Tracing Anonymity with Coalgebras

Ichiro Hasuo

ISBN: 978-90-9022827-3
IPA Dissertation Series 2008-09

Typeset with $\LaTeX\,2_\varepsilon$
Cover design by Kumiko Arai
Printed by PrintPartners Ipskamp B.V.

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics). The author was employed at Radboud University Nijmegen and funded by the NWO *Pionier* project for security and correctness of computer programs in April 2004–April 2007. Since May 2007 he has been employed at Kyoto University, Japan.

Tracing Anonymity with Coalgebras

een wetenschappelijke proeve op het gebied
van de Natuurwetenschappen, Wiskunde en Informatica

Proefschrift

ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de rector magnificus prof. mr. S.C.J.J. Kortmann,
volgens besluit van het College van Decanen
in het openbaar te verdedigen op maandag 10 maart 2008
om 13.30 uur precies

door

Ichiro Hasuo

geboren op 27 september 1978
te Omuta, Japan

Promotor:

prof. dr. B.P.F. Jacobs

Manuscriptcommissie:

prof. dr. J. Adámek	Technical University of Braunschweig, Germany
prof. dr. J.H. Geuvers	
prof. dr. M. Hasegawa	Kyoto University, Japan
dr. D. Pavlović	Univ. of Oxford, UK and Kestrel Institute, US
prof. dr. J.J.M.M. Rutten	CWI and Free University Amsterdam

Preface

This thesis could have never been finished without the teaching, support, encouragement, advice, criticism and help of many. It is a hopeless endeavor to list them all. The first to name is of course my PhD supervisor Bart Jacobs who not only went beyond perfection in his supervision but also supported me as my best friend. The starting sentences of this preface (which I found hard to come up with) are shamelessly stolen from his book [61].

I have had great support from my friends in academia. Thanks to Jiří Adámek, Herman Geuvers, Masahito Hasegawa, Duško Pavlović and Jan Rutten for examining this thesis as the reading committee members. I benefited from their criticism and encouragement. It was a lot of pleasure to work with the following coauthors: David Galindo, Flavio Garcia, Chris Heunen, Bart Jacobs, Yoshinobu Kawabe, Wolter Pieters, Peter van Rossum, Ana Sokolova and Tarmo Uustalu. During my PhD study I have made several visits and I thank the hosts for their hospitality, comments and teaching: Yoshinobu Kawabe, Joseph Kiniry, Bartek Klin, Naoki Kobayashi, Alexander Kurz, Catuscia Palamidessi, John Power, Ana Sokolova, Yde Venema and Hiroshi Watanabe. Special thanks to Dirk Pattinson who led me to the theory of coalgebras through his lectures at NASSLLI'03. In my MSc study I learned a lot from Ryo Kashima, Izumi Takeuti and Hiroshi Watanabe. Without their encouragement, I would never have come to the Netherlands for my PhD study. Besides, I owe much to comments from and discussion with: Kazuyuki Asada, David Costa, Stefan Milius, Milad Niqui, Koki Nishizawa, Shinya Katsumata, Clemens Kupke, Christian Kissig, Erik Poll, Bas Spitters, and Frits Vaandrager.

I had great support also from outside academia. Thanks to Désirée Hermans, Maria van Kuppeveld, Mirèse Willems from Radboud University Nijmegen and Keiko Nishimori from Kyoto University for their administrative support. Coming to the Netherlands was my first time living abroad. It is probably not the easiest thing but

I enjoyed it a lot. Special thanks to all the friends and colleagues here in Nijmegen, especially to Jisk Attema, Igor Di Marco, Ab Polman and Ellie Polman who shared houses with me.

Finally, some words go to my family. Thanks to Kumiko Arai for the cover design and everything. Thanks to Yoshio Egashira, Hiromi Hasuo and Takeshi Hasuo, for just everything.

Ichiro Hasuo
Nijmegen, January 2008

Contents

Chapter 1

Introduction

The aim of this thesis is to obtain better understanding of the nature of *computer systems*; in other words, to develop a *mathematical theory of computer systems. Coalgebras* are employed in our expedition as mathematical presentations of computer systems.

1.1 Computer systems

Computer systems—information-processing systems consisting of (possibly multiple) computers—are pervasive in the modern world, playing crucial roles in every aspect of human activities. However, computer systems are also known to be extremely error-prone: newspapers are full of incidents caused by failure of computer systems and their enormous consequences. Given the fact that computer systems are ultimately created by human hands, it is amazing how rarely they behave as we, their creators, expect them to do. It must be the case that *we do not quite understand computer systems.*

To understand the nature of computer systems is therefore a fundamental challenge nowadays. This thesis is a step forward towards this goal. History tells us that clear understanding of something is very often brought by mathematics, a lesson which we shall follow. That is, we aim at a *mathematical theory of computer systems.*

1.2 Coalgebras as systems

In this thesis *coalgebras* are employed to formulate diverse phenomena in computer systems. Why coalgebras? We use them because the theory of coalgebras is one of the most promising candidates for a mathematical foundation for computer systems, equipped with both *ample applicability* and *mathematical simplicity.*

- *Applicability.* Our mathematical presentations of computer systems should be able to address diverse (theoretical) problems in computer systems and their solutions.
- *Simplicity.* Influential work in mathematics is very often marked with its simplicity (or “beauty” as one may even say). Mathematical simplicity is a necessity for further development of a theory; it is also a source of abstraction which brings wide applicability.

Hopefully this thesis will provide grounds for the claim about such qualifications of coalgebras, through:

- extending the application fields of coalgebras and demonstrating the notion’s potential applicability; and
- in each of the additional application fields, providing mathematically clear formulations of phenomena which have been identified as major obstacles in getting computer systems right. In doing so we exploit mathematical simplicity of the notion of coalgebras.

We begin with the definition of coalgebras (which is so simple) and then proceed to the first few examples that illustrate the idea “coalgebras as systems.”

1.2.1 Definition

The notion of coalgebras (*co*-algebras) is the categorical dual of that of algebras. Hence one can say: the notion of coalgebras is as primitive as that of algebras.

1.2.1 Definition (Coalgebras)**.** Let $\mathbb{C}$ be a category and $F : \mathbb{C} \to \mathbb{C}$ be an endofunctor. An *F-coalgebra* is a pair of an object $X \in \mathbb{C}$ (called its *carrier*) and an arrow (its *structure arrow*)

$$\begin{tikzcd} FX \\ X \arrow[u, "c"] \end{tikzcd} \quad \text{in } \mathbb{C}\ .$$

The functor F is called the coalgebra’s *signature functor*.

An F-coalgebra will be called simply a coalgebra when the signature functor F is obvious from the context.

1.2.2 Remark (Functor coalgebras vs. Eilenberg-Moore coalgebras)**.** The notion of coalgebras just introduced is sometimes called *functor coalgebras*, in contrast to *Eilenberg-Moore coalgebras* for comonads (see e.g. [96]). It should be noted that often in a categorical setting the latter notion is referred to (simply) as coalgebras too. Throughout this thesis the word “coalgebra” is reserved for the former notion.

1.2.2 First examples

In the last few decades, coalgebras have found their use as mathematical, categorical presentations of *state-based systems* such as automata; computer systems are also instances of state-based systems. There are several introductory articles to the view of "coalgebras as state-based systems," such as [65, 66, 86, 108, 114]; we refer to them for a historical account of the field, extensive investigation of examples, a list of references, and so on. Here we present a few small, elementary examples to provide an intuition.

$$\begin{tikzcd}
\{\text{coalgebras}\} \\
\{\text{state-based systems}\} \arrow[u, dashed, "\text{abstraction}"'] \\
\{\text{computer systems}\} \arrow[u, phantom, "\rotatebox{90}{$\subseteq$}" description]
\end{tikzcd}$$

Coalgebras are state-based systems. Let us take the category **Sets** of sets and functions as the base category $\mathbb{C}$. This means that the objects (such as X and FX) are sets; and arrows (such as $X \xrightarrow{c} FX$) are set-theoretic functions. As an endofunctor $F : \mathbf{Sets} \to \mathbf{Sets}$, let us take $F = \Sigma \times \_$, that is

$$FX = \Sigma \times X$$

where Σ is a fixed set.

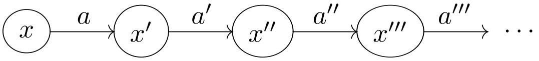
In this case, a coalgebra $X \xrightarrow{c} FX$ is identified with a function

$$\begin{tikzcd}
\Sigma \times X & \text{such as} & (a, x') \\
X \arrow[u, "c"] & & x \arrow[u, mapsto]
\end{tikzcd} .$$

The key idea is that we can think of

- an element $x \in X$ as a *state*, hence a set X as a *state space*;
- an application of a function c as a one-step *transition* or *dynamics*.

Indeed, if $c(x)$ is a pair (a, x') as above, we can interpret its first component $a \in \Sigma$ as the *output* of the state x when it makes a transition; its second component $x' \in X$ as the *next state* of x. Therefore we can draw the following graphical representation of the coalgebra.

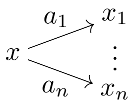


(1.1)

To summarize, a coalgebra $X \xrightarrow{c} \Sigma \times X$ is a state-based system which, by making a transition, outputs a symbol in Σ and moves to another state.

Different types of systems by different endofunctor F. A coalgebra $X \xrightarrow{c} \Sigma \times X$ (that we have just observed) denotes what is usually called a *stream automaton*: its behavior—i.e. what we observe by continuously running the system—is an (infinite) stream over the alphabet Σ, such as $aa'a''a''' \cdots$ in (1.1).

Can we represent, as coalgebras, other kinds of state-based systems say *labeled transition systems* (LTSs)? A state of LTSs has a set of possible transitions, each of which is labeled with an element in a fixed alphabet Σ.



This is different from stream automata in which a state has its unique output and successor; an LTS exhibits *non-deterministic* branching. For a technical reason we shall focus on *finitely branching* LTSs in the sequel.[1] This means that a state can have only a finite number of possible outgoing transitions.

We can express such a system as a function

$$\begin{tikzcd} \mathcal{P}_{\mathrm{fin}}(\Sigma \times X) \\ X \arrow[u, "c"] \end{tikzcd} \qquad \text{such as} \qquad \begin{tikzcd} \{\,(a_1, x_1),\ \dots,\ (a_k, x_k)\,\} \\ x \arrow[u, mapsto] \end{tikzcd} \qquad ,$$

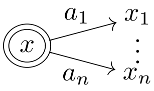
where $\mathcal{P}_{\mathrm{fin}}$ denotes the operation of taking the set of all finite subsets:

$$\mathcal{P}_{\mathrm{fin}} X = \{u \subseteq X \mid u \text{ is finite}\} \enspace .$$

Therefore (finitely branching) LTSs are coalgebras for the base category $\mathbb{C} = \mathbf{Sets}$ and the signature functor $F = \mathcal{P}_{\mathrm{fin}}(\Sigma \times \_)$.

Non-deterministic automata are yet another kind of state-based systems. They are almost the same as LTSs but each state has an additional 2-value attribute ($\mathbf{T}$ or $\mathbf{F}$) denoting whether it is one of the terminating states or not.[2] Such systems can be represented as coalgebras for $\mathbb{C} = \mathbf{Sets}$ and now $F = 2 \times \mathcal{P}_{\mathrm{fin}}(\Sigma \times \_)$, where $2 = \{\mathbf{T}, \mathbf{F}\}$ is a 2-element set. For example,

$$\begin{tikzcd} 2 \times \mathcal{P}_{\mathrm{fin}}(\Sigma \times X) \\ X \arrow[u, "c"] \end{tikzcd} \qquad \text{such as} \qquad \begin{tikzcd} \big(\,\mathbf{T},\ \{(a_1, x_1), \dots, (a_k, x_k)\}\,\big) \\ x \arrow[u, mapsto] \end{tikzcd} \quad , \ \text{meaning}$$



A much greater variety of systems allows representation as coalgebras using different functors F; see e.g. [108].

1.3 Theory of coalgebras

We have sketched how state-based systems can be presented using the abstract notion of coalgebras. What do we gain from this presentation? Isn't it just a fancy (or scary)

[1] If we allow arbitrary degree of branching, a final coalgebra (which we explain in Section 1.3) does not exist due to the size problem. See e.g. [66].

[2] We disregard the finiteness conditions (on a state space X and on an alphabet Σ) which are usually present in the definition of non-deterministic automata.

way of presenting something one already knows, as if it were something one does not know?

We claim that it is not the case. One ground for our claim is the following fact (which has been recognized in the last couple of decades): some natural mathematical constructions on coalgebras have indeed important meanings in the theory of state-based systems. Among them, *morphisms of coalgebras* and the principle of *coinduction*—the two notions which we elaborate in this section—play important roles in this thesis.

1.3.1 Morphisms of coalgebras as behavior-preserving maps

A *morphism* from one coalgebra to another is an arrow between their carriers which "preserves coalgebraic structures."

