Generic Forward and Backward Simulations[⋆]

Ichiro Hasuo[⋆⋆]

Institute for Computing and Information Sciences
Radboud University Nijmegen, the Netherlands
`http://www.cs.ru.nl/~ichiro`

Abstract. The technique of forward/backward simulations has been applied successfully in many distributed and concurrent applications. In this paper, however, we claim that the technique can actually have more genericity and mathematical clarity. We do so by identifying forward/backward simulations as lax/oplax morphisms of coalgebras. Starting from this observation, we present a systematic study of this generic notion of simulations. It is meant to be a generic version of the study by Lynch and Vaandrager, covering both non-deterministic and probabilistic systems. In particular we prove soundness and completeness results with respect to trace inclusion: the proof is by coinduction using the generic theory of traces developed by Jacobs, Sokolova and the author. By suitably instantiating our generic framework, one obtains the appropriate definition of forward/backward simulations for various kinds of systems, for which soundness and completeness come *for free*.

1 Introduction

The theory of forward/backward simulations for non-deterministic automata has been extensively studied, notably by Lynch and Vaandrager [12]. It has been applied successfully in many distributed and concurrent applications, described as transition systems. For example, in [9] trace-based anonymity properties for network protocols are proved by building backward simulations. The notions of forward/backward simulations are also extended to different kinds of state-based systems such as probabilistic ones [15].

In this paper we claim that this theory of forward/backward simulations can actually have more genericity and mathematical clarity. We do so by revealing a simple mathematical structure hidden behind various notions of simulations defined for different kinds of systems. The slogan is:

Forward/backward simulations are lax/oplax morphisms
of coalgebras in Kleisli categories.

[⋆] An extended version of this paper appears as [6].

[⋆⋆] Part of this work was done during the author's stay at Research Center for Verification and Semantics, National Institute of Advanced Industrial Science and Technology (AIST), Japan. The author is grateful for the hospitality.

Based on this observation, we aim at presenting a generic version of the systematic study [12]. The outcome is satisfactory. We employ the generic theory of traces in [5] and show:

– *Soundness.* Existence of a forward or backward simulation implies trace inclusion.
– *Completeness.* Trace inclusion implies existence of a certain kind of hybrid simulation, namely a backward-forward simulation.

The important point is that all these definitions and proofs are stated in abstract coalgebraic terms, hence come with ample genericity. In fact they are parametrized by:

– The type of branching. It can be either non-determinism (with a set of possible transitions) or probabilism (with a distribution over possible transitions).
– The type of transitions. For example, a context-free grammar can be considered as a state-based system—non-terminals as states—with non-deterministic branching. It has a different transition type from, say, LTS's: a CFG transits to a word over symbols and states, while an LTS transits to a pair of a symbol and a (next) state. Our result covers a wide variety of transition types.

Hence for each application from such a wide variety, one can obtain a definition of forward/backward simulations by instantiating our general framework with suitable parameters. Moreover one is assured that this definition is the *right* one: good properties such as soundness and completeness come for free. Therefore we expect abundant practical implication of this work.

Now let us take a completely different standpoint, namely that of a coalgebra-theorist. This work cultivates a new field of coalgebraic methods in computer science: coalgebras in a Kleisli category. The standard theory of coalgebras (e.g. [14]) is based in **Sets**, establishing the (successful) second row of the table. This paper, following the previous work [5], extends this table downwards.

base category	morphisms of coalgebras	coinduction gives
Sets	functional bisimulation	bisimilarity
Kleisli	lax ··· forward simulation; oplax ··· backward simulation [this paper]	trace semantics [5]

The paper is organized as follows. In Section 2 our basic (coalgebraic) setting is presented. State-based systems are formulated as coalgebras with explicit start states in Section 3. The key notion of generic forward/backward simulations is presented in Section 4. In Section 5 we recall the generic theory of coalgebraic traces from [5]. The materials of the previous two sections are combined in Section 6 to prove soundness and completeness. We conclude in Section 7.

2 Preliminaries

This section presents preliminaries from category theory and theory of coalgebras. They are put in an elementary and descriptive manner. For more details the reader is referred to [5].

In this paper we identify forward/backward simulations as lax/oplax morphisms of coalgebras in a Kleisli category $\mathcal{K}\ell(T)$ for a monad T on **Sets**. This observation is inspired by a series of work (stemming from [13]) on trace semantics for/via coalgebras: a Kleisli category is a suitable base category there. Our basic story is as follows.

We model a state-based system as a coalgebra $X \to TFX$ in **Sets**, with T a monad, F a functor and a distributive law $FT \Rightarrow TF$ implicit. The intuition is:

– a monad T describes the type of *branching* (non-determinism, probabilism, etc.) of the system;
– a functor F describes the *transition* type of the system, which determines the type of linear-time behavior (e.g. words over action symbols);
– a distributive law $FT \Rightarrow TF$ describes the way how T's effect of branching is distributed over the transition type represented by F.

It turns out that having $X \to TFX$ in **Sets** is equivalent to having a coalgebra $X \to \overline{F}X$ in the Kleisli category $\mathcal{K}\ell(T)$, where $\overline{F} : \mathcal{K}\ell(T) \to \mathcal{K}\ell(T)$ is a canonical lifting of F : **Sets** $\to$ **Sets** with $\overline{F}X = FX$. This lifting $\overline{F}$ is induced by the distributive law. To summarize:

– In modeling a system as a coalgebra $X \to TFX$, we separate the type of branching T from the transition type F.
– By moving from **Sets** to $\mathcal{K}\ell(T)$, this coalgebra becomes a coalgebra $X \to \overline{F}X$ for a functor $\overline{F}$—instead of a combination TF. Then we can start the usual coalgebraic business such as morphisms, final coalgebras and coinduction.

2.1 Monads for types of branching

A *monad* T on **Sets** is an endofunctor on **Sets** equipped with two kinds of functions: for each set X, the *unit* $X \overset{\eta_X}{\to} TX$ and the *multiplication* $TTX \overset{\mu_X}{\to} TX$. These functions must satisfy certain coherence conditions.

