Quantitative Simulations by Matrices ☆

Natsuki Urabe[a,*], Ichiro Hasuo[a]

[a] *Department of Computer Science, Graduate School of Information Science and Technology, the University of Tokyo, 7-3-1, Hongo, Bunkyo-ku, Tokyo, Japan*

Abstract

We introduce notions of simulation between semiring-weighted automata as models of quantitative systems. Our simulations are instances of the categorical/coalgebraic notions previously studied by Hasuo—hence soundness against language inclusion comes for free—but are concretely presented as matrices that are subject to linear inequality constraints. Pervasiveness of these formalisms allows us to exploit existing algorithms in: searching for a simulation, and hence verifying quantitative correctness that is formulated as language inclusion. Transformations of automata that aid search for simulations are introduced, too. This verification workflow is implemented for the plus-times and max-plus semirings. Furthermore, an extension to weighted *tree* automata is presented and implemented.

1. Introduction

Quantitative aspects of various systems are more and more emphasized in recent verification scenarios. Probabilities in randomized or fuzzy systems are a classic example; utility in economics and game theory is another. Furthermore, now that many computer systems are integrated into physical ambience—realizing so-called *cyber-physical systems*—physical quantities like energy consumption are necessarily taken into account.

1.1. Semiring-Weighted Automata

It is standard in the concurrency community to model such quantitative systems by state-transition systems in which *weights* are assigned to their states and/or transitions. The semantics of such systems varies, however, depending on the interpretation of weights. If they are probabilities, they are accumulated by $\times$ along a path and summed across different paths; if weights are (worst-case) costs, they are summed up along a path and we would take max across different paths.

The algebraic structure of *semirings* then arises as a uniform mathematical language for different notions of "weight," as is widely acknowledged in the community. The

☆ An earlier version of this paper [38] has been presented at: *Concurrency Theory—25th International Conference (CONCUR 2014)*, September 2–5 2014, Rome, Italy.

* Corresponding author

Email addresses: urabenatsuki@is.s.u-tokyo.ac.jp (Natsuki Urabe), ichiro@is.s.u-tokyo.ac.jp (Ichiro Hasuo)

subject of the current study is state-based systems with labeled transitions, in which each transition is assigned a weight from a prescribed semiring $\mathcal{S}$. We shall call them *$\mathcal{S}$-weighted automata*; and we are more specifically interested in the (weighted, finite) *language inclusion* problem and a *simulation-based* approach to it.

1.2. Language Inclusion

Let $\mathcal{A}$ be an $\mathcal{S}$-weighted automaton with labels from an alphabet Σ. It assigns to each word $w \in \Sigma^*$ a weight taken from $\mathcal{S}$—this is much like a (purely) probabilistic automaton assigns a probability to each word. Let us denote this function by $L(\mathcal{A})\colon \Sigma^* \to \mathcal{S}$ and call it the *(weighted) language* of $\mathcal{A}$ by analogy with classic automata theory. The *language inclusion* problem $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$ asks if: $L(\mathcal{A})(w) \sqsubseteq L(\mathcal{B})(w)$ for each word $w \in \Sigma^*$, where $\sqsubseteq$ is a natural order on the semiring $\mathcal{S}$.

It is not hard to see that language inclusion $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$ has numerous applications in verification. In a typical scenario, one of $\mathcal{A}$ and $\mathcal{B}$ is a model of a *system* and the other expresses *specification*; and $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$ gives the definition of "the system meeting the specification." More concrete examples are as follows.

- $\mathcal{S}$ represents probabilities; $\mathcal{A}$ models a system; and $\mathcal{B}$ expresses the specification that certain bad behaviors—identified with words—occur with a certain probability. Then $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$ is a *safety* statement: each bad behavior occurs in $\mathcal{A}$ at most as likely as in $\mathcal{B}$.
- $\mathcal{S}$ represents profit, $\mathcal{A}$ is a specification and $\mathcal{B}$ is a system. Then $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$ guarantees the minimal profit yielded by the system $\mathcal{B}$.
- There are other properties reduced to language inclusion in a less trivial manner. An example is *probable innocence* [35], a quantitative notion of anonymity. See [22].

1.3. Simulation

Direct check of language inclusion is simply infeasible because there are infinitely many words $w \in \Sigma^*$. One finitary proof method—well-known for nondeterministic (i.e. possibilistic) systems—is by *(forward or backward) simulations*, whose systematic study is initiated in [31]. In the nondeterministic setting, a simulation R is a relation between states of $\mathcal{A}$ and $\mathcal{B}$ that witnesses "local language inclusion"; moreover, from the coinductive way in which it is defined, a simulation persistently witnesses local language inclusion—ultimately yielding (global) language inclusion. This property—existence of a simulation implies language inclusion—is called *soundness*.

1.4. Contribution: Weighted Forward/Backward Simulations by Matrices

In this paper we extend this simulation approach to language inclusion [31] to the quantitative setting of semiring-weighted automata. Our notions of (forward and backward) weighted simulation are not given by relations, but by *matrices* with entries from a semiring $\mathcal{S}$.

Use of matrices in automata theory is classic—in fact our framework instantiates to that in [31] when we take as $\mathcal{S}$ the Boolean semiring. This is not how we arrived here; conversely, the current results are obtained as instances of a more general theory of *coalgebraic simulations* [18, 21, 19]. There various systems are identified with a categorical

construct of *coalgebras* in a Kleisli category; and forward and backward simulations are characterized as lax/oplax morphisms between coalgebras. A generic soundness result (with respect to language/trace inclusion) is also proved in the general categorical terms.

This paper is devoted to concrete presentations of these categorical notions by matrices, and to their application to actual verification of quantitative systems. Presentation by matrices turns out to be an advantage: a simulation is now a matrix X that satisfies certain *linear inequalities*; and existence of such X—i.e. feasibility of linear inequalities—is so common a problem in many fields that there is a large body of existing work that is waiting to be applied. For example *linear programming (LP)* can be exploited for the plus-times semiring for probabilities; and there are algorithms proposed for other semirings such as the max-plus (tropical) one.

Our (mostly semiring-independent) workflow is as follows. A verification goal is formulated as language inclusion $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$, which we aim to establish by finding a forward or backward simulation from $\mathcal{A}$ to $\mathcal{B}$. Soundness of simulations follows from the general result in [18]. A simulation we seek for is a matrix subject to certain linear inequalities, existence of which is checked by various algorithms that exist for different semirings. We implemented this workflow for the plus-times and max-plus semirings.

This simulation-based method is sound but not necessarily complete with respect to language inclusion. Therefore we introduce transformations of weighted automata—called *(forward/backward) partial execution*—that potentially create matrix simulations. Via our equivalence results between our matrix simulation and some known ones (including the one in [7]), the partial execution transformations potentially create those simulations, too.

Compared to the earlier version [38] of this paper, the current version additionally contains the following materials.

- Section 7 is added, where we exploit the coalgebraic theory behind and generalize matrix simulation from weighted (word) automata to weighted *tree* automata. We describe the definition of forward partial execution in categorical terms, too, so that it transfers to weighted tree automata. We also have a preliminary implementation.
- We now have more extensive discussions of related work, including [3, 15, 8] of which we were not aware before.
- We conducted experiments again with a faster machine, enlarging the size of problem instances that can be handled.
- We have some examples that were absent in the previous version [38].
- Concrete description of the procedure (forward/backward) partial execution is included.
- Some proofs were omitted in [38] for space reasons; they are present here.

1.5. Organization of the paper

In Section 2 that is devoted to preliminaries, we define semiring-weighted automata, characterize them in coalgebraic terms and recap the coalgebraic theory in [18]. These are combined to yield the notion of simulation matrix in Section 3. In Section 4 partial execution transformations of automata are described and proved correct. The framework

obtained so far is applied to the plus-times and max-plus semirings, in Section 5 and Section 6, respectively. There our proof-of-concept implementations (the code is found at the first author's webpage) and relationship to other known simulation notions are discussed, too. In Section 7 we generalize the framework so far from words to trees: the generalization is straightforward—thanks to the coalgebraic backend—although linearity of constraints, as well as backward partial execution, is lost. In Section 8 (and in earlier sections) we discuss related work; in Section 9 we conclude.

2. Preliminaries

We review the generic theory of traces and simulations in [21, 18] that is based on (T, F)-systems, which will eventually lead to the notion of simulation matrix in Section 3.

2.1. Semiring-Weighted Automata

The notion of semiring-weighted automaton is parametrized by a semiring $\mathcal{S}$. For our purpose of applying coalgebraic theory in [18, 21], we impose the following properties.

Definition 2.1 A *commutative cppo-semiring* is a tuple $\mathcal{S} = (S, +_{\mathcal{S}}, 0_{\mathcal{S}}, \times_{\mathcal{S}}, 1_{\mathcal{S}}, \sqsubseteq)$ that satisfies the following conditions.

- $(S, +_{\mathcal{S}}, 0_{\mathcal{S}}, \times_{\mathcal{S}}, 1_{\mathcal{S}})$ is a semiring in which $\times_{\mathcal{S}}$, in addition to $+_{\mathcal{S}}$, is commutative.
- A relation $\sqsubseteq$ is a partial order on S and $(S, \sqsubseteq)$ is ω-*complete*, i.e. an increasing chain $s_0 \sqsubseteq s_1 \sqsubseteq \cdots$ has a supremum.
- Any element $s \in S$ is *positive* in the sense that $0_{\mathcal{S}} \sqsubseteq s$.
- Addition $+_{\mathcal{S}}$ and multiplication $\times_{\mathcal{S}}$ are monotone with respect to $\sqsubseteq$.

It follows from positivity and ω-completeness that countable sum can be straightforwardly defined in a commutative cppo-semiring $\mathcal{S}$. We will use this fact throughout the paper.

Example 2.2 (semirings $\mathcal{S}_{+,\times}, \mathcal{S}_{\max,+}, \mathcal{B}$) The *plus-times semiring* $\mathcal{S}_{+,\times} = ([0,\infty], +, 0, \times, 1, \leq)$ is a commutative cppo-semiring, where $+$ and $\times$ are usual addition and multiplication of real numbers. This is the semiring that we will use for modeling probabilistic branching. Specifically, probabilities of successive transitions are accumulated using $\times$, and those of different branches are combined with $+$.

The *max-plus semiring* $\mathcal{S}_{\max,+} = ([-\infty,\infty], \max, -\infty, +, 0, \leq)$—also sometimes called the *tropical semiring* [33]—is also a commutative cppo-semiring. Here a number $r \in [-\infty,\infty]$ can be understood as (best-case) *profit*: they are summed up along a path, and an optimal one (max) is chosen among different branches. Another possible understanding of r is as (worst-case) *cost*. The unit for the semiring addition max is given by $-\infty$; since it must also be a zero element of the semiring multiplication $+$, we define $(-\infty) + \infty = -\infty$. In the two examples $\mathcal{S}_{+,\times}$ and $\mathcal{S}_{\max,+}$ we added ∞ so that they become ω-complete.

Finally, the *Boolean semiring* $\mathcal{B} = (\{0,1\}, \vee, 0, \wedge, 1, \leq)$ is an example that is qualitative rather than quantitative.

Definition 2.3 ($\mathcal{S}$-weighted automaton, weighted language) Let $\mathcal{S} = (S, +_{\mathcal{S}}, 0_{\mathcal{S}}, \times_{\mathcal{S}}, 1_{\mathcal{S}}, \sqsubseteq)$ be a commutative cppo-semiring. An *$\mathcal{S}$-weighted automaton* $\mathcal{A} = (Q, \Sigma, M, \alpha, \beta)$ consists of a countable state space Q, a countable alphabet Σ, transition matrices $M(a) \in S^{Q \times Q}$ for all $a \in \Sigma$, the initial row vector $\alpha \in S^Q$ and the final column vector $\beta \in S^Q$.

Let $x, y \in Q$ and $a \in \Sigma$. We write α_x and β_x for the x-th entry of α and β, respectively, and $M(a)_{x,y}$ for the (x, y)-entry of the matrix $M(a)$. Note that these entries are all elements of the semiring $\mathcal{S}$.

An $\mathcal{S}$-weighted automaton $\mathcal{A} = (Q, \Sigma, M, \alpha, \beta)$ yields a *weighted language* $L(\mathcal{A})\colon \Sigma^* \to \mathcal{S}$. It is given by the following multiplication of matrices and vectors.

$$L(\mathcal{A})(w) \;:=\; \alpha \cdot M(a_1) \cdot \;\cdots\; \cdot M(a_k) \cdot \beta \qquad \text{for each } w = a_1 \cdots a_k \in \Sigma^*. \tag{1}$$

We require a state space Q to be at most countably infinite. This is so that the matrix multiplications in (1)—by addition and multiplication of $\mathcal{S}$—are well-defined. Recall that $\mathcal{S}$ has countable sum given by supremums of suitable ω-chains.

Our interest is in establishing language inclusion between two weighted automata.

Definition 2.4 (language inclusion) We write $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$ if, for each $w \in \Sigma^*$, $L(\mathcal{A})(w) \sqsubseteq L(\mathcal{B})(w)$. The last $\sqsubseteq$ is the order of $\mathcal{S}$.

2.2. Coalgebraic Modeling of Semiring-Weighted Automata

Here we characterize semiring-weighted automata as instances of a generic coalgebraic model of branching systems—so-called *(T, F)-systems* with parameters T, F [21, 18].

Definition 2.5 ((T, F)-system) Let T be a monad and F be a functor, both on the category **Sets** of sets and functions. A *(T, F)-system* is a triple

$$\mathcal{X} \;=\; \big(\, X,\;\; s\colon \{\bullet\} \to TX,\;\; c\colon X \to TFX \,\big)$$

of a set X (the *state space*), and functions s (the *initial states*) and c (the *dynamics*).

This modeling is coalgebraic [24] in the sense that c is so-called a TF-coalgebra. In the definition we have two parameters T and F. Let us forget about their categorical structures (a *monad* or a *functor*) for a moment and think of them simply as constructions on sets. Intuitively speaking, T specifies what kind of *branching* the systems in question exhibit; and F specifies a type of *linear-time behaviors*. Here are some examples; in the example $F = 1 + \Sigma \times (\_)$ the only element of 1 is denoted by $\checkmark$ (i.e. $1 = \{\checkmark\}$).

T	"branching"	F	"linear-time behavior"
$\mathcal{P}$	nondeterministic	$1 + \Sigma \times (\_)$	$\to \checkmark$ or $\xrightarrow{a}$ (where $a \in \Sigma$)
$\mathcal{D}$	probabilistic	$(\Sigma + (\_))^*$	words over terminals ($a \in \Sigma$)
$\mathcal{M}_{\mathcal{S}}$	$\mathcal{S}$-weighted		& nonterminals, suited for CFG [20]

The above examples of a monad T—the *powerset monad* $\mathcal{P}$, the *subdistribution monad* $\mathcal{D}$, and the *$\mathcal{S}$-multiset monad* $\mathcal{M}_{\mathcal{S}}$ for $\mathcal{S}$—are described as follows.

$$\begin{aligned} \mathcal{P}X &= \{X' \mid X' \subseteq X\} \qquad \mathcal{D}X = \{f : X \to [0,1] \mid \textstyle\sum_{x \in X} f(x) \leq 1\} \\ \mathcal{M}_{\mathcal{S}}X &= \{f : X \to S \mid \mathrm{supp}(f) \text{ is countable}\} \end{aligned} \tag{2}$$

Here $\mathrm{supp}(f) = \{x \in X \mid f(x) \neq 0_{\mathcal{S}}\}$. Countable support in $\mathcal{M}_{\mathcal{S}}$ is a technical requirement so that composition $\odot$ of Kleisli arrows is well-defined (Definition 2.7).

It should not be hard to see that a (T, F)-system models a state-based system with T-branching and F-linear-time behaviors. For example, when $T = \mathcal{P}$ and $F = 1 + \Sigma \times (\_)$, $s\colon \{\bullet\} \to \mathcal{P}X$ represents the set of initial states and $c\colon X \to \mathcal{P}(1 + \Sigma \times X)$ represents one-step transitions—that $\checkmark \in c(x)$ means x is accepting ($x \to \checkmark$), and $(a, x') \in c(x)$ means there is a transition $x \xrightarrow{a} x'$. Overall, a $(\mathcal{P}, 1 + \Sigma \times (\_))$-system is nothing but a nondeterministic automaton.

Analogously we obtain the following, by the definition of $\mathcal{M}_{\mathcal{S}}$ in (2).

Proposition 2.6 (weighted automata as (T, F)-systems) *Let $\mathcal{S}$ be a commutative cppo-semiring. There is a bijective correspondence between: 1) $\mathcal{S}$-weighted automata (Definition 2.3); and 2) $\big(\mathcal{M}_{\mathcal{S}}, 1 + \Sigma \times (\_)\big)$-systems whose state spaces are at most countably infinite.*

Concretely, an $\mathcal{S}$-weighted automaton $\mathcal{A} = (Q, \Sigma, M, \alpha, \beta)$ gives rise to an $\big(\mathcal{M}_{\mathcal{S}}, 1 + \Sigma \times (\_)\big)$-system $\mathcal{X}_{\mathcal{A}} = (Q, s_{\mathcal{A}}, c_{\mathcal{A}})$ defined as follows. $s_{\mathcal{A}}\colon \{\bullet\} \to \mathcal{M}_{\mathcal{S}}Q$ is given by $s_{\mathcal{A}}(\bullet)(x) = \alpha_x$; and $c_{\mathcal{A}}\colon Q \to \mathcal{M}_{\mathcal{S}}(1+\Sigma\times Q)$ is given by $c_{\mathcal{A}}(x)(\checkmark) = \beta_x$ and $c_{\mathcal{A}}(x)(a, y) = M(a)_{x,y}$. □

2.3. Coalgebraic Theory of Traces and Simulations

We review the theory of traces and simulations in [21, 18] that is based on (T, F)-systems. In presentation we restrict to $T = \mathcal{M}_{\mathcal{S}}$ and $F = 1 + \Sigma \times (\_)$ for simplicity.