1.3.1 Definition (Morphisms of coalgebras)**.** Let $X \overset{c}{\to} FX$ and $Y \overset{d}{\to} FY$ be two coalgebras (in the base category $\mathbb{C}$). A *morphism* from the former coalgebra c to the latter d is an arrow $f : X \to Y$ in $\mathbb{C}$ such that the following diagram commutes.

$$\begin{tikzcd}
FX \arrow[r, "Ff"] & FY \\
X \arrow[u, "c"] \arrow[r, "f"'] & Y \arrow[u, "d"']
\end{tikzcd} \tag{1.2}$$

F-coalgebras and morphisms between them form a category which we denote by $\mathbf{Coalg}_F$. Therefore there is the following one-to-one correspondence:

$$\begin{array}{c}
\left(\begin{array}{c} FX \\ c\uparrow \\ X \end{array}\right) \xrightarrow{f} \left(\begin{array}{c} FY \\ d\uparrow \\ Y \end{array}\right) \quad \text{in } \mathbf{Coalg}_F \\
\hline\hline
X \xrightarrow{f} Y \quad \text{in } \mathbb{C}, \ \text{ such that } \begin{tikzcd}[ampersand replacement=\&]
FX \arrow[r, "Ff"] \& FY \\
X \arrow[u, "c"] \arrow[r, "f"'] \& Y \arrow[u, "d"']
\end{tikzcd}
\end{array}$$

The significance of morphisms of coalgebras is the fact that they represent *behavior-preserving maps* between systems. To illustrate this point, let us take the setting in our first example ($\mathbb{C} = \mathbf{Sets}$ and $F = \Sigma \times \_$), in which case coalgebras are stream automata. A morphism from $X \overset{c}{\to} \Sigma \times X$ to $Y \overset{d}{\to} \Sigma \times Y$ is a (set-theoretic) function $X \overset{f}{\to} Y$ such that the diagram

$$\begin{tikzcd}
\Sigma \times X \arrow[r, "\Sigma \times f"] & \Sigma \times Y \\
X \arrow[u, "c"] \arrow[r, "f"'] & Y \arrow[u, "d"']
\end{tikzcd}$$

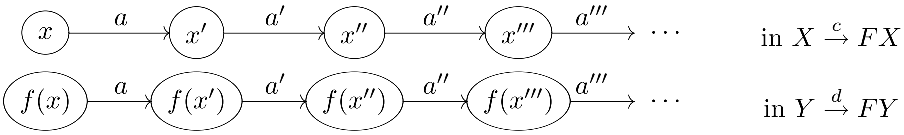
commutes. In order to examine what this commutativity means, let us denote the arrow c by

$$c \;=\; \langle \mathsf{out}_c, \mathsf{next}_c \rangle \quad : \quad X \longrightarrow \Sigma \times X \;\; ,$$

making the two components $\mathsf{out}_c : X \to \Sigma$ and $\mathsf{next}_c : X \to X$ explicit. Similarly we denote $d = \langle \mathsf{out}_d, \mathsf{next}_d \rangle$. Then the above commutativity is obviously equivalent to the following condition: for any "state" $x \in X$,

$$\mathsf{out}_c(x) = \mathsf{out}_d(f(x)) \quad \text{and} \quad f(\mathsf{next}_c(x)) = \mathsf{next}_d(f(x)) \;\; . \tag{1.3}$$

Its immediate consequence is that the "behavior" of a state x in the system $X \xrightarrow{c} FX$ coincides with that of $f(x)$ in $Y \xrightarrow{d} FY$. Here by the "behavior" of x we mean the infinite stream over Σ that arises by running the system $X \xrightarrow{c} FX$ continuously, starting from x.



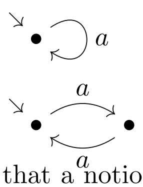
Indeed, the output $a \in \Sigma$ of $f(x)$ must be the same as that of x by the first equality of (1.3); moreover, if $x \xrightarrow{a} x'$ then the next state of $f(x)$ must be $f(x')$ by the second equality.

The slogan "morphisms of coalgebras as behavior-preserving maps" is true for other kinds of systems as well. For instance, for the other two examples of a signature functor F in Section 1.2.2 (namely functors for LTSs and non-deterministic automata), the notion of coalgebra morphisms coincides with the usual notion of *functional bisimulation* (i.e. functions whose graphs are bisimulation).

1.3.2 Behavior by coinduction

We just talked about an informal notion of *behaviors* of state-based systems. The behavior of a system is what we "observe" by running a system. We shall take the *black-box view* about our "observation": we (as outside observers) observe the actions executed by the system, but have no direct means of knowing which internal states are realizing those actions.

Or rather, we *do not want to care about* the internal mechanism which realizes our (black-box) observation. In fact, identifying two different systems with the same behavior—such as the two stream automata on the right yielding the same stream a^ω—is often a very useful technique in verification of state-based systems.



One of the nicest things about the abstract notion of coalgebras is that a notion of such "behaviors" can be elegantly formulated by a simple mathematical principle of *coinduction*, i.e. by final coalgebras.

1.3.2 Definition (Final coalgebras, coinduction)**.** A *final* coalgebra $Z \xrightarrow{\zeta} FZ$ is a final object in the category $\mathbf{Coalg}_F$.[3] This means that, for any F-coalgebra $X \xrightarrow{c} FX$, there exists a unique morphism from c to ζ.

$$\begin{tikzcd}
FX \arrow[r, dashed, "Ff"] & FZ \\
X \arrow[u, "c"] \arrow[r, dashed, "f"'] & Z \arrow[u, "\zeta"']
\end{tikzcd}$$

Making an explicit use of finality of a final coalgebra is often referred to as the principle of *coinduction.*

1.3.3 Remark (Usage of "coinduction")**.** In [65] the word *coinduction* means the *use* of the finality principle employed in a specific category $\mathbf{Coalg}_F$. We follow this view.[4] It is called *co*-induction because it is the categorical dual of *induction* which, in a categorical setting, refers to the use of the initiality principle for algebras.

Here are some typical usages of the word: the unique morphism f above is induced "by coinduction"; if g is another morphism from c to ζ then we conclude $f = g$ "by coinduction." Notice that finality—called coinduction in this setting—has two separate aspects: *existence* of an arrow and its *uniqueness.* In fact, coinduction in the former usage (exploiting the existence aspect) is sometimes called the coinduction *definition* principle; coinduction in the latter usage (exploiting the uniqueness aspect) is called the coinduction *proof* principle.

Coinduction as a uniform mathematical principle that assigns to a system its behavior is our principal tool in this thesis. The basic scenario is as follows. For the kind of state-based systems as specified by an endofunctor F,

- the set Z of all possible behaviors is itself found to carry an F-coalgebraic structure $Z \xrightarrow{\zeta} FZ$. It turns out that this ζ is a final coalgebra.
- Given an arbitrary coalgebra (i.e. a system) $X \xrightarrow{c} FX$, there exists a unique morphism from c to ζ, by the definition of final coalgebras.

$$\begin{tikzcd}
FX \arrow[r, dashed, "F(\mathsf{beh}(c))"] & FZ \\
X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{beh}(c)"'] & Z \arrow[u, "\zeta"']
\end{tikzcd} \tag{1.4}$$

The unique morphism thus induced is what assigns to (a state of) the coalgebra c its behavior, and is denoted by $\mathsf{beh}(c)$.

[3] In category theory, a final object is often referred to as a *terminal* object as well. Consequently, a final coalgebra is sometimes called a *terminal* coalgebra.

[4] In computer science the word *coinduction* can be used in a different, order-theoretic meaning, which refers to the use of greatest fixed points. See e.g. [118]. In fact, our categorical usage is a generalization of the other order-theoretic one, in the same way as categories are generalization of poet's.

It is illustrative to think of this scenario employed in the category **Sets**. Then the arrows are (set-theoretic) functions; the arrow $\mathsf{beh}(c) : X \to Z$ is therefore a function which carries each state $x \in X$ to an element $\mathsf{beh}(c)(x) \in Z$ which is the "behavior" of the state x of the system $X \xrightarrow{c} FX$.

This basic scenario works for a wide variety of systems, i.e. F-coalgebras for a variety of F. The notion of "behaviors" (represented by a final F-coalgebra $Z \xrightarrow{\zeta} FZ$) varies according to the choice of a functor F. Furthermore, as we see later in Chapter 2, coinduction can capture a different kind of "behaviors" if employed in a different category (than **Sets**). Let us look at some first (standard) examples.

Coinduction for stream automata A stream automaton is an F-coalgebra for $F = \Sigma \times \_$ in $\mathbb{C} = \mathbf{Sets}$. A natural notion of its "behavior" is the infinite stream over Σ that we observe by running the system continuously. The set of infinite streams over Σ (which we denote by Σ^ω) indeed carries the following F-coalgebra structure.

$$\begin{tikzcd}
\Sigma \times \Sigma^\omega & (\, a_0, \quad a_1 a_2 a_3 \ldots \,) \\
\Sigma^\omega \arrow[u, "\langle \mathsf{hd}, \mathsf{tl} \rangle"] & a_0 a_1 a_2 \ldots \arrow[u, mapsto]
\end{tikzcd}$$

Here the function hd returns the *head* element of a stream; the function tl returns the *tail* stream. Now it is not hard to prove the following.

1.3.4 Proposition (Coinduction for stream automata). *Let $\mathbb{C} = \mathbf{Sets}$ and $F = \Sigma \times \_$. For any F-coalgebra $X \xrightarrow{c} \Sigma \times X$, there exists a unique coalgebra morphism*

$$\left(\begin{tikzcd} \Sigma \times X \\ X \arrow[u, "c"] \end{tikzcd} \right) \dashrightarrow \left(\begin{tikzcd} \Sigma \times \Sigma^\omega \\ \Sigma^\omega \arrow[u, "\langle \mathsf{hd}, \mathsf{tl} \rangle"] \end{tikzcd} \right) .$$

In other words, there exists a unique function $\mathsf{beh}(c) : X \to \Sigma^\omega$ that makes the following coinduction diagram commute.

$$\begin{tikzcd}
\Sigma \times X \arrow[r, dashed, "\Sigma \times \mathsf{beh}(c)"] & \Sigma \times \Sigma^\omega \\
X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{beh}(c)"'] & \Sigma^\omega \arrow[u, "\langle \mathsf{hd}, \mathsf{tl} \rangle"']
\end{tikzcd} \tag{1.5}$$

The diagram is an instance of (1.4); hence the coalgebra $\langle \mathsf{hd}, \mathsf{tl} \rangle : \Sigma^\omega \to \Sigma \times \Sigma^\omega$ is a final coalgebra. □

Commutativity of the diagram (1.5) can be rephrased into the following equations (as we did in (1.3)). Recall that we write $c = \langle \mathsf{out}_c, \mathsf{next}_c \rangle : X \to \Sigma \times X$.

$$\mathsf{hd}\big(\mathsf{beh}(c)(x)\big) = \mathsf{out}_c(x) \quad \text{and} \quad \mathsf{tl}\big(\mathsf{beh}(c)(x)\big) = \mathsf{beh}(c)\big(\mathsf{next}_c(x)\big). \tag{1.6}$$

Putting it graphically,

$$x \xrightarrow{a} x' \quad \text{in} \quad \begin{array}{c} \Sigma \times X \\ c\uparrow \\ X \end{array} \qquad \Longrightarrow \qquad \mathsf{beh}(c)(x) = a\ \underbrace{a'a''a'''\cdots}_{\mathsf{beh}(c)(x')} \ .$$

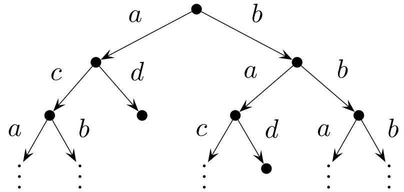
The equations (1.6) are therefore a natural definition of the infinite stream (the "behavior") $\mathsf{beh}(c)(x)$ induced by a state x of a stream automaton c, presented in a conventional manner. The coinduction diagram (1.5)—which is equivalent to the equations (1.6)—is a categorical presentation of the definition. Our point here is that the latter allows generalization to different kinds of systems, due to the abstraction inherent in category theory.

Coinduction for LTSs Coinduction as a mathematical principle for assigning a "behavior" can be applied also to (finitely branching) LTSs and non-deterministic automata.

For the functor $F = \mathcal{P}_{\mathrm{fin}}(\Sigma \times \_)$—for which a coalgebra is a finitely branching LTS—a final coalgebra indeed exists [4, 10]. In the sequel we shall sketch what it looks like, without going too much into details.

First, the carrier set of the final coalgebra—which we denote by $\mathsf{STree}^{\Sigma}_{\sim}$—consists of equivalence classes of so-called *synchronization trees* over Σ, modulo *bisimilarity.*

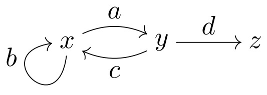
- *Synchronization trees* [98] over Σ are finitely branching, (possibly) infinite-depth trees whose edges are labeled with elements of Σ, such as the following.

  The nodes after d-edges (without any successors) are understood to have the branching degree 0.

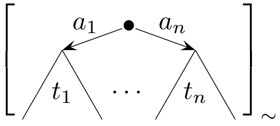
- *Bisimilarity* is one of the standard notions of process equivalence, originally introduced in [107]. We shall denote the bisimilarity equivalence (on synchronization trees) by $\sim$.

A synchronization tree emerges when we "unfold" an LTS and depict the possible execution paths of the LTS, starting from a specific state. For example, the above synchronization tree arises by unfolding the following LTS, starting from the state x.



The structure arrow of the final coalgebra—which we denote by dstr—is the following one, “destructing” a tree into its immediate subtrees.

$$\mathsf{STree}^{\Sigma}_{\sim} \xrightarrow{\mathsf{dstr}} \mathcal{P}_{\mathrm{fin}}(\Sigma \times \mathsf{STree}^{\Sigma}_{\sim})$$



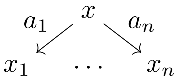
$$\longmapsto \; \{\, (a_1, [t_1]_{\sim}),\; \ldots,\; (a_n, [t_n]_{\sim}) \,\}$$

One can prove that this function dstr is well-defined, because of the definition of bisimilarity $\sim$.