In coalgebraic settings, it is shown in [5] that monads with a certain order structure are suitable for modeling state-based systems with branching, especially for analyzing their trace semantics. We are interested in such monads in this paper. We have two examples:

– The *powerset monad* $\mathcal{P}$, modeling the *non-deterministic* branching.
– The *subdistribution monad* $\mathcal{D}$, modeling the *probabilistic* branching. For a set X, $\mathcal{D}X$ is given by: $\mathcal{D}X = \{\, \xi : X \to [0,1] \mid \sum_{x \in X} \xi(x) \le 1 \,\}$. Here ξ is called a *probability subdistribution* over X. It is "sub" because the sum of all probabilities is not necessarily equal to 1.

We take the subdistribution monad $\mathcal{D}$, instead of the distribution monad $\mathcal{D}_{=1}X = \{\xi \mid \sum_x \xi(x) = 1\}$, since the latter lacks a suitable order structure. This point is elaborated in Section 2.3.

2.2 Kleisli categories for monads

For each monad T on **Sets**, we construct the *Kleisli category* for T, denoted by $\mathcal{K}\ell(T)$, in the following way. The crucial part is that an arrow $X \to Y$ in $\mathcal{K}\ell(T)$ is actually a function $X \to TY$ in **Sets**.

- Objects in $\mathcal{K}\ell(T)$ are the same as in **Sets**: they are just sets.
- An arrow $X \to Y$ in $\mathcal{K}\ell(T)$ is a function $X \to TY$ in **Sets**.
- Composition of arrows is defined using multiplication $\mu_X : TTX \to TX$.
- The identity arrow $X \xrightarrow{\mathrm{id}} X$ in $\mathcal{K}\ell(T)$ is the unit $X \xrightarrow{\eta_X} TX$ in **Sets**.

This $\mathcal{K}\ell(T)$ will be our base category. Notice that when we write $X \to Y$ in $\mathcal{K}\ell(T)$, a branching nature of this arrow is implicit because it is a function $X \to TY$.

For the monads $\mathcal{P}$ and $\mathcal{D}$ of our interest, we shall describe more details of their Kleisli categories.

The category $\mathcal{K}\ell(\mathcal{P})$ is in fact isomorphic to the category **Rel** of sets and relations. That is, an arrow $X \to Y$ in $\mathcal{K}\ell(\mathcal{P})$ is a relation between X and Y via the standard "relation-into-function" trick: given a function $f : X \to \mathcal{P}Y$ in **Sets** we obtain a relation $R_f = \{(x,y) \mid y \in f(x)\}$. In particular, composition of arrows in $\mathcal{K}\ell(\mathcal{P})$ is given by the relational composition $S \circ R = \{(x,z) \mid \exists y.\, xRy \wedge ySz\}$ of the corresponding relations. The identity arrow id_X is the diagonal relation $\{(x,x) \mid x \in X\}$.

In $\mathcal{K}\ell(\mathcal{D})$ an arrow $X \to Y$ assigns to each $x \in X$ a probability subdistribution over Y. The identity arrow $X \xrightarrow{\mathrm{id}} X$ maps $x \in X$ to the so-called *Dirac distribution* for x. The composition of arrows $X \xrightarrow{f} Y \xrightarrow{g} Z$ in $\mathcal{K}\ell(\mathcal{D})$ is such that: for $x \in X$ and $z \in Z$, $(g \circ f)(x)(z) = \sum_{y \in Y} f(x)(y) \cdot g(y)(z)$.

2.3 Order-enriched structure of Kleisli categories

The notion of branching—such as non-determinism and probabilism—come with natural notions of order. For non-determinism we have the inclusion order between sets of possible transitions. For probabilism a subdistribution ξ is bigger than ψ if, to each possible transition, ξ assigns bigger probability than ψ does.

These natural orders accompanying the notion of branching appear in our setting as a $\mathbf{DCpo}_\perp$-enriched structure of Kleisli categories. This order structure is fully exploited in the definition of forward/backward simulations: a system simulates another one if it has *more* behavior.

For $T = \mathcal{P}$ or $\mathcal{D}$, the Kleisli category $\mathcal{K}\ell(T)$ is $\mathbf{DCpo}_\perp$*-enriched*. This means:

- For any pair of sets X and Y, the set $\mathrm{Hom}_{\mathcal{K}\ell(T)}(X,Y)$ of the arrows from X to Y has a dcpo structure $\sqsubseteq$ with bottom. In particular we can take the supremum $\bigsqcup_{n<\omega} f_n$ of an increasing chain $f_0 \sqsubseteq f_1 \sqsubseteq \cdots$ of arrows, and there is the minimum arrow $\perp_{X,Y} : X \to Y$.
- Composition of arrows is continuous: $g \circ (\bigsqcup_n f_n) = \bigsqcup_n (g \circ f_n)$ and $(\bigsqcup_n f_n) \circ h = \bigsqcup_n (f_n \circ h)$. In particular composition is monotone.

Indeed, for $T = \mathcal{P}$ or $\mathcal{D}$, a set TY has a $\mathbf{DCpo}_\perp$ structure $\sqsubseteq_{TY}$. This extends to the order between arrows in $\mathcal{K}\ell(T)$ in a pointwise manner: for $f, g : X \rightrightarrows Y$, $f \sqsubseteq g$ if for each $x \in X$, $f(x) \sqsubseteq_{TY} g(x)$.

We need the minimum arrow $\perp_{X,Y} : X \to Y$ in $\mathcal{K}\ell(T)$ for the trace semantics results in Section 5. It is not available for the distribution monad $\mathcal{D}_{=1}$: that is why we use the subdistribution monad $\mathcal{D}$ instead.

2.4 Shapely functors for transition types

We restrict a functor F—which models the transition type of a system—to be *shapely*. The reason to do so is: we know the results on coalgebraic trace semantics in Section 5 hold for shapely functors,[1] and also in most of the interesting examples we can take as F a shapely functor. The family of shapely functors is almost as broad as that of polynomial functors: it is defined inductively by the following BNF notation.

$$F, G, F_i ::= \mathrm{id} \mid \Sigma \mid F \times G \mid \textstyle\coprod_{i \in I} F_i \ ,$$

where Σ denotes the constant functor into an arbitrary set Σ, and I is an arbitrary index set. Here are some virtue of shapely functors which we will exploit.