2.3.1. Kleisli Arrows

One notable success of coalgebra was a uniform characterization, in terms of the same categorical diagram, of *bisimulations* for various kinds of systems (nondeterministic, probabilistic, etc.) [24]. This works quite well for branching-time process semantics. For linear-time semantics—i.e. trace semantics—it is noticed in [34] that so-called a *Kleisli category*, in place of the category **Sets**, gives a suitable base category for coalgebraic treatment. This idea—replacing functions $X \to Y$ with *Kleisli arrows* $X \nrightarrow Y$ and drawing the same diagrams—led to the development in [21, 18, 19] of an extensive theory of traces and simulations. The notion of Kleisli arrow is parametrized by a monad T: a T-Kleisli arrow $X \nrightarrow_T Y$ (or simply $X \nrightarrow Y$) is defined to be a function $X \to TY$, hence represents a "T-branching function from X to Y."

We restrict to $T = \mathcal{M}_{\mathcal{S}}$ for simplicity of presentation. An $\mathcal{M}_{\mathcal{S}}$-Kleisli arrow $f\colon X \nrightarrow Y$ below is "an $\mathcal{S}$-weighted function from X to Y." In particular, for each $x \in X$ and $y \in Y$ it assigns a *weight* $f(x)(y) \in S$.

Definition 2.7 (Kleisli arrow) Let X, Y be sets. An *$\mathcal{M}_{\mathcal{S}}$-Kleisli arrow* (or simply a *Kleisli arrow*) from X to Y, denoted by $X \nrightarrow Y$, is a function from X to $\mathcal{M}_{\mathcal{S}}Y$.

We list some special Kleisli arrows: η_X, $g \odot f$ and Jf.

- For each set X, the *unit arrow* $\eta_X\colon X \nrightarrow X$ is given by: $\eta(x)(x) = 1_{\mathcal{S}}$; and $\eta(x)(x') = 0_{\mathcal{S}}$ for $x' \neq x$. Here $0_{\mathcal{S}}$ and $1_{\mathcal{S}}$ are units in the semiring $\mathcal{S}$.

- For consecutive Kleisli arrows $f : X \nrightarrow Y$ and $g : Y \nrightarrow Z$, their *composition* $g \odot f : X \nrightarrow Z$ is given as follows:

$$(g \odot f)(x)(z) \;:=\; \textstyle\sum_{y \in \mathrm{supp}(f(x))} f(x)(y) \times_{\mathcal{S}} g(y)(z) \;\; .$$

  Since $\mathrm{supp}(f(x))$ is countable, the above sum in a cppo-semiring $\mathcal{S}$ is well-defined.

- For a (usual) function $f : X \to Y$, its *lifting* to a Kleisli arrow $Jf : X \nrightarrow Y$ is given by $Jf = \eta_Y \circ f$. Here we identified $\eta_Y : Y \nrightarrow Y$ with a function $\eta_Y : Y \to \mathcal{M}_{\mathcal{S}} Y$.

Categorically speaking: the first two (η and $\odot$) organize Kleisli arrows as a category (the *Kleisli category* $\mathcal{K}\ell(\mathcal{M}_{\mathcal{S}})$); and the third gives a functor $J\colon \mathbf{Sets} \to \mathcal{K}\ell(\mathcal{M}_{\mathcal{S}})$ that is identity on objects.

In Proposition 2.6 we characterized an $\mathcal{S}$-weighted automaton $\mathcal{A}$ in coalgebraic terms. Using Kleisli arrows it is presented as a triple

$$\mathcal{X}_{\mathcal{A}} \;=\; \big(\; Q,\;\; s_{\mathcal{A}}\colon \{\bullet\} \longrightarrow\!\!\!\!\!\rightarrow Q,\;\; c_{\mathcal{A}}\colon Q \longrightarrow\!\!\!\!\!\rightarrow 1 + \Sigma \times Q \;\big) \;\; . \tag{3}$$

2.3.2. Generic Trace Semantics

In [21], for monads T with a suitable order, a final coalgebra in $\mathcal{K}\ell(T)$ is identified. It (somehow interestingly) coincides with an initial algebra in **Sets**. Moreover, the universality of this final coalgebra is shown to capture natural notions of (finite) *trace semantics* for a variety of branching systems—i.e. for different T and F. What is important for the current work is the fact that the weighted language $L(\mathcal{A})$ in (1) is an instance of this generic trace semantics, as we will show in Theorem 2.11.

$$\begin{array}{ccc}
1 + \Sigma \times X & \xrightarrow{\;1 + \Sigma \times (\mathsf{tr}(c))\;} & 1 + \Sigma \times \Sigma^* \\
\uparrow c & = & \mathrm{final} \uparrow J([\mathsf{nil}, \mathsf{cons}]^{-1}) \\
X & \xrightarrow{\qquad \mathsf{tr}(c) \qquad} & \Sigma^* \\
\uparrow s & \nearrow_{\mathsf{tr}(\mathcal{X})} & \\
\{\bullet\} & &
\end{array} \tag{4}$$

We shall state the results in [21] on coalgebraic traces, restricting again to $T = \mathcal{M}_{\mathcal{S}}$ and $F = 1 + \Sigma \times (\_)$ for simplicity. In the diagram (4) above, composition of Kleisli arrows are given by $\odot$ in Definition 2.7; J on the right is the lifting in Definition 2.7; and nil and cons are the obvious constructors of words in Σ^*. The top arrow $1 + \Sigma \times (\mathsf{tr}(c))$ is the functor $1 + \Sigma \times (\_)$ on **Sets**, lifted to the Kleisli category $\mathcal{K}\ell(\mathcal{M}_{\mathcal{S}})$, and applied to the Kleisli arrow $\mathsf{tr}(c)$; its concrete description is as follows. See [21] for more details.

Definition 2.8 For a Kleisli arrow $f\colon X \nrightarrow Y$, its lifting $1 + \Sigma \times f\colon 1 + \Sigma \times X \nrightarrow 1 + \Sigma \times Y$ is defined as follows:

$$(1 + \Sigma \times f)(*)(t) = \begin{cases} 1_{\mathcal{S}} & (t = *) \\ 0_{\mathcal{S}} & (\text{otherwise}) \end{cases} \qquad (1 + \Sigma \times f)(a, x)(t) = \begin{cases} f(x)(y) & (t = (a, y)) \\ 0_{\mathcal{S}} & (\text{otherwise}) \end{cases}$$

Theorem 2.9 (final coalgebra in $\mathcal{K}\ell(\mathcal{M}_{\mathcal{S}})$) *Given any set X and any Kleisli arrow $c\colon X \nrightarrow 1 + \Sigma \times X$, there exists a unique Kleisli arrow $\mathsf{tr}(c)$ that makes the top square in the diagram (4) commute.* □

Definition 2.10 ($\mathsf{tr}(\mathcal{X})$) Given an $\big(\mathcal{M}_{\mathcal{S}}, 1 + \Sigma \times (\_)\big)$-system $\mathcal{X} = (X, s, c)$ (this is on the left in the diagram (4)), its component c induces an arrow $\mathsf{tr}(c)\colon X \nrightarrow \Sigma^*$ by Theorem 2.9. We define $\mathsf{tr}(\mathcal{X})$ to be the composite $\mathsf{tr}(c) \odot s$ (the bottom triangle in the diagram (4)), and call it the *trace semantics* of $\mathcal{X}$.

Theorem 2.11 (weighted language as trace semantics) *Let $\mathcal{A}$ be an $\mathcal{S}$-weighted automaton. For $\mathcal{X}_{\mathcal{A}} = (Q, s_{\mathcal{A}}, c_{\mathcal{A}})$ induced by $\mathcal{A}$ in (3), its trace semantics $\mathsf{tr}(\mathcal{X}_{\mathcal{A}})\colon \{\bullet\} \nrightarrow \Sigma^*$—identified with a function $\{\bullet\} \to \mathcal{M}_{\mathcal{S}}\Sigma^*$, hence with a function $\Sigma^* \to \mathcal{S}$—coincides with the weighted language $L(\mathcal{A})\colon \Sigma^* \to \mathcal{S}$ in (1).* □

In the last theorem we need that Σ^* is countable; this is why we assumed that Σ is countable in Definition 2.3. Henceforth we do not distinguish $L(\mathcal{A})$ and $\mathsf{tr}(\mathcal{X}_{\mathcal{A}})\colon \{\bullet\} \nrightarrow \Sigma^*$.

2.3.3. Forward and Backward Kleisli *simulations*

In [18], the classic results in [31] on forward and backward simulations—for (nondeterministic) labeled transition systems—are generalized to (T, F)-systems. Specifically, forward and backward simulations are characterized as *lax/oplax coalgebra homomorphisms* in a Kleisli category; and *soundness*—their existence witnesses trace inclusion—is proved once for all in a general categorical setting.

As before, we present those notions and results in [18] restricting to $T = \mathcal{M}_{\mathcal{S}}$ and $F = 1 + \Sigma \times (\_)$. If $T = \mathcal{P}$ and $F = 1 + \Sigma \times (\_)$ they instantiate to the results in [31].

Definition 2.12 (Kleisli simulation) Let $\mathcal{X} = (X, s, c)$ and $\mathcal{Y} = (Y, t, d)$ be $(\mathcal{M}_{\mathcal{S}}, 1 + \Sigma \times (\_))$-systems (cf. Definition 2.5, Proposition 2.6 and (3)).

1. A *forward (Kleisli) simulation* from $\mathcal{X}$ to $\mathcal{Y}$ is a Kleisli arrow $f : Y \nrightarrow X$ such that $s \sqsubseteq f \odot t$ and $c \odot f \sqsubseteq (1 + \Sigma \times f) \odot d$. See Figure 1.
2. A *backward simulation* from $\mathcal{X}$ to $\mathcal{Y}$ is a Kleisli arrow $b : X \nrightarrow Y$ such that $s \odot b \sqsubseteq t$ and $(1 + \Sigma \times b) \odot c \sqsubseteq d \odot b$.
3. A *forward-backward simulation* from $\mathcal{X}$ to $\mathcal{Y}$ consists of: a (T, F)-system $\mathcal{Z}$; a forward simulation f from $\mathcal{X}$ to $\mathcal{Z}$; and a backward simulation b from $\mathcal{Z}$ to $\mathcal{Y}$.
4. A *backward-forward simulation* from $\mathcal{X}$ to $\mathcal{Y}$ consists of: a (T, F)-system $\mathcal{Z}$; a backward simulation b from $\mathcal{X}$ to $\mathcal{Z}$; and a forward simulation f from $\mathcal{Z}$ to $\mathcal{Y}$.

We write $\mathcal{X} \sqsubseteq_{\mathbf{F}} \mathcal{Y}$, $\mathcal{X} \sqsubseteq_{\mathbf{B}} \mathcal{Y}$, $\mathcal{X} \sqsubseteq_{\mathbf{FB}} \mathcal{Y}$ or $\mathcal{X} \sqsubseteq_{\mathbf{FB}} \mathcal{Y}$ if there exists a forward, backward, forward-backward, or backward-forward simulation, respectively.

(Generic) soundness is proved using the maximality of $\mathsf{tr}(c)$ in (4) among (op)lax coalgebra homomorphisms, arguing in the language of enriched category theory [18].

Theorem 2.13 (soundness) *Let $\mathcal{X}$ and $\mathcal{Y}$ be $(\mathcal{M}_{\mathcal{S}}, 1 + \Sigma \times (\_))$-systems. Each of the following yields $\mathsf{tr}(\mathcal{X}) \sqsubseteq \mathsf{tr}(\mathcal{Y})\colon \{\bullet\} \nrightarrow \Sigma^*$ (cf. Definition 2.10).*
1. $\mathcal{X} \sqsubseteq_{\mathbf{F}} \mathcal{Y}$ 2. $\mathcal{X} \sqsubseteq_{\mathbf{B}} \mathcal{Y}$ 3. $\mathcal{X} \sqsubseteq_{\mathbf{FB}} \mathcal{Y}$ 4. $\mathcal{X} \sqsubseteq_{\mathbf{BF}} \mathcal{Y}$ □

Theorem 2.14 (completeness) *The converse of soundness holds for backward-forward simulations. That is: $\mathsf{tr}(\mathcal{X}) \sqsubseteq \mathsf{tr}(\mathcal{Y})$ implies $\mathcal{X} \sqsubseteq_{\mathbf{BF}} \mathcal{Y}$.* □

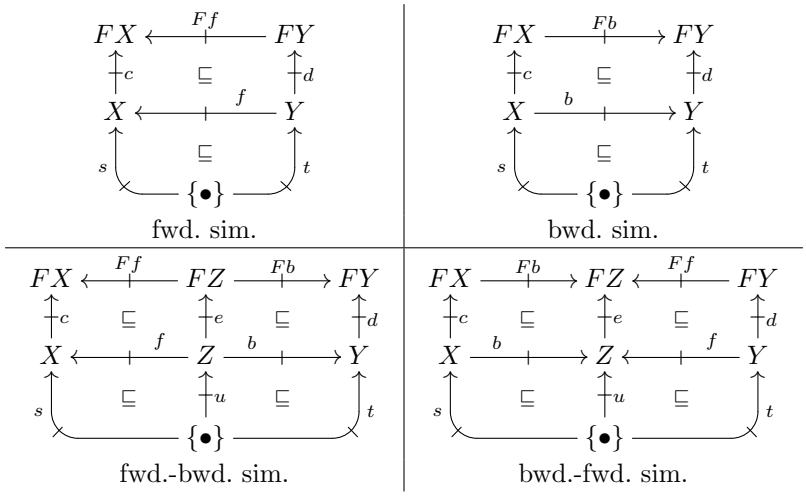


Figure 1: Kleisli simulations (here $F = 1 + \Sigma \times (\_)$)

3. Simulation Matrices for Semiring-Weighted Automata

In this section we fix parameters $T = \mathcal{M}_{\mathcal{S}}$ and $F = 1 + \Sigma \times (\_)$ in the generic theory in Section 2.3 and rephrase the coalgebraic framework in terms of matrices (whose entries are taken from $\mathcal{S}$). Specifically: Kleisli arrows become matrices; and Kleisli simulations become matrices subject to certain linear inequalities. Such matrix representations ease implementation, a feature we will exploit in later sections.

Recall that a Kleisli arrow $A \mathrel{+\!\!\to} B$ is a function $A \to \mathcal{M}_{\mathcal{S}} B$ (Definition 2.7).

Definition 3.1 (matrix representation M_f) Given a Kleisli arrow $f \colon A \mathrel{+\!\!\to} B$, its *matrix representation* $M_f \in S^{A \times B}$ is given by $(M_f)_{x,y} = f(x)(y)$.

In what follows we shall use the notations f and M_f interchangeably.

Lemma 3.2 *Let $f, f' \colon A \mathrel{+\!\!\to} B$ and $g \colon B \mathrel{+\!\!\to} C$ be Kleisli arrows.*

1. *$f \sqsubseteq f'$ if and only if $M_f \sqsubseteq M_{f'}$. Here the former $\sqsubseteq$ is between $\mathcal{M}_{\mathcal{S}}$-Kleisli arrows, and the latter order $\sqsubseteq$ is between matrices, defined entrywise.*
2. *$M_{g \odot f} = M_f M_g$, computed by matrix multiplication.* □

The correspondence from $A \overset{f}{\mathrel{+\!\!\to}} B$ to $1 + \Sigma \times A \overset{1+\Sigma\times f}{\mathrel{+\!\!\to}} 1 + \Sigma \times B$— used in (4) and in Figure 1—can be described using matrices, too.

Lemma 3.3 *Let $f : A \mathrel{+\!\!\to} B$ be a Kleisli arrow and M_f be its matrix representation. Then the matrix representation $M_{1+\Sigma\times f}$ is given by*

$$I_1 \oplus (I_\Sigma \otimes M_f) \;\in\; \mathcal{S}^{(1+\Sigma\times A)\times(1+\Sigma\times B)} \;,$$

where $\oplus$ and $\otimes$ denote coproduct *and* the Kronecker product *of matrices:*

$$X \oplus Y \;=\; \begin{pmatrix} \boxed{X} & O \\ O & \boxed{Y} \end{pmatrix} , \quad \begin{pmatrix} & \vdots & \\ \cdots & x_{i,j} & \cdots \\ & \vdots & \end{pmatrix} \otimes \boxed{Y} = \begin{pmatrix} & \vdots & \\ \cdots & \boxed{x_{i,j}Y} & \cdots \\ & \vdots & \end{pmatrix} .$$

□

This description of M_{Ff} generalizes from $F = 1 + \Sigma \times (\_)$ to any polynomial functor F, inductively on the construction of F. In this paper the generality is not needed.

Using Lemma 3.2–3.3, we can present Kleisli simulations (Definition 2.12) as matrices. Recall that a state space of a weighted automaton is assumed to be countable (Definition 2.3); hence all the matrix multiplications in the definition below make sense.

Definition 3.4 (forward/backward simulation matrix) Let $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, M_{\mathcal{A}}, \alpha_{\mathcal{A}}, \beta_{\mathcal{A}})$ and $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, M_{\mathcal{B}}, \alpha_{\mathcal{B}}, \beta_{\mathcal{B}})$ be $\mathcal{S}$-weighted automata.

- A matrix $X \in S^{Q_{\mathcal{B}} \times Q_{\mathcal{A}}}$ is a *forward simulation matrix* from $\mathcal{A}$ to $\mathcal{B}$ if

$$\alpha_{\mathcal{A}} \sqsubseteq \alpha_{\mathcal{B}} X \;\;, \quad X \cdot M_{\mathcal{A}}(a) \sqsubseteq M_{\mathcal{B}}(a) \cdot X \quad (\forall a \in \Sigma) \;\;, \quad \text{and} \quad X\beta_{\mathcal{A}} \sqsubseteq \beta_{\mathcal{B}} \;\; .$$

- A matrix $X \in S^{Q_{\mathcal{A}} \times Q_{\mathcal{B}}}$ is a *backward simulation matrix* from $\mathcal{A}$ to $\mathcal{B}$ if

$$\alpha_{\mathcal{A}} X \sqsubseteq \alpha_{\mathcal{B}} \;\;, \quad M_{\mathcal{A}}(a) \cdot X \sqsubseteq X \cdot M_{\mathcal{B}}(a) \quad (\forall a \in \Sigma) \;\;, \quad \text{and} \quad \beta_{\mathcal{A}} \sqsubseteq X\beta_{\mathcal{B}} \;\; .$$

The requirements on X are obtained by first translating Figure 1 into matrices, and then breaking them up into smaller matrices using Lemma 3.3. It is notable that the requirements are given in the form of *linear inequalities*, a format often used in constraint solvers. Solving them is a topic of extensive research efforts that include [1, 6]. This fact becomes an advantage in implementing search algorithms, as we see later.