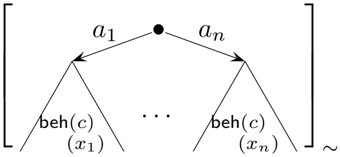
Now, the following coinduction diagram in **Sets** is an instantiation of (1.4) in the current LTS setting.

$$\begin{tikzcd}
\mathcal{P}_{\mathrm{fin}}(\Sigma \times X) \arrow[r, dashed, "\mathcal{P}_{\mathrm{fin}}(\Sigma \times \mathsf{beh}(c))"] & \mathcal{P}_{\mathrm{fin}}(\Sigma \times \mathsf{STree}^{\Sigma}_{\sim}) \\
X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{beh}(c)"'] & \mathsf{STree}^{\Sigma}_{\sim} \arrow[u, "\mathsf{dstr}"']
\end{tikzcd} \tag{1.7}$$

In this way a finitely branching LTS (represented by a coalgebra c) is assigned its “behavior” $\mathsf{beh}(c) : X \to \mathsf{STree}^{\Sigma}_{\sim}$. The commutativity of the diagram (1.7) boils down to the following condition.



in the system c $\Longrightarrow$ $\mathsf{beh}(c)(x) =$



Therefore the “behavior” $\mathsf{beh}(c)(x) \in \mathsf{STree}^{\Sigma}_{\sim}$ is (the equivalence class of) the unfolding of the LTS c, starting from the state x. The following points should be noted here.

- Information on internal states $(x, x_1, x_2, \ldots)$ is not present in the behavior, because in a synchronization tree nodes are not labeled.
- Moreover, two states which are *bisimilar* (according to the standard definition) are mapped to the same element of $\mathsf{STree}^{\Sigma}_{\sim}$, because elements of $\mathsf{STree}^{\Sigma}_{\sim}$ are equivalence classes modulo $\sim$.

Therefore we have illustrated the following result.

1.3.5 Proposition (Coinduction captures bisimilarity)**.** *There exists a final coalgebra*

$$\mathsf{dstr} \;:\; \mathsf{STree}^{\Sigma}_{\sim} \longrightarrow \mathcal{P}_{\mathrm{fin}}(\Sigma \times \mathsf{STree}^{\Sigma}_{\sim})$$

for the functor $\mathcal{P}_{\mathrm{fin}}(\Sigma \times \_)$ *on* **Sets***.*

Moreover, the equivalence relation induced by coinduction coincides with the usual notion of bisimilarity. That is, if $x \in X$ is a state of a coalgebra $X \xrightarrow{c} \mathcal{P}_{\mathrm{fin}}(\Sigma \times X)$ and $y \in Y$ is a state of $Y \xrightarrow{d} \mathcal{P}_{\mathrm{fin}}(\Sigma \times Y)$, then we have

$$\mathsf{beh}(c)(x) = \mathsf{beh}(d)(y) \quad \textit{in } \mathsf{STree}^{\Sigma}_{\sim} \quad \Longleftrightarrow \quad x \textit{ and } y \textit{ are bisimilar} \;. \qquad \square$$

We refer to the second point of the proposition as: *coinduction captures bisimilarity;* or *a final coalgebra is fully abstract with respect to bisimilarity.*

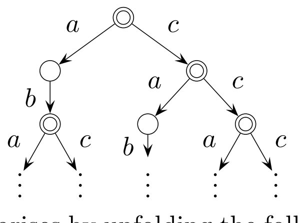
Coinduction in Sets captures bisimilarity It turns out that the last point in the previous paragraph "coinduction captures bisimilarity" holds not only for the specific endofunctor $F = \mathcal{P}_{\mathrm{fin}}(\Sigma \times \_)$, but also for a variety of endofunctors F on **Sets**.

Specifically, for an arbitrary endofunctor $F : \mathbf{Sets} \to \mathbf{Sets}$ (i.e. for systems other than LTSs) we can define the generalized notion of *bisimulations* using suitable spans. Besides this categorical formulation of bisimulations, the notion of *bisimilarity* also arises categorically, as the universal one among all the bisimulations. It can be shown (see e.g. [114]) that this generic notion of bisimilarity is captured by coinduction (in the sense of Proposition 1.3.5), if the functor F *preserves weak pullbacks*, a condition satisfied by most functors on **Sets** of our interest.

Therefore here comes another slogan: *coinduction in* **Sets** *captures bisimilarity.* Note that, although this slogan is valid for a variety of endofunctors F, the base category $\mathbb{C}$ is fixed to be **Sets**. In Chapter 2 we will see that *trace equivalence*—another standard process equivalence which is coarser than bisimilarity—is captured by coinduction in a different universe, namely in a Kleisli category instead of in **Sets**.

1.3.6 Example (Non-deterministic automata)**.** Our arguments on the final coalgebra for the functor $F = \mathcal{P}_{\mathrm{fin}}(\Sigma \times \_)$ (modeling finitely branching LTSs) remain valid if we consider the functor $F = 2 \times \mathcal{P}_{\mathrm{fin}}(\Sigma \times \_)$ (modeling finitely branching non-deterministic automata).

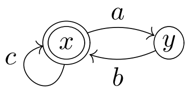
The final coalgebra for $F = 2 \times \mathcal{P}_{\mathrm{fin}}(\Sigma \times \_)$ consists of synchronization trees—whose nodes are now labeled by **T** or **F**—modulo a suitable notion of bisimilarity. The following is an example of such a synchronization tree.



◎ : labeled by **T**,
i.e. a terminating state

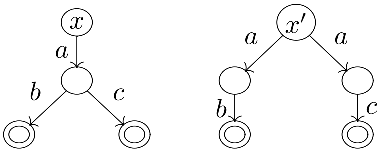
○ : labeled by **F**,
i.e. a non-terminating state

This tree arises by unfolding the following non-deterministic automaton starting from the state x. In other words, the state x's "behavior" assigned by coinduction is the above tree (modulo bisimilarity).



One thing to note here is that bisimilarity (captured by coinduction) is different from *language equivalence*, another standard notion of equivalence for non-deterministic automata. A state of a non-deterministic automaton determines a subset of the set Σ^* of words in Σ, called its *accepted language*—$(ab + c)^*$ for the state x in the above example. Two states of non-deterministic automata are *language equivalent* if their accepted languages are the same.

Language equivalence is as a relation strictly coarser than bisimilarity, as the following classic example of "language equivalent but not bisimilar states" demonstrates (see e.g. [129]).



(1.8)

Indeed, the two states x and x' have the same accepted language $\{ab, ac\}$; but x' cannot simulate x's ability that "after making an a-move, the system can still choose between b- and c-moves," which makes x *not* bisimilar to x'. How to capture this notion of language equivalence by coinduction—besides bisimilarity—will be our theme in Chapter 2.

1.4 Thesis outline

The following table summarizes the correspondence we have observed, between coalgebraic constructs and basic concepts in the theory of computer systems.

	system	behavior-preserving map	behavior
coalgebraically	coalgebra $\begin{tikzcd} FX \\ X \arrow[u] \end{tikzcd}$	morphism of coalgebras $\begin{tikzcd} FX \arrow[r, "Ff"] & FY \\ X \arrow[u] \arrow[r, "f"'] & Y \arrow[u] \end{tikzcd}$	by coinduction $\begin{tikzcd} FX \arrow[r, dashed] & FZ \\ X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{beh}(c)"'] & Z \arrow[u, "\text{final}"'] \end{tikzcd}$

(1.9)

The examples so far have been accommodated in the category **Sets**, where the "behavior" captured by coinduction coincides with bisimilarity.

This thesis selects some noted phenomena in computer systems and provides clear understanding of them, by identifying fundamental mathematical structures hidden behind. Our mathematical formulations of those phenomena are based on coalgebras; thus this thesis extends the (existing) theory of coalgebras (such as table (1.9)) in several directions.

Let us now take a brief look at the issues about computer systems that we will address in later chapters. In each chapter, we focus on one specific issue.

Chapter 2. Trace semantics via coinduction We have seen that coinduction in **Sets** captures a “behavior” modulo bisimilarity. However, there are standard notions of behavioral equivalence other than bisimilarity, such as language equivalence for non-deterministic automata. So one may ask: can we capture other notions of equivalence also by coinduction?

In Chapter 2 we present a partial, yet positive answer to this question. We present a coalgebraic framework for capturing *trace semantics*; language equivalence is an instance of trace semantics. Informally speaking, the difference from bisimilarity is the fact that in trace semantics inner branching structures are abstracted away; see (1.8).

Our contributions are summarized as follows.

- **Trace semantics via coinduction in $\mathcal{K}\ell(T)$** We identify coinduction in a *Kleisli category* $\mathcal{K}\ell(T)$ as a mathematical principle for capturing trace semantics. That is, the coinduction diagram in $\mathcal{K}\ell(T)$

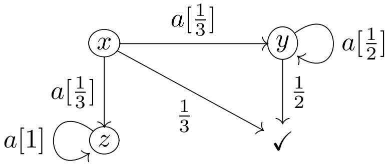
$$\begin{tikzcd} FX \arrow[r, dashed, "F(\mathsf{beh}(c))"] & FZ \\ X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{beh}(c)"'] & Z \arrow[u, "\zeta"'] \end{tikzcd}$$

  yields an arrow $\mathsf{beh}(c)$ which assigns to each state $x \in X$ its trace semantics. Therefore the theory of coalgebras, when employed in $\mathcal{K}\ell(T)$, can be thought of as the theory of state-based systems “modulo trace semantics.” Intuitively, it is the use of $\mathcal{K}\ell(T)$ as the base category which abstracts away inner branching structures.

- **Identification of a final coalgebra in $\mathcal{K}\ell(T)$** A final coalgebra in $\mathcal{K}\ell(T)$ turns out to coincide with an initial algebra in **Sets** for the corresponding endofunctor. This (seemingly surprising) technical result follows from

  - a suitable adjunction lifting which lifts an initial algebra in **Sets** to an initial algebra in $\mathcal{K}\ell(T)$; and
  - the *initial algebra-final coalgebra coincidence* in $\mathcal{K}\ell(T)$ which results from the limit-colimit coincidence due to a suitable order-enriched structure of $\mathcal{K}\ell(T)$.

Here we gain remarkable genericity by using abstract categorical terms. Notice that the framework has a parameter T—a monad on **Sets**—which in fact specifies the type of branching. For one choice, systems have *non-deterministic branching* as in LTSs; for another choice systems come to have *probabilistic branching.* The coinduction framework works also in the latter setting, which enables us to capture “trace semantics” for probabilistic systems, such as the following. For the state x of the probabilistic system



"trace semantics" is the probability (sub)distribution

$$\left[\begin{array}{l} \langle\rangle \mapsto \frac{1}{3}, \quad a \mapsto \frac{1}{3}\cdot\frac{1}{2}, \quad a^2 \mapsto \frac{1}{3}\cdot\frac{1}{2}\cdot\frac{1}{2}, \quad \cdots \\ a^n \mapsto \frac{1}{3}\cdot\left(\frac{1}{2}\right)^n, \quad \cdots \end{array}\right]$$

that tells the probability with which the word a^n occurs before the system hits the termination symbol ✓.

To summarize, this chapter extends the application fields of coalgebras by adding trace semantics to the list of process semantics that are captured by coinduction. From a different point of view, "trace semantics" (for various kinds of systems, such as non-deterministic vs. probabilistic) is given a uniform mathematical description, namely as coinduction in a Kleisli category.

The material in Chapter 2 is based on joint work [43, 44, 47] with Bart Jacobs and Ana Sokolova. Useful comments from Jiří Adámek, Chris Heunen, Stefan Milius, Tarmo Uustalu and the anonymous referees of these papers are gratefully acknowledged.

Chapter 3. Generic forward and backward simulations In this chapter we continue our investigation into coalgebras in a Kleisli category $\mathcal{K}\ell(T)$. We have shown that coinduction in $\mathcal{K}\ell(T)$ gives trace semantics. What about the other coalgebraic constructs in the table (1.9), namely morphisms of coalgebras? How can we interpret them in $\mathcal{K}\ell(T)$?

It turns out that two relaxed versions of morphisms of coalgebras—*lax* and *oplax* morphisms—have significant meanings in the study of computer systems, namely as *forward* and *backward simulations* between systems.

$$\begin{tikzcd} FX \arrow[r, "Ff"] & FY \\ X \arrow[u, "c"] \arrow[r, "f"'] \arrow[ur, phantom, "\sqsupseteq"] & Y \arrow[u, "d"'] \end{tikzcd} \qquad \begin{tikzcd} FX \arrow[r, "Ff"] & FY \\ X \arrow[u, "c"] \arrow[r, "f"'] \arrow[ur, phantom, "\sqsubseteq"] & Y \arrow[u, "d"'] \end{tikzcd} \tag{1.10}$$

lax coalgebra morphism
= forward simulation

oplax coalgebra morphism
= backward simulation

The partial order $\sqsubseteq$ in the diagram refers to the natural notion of "more behavior" associated with the branching type in question—such as set inclusion for non-deterministic branching. Therefore this chapter, combined with the previous Chapter 2, establishes the view of the theory of coalgebras in $\mathcal{K}\ell(T)$ as the *generic theory of traces and simulations.*

The theory of traces and simulations has been extensively studied due to its significance in system verification; see e.g. [94]. A typical application scenario that involves traces and simulations is as follows.