- An initial F-algebra exists, obtained via the initial sequence of length ω.
- For $T = \mathcal{P}$ or $\mathcal{D}$, there is a canonical distributive law $FT \Rightarrow TF$. Equivalently, F has a canonical lifting $\overline{F}$ on $\mathcal{K}\ell(T)$. On objects $\overline{F}X = FX$, and on arrows $\overline{F}$'s action is what one might think of at first sight.

3 Coalgebraic modeling of systems

In this section we model a wide variety of branching state-based systems as what we call (T, F)-*systems*. A (T, F)-system is a $\overline{F}$-coalgebra in the Kleisli category $\mathcal{K}\ell(T)$ plus explicit start states. This definition of (T, F)-systems will be motivated by several illustrating examples.

Two parameters in the notion of (T, F)-systems are: T is a monad, being either $\mathcal{P}$ or $\mathcal{D}$, representing the branching type; F is a shapely functor describing the transition type. In the sequel we assume that T and F are such.

Definition 3.1 **((T, F)-systems)** A (T, F)-*system* is a pair of arrows

$$1 \xrightarrow{s} X \xrightarrow{c} \overline{F}X \qquad \text{in the Kleisli category } \mathcal{K}\ell(T).$$

That is, a pair of functions $(s : 1 \to TX,\ c : X \to TFX)$ in **Sets**, recalling that $\overline{F}X = FX$. The arrow s is called the *start states map*, and the $\overline{F}$-coalgebra c is called the *dynamics*. The set X is called the *state space*. The only element of the singleton 1 appearing here[2] is denoted by $*$.

[1] This does not say that those results hold exclusively for shapely functors.

[2] In this paper we will have singletons with different computational meanings. Accordingly, their only elements will be denoted by different symbols.

In most literature on coalgebras the start state (or the set of start states) is left implicit. However in our setting we need to have them explicit. See [6, Appendix 1] for further discussion.

Example 3.2 (Non-deterministic automata) Let us take the powerset monad $\mathcal{P}$ for T, hence non-deterministic branching. For an endofunctor F we take $1 + \Sigma \times \_$, where $1 = \{\checkmark\}$ is a singleton and Σ is a non-empty set of symbols. A (T, F)-system then is a pair of functions in **Sets**,

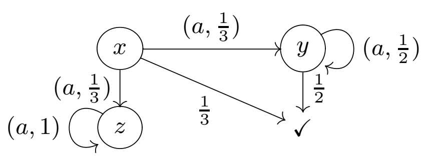
$$\big(\, 1 \xrightarrow{s} \mathcal{P}X, \qquad X \xrightarrow{c} \mathcal{P}(1 + \Sigma \times X)\, \big) \ ,$$

which should be interpreted as follows. The subset $s(*)$ of X is the set of possible start states. For a state $x \in X$, the set $c(x)$ contains $\checkmark$ if x is an accepting state; it contains a tuple (a, x') if there is a (possible) transition $x \overset{a}{\to} x'$. In this way a (T, F)-system for these T and F is thought of as a non-deterministic automaton.

Example 3.3 (Probabilistic automata) Let us take $T = \mathcal{D}$ instead of $\mathcal{P}$ in the previous example. A (T, F)-system is a pair of functions in **Sets**:

$$\big(\, 1 \xrightarrow{s} \mathcal{D}X, \qquad X \xrightarrow{c} \mathcal{D}(1 + \Sigma \times X)\, \big) \ .$$

This is understood as follows. The subdistribution $s(*)$ over X represents the probability with which each state $x \in X$ is chosen as a starting state. An execution successfully terminates at x with the probability $c(x)(\checkmark)$; a transition $x \overset{a}{\to} x'$ is made with the probability $c(x)(a, x')$. Such a system is called a *generative probabilistic transition system* [17,16]: in this paper we shall call it simply a *probabilistic automaton*. Here is an example of a probabilistic automaton.



with start states: $\begin{bmatrix} x \mapsto 1/3 \\ y \mapsto 2/3 \end{bmatrix}$.

This is modeled as the following $(\mathcal{D}, 1 + \Sigma \times \_)$-system.

- The start state map $1 \overset{s}{\to} \mathcal{D}X$ is such that $s(*) = \begin{bmatrix} x \mapsto 1/3 \\ y \mapsto 2/3 \end{bmatrix}$, and
- the dynamics coalgebra $X \overset{c}{\to} \mathcal{D}(1 + \Sigma \times X)$ is such that $c(x) = [\ (a, y) \mapsto 1/3,\ (a, z) \mapsto 1/3,\ \checkmark \mapsto 1/3\ ]$, etc.

Remark 3.4 (Systems with both non-determ./probabilistic branching) Probabilistic I/O automata [18] are another kind of well-studied models for state-based systems. One of their features is that they are equipped with both non-deterministic and probabilistic branching. Unfortunately, we are yet to find a suitable monad to model this combined branching: that is why probabilistic I/O automata are currently out of the scope of our generic framework.

Context-free grammars (without finiteness assumptions) can be also modeled as (T, F)-systems with $T = \mathcal{P}$ and a suitable F (see [4]).

The notion of morphisms of coalgebras extends to (T, F)-systems in an obvious manner [6].

4 Forward/backward simulations, coalgebraically

This section presents the key notions of this paper: generic forward, backward and backward-forward simulations. The intuition about order accompanying the notion of "branching"—now substantiated as the $\mathbf{DCpo}_\bot$-enriched structure of a Kleisli category—is fully exploited here.

In this section again $T = \mathcal{P}$ or $\mathcal{D}$, and F is a shapely functor.

