842	34070	257	1311	44s	36s	425
firewire3-nt	62	664	833	48	166	2s	1s	4
wlan_dl0dl6	94	107	277	46	405	4s	1s	59
wlan0col0	1591	48284	102296	14902	30204	1h3m	1h1m	798
zeroconf	433	2063	30829	453	2065	59s	47s	334
zeroconf-nt	75	361	265	99	348	5s	2s	41

Table 4. Caching overview for LP with inexact to exact optimization

Problem	$\lvert T_{\bowtie}\rvert$	Δ	DT	CH	TC	$t_{\approx}$	t_{Solver}	$\lvert\approx\rvert$
csma2	849	24289	73229	150	11650	560s	556s	449
csma2-sa	630	9111	67657	0	6078	121s	118s	233
csma2-sa-nt	98	1739	186	14	1118	6s	6s	87
dining4	300	45440	240	2175	145	4s	3s	1
firewire3	469	30842	33878	257	1311	68s	66s	425
firewire3-nt	62	664	833	48	166	1s	1s	4
wlan_dl0dl6	94	107	275	45	410	1s	< 1s	59
wlan0col0	1591	53768	109604	16584	30362	1h17m	1h17m	798
zeroconf	433	2258	35098	515	2063	71s	69s	334
zeroconf-nt	75	379	281	105	333	2s	2s	41

Table 5. Caching overview for GLP (exact solver)

Problem	$\lvert T_{\bowtie}\rvert$	Δ	DT	CH	TC	$t_{\approx}$	t_{Solver}	$\lvert\approx\rvert$
csma2	849	24297	55499	150	12139	1h2m	1h2m	449
csma2-sa	630	9111	66969	0	6136	555s	544s	233
csma2-sa-nt	98	1739	186	14	1118	18s	17s	87
dining4	300	45440	240	2175	145	7s	6s	1
firewire3	469	30842	34064	257	1311	404s	399s	425
firewire3-nt	62	664	833	48	166	2s	2s	4
wlan_dl0dl6	94	107	275	45	410	2s	2s	59
wlan0col0	1591			—time-out—				
zeroconf	433	2009	29781	421	2069	207s	200s	334
zeroconf-nt	75	362	269	99	348	5s	4s	41

$S_{\bowtie}$ as class, so every internal transition reaches such class with probability 1. The column DT shows the number of times the defender t has been able to use the direct transition optimization, i.e., by a transition $t \xrightarrow{a} \mu_t$, to match a challenging transition $s \xrightarrow{a} \mu_s$. Column CH reports the number of cache hits, that is, the challenging transitions $s \xrightarrow{a} \mu_s$ that have been matched by a transition stored in the transition cache. The column TC contains the number of challenging transitions for which we have solved the $LP(t, a, \mu_s, \mathcal{R})$ problem.

The following two columns show the time $t_{\approx}$ spent computing the weak bisimilarity $\approx$ including the time t_{Solver} spent by the solvers for verifying all transitions counted in column TC. Since we use a pool of solvers running in parallel, t_{Solver} is the time spent by the slowest solvers in the pool, i.e., the time elapsing from the activation of the first solver to the completion of all solvers in the pool. Finally, the last column $|\approx|$ gives the size of the partition, i.e., the number of classes of bisimilar states. This value, decreased by 1, is also the number of refinements we perform in order to terminate the **while** loop of QUOTIENT.

By comparing the running times for the SMT, LP, and GLP solvers, we can see that GLP is always the slowest one while SMT is the best performing among them. This can possibly be explained by the highly optimized code of Z3 and the remarkable results achieved by the SAT community on satisfaction modulo theory problems. The use of SMT, however, introduces an overhead in the computation, as highlighted by the comparision between the colums $t_{\approx}$ and t_{Solver} of Table 3. Such overhead is mainly caused by the need of translating the LP problem construction into the textual SMT-LIB format [BST10] and then converting the solution (when the problem is satisfiable) back to numeric values. This induces a considerable usage of string operations and conversions that are not needed for the other solvers.