- We have two systems at hand, $\mathcal{S}$ (called a *specification*) and $\mathcal{I}$ (called an *implementation*).
- The specification $\mathcal{S}$ is a simple system and is known to satisfy a certain desired property P. Let us assume that P is a *safety* property, meaning: a certain (unwanted) phenomenon never occurs in any execution trace of the system. Our goal is to establish that the implementation system $\mathcal{I}$ also satisfies the safety property P.
- To that goal, it suffices for us to show

  $$\text{(trace semantics of } \mathcal{I}) \;\sqsubseteq\; \text{(trace semantics of } \mathcal{S}) \enspace , \tag{1.11}$$

  which means for example that the former trace set is included in the latter (when systems are non-deterministic). Indeed, since we already know that none of the execution traces of the specification system $\mathcal{S}$ exhibits the unwanted phenomenon, from the above "trace inclusion" (1.11) we can conclude that neither the implementation $\mathcal{I}$ exhibits the unwanted phenomenon. Hence $\mathcal{I}$ is shown to satisfy the safety property P.
- In order to show the trace inclusion (1.11), we search for either a forward or a backward simulation from $\mathcal{I}$ to $\mathcal{S}$. We do so because traces and forward/backward simulations are related via a so-called *soundness theorem*: for two systems $\mathcal{X}$ and $\mathcal{Y}$,

  $$\begin{aligned} &\exists \text{ (forward or backward simulation from } \mathcal{X} \text{ to } \mathcal{Y}) \\ &\quad\Longrightarrow \quad \text{(trace sem. for } \mathcal{X}) \;\sqsubseteq\; \text{(trace sem. for } \mathcal{Y}) \enspace . \end{aligned}$$

In this way we prove $\mathcal{I}$'s satisfaction of the trace-based property P, essentially by finding a simulation. Such a simulation-based method is often useful because, in many applications, establishing trace inclusion (1.11) by directly computing the trace semantics (which takes arbitrarily many steps into account) is much harder than checking that a relation forms a forward/backward simulation (which only involves one-step transitions).

Therefore the soundness theorem—relating simulations and trace semantics—is an essential piece in the theory of traces and simulations. Indeed, our main result in this chapter is a *generic soundness theorem* which relates coalgebraic trace semantics (via coinduction, Chapter 2) and coalgebraic forward/backward simulations in (1.10). It is proved once for all, using abstract categorical terms.

Here again we gain ample genericity by using abstract categorical terms. In particular, the framework covers both non-deterministic systems and probabilistic ones. Assume that we are interested in some (exotic) kind of systems, no matter if they are non-deterministic or probabilistic. When those systems allow suitable coalgebraic modeling (which happens reasonably often), our coalgebraic framework provides a definition of forward/backward simulations tailored for this kind of systems as an instantiation of (1.10). This definition is guaranteed to be a “useful” one, because for this definition of simulations the soundness theorem comes *for free.*

From a viewpoint of a coalgebra-theorist, in Chapters 2 and 3 we have established the view:

> theory of coalgebras in a Kleisli category
> as
> generic theory of traces and simulations

which is summarized in the following table of correspondences.

	system	simulation		trace semantics
		forward	backward	
in $\mathcal{K}\ell(T)$	coalgebra	lax morphism	oplax morphism	coinduction
	$\begin{tikzcd} FX \\ X \arrow[u] \end{tikzcd}$	$\begin{tikzcd} FX \arrow[r, "Ff"] & FY \\ X \arrow[u] \arrow[r, "f"'] \arrow[ur, phantom, "\sqsupseteq"] & Y \arrow[u] \end{tikzcd}$	$\begin{tikzcd} FX \arrow[r, "Ff"] & FY \\ X \arrow[u] \arrow[r, "f"'] \arrow[ur, phantom, "\sqsubseteq"] & Y \arrow[u] \end{tikzcd}$	$\begin{tikzcd} FX \arrow[r, dashed] & FZ \\ X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{beh}(c)"'] & Z \arrow[u, "\mathsf{final}"'] \end{tikzcd}$

(1.12)

This is a new way of understanding the basic table (1.9), besides the standard one (namely, coalgebras in **Sets** are systems modulo bisimilarity). In this way we have extended the application fields of coalgebras.

The material in Chapter 3 has been presented in [46]. Part of the 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, in March 2006. Their hospitality as well as helpful comments from Chris Heunen, Bart Jacobs, Yoshinobu Kawabe, Koki Nishizawa, Ana Sokolova, Frits Vaandrager and Hiroshi Watanabe are gratefully acknowledged.

Chapter 4. Probabilistic anonymity (case study) The table (1.12) summarizes the view of *theory of coalgebras in $\mathcal{K}\ell(T)$ as theory of traces and simulations.* As already mentioned, what is remarkable in this table is the fact that non-deterministic systems and probabilistic systems are given the same, uniform treatment. In particular, the shift from non-deterministic branching to probabilistic one is just a matter of replacing the parameter T. That is to say, *anything we can (coalgebraically) do in a non-deterministic setting, we can also do in a probabilistic setting* (by replacing T).

We will exploit this genericity in this chapter which is devoted to a case study of verifying anonymity properties of network protocols. *Online anonymity* is an increasingly important property, at this age when most human activities involve communication on the Internet. Advances in information communication technology have made exchange of enormous amount of information a daily matter, from which we benefit vastly. However, it is the same technology that poses threats to our privacy: so much information on us is on the Internet and retrieving it has never been an easier task.

Therefore formalization and verification of anonymity properties have attracted increasing attention in the computer security community. In this chapter we set our starting point at one definition of anonymity introduced in [119]—as a property of the *trace* set of an automaton—and a *simulation*-based proof method [71, 72] for this notion of anonymity. These prior results are all obtained in a non-deterministic setting.

However, an important role of *probability* has been claimed recently by many authors in the field. Non-deterministic anonymity is *qualitative*: each participant of the protocol is either "suspicious" or "not suspicious at all," but nothing in between. In contrast, in a probabilistic notion of anonymity we talk about *quantitative* suspicion, a fact which is essential in modeling e.g. those anonymizing protocols which use probabilistic mechanisms for disguising identities.

The main contribution in this chapter is hence transferring the non-deterministic simulation-based proof method for anonymity—introduced in [71, 72]—to a probabilistic setting. In its course we fully exploit the coalgebraic theory developed in previous chapters: we obtain the definition of suitable "probabilistic simulations" by instantiating the coalgebraic definition and the soundness theorem (which comes for free) plays an essential part in the proof method.

The material in Chapter 4 is based on joint work [50, 51] with Yoshinobu Kawabe and Hideki Sakurada, carried out during the author's stay at NTT Communication Science Laboratories in September–October 2006 and April 2007. Helpful comments from Kostas Chatzikokolakis, Tom Chothia, Ken Mano, Catuscia Palamidessi, Peter van Rossum, Ana Sokolova, Yasuaki Tsukada and the anonymous referees for the paper [50] are gratefully acknowledged.

Chapter 5. Concurrency and the microcosm principle (foundational study) In this chapter we address the issue of *concurrency* in computer systems. Concurrency is about running multiple systems in parallel, possibly involving communication between them. Concurrency is everywhere at this age when almost all the computers are mutually connected via the Internet. At the same time, the exponentially growing complexity of systems with concurrency makes verification of such systems a fundamental challenge in computer science.

Our special attention is to the *compositionality* property: informally it means that the behavior of a composed system $\mathcal{C} \parallel \mathcal{D}$ is determined by the behavior of $\mathcal{C}$ and that of $\mathcal{D}$. The significance of compositionality in the study of computer systems is that it supports *modular* verification methods. In modular verification, correctness of a composed system $\mathcal{C}_1 \parallel \cdots \parallel \mathcal{C}_n$ is established using correctness of each component $\mathcal{C}_i$.

We start with the following—seemingly plausible—formulation of "coalgebraic compositionality." The operator beh is the one induced by coinduction; see (1.9).

$$\mathsf{beh}\left(\begin{array}{c} FX \\ c\uparrow \\ X \end{array} \,\Bigg\|\, \begin{array}{c} FY \\ d\uparrow \\ Y \end{array}\right) = \mathsf{beh}\left(\begin{array}{c} FX \\ c\uparrow \\ X \end{array}\right) \Bigg\| \,\mathsf{beh}\left(\begin{array}{c} FY \\ d\uparrow \\ Y \end{array}\right) \tag{1.13}$$

However, a closer look at the equation raises some questions. Most notably, the two operators $\|$ on each side of the equation have in fact different domains: the one on the left has a type

$$\mathbf{Coalg}_F \times \mathbf{Coalg}_F \xrightarrow{\|} \mathbf{Coalg}_F \ , \quad \text{composing coalgebras as systems;}$$

while the one on the right has a type

$$Z \times Z \xrightarrow{\|} Z \ , \quad \text{composing “states” of the final coalgebra } Z \text{ as behaviors.}$$

Moreover, these two domains ($\mathbf{Coalg}_F$ and Z) are nested: the latter is an object of the former (which is a category). Therefore what we observe here is a phenomenon that the same algebraic structure—namely a binary operator $\|$ possibly with some equational properties such as associativity—interpreted in two separate yet nested domains. This is the mathematical structure of our interest in this chapter.

Although examples of such phenomena abound in computer science—think of concatenation of deterministic automata and of regular languages—there does not seem to be much emphasis on them. However in mathematics the phenomenon of such nested algebraic structures has been long known; it is called the *microcosm principle* by Baez and Dolan [9]. We take this abstract, mathematical principle in itself as an interesting topic to investigate.

$$\begin{tikzcd}
\mathbb{L} \arrow[r, bend left=50, "\mathbf{1}", ""{name=U, below}] \arrow[r, "\mathbb{C}"', ""{name=D, above}] & \mathbf{CAT}
\arrow[from=U, to=D, phantom, "\Downarrow X"]
\end{tikzcd}$$

In this chapter—starting with the above observation on parallel composition and compositionality—we present a mathematically rigorous formalization of the microcosm principle. The formalization is 2-categorical and looks like the diagram on the right. An algebraic theory to be interpreted twice is presented categorically as a *Lawvere theory* $\mathbb{L}$; an outer model (the category $\mathbf{Coalg}_F$ in the above example) is a product-preserving functor $\mathbb{C}$; and an inner model (the object Z above) is a *lax* natural transformation X. Here $\mathbf{1}$ denotes the constant functor into the category $\mathbf{1}$ with one object and one arrow.

Based on this formalization, we turn our attention back to coalgebraic settings. As our main result we prove a generic compositionality result regarding an $\mathbb{L}$-structure on coalgebras, for an arbitrary algebraic theory $\mathbb{L}$. This generic result instantiates to the equation (1.13) when $\mathbb{L}$ is suitably chosen. Therefore our expedition in this chapter establishes mathematical foundation for modular verification of systems with concurrency.

Additionally, we investigate the relationship between the general framework for parallel composition (in this chapter) and the material in the previous chapters: the former indeed yields “compositionality” results for trace semantics and simulations. Specifically, our main result in this chapter (generic compositionality for behavior by coinduction) immediately yields compositionality of trace semantics which is “behavior by coinduction” in $\mathcal{K}\ell(T)$ (Chapter 2). Moreover, in relation to coalgebraic simulations (Chapter 3), we show that the forward/backward similarity relations are compositional under suitable assumptions. Let us write $c \sqsubseteq_{\mathbf{F}} d$ (“c is forward-simulated by d”) if there exists a forward simulation from a coalgebra c to another d. Then we have

$$\begin{array}{c} FX \\ c\uparrow \\ X \end{array} \sqsubseteq_{\mathbf{F}} \begin{array}{c} FY \\ d\uparrow \\ Y \end{array} \quad\Longrightarrow\quad \left(\begin{array}{c} FX \\ c\uparrow \\ X \end{array} \middle\| \begin{array}{c} FV \\ e\uparrow \\ V \end{array} \right) \sqsubseteq_{\mathbf{F}} \left(\begin{array}{c} FY \\ d\uparrow \\ Y \end{array} \middle\| \begin{array}{c} FV \\ e\uparrow \\ V \end{array} \right) .$$

We obtain a similar result for backward similarity as well.

We consider the material in Chapter 5 as a starting point of our research program on *concurrency in coalgebra*. The material is based on joint work [45] with Bart Jacobs and Ana Sokolova. Helpful comments from Kazuyuki Asada, John Baez, Masahito Hasegawa, Bill Lawvere, Duško Pavlović, John Power and the participants of CALCO-jnr workshop including Alexander Kurz are gratefully acknowledged.

1.5 Information for reading

Chapters 2–4 form a single thread, investigating coalgebras in a Kleisli category and applying them in a case study. The topic of Chapter 5 is outside this thread; although we also describe a relationship to earlier chapters, Chapter 5 can be read as an independent material.

This thesis contains no more introductory treatment of the standard theory of coalgebras than we have already had. See [65, 66, 86, 108, 114] for more details.

Nor have we any introductory material for basic category theory. For Chapters 2 and 3, the needed categorical background is available from the references at each point or from one of the aforementioned standard references on coalgebras. Chapter 4 does not have many prerequisites as it is a case study. Chapter 5 is mathematically more demanding compared to the previous chapters. Although it is meant to be self-contained, the reader may feel more comfortable being familiar with monoidal categories [96, Chapter VII]; Lawvere theories [82]; and with basic higher-dimensional category theory [19, Chapter 7].

Chapter 2

Trace semantics via coinduction

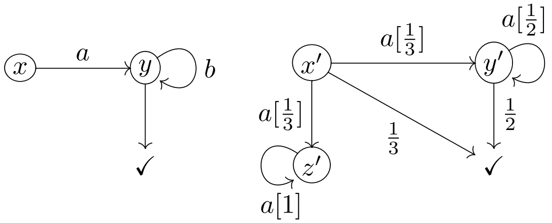
Trace semantics has been defined for various kinds of state-based systems, notably with different forms of branching such as non-determinism vs. probability. In this chapter we claim to identify one underlying mathematical structure behind these "trace semantics," namely coinduction in a Kleisli category. This claim is based on our technical result that, under a suitably order-enriched setting, a final coalgebra in a Kleisli category is given by an initial algebra in the category **Sets**. Formerly the theory of coalgebras has been employed mostly in **Sets** where coinduction yields a finer process semantics of bisimilarity. Therefore we extend the application fields of coalgebras, providing a new instance of the principle "process semantics via coinduction."