Definition 4.1 (Forward simulation) Let $1 \xrightarrow{s} X \xrightarrow{c} FX$ and $1 \xrightarrow{t} Y \xrightarrow{d} FY$ be (T, F)-systems, presented in $\mathcal{K}\ell(T)$. A *forward simulation* from (t, d) to (s, c) is an arrow $f : X \to Y$ in $\mathcal{K}\ell(T)$ such that:

$$t \sqsubseteq f \circ s \quad \text{and} \quad d \circ f \sqsubseteq \overline{F}f \circ c \ ,$$

where $\sqsubseteq$ refers to the order available due to the $\mathbf{DCpo}_\bot$-enriched structure of the Kleisli category. Diagrammatically presented,

$$\begin{tikzcd}
FX \arrow[rr, "\overline{F}f"] & & FY \\
X \arrow[u, "c"] \arrow[rr, "f"] & \sqsupseteq & Y \arrow[u, "d"'] \\
& 1 \arrow[ul, "s"] \arrow[ur, "t"'] \arrow[u, phantom, "\sqsupseteq"] &
\end{tikzcd} \qquad (1)$$

In other words, a forward simulation is a *lax morphism* from (s, c) to (t, d).

We write $(t, d) \sqsubseteq_{\mathbf{F}} (s, c)$ if there is a forward simulation from (t, d) to (s, c).

The use of lax morphisms in categorical accounts of simulation/refinement is found in [10]. In a coalgebraic setting, [2] uses lax morphisms to investigate order-enriched version of bisimulation. However, to the best of our knowledge, we are the first to notice the significance of lax morphisms in Kleisli categories.

The dual notion, with the order of arrows opposed, has also a significant computational meaning.

Definition 4.2 (Backward simulation) Let $1 \xrightarrow{s} X \xrightarrow{c} FX$ and $1 \xrightarrow{t} Y \xrightarrow{d} FY$ be (T, F)-systems, presented in $\mathcal{K}\ell(T)$. A *backward simulation* from (s, c) to (t, d) is an arrow $f : X \to Y$ in $\mathcal{K}\ell(T)$ such that: $f \circ s \sqsubseteq t$ and $\overline{F}f \circ c \sqsubseteq d \circ f$.

$$\begin{tikzcd}
FX \arrow[rr, "\overline{F}f"] & & FY \\
X \arrow[u, "c"] \arrow[rr, "f"] & \sqsubseteq & Y \arrow[u, "d"'] \\
& 1 \arrow[ul, "s"] \arrow[ur, "t"'] \arrow[u, phantom, "\sqsubseteq"] &
\end{tikzcd} \qquad (2)$$

Hence a backward simulation is an *oplax morphism* of systems.

We write $(s,c) \sqsubseteq_{\mathbf{B}} (t,d)$ if there is a backward simulation from (s,c) to (t,d).

Remark 4.3 Note the direction of forward/backward simulations and lax/oplax morphisms. In general, the system which appears on the smaller sides of inequalities is simulated by the other one. For example, a lax morphism from (s,c) to (t,d) in Diagram (1) is a forward simulation *from* (t,d) *to* (s,c), through which (s,c) *forward-simulates* (t,d); hence $(t,d) \sqsubseteq_{\mathbf{F}} (s,c)$.

Let us be convinced of these abstract definitions by looking at examples.

Example 4.4 (Non-deterministic automata) In the setting of Example 3.2, an arrow $X \to Y$ in $\mathcal{K}\ell(T)$ is a relation R from X to Y since $\mathcal{K}\ell(\mathcal{P}) \cong \mathbf{Rel}$. The previous definitions boil down as follows: R is a forward simulation from (t,d) to (s,c) if and only if it satisfies the following conditions.

$$\begin{array}{rcl} y \in \mathsf{start}_{(t,d)} & \Longrightarrow & \exists x \in \mathsf{start}_{(s,c)}.\quad xRy\ , \\ xRy \ \wedge\ y \to_d \checkmark & \Longrightarrow & x \to_c \checkmark\ , \\ xRy \ \wedge\ y \xrightarrow{a}_d y' & \Longrightarrow & \exists x' \in X.\quad \big(\, x \xrightarrow{a}_c x' \ \wedge\ x'Ry' \,\big)\ , \end{array}$$

where $\mathsf{start}_{(s,c)}$ denotes the set $s(*)$. These conditions are much like those in the standard literature [12]. Notice in particular that the third condition is of the following familiar form, working "forwards".

$$\begin{tikzcd} x \arrow[d, dash, dashed, "R"'] & & & x \arrow[r, "a"] \arrow[d, dash, dashed, "R"'] & \exists x' \arrow[d, dash, dashed, "R"] \\ y \arrow[r, "a"] & y' & \Longrightarrow & y \arrow[r, "a"] & y' \end{tikzcd}$$

Similarly, a relation R from X to Y is a backward simulation from (s,c) to (t,d) if and only if:

$$\begin{array}{rcl} x \in \mathsf{start}_{(s,c)} \ \wedge\ xRy & \Longrightarrow & y \in \mathsf{start}_{(t,d)}\ , \\ x \to_c \checkmark & \Longrightarrow & \exists y \in Y.\quad \big(\, xRy \ \wedge\ y \to_d \checkmark \,\big)\ , \\ x \xrightarrow{a}_c x' \ \wedge\ x'Ry' & \Longrightarrow & \exists y \in Y.\quad \big(\, xRy \ \wedge\ y \xrightarrow{a}_d y' \,\big)\ . \end{array}$$

The third condition here works "backwards" in the following way.

$$\begin{tikzcd} x \arrow[r, "a"] & x' \arrow[d, dash, dashed, "R"] & & x \arrow[r, "a"] \arrow[d, dash, dashed, "R"'] & x' \arrow[d, dash, dashed, "R"] \\ & y' & \Longrightarrow & \exists y \arrow[r, "a"] & y' \end{tikzcd}$$

Example 4.5 (Probabilistic automata) In the setting of Example 3.3, the abstract Definition 4.1 is instantiated as follows: a function $f : X \to \mathcal{D}Y$ in **Sets** is a forward simulation from (t,d) to (s,c) if and only if:

$$\begin{array}{rcl} t(*)(y) & \leq & \sum_{x\in X} s(*)(x) \cdot f(x)(y)\ , \\ \sum_{y\in Y} f(x)(y) \cdot d(y)(\checkmark) & \leq & c(x)(\checkmark)\ , \\ \sum_{y\in Y} f(x)(y) \cdot d(y)(a,y') & \leq & \sum_{x'\in X} c(x)(a,x') \cdot f(x')(y')\ . \end{array} \tag{3}$$

It is also straightforward to instantiate Definition 4.2 of backward simulations.