It is worthwhile to note that the values relative to the in-loop optimizations, including Δ and DT, strictly depend on the order in which we check the pairs of states belonging to the equivalence classes of the current partition $\mathcal{R}$. In fact, if we have a class $\mathcal{C}$ that has to be split in $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$, we may first split out $\mathcal{C}_1$, or $\mathcal{C}_2$, or $\mathcal{C}_3$. This means, for instance, that if we have to match from a state $t \in \mathcal{C}_1$ a transition $s \xrightarrow{\tau} \mu$ such that $\mu(\mathcal{C}_1 \cup \mathcal{C}_2) = 1$ but $\mu(\mathcal{C}_1) < 1$, only the latter split permits to increase Δ. Moreover, the cache hits values are also affected by the fact that when we solve $LP(t, a, \mu_s, \mathcal{R})$, we look for one possible weak transition $t \overset{a}{\Longrightarrow}_{c} \mu_t$ such that $\mu_s \; \mathcal{L}(\mathcal{R}) \; \mu_t$ and in case there are several of them, we just store in the cache the transition computed by the solver. However there is no guarantee that such transition is the same for all solvers, thus the actual content of the cache can be different. This influences the successive cache hits, in particular after the split of one class.

It is worthwhile to remark that there are cases where an unlucky order of the partition splits may scale the transitions to check by a factor 20 that causes a significant increase of both $t_{\approx}$ and t_{Solver}. For this motivation, we do not fix an order on the pairs of states we check, and we let it depend on the nondeterministic insertion of states in the classes. In fact, for a fixed challenger state s, we generate the required $LP(t, a, \mu_s, \mathcal{R})$ problems for each challenging transition $s \xrightarrow{a} \mu_s$ and each defender state t and then we use a pool of solvers running in parallel to verify them. In case of failure, we add the failing defender t to $\mathcal{C}_f$ but the order of such additions is nondeterministic since it depends on the solver running time, the scheduling of the Java threads interacting with the solvers, the operating system scheduling of the solvers, and so on.

As the tables show, the in-loop optimizations reduce considerably the number of transitions that have to be checked by calling the LP or SMT solver, thus making the program faster. In fact, there is a strict correlation between the number of checked transitions and the time spent by the solver. This can be taken as a justification for the claim that weak bisimulation minimization does not scale very well to larger automata, unless the automaton is given as the composition of several smaller automata running in parallel.

In particular, the experiments demonstrate that our implementation uses a reasonable amount of time on automata whose size is the order of up to $3 \cdot 10^3$ states and transitions. Larger automata are likely to induce a time-out, as exemplified in Table 6. In these cases, compositional minimization is the suggested way to overcome this limitation, as we discuss in the next section.

7.3. Compositional Minimization

To show the practical effectiveness of the minimization in a compositional context, which we discussed in theory in Section 4.2, we consider two case studies that we fail to reduce otherwise, due to their prohibitive size: the Consensus Protocol with three parties and the Dining Cryptographers with four, eight, and ten cryptographers. For instance, by applying Definition 2, the four cryptographers case requires 38416 states and 6380 transitions; eight and ten cryptographers are essentially intractable since they involve around 1.5

Table 6. Compositional minimization of consensus protocol for three parties

Components	$\lvert S_{\circlearrowleft}\rvert$	$\lvert T_{\circlearrowleft}\rvert$	$\lvert[S]_{\approx}\rvert$	$\lvert[T]_{\approx}\rvert$	$t_{\approx}$
$c_3 \parallel p_1$	114	772	95	643	6s
$[c_3 \parallel p_1]_{\approx} \parallel p_2$	570	2502	285	1214	1h31m
$[[c_3 \parallel p_1]_{\approx} \parallel p_2]_{\approx} \parallel p_3$	1172	2352	1	1	1h38m
$c_3 \parallel p_1 \parallel p_2 \parallel p_3$	2720	5568	—time-out—		