2.1 Overview

Trace semantics is a commonly used semantic relation for reasoning about state-based systems. Trace semantics for labeled transition systems is found on the coarsest edge of the linear time-branching time spectrum [39]. Moreover, trace semantics is defined for a variety of systems, among which are probabilistic systems [122]. Our claim in this chapter is: these various forms of "trace semantics" are instances of a general construction, namely coinduction in a Kleisli category.

2.1.1 "Trace semantics" in various contexts

First we motivate our contribution through examples of various forms of "trace semantics." Think of the following three state-based, branching systems.



A context-free grammar
(for Peano Arithmetic)

Terminal symbols:	$\mathsf{0}, \mathsf{s}$
Non-terminal symbol:	$\mathbf{T}$
Generation rules:	
$\mathbf{T} \to \mathsf{0}$	
$\mathbf{T} \to \mathsf{s}\mathbf{T}$	

(2.1)

- The first one is a non-deterministic system with a special state ✓ denoting successful termination. To its state x we can assign its *trace set*:

$$\mathsf{tr}(x) = \{a, ab, abb, \dots\} = ab^* \ , \tag{2.2}$$

that is, the *set* of the possible linear-time behaviors (namely words) that can arise through an execution of the system.[1] In this case the trace set $\mathsf{tr}(x)$ is also called the *accepted language*; formally it is defined (co)recursively by the following equations. For an arbitrary state x,

$$\begin{array}{rcl} \langle\rangle \in \mathsf{tr}(x) & \Longleftrightarrow & x \to \checkmark \\ a \cdot \sigma \in \mathsf{tr}(x) & \Longleftrightarrow & \exists y.\ (\, x \xrightarrow{a} y \ \wedge \ \sigma \in \mathsf{tr}(y)\,) \end{array} \tag{2.3}$$

Here $\langle\rangle$ denotes the empty word; $\sigma = a_1 a_2 \dots a_n$ is a word.

- The second system has a different type of branching, namely probabilistic branching. Here $x' \xrightarrow{a[1/3]} y'$ denotes: at the state x', a transition to y' which outputs a occurs with probability 1/3. Now, to the state x', we can assign its *trace distribution*:

$$\mathsf{tr}(x) = \left[\begin{array}{llll} \langle\rangle \mapsto \frac{1}{3}, & a \mapsto \frac{1}{3}\cdot\frac{1}{2}, & a^2 \mapsto \frac{1}{3}\cdot\frac{1}{2}\cdot\frac{1}{2}, & \cdots \\ a^n \mapsto \frac{1}{3}\cdot\left(\frac{1}{2}\right)^n, & \cdots & & \end{array}\right] \ , \tag{2.4}$$

that is, the probability distribution over the set of linear-time behaviors.[2] Its formal (corecursive) definition is as follows.

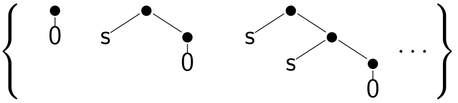
$$\begin{array}{rcl} \mathsf{tr}(x)(\langle\rangle) & = & \mathsf{Pr}(x \to \checkmark) \ , \\ \mathsf{tr}(x)(a\cdot\sigma) & = & \sum_{y \in X} \mathsf{Pr}(x \xrightarrow{a} y) \cdot \mathsf{tr}(y)(\sigma) \ , \end{array} \tag{2.5}$$

where $\mathsf{Pr}(\dots)$ denotes the probability of a transition.

[1] The infinite trace ab^ω is out of our scope here: we will elaborate this point later in Section 2.4.2.
[2] Here again, we do not consider the infinite trace $a^\omega \mapsto 1/3$.

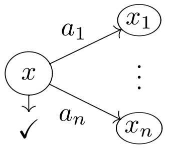
– The third example can be thought of as a state-based system, with non-terminal symbols as states. It is non-deterministic because a state $\mathbf{T}$ has two possible transitions. It is natural to call the following set of parse-trees its "trace semantics."

$\mathsf{tr}(\mathbf{T}) =$

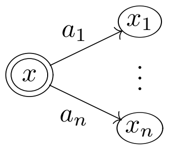


It is again a set of "linear-time behaviors" as in the first example, although the notion of linear-time behaviors is different here. Linear-time behaviors—that is, what we observe after we have resolved all the non-deterministic branchings in the system—are now parse-trees instead of words.

2.1.1 Remark. Notice that the first system in (2.1) can be thought of as a (finitely branching) non-deterministic automaton as described in Chapter 1. This is by identifying states which have a transition to termination ✓, with terminating states ◎ in non-deterministic automata. That is,



if and only if



. (2.6)

We will come back to this point later in Remark 2.2.2.

2.1.2 Coalgebras and coinduction

In recent years the theory of *coalgebras* has emerged as the "mathematics of state-based systems" [65, 66, 114]. In the categorical theory of coalgebras, an important definition/reasoning principle is *coinduction*: a system (identified with a coalgebra $c : X \to FX$) is assigned a unique morphism $\mathsf{beh}(c)$ into a final coalgebra.

$$\begin{tikzcd}
FX \arrow[r, dashed, "F(\mathsf{beh}(c))"] & FZ \\
X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{beh}(c)"'] & Z \arrow[u, "\cong"', "\text{final}"]
\end{tikzcd}$$

The success of coalgebras is largely due to the fact that, when **Sets** is taken as the base category, final coalgebra semantics is fully abstract with respect to the conventional notion of *bisimilarity*: for states x and y of coalgebras $X \xrightarrow{c} FX$ and $Y \xrightarrow{d} FY$,

$$\mathsf{beh}(c)(x) = \mathsf{beh}(d)(y) \qquad \Longleftrightarrow \qquad x \text{ and } y \text{ are bisimilar.}$$

This is the case for a wide variety of systems (i.e. for a variety of functors F), hence *coinduction in* **Sets** *captures bisimilarity.*

However, there is not so much work so far that captures other behavioral equivalences (coarser than bisimilarity) by the categorical principle of coinduction. The current work—capturing trace semantics by coinduction in a Kleisli category—therefore extends the application fields of the theory of coalgebras.

2.1.3 Our contributions

Our technical contributions are summarized as follows. Assume that T is a monad on **Sets** which has a suitable order structure; we shall denote its Kleisli category by $\mathcal{K}\ell(T)$.

– *Trace semantics via coinduction in a Kleisli category.* Commutativity of the coinduction diagram

$$\begin{tikzcd} \overline{F}X \arrow[r, dashed, "\overline{F}(\mathsf{tr}(c))"] & \overline{F}Z \\ X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{tr}(c)"'] & Z \arrow[u, "\cong"', "\text{final}"] \end{tikzcd} \qquad \text{in} \quad \begin{array}{l} \mathcal{K}\ell(T), \\ \text{the Kleisli category for } T \end{array} \tag{2.7}$$

is shown to be equivalent to the conventional recursive definition of trace semantics such as (2.3) and (2.5). This is true for both trace set semantics (for non-deterministic systems) and trace distribution semantics (for probabilistic systems). The induced arrow $\mathsf{tr}(c)$ thus gives (conventional) trace semantics for a system c.

– *Identification of a final coalgebra in a Kleisli category.* We show that

$$\boxed{\begin{array}{c} \text{an initial algebra in } \textbf{Sets} \\ \text{coincides with} \\ \text{a final coalgebra in } \mathcal{K}\ell(T). \end{array}}$$

In particular, a final coalgebra in **Rel** is an initial algebra in **Sets**, because the category **Rel** of sets and relations is a Kleisli category for a suitable monad. This coincidence happens in the following two steps:

- an initial algebra in **Sets** lifts to an initial algebra in a Kleisli category, due to a suitable adjunction-lifting result;
- in a Kleisli category we have *initial algebra-final coalgebra coincidence.* Here we use the classical result by Smyth and Plotkin [127], namely *limit-colimit coincidence* which is applicable in a suitably order-enriched category.

Note the presence of two parameters in (2.7): a monad T and an endofunctor F, both on **Sets**. The monad T specifies the *branching type* of systems. We have three leading examples:[3]

- the powerset monad $\mathcal{P}$ modeling *non-deterministic* or *possibilistic* branching;
- the subdistribution monad $\mathcal{D}$

$$\mathcal{D}X = \{d : X \to [0,1] \mid \sum_{x \in X} d(x) \le 1\}$$

  modeling *probabilistic* branching; and
- the lift monad $\mathcal{L} = 1 + (\_)$ modeling system with *exception* (or *deadlock*, *non-termination*).

The functor F specifies the *transition type* of systems: our understanding of "transition type" shall be clarified by the following examples.

- In labeled transition systems (LTSs) with explicit termination—no matter if they are non-deterministic or even probabilistic—a state either
  - terminates ($x \to \checkmark$), or
  - outputs one symbol and moves to another state ($x \xrightarrow{a} x'$),

  in one transition. This "transition type" is expressed by the functor $FX = 1 + \Sigma \times X$, where Σ is the output alphabet and $1 = \{\checkmark\}$.
- In context-free grammars (CFGs) as state-based systems, a state evolves into a sequence of terminal and non-terminal symbols in a transition. The functor

$$FX = (\Sigma + X)^*$$

  with Σ being the set of terminal symbols, expresses this transition type.

Clear separation of branching and transition types is essential in our generic treatment of trace semantics. The transition type F determines the set of linear-time behaviors (which is in fact given by an initial F-algebra in **Sets**). We model a system by a coalgebra $X \xrightarrow{c} \overline{F}X$ in the Kleisli category $\mathcal{K}\ell(T)$—see (2.7)—where $\overline{F}$ is a suitable lifting of F in $\mathcal{K}\ell(T)$. By the definition of a Kleisli category we will easily see the following bijective correspondence.

$$\frac{X \xrightarrow{\quad c \quad} \overline{F}X \ \text{ in } \mathcal{K}\ell(T)}{X \xrightarrow{\quad c \quad} TFX \ \text{ in } \mathbf{Sets}}$$

Hence our system—a *function* of the type $X \to TFX$—first resolves a branching of type T and then makes a transition of type F. Many branching systems allow such representation so that our generic coalgebraic trace semantics applies to them.

[3] Other examples include the monad $X \mapsto (\mathbb{N} \cup \{\infty\})^X$ for multisets, the monad $X \mapsto [0,\infty]^X$ for real valuations, and the monad $X \mapsto \mathcal{P}(M \times \_)$ with a monoid M for timed systems (cf. [75]). These monads can be treated in a similar way as our leading examples. We leave out the details.

2.1.4 Testing and trace semantics

Since the emergence of the theory of coalgebras, the significance of *modal logics* as specification languages has been noticed by many authors. This is exemplified by the slogan in [85]: 'modal logic is to coalgebras what equational logic is to algebras'. Inspired by coalgebras on Stone spaces and the corresponding modal logic, recent developments [17, 18, 79, 80, 83, 87, 109] have identified the following situation as the essential mathematical structure underlying modal logics for coalgebras.

$$\begin{tikzcd} \mathbb{C}^{\mathrm{op}} \arrow[loop left, "F^{\mathrm{op}}"] \arrow[r, bend left, "P"] \arrow[r, phantom, "\top"] & \mathbb{A} \arrow[l, bend left, "S^{\mathrm{op}}"] \arrow[loop right, "M"] \end{tikzcd} \quad \text{together with} \quad MP \overset{\delta}{\Longrightarrow} PF^{\mathrm{op}}$$

In fact, it is noticed in [109] that such a situation not only hosts a modal logic but also a more general notion of *testing* (in the sense of [39, 130], also called *testing scenarios*). Therefore we shall call the above situation a *testing situation*.

In the last technical section of the chapter we investigate coalgebraic trace semantics for the special case $T = \mathcal{P}$ (modeling non-determinism) from this testing point of view. First, we present some basic facts on testing situations, especially on the relationship between the induced *testing equivalence* and final coalgebra semantics. These two process equivalences are categorically presented as kernel pairs, which enables a fairly simple presentation of the theory of coalgebraic testing. In addition, we observe that the coinduction scheme in the Kleisli category $\mathcal{K}\ell(\mathcal{P})$ gives rise to a canonical testing situation, in which the set of tests is given by an initial F-algebra.

2.1.5 Organization of the chapter

In Section 2.2 we observe that a coalgebra in a Kleisli category is an appropriate "denotation" of a branching system, when we focus on trace semantics. In Section 2.3 we present our main technical result that an initial algebra in **Sets** yields a final coalgebra in $\mathcal{K}\ell(T)$. The relationship to axiomatic domain theory—which employs similar mathematical toolkit—is also discussed here. Section 2.4 presents some examples of the use of coinduction in $\mathcal{K}\ell(T)$ and argues that the coinduction principle is a general form of trace semantics. In Section 2.5 we review the preceding material from the testing point of view.

2.1.6 Bibliographical remarks

Capturing trace semantics via coinduction is the central theme of this chapter. One of the first attempts to do so is in [116], which focuses on (non-deterministic) labeled transition systems. The use of the Kleisli category $\mathcal{K}\ell(\mathcal{P})$ to accommodate non-deterministic systems is claimed in [111] and pursued in [62]. We follow this idea. In [111] they also present a scheme to lift a functor from **Sets** to a Kleisli category via a distributive law. Their scheme constitutes a part of ours presented in Section 2.2.2.

Talking about distributive laws, there is another separate line of studies in the theory of coalgebras which utilize distributive laws. It is about *bialgebras*, concerning about process terms as algebras, transition systems as coalgebras, and structural operational semantics—specifying how algebra meets coalgebra—expressed in terms of distributive laws. The seminal paper [134] commences this line of work (although some ideas already appear in [116]), later followed by [12, 67, 75, 78, 79]. The material later in Chapter 5 can be also seen as a further development along this line.