One may wonder why we can call such f a forward simulation, although one can notice that a "forward" argument similar to the previous example is going on. The point is that, however, by the abstract theorems in the following sections we know that this definition (3) of forward simulations—derived from the coalgebraic definition—satisfies desirable properties such as soundness/completeness with respect to trace inclusion.

We define a simulation from one probabilistic system to another to be a function $X \to \mathcal{D}Y$. This is different from the approach in [7]: there a simulation is always a relation between state spaces X and Y.

By suitably instantiating the generic definitions, we also obtain appropriate notions of simulations for context-free grammars.

Forward and backward simulations will be shown to be sound with respect to trace inclusion. But they in general fail to be complete. Instead, a completeness result is proved for a certain combination of forward and backward simulations (*hybrid* simulations), as is done in [12].

Definition 4.6 (Backward-forward simulations) Let (s,c) and (t,d) be (T,F)-systems. A *backward-forward simulation* from (s,c) to (t,d) is a pair of

- a backward simulation f from (s,c) to an intermediate system (r,b), and
- a forward simulation g from the intermediate system (r,b) to (t,d).

Diagrammatically presented in $\mathcal{K}\ell(T)$ (note the direction of arrows),

$$\begin{tikzcd}
FX \arrow[r, "\overline{F}f"] & FU & FY \arrow[l, "\overline{F}g"'] \\
X \arrow[u, "c"] \arrow[r, "f"] \arrow[ur, phantom, "\sqsubseteq"] & U \arrow[u, "b"] \arrow[ur, phantom, "\sqsubseteq"] & Y \arrow[u, "d"'] \arrow[l, "g"'] \\
& 1 \arrow[u, "r"] \arrow[ul, bend left, "s"] \arrow[ur, bend right, "t"'] \arrow[ul, phantom, "\sqsubseteq"] \arrow[ur, phantom, "\sqsubseteq"] &
\end{tikzcd} \qquad (4)$$

We write $(s,c) \sqsubseteq_{\mathbf{BF}} (t,d)$ if there is a backward-forward simulation from (s,c) to (t,d). Obviously,

$$(s,c) \sqsubseteq_{\mathbf{BF}} (t,d) \qquad \Longleftrightarrow \qquad \exists (r,b).\quad \big(\, (s,c) \sqsubseteq_{\mathbf{B}} (r,b) \;\wedge\; (r,b) \sqsubseteq_{\mathbf{F}} (t,d) \,\big) \ .$$

Remark 4.7 (Forward-backward simulations) It is straightforward to define the notion of *forward-backward simulations* and the relation $\sqsubseteq_{\mathbf{FB}}$, as a suitable dual of Definition 4.6. This is done in [12] for a restricted class of non-deterministic systems. In the same paper $\sqsubseteq_{\mathbf{BF}}$ and $\sqsubseteq_{\mathbf{FB}}$ are shown to coincide.

However we have not yet found the coincidence of $\sqsubseteq_{\mathbf{BF}}$ and $\sqsubseteq_{\mathbf{FB}}$ in general: in the light of Theorem 6.2, it seems that $\sqsubseteq_{\mathbf{BF}}$ is the more fundamental notion. The coincidence for non-deterministic systems in [12] may be because $\mathcal{K}\ell(\mathcal{P})$ is self-dual, i.e. $\mathcal{K}\ell(\mathcal{P}) \cong \mathcal{K}\ell(\mathcal{P})^{\mathrm{op}}$. Details are yet to be elaborated.

5 Finite trace semantics via coinduction

In this paper we take (finite) traces as our semantics for systems. It is with respect to trace semantics that soundness and completeness of forward/backward simulations are shown. This section establishes the basics of trace semantics for systems by revisiting our previous work [5]. The main points are:

- a final coalgebra in the Kleisli category $\mathcal{K}\ell(T)$ is (interestingly) induced by an initial algebra in **Sets**;
- the principle of coinduction, when employed in $\mathcal{K}\ell(T)$, yields finite trace semantics for branching systems.

We also cite a fact from [2] about an order-theoretic property of a final coalgebra. Again in this section a monad T is $\mathcal{P}$ or $\mathcal{D}$ and F is a shapely functor.

The following result identifies a final coalgebra in the Kleisli categories.

Theorem 5.1 (Main theorem of [5]) *Let $\alpha : FA \overset{\cong}{\Rightarrow} A$ be an initial F-algebra in* **Sets***.*

1. *An initial $\overline{F}$-algebra in $\mathcal{K}\ell(T)$ is induced by α as $\eta_A \circ \alpha : FA \overset{\cong}{\Rightarrow} A$ in $\mathcal{K}\ell(T)$.*
2. *In $\mathcal{K}\ell(T)$, an initial $\overline{F}$-algebra and a final $\overline{F}$-coalgebra coincide. The latter is given as follows. We shall denote this coalgebraic structure map by ζ.*

$$\zeta = (\eta_A \circ \alpha)^{-1} = \eta_{FA} \circ \alpha^{-1} : \ A \xrightarrow{\cong} FA \qquad \text{in } \mathcal{K}\ell(T) \ . \qquad \square$$

As a corollary we obtain the final coalgebra semantics for an $\overline{F}$-coalgebra. Recall that such a coalgebra is a dynamics of a (T, F)-system.