We also note that *forward* and *backward* simulation matrices have different dimensions. This difference comes from the different directions of arrows in Figure 1.

Theorem 3.5 *Let $\mathcal{A}$ and $\mathcal{B}$ be $\mathcal{S}$-weighted automata. There is a bijective correspondence between: 1) forward simulation matrices from $\mathcal{A}$ to $\mathcal{B}$; and 2) forward Kleisli simulations from $\mathcal{X}_{\mathcal{A}}$ to $\mathcal{X}_{\mathcal{B}}$. The same holds for the backward variants.* □

In what follows we write $\sqsubseteq_{\mathbf{F}}, \sqsubseteq_{\mathbf{B}}$ also between $\mathcal{S}$-weighted automata. Theorem 3.5 yields: $\mathcal{A} \sqsubseteq_{\mathbf{F}} \mathcal{B}$ if and only if there is a forward simulation matrix.

Here is our core result; the rest of the paper is devoted to its application.

Corollary 3.6 (soundness of simulation matrices) *Let $\mathcal{A}$ and $\mathcal{B}$ be $\mathcal{S}$-weighted automata. Existence of a forward (or backward) simulation matrix from $\mathcal{A}$ to $\mathcal{B}$—i.e. $\mathcal{A} \sqsubseteq_{\mathbf{F}} \mathcal{B}$ or $\mathcal{A} \sqsubseteq_{\mathbf{B}} \mathcal{B}$—witnesses language inclusion $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$.*

Proof. $\exists$ (fwd./bwd. simulation matrix from $\mathcal{A}$ to $\mathcal{B}$)
$\overset{\text{Thm. 3.5}}{\Longleftrightarrow}$ $\exists$ (fwd./bwd. Kleisli simulation from $\mathcal{X}_{\mathcal{A}}$ to $\mathcal{X}_{\mathcal{B}}$)
$\overset{\text{Thm. 2.13}}{\Longrightarrow}$ $\mathsf{tr}(\mathcal{X}_{\mathcal{A}}) \sqsubseteq \mathsf{tr}(\mathcal{X}_{\mathcal{B}})$ $\overset{\text{Thm. 2.11}}{\Longleftrightarrow}$ $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$. □

It is classic to represent nondeterministic automata by Boolean matrices. This corresponds to the special case $\mathcal{S} = \mathcal{B}$ (the Boolean semiring) of the current framework; and a simulation matrix becomes the same thing as a (relational) simulation in [31].

Remark 3.7 The *opposite* of an $\mathcal{S}$-weighted automaton $\mathcal{A} = (Q, \Sigma, M, \alpha, \beta)$—obtained by reversing transitions and swapping initial/final states—can be naturally defined by matrix transpose, that is, ${}^t\mathcal{A} := (Q, \Sigma, {}^tM, {}^t\beta, {}^t\alpha)$. It is easy to see that: if X is a forward simulation matrix from $\mathcal{A}$ to $\mathcal{B}$, then ${}^tX$ is a backward simulation matrix from ${}^t\mathcal{A}$ to ${}^t\mathcal{B}$.

4. Forward and Backward Partial Execution

In this section we introduce for semiring-weighted automata their transformations—called *forward* and *backward partial execution*—that increase the number of forward or backward simulation matrices. We also prove some correctness results.

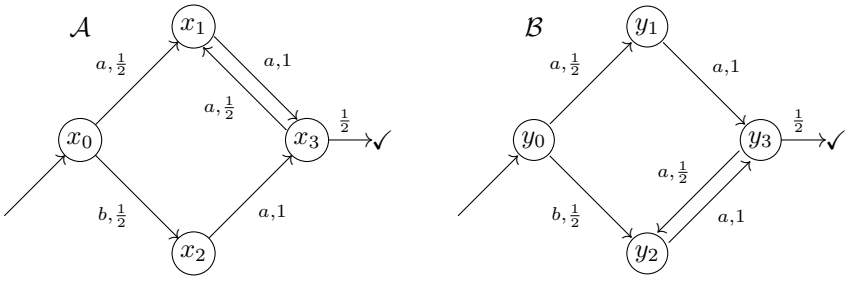
4.1. Incompleteness of Matrix Simulations

We have four different notions of simulation (Definition 2.12): forward, backward, forward-backward, and backward-forward. Our view on these is as (possibly finitary) witnesses of language inclusion.

The combined ones (forward-backward and backward-forward) subsume the one-direction ones (forward and backward)—simply take the identity arrow as one of the two simulations required. Moreover, backward-forward is complete (Theorem 2.14). Despite these theoretical advantages, the combined simulations are generally harder to find: in addition to two simulations, we have to find an intermediate system too ($\mathcal{Z}$ in Definition 2.12). Furthermore, since language inclusion for finite $\mathcal{S}_{+,\times}$-weighted automata—models of probabilistic systems—is known to be undecidable [5], existence of a backward-forward simulation is undecidable too.

Therefore in what follows we focus on the one-directional (i.e. forward or backward) simulations as proof methods for language inclusion. They have convenient matrix presentations, too, as we saw in Section 3. However forward or backward simulations are not necessarily complete. We can see it from the following counterexample.

Example 4.1 ($\sqsubseteq_{\mathbf{F}}$ and $\sqsubseteq_{\mathbf{B}}$ are not complete) The following $\mathcal{S}_{+,\times}$-weighted automata exhibit $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$ (in fact $L(\mathcal{A}) = L(\mathcal{B})$).



Indeed, for each word $w \in \Sigma^*$, we have

$$L(\mathcal{A})(w) = L(\mathcal{B})(w) = \begin{cases} \frac{1}{4}\left(\frac{1}{2}\right)^n & (w = aaa^{2n} \text{ or } baa^{2n}) \\ 0 & (\text{otherwise}) \ . \end{cases}$$

However there is no forward or backward simulation from $\mathcal{A}$ to $\mathcal{B}$: one can show by direct calculation that there is no X that satisfies the requirements in Definition 3.4. Hence this pair is a counterexample for the completeness of $\sqsubseteq_{\mathbf{F}}$ and that of $\sqsubseteq_{\mathbf{B}}$.

It turns out that a simulation in the sense of Jonsson and Larsen [25] does exist from $\mathcal{A}$ to $\mathcal{B}$. See also Section 5.1 later, where we systematically compare our current notions of simulation with existing ones.

The incompleteness of forward and backward simulations can be also deduced from complexity arguments. See Section 5.1.

4.2. Forward and Backward Partial Execution for Semiring-weighted Automata

We shall define transformations, called FPE and BPE, that increase matrix simulations for semiring-weighted automata. We prove some of its properties, too.

Definition 4.2 (FPE, BPE) *Forward partial execution (FPE)* is a transformation of a weighted automaton that "replaces some states with their one-step behaviors." Concretely, given an $\mathcal{S}$-weighted automaton $\mathcal{A} = (Q, \Sigma, M, \alpha, \beta)$ and a parameter $P \subseteq Q$, the resulting automaton $\mathcal{A}_{\mathsf{FPE},P} = (Q', \Sigma, M', \alpha', \beta')$ has a state space

$$Q' \;=\; \big\{\checkmark \,\big|\, \exists x \in P.\, \beta_x \neq 0_{\mathcal{S}}\big\} + \big\{(a,y) \,\big|\, \exists x \in P.\, M(a)_{x,y} \neq 0_{\mathcal{S}}\big\} + (Q \setminus P) \;\;, \qquad (5)$$

replacing each $p \in P$ with its one-step behaviors ($\checkmark$ or (a,q)) as new states. The other data M', α', β' are defined as follows. For the transition matrices M':

$$\begin{array}{rclrclrcl} M'(a)_{(a,x),\checkmark} &=& \beta_x & M'(a)_{(a,x),(a',y)} &=& M(a')_{x,y} & M'(a)_{(a,x),x} &=& 1_{\mathcal{S}} \\ M'(a)_{x,\checkmark} &=& \big(M(a)\beta\big)_x & M'(a)_{x,(a',y)} &=& \big(M(a)M(a')\big)_{x,y} & M'(a)_{x,y} &=& M(a)_{x,y} \end{array}$$

where $a, a' \in \Sigma, x, y \in Q$. For all the other cases we define $M'(a)_{u,v} = 0_{\mathcal{S}}$, where $u, v \in Q'$. For the initial and final vectors α' and β', the definition is shown below.

$$\begin{array}{rclrclrcl} \alpha'_{\checkmark} &=& \alpha\beta & \alpha'_{(a,x)} &=& \big(\alpha M(a)\big)_x & \alpha'_x &=& \alpha_x \\ \beta'_{\checkmark} &=& 1_{\mathcal{S}} & \beta'_{(a,x)} &=& 0_{\mathcal{S}} & \beta'_x &=& \beta_x \end{array}$$

Backward partial execution (BPE) in contrast "replaces states in a parameter $P \subseteq Q$ with their backward one-step behaviors." For the same $\mathcal{A}$ as above, the resulting automaton $\mathcal{A}_{\mathsf{BPE},P} = (Q', \Sigma, M', \alpha', \beta')$ has a state space

$$Q' \;=\; \big\{\bullet \,\big|\, \exists x \in P.\, \alpha_x \neq 0_{\mathcal{S}}\big\} + \big\{(a,y) \,\big|\, \exists x \in P.\, M(a)_{y,x} \neq 0_{\mathcal{S}}\big\} + (Q \setminus P) \;\;, \qquad (6)$$

replacing each $p \in P$ with its backward one-step behaviors—(a,q) with $q \xrightarrow{a} p$, and $\bullet$ if p is initial—as new states. The other data M', α', β' are defined as follows. For the transition matrices M':

$$\begin{array}{rclrclrcl} M'(a)_{\bullet,(a,y)} &=& \alpha_y & M'(a)_{(a',x),(a,y)} &=& M(a')_{x,y} & M'(a)_{x,(a,x)} &=& 1_{\mathcal{S}} \\ M'(a)_{\bullet,y} &=& \big(\alpha M(a)\big)_y & M'(a)_{(a',x),y} &=& (M(a')M(a))_{x,y} & M'(a)_{x,y} &=& M(a)_{x,y} \end{array}$$

where $a, a' \in \Sigma, x, y \in Q$. For all the other cases we define $M'(a)_{u,v} = 0_{\mathcal{S}}$, where $u, v \in Q'$. For the initial and final vectors α' and β', the definition is shown below.

$$\begin{array}{rclrclrcl} \alpha'_{\bullet} &=& 1_{\mathcal{S}} & \alpha_{(a,x)} &=& 0_{\mathcal{S}} & \alpha'_x &=& \alpha_x \\ \beta'_{\bullet} &=& \alpha\beta & \beta'_{(a,x)} &=& (M(a)\beta)_x & \beta'_x &=& \beta_x \end{array}$$

Pictorially, the actions of FPE and BPE can be illustrated as in Figure 2. Roughly speaking, FPE replaces a *concrete* state $p \in P$ with an *abstract* state, such as (a,q) in Q' of (5) that is thought of as a description "a state that makes an a-transition to q." The idea comes from *partial evaluation* of a program; hence the name.

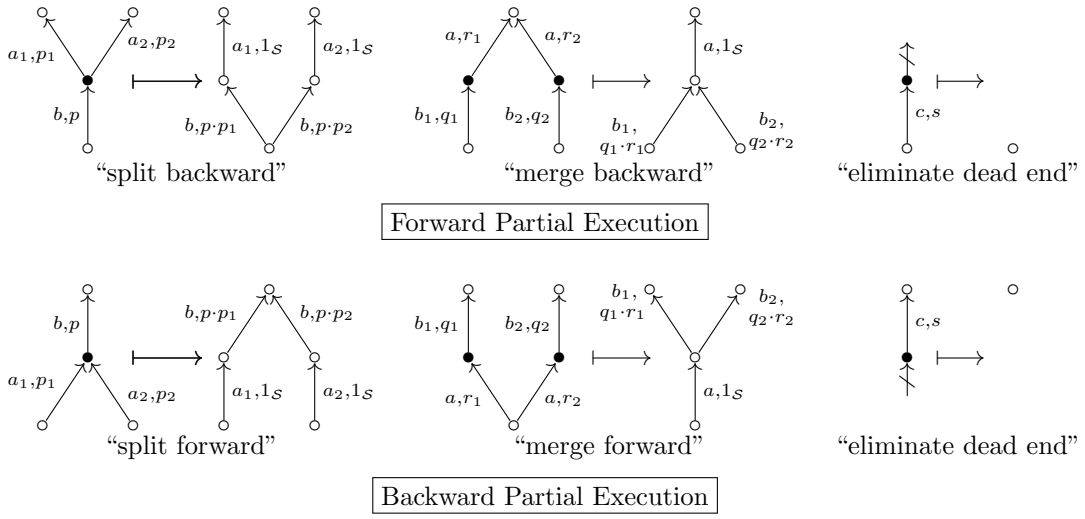


Figure 2: Fwd./bwd. partial execution (FPE, BPE), pictorially. Black nodes need to be in P

4.3. Correctness of FPE and BPE

The use of FPE/BPE is as follows: we aim to establish $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$; depending on whether we search for a forward or backward simulation matrix, we apply one of FPE and BPE to each of $\mathcal{A}$ and $\mathcal{B}$, according to the above table (7).

goal: $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$	$\mathcal{A}$	$\mathcal{B}$
by $\sqsubseteq_{\mathbf{F}}$	FPE	BPE
by $\sqsubseteq_{\mathbf{B}}$	BPE	FPE

(7)

We shall now state correctness properties of this strategy. *Soundness* means that discovery of a simulation after transformation indeed witnesses the language inclusion for the *original* automata. The second property—we call it *adequacy*—states that simulations that are already there are preserved by partial execution.

Theorem 4.3 (soundness of FPE/BPE) *Let P and P' be arbitrary subsets of the state spaces of $\mathcal{A}$ and $\mathcal{B}$, respectively. Each of the following implies $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$.*

1. $\mathcal{A}_{\mathsf{FPE},P} \sqsubseteq_{\mathbf{F}} \mathcal{B}_{\mathsf{BPE},P'}$
2. $\mathcal{A}_{\mathsf{BPE},P} \sqsubseteq_{\mathbf{B}} \mathcal{B}_{\mathsf{FPE},P'}$ □

Theorem 4.4 (adequacy of FPE/BPE) *Let P and P' be arbitrary subsets of the state spaces of $\mathcal{A}$ and $\mathcal{B}$, respectively. We have:*

1. $\mathcal{A} \sqsubseteq_{\mathbf{F}} \mathcal{B} \;\Rightarrow\; \mathcal{A}_{\mathsf{FPE},P} \sqsubseteq_{\mathbf{F}} \mathcal{B}_{\mathsf{BPE},P'}$
2. $\mathcal{A} \sqsubseteq_{\mathbf{B}} \mathcal{B} \;\Rightarrow\; \mathcal{A}_{\mathsf{BPE},P} \sqsubseteq_{\mathbf{B}} \mathcal{B}_{\mathsf{FPE},P'}$ □

To prove these two theorems, we should first prove the following lemma.

Lemma 4.5 *For each subset P, we have the following.*
1. $\mathcal{A} \sqsubseteq_{\mathbf{B}} \mathcal{A}_{\mathsf{FPE},P}$ *2.* $\mathcal{A}_{\mathsf{FPE},P} \sqsubseteq_{\mathbf{F}} \mathcal{A}$ *3.* $\mathcal{A} \sqsubseteq_{\mathbf{F}} \mathcal{A}_{\mathsf{BPE},P}$ *4.* $\mathcal{A}_{\mathsf{BPE},P} \sqsubseteq_{\mathbf{B}} \mathcal{A}$

PROOF. 1. Let $\mathcal{A} = (Q, \Sigma, M, \alpha, \beta)$ and $\mathcal{A}_{\mathsf{FPE},P} = (Q', \Sigma, M', \alpha', \beta')$. We define $X \in S^{Q \times Q'}$ as follows:

$$\begin{array}{llll} X_{x,\checkmark} = \beta_x & (x \in P), & X_{x,(y,a)} = M(a)_{x,y} & (x \in P), \\ X_{x,x} = 1_{\mathcal{S}} & (x \notin P), & X_{u,v} = 0_{\mathcal{S}} & \text{(otherwise)} \end{array}$$

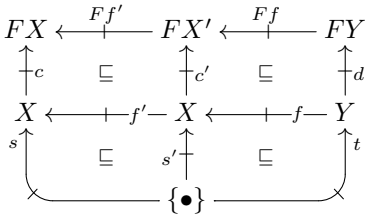
Then, this X is a backward simulation matrix from $\mathcal{A}$ to $\mathcal{A}_{\mathsf{FPE},P}$

The items 2., 3. and 4. are proved similarly. □

PROOF OF THEOREM 4.3. 1. From Proposition 4.5.1, $\mathcal{A} \sqsubseteq_{\mathbf{B}} \mathcal{A}_{\mathsf{FPE},P} \sqsubseteq_{\mathbf{F}} \mathcal{B}_{\mathsf{BPE},P'} \sqsubseteq_{\mathbf{B}} \mathcal{B}$. Hence from Corollary 3.6, $L(\mathcal{A}) \sqsubseteq L(\mathcal{A}_{\mathsf{FPE},P}) \sqsubseteq L(\mathcal{B}_{\mathsf{BPE},P'}) \sqsubseteq L(\mathcal{B})$ holds.

The item 2. is proved similarly. □

PROOF OF THEOREM 4.4. 1. From Proposition 4.5.2, $\mathcal{A}_{\mathsf{FPE},P} \sqsubseteq_{\mathbf{F}} \mathcal{A} \sqsubseteq_{\mathbf{F}} \mathcal{B} \sqsubseteq_{\mathbf{F}} \mathcal{B}_{\mathsf{BPE},P'}$. Because $\sqsubseteq_{\mathbf{F}}$ is transitive (see the diagram below where $F = 1 + \Sigma \times (\_)$), this implies $\mathcal{A}_{\mathsf{FPE},P} \sqsubseteq_{\mathbf{F}} \mathcal{B}_{\mathsf{BPE},P'}$.