In this chapter the main technical ingredient that leads to identification of a final coalgebra in $\mathcal{K}\ell(T)$ is the theory of algebras and coalgebras in order-enriched categories. Originally it is used in the domain-theoretic semantics of programming languages, such as in [31, 34, 35, 106, 125, 127]. Besides, identification of an initial algebra and a final coalgebra in $\mathbf{Rel} \cong \mathcal{K}\ell(\mathcal{P})$—which is a special case of our general result—has played an important role in *relational refinement calculus* to reason about programs involving algebraic data types [16, 101]. Relations to these references are described in detail later in Section 2.3.3–2.3.4.

2.2 Coalgebras in a Kleisli category

In the study of coalgebras as "categorical presentations of state-based systems," the category **Sets** of sets and functions has been traditionally taken as a base category (see e.g. [65, 114]). An important fact in such a setting is that bisimilarity is often captured by coinduction.[4]

However, bisimilarity is not the only process equivalence. In some applications one would like coarser equivalences, for example in order to abstract away internal branching structures. One of such coarser semantics, which has been extensively studied, is *trace equivalence.* Trace equivalence appeared in Chapter 1 as language equivalence of non-deterministic automata. Another example is in the process algebra CSP [55] where trace semantics is used as an operational model. Trace equivalence is coarser than bisimilarity, as the classic example (1.8) of "trace-equivalent but not bisimilar" states demonstrates.

It is first noticed in [111] that the Kleisli category for the powerset monad is an appropriate base category for trace semantics for non-deterministic systems. This observation is pursued further in [43, 44, 47, 62]. In [42] it is recognized that the same is true for the subdistribution monad for probabilistic systems. The current chapter of this thesis provides a unified framework which yields those preceding results, in terms of **Cppo**-enrichment of a Kleisli category; see Section 2.2.3. In this section we first aim to justify the use of coalgebras in a Kleisli category.

[4] Non-examples include LTSs with unbounded branching degree; this is why we exclusively considered finitely branching systems in the introduction (Chapter 1). LTSs are modeled as coalgebras for $FX = \mathcal{P}(\Sigma \times X)$—using $\mathcal{P}$ instead of $\mathcal{P}_{\mathrm{fin}}$. Lambek's Lemma readily shows that this choice of F does not have a final coalgebra in **Sets**, because it would imply an isomorphism $Z \xrightarrow{\cong} \mathcal{P}(\Sigma \times Z)$ which is impossible for cardinality reasons.

2.2.1 Monads and Kleisli categories

Here we recall the relevant facts about monads and Kleisli categories. For simplicity we exclusively consider monads on **Sets**.

A *monad* on **Sets** is a categorical construct. It consists of

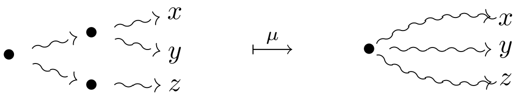
- an endofunctor T on **Sets**;
- a *unit* natural transformation $\eta : \mathrm{id} \Rightarrow T$, that is, a function $X \overset{\eta_X}{\to} TX$ for each set X satisfying a suitable naturality condition; and
- a *multiplication* natural transformation $\mu : T^2 \Rightarrow T$, consisting of functions $T^2X \overset{\mu_X}{\to} TX$ with X ranging over sets.

The unit and multiplication are required to satisfy the following compatibility conditions.

$$\begin{tikzcd}
TX \arrow[r, "\eta_{TX}"] \arrow[dr, "\mathrm{id}"'] & T^2X \arrow[d, "\mu_X"'] & TX \arrow[l, "T\eta_X"'] \arrow[dl, "\mathrm{id}"] \\
 & TX &
\end{tikzcd}
\qquad
\begin{tikzcd}
T^3X \arrow[r, "T\mu_X"] \arrow[d, "\mu_{TX}"'] & T^2X \arrow[d, "\mu_X"] \\
T^2X \arrow[r, "\mu_X"'] & TX
\end{tikzcd}$$

See [11, 96] for the details.

The monad structures play a crucial role in modeling "branching." Intuitively, the unit η embeds a non-branching behavior as a trivial branching (with only one possibility to choose). The multiplication μ "flattens" two successive branchings into one branching, abstracting away internal branchings:



(2.8)

The following examples of monads will illustrate how this flattening phenomenon is a crucial feature of trace semantics.

In this chapter we concentrate on the three monads mentioned in Section 2.1: $\mathcal{L}$, $\mathcal{P}$ and $\mathcal{D}$.

- The lift monad $\mathcal{L} = 1 + (\_)$—where we denote $1 = \{\perp\}$ with $\perp$ meaning *deadlock*—has a standard monad structure induced by a coproduct. For example, the multiplication $\mu^{\mathcal{L}}_X : 1 + 1 + X \to 1 + X$ carries $x \in X$ to itself and both $\perp$'s to $\perp$.
- The powerset monad $\mathcal{P}$ has a unit given by singletons and a multiplication given by unions. The monad $\mathcal{P}$ models non-deterministic branching: the "flattening" in (2.8) corresponds to the following application of the multiplication of $\mathcal{P}$.

$$\begin{tikzcd}
\mathcal{P}\mathcal{P}X \arrow[r, "\mu^{\mathcal{P}}_X"] & \mathcal{P}X \\
\{\,\{x,y\},\{z\}\,\} \arrow[r, mapsto] & \{x,y,z\}
\end{tikzcd}$$

The monad $\mathcal{P}$'s action on arrows (as a functor) needs mention as well. It is given by direct images: for $f : X \to Y$, the function $\mathcal{P}f : \mathcal{P}X \to \mathcal{P}Y$ carries a subset $u \subseteq X$ to the subset $\{f(x) \mid x \in u\} \subseteq Y$.

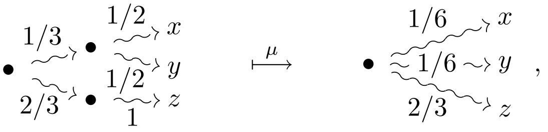
– The subdistribution monad $\mathcal{D}$ has a unit given by the *Dirac distributions.*

$$\begin{tikzcd} X \arrow[r, "\eta^{\mathcal{D}}_X"] & \mathcal{D}X \\ x \arrow[r, mapsto] & \begin{bmatrix} x \mapsto 1 & \\ x' \mapsto 0 & (\text{for } x' \neq x) \end{bmatrix} \end{tikzcd}$$

Its multiplication is given by multiplying the probabilities along the way. That is,

$$\mu^{\mathcal{D}}_X(\xi) = \lambda x. \sum_{d \in \mathcal{D}X} \xi(d) \cdot d(x) \ ,$$

which models "flattening" of the following kind.



that is,

$$\begin{bmatrix} \begin{bmatrix} x \mapsto 1/2 \\ y \mapsto 1/2 \end{bmatrix} & \mapsto & 1/3 \\ [z \mapsto 1] & \mapsto & 2/3 \end{bmatrix} \xmapsto{\mu} \begin{bmatrix} x \mapsto 1/6 \\ y \mapsto 1/6 \\ z \mapsto 2/3 \end{bmatrix} .$$

The monad $\mathcal{D}$'s action on arrows (as a functor) is given as a suitable adaptation of "direct images." Namely, for $f : X \to Y$, the function $\mathcal{D}f : \mathcal{D}X \to \mathcal{D}Y$ carries $d \in \mathcal{D}X$ to $[y \mapsto \sum_{x \in f^{-1}(y)} d(x)] \in \mathcal{D}Y$.

Given any monad T, its *Kleisli category* $\mathcal{K}\ell(T)$ is defined as follows. Its objects are the objects of the base category, hence sets in the current setting. An arrow $X \to Y$ in $\mathcal{K}\ell(T)$ is the same thing as an arrow $X \to TY$ in the base category, here **Sets**.

$$\frac{X \longrightarrow Y \ \text{ in } \mathcal{K}\ell(T)}{\overline{X \longrightarrow TY \ \text{ in } \mathbf{Sets}}}$$

Identities and composition of arrows are defined using the unit and the multiplication of T. Moreover, there is a canonical adjunction

$$\begin{tikzcd} \mathbf{Sets} \arrow[r, bend left, "J"] \arrow[r, phantom, "\perp"] & \mathcal{K}\ell(T) \arrow[l, bend left, "K"] \end{tikzcd} \tag{2.9}$$

in which the "Kleisli inclusion" functor J carries $X \xrightarrow{f} Y$ in **Sets** to $X \xrightarrow{\eta_Y \circ f} Y$ in $\mathcal{K}\ell(T)$; its right adjoint K carries

$$X \xrightarrow{f} Y \quad \text{in } \mathcal{K}\ell(T), \quad \text{that is, a function} \quad X \xrightarrow{f} TY \quad \text{in } \mathbf{Sets}$$

to $TX \xrightarrow{Tf} TTY \xrightarrow{\mu_Y} TY$ in **Sets**. See [11, 96] for details.

The relevance of Kleisli categories in our coalgebraic expedition is that a Kleisli category can be thought of as a category where the branching is implicit. For example, an arrow $X \to Y$ in the Kleisli category $\mathcal{K}\ell(\mathcal{P})$ is a function $X \to \mathcal{P}Y$ hence a "non-deterministic function." When $T = \mathcal{D}$, then by writing $X \to Y$ in the Kleisli category we mean a function with probabilistic branching. Moreover, composition of arrows in $\mathcal{K}\ell(T)$ is given by

$$X \xrightarrow{f} Y \xrightarrow{g} Z \quad \text{in } \mathcal{K}\ell(T) \quad = \quad X \xrightarrow{f} TY \xrightarrow{Tg} T^2 Z \xrightarrow{\mu_Z} TZ \quad \text{in } \mathbf{Sets};$$

that is, making one transition (by g) after another (by f), and then flattening (by μ_Z). For example, this general definition instantiates as follows when $T = \mathcal{D}$. Given $X \xrightarrow{f} Y \xrightarrow{g} Z$ in $\mathcal{K}\ell(\mathcal{D})$,

$$(g \circ f)(x)(z) = \textstyle\sum_{y \in Y} f(x)(y) \cdot g(y)(z) \ .$$

2.2.1 Remark. Our use of the *sub*-distribution monad instead of the distribution monad

$$\mathcal{D}_{=1}(X) = \{d : X \to [0,1] \mid \sum_{x \in X} d(x) = 1\}$$

needs some justification. Looking at the trace distribution (2.4), one sees that the probabilities add up only to 2/3 and not to 1; this is because the infinite trace (namely $a^\omega \mapsto 1/3$) are not present. Therefore in this example, although the state-based system can be modeled as a coalgebra in the category $\mathcal{K}\ell(\mathcal{D}_{=1})$, its trace semantics can only be expressed as an arrow in $\mathcal{K}\ell(\mathcal{D})$.

When a system is modeled as a coalgebra in $\mathcal{K}\ell(\mathcal{D})$, a state may have a (sub)-distribution over possible transitions which adds up to less than 1. In that case the missing probability can be understood as the probability for *deadlock*.

Technically, we use the monad $\mathcal{D}$ instead of $\mathcal{D}_{=1}$ because we need the minimum element (a *bottom*) so that the Kleisli category becomes **Cppo**-enriched (Theorem 2.3.3). A bottom is available for $\mathcal{D}$ as the zero distribution $[x \mapsto 0]$, but not for $\mathcal{D}_{=1}$.

2.2.2 Lifting functors by distributive laws

In this chapter a state-based system is presented as a coalgebra $X \to \overline{F}X$ in $\mathcal{K}\ell(T)$, where $\overline{F} : \mathcal{K}\ell(T) \to \mathcal{K}\ell(T)$ is a lifting of $F : \mathbf{Sets} \to \mathbf{Sets}$. This lifting $F \mapsto \overline{F}$ is equivalent to a *distributive law* $FT \Rightarrow TF$. The rest of this section elaborates on this point.

Various kinds of state-based, branching systems are expressed as a function of the form $X \xrightarrow{c} TFX$ with T a monad (for branching type) and F a functor (for transition type). The following examples are already hinted at in the previous section.

- For $T = \mathcal{P}$ and $F = 1 + \Sigma \times \_$, a function $X \xrightarrow{c} TFX$ is an LTS with explicit termination. For example, consider the following system

$$\begin{array}{rcl} X & \xrightarrow{\qquad c \qquad} & \mathcal{P}(1 + \Sigma \times X) \\ x & \longmapsto & \{\checkmark, (a_1, x_1), (a_2, x_2)\} \end{array}$$

where $\checkmark$ is the element of 1.[5] Then the state x can make three possible transitions, namely: $x \rightarrow \checkmark$ (successful termination), $x \xrightarrow{a_1} x_1$, and $x \xrightarrow{a_2} x_2$, when written in a conventional way.

- By replacing $T = \mathcal{P}$ by $\mathcal{D}$, but keeping F the same, we obtain a probabilistic system such as the one in the middle of (2.1). For example,

$$\begin{array}{rcl} X & \xrightarrow{\qquad c \qquad} & \mathcal{D}(1 + \Sigma \times X) \\ x' & \longmapsto & \begin{bmatrix} (a, y') \mapsto 1/3 \\ (a, z') \mapsto 1/3 \\ \checkmark \mapsto 1/3 \end{bmatrix} \end{array} .$$

- For $T = \mathcal{P}$ and $F = (\Sigma + \_)^*$, a function $X \xrightarrow{c} TFX$ is a CFG with Σ the terminal alphabet (but without finiteness conditions e.g. on the state space). See [43] for more details.