Corollary 5.2 (Trace semantics for coalgebras, [5]) *Given an $\overline{F}$-coalgebra $X \overset{c}{\to} FX$ in $\mathcal{K}\ell(T)$, there exists a unique morphism tr_c which makes the following diagram commute. Here $\alpha : FA \overset{\cong}{\Rightarrow} A$ is an initial F-algebra in* **Sets***.*

$$\begin{tikzcd}
FX \arrow[r, dashed, "\overline{F}(\mathsf{tr}_c)"] & FA \\
X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{tr}_c"'] & A \arrow[u, "\cong", "\zeta \text{ (final)}"']
\end{tikzcd} \qquad (5) \qquad \square$$

The induced map $\mathsf{tr}_c : X \to A$ in $\mathcal{K}\ell(T)$, i.e. $\mathsf{tr}_c : X \to TA$ in **Sets**, in fact becomes what is usually called the finite trace map: it assigns to each state its "trace" in a suitable sense. The following examples show that the commutation of Diagram (5) actually amounts to standard and natural recursive definition of finite trace maps.

Example 5.3 (Non-deterministic automata) In the setting of Example 3.2, an initial F-algebra in **Sets** is carried by finite lists, or words, over Σ.

$$1 + \Sigma \times \Sigma^* \xrightarrow[\cong]{[\mathsf{nil}, \mathsf{cons}]} \Sigma^*$$

Now Diagram (5) commutes if and only if the function $\mathsf{tr}_c : X \to \mathcal{P}(\Sigma^*)$ satisfies the following conditions. For each $a \in \Sigma$ and $\sigma \in \Sigma^*$.

$$\begin{array}{rcl} \langle\rangle \in \mathsf{tr}_c(x) & \Longleftrightarrow & x \to \checkmark \ , \\ a \cdot \sigma \in \mathsf{tr}_c(x) & \Longleftrightarrow & \exists x' \in X. \ \ \big(\, x \xrightarrow{a} x' \ \wedge \ \sigma \in \mathsf{tr}_c(x') \, \big) \ . \end{array}$$

This is the standard recursive (or corecursive, if you like) definition of the *accepted languages* of non-deterministic automata. The language $\mathsf{tr}_c(x) \subseteq \Sigma^*$ is the set of all the linear-time behavior of x which eventually terminates within a finite number of steps (hence the name *finite* trace).

Example 5.4 (Probabilistic automata) Let us look at the example of a probabilistic automaton in Example 3.3. What is the "trace" of the state x of this system? A natural answer, as suggested in [8], is the probability subdistribution over lists on Σ:

$$\langle\rangle \mapsto \frac{1}{3}, \quad a \mapsto \frac{1}{3} \cdot \frac{1}{2}, \quad aa \mapsto \frac{1}{3} \cdot \frac{1}{2} \cdot \frac{1}{2}, \quad \cdots, \quad a^n \mapsto \frac{1}{3} \cdot \left(\frac{1}{2}\right)^{n-1} \cdot \frac{1}{2}, \quad \cdots \tag{6}$$

This is explained as follows. For the state x to output the list aa, it has to take the path of transitions: $x \xrightarrow{a} y \xrightarrow{a} y \to \checkmark$. This path occurs with the probability $\frac{1}{3} \cdot \frac{1}{2} \cdot \frac{1}{2}$.

This notion of "probabilistic trace" is again obtained via coinduction in the Kleisli category. Let us instantiate Diagram (5) with T and F in Example 3.3. The commutativity of the diagram amounts to the following (co)recursive definition of a function $\mathsf{tr}_c : X \to \mathcal{D}(\Sigma^*)$:

$$\mathsf{tr}_c(x) = \begin{bmatrix} \langle\rangle & \mapsto & c(x)(\checkmark) \\ a \cdot \sigma & \mapsto & \sum_{y \in X} c(x)(a,y) \cdot \mathsf{tr}_c(y)(\sigma) \end{bmatrix} \ .$$

Here the probability $c(x)(a,y) \cdot \mathsf{tr}_c(y)(\sigma)$ is for the event that x makes an a-move to y and then y yields the list σ as its trace. Taking the sum over all the possible successors y of x, we get a natural recursive definition of the probability with which x yields $a \cdot \sigma$ as its trace.

As an additional remark we point out that the subdistribution (6) sums up only to 2/3. The remaining 1/3 is for the path $x \xrightarrow{a} z \xrightarrow{a} z \xrightarrow{a} \cdots$: the probability for a^ω, or *livelock*. This entry $a^\omega \mapsto 1/3$ is absent in $\mathsf{tr}_c(x)$ because $\mathsf{tr}_c : X \to \mathcal{D}(\Sigma^*)$ is the *finite* trace. This also demonstrates why we use the subdistribution monad $\mathcal{D}$ instead of the distribution monad $\mathcal{D}_{=1}$: although the system can be described using $\mathcal{D}_{=1}$, we do not get tr_c of the type $X \to \mathcal{D}_{=1}(\Sigma^*)$.

Example 5.5 (Context-free grammar, [4]) Take $T = \mathcal{P}$ and a suitable F for context-free grammars. Then the trace map tr_c assigns to each non-terminal x the set of finite-depth parse trees generated by the context-free grammar c starting from x.

From a different point of view, the previous examples are seen as proofs that standard recursive definitions uniquely determine trace maps, due to the finality result in Corollary 5.2.

The trace map tr_c, being a morphism of coalgebras, automatically becomes a lax morphism of coalgebras. It is in fact characterized as the biggest one.

Proposition 5.6 (Trace map as the biggest lax morphism) *In the situation of Diagram (5), the trace map tr_c is the biggest one among the lax coalgebra morphisms from c to the final ζ.*

Dually, the trace map tr_c is the smallest one among the oplax coalgebra morphisms from c to the final ζ.

Proof. Although the proposition follows from a general result [2, Proposition 6.7], in this specific setting of a Kleisli category we can give another proof. It does not depend on the local continuity of $\overline{F}$ but only on its local monotonicity. This alternative proof is found in [6, Appendix 2]. □

So far the trace map induced by coinduction gives the semantics for a single state of a coalgebra. This is extended to the semantics of a (T, F)-system—a coalgebra with explicit start states—in the obvious way.