The item 2. is proved similarly. □

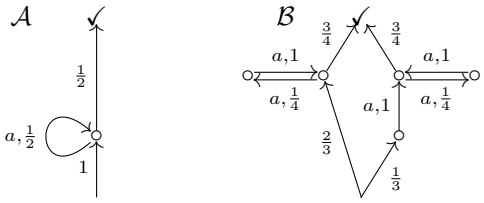
We also show that a bigger parameter P yields a greater number of simulations. In implementation, however, a bigger P generally gives us a bigger state space which slows down search for a simulation. Hence we are in a trade-off situation.

Proposition 4.6 (monotonicity) *Assume $P_1 \subseteq P_1'$ and $P_2 \subseteq P_2'$. We have:*
1. $\mathcal{A}_{\mathsf{FPE},P_1} \sqsubseteq_{\mathbf{F}} \mathcal{B}_{\mathsf{BPE},P_2} \Rightarrow \mathcal{A}_{\mathsf{FPE},P_1'} \sqsubseteq_{\mathbf{F}} \mathcal{B}_{\mathsf{BPE},P_2'}$,
2. $\mathcal{A}_{\mathsf{BPE},P_1} \sqsubseteq_{\mathbf{B}} \mathcal{B}_{\mathsf{FPE},P_2} \Rightarrow \mathcal{A}_{\mathsf{BPE},P_1'} \sqsubseteq_{\mathbf{B}} \mathcal{B}_{\mathsf{FPE},P_2'}$. □

For $\mathcal{S} = \mathcal{S}_{+,\times}$ or $\mathcal{S}_{\max,+}$, we can easily see that the complement problem of language inclusion between finite $\mathcal{S}$-weighted automata is semi-decidable. Since language inclusion itself is undecidable [5, 30], language inclusion is not even semidecidable. Because existence of a simulation matrix is decidable, it can be the case that however many times we apply FPE or BPE, simulation matrices do not exist while language inclusion holds. A concrete example is found in the following example.

Example 4.7 (limitation of FPE) The following $\mathcal{S}_{+,\times}$-weighted automata exhibit $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$, but a forward simulation does not exist no matter how many times FPE is applied

to $\mathcal{A}$.

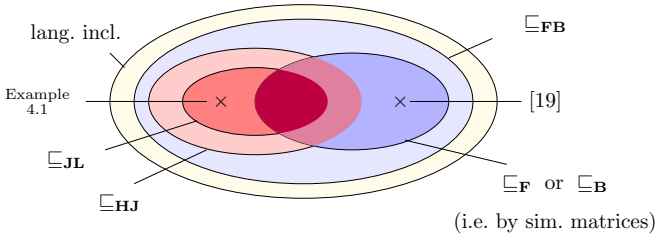


It is possible to describe FPE on the coalgebraic level of abstraction. Besides providing an insight into the essence of the construction, it also allows for the application of FPE to quantitative tree automata. In contrast, the definition of BPE seems to rely on the fact that we can "reverse" word automata, and hence is hard to generalize e.g. to tree automata. See Section 7.

5. Simulation Matrices for Probabilistic Systems by $\mathcal{S} = \mathcal{S}_{+,\times}$

In this section we focus on $\mathcal{S}_{+,\times}$-weighted automata which we identify as (purely) probabilistic automata (cf. Example 2.2). In Section 5.1 our method by simulation matrices is compared with other notions of probabilistic simulation; in Section 5.2 we discuss our implementation.

5.1. *Other Simulation Notions for Probabilistic Systems*



Various simulation notions have been introduced for probabilistic systems, either as a behavioral order by itself or as a proof method for language inclusion. Jonsson and Larsen's one [25] (denoted by $\sqsubseteq_{\mathbf{JL}}$) is well-known; it is shown in [19] to be a special case of Hughes and Jacobs' coalgebraic notion of simulation [23] ($\sqsubseteq_{\mathbf{HJ}}$), which in turn is a special case of forward-backward (Kleisli) simulation ($\sqsubseteq_{\mathbf{FB}}$, Definition 2.12). Comparison of all these notions (observed in [19]) is as depicted above; it follows from Theorem 2.13 that all these simulation notions are sound with respect to language inclusion.

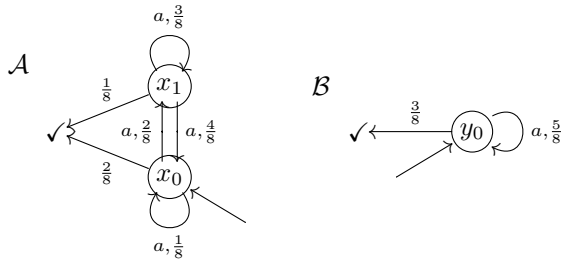
We note that language inclusion between finite $\mathcal{S}_{+,\times}$-weighted automata is known to be undecidable [5] while language equivalence can be determined in polynomial time [27]. The former result can account for the fact that there does not seem to be many proof methods for probabilistic/quantitative language inclusion. For example, *probabilistic simulation* in [2] is possibilistic simulation between systems with both probabilistic and nondeterministic choice and not a quantitative notion like in the current study.

We also note that given finite-state $\mathcal{S}_{+,\times}$-weighted automata $\mathcal{A}$ and $\mathcal{B}$, if $\mathcal{A} \sqsubseteq_{\mathbf{F}} \mathcal{B}$ or not is decidable: existence of a solution X of the linear constraints in Definition 3.4

can be reduced to linear programming (LP) problems, and the latter are known to be decidable. The same applies to $\sqsubseteq_{\mathbf{B}}$ too.

Probabilistic systems are commonly modeled using the monad $\mathcal{D}$ (see (2))—with an explicit *normalization* condition $\sum_x d(x) \leq 1$—instead of $\mathcal{M}_{\mathcal{S}_{+,\times}}$. However there is no need to impose normalization on simulations: sometimes only "non-normalized" simulation matrices are found and they are still sound. Here is such an example.

Example 5.1 The following $\mathcal{S}_{+,\times}$-weighted automata exhibit $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$. Neither forward nor backward Kleisli simulation (in the categorical sense of Definition 2.12) exists between them as long as we represent the automata $\mathcal{A}$ and $\mathcal{B}$ as $(\mathcal{D}, 1+\Sigma\times(\_))$-systems. However Kleisli simulations (forward and backward) are found once we represent $\mathcal{A}$ and $\mathcal{B}$ as $(\mathcal{M}_{\mathcal{S}}, 1+\Sigma\times(\_))$-systems.



Indeed, the only matrix X that is a forward simulation (Definition 3.4) is $X = \begin{pmatrix}1 & 1\end{pmatrix}$, and the only backward simulation matrix is $X = \begin{pmatrix}1\\2\end{pmatrix}$. Neither of these satisfies the normalization condition imposed on the subdistribution monad $\mathcal{D}$.

5.2. Implementation, Experiments and Discussions

Our implementation consists of two components: $+\times$-sim and $+\times$-PE.

- The program $+\times$-sim (implemented in C++) computes if a forward or backward simulation matrix X between $\mathcal{S}_{+,\times}$-weighted automata exists, and returns X if it does exist. It first combines the constraints in Definition 3.4 into a single linear inequality $A\boldsymbol{x} \leq \boldsymbol{b}$ and solves it with a linear programming solver *glpk* [17]. We note that the matrix A is sparse because of the way the different constraints in Definition 3.4 are combined. It has $n + anm + m$ rows, nm columns and at most $2nm + a(n^2m + nm^2)$ nonzero entries.
- The program $+\times$-PE (implemented in OCaml) takes an automaton $\mathcal{A}$ and $d \in \mathbb{N}$ as input, and returns $\mathcal{A}_{\mathsf{FPE},P}$ (or $\mathcal{A}_{\mathsf{BPE},P}$, by choice). Here P is chosen, by heuristics, to be $P = \{x \mid x \overbrace{\to \cdots \to}^{d} \checkmark\}$ (or $P = \{x \mid \bullet \overbrace{\to \cdots \to}^{d} x\}$, respectively).

The two programs are alternately applied to the given automaton, for $d = 1, 2, \ldots$, each time incrementing the parameter d for $+\times$-PE. The experiments were on a MacBook Pro laptop with a Core i5 processor (2.6 GHz, 2 cores) and 16 GB RAM.

param. G	param. S	$\mathcal{A}_\mathsf{P}$ #st.	$\mathcal{A}_\mathsf{P}$ #tr.	$\mathcal{A}_\mathsf{S}$ #st.	$\mathcal{A}_\mathsf{S}$ #tr.	$\lvert\Sigma\rvert$	direction, fwd./bwd.	time (sec)	space (GB)
2	8	578	1522	130	642	11	$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{F} \mathcal{A}_\mathsf{S}$	1.02	0.30
							$\mathcal{A}_\mathsf{P} \sqsupseteq_\mathbf{B} \mathcal{A}_\mathsf{S}$	1.00	0.30
2	10	1102	2982	202	1202	13	$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{F} \mathcal{A}_\mathsf{S}$	5.37	1.04
							$\mathcal{A}_\mathsf{P} \sqsupseteq_\mathbf{B} \mathcal{A}_\mathsf{S}$	5.30	1.05
2	12	1874	5162	290	2018	15	$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{F} \mathcal{A}_\mathsf{S}$	22.21	2.91
							$\mathcal{A}_\mathsf{P} \sqsupseteq_\mathbf{B} \mathcal{A}_\mathsf{S}$	21.99	2.94
2	14	2942	8206	394	3138	17	$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{F} \mathcal{A}_\mathsf{S}$	76.71	6.86
							$\mathcal{A}_\mathsf{P} \sqsupseteq_\mathbf{B} \mathcal{A}_\mathsf{S}$	76.03	6.94
2	16	4354	12258	514	4610	19	$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{F} \mathcal{A}_\mathsf{S}$	232.67	13.01
							$\mathcal{A}_\mathsf{P} \sqsupseteq_\mathbf{B} \mathcal{A}_\mathsf{S}$	231.33	13.20
3	8	1923	7107	243	2163	20	$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{F} \mathcal{A}_\mathsf{S}$	25.51	3.33
							$\mathcal{A}_\mathsf{P} \sqsupseteq_\mathbf{B} \mathcal{A}_\mathsf{S}$	25.35	3.36
3	10	3803	14323	383	4183	24	$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{F} \mathcal{A}_\mathsf{S}$	167.68	12.21
							$\mathcal{A}_\mathsf{P} \sqsupseteq_\mathbf{B} \mathcal{A}_\mathsf{S}$	166.86	12.27
4	6	1636	7468	196	1924	23	$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{F} \mathcal{A}_\mathsf{S}$	17.58	2.61
							$\mathcal{A}_\mathsf{P} \sqsupseteq_\mathbf{B} \mathcal{A}_\mathsf{S}$	17.34	2.64
4	8	4052	19076	356	4580	29	$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{F} \mathcal{A}_\mathsf{S}$	210.09	13.13
							$\mathcal{A}_\mathsf{P} \sqsupseteq_\mathbf{B} \mathcal{A}_\mathsf{S}$	212.58	13.12

Table 1: Results for the grades protocol [27]

5.2.1. Grades Protocol

The *grades protocol* is introduced in [27] and is used there as a benchmark: the protocol and its specification are expressed as probabilistic programs P and S; they are then translated into (purely) probabilistic automata $\mathcal{A}_\mathsf{P}$ and $\mathcal{A}_\mathsf{S}$ by a game semantics-based tool APEX [28]. By establishing $L(\mathcal{A}_\mathsf{P}) = L(\mathcal{A}_\mathsf{S})$, the protocol is shown to exhibit the same behaviors as the specification—hence is verified. The protocol has two parameters G and S.

In our experiment we proved $L(\mathcal{A}_\mathsf{P}) = L(\mathcal{A}_\mathsf{S})$ by establishing two-way language inclusion ($\sqsubseteq$ and $\sqsupseteq$). The results are shown in Table. 1. For all the choices of parameters G and S, our program $+\times$-sim was able to establish, without applying $+\times$-PE: $\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{F} \mathcal{A}_\mathsf{S}$ (but not $\sqsubseteq_\mathbf{B}$) for the $\sqsubseteq$ direction; and $\mathcal{A}_\mathsf{P} \sqsupseteq_\mathbf{B} \mathcal{A}_\mathsf{S}$ (but not $\sqsupseteq_\mathbf{F}$) for the $\sqsupseteq$ direction. In the table, #st. and #tr. denote the numbers of states and transitions, respectively, and $|\Sigma|$ is the size of the alphabet. All these numbers are determined by APEX.

The table indicates that space is a bigger problem for our approach than time. In [27] four algorithms for checking language *equivalence* between $\mathcal{S}_{+,\times}$-weighted automata are implemented and compared: two are deterministic [37, 12] and the other two are randomized [27]. These algorithms can process bigger problem instances (e.g. $G = 2, S = 100$ in ca. 10 sec) and, in comparison, the results in Table 1 are far from impressive. Note however that our algorithm is for language *inclusion*—an undecidable problem, unlike language *equivalence* that is in $\mathbf{P}$, see Section 5.1—and hence is more general.

5.2.2. Crowds Protocol

Our second experiment calls for checking language *inclusion*, making the algorithms studied in [27] unapplicable. We verified some instances of the *Crowds protocol* [35]

param. n	c	p_f	$\mathcal{A}_\mathsf{P}$ #st.	#tr.	$\mathcal{A}_\mathsf{S}$ #st.	#tr.	$\lvert\Sigma\rvert$	direction fwd./bwd.	time (sec)	space (GB)	d
5	1	$\frac{9}{10}$	7	44	7	56	18	$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{F} \mathcal{A}_\mathsf{S}$	0.10	0.02	2
								$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{B} \mathcal{A}_\mathsf{S}$	0.04	0.01	2
7	1	$\frac{3}{4}$	9	88	9	118	26	$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{F} \mathcal{A}_\mathsf{S}$	0.59	0.14	2
								$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{B} \mathcal{A}_\mathsf{S}$	0.05	0.01	2
10	2	$\frac{4}{5}$	12	224	12	280	54	$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{F} \mathcal{A}_\mathsf{S}$	42.63	3.86	2
								$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{B} \mathcal{A}_\mathsf{S}$	0.07	0.01	2
20	6	$\frac{4}{5}$	22	1514	22	1696	238	$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{F} \mathcal{A}_\mathsf{S}$	T/O		
								$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{B} \mathcal{A}_\mathsf{S}$	0.84	0.20	2
30	10	$\frac{4}{5}$	32	4732	32	5112	550	$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{F} \mathcal{A}_\mathsf{S}$	S/O		
								$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{B} \mathcal{A}_\mathsf{S}$	6.17	1.44	2
40	14	$\frac{4}{5}$	42	10742	42	11392	990	$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{F} \mathcal{A}_\mathsf{S}$	S/O		
								$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{B} \mathcal{A}_\mathsf{S}$	30.70	6.07	2
50	17	$\frac{4}{5}$	52	20504	52	21560	1494	$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{F} \mathcal{A}_\mathsf{S}$	S/O		
								$\mathcal{A}_\mathsf{P} \sqsubseteq_\mathbf{B} \mathcal{A}_\mathsf{S}$	102.89	13.64	2

Table 2: Results for the Crowds protocol

against a quantitative anonymity specification called *probable innocence* [29]. We used a general trace-based verification method in [22] for probable innocence: language inclusion $L(\mathcal{A}_\mathsf{P}) \sqsubseteq L(\mathcal{A}_\mathsf{S})$, from the model $\mathcal{A}_\mathsf{P}$ of a protocol in question to $\mathcal{A}_\mathsf{P}$'s suitable modification $\mathcal{A}_\mathsf{S}$, guarantees probable innocence.

The Crowds protocol has parameters n, c and p_f. In fact, for this specific protocol, a sufficient condition for probable innocence is known [35] (namely $n \geq \frac{p_f}{p_f - 1/2}(c+1)$); we used parameters that satisfy this condition. We implemented a small program that takes a choice of n, c, p_f and generates an automaton $\mathcal{A}_\mathsf{P}$; it is then passed to another program that generates $\mathcal{A}_\mathsf{S}$.

The results are in Table. 2. For each problem instance we tried both $\sqsubseteq_\mathbf{F}$ and $\sqsubseteq_\mathbf{B}$. The last column shows the final value of the parameter d for $+\times$-PE—i.e. how many times partial execution (Section 4) was applied.

The entry "S/O" designates that $+\times$-PE was killed because of stack overflow caused by an oversized automaton. "T/O" means that alternate application of $+\times$-sim and $+\times$-PE did not terminate within a time limit (one hour).

We observe that backward simulation matrices were much faster to be found than forward ones. This seems to result from the shapes of the automata for this specific problem; after all it is an advantage of our forward and backward approach that we can try two different directions and use the faster one. Space consumption seems again serious.

6. Simulation Matrices for $\mathcal{S}_{\mathbf{max},+}$-Weighted Automata

In this section we discuss $\mathcal{S}_{\mathrm{max},+}$-weighted automata, in which weights are understood as (best-case) profit or (worst-case) cost (see Example 2.2). Such automata are studied in [7] (called Sum-*automata* there). In fact we observe that their notion of simulation—formulated in game-theoretic terms and hence called *G-simulation* here—coincides with

the notion of forward simulation matrix. This observation—that is presented in Section 6.1—follows from the game-theoretic characterization in [1] of linear inequalities in $\mathcal{S}_{\max,+}$. In Section 6.2 our implementation is presented.

6.1. G-Simulation by Forward Simulation Matrices

In this section we restrict to finite-state automata. In this case we can dispose of the weight ∞, and have $[-\infty, \infty)$ as the domain of weights (see Example 2.2).

What we shall call *G-simulation* is introduced in [7], and its soundness with respect to weighted languages over *infinite-length words* $\Sigma^\omega \to [-\infty, \infty)$ is proved there. Here we adapt their definition to the current setting of finite-length words; the adaptation is concerned with termination ✓.