2.2.2 Remark. LTSs with explicit termination $\checkmark$ can be roughly seen as non-deterministic automata (Remark 2.1.1). Coalgebraically, it is because of the following isomorphism (which is natural in X).

$$2 \times \mathcal{P}_{\mathrm{fin}}(\Sigma \times X) \xrightarrow{\cong} \mathcal{P}_{\mathrm{fin}}(1 + \Sigma \times X) \tag{2.10}$$

It establishes an isomorphism between

$$(2 \times \mathcal{P}_{\mathrm{fin}}(\Sigma \times \_))\text{-coalgebras} \quad \text{and} \quad \mathcal{P}_{\mathrm{fin}}(1 + \Sigma \times \_)\text{-coalgebras} ,$$

hence between (finitely branching) non-deterministic automata and (finitely branching) LTSs with $\checkmark$. These systems are almost LTSs with $\checkmark$ (which we described before this remark), except for the additional assumption of finite branching.

We shall not take special care of this condition of finite branching. Technically speaking, the subsequent results work for $T = \mathcal{P}$ but not for $T = \mathcal{P}_{\mathrm{fin}}$; it is because we need a **Cppo**-enriched structure in $\mathcal{K}\ell(T)$. To illustrate the problem, look at the first system in (2.1). Although every branching in the system is finite, its trace map (which we will accommodate in $\mathcal{K}\ell(T)$) is no longer "finitely branching."

[5] Note that the singleton $1 = \{\checkmark\}$ here in $F = 1 + \Sigma \times \_$ has a different interpretation from $1 = \{\perp\}$ in $T = \mathcal{L} = 1 + \_$. The intuition is as follows. On the one hand, when an execution hits successful termination $\checkmark$, it yields its history of observations as its trace. On the other hand, when an execution hits deadlock $\perp$ then it yields no trace no matter what is the history before hitting $\perp$. This distinction will be made formal in Example 2.4.3.

All the systems listed above are modeled by a function $X \xrightarrow{c} TFX$, hence an arrow $X \xrightarrow{c} FX$ in $\mathcal{K}\ell(T)$. Our question here is: is c a coalgebra in $\mathcal{K}\ell(T)$? In other words: is the functor F on **Sets** also a functor on $\mathcal{K}\ell(T)$?

Hence, to develop a generic theory of traces in $\mathcal{K}\ell(T)$, we need to lift F to a functor $\overline{F}$ on $\mathcal{K}\ell(T)$. A functor $\overline{F}$ is said to be a *lifting* of F if the following diagram commutes. Here J is the left adjoint in (2.9).

$$\begin{tikzcd}
\mathcal{K}\ell(T) \arrow[r, "\overline{F}"] & \mathcal{K}\ell(T) \\
\mathbf{Sets} \arrow[u, "J"] \arrow[r, "F"'] & \mathbf{Sets} \arrow[u, "J"']
\end{tikzcd} \qquad (2.11)$$

The following fact is presented in [103]; see also [92, 93]. Its proof is straightforward.

2.2.3 Lemma. *A lifting $\overline{F}$ of F is in bijective correspondence with a* ***distributive law*** *$\lambda : FT \Rightarrow TF$. A distributive law λ is a natural transformation which is compatible with T's monad structure, in the following way.*

$$\begin{tikzcd}
FX \arrow[r, "F\eta_X"] \arrow[dr, "\eta_{FX}"'] & FTX \arrow[d, "\lambda_X"] \\
& TFX
\end{tikzcd} \qquad
\begin{tikzcd}
FT^2X \arrow[r, "\lambda_{TX}"] \arrow[d, "F\mu_X"'] & TFTX \arrow[r, "T\lambda_X"] & T^2FX \arrow[d, "\mu_{FX}"] \\
FTX \arrow[rr, "\lambda_X"'] & & TFX
\end{tikzcd}$$

□

A distributive law λ induces a lifting $\overline{F}$ as follows. On objects: $\overline{F}X = FX$. Given $f : X \to Y$ in $\mathcal{K}\ell(T)$, we need an arrow $\overline{F}f : FX \to FY$ in $\mathcal{K}\ell(T)$. Recall that f is a function $X \to TY$ in **Sets**; one defines $\overline{F}f$ to be the arrow in $\mathcal{K}\ell(T)$ which corresponds to the function

$$FX \xrightarrow{Ff} FTY \xrightarrow{\lambda_Y} TFY \quad \text{in } \mathbf{Sets} \ .$$

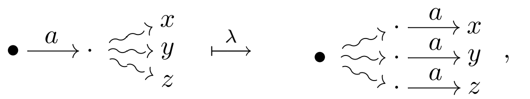
Conversely, given a lifting $\overline{F}$ of F, one obtains a distributive law in the following way.

$$\begin{array}{ll}
TX \xrightarrow{\text{id}} TX \quad \text{in } \mathbf{Sets} & \\
\hline\hline
TX \longrightarrow X \quad \text{in } \mathcal{K}\ell(T) & \\
\hline
\overline{F}TX \longrightarrow \overline{F}X \quad \text{in } \mathcal{K}\ell(T) & \text{applying } \overline{F} \text{ to the arrow} \\
\hline\hline
FTX \longrightarrow TFX \quad \text{in } \mathbf{Sets} & \overline{F}X = FX \text{ on objects, see (2.11)}
\end{array}$$

A distributive law specifies how a transition (of type F) "distributes" over a branching (of type T). Let us look at an example. For $T = \mathcal{P}$ and $F = 1 + \Sigma \times \_$ (the combination for LTSs with explicit termination), we have the following distributive law.

$$\begin{array}{rcl}
1 + \Sigma \times (\mathcal{P}X) & \xrightarrow{\quad \lambda_X \quad} & \mathcal{P}(1 + \Sigma \times X) \\
\checkmark & \longmapsto & \{\checkmark\} \\
(a, S) & \longmapsto & \big\{(a, x) \mid x \in S\big\}
\end{array}$$

For example,



$$\text{that is} \qquad \big(a, \{x, y, z\}\big) \quad \overset{\lambda}{\longmapsto} \quad \big\{(a, x), (a, y), (a, z)\big\} \ ,$$

where waving arrows $\rightsquigarrow$ denote branchings.

Throughout this chapter we need the global assumption that a functor F has a lifting $\overline{F}$ on $\mathcal{K}\ell(T)$, or equivalently, that there is a distributive law $\lambda : FT \Rightarrow TF$. Now we present some sufficient conditions for existence of λ. In most examples one of these conditions holds.

First, take $T = \mathcal{P}$, in which case we have $\mathcal{K}\ell(\mathcal{P}) \cong \mathbf{Rel}$, the category of sets and binary relations. We can provide the following condition that uses relation liftings, whose definition is found e.g. in [62].

2.2.4 Lemma ([133]). *Let F:* **Sets** $\to$ **Sets** *be a functor that preserves weak pullbacks. Then there exists a distributive law λ: $F\mathcal{P} \Rightarrow \mathcal{P}F$ given by*

$$\lambda_X(u) = \big\{\, v \in FX \mid (v, u) \in \mathrm{Rel}_F(\in_X) \,\big\} \ ,$$

where $u \in F\mathcal{P}X$ and $\mathrm{Rel}_F(\in_X) \subseteq FX \times F\mathcal{P}X$ is the F-relation lifting of the membership relation $\in_X$. □

In fact, the functor $\overline{F} : \mathbf{Rel} \to \mathbf{Rel}$ induced by this distributive law carries an arrow $R : X \to Y$ in $\mathcal{K}\ell(\mathcal{P})$—which we can identify with a binary relation between X and Y—to its F-relation lifting $\mathrm{Rel}_F(R)$. That is,

$$\overline{F}R = \mathrm{Rel}_F(R) \quad : \ FX \longrightarrow FY \tag{2.12}$$

in $\mathcal{K}\ell(\mathcal{P}) \cong \mathbf{Rel}$.

Now let us consider a monad T which is other than $\mathcal{P}$. When a monad T is *commutative* and a functor F is *shapely*, we can provide a canonical distributive law. The class of such monads and functors is wide and all the examples in this chapter are contained.

- A *commutative* monad [81] is intuitively a monad whose corresponding algebraic theory has only commutative operators. We exploit the fact that a commutative monad is equipped with an arrow called *double strength*

  $$\mathsf{dst}_{X,Y} : TX \times TY \longrightarrow T(X \times Y) \tag{2.13}$$

  for any sets X and Y; the double strength must be compatible with the monad structure of T in an obvious way.

Our three examples of monads are all commutative, with the following double strengths.

$$\begin{array}{rcl}\mathsf{dst}^{\mathcal{L}}_{X,Y}(u,v) & = & \begin{cases}(u,v) & \text{if } u\in X \text{ and } v\in Y,\\ \perp & \text{if } u=\perp \text{ or } v=\perp,\end{cases}\\ \mathsf{dst}^{\mathcal{P}}_{X,Y}(u,v) & = & u\times v\ ,\\ \mathsf{dst}^{\mathcal{D}}_{X,Y}(u,v) & = & \lambda(x,y).\ u(x)\cdot v(y)\ .\end{array} \tag{2.14}$$

- The family of *shapely functors*[6] [69] on **Sets** is defined inductively by the following BNF notation:

  $$F ::= \mathrm{id} \mid \Sigma \mid F_1\times F_2 \mid \textstyle\coprod_{i\in I} F_i\ ,$$

  where Σ denotes the constant functor into an (arbitrary) set Σ. Notice that taking an infinite product is not allowed, nor an exponentiation to the power of an infinite set. This is in order to ensure that we find an initial F-algebra as a suitable ω-colimit—see Proposition A.1.1.

2.2.5 Lemma. *Let $T\colon$ **Sets** $\to$ **Sets** be a commutative monad, and $F\colon$ **Sets** $\to$ **Sets** a shapely functor. Then there is a distributive law $\lambda\colon FT \Rightarrow TF$.*

Proof. The construction of a distributive law is done inductively on the construction of shapely F.

- If F is the identity functor, then the λ is the identity natural transformation $T\Rightarrow T$.
- If F is a constant functor, say $X\mapsto\Sigma$, then λ is the unit $\eta_\Sigma\colon \Sigma\to T\Sigma$ at $\Sigma\in$ **Sets**.
- If $F=F_1\times F_2$ we use induction in the form of distributive laws $\lambda^{F_i}\colon F_iT\Rightarrow TF_i$ for $i\in\{1,2\}$ to form the composite:

  $$F_1TX\times F_2TX \xrightarrow{\lambda^{F_1}\times\lambda^{F_2}} TF_1X\times TF_2X \xrightarrow{\mathsf{dst}} T(F_1X\times F_2X)\ .$$

- If F is a coproduct $\coprod_{i\in I} F_i$ then we use laws $\lambda^{F_i}\colon F_iT\Rightarrow TF_i$ for $i\in I$ in:

  $$\textstyle\coprod_{i\in I} F_i(TX) \xrightarrow{[T(\kappa_i)\circ\lambda^{F_i}]_{i\in I}} T(\coprod_{i\in I} F_iX)\ .$$

It is straightforward to check that such λ is natural and compatible with the monad structure. □

Note that the results in the sequel rely on existence of a distributive law, but not on e.g. commutative T and shapely F. The latter is a sufficient condition for the former but not a necessary one, although this marginal generality is yet to be exploited in our examples.

[6] Shapely functors as they are called here are referred to as *polynomial* functors by some authors, although other authors allow infinite powers or the powerset construction in their "polynomial" functors.

2.2.3 Order-enriched structures of Kleisli categories

The notion of branching naturally involves a partial order: one branching is bigger than another if the former offers "more possibilities" than the latter. Formally, this order appears as the **Cppo**-*enriched structure* of a Kleisli category. It plays an important role in the initial algebra-final coalgebra coincidence in Section 2.3.1.

A **Cppo**-*enriched category* $\mathbb{C}$ is a category where:

- Each homset $\mathbb{C}(X,Y)$ carries a partial order $\sqsubseteq$ as in

$$\begin{tikzcd} X \arrow[r, bend left, "g"] \arrow[r, bend right, "f"'] \arrow[r, phantom, "\rotatebox{90}{$\sqsubseteq$}"] & Y \end{tikzcd}$$

  which makes $\mathbb{C}(X,Y)$ an ω-cpo with a bottom. This means:

  - for an increasing ω-chain of arrows from X to Y,

$$f_0 \sqsubseteq f_1 \sqsubseteq \dots \quad : \quad X \longrightarrow Y \enspace ,$$

    there exists its join $\bigsqcup_{n<\omega} f_n : X \to Y$;
  - for any X and Y there exists a bottom arrow $\bot_{X,Y} : X \to Y$ which is the minimum in $\mathbb{C}(X,Y)$.

- Moreover, composition of arrows is continuous as a function $\mathbb{C}(X,Y)\times\mathbb{C}(Y,Z) \to \mathbb{C}(X,Z)$. This means preservation of the following joins:[7]

$$g \circ \bigl(\bigsqcup_{n<\omega} f_n\bigr) \;=\; \bigsqcup_{n<\omega}(g \circ f_n) \quad \text{and} \quad \bigl(\bigsqcup_{n<\omega} f_n\bigr) \circ h \;=\; \bigsqcup_{n<\omega}(f_n \circ h) \enspace .$$

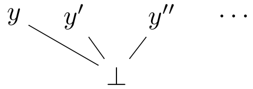
  Note that composition need not preserve bottoms (i.e. it is not necessarily *strict*).

This is in fact an instance of a more general notion of $\mathbb{V}$-*enriched categories* where $\mathbb{V}$ is the category **Cppo** of pointed (i.e. with $\bot$) cpo's and continuous (but not necessarily strict) functions. See [19, 73, 89] for more details on enriched category theory, and [2] on cpo's and domain theory.