Definition 5.7 (Finite trace semantics of (T, F)-systems) Given a (T, F)-system $1 \xrightarrow{s} X \xrightarrow{c} FX$ in $\mathcal{K}\ell(T)$, its *finite trace* (or just *trace*) $\mathsf{tr}_{(s,c)}$ is the following composite in $\mathcal{K}\ell(T)$.

$$\begin{tikzcd}
FX \arrow[r, dashed, "\overline{F}(\mathsf{tr}_c)"] & FA \\
X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{tr}_c"] & A \arrow[u, "\zeta"', "\cong"] \\
1 \arrow[u, "s"] \arrow[ur, bend right, "\mathsf{tr}_{(s,c)}"'] &
\end{tikzcd}$$

One can readily show that a morphism of systems preserves trace semantics.

6 Soundness and completeness theorems

In the last two sections we have built up the notions of (and some results on) forward/backward simulations and trace semantics, with a high level of genericity and abstraction. In this section we relate those materials—with the same genericity and abstraction—by proving soundness of $\sqsubseteq_{\mathbf{F}}, \sqsubseteq_{\mathbf{B}}, \sqsubseteq_{\mathbf{BF}}$ and completeness of $\sqsubseteq_{\mathbf{BF}}$ with respect to trace inclusion. This is the main technical result of this paper.

In the rest of this section we assume $1 \xrightarrow{s} X \xrightarrow{c} FX$ and $1 \xrightarrow{t} Y \xrightarrow{d} FY$ to be (T, F)-systems, where $T = \mathcal{P}$ or $\mathcal{D}$ and F is shapely.

Theorem 6.1 (Soundness of $\sqsubseteq_{\mathbf{F}}, \sqsubseteq_{\mathbf{B}}, \sqsubseteq_{\mathbf{BF}}$)

1. $(s, c) \sqsubseteq_{\mathbf{F}} (t, d) \implies \mathsf{tr}_{(s,c)} \sqsubseteq \mathsf{tr}_{(t,d)}$,
2. $(s, c) \sqsubseteq_{\mathbf{B}} (t, d) \implies \mathsf{tr}_{(s,c)} \sqsubseteq \mathsf{tr}_{(t,d)}$,
3. $(s, c) \sqsubseteq_{\mathbf{BF}} (t, d) \implies \mathsf{tr}_{(s,c)} \sqsubseteq \mathsf{tr}_{(t,d)}$.

Proof. 1. By definition of $\sqsubseteq_{\mathbf{F}}$ we have a forward simulation $f : Y \to X$. In particular we have in $\mathcal{K}\ell(T)$,

$$\begin{tikzcd}
FY \arrow[r, "\overline{F}f"] & FX \arrow[r, dashed, "\overline{F}(\mathsf{tr}_c)"] & FA \\
Y \arrow[u, "d"] \arrow[r, "f"'] \arrow[ur, phantom, "\sqsupseteq"] & X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{tr}_c"'] \arrow[ur, phantom, "="] & A \arrow[u, "\zeta"', "\cong"] \arrow[u, phantom, shift left=6, "\text{(final)}"]
\end{tikzcd}$$

where the coinduction diagram appears on the right. This shows that the arrow $\mathsf{tr}_c \circ f$ is a lax coalgebra morphism from d to the final coalgebra. Since the trace map is the biggest lax coalgebra morphism (Proposition 5.6), we have $\mathsf{tr}_c \circ f \sqsubseteq \mathsf{tr}_d$. This inequality is combined with f's condition on start states.

$$\mathsf{tr}_{(s,c)} \;=\; \mathsf{tr}_c \circ s \;\sqsubseteq\; \mathsf{tr}_c \circ f \circ t \;\sqsubseteq\; \mathsf{tr}_d \circ t \;=\; \mathsf{tr}_{(t,d)}$$

This proves 1. Similar arguments prove 2.

3. The relation $\sqsubseteq_{\mathbf{BF}}$ is a relational composition $\sqsubseteq_{\mathbf{F}} \circ \sqsubseteq_{\mathbf{B}}$. We use 1. and 2. of the theorem and transitivity of the order $\sqsubseteq$ between arrows $1 \rightrightarrows A$. $\square$

Completeness—the converse of the soundness result above—does not hold for $\sqsubseteq_{\mathbf{F}}, \sqsubseteq_{\mathbf{B}}$ but does hold for the weaker notion of $\sqsubseteq_{\mathbf{BF}}$. For a restricted class of non-deterministic systems the completeness result is shown in [11,12].

Theorem 6.2 (Completeness of $\sqsubseteq_{\mathbf{BF}}$)

$$\mathsf{tr}_{(s,c)} \sqsubseteq \mathsf{tr}_{(t,d)} \quad \Longrightarrow \quad (s,c) \sqsubseteq_{\mathbf{BF}} (t,d) \ .$$

Proof. From a (T,F)-system (s,c), we construct its "canonical system" as

$$\begin{tikzcd}
1 \arrow[r, "\mathsf{tr}_{(s,c)}"] & A \arrow[r, "\zeta", "\cong"'] & FA
\end{tikzcd} \qquad \text{in } \mathcal{K}\ell(T) \ .$$

That is, the dynamics is the final $\overline{F}$-coalgebra and the start states map is the trace of the system. It is obvious by definition that the map tr_c is a morphism of systems from (s,c) to this canonical system (the left side of Diagram (7) below). We apply the same construction to (t,d) yielding the right side of the diagram. Then the assumption $\mathsf{tr}_{(s,c)} \sqsubseteq \mathsf{tr}_{(t,d)}$ fits in the lower middle of the diagram.

$$\begin{tikzcd}
FX \arrow[r, dashed, "\overline{F}(\mathsf{tr}_c)"] & FA & FY \arrow[l, dashed, "\overline{F}(\mathsf{tr}_d)"'] \\
X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{tr}_c"] & A \arrow[u, "\zeta"', "\cong"] & Y \arrow[u, "d"'] \arrow[l, dashed, "\mathsf{tr}_d"'] \\
 & 1 \arrow[ul, bend left, "s"] \arrow[ur, bend right, "t"'] \arrow[u, bend left, "\mathsf{tr}_{(s,c)}"] \arrow[u, bend right, "\mathsf{tr}_{(t,d)}"'] \arrow[u, phantom, "\sqsubseteq"] &
\end{tikzcd} \tag{7}$$