Table 7. Compositional minimization of dining cryptographers: termination

Components	$\lvert S_{\circlearrowleft}\rvert$	$\lvert T_{\circlearrowleft}\rvert$	$\lvert[S]_{\approx}\rvert$	$\lvert[T]_{\approx}\rvert$	$t_{\approx}$
$d_1 \parallel d_2$	33	76	6	14	2s
$[d_1 \parallel d_2]_{\approx} \parallel d_3$	39	92	6	14	4s
$[[d_1 \parallel d_2]_{\approx} \parallel d_3]_{\approx} \parallel d_4$	39	92	1	1	5s
$d_1 \parallel d_2 \parallel d_3 \parallel d_4$	2165	4540	1	1	5s
$d_1 \parallel d_2 \parallel \dots \parallel d_8$	1687113	6952248	1	1	13s
$d_1 \parallel d_2 \parallel \dots \parallel d_{10}$	42906171	220947474	1	1	18s

and 300 billions states, respectively. We avoid this by constructing the model compositionally, applying weak bisimulation minimization on the intermediate automata. Moreover, to make this compositional minimization more effective, we use the hiding operator as soon as possible to restrict the visibility of the actions that are "private" between two automata.

Each of the Tables 6, 7 and 8 is split in two parts: the top part contains all intermediate steps performed by the compositional minimization leading to the minimization of the final automaton; in each row, the column $t_{\approx}$ includes the value of the previous row, thus reporting the total time used thus far. The bottom part of the table contains the number of states and transitions of the composed automata without intermediate minimization, and the time for the corresponding compositional minimization.

For the consensus protocol, we can see from the top part of Table 6 that the compositional minimization allows us to reduce the automaton to a single state and transition, representing the fact that the consensus is reached with probability 1, whereas the same reduction can not be obtained within the time-out by first composing the parties and then minimizing the composed automaton. The time required for the former approach actually depends on the intermediate step, where we reduce the automaton $[c_3 \parallel p_1]_{\approx} \parallel p_2$, that returns an automaton that is essentially half of the original one. The main motivation for this situation is that the intermediate automaton has still a lot of visible actions that can not be hidden since they are needed to synchronize with p_3.

On the contrary, the dining cryptographers protocol is a good example that shows how using the hiding operator as soon as possible permits to drastically reduce the size of the minimized automaton. In fact, since the synchronization happens only between cryptographers that are neighbors, such as d_i and d_{i+1}, and such synchronization has to be secret, it makes sense to hide it just after having composed d_i and d_{i+1}. Consider the termination of the dining cryptographers protocol with $n = 4$ cryptographers, as shown in the top part

Table 8. Compositional minimization of dining cryptographers: Anonymity

Components	$\lvert S_{\circlearrowleft}\rvert$	$\lvert T_{\circlearrowleft}\rvert$	$\lvert[S]_{\approx}\rvert$	$\lvert[T]_{\approx}\rvert$	$t_{\approx}$
$i_1 = d_1 \parallel d_2$	41	92	20	41	4s
$i_2 = [i_1]_{\approx} \parallel d_3$	105	247	33	75	33s
$i_3 = [i_2]_{\approx} \parallel d_4$	180	482	45	107	330s
$i_4 = [i_3]_{\approx} \parallel d_5$	248	706	57	139	20m
$[i_4]_{\approx} \parallel d_6$	178	372	7	6	22m
$d_1 \parallel d_2 \parallel d_3 \parallel d_4$	2242	4708	5	4	39s
$d_1 \parallel d_2 \parallel \dots \parallel d_5$	12042	31184	6	5	335s
$d_1 \parallel d_2 \parallel \dots \parallel d_6$	63511	196642	7	6	22m
$d_1 \parallel d_2 \parallel \dots \parallel d_7$	329784	1189626	8	7	59m
$d_1 \parallel d_2 \parallel \dots \parallel d_8$	1689417	6961480	9	8	161m

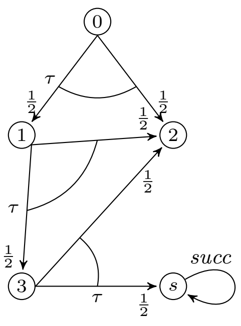


Figure 4. The minimized four dining cryptographers (anonymity)

of Table 7: the proposed combination of hiding and compositional minimization permits to reduce any chain $d_1 \parallel \cdots \parallel d_l$, where $1 < l < n$, to an automaton with 6 states and 14 transitions. Then, for $l = n - 1$, the synchronization of d_1 and d_{n-1} with d_n closes the circle of cryptographers that once minimized shows that the protocol terminates with probability 1.