Definition 6.1 ($\sqsubseteq_{\mathbf{G}}$) Let $\mathcal{A} = (Q_\mathcal{A}, \Sigma, M_\mathcal{A}, \alpha_\mathcal{A}, \beta_\mathcal{A})$ and $\mathcal{B} = (Q_\mathcal{B}, \Sigma, M_\mathcal{B}, \alpha_\mathcal{B}, \beta_\mathcal{B})$ be finite-state $\mathcal{S}_{\max,+}$-weighted automata. A *finite simulation game* from $\mathcal{A}$ to $\mathcal{B}$ is played by two players called *Challenger* and *Simulator*: a *strategy for Challenger* is a pair $\big(\rho_1 : 1 \to Q_\mathcal{A},\ \tau_1 : (Q_\mathcal{A}\times Q_\mathcal{B})\times(\Sigma\times Q_\mathcal{A}\times Q_\mathcal{B})^* \to 1+\Sigma\times Q_\mathcal{A}\big)$ of functions; and a *strategy for Simulator* is a pair $\big(\rho_2 : Q_\mathcal{A} \to Q_\mathcal{B},\ \tau_2 : (Q_\mathcal{A}\times Q_\mathcal{B})\times(\Sigma\times Q_\mathcal{A}\times Q_\mathcal{B})^*\times\Sigma\times Q_\mathcal{A} \to Q_\mathcal{B}\big)$.

A pair $(p_0a_1\dots a_np_n, q_0a_1\dots a_nq_n)$ of runs on $\mathcal{A}$ and $\mathcal{B}$ is called the *outcome of strategies* (ρ_1,τ_1) *and* (ρ_2,τ_2) if:

- $\rho_1(\bullet) = p_0$ and $\rho_2(p_0) = q_0$ where $\bullet$ is the unique element of the domain of ρ_1.
- $\tau_1\big((p_0,q_0)(a_1,p_1,q_1)\dots(a_i,p_i,q_i)\big) = (a_{i+1},p_{i+1})$ for each $i \in [0, n-1]$.
- $\tau_2\big((p_0,q_0)(a_1,p_1,q_1)\dots(a_i,p_i,q_i),(a_{i+1},p_{i+1})\big) = q_{i+1}$ for each $i \in [0, n-1]$.
- $\tau_1((p_0,q_0)(a_1,p_1,q_1)\dots(a_n,p_n,q_n)) = \checkmark$.

A strategy (ρ_1,τ_1) for Challenger is *winning* if for any strategy (ρ_2,τ_2) for Simulator, their outcome (r_1,r_2) exists and it satisfies $L(\mathcal{A})(r_1) > L(\mathcal{B})(r_2)$. Here the weight $L(\mathcal{A})(r)$ of a run r is defined in the obvious way, exploiting the structure of the semiring $\mathcal{S}_{\max,+}$.

Finally, we write $\mathcal{A} \sqsubseteq_{\mathbf{G}} \mathcal{B}$ if there is no winning strategy for Challenger.

Theorem 6.2 *Let $\mathcal{A}$ and $\mathcal{B}$ be finite-state $\mathcal{S}_{\max,+}$-weighted automata. Assume that $\mathcal{A}$ has no trap states, that is, every state has a path to ✓ whose weight is not $-\infty$. Then, $\mathcal{A} \sqsubseteq_{\mathbf{F}} \mathcal{B}$ if and only if $\mathcal{A} \sqsubseteq_{\mathbf{G}} \mathcal{B}$.*

The extra assumption can be easily enforced by eliminating trap states through backward reachability check. This does not change the (finite) weighted language.

The proof of Theorem 6.2 is sketched as follows. We first reduce G-simulation (between the original automata $\mathcal{A},\mathcal{B}$ on finite words) to G-simulation $\sqsubseteq_{\mathbf{G}}^{\mathsf{Limavg}}$ between Limavg *automata*, the original setting in [7] with infinite words. This reduction (that is the first equivalence below) allows us to exploit the characterization in [7] of G-simulation in terms of a *mean payoff game*, yielding the second equivalence below.

$$\mathcal{A} \sqsubseteq_{\mathbf{G}} \mathcal{B} \iff \mathcal{A}^{\mathsf{Limavg}} \sqsubseteq_{\mathbf{G}}^{\mathsf{Limavg}} \mathcal{B}^{\mathsf{Limavg}} \iff \text{Max wins in } \mathcal{G}^{\mathbf{G}}_{\mathcal{A}^{\mathsf{Limavg}},\mathcal{B}^{\mathsf{Limavg}}} \ . \tag{8}$$

Conversely, starting from $\mathcal{A} \sqsubseteq_{\mathbf{F}} \mathcal{B}$, we use the fundamental result in [1] that characterizes feasibility of inequalities in $\mathcal{S}_{\max,+}$ in terms of mean payoff games.

$$\mathcal{A} \sqsubseteq_{\mathbf{F}} \mathcal{B} \iff \text{a certain linear inequality is feasible} \iff \text{Max wins in } \mathcal{G}^{\mathbf{F}}_{\mathcal{A},\mathcal{B}} \ . \tag{9}$$

Finally, we observe that the mean payoff games $\mathcal{G}^{\mathbf{G}}_{\mathcal{A}^{\mathsf{Limavg}},\mathcal{B}^{\mathsf{Limavg}}}$ in (8) and $\mathcal{G}^{\mathbf{F}}_{\mathcal{A},\mathcal{B}}$ in (9) are in fact the same, establishing $\mathcal{A} \sqsubseteq_{\mathbf{F}} \mathcal{B}$ if and only if $\mathcal{A} \sqsubseteq_{\mathbf{G}} \mathcal{B}$.

In what follows we introduce necessary definitions and lemmas, eventually leading to the proof of Theorem 6.2.

Definition 6.3 (Limavg automaton) A Limavg *automaton* $\mathcal{C} = (Q, \Sigma, M, q_0)$ consists of a finite state space Q, a finite alphabet Σ, transition matrices $M : \Sigma \to [-\infty, \infty)^{Q \times Q}$, and the initial state $q_0 \in Q$.

For an infinite word $w = a_0 a_1 \ldots \in \Sigma^\omega$, the automaton $\mathcal{C}$ assigns a value $L(\mathcal{C})(w)$ that is calculated by $L(\mathcal{C})(w) = \sup_{q_0 q_1 \ldots \in Q^\infty} \liminf_{N\to\infty} \frac{1}{N} \sum_{i=0}^{N} M(a_i)_{q_i, q_{i+1}}$.

Lemma 6.4 *Given an $\mathcal{S}_{\max,+}$-weighted automaton $\mathcal{C} = (Q, \Sigma, M, \alpha, \beta)$, we define a* Limavg *automaton $\mathcal{C}^{\mathsf{Limavg}} = (Q^{\mathsf{Limavg}}, \Sigma^{\mathsf{Limavg}}, M^{\mathsf{Limavg}}, \star)$ by*

$$\begin{aligned} Q^{\mathsf{Limavg}} &= Q + \{\star\} \ , \\ \Sigma^{\mathsf{Limavg}} &= \Sigma + \{?\} + \{!\} \ , \text{ and} \\ M^{\mathsf{Limavg}}(a)_{x,y} &= \begin{cases} M(a)_{x,y} & (a \in \Sigma, x, y \neq \star) \\ \alpha_y & (a = ?, x = \star) \\ \beta_x & (a = !, y = \star) \\ -\infty & (\textit{otherwise}) \ . \end{cases} \end{aligned}$$

Let $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, M_{\mathcal{A}}, \alpha_{\mathcal{A}}, \beta_{\mathcal{A}})$ and $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, M_{\mathcal{B}}, \alpha_{\mathcal{B}}, \beta_{\mathcal{B}})$ be $\mathcal{S}_{\max,+}$-weighted automata; assume further that $\mathcal{A}$ has no trap states. Then we have

$$\mathcal{A} \sqsubseteq_{\mathbf{G}} \mathcal{B} \iff \mathcal{A}^{\mathsf{Limavg}} \sqsubseteq^{\mathsf{Limavg}}_{\mathbf{G}} \mathcal{B}^{\mathsf{Limavg}} \ . \qquad \square$$

Intuitively, the automaton $\mathcal{C}^{\mathsf{Limavg}}$ is obtained from $\mathcal{C}$ by connecting the initial and final states of $\mathcal{C}$. More concretely, we add a new state $\star$ that represents both the initial and final state in $\mathcal{C}$, add transitions from $\star$ to states in $\mathcal{C}$ according to the initial vector α, and add transitions from states in $\mathcal{C}$ to $\star$ according to the final vector β.

The proof of the lemma is technical and deferred to Appendix Appendix A.1. The basic idea is as follows. A winning strategy for the (finite) game for $\mathcal{A} \sqsubseteq_{\mathbf{G}} \mathcal{B}$ yields that for the other (infinite) game, by repeating the strategy. The other direction is similar, except that trap states call for special care.

We shall now describe the second equivalence in (8). The notion of mean payoff game is from [13].

Definition 6.5 (mean payoff game) A *weighted bipartite graph* $\mathcal{G} = (Q_{\mathrm{Min}}, Q_{\mathrm{Max}}, q_I, E, \gamma)$ consists of a set Q_{Min} of states for Min, a set Q_{Max} of states for Max, the initial state $q_I \in Q_{\mathrm{Min}}$, a set $E = Q_{\mathrm{Min}} \times Q_{\mathrm{Max}} + Q_{\mathrm{Max}} \times Q_{\mathrm{Min}}$ of edges, and a weight function $\gamma : E \to \mathbb{R}$.

A *mean payoff game* is a game played by two players Min and Max on a weighted bipartite graph $\mathcal{G}$. A *strategy for* Min is a function $\tau_{\mathrm{Min}} : (Q_{\mathrm{Min}} \times Q_{\mathrm{Max}})^* \times Q_{\mathrm{Min}} \to Q_{\mathrm{Max}}$ and a *strategy for* Max is a function $\tau_{\mathrm{Max}} : (Q_{\mathrm{Min}} \times Q_{\mathrm{Max}})^+ \to Q_{\mathrm{Min}}$. An infinite run $p_0q_0p_1q_1\ldots$ on $\mathcal{G}$ (here $p_i \in Q_{\mathrm{Min}}$ and $q_i \in Q_{\mathrm{Max}}$) is the *outcome* of strategies τ_{Min} and τ_{Max} if $p_0 = q_I$ and for any $i \geq 1$, $\tau_{\mathrm{Min}}(p_0q_0\ldots q_{i-1}p_i) = q_i$ and $\tau_{\mathrm{Min}}(p_0q_0\ldots p_iq_i) = p_{i+1}$. A strategy τ_{Max} for Max is *winning* if for any strategy τ_{Max} for Min, their outcome $r_0r_1r_2r_3\ldots$ satisfies $\liminf_{N\to\infty} \frac{1}{N}\sum_{i=0}^{N}\big(\gamma(r_i, r_{i+1})\big) \geq 0$.

Lemma 6.6 ([7]) *Let $\mathcal{E} = (Q_{\mathcal{E}}, \Sigma, M_{\mathcal{E}}, q^0_{\mathcal{E}})$ and $\mathcal{F} = (Q_{\mathcal{F}}, \Sigma, M_{\mathcal{F}}, q^0_{\mathcal{F}})$ be* Limavg *automata. We have $\mathcal{E} \sqsubseteq_{\mathbf{G}} \mathcal{F}$ if and only if Max wins in the mean payoff game $\mathcal{G}_{\mathcal{E},\mathcal{F}}$, where the game $\mathcal{G}_{\mathcal{E},\mathcal{F}} = (Q_{\mathrm{Min}}, Q_{\mathrm{Max}}, q_I, E, \gamma)$ is defined by*

$$Q_{\mathrm{Min}} = Q_{\mathcal{E}} \times Q_{\mathcal{F}} \ , \qquad Q_{\mathrm{Max}} = Q_{\mathcal{E}} \times Q_{\mathcal{F}} \times \Sigma \ , \qquad q_I = (q^0_{\mathcal{E}}, q^0_{\mathcal{F}}) \ ,$$
$$E = \big\{((p,q),(p',q,a)) \,\big|\, M_{\mathcal{E}}(a)_{p,p'} \neq -\infty\big\} + \big\{((p,q,a),(p,q')) \,\big|\, M_{\mathcal{F}}(a)_{q,q'} \neq -\infty\big\} \ ,$$
$$\gamma((p,q),(p',q,a)) = -M_{\mathcal{E}}(a)_{p,p'} \quad \textit{and} \quad \gamma((p,q,a),(p,q')) = M_{\mathcal{F}}(a)_{q,q'} \ . \qquad \square$$

We turn to the equivalences in (9). The following proposition is interesting for its own sake, characterizing $\sqsubseteq_{\mathbf{F}}$ for $\mathcal{S}_{\mathrm{max},+}$-weighted automata in terms of mean payoff games. We crucially rely on a result in [1].

Proposition 6.7 *For a pair of $\mathcal{S}_{\mathrm{max},+}$-weighted automata $\mathcal{A}$ and $\mathcal{B}$, there exists a mean payoff game $\mathcal{G}^{\mathbf{F}}_{\mathcal{A},\mathcal{B}}$ such that: $\mathcal{A} \sqsubseteq_{\mathbf{F}} \mathcal{B}$ if and only if Max wins in $\mathcal{G}^{\mathbf{F}}_{\mathcal{A},\mathcal{B}}$.*

Proof. Let $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, M_{\mathcal{A}}, \alpha_{\mathcal{A}}, \beta_{\mathcal{A}})$ and $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, M_{\mathcal{B}}, \alpha_{\mathcal{B}}, \beta_{\mathcal{B}})$. By Definition 3.4, a forward simulation matrix from $\mathcal{A}$ to $\mathcal{B}$ is a matrix $X \in \mathcal{S}_{\mathrm{max},+}^{Q_{\mathcal{B}} \times Q_{\mathcal{A}}}$ that satisfies

$$\alpha_{\mathcal{A}} \leq \alpha_{\mathcal{B}} X \ \wedge \ \forall a \in \Sigma.\, X M_{\mathcal{A}}(a) \leq M_{\mathcal{B}}(a) X \ \wedge \ X\beta_{\mathcal{A}} \leq \beta_{\mathcal{B}} \,. \tag{10}$$

The result in [1] reduces:

- existence of a nontrivial (i.e. not $-\infty$) solution of a linear inequality $A\mathbf{x} \leq B\mathbf{x}$, where A, B are matrices over $\mathcal{S}_{\mathrm{max},+}$ and $\mathbf{x}$ is a column vector of variables, to
- a mean payoff game.

We therefore need to transform (10) into the format $A\mathbf{x} \leq B\mathbf{x}$. In particular, $\alpha_{\mathcal{A}}$ and $\beta_{\mathcal{B}}$ on both ends of (10) should be taken care of.

We shall prove that: there exists a matrix X that satisfies (10), if and only if, there exist $x_{\star,\star} \in \mathcal{S}_{\mathrm{max},+}$ and $X' \in \mathcal{S}_{\mathrm{max},+}^{Q_{\mathcal{B}} \times Q_{\mathcal{A}}}$ that satisfy $x_{\star,\star} \neq -\infty$ and

$$x_{\star,\star}\alpha_{\mathcal{A}} \leq \alpha_{\mathcal{B}} X' \ \wedge \ \forall a \in \Sigma.\, X' M_{\mathcal{A}}(a) \leq M_{\mathcal{B}}(a) X' \ \wedge \ X'\beta_{\mathcal{A}} \leq x_{\star,\star}\beta_{\mathcal{B}} \,. \tag{11}$$

Note here that $x_{\star,\star}\alpha_{\mathcal{A}}$ denotes the vector $\alpha_{\mathcal{A}}$ multiplied by the scalar $x_{\star,\star}$. Here "multiplication" is by the semiring multiplication of $\mathcal{S}_{\mathrm{max},+}$, that is, addition of real numbers.

Indeed, if X satisfying (10) exists, then $x_{\star,\star} = 0$ and $X' = X$ satisfy (11). Conversely, if $x_{\star,\star} \in \mathcal{S}_{\mathrm{max},+}$ (where $x_{\star,\star} \neq -\infty$) and $X' \in \mathcal{S}_{\mathrm{max},+}^{Q_{\mathcal{B}} \times Q_{\mathcal{A}}}$ satisfy (11), then $X \in \mathcal{S}_{\mathrm{max},+}^{Q_{\mathcal{B}} \times Q_{\mathcal{A}}}$ defined by $X_{q,p} = X'_{q,p} - x_{\star,\star}$ satisfies (10). Here $-x_{\star,\star}$ denotes subtraction of the real number $x_{\star,\star}$. It is well-defined and constitutes the inverse of (semiring-)multiplication by $x_{\star,\star}$, since $x_{\star,\star} \neq -\infty$.

It is straightforward to translate (11) into the format $A\mathbf{x} \leq B\mathbf{x}$. Then applying the result [1] yields the following mean payoff game, Max's winning in which is equivalent to the feasibility of (11), hence to that of (10).

The game is played on a graph $\mathcal{G}^{\mathbf{F}}_{\mathcal{A},\mathcal{B}} = (Q^{\mathbf{F}}_{\mathrm{Min}}, Q^{\mathbf{F}}_{\mathrm{Max}}, q^{\mathbf{F}}_I, E^{\mathbf{F}}, \gamma^{\mathbf{F}})$, where

$$
\begin{aligned}
Q^{\mathbf{F}}_{\mathrm{Min}} &= \{x_{\star,\star}\} + \{x_{q,p} \mid q \in Q_{\mathcal{B}}, p \in Q_{\mathcal{A}}\}\,,\\
Q^{\mathbf{F}}_{\mathrm{Max}} &= Q_{\mathcal{A}} + \Sigma \times Q_{\mathcal{B}} \times Q_{\mathcal{A}} + Q_{\mathcal{B}}\,,\\
q^{\mathbf{F}}_I &= x_{\star,\star}\\
E^{\mathbf{F}} &= E^{\mathbf{F}}_1 + E^{\mathbf{F}}_2 \text{ where } E^{\mathbf{F}}_1 \subseteq Q^{\mathbf{F}}_{\mathrm{Min}} \times Q^{\mathbf{F}}_{\mathrm{Max}}, E^{\mathbf{F}}_2 \subseteq Q^{\mathbf{F}}_{\mathrm{Max}} \times Q^{\mathbf{F}}_{\mathrm{Min}} \text{ and}\\
&\quad E^{\mathbf{F}}_1 = \{(x_{\star,\star}, p) \mid (\alpha_{\mathcal{A}})_p \neq -\infty\}\} + \{(x_{q,p}, (a, q, p')) \mid M_{\mathcal{A}}(a)_{p,p'} \neq -\infty\}\\
&\qquad + \{(x_{q,p}, q) \mid (\beta_{\mathcal{A}})_p \neq -\infty\}\\
&\quad E^{\mathbf{F}}_2 = \{(p, x_{q,p}) \mid (\alpha_{\mathcal{B}})_q \neq -\infty\} + \{((a, q, p), x_{q',p}) \mid M_{\mathcal{B}}(a)_{q,q'} \neq -\infty\}\\
&\qquad + \{(q, x_{\star,\star}) \mid (\beta_{\mathcal{B}})_q \neq -\infty\}\,,
\end{aligned}
$$

$$
\begin{aligned}
&\gamma^{\mathbf{F}}(x_{\star,\star}, p) = -(\alpha_{\mathcal{A}})_p, \quad \gamma^{\mathbf{F}}(x_{q,p}, (a, q, p')) = -M_{\mathcal{A}}(a)_{p,p'}, \quad \gamma^{\mathbf{F}}(x_{q,p}, q) = -(\beta_{\mathcal{A}})_p\\
&\gamma^{\mathbf{F}}(p, x_{q,p}) = (\alpha_{\mathcal{B}})_q, \quad \gamma^{\mathbf{F}}((a, q, p), x_{q',p}) = M_{\mathcal{B}}(a)_{q,q'}, \quad \gamma^{\mathbf{F}}(q, x_{\star,\star}) = (\beta_{\mathcal{B}})_q\,.
\end{aligned}
$$

This concludes the proof. □

Finally, we bridge the rightmost conditions in (8–9) and prove Theorem 6.2.