From this we have two diagrams of backward-forward simulations—like Diagram (4) in Definition 4.6—depending on our choice of the intermediate system.

$$\begin{tikzcd}
FX \arrow[r, dashed] & FA & FY \arrow[l, dashed] \\
X \arrow[u, "c"] \arrow[r, dashed] & A \arrow[u, "\cong"] & Y \arrow[u, "d"'] \arrow[l, dashed] \\
 & 1 \arrow[ul, bend left] \arrow[ur, bend right] \arrow[u, "\mathsf{tr}_{(s,c)}"] \arrow[ur, phantom, "\sqsubseteq"] &
\end{tikzcd} \quad \text{or} \quad \begin{tikzcd}
FX \arrow[r, dashed] & FA & FY \arrow[l, dashed] \\
X \arrow[u, "c"] \arrow[r, dashed] & A \arrow[u, "\cong"] & Y \arrow[u, "d"'] \arrow[l, dashed] \\
 & 1 \arrow[ul, bend left] \arrow[ur, bend right] \arrow[u, "\mathsf{tr}_{(t,d)}"'] \arrow[ul, phantom, "\sqsubseteq"] &
\end{tikzcd}$$

Either diagram shows $(s, c) \sqsubseteq_{\mathbf{BF}} (t, d)$. $\square$

Using the completeness result we can prove that the simulation relation $\sqsubseteq_{\mathbf{BF}}$—now equivalent to trace inclusion—is actually transitive.

Proposition 6.3 *The simulation relations $\sqsubseteq_{\mathbf{F}}, \sqsubseteq_{\mathbf{B}}$ and $\sqsubseteq_{\mathbf{BF}}$ are preorders. That is, they are reflexive and transitive.*

Proof. Except for transitivity of $\sqsubseteq_{\mathbf{BF}}$, the proof is straightforward. See [6, Propositions 6.3, 6.4]. $\square$

7 Conclusions and future work

We have developed a generic theory of branching state-based systems in terms of coalgebras in Kleisli categories. Notions such as forward/backward simulations and traces are defined and related via soundness and completeness results. Several illustrating examples suggest practical implications of this theory.

There are a number of issues on branching systems that remain to be elaborated in our generic framework. To name a few: composition of systems, compositionality of semantics, modal logic, preservation of logical formulas, infinite traces and internal actions.

As mentioned in Remark 3.4, systems with both non-deterministic and probabilistic branching do not fit in our general framework. There are many semantical questions (see e.g. [1]) about this combination of different branching: hopefully categorical approaches will contribute to clarify the picture.

More examples of types of systems to which our framework applies are to be found. For example, the author is interested in a probabilistic version of anonymous simulations [9] .

IOA Toolset [3] is a formal verification tool in which systems are described as I/O automata and analyzed using simulations. Now that its base theory is made generic, one might as well work on a generic version of the toolset itself.

Acknowledgments

Thanks are due to Chris Heunen, Bart Jacobs, Yoshinobu Kawabe, Koki Nishizawa, Ana Sokolova, Frits Vaandrager and Hiroshi Watanabe for helpful discussions.

References

1. L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices.* PhD thesis, Radboud Univ. Nijmegen, 2006.
2. M. Fiore. A coinduction principle for recursive data types based on bisimulation. *Inf. & Comp.*, 127(2):186–198, 1996.
3. S. Garland, N. Lynch, and M. Vaziri. *IOA: a language for specifying, programming, and validating distributed systems.* MIT Laboratory for Computer Science, 1997.

4. I. Hasuo and B. Jacobs. Context-free languages via coalgebraic trace semantics. In *Algebra and Coalgebra in Computer Science (CALCO'05)*, volume 3629 of *Lect. Notes Comp. Sci.*, pages 213–231. Springer, Berlin, 2005.
5. I. Hasuo, B. Jacobs, and A. Sokolova. Generic trace theory. In *Coalgebraic Methods in Computer Science (CMCS 2006)*, Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2006.
6. I. Hasuo. Generic forward and backward simulations. Technical report, Research Center for Verification and Semantics, National Institute of Advanced Industrial Science and Technology (AIST), Japan, 2006. `http://www.cs.ru.nl/~ichiro/papers/`.
7. J. Hughes and B. Jacobs. Simulations in coalgebra. *Theor. Comp. Sci.*, 327(1-2):71–108, 2004.
8. C. Jou and S. Smolka. Equivalences, congruences and complete axiomatizations for probabilistic processes. In *CONCUR'90*, volume 458 of *Lect. Notes Comp. Sci.*, pages 367–383. Springer-Verlag, 1990.
9. Y. Kawabe, K. Mano, H. Sakurada, and Y. Tsukada. Backward simulations for anonymity. In *International Workshop on Issues in the Theory of Security (WITS '06)*, 2006.
10. Y. Kinoshita and J. Power. Data refinement and algebraic structure. *Acta Informatica*, 36:693–719, 2000.
11. N. Klarlund and F. Schneider. Verifying safety properties using infinite-state automata. Technical Report 89-1039, Department of Computer Science, Cornell University, Ithaca, New York, 1989.
12. N. Lynch and F. Vaandrager. Forward and backward simulations. I. Untimed systems. *Inf. & Comp.*, 121(2):214–233, 1995.
13. J. Power and D. Turi. A coalgebraic foundation for linear time semantics. In *Category Theory and Computer Science*, number 29 in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 1999.
14. J. Rutten. Universal coalgebra: a theory of systems. *Theor. Comp. Sci.*, 249:3–80, 2000.
15. R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journ. Comput.*, 2(2):250–273, 1995.
16. A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. PhD thesis, TU Eindhoven, 2005.
17. R. van Glabbeek, S. Smolka, and B. Steffen. Reactive, generative, and stratified models of probabilistic processes. *Inf. & Comp.*, 121:59–80, 1995.
18. S.H. Wu, S.A. Smolka, and E.W. Stark. Composition and behaviors of probabilistic I/O automata. *Theor. Comp. Sci.*, 176(1–2):1–38, 1997.