For the anonymity property, the reduction of each chain does not lead to the same size but to an automaton whose size grows linearly with the number of cryptographers. This is caused by the fact that we have to keep track of the sequence of *agree*s announced by the cryptographers and this number clearly depends on the involved cryptographers. As in the PRISM benchmark, we assume that one cryptographer is paying and we check a particular outcome of the agreement, that is, we check that the probability of a given sequence of *agree*s and *disagree*s is $1/2^{n-1}$. It is immediate to see that the minimized automaton satisfies this property; see for instance the anonymity of four cryptographers in Figure 4, where the probability of reaching the state s is $1/2^3$.

It is clear that for the dining cryptographers protocol the compositional minimization approach outperforms the minimization of the composition and we expect that this extends to all systems where few components share the same actions.

8. Conclusion

This paper has considered deciding *PA* weak bisimulation which is known to be polynomial [TH15]. After a survey of available polynomial algorithms to solve an LP problem, we established an upper bound on the worst case complexity of the decision problem for general *PA*. We demonstrated that a small modification of the LP problem discussed in [TH15] enables taking advantage of the underlying network structure to improve the practical efficiency of solving the problem.

In addition, we have presented an implementation of the decision algorithm, in the form of a quotienting algorithm enabling to minimise probabilistic automata with respect to weak probabilistic bisimulation. We enhanced this algorithm with several heuristics that permit to reduce the running time of the program considerably, and have shown that minimization can be applied effectively to standard benchmark models. We have also investigated how compositional minimization techniques can be exploited for models consisting of several sub-automata running in parallel.

As future work, we plan to improve the efficiency of our heuristics as well as to optimize the code in order to speed up the response time. Moreover, we plan to investigate heuristics that allow us to optimize the sequence of the parallel compositions in order to take advantage from the compositional minimization approach, as done in [CL11, CH10]. Furthermore, the results of this paper allow a number of directions for further research: the network simplex algorithm specialized for the minimum cost flow problem with additional side constraints can be seen itself as the foremost next step. In fact, designing a new data structure to be able to deal with a large number of additional side constraints is not only a very important contribution in theoretical setting but also it improves the practical efficiency of the decision problem under our consideration.

Acknowledgments This work has been supported by the DFG/NWO Bilateral Research Programme ROCKS, by the DFG as part of the SFB/TR 14 "Automatic Verification and Analysis of Complex Systems" (AVACS), by the European Union Seventh Framework Programme under grant agreements 295261 (MEALS) and 318490 (SENSATION), by the Chinese Academy of Sciences Fellowship for International Young Scientists (Grant 2015VTC029), by the National Natural Science Foundation of China (Grants 61472473 and 61550110249), by the CAS/SAFEA International Partnership Program for Creative Research Teams, and by the CDZ project CAP (GZ 1023).

Part of this work has been done when Andrea Turrini was at Saarland University supported by the Cluster of Excellence "Multimodal Computing and Interaction" (MMCI), part of the German Excellence Initiative.

References

[AHK12] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: A meta-algorithm and applications. *Theory of Computing*, 8:121–164, 2012.

[AMO93] Ravindra K. Ahuja, Thomas J. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.

[Ans99] Kurt M. Anstreicher. Linear programming in $\mathcal{O}(\frac{n^3}{\ln n}L)$ operations. *SIAM J. on Optimization*, 9(4):803–812, 1999.

[Bel01] Peter A. Beling. Exact algorithms for linear programming over algebraic extensions. *Algorithmica*, 31(4):459–478, 2001.