Proof of Theorem 6.2. Let $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, M_{\mathcal{A}}, \alpha_{\mathcal{A}}, \beta_{\mathcal{A}})$ and $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, M_{\mathcal{B}}, \alpha_{\mathcal{B}}, \beta_{\mathcal{B}})$.

By Lemma 6.4 and Lemma 6.6, $\mathcal{A} \sqsubseteq_{\mathbf{G}} \mathcal{B}$ is equivalent to existence of a winning strategy for Max in the mean payoff game played on a graph $\mathcal{G}^{\mathbf{G}}_{\mathcal{A}^{\mathsf{Limavg}},\mathcal{B}^{\mathsf{Limavg}}} = (Q^{\mathbf{G}}_{\mathrm{Min}}, Q^{\mathbf{G}}_{\mathrm{Max}}, q^{\mathbf{G}}_I, E^{\mathbf{G}}, \gamma^{\mathbf{G}})$. It is defined by

$$
\begin{aligned}
Q^{\mathbf{G}}_{\mathrm{Min}} &= (\{\star\} + Q_{\mathcal{A}}) \times (\{\star\} + Q_{\mathcal{B}})\,,\\
Q^{\mathbf{G}}_{\mathrm{Max}} &= (\{\star\} + Q_{\mathcal{A}}) \times (\{\star\} + Q_{\mathcal{B}}) \times (\{?\} + \{!\} + \Sigma)\,,\\
q^{\mathbf{G}}_I &= (\star, \star)\,,\\
E &= E^{\mathbf{G}}_1 + E^{\mathbf{G}}_2 \text{ s.t. } E^{\mathbf{G}}_1 \subseteq Q^{\mathbf{G}}_{\mathrm{Min}} \times Q^{\mathbf{G}}_{\mathrm{Max}}, E^{\mathbf{G}}_2 \subseteq Q^{\mathbf{G}}_{\mathrm{Max}} \times Q^{\mathbf{G}}_{\mathrm{Min}} \text{ and}\\
&\quad E^{\mathbf{G}}_1 = \{((\star, \star), (p, \star, ?)) \mid (\alpha_{\mathcal{A}})_p \neq -\infty\}\} + \{((p, q), (p', q, a)) \mid M_{\mathcal{A}}(a)_{p,p'} \neq -\infty\}\\
&\qquad + \{((p, q), (\star, q, !)) \mid (\beta_{\mathcal{A}})_p \neq -\infty\}\\
&\quad E^{\mathbf{G}}_2 = \{((p, \star, ?), (p, q)) \mid (\alpha_{\mathcal{B}})_q \neq -\infty\} + \{((p, q, a), (p, q')) \mid M_{\mathcal{B}}(a)_{q,q'} \neq -\infty\}\\
&\qquad + \{((\star, q, !), (\star, \star)) \mid (\beta_{\mathcal{B}})_q \neq -\infty\}\,,
\end{aligned}
$$

$$
\begin{aligned}
&\gamma^{\mathbf{G}}((\star, \star), (p, \star, ?)) = -(\alpha_{\mathcal{A}})_p, \gamma^{\mathbf{G}}((p, q), (p', q, a)) = -M_{\mathcal{A}}(a)_{p,p'}, \gamma^{\mathbf{G}}((p, q), (\star, q, !)) = -(\beta_{\mathcal{A}})_p\\
&\gamma^{\mathbf{G}}((p, \star, ?), (p, q)) = (\alpha_{\mathcal{B}})_q, \quad \gamma^{\mathbf{G}}((p, q, a), (p, q')) = M_{\mathcal{B}}(a)_{q,q'}, \quad \gamma^{\mathbf{G}}((\star, q, !), (\star, \star)) = (\beta_{\mathcal{B}})_q\,.
\end{aligned}
$$

It is not hard to see that the last graph $\mathcal{G}^{\mathbf{G}}_{\mathcal{A}^{\mathsf{Limavg}},\mathcal{B}^{\mathsf{Limavg}}}$ is equivalent to $\mathcal{G}^{\mathbf{F}}_{\mathcal{A},\mathcal{B}}$ in Proposition 6.7: the former has extra states but they are all unreachable. □

Remark 6.8 (complexity) The decision problem of mean payoff games is known to be in **NP** ∩ **co-NP** [39]; it has a pseudo polynomial-time algorithm, too [39]. By [1]

this problem is equivalent to the feasibility problem of linear inequalities in $\mathcal{S}_{\max,+}$. For the latter problem, the algorithm proposed in [6] (for solving linear equalities) can be utilized; this algorithm is shown in [4] to be superpolynomial. These results give upper bounds for the complexity of $\sqsubseteq_{\mathbf{F}}$ and $\sqsubseteq_{\mathbf{G}}$, by Theorem 6.2 and the subsequent lemmas.

Similarly to $\mathcal{S}_{+,\times}$-weighted automata, language inclusion between $\mathcal{S}_{\max,+}$-weighted automata is known to be undecidable [30]. We note that, by Theorem 6.2, applying FPE or BPE (Section 4) increases the likelihood of $\sqsubseteq_{\mathbf{G}}$ (in the sense of Theorem 4.4). We additionally note that, by exploiting symmetry of forward and backward simulation matrices (Remark 3.7), we could define "backward G-simulation" as a variation of Definition 6.1.

6.2. Implementation, Experiments and Discussions

We implemented two programs: max+-sim and max+-PE.

- We have seen that finding simulation matrices can be reduced to some problems that have known algorithms. Since we did not find actual software available, we implemented (in C++) the algorithm in [6] (for solving $\mathcal{S}_{\max,+}$-linear equalities) as part of the program max+-sim. It transforms the constraints in Definition 3.4 into an inequality $A\mathbf{x} \leq B\mathbf{x}$, which in turn is made into a linear equality $A'\mathbf{x}' = B'\mathbf{x}'$ by adding slack variables. The last equality is solved by the algorithm in [6].
- max+-PE is as in Section 5. It simply uses the whole state space as the parameter P.

Experiments were done on a MacBook Pro laptop with a Core i5 processor (2.6 GHz, 2 cores) and 16 GB RAM. There we faced a difficulty of finding a benchmark example: although small examples are not hard to come up with by human efforts, we could not find a good example that has parameters (like G, S in Table 1) and allows for experiments with problem instances of a varying size.

We therefore ran max+-sim for:

- the problem if $\mathcal{A} \sqsubseteq_{\mathbf{F}} \mathcal{A}$ for randomly generated $\mathcal{A}$, and
- the problem if $\mathcal{A} \sqsubseteq_{\mathbf{F}} \mathcal{B}$ for randomly generated $\mathcal{A}, \mathcal{B}$,

and measured time and memory consumption. Although the answers are known by construction (positive for the former, and almost surely negative for the latter), actual calculation via linear inequality constraints gives us an idea about resource consumption of our simulation-based method when it is applied to real-world problems.

The outcome is as shown in Figure 3. The parameter p is the probability with which an a-transition exists given a source state, a target state, and a character $a \in \Sigma$. Its weight is chosen from $\{0, 1, \ldots, 16\}$ subject to the uniform distribution. "Same" means checking $\mathcal{A} \sqsubseteq_{\mathbf{F}} \mathcal{A}$ and "difference" means checking $\mathcal{A} \sqsubseteq_{\mathbf{F}} \mathcal{B}$ (see above). The two problem settings resulted in comparable performance.

We observe that space consumption is not so big a problem as in the $\mathcal{S}_{+,\times}$ case (Section 5.2). Somehow unexpectedly, there is no big performance gap between the sparse case ($p = 0.1$) and the dense case ($p = 0.9$); in fact the sparse case consumes slightly more memory. Consumption of both time and space grows faster than linearly, which poses a question about the scalability of our approach. That said, our current

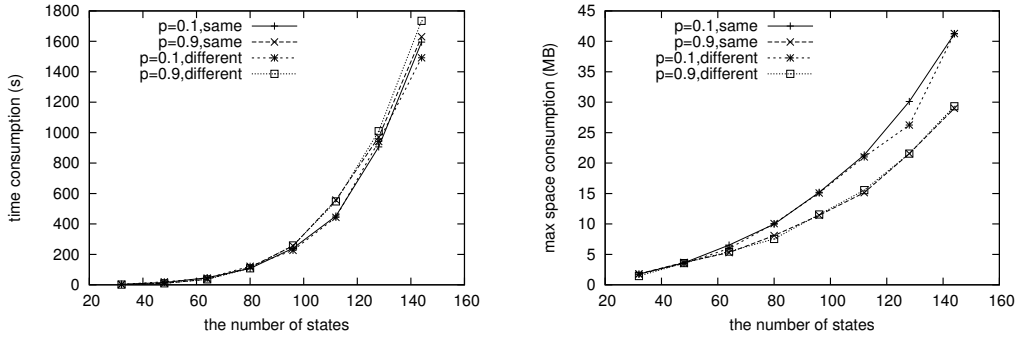


Figure 3: Time and max space consumption for max+-sim

implementation of the algorithm in [6] leaves a lot of room for further optimization: one possibility is use of dynamic programming (DP). After all, it is an advantage of our approach that a simulation problem is reduced to linear inequality constraints, a subject of extensive research efforts (cf. Section 5.1 and Section 6.1).

7. Matrix Simulation for Polynomial Functors

In the previous sections, we concentrated our attention on matrix simulations for weighted (word) automata. They are special cases of (T, F)-systems where T is a multiset monad over some semiring and $F = 1+\Sigma\times(\_)$. However, according to the general theory developed in [21, 18], Kleisli simulation can be defined and its soundness can be proved for more general (T, F)-systems.

In this section, we generalize the functor F from $1 + \Sigma \times (\_)$ (that we have been using) to an arbitrary polynomial functor:

$$F \;::=\; (\_) \mid \Sigma \mid F_1 \times F_2 \mid \coprod_{i\in I} F_i \,. \tag{12}$$

Such a generalized (T, F)-system—where T is again a multiset monad—represents a system called a *weighted tree automaton*. Here, the choice of F determines the shape of trees to which the automaton assigns a weight.

This section is organized as follows. In Section 7.1 we build on [18] and introduce the notion of forward partial execution (FPE) on the coalgebraic level of abstraction. We also prove its correctness (soundness and adequacy); the overall coalgebraic theory (i.e. the one in [18] augmented with FPE) generalizes the one in Section 4 for weighted (word) automata. The abstract theory thus obtained is applied in Section 7.2 to weighted tree automata—i.e. (T, F)-systems with T being a multiset monad and F being a polynomial functor. Much like for word automata, Kleisli simulations for tree automata are represented by matrices, subject however to *nonlinear* inequality constraints. Finally in Section 7.3 our proof-of-concept implementation is presented.

7.1. Forward Partial Execution, Categorically

Definition 7.1 (FPE, categorically) *Forward partial execution* (FPE) for (T, F)-systems is a transformation that takes: a (T, F)-system $\mathcal{X} = (X,\ s : \{\bullet\} \nrightarrow X,\ c : X \nrightarrow FX)$ and a parameter $X_1 \subseteq X$ as input; and returns a (T, F)-system $\mathcal{X}_{\mathsf{FPE},X_1}$.

The outcome $\mathcal{X}_{\mathsf{FPE},X_1}$ is defined as follows. Let c_1 and c_2 be the domain restrictions of c to X_1 and X_2, respectively, via the coprojections $\kappa_i : X_i \nrightarrow X$. That is explicitly:

$$c_1 \;=\; c \odot \kappa_1 \;:\; X_1 \nrightarrow FX \quad \text{and} \quad c_2 \;=\; c \odot \kappa_2 \;:\; X_2 \nrightarrow FX \;\;,$$

where $\odot$ denotes composition of Kleisli arrows (Definition 2.7). We define $X_2 = X \setminus X_1$ (hence $X = X_1 + X_2$); the system $\mathcal{X}_{\mathsf{FPE},X_1}$ is now given by

$$\mathcal{X}_{\mathsf{FPE},X_1} \;=\; \left(\begin{array}{l} F(X) + X_2 \,, \\ \{\bullet\} \xrightarrow{\ s\ } X_1 + X_2 \xrightarrow{\ c_1 + \mathrm{id}\ } F(X) + X_2 \,, \\ F(X) + X_2 \xrightarrow{\ [\mathrm{id}, c_2]\ } F(X_1 + X_2) \xrightarrow{\ \overline{F}(c_1 + \mathrm{id})\ } F\big(F(X) + X_2\big) \end{array} \right) .$$

Here $\overline{F}\colon \mathcal{K}\ell(T) \to \mathcal{K}\ell(T)$ is the canonical lifting of $F : \mathbf{Sets} \to \mathbf{Sets}$ (see [24] for a concrete definition).

For the last categorical generalization of FPE, we shall establish its correctness—soundness, adequacy and monotonicity—much like in Section 4.3.

Theorem 7.2 (soundness of categorical FPE) *Let X_1 and Y_1 be arbitrary subsets of the state spaces of $\mathcal{X}$ and $\mathcal{Y}$, respectively. Each of the following implies $\mathsf{tr}(\mathcal{X}) \sqsubseteq \mathsf{tr}(\mathcal{Y})$.*

1. $\mathcal{X}_{\mathsf{FPE},X_1} \sqsubseteq_{\mathbf{F}} \mathcal{Y}$
2. $\mathcal{X} \sqsubseteq_{\mathbf{B}} \mathcal{Y}_{\mathsf{FPE},Y_1}$ □

Theorem 7.3 (adequacy of categorical FPE) *Let X_1 and Y_1 be arbitrary subsets of the state spaces of $\mathcal{X}$ and $\mathcal{Y}$, respectively. We have:*

1. $\mathcal{X} \sqsubseteq_{\mathbf{F}} \mathcal{Y} \;\Rightarrow\; \mathcal{X}_{\mathsf{FPE},X_1} \sqsubseteq_{\mathbf{F}} \mathcal{Y}$
2. $\mathcal{X} \sqsubseteq_{\mathbf{B}} \mathcal{Y} \;\Rightarrow\; \mathcal{X} \sqsubseteq_{\mathbf{B}} \mathcal{Y}_{\mathsf{FPE},Y_1}$ □

The last two theorems are immediate consequences of the following lemma. The last is a categorical generalization of Lemma 4.5.

Lemma 7.4 *For each subset X_1 of the state space of $\mathcal{X}$, we have:*

1. $\mathcal{X} \sqsubseteq_{\mathbf{B}} \mathcal{X}_{\mathsf{FPE},X_1}$
2. $\mathcal{X}_{\mathsf{FPE},X_1} \sqsubseteq_{\mathbf{F}} \mathcal{X}$

Proof. 1. We define $g : X_1 + X_2 \nrightarrow F(X) + X_2$ by $g = c_1 + \mathrm{id}$. Then we have

$$\begin{aligned} \overline{F}(g) \odot c &= \overline{F}(c_1 + \mathrm{id}) \odot [c_1, c_2] \\ &= \overline{F}(c_1 + \mathrm{id}) \odot [\mathrm{id}, c_2] \odot (c_1 + \mathrm{id}) \\ &= \big(\overline{F}(c_1 + \mathrm{id}) \odot [\mathrm{id}, c_2]\big) \odot g \;\;, \\ g \odot s &= (c_1 + \mathrm{id}) \odot s \;. \end{aligned}$$

Note here that $\overline{F}(c_1 + \mathrm{id}) \odot [\mathrm{id}, c_2]$ is the dynamics of the system $\mathcal{X}_{\mathsf{FPE},X_1}$. The above equalities witness that g indeed satisfies the inequalities required in the definition of backward Kleisli simulation, a generalization of Definition 2.12 that is found in [18]. Hence g is a backward simulation from $\mathcal{X}$ to $\mathcal{X}_{\mathsf{FPE}}$.

The item 2. is proved similarly: the same g that we used in the proof of the item 1. is shown to be a forward simulation from $\mathcal{X}_{\mathsf{FPE}}$ to $\mathcal{X}$. □

Finally, we present a monotonicity result. It generalizes Proposition 4.6.

Proposition 7.5 (monotonicity of categorical FPE) *Assume $X_1 \subseteq X_1'$ and $X_2 \subseteq X_2'$. We have:*

1. $\mathcal{X}_{\mathsf{FPE},X_1} \sqsubseteq_{\mathbf{F}} \mathcal{Y} \;\Rightarrow\; \mathcal{X}_{\mathsf{FPE},X_1'} \sqsubseteq_{\mathbf{F}} \mathcal{Y}$
2. $\mathcal{X} \sqsubseteq_{\mathbf{B}} \mathcal{Y}_{\mathsf{FPE},Y_2} \;\Rightarrow\; \mathcal{X} \sqsubseteq_{\mathbf{B}} \mathcal{Y}_{\mathsf{FPE},Y_2'}$ □

Categorical formalization of BPE is still open—it seems that BPE in Section 4 exists somewhat coincidentally, for the specific functor $F = 1 + \Sigma \times (\_)$ for which an opposite automaton is canonically defined (cf. Remark 3.7).