2.2.6 Lemma. *For our three examples $\mathcal{L}$, $\mathcal{P}$ and $\mathcal{D}$ of a monad T, the Kleisli category $\mathcal{K}\ell(T)$ is* **Cppo**-*enriched. Moreover, composition of arrows is left-strict: $\bot \circ f = \bot$.*

The left-strictness of composition will be necessary later.

Proof. Notice first that a set TY for $T \in \{\mathcal{L}, \mathcal{P}, \mathcal{D}\}$ carries a cpo structure with $\bot$. The set $\mathcal{L}Y = \{\bot\} + Y$ carries the flat order with a bottom:



[7] This component-wise preservation of joins is equivalent to the continuity of the composition function. See [2, Lemma 3.2.6].

embodying the idea that $\perp$ denotes non-termination or *deadlock*—in contrast to ✓ for successful termination. The set $\mathcal{P}Y$ carries an inclusion order; in $\mathcal{D}Y$ we have the pointwise order—$d \sqsubseteq e$ if $d(y) \leq e(y)$ for each $y \in Y$. The bottom element in $\mathcal{D}Y$ is the zero distribution $[y \mapsto 0]$: this belongs to the set $\mathcal{D}Y$ because $\mathcal{D}$ is the *sub*-distribution monad.

The cpo structure of a homset $\mathcal{K}\ell(T)(X, Y)$ comes from that of TY in a pointwise manner:

$$\begin{tikzcd} X \arrow[r, bend left, "g"] \arrow[r, bend right, "f"'] \arrow[r, phantom, "\sqsupseteq"] & Y \end{tikzcd} \qquad \text{if and only if} \qquad \forall x \in X.\ f(x) \sqsubseteq_{TY} g(x) \ .$$

It is laborious but straightforward to show that composition in $\mathcal{K}\ell(T)$ is continuous and left-strict. □

We are concerned with coalgebras $X \to \overline{F}X$ in the category $\mathcal{K}\ell(T)$, which we assume is **Cppo**-enriched. Hence it comes natural to require that the functor $\overline{F}$ should be somehow compatible with the **Cppo**-enriched structure of $\mathcal{K}\ell(T)$. The obvious choice is to require that $\overline{F}$ is a **Cppo**-*enriched functor* (see e.g. [19]), i.e. $\overline{F}$ is *locally continuous* in this setting. It means that for an increasing ω-chain $f_n : X \to Y$, we have

$$\overline{F}(\bigsqcup_{n<\omega} f_n) = \bigsqcup_{n<\omega} (\overline{F} f_n) \ .$$

This is indeed the assumption universally chosen in axiomatic domain theory; we will come back to this point later in Section 2.3.3. However, for our later purpose, only *local monotonicity* is sufficient: $f \sqsubseteq g$ implies $\overline{F}f \sqsubseteq \overline{F}g$.

For a monad $T \in \{\mathcal{L}, \mathcal{P}, \mathcal{D}\}$ and a shapely functor F (recall Lemma 2.2.5), the lifted $\overline{F}$ is indeed locally continuous. We emphasize again that this does not imply our results in Section 2.3 hold exclusively for shapely functors: being shapely is a *sufficient* condition for the assumption of those later results.

2.2.7 Lemma. *Let F be a shapely functor and $T \in \{\mathcal{L}, \mathcal{P}, \mathcal{D}\}$. The lifting $\overline{F}$: $\mathcal{K}\ell(T) \to \mathcal{K}\ell(T)$ induced by Lemma 2.2.5 is locally continuous.*

Proof. By induction on the construction of shapely functors.

– $F = \mathrm{id}$, the identity functor. Then $\overline{F} = \mathrm{id}$ which satisfies the condition.

– $F = \Sigma$, a constant functor. Then $\overline{F}$ maps every arrow to the identity map on Σ in $\mathcal{K}\ell(T)$. This is obviously locally continuous.

– $F = F_1 \times F_2$. First notice that, for $f : X \to Y$ in $\mathcal{K}\ell(T)$, we obtain $\overline{F}f$ as the following composite in **Sets**.

$$\begin{tikzcd} F_1X \times F_2X \arrow[r, "\overline{F_1}f \times \overline{F_2}f"] \arrow[dr, "\overline{F}f"'] & TF_1Y \times TF_2Y \arrow[d, "\mathsf{dst}_{F_1Y, F_2Y}"] \\ & T(F_1Y \times F_2Y) \end{tikzcd}$$

Because the order in $\mathcal{K}\ell(T)(FX, FY)$ is pointwise, it suffices to show the following: $\mathsf{dst} : TX \times TY \to T(X \times Y)$ is a continuous map between cpo's. It is easy to check that this is indeed the case. See (2.14).

- $F = \coprod_{j \in J} F_j$. For $f : X \to Y$ in $\mathcal{K}\ell(T)$, we obtain the map $\overline{F}f$ as the composite $[T\kappa_j]_{j \in J} \circ \coprod_{j \in J} \mathcal{K}\ell(F_j)(f)$ in **Sets**. Since the order on the homset is pointwise, it suffices to show that each $T\kappa_j : TF_jY \to T(\coprod_{j \in J} F_jY)$ is continuous. This is easy. □

2.3 Final coalgebra in a Kleisli category

In this section we shall prove our main technical result of this chapter: an initial F-algebra in **Sets** yields a final $\overline{F}$-coalgebra in $\mathcal{K}\ell(T)$. It happens in the following two steps: first, an initial algebra in **Sets** is lifted to an initial algebra in $\mathcal{K}\ell(T)$; second we have the initial algebra-final coalgebra coincidence in $\mathcal{K}\ell(T)$. For the latter we use the classical result [127] of limit-colimit coincidence. This is where the **Cppo**-enriched structure of $\mathcal{K}\ell(T)$ plays a role.

In the proof we use two standard constructions: initial/final sequences [5] and limit-colimit coincidence [127]. The reader who is not familiar with these constructions is invited to look at Appendices A.1 and A.2 where we briefly recall them.

2.3.1 Remark. The proof of our main theorem (Theorem 2.3.3) can be simplified if we suitably strengthen the assumptions. First, if we assume local *continuity* of the lifted functor $\overline{F}$ (instead of local *monotonicity* that is assumed in our main theorem), then the initial algebra-final coalgebra coincidence follows from a standard result in axiomatic domain theory; see Section 2.3.3. Furthermore, for the special case $T = \mathcal{P}$ in which case $\mathcal{K}\ell(\mathcal{P}) \cong$ **Rel**, the initial algebra-final coalgebra coincidence is almost obvious due to the duality **Rel** $\cong$ **Rel**$^{\mathrm{op}}$; see Section 2.3.2.

2.3.1 An initial algebra in Sets yields a final coalgebra in $\mathcal{K}\ell(T)$

First, it is standard that an initial algebra in **Sets** is lifted to an initial algebra in $\mathcal{K}\ell(T)$. Such a phenomenon is studied for instance in [33, 106] in the context of combining datatypes (modeled by an initial algebra) and effectful computations (modeled by a Kleisli category). For this result we do not need an order structure.

2.3.2 Proposition. *Let T be a monad and F be a endofunctor, both on a category $\mathbb{C}$. Assume that we have a distributive law $FT \Rightarrow TF$—or equivalently, we have a lifting $\overline{F}$ on $\mathcal{K}\ell(T)$. If F has an initial algebra $\alpha : FA \xrightarrow{\cong} A$ in $\mathbb{C}$, then*

$$J\alpha = \eta_A \circ \alpha \ : \quad \overline{F}A \longrightarrow A \qquad \text{in } \mathcal{K}\ell(T)$$

is an initial $\overline{F}$-algebra. Here J is the canonical Kleisli left adjoint as in (2.9).

We will use an instance of this result for $\mathbb{C} =$ **Sets**.

Proof. It follows from [54, Theorem 2.14] that a distributive law lifts the canonical Kleisli adjunction to an adjunction between the categories $\mathbf{Alg}_F$ and $\mathbf{Alg}_{\overline{F}}$ of algebras.

$$\begin{tikzcd}
& \mathbf{Alg}_F \arrow[r, bend left, "J'"] \arrow[r, phantom, "\perp"] \arrow[d] & \mathbf{Alg}_{\overline{F}} \arrow[l, bend left] \arrow[d] & \\
F \arrow[r, phantom] & \mathbb{C} \arrow[loop left, "F"] \arrow[r, bend left, "J"] \arrow[r, phantom, "\perp"] & \mathcal{K}\ell(T) \arrow[l, bend left, "K"] \arrow[loop right, "\overline{F}"] &
\end{tikzcd}$$

The left adjoint J' preserves an initial object (see e.g. [96]). □

Second, we use the initial algebra-final coalgebra coincidence in $\mathcal{K}\ell(T)$—which holds in a suitable order-enriched setting—to identify a final coalgebra in $\mathcal{K}\ell(T)$. This is our main results of the chapter.

2.3.3 Theorem (Main theorem of Chapter 2)**.** *Assume the following conditions.*

1. *A monad T on* **Sets** *is such that its Kleisli category $\mathcal{K}\ell(T)$ is* **Cppo**-*enriched and composition in $\mathcal{K}\ell(T)$ is left-strict.*
2. *For an endofunctor F on* **Sets**, *we have a distributive law $\lambda : FT \Rightarrow TF$. Equivalently, F has a lifting $\overline{F}$ on $\mathcal{K}\ell(T)$. Moreover, the lifting $\overline{F}$ is locally monotone.*
3. *The functor F preserves ω-colimits in* **Sets**, *hence has an initial algebra via the initial sequence (see Proposition A.1.1).*

Such a triple (F, T, λ) shall be called a ***trace situation.***

Then the initial F-algebra $\alpha : FA \xrightarrow{\cong} A$ yields a final $\overline{F}$-coalgebra in $\mathcal{K}\ell(T)$ by

$$(J\alpha)^{-1} = J(\alpha^{-1}) = \eta_{FA} \circ \alpha^{-1} \ : \quad A \longrightarrow \overline{F}A \qquad \text{in } \mathcal{K}\ell(T) \ .$$

We first present the main line of the proof. Some details are provided in the form of subsequent lemmas. Note that the assumptions are satisfied by $T \in \{\mathcal{L}, \mathcal{P}, \mathcal{D}\}$ and shapely F; see Lemmas 2.2.6 and 2.2.5.

Proof. By the assumption (3) we obtain an initial algebra via the initial sequence in **Sets**.

<u>In **Sets**</u>

$$\begin{tikzcd}
& & & & A \arrow[dd, bend left, "\alpha^{-1}"] & \text{(colimit)} \\
\cdots \arrow[r, "F^{n-1}¡"] & F^n 0 \arrow[r] \arrow[urrr, bend left, "\alpha_n"] \arrow[drrr, bend right, "F\alpha_{n-1}"'] & F^{n+1}0 \arrow[r] \arrow[urr, "\alpha_{n+1}"] \arrow[drr, "F\alpha_n"'] & \cdots & & \\
& & & & FA \arrow[uu, bend left, "\alpha", "\cong"'] & \text{(colimit)}
\end{tikzcd} \tag{2.15}$$

Here $0 = \emptyset \in$ **Sets** is initial and $¡ : 0 \to X$ is the unique arrow from 0 to an arbitrary X. We apply the functor $J :$ **Sets** $\to \mathcal{K}\ell(T)$ to the whole diagram. Since a left adjoint

J preserves colimits, the two cocones in the following diagram are both colimits again.

$$\begin{tikzcd}
\underline{\text{In } \mathcal{K}\ell(T)} & & & & A \arrow[dd, bend left, "J\alpha^{-1}"] & \text{(colimit)} \\
\cdots \arrow[r, "JF^{n-1}\,\text{¡}"] & F^n 0 \arrow[r] \arrow[urrr, bend left, "J\alpha_n"] \arrow[drrr, bend right, "JF\alpha_{n-1}"'] & F^{n+1}0 \arrow[r] \arrow[urr, "J\alpha_{n+1}"] \arrow[drr, "JF\alpha_n"'] & \cdots & \cong & \\
 & & & & FA \arrow[uu, bend left, "J\alpha"] & \text{(colimit)}
\end{tikzcd}\tag{2.16}$$

The ω-chain in this diagram is in fact the initial sequence for the functor $\overline{F}$ (Lemma 2.3.4) because, for example, a left adjoint J preserves initial objects. Moreover the lower cone is the image of the upper cone under $\overline{F}$ because $JF = \overline{F}J$; see the diagram (2.11). Hence the diagram (2.16) is equal to the following one. Recall that $\overline{F}X = FX$ on objects.

$$\begin{tikzcd}
\underline{\text{In } \mathcal{K}\ell(T)} & & & & A \arrow[dd, bend left, "J\alpha^{-1}"] & \text{(colimit)} \\
\cdots \arrow[r, "\overline{F}^{n-1}\,\text{¡}"] & \overline{F}^n 0 \arrow[r] \arrow[urrr, bend left, "J\alpha_n"] \arrow[drrr, bend right, "\overline{F}J\alpha_{n-1}"'] & \overline{F}^{n+1}0 \arrow[r] \arrow[urr, "J\alpha_{n+1}"] \arrow[drr, "\overline{F}J\alpha_n"'] & \cdots & \cong & \\
 & & & & \overline{F}A \arrow[uu, bend left, "J\alpha"] & \text{(colimit)}
\end{tikzcd}\tag{2.17}$$

Thus Proposition A.1.1 yields that $J\alpha : \overline{F}A \xrightarrow{\cong} A$ is an initial $\overline{F}$-algebra. This can be seen as a more concrete proof of Proposition 2.3.2.

Now we show the initial algebra-