[BF12] Ufuk Bahçeci and Orhan Feyzioğlu. A network simplex based algorithm for the minimum cost proportional flow problem with disconnected subnetworks. *Optimization Letters*, 6:1173–1184, 2012.

[BHH$^+$09] Eckard Böde, Marc Herbstritt, Holger Hermanns, Sven Johr, Thomas Peikenkamp, Reza Pulungan, Jan Rakow, Ralf Wimmer, and Bernd Becker. Compositional dependability evaluation for STATEMATE. *ITSE*, 35(2):274–292, 2009.

[BSS89] Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers; *NP*-completeness, recursive functions and universal machines. *Bullettin of the American Mathematical Society*, 21(1):1–46, 1989.

[BST10] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB standard: Version 2.0. In *SMT*, 2010.

[BT97] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, 1997.

[Cal02] Herminia I. Calvete. Network simplex algorithm for the general equal flow problem. *European J. Operational Research*, 150(3):585–600, 2002.

[CGM$^+$96] Ghassan Chehaibar, Hubert Garavel, Laurent Mounier, Nadia Tawbi, and Ferruccio Zulian. Specification and verification of the PowerScale® bus arbitration protocol: An industrial experiment with LOTOS. In *FORTE*, pages 435–450, 1996.

[CH10] Pepijn Crouzen and Holger Hermanns. Aggregation ordering for massively compositional models. In *ACSD*, pages 171–180, 2010.

[CHLS09] Nicolas Coste, Holger Hermanns, Etienne Lantreibecq, and Wendelin Serwe. Towards performance prediction of compositional models in industrial GALS designs. In *CAV*, volume 5643 of *LNCS*, pages 204–218, 2009.

[CKMS11] Paul Christiano, Jonathan A. Kelner, Aleksander Mądry, and Daniel Spielman. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *STOC*, pages 273–282, 2011.

[CL11] Pepijn Crouzen and Frédéric Lang. Smart reduction. In *FASE*, volume 6603 of *LNCS*, pages 111–126, 2011.

[CS02] Stefano Cattani and Roberto Segala. Decision algorithms for probabilistic bisimulation. In *CONCUR*, volume 2421 of *LNCS*, pages 371–385, 2002.

[Den05] Yuxin Deng. *Axiomatisations and Types for Probabilistic and Mobile Processes.* PhD thesis, École des Mines de Paris, 2005.

[Der70] Cyrus Derman. *Finite State Markovian Decision Processes.* Academic Press, Inc., 1970.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *TACAS*, volume 4963 of *LNCS*, pages 337–340, 2008.

[EHS$^+$13] Christian Eisentraut, Holger Hermanns, Johann Schuster, Andrea Turrini, and Lijun Zhang. The quest for minimal quotients for probabilistic automata. In *TACAS*, volume 7795 of *LNCS*, pages 16–31, 2013.

[EHZ10a] Christian Eisentraut, Holger Hermanns, and Lijun Zhang. Concurrency and composition in a stochastic world. In *CONCUR*, volume 6269 of *LNCS*, pages 21–39, 2010.

[EHZ10b] Christian Eisentraut, Holger Hermanns, and Lijun Zhang. On probabilistic automata in continuous time. In *LICS*, pages 342–351, 2010.

[GHT14] Daniel Gebler, Vahid Hashemi, and Andrea Turrini. Computing behavioral relations for probabilistic concurrent systems. In *Stochastic Model Checking. Rigorous Dependability Analysis Using Model Checking Techniques for Stochastic Systems*, volume 8453 of *LNCS*, pages 117–155. Springer Berlin Heidelberg, 2014.

[GLP] GNU linear programming kit. `http://www.gnu.org/software/glpk/`.

[GSL96] Susanne Graf, Bernhard Steffen, and Gerald Lüttgen. Compositional minimisation of finite state systems using interface specifications. *Formal Aspects of Computing*, 8(5):607–616, 1996.

[Han91] Hans A. Hansson. *Time and Probability in Formal Design of Distributed Systems.* PhD thesis, Uppsala University, 1991.