7.2. Matrix Simulations for Weighted Tree Automata

Here we exploit the general theory we have just obtained (by augmenting [18] with FPE). We shall apply it to a class of systems that is more general than what we have been dealing with in the previous sections (namely weighted (word) automata). Specifically, we use the same monads for T but allow arbitrary polynomial functors for F. Such systems are naturally identified with *weighted tree automata*, where a finite-depth tree, instead of a finite word, gets a weight assigned.

We first define the notion of tree.

Definition 7.6 A *ranked alphabet* is a family $\Sigma = (\Sigma_n)_{n\in\mathbb{N}}$ of countable sets that are indexed by natural numbers called *arities*.

The set $\mathrm{Tree}(\Sigma)$ of (finite-depth) *trees* over a ranked alphabet Σ is defined in the obvious way. Concretely, $\mathrm{Tree}(\Sigma)$ is the smallest set such that: for each $a \in \Sigma_n$, $t_0, t_1, \ldots, t_{n-1} \in \mathrm{Tree}(\Sigma)$ implies $a(t_0, t_1, \ldots, t_{n-1}) \in \mathrm{Tree}(\Sigma)$.

We introduce weighted tree automata, firstly in concrete terms.

Definition 7.7 ($\mathcal{S}$-weighted tree automaton, weighted tree language) Let $\mathcal{S} = (S, +_{\mathcal{S}}, 0_{\mathcal{S}}, \times_{\mathcal{S}}, 1_{\mathcal{S}}, \sqsubseteq)$ be a commutative cppo-semiring. An *$\mathcal{S}$-weighted tree automaton* is a quadruple $\mathcal{A} = (Q, \Sigma, M, \alpha)$ consisting of a countable state space Q, a ranked alphabet $\Sigma = (\Sigma_n)_{n\in\mathbb{N}}$, transition matrices $M(a) \in \mathcal{S}^{Q\times Q^n}$ for each $a \in \Sigma_n$, and the initial row vector $\alpha \in \mathcal{S}^Q$.

An $\mathcal{S}$-weighted tree automaton $\mathcal{A} = (Q, \Sigma, M, \alpha)$ yields a *weighted tree language* $L(\mathcal{A}) : \mathrm{Tree}(\Sigma) \to S$. It is defined by $L(\mathcal{A})(t) = \alpha \cdot \Phi(t)$—the product of a row vector α and a column vector $\Phi(t)$—where $\Phi : \mathrm{Tree}(\Sigma) \to \mathcal{S}^Q$ is defined as follows, by induction on the depth of trees.

$$\Phi\big(a(t_0, t_1, \ldots, t_{n-1})\big) \;=\; M(a)\big(\,\Phi(t_0) \otimes \Phi(t_1) \otimes \cdots \otimes \Phi(t_{n-1})\,\big) \qquad \text{for each } a \in \Sigma_n.$$

The final column vector β in Definition 2.3 do not appear here; transition matrices $M(a)$ for $a \in \Sigma_0$ play the corresponding role.

Language inclusion between two $\mathcal{S}$-weighted tree automata is defined similarly to the case with $\mathcal{S}$-weighted automata.

Definition 7.8 (language inclusion) Let $\mathcal{A}$ and $\mathcal{B}$ be $\mathcal{S}$-weighted tree automata. We say the language of $\mathcal{A}$ is *included* in the language of $\mathcal{B}$ and write $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$ if, for all $t \in \mathrm{Tree}(\Sigma)$, $L(\mathcal{A})(t) \sqsubseteq L(\mathcal{B})(t)$.

Similarly to $\mathcal{S}$-weighted (word) automata (see Proposition 2.6), $\mathcal{S}$-weighted tree automata are instances of (T, F)-systems.

Definition 7.9 (the functor F_Σ) It is standard that a ranked alphabet Σ gives rise to a polynomial functor. It is given as follows and is denoted by F_Σ.

$$F_\Sigma \;=\; \coprod_{n\in\mathbb{N}} \Sigma_n \times (\_)^n \quad : \mathbf{Sets} \longrightarrow \mathbf{Sets} \;.$$

Proposition 7.10 (weighted tree automata as (T, F)-systems) *Let $\mathcal{S}$ be a commutative cppo-semiring. An $\mathcal{S}$-weighted tree automaton $\mathcal{A} = (Q, (\Sigma_n)_{n\in\mathbb{N}}, M, \alpha)$ gives rise to an $(\mathcal{M}_\mathcal{S}, F_\Sigma)$-system $\mathcal{X}_\mathcal{A} = (Q, s_\mathcal{A}, c_\mathcal{A})$ defined as follows. The function $s_\mathcal{A} \colon \{\bullet\} \to \mathcal{M}_\mathcal{S} Q$ is given by $s_\mathcal{A}(\bullet)(x) = \alpha_x$; and $c_\mathcal{A} \colon Q \to \mathcal{M}_\mathcal{S}(F_\Sigma Q)$ is given by*

$$c_\mathcal{A}(x)\big(a, (y_0, y_1, \ldots, y_{n-1})\big) = M(a)_{x,(y_0,y_1,\ldots,y_{n-1})}$$

where $a \in \Sigma_n$. □

The last identification allows us to apply the general results in [18, 21] to weighted tree automata. One of the results characterizes *coalgebraic trace semantics* by a final coalgebra in the Kleisli category $\mathcal{K}\ell(\mathcal{M}_\mathcal{S})$; it is easy to see that, for weighted tree automata, coalgebraic trace semantics is nothing but the weighted tree language concretely defined in Definition 7.7.

The notions of forward and backward Kleisli simulation is defined in [18] in categorical terms; and a (categorical) proof of their soundness against coalgebraic trace semantics is presented. Much like in the previous sections, we shall now characterize Kleisli simulations for weighted tree automata by matrices; their soundness then follows from the above mentioned categorical proof.

The following lemma is a generalization of Lemma 3.3. It introduces the matrix representation of the action of the functor F_Σ on Kleisli arrows.

Lemma 7.11 *Let $f : A \nrightarrow B$ be a Kleisli arrow in $\mathcal{K}\ell(\mathcal{M}_\mathcal{S})$ and M_f be its matrix representation (see Definition 3.1). Then the matrix representation $M_{\overline{F_\Sigma} f}$ of the arrow $\overline{F_\Sigma} f$ is given by*

$$\bigoplus_{n\in\mathbb{N}} \big(I_{\Sigma_n} \otimes M_f^{\otimes n}\big) \quad \in \mathcal{S}^{(F_\Sigma A)\times(F_\Sigma B)} \,. \tag{13}$$

Here, for each $n \in \mathbb{N}$ and $X_n \in \mathcal{S}^{A_n \times B_n}$, $\big(\bigoplus_{n\in\mathbb{N}} X_n\big) \in \mathcal{S}^{(\coprod_{n\in\mathbb{N}} A_n)\times(\coprod_{n\in\mathbb{N}} B_n)}$ is defined by

$$\Big(\bigoplus_{n\in\mathbb{N}} X_n\Big)_{x,y} = \begin{cases} (X_n)_{x,y} & (x \in A_n, y \in B_n) \\ 0 & (\text{otherwise}) \end{cases}$$

that is a generalization of the binary operation $\oplus$. The matrix $X^{\otimes n}$ is defined by

$$X^{\otimes n} = \underbrace{X \otimes X \otimes \cdots \otimes X}_{n} \quad . \qquad \square$$

We note that the matrix in (13) is equivalently expressed as $\bigoplus_{n\in\mathbb{N}} \big(\underbrace{M_f^{\otimes n} \oplus M_f^{\otimes n} \oplus \cdots \oplus M_f^{\otimes n}}_{|\Sigma_n|} \big)$.

In what follows, Definition 7.12 and Theorem 7.13 are parallel to Definition 3.4 and Corollary 3.6, respectively.

Definition 7.12 (simulation matrix) Let $\mathcal{A} = (Q_\mathcal{A}, \Sigma, M_\mathcal{A}, \alpha_\mathcal{A})$ and $\mathcal{B} = (Q_\mathcal{B}, \Sigma, M_\mathcal{B}, \alpha_\mathcal{B})$ be $\mathcal{S}$-weighted tree automata.

- A matrix $X \in S^{Q_\mathcal{B} \times Q_\mathcal{A}}$ is a *forward simulation matrix* from $\mathcal{A}$ to $\mathcal{B}$ if

  $$\alpha_\mathcal{A} \sqsubseteq \alpha_\mathcal{B} X \ , \quad \text{and} \quad X \cdot M_\mathcal{A}(a) \sqsubseteq M_\mathcal{B}(a) \cdot (X^{\otimes n}) \quad \text{for any } n \in \mathbb{N} \text{ and } a \in \Sigma_n.$$

- A matrix $X \in S^{Q_\mathcal{A} \times Q_\mathcal{B}}$ is a *backward simulation matrix* from $\mathcal{A}$ to $\mathcal{B}$ if

  $$\alpha_\mathcal{A} X \sqsubseteq \alpha_\mathcal{B} \ , \quad \text{and} \quad M_\mathcal{A}(a) \cdot (X^{\otimes n}) \sqsubseteq X \cdot M_\mathcal{B}(a) \quad \text{for any } n \in \mathbb{N} \text{ and } a \in \Sigma_n.$$

Similarly to the case of semiring weighted automata, we write $\mathcal{A} \sqsubseteq_\mathbf{F} \mathcal{B}$ and $\mathcal{A} \sqsubseteq_\mathbf{B} \mathcal{B}$ if there exists a forward and backward matrix simulation from $\mathcal{A}$ to $\mathcal{B}$, respectively.

Theorem 7.13 (soundness) *Let $\mathcal{A}$ and $\mathcal{B}$ be $\mathcal{S}$-weighted tree automata. Existence of a forward or backward simulation matrix from $\mathcal{A}$ to $\mathcal{B}$—i.e. $\mathcal{A} \sqsubseteq_\mathbf{F} \mathcal{B}$ or $\mathcal{A} \sqsubseteq_\mathbf{B} \mathcal{B}$—witnesses language inclusion $L(\mathcal{A}) \sqsubseteq L(\mathcal{B})$.*

Proof. Simulation matrices in Definition 7.12 coincide with Kleisli simulations in the general theory of [18]. The latter is sound with respect to coalgebraic trace semantics [18, 21]; and the last coincides with the weighted tree language in Definition 7.7. $\square$

We note that, differently from matrix simulations for semiring weighted automata (Definition 3.4), the inequalities in Definition 7.12 are not necessarily linear. For example,

$$X^{\otimes 2} \ = \ \begin{pmatrix} & \vdots & \\ \cdots & \boxed{x_{i,j} X} & \cdots \\ & \vdots & \end{pmatrix} \ .$$

This nonlinearity poses an algorithmic challenge: many known algorithms for feasibility of inequalities are restricted to linear ones. See Section 7.3 for further discussions.

In the remainder of this section, we present a concrete definition of forward partial execution for weighted tree automata. It is an instance of Definition 7.1. Its soundness, adequacy and monotonicity follow from the general results in Section 7.1.

Definition 7.14 (FPE for weighted tree automata) *Forward partial execution (FPE)* is transformation of a weighted tree automata such that: given an $\mathcal{S}$-weighted tree automaton $\mathcal{A} = (Q, \Sigma, M, \alpha)$ where $\Sigma = (\Sigma_n)_n$, and a parameter $P \subseteq Q$, the resulting automaton $\mathcal{A}_{\mathsf{FPE},P} = (Q', \Sigma, M', \alpha')$ is as follows. It has a state space

$$Q' \;=\; \big\{(a,(y_0,y_1,\ldots,y_{n-1})) \,\big|\, a \in \Sigma_n, \exists x \in P.\, M(a)_{x,(y_0,y_1,\ldots,y_{n-1})} \neq 0_{\mathcal{S}}\big\} + (Q \setminus P) \;, \tag{14}$$

that replaces each state $x \in P$ with its one-step behaviors $(a,(y_0,y_1,\ldots,y_{n-1}))$ as new states. As for the transition matrices M',

$$M'(a)_{(a,(x_0,\ldots,x_{n-1})),\left((a_0,(y_0^0,\ldots,y_{m_0-1}^0)),\ldots,(a_{n-1},(y_0^{n-1},\ldots,y_{m_{n-1}-1}^{n-1}))\right)} = \prod_{i=0}^{n-1} M(a_i)_{x_i,(y_0^i,\ldots,y_{m_i-1}^i)}$$

$$M'(a)_{(a,(x_0,x_1,\ldots,x_{n-1})),(x_0,x_1,\ldots,x_{n-1})} = 1_{\mathcal{S}}$$

$$M'(a)_{x,\left((a_0,(y_0^0,\ldots,y_{m_0-1}^0)),\ldots,(a_{n-1},(y_0^{n-1},\ldots,y_{m_{n-1}-1}^{n-1}))\right)}$$

$$= \Big(M(a)\cdot\Big(\bigotimes_{i=0}^{n-1} M(a_i)\Big)\Big)_{x,\left((y_0^0,\ldots,y_{m_0-1}^0),\ldots,(y_0^{n-1},\ldots,y_{m_{n-1}-1}^{n-1})\right)}$$

$$M'(a)_{x,y} = M(a)_{x,y}$$

where $a \in \Sigma_n$, $a_i \in \Sigma_{m_i}$, and $x, x_i, y_j^i \in Q$. For all the other cases we define $M'(a)_{u,v} = 0_{\mathcal{S}}$. As for the initial vector α', the definition is shown below.

$$\alpha'_{(a,(x_0,\ldots,x_{n-1}))} \;=\; (\alpha M(a))_{(x_0,\ldots,x_{n-1})} \;, \qquad \alpha'_x \;=\; \alpha_x$$

We do not yet have a good definition of backward partial execution for weighted tree automata, probably for the reason that we argued at the end of Section 7.1.

7.3. Implementation, Experiments and Discussions

We implemented, in OCaml, a program +×-treesim that searches for forward and backward simulation matrices between two given $\mathcal{S}_{+,\times}$-weighted tree automata. It first combines the constraints in Definition 7.12 into a system of (possibly nonlinear) polynomial inequalities, and tries to solve it with the `FindInstance` function of *Mathematica* [32].

We discuss the size of the system of inequalities to be solved. Assume that our goal is to establish $\mathcal{A} \sqsubseteq_{\mathbf{F}} \mathcal{B}$. Let n be the number of states of $\mathcal{A}$ and m be that of $\mathcal{B}$. (In case our goal is $\mathcal{A} \sqsubseteq_{\mathbf{B}} \mathcal{B}$ we swap n and m.) Let d be the maximum arity in the ranked alphabet Σ. Then the number of inequality constraints is at most $\sum_{k\in\mathbb{N}} |\Sigma_k| \cdot m \cdot n^k$; and the degree of each polynomial inequality constraint is at most d.

Experiments were done on a MacBook Pro laptop with a Core i5 processor (2.6 GHz, 2 cores) and 16 GB RAM. In our experiments we let the program +×-treesim try to establish $\mathcal{A} \sqsubseteq_{\mathbf{F}} \mathcal{A}$ (or $\mathcal{A} \sqsubseteq_{\mathbf{B}} \mathcal{A}$) for a randomly generated tree automaton $\mathcal{A}$.

Our current implementation turned out to be far from scalable: for the maximum arity $d = 2, 3$ and an automaton $\mathcal{A}$ with several states and transitions, the program barely manages to establish the goal; it becomes hopeless for bigger problem instances. This is in a sense as we expected: use of a general purpose algorithm like `FindInstance`

in *Mathematica* would never be a performance advantage; in fact, `FindInstance` tends to abort after tens of seconds, consuming a few MBs of memory, for reasons that we cannot know. An obvious alternative is to use special purpose algorithms—like the ones that are known in the field of convex optimization. This is left as future work.

8. Related Work

(Bi)simulation notions and (weighted) language equivalence/inclusion for quantitative systems have been an active research topic in the field of formal verification and concurrency. Some related work in this direction has already discussed in the earlier sections, including [25, 7, 2, 27, 5]. Moreover, works like [23, 18, 22, 36] take a categorical/coalgebraic approach.

Use of matrices as witnesses of quantitative language equivalence/inclusion is in fact not uncommon. The rest of this section is devoted to the discussion of such works, and their comparison to the current work. Overall, the current work is distinguished in the following aspects.

- The categorical backend of Kleisli simulation that allows clean theoretical developments. The latter include: the duality between forward and backward simulation; a general soundness proof; and generalization to tree automata (Section 7).
- Our simulations witness language *inclusion*, a problem that is harder than language *equivalence* (see Section 5.1).
- Forward/backward partial execution (Section 4) that enhances effectivity of the approach by matrix simulations.
- Actual implementation of the algorithms and experiments.

In [3], a notion called *conjugacy* between semiring weighted (word) automata is introduced. It is an equivalence notion—it is a special case of $\sqsubseteq_{\mathbf{F}}$ in the current work, with the inequalities in Definition 3.4 replaced with equalities. The notion of conjugacy comes with "completeness": assuming that the weight semiring is so-called a *division ring*, two automata are equivalent if and only if they are connected by some finite chain of conjugacies.

The notion of *simulation* in [14] is essentially the same as conjugacy in [3] (it is therefore an equivalence notion unlike the name). A simulation in [14] witnesses language equivalence. In [14] a semiring $\mathcal{S}$ is called *proper* when: two $\mathcal{S}$-weighted automata are language equivalent if and only if they are connected by a finite chain of simulations. The authors go on to study proper semirings: they present a necessary condition for a semiring to be proper, and an example that is not proper (namely $\mathcal{S}_{\max,+}$).

The results in [14] have been extended to weighted *tree* automata in [15]. Their simulation is a special case of ours (Definition 7.12) where inequalities are replaced with equalities; soundness with respect to tree language equivalence is proved; and completeness of a combined simulation (with an intermediate automaton, much like in our $\sqsubseteq_{\mathbf{BF}}$ and $\sqsubseteq_{\mathbf{FB}}$) is shown, under some assumptions.

Unlike the work discussed in the above, the works [8, 9, 11] study simulations given by matrices in the context of *fuzzy automata*. Here simulation is an oriented notion and

witnesses language inclusion (instead of language equivalence); its definition is essentially the same as ours (Definition 3.4). A principal difference between [8] and the current work is in the domain of weights: in [8] it is a structure called *residuated lattice*.