[HHT13] Vahid Hashemi, Holger Hermanns, and Andrea Turrini. On the efficiency of deciding probabilistic automata weak bisimulation. *ECEASST*, 66, 2013.

[HK95] Richard V. Helgason and Jeffery L. Kennington. Primal simplex algorithms for minimum cost network flows.

In *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, chapter 2, pages 85–113. Elsevier, 1995.

[HK00] Holger Hermanns and Joost-Pieter Katoen. Automated compositional Markov chain generation for a plain-old telephone system. *Science of Computer Programming*, 36(1):97–127, 2000.

[HN94] Dorit S. Hochbaum and Joseph Seffi Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM J. on Computing*, 23(6):1179–1192, 1994.

[JL91] Bengt Jonsson and Kim Guldstrand Larsen. Specification and refinement of probabilistic processes. In *LICS*, pages 266–277, 1991.

[JT80] William B. Jones and Wolfgang Joseph Thron. *Continued Fractions: Analytic Theory and Applications.* Encyclopedia of Mathematics and its Applications. Addison-Wesley, 1980.

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[Kha79] Leonid Genrikhovich Khachyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20(1):191–194, 1979.

[KKZJ07] Joost-Pieter Katoen, Tim Kemna, Ivan S. Zapreev, and David N. Jansen. Bisimulation minimisation mostly speeds up probabilistic model checking. In *TACAS*, volume 4424 of *LNCS*, pages 76–92, 2007.

[KM72] Victor Klee and George J. Minty. How good is the simplex algorithm? In *Inequalities*, volume III, pages 159–175. Defense Technical Information Center, 1972.

[KM00] Jean-Pierre Krimm and Laurent Mounier. Compositional state space generation with partial order reductions for asynchronous communicating systems. In *TACAS*, volume 1785 of *LNCS*, pages 266–282, 2000.

[KNP11] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, pages 585–591, 2011.

[KS90] Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *I&C*, 86(1):43–68, 1990.

[LpS] LpSolve mixed integer linear programming solver. `http://lpsolve.sourceforge.net`.

[Mil89] Robin Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.

[MSJ11] David R. Morrison, Jason J. Sauppe, and Sheldon H. Jacobson. A network simplex algorithm for the equal flow problem on a generalized network. *INFORMS J. on Computing*, 25(1):2–12, 2011.

[MSJ13] David R. Morrison, Jason J. Sauppe, and Sheldon H. Jacobson. An algorithm to solve the proportional network flow problem. *Optimization Letters*, 8(3):801–809, 2013.

[PLS00] Anna Philippou, Insup Lee, and Oleg Sokolsky. Weak bisimulation for probabilistic systems. In *CONCUR*, volume 1877 of *LNCS*, pages 334–349, 2000.

[PRI] PRISM model checker. `http://www.prismmodelchecker.org/`.

[PS04] Augusto Parma and Roberto Segala. Axiomatization of trace semantics for stochastic nondeterministic processes. In *QEST*, pages 294–303, 2004.

[PT87] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM J. on Computing*, 16(6):973–989, 1987.

[Pul89] P. Simin Pulat. A decomposition algorithm to determine the maximum flow in a generalized network. *Computers & Operations Research*, 16:161–172, 1989.

[Sch03] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.

[Seg95] Roberto Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, MIT, 1995.

[Seg06] Roberto Segala. Probability and nondeterminism in operational models of concurrency. In *CONCUR*, volume 4137 of *LNCS*, pages 64–78, 2006.

[Sha87] Ron Shamir. The efficiency of the simplex method: A survey. *Management Science*, 33(3):301–334, 1987.

[SL95] Roberto Segala and Nancy A. Lynch. Probabilistic simulations for probabilistic processes. *Nordic J. Computing*, 2(2):250–273, 1995.

[TH15] Andrea Turrini and Holger Hermanns. Polynomial time decision algorithms for probabilistic automata. *I&C*, 244:134–171, 2015.

[Var85] Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *FOCS*, pages 327–338, 1985.

[Vaz04] Vijay V. Vazirani. *Approximation Algorithms.* Springer, 2004.