Algorithmic aspects of the simulation notion in [8] is pursued in [9], where an algorithm for computing the greatest simulation is presented. Their algorithm works for a general residuated lattice, unlike ours where linear inequalities are solved in semiring-specific manners.

These results in [8, 9] are adapted in [11] to automata weighted in a *semiring* (instead of a residuated lattice)—although some assumptions are imposed on a semiring and this makes the semiring $\mathcal{S}_{+,\times}$ unqualified.

9. Conclusions and Future Work

We introduced simulation matrices for weighted automata. While they are instances of (categorical) Kleisli simulations, their concrete presentation by matrices and linear inequalities yields concrete algorithms for simulation-based quantitative verification. Generalization to weighted tree automata follows immediately from the categorical theory behind, too, although linearity is lost in general.

There are some directions in which the current matrix-based simulation framework can be further generalized. Our idea of $\mathcal{S}_{+,\times}$-weighted automata was that they are probabilistic systems; when we wish to accommodate uncountable state spaces (for which discrete probabilities are hardly meaningful), we would need suitable measure theoretic machinery. In the context of the current work of traces and simulations, this will involve replacing **Sets** with **Meas** (the category of measurable sets and measurable functions), and matrix multiplication with Lebesgue integration. Trace semantics for probabilistic automata in **Meas** has been studied e.g. in [10, 26].

In fact, use of **Meas** as a base category becomes necessary if we consider *infinite* trace semantics—i.e. a language of accepted infinite words—even if a state space is countable (i.e. discrete). This is simply because the set Σ^ω of all infinite words is not countable. We are currently working on the soundness of matrix simulations against languages of infinite words; details will be presented in another venue.

Further generalization of the current theory will be concerned with acceptance conditions that are unique to infinite words. An example is the Büchi acceptance condition, for which a simulation notion (for the nondeterministic setting) has been studied in [16].

Finally, further optimization of our implementation is obvious future work.

Acknowledgments

Thanks are due to Andrzej Murawski, Shota Nakagawa and the anonymous referees for the conference version for useful discussions and comments; and to Björn Wachter for the user support of the tool APEX. The authors are supported by Grants-in-Aid for Young Scientists (A) No. 24680001, JSPS.

Appendix A. Omitted Proofs

Appendix A.1. Proof of Lem. 6.4

Proof. We prove the contraposition of each direction.

Assume that $\mathcal{A} \not\sqsubseteq_{\mathbf{G}} \mathcal{B}$, i.e. there exists a winning strategy (ρ_1, τ_1) for Challenger in a finite simulation game played on $\mathcal{A}$ and $\mathcal{B}$. Then a strategy that repeats ρ_1 and τ_1 is a winning strategy for Challenger in an infinite simulation game played on $\mathcal{A}^{\mathsf{Limavg}}$ and $\mathcal{B}^{\mathsf{Limavg}}$. Hence $\mathcal{A}^{\mathsf{Limavg}} \not\sqsubseteq_{\mathbf{G}}^{\mathsf{Limavg}} \mathcal{B}^{\mathsf{Limavg}}$.

Conversely, assume $\mathcal{A}^{\mathsf{Limavg}} \not\sqsubseteq_{\mathbf{G}}^{\mathsf{Limavg}} \mathcal{B}^{\mathsf{Limavg}}$. Then there exists a winning strategy τ_1^∞ for Challenger in an infinite simulation game played on $\mathcal{A}^{\mathsf{Limavg}}$ and $\mathcal{B}^{\mathsf{Limavg}}$. By [13], without loss of generality, we may assume that τ_1^∞ is a positional strategy; i.e. the value of $\tau_1^\infty((p_0, q_0)(a_1, p_1, q_1) \dots (a_i, p_i, q_i))$ only depends on p_i and q_i.

One might hope to use τ_1^∞ itself as Challenger's winning strategy for the finite game for $\mathcal{A} \sqsubseteq_{\mathbf{G}} \mathcal{B}$. This does not work in general, since a resulting run may not come back to $\star$. Modification of τ_1^∞ to force to visit $\star$ may make the strategy less advantageous. We shall show that such modification is nevertheless feasible, by finding an upper bound r for "the disadvantage that results from visiting $\star$." In the infinite game for $\mathcal{A}^{\mathsf{Limavg}} \sqsubseteq_{\mathbf{G}}^{\mathsf{Limavg}} \mathcal{B}^{\mathsf{Limavg}}$, a winning strategy can always additionally "save" the advantage r; and it will be spent to visit $\star$.

To each $x \in Q_{\mathcal{A}}$ we shall choose and assign an *exit path* π_x. Specifically, by the assumption that $\mathcal{A}$ has no trap states, each state $x \in Q_{\mathcal{A}}$ has a finite path $\pi_x = x b_1 u_1 b_2 \dots b_m u_m$ in $\mathcal{A}$ that "reaches the final state," that is, the path satisfies

$$M_{\mathcal{A}}(b_1)_{x,u_1} + M_{\mathcal{A}}(b_2)_{u_1,u_2} + \dots + M_{\mathcal{A}}(b_n)_{u_{m-1},u_m} + (\beta_{\mathcal{A}})_{u_m} \neq -\infty \ .$$

Within the path π_x, the advantages that Simulator can make are bounded: this follows from the assumptions that no weight in $\mathcal{B}$ is ∞, and that $\mathcal{B}$ is finite state (so that there are only finitely many choices of an initial state). Since there are only finitely many $x \in Q_{\mathcal{A}}$, we can take a global upper bound $r \in \mathbb{R}$. To summarize, the real number r is chosen so that: for each $x \in Q_{\mathcal{A}}$, for the choice of an exit path $\pi_x = x\, b_{x,1}\, u_{x,1}\, b_{x,2} \dots b_{x,m(x)}\, u_{x,m(x)}$ as described above, and for each path $\pi = y\, b_{x,1}\, y_1\, b_{x,2} \dots b_{x,m(x)}\, y_{m(x)}$ in $\mathcal{B}$ on the same word $b_{x,1}\, b_{x,2} \dots b_{x,m(x)}$, we have

$$\begin{aligned} M_{\mathcal{A}}(b_{x,1})_{x,u_{x,1}} + M_{\mathcal{A}}(b_{x,2})_{u_{x,1},u_{x,2}} + \dots + M_{\mathcal{A}}(b_{x,m(x)})_{u_{x,m(x)-1},u_{x,m(x)}} + (\beta_{\mathcal{A}})_{u_{x,m(x)}} + r \\ > \ M_{\mathcal{B}}(b_{x,1})_{y,y_1} + M_{\mathcal{B}}(b_{x,2})_{y_1,y_2} + \dots + M_{\mathcal{B}}(b_{x,m(x)})_{y_{m(x)-1},y_{m(x)}} + (\beta_{\mathcal{B}})_{y_{m(x)}} \ . \end{aligned} \tag{A.1}$$

Using this $r \in \mathbb{R}$ ("an upper bound for the cost of visiting $\star$"), we shall construct a strategy $\big(\rho_1' : 1 \to Q_{\mathcal{A}}, \tau_1' : (Q_{\mathcal{A}} \times Q_{\mathcal{B}}) \times (\Sigma \times Q_{\mathcal{A}} \times Q_{\mathcal{B}})^* \to 1 + \Sigma \times Q_{\mathcal{A}}\big)$ for Challenger in the finite simulation game for $\mathcal{A} \sqsubseteq_{\mathbf{G}} \mathcal{B}$ as follows. For a pair of runs $R = (p_0 a_1 \dots a_i p_i, q_0 a_1 \dots a_i q_i)$ on $\mathcal{A}$ and $\mathcal{B}$, we define the accumulated weights $S_{\mathcal{A}}^R$ and $S_{\mathcal{B}}^R$ by

$$S_{\mathcal{A}}^R = (\alpha_{\mathcal{A}})_{p_0} + M_{\mathcal{A}}(a_1)_{p_0,p_1} + \dots + M_{\mathcal{A}}(a_i)_{p_{i-1},p_i} \text{ and } S_{\mathcal{B}}^R = (\alpha_{\mathcal{B}})_{q_0} + M_{\mathcal{B}}(a_1)_{q_0,q_1} + \dots + M_{\mathcal{B}}(a_i)_{q_{i-1},q_i} \ .$$

While $S_{\mathcal{A}}^R \leq S_{\mathcal{B}}^R + r$ (i.e. the saving is not enough for the cost of visiting $\star$), ρ_1' and τ_1' are defined as follows. They do essentially the same thing as τ_1^∞ does, except that τ_1'

terminates when $\star$ is visited.

$$\rho_1'(\bullet) = \tau_1^\infty((\star,\star))$$
$$\tau_1'\big((p_0,q_0)(a_1,p_1,q_1)\dots(a_i,p_i,q_i)\big)$$
$$= \begin{cases} \checkmark & \big(\tau_1^\infty\big((p_0,q_0)(a_1,p_1,q_1)\dots(a_i,p_i,q_i)\big) = (!,\star)\big) \\ \tau_1^\infty\big((p_0,q_0)(a_1,p_1,q_1)\dots(a_i,p_i,q_i)\big) & (\text{otherwise}). \end{cases} \tag{A.2}$$

Assume that $S_{\mathcal{A}}^R > S_{\mathcal{B}}^R + r$ is satisfied at a certain stage, say at the state $p_n \in Q_{\mathcal{A}}$ for the first time. After that, τ_1' is defined for each $1 \le j \le m(p_n)$ by:

$$\tau_1'\big((p_0,q_0)(a_1,p_1,q_1)\dots(a_n,p_n,q_n)(b_{p_n,1},u_{p_n,1},y_1)\dots(b_{p_n,j},u_{p_n,j},y_j)\big)$$
$$= \begin{cases} (b_{p_n,j+1},u_{p_n,j+1}) & \big(j < m(p_n)\big) \\ \checkmark & \big(j = m(p_n)\big) \end{cases} \tag{A.3}$$

where $\pi_{p_n} = p_n\, b_{p_n,1}\, u_{p_n,1}\, b_{p_n,2} \dots b_{p_n,m(p_n)}\, u_{p_n,m(p_n)}$ is the exit path for p_n that we have fixed in the above. Here τ_1' is headed to the exit along π_{p_n}, no matter what Simulator's move y_j is.

It remains to show that the strategy (ρ_1', τ_1') of Challenger's is winning in the finite game for $\mathcal{A} \sqsubseteq_{\mathbf{G}} \mathcal{B}$. In the case where the clause (A.3) is never invoked, we must have that $\tau_1^\infty\big((p_0,q_0)(a_1,p_1,q_1)\dots(a_i,p_i,q_i)\big) = (!,\star)$ holds for some i. At such i Challenger must have an accumulated weight that is strictly bigger than Simulator does: by the assumption that τ_1^∞ is positional, the strategy τ_1^∞ will just repeat what it has done, and it will not win unless it is winning so far.

Finally, in the case where the clause (A.3) is invoked, the advantage that Simulator makes between the time $n+1$ and $n+m(p_n)$ is at most r by the definition of r. This does not eat up the "saving" r.

Therefore we have shown that $\mathcal{A} \not\sqsubseteq_{\mathbf{G}} \mathcal{B}$. This concludes the proof. □

References

[1] Akian, M., Gaubert, S., Guterman, A.E.: Tropical polyhedra are equivalent to mean payoff games. International Journal of Algebra and Computation 22(1) (2012)

[2] Baier, C., Hermanns, H., Katoen, J.P.: Probabilistic weak simulation is decidable in polynomial time. Inf. Process. Lett. 89(3), 123–130 (2004)

[3] Béal, M., Lombardy, S., Sakarovitch, J.: On the equivalence of $\mathbb{Z}$-automata. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3580, pp. 397–409. Springer (2005)

[4] Bezem, M., Nieuwenhuis, R., Rodríguez-Carbonell, E.: Exponential behaviour of the butkovic-zimmermann algorithm for solving two-sided linear systems in max-algebra. Discrete Applied Mathematics 156(18), 3506–3509 (2008)

[5] Blondel, V.D., Canterini, V.: Undecidable problems for probabilistic automata of fixed dimension. Theory Comput. Syst. 36(3), 231–245 (2003)

[6] Butkovic, P., Zimmermann, K.: A strongly polynomial algorithm for solving two-sided linear systems in max-algebra. Discrete Applied Math. 154(3), 437–446 (2006)

[7] Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. ACM Trans. Comput. Log. 11(4) (2010), `http://doi.acm.org/10.1145/1805950.1805953`

[8] Ciric, M., Ignjatovic, J., Damljanovic, N., Basic, M.: Bisimulations for fuzzy automata. Fuzzy Sets and Systems 186(1), 100–139 (2012)

[9] Ciric, M., Ignjatovic, J., Jancic, I., Damljanovic, N.: Computation of the greatest simulations and bisimulations between fuzzy automata. Fuzzy Sets and Systems 208, 22–42 (2012), http://dx.doi.org/10.1016/j.fss.2012.05.006
[10] Cîrstea, C.: Maximal traces and path-based coalgebraic temporal logics. Theor. Comput. Sci. 412(38), 5025–5042 (2011)
[11] Damljanovic, N., Ciric, M., Ignjatovic, J.: Bisimulations for weighted automata over an additively idempotent semiring. Theor. Comput. Sci. 534, 86–100 (2014)
[12] Doyen, L., Henzinger, T.A., Raskin, J.F.: Equivalence of labeled markov chains. Int. J. Found. Comput. Sci. 19(3), 549–563 (2008)
[13] Ehrenfeucht, A., Mycielski, J.: Positional strategies for mean payoff games. International Journal of Game Theory 8(2), 109–113 (1979)
[14] Ésik, Z., Maletti, A.: Simulation vs. equivalence. In: Arabnia, H.R., Gravvanis, G.A., Solo, A.M.G. (eds.) Proceedings of the 2010 International Conference on Foundations of Computer Science, FCS 2010, July 12-15, 2010, Las Vegas, Nevada, USA. pp. 119–124. CSREA Press (2010)
[15] Ésik, Z., Maletti, A.: The category of simulations for weighted tree automata. Int. J. Found. Comput. Sci. 22(8), 1845–1859 (2011)
[16] Etessami, K., Wilke, T., Schuller, R.A.: Fair simulation relations, parity games, and state space reduction for büchi automata. SIAM J. Comput. 34(5), 1159–1175 (2005)
[17] The GNU linear programming kit, http://www.gnu.org/software/glpk
[18] Hasuo, I.: Generic forward and backward simulations. In: Baier, C., Hermanns, H. (eds.) CONCUR. Lect. Notes Comp. Sci., vol. 4137, pp. 406–420. Springer (2006)
[19] Hasuo, I.: Generic forward and backward simulations II: Probabilistic simulation. In: Gastin, P., Laroussinie, F. (eds.) CONCUR. Lecture Notes in Computer Science, vol. 6269, pp. 447–461. Springer (2010)
[20] Hasuo, I., Jacobs, B.: Context-free languages via coalgebraic trace semantics. In: Fiadeiro, J.L., Harman, N., Roggenbach, M., Rutten, J.J.M.M. (eds.) CALCO. Lecture Notes in Computer Science, vol. 3629, pp. 213–231. Springer (2005)
[21] Hasuo, I., Jacobs, B., Sokolova, A.: Generic trace semantics via coinduction. Logical Methods in Computer Science 3(4) (2007)
[22] Hasuo, I., Kawabe, Y., Sakurada, H.: Probabilistic anonymity via coalgebraic simulations. Theor. Comput. Sci. 411(22-24), 2239–2259 (2010)
[23] Hughes, J., Jacobs, B.: Simulations in coalgebra. Theor. Comput. Sci. 327(1-2), 71–108 (2004)
[24] Jacobs, B.: Introduction to coalgebra. Towards mathematics of states and observations (2012), Draft of a book (ver. 2.0), available online
[25] Jonsson, B., Larsen, K.G.: Specification and refinement of probabilistic processes. In: LICS. pp. 266–277. IEEE Computer Society (1991)
[26] Kerstan, H., König, B.: Coalgebraic trace semantics for continuous probabilistic transition systems. Logical Methods in Computer Science 9(4) (2013), http://dx.doi.org/10.2168/LMCS-9(4:16)2013
[27] Kiefer, S., Murawski, A.S., Ouaknine, J., Wachter, B., Worrell, J.: Language equivalence for probabilistic automata. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV. Lecture Notes in Computer Science, vol. 6806, pp. 526–540. Springer (2011)
[28] Kiefer, S., Murawski, A.S., Ouaknine, J., Wachter, B., Worrell, J.: Algorithmic probabilistic game semantics—playing games with automata. Formal Methods in System Design 43(2), 285–312 (2013)
[29] Konstantinos, C., Catuscia, P.: Probable innocence revisited. Theoretical Computer Science 367(1), 123–138 (2006)
[30] Krob, D.: The equality problem for rational series with multiplicities in the tropical semiring is undecidable. In: Kuich, W. (ed.) ICALP. Lecture Notes in Computer Science, vol. 623, pp. 101–112. Springer (1992)
[31] Lynch, N.A., Vaandrager, F.W.: Forward and backward simulations: I. Untimed systems. Inf. Comput. 121(2), 214–233 (1995)
[32] Mathematica, http://www.wolfram.com/mathematica/
[33] Pin, J.E.: Tropical semirings. Idempotency (Bristol, 1994) pp. 50–69 (1998)
[34] Power, J., Turi, D.: A coalgebraic foundation for linear time semantics. Electr. Notes Theor. Comput. Sci. 29, 259–274 (1999)
[35] Reiter, M.K., Rubin, A.D.: Crowds: Anonymity for web transactions. ACM Trans. Inf. Syst. Secur. 1(1), 66–92 (1998)
[36] Sokolova, A.: Coalgebraic Analysis of Probabilistic Systems. Ph.D. thesis, Techn. Univ. Eindhoven (2005)

[37] Tzeng, W.G.: A polynomial-time algorithm for the equivalence of probabilistic automata. SIAM J. Comput. 21(2), 216–227 (1992)

[38] Urabe, N., Hasuo, I.: Generic forward and backward simulations III: quantitative simulations by matrices. In: Baldan, P., Gorla, D. (eds.) CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8704, pp. 451–466. Springer (2014), http://dx.doi.org/10.1007/978-3-662-44584-6_31

[39] Zwick, U., Paterson, M.: The complexity of mean payoff games on graphs. Theor. Comput. Sci. 158(1&2), 343–359 (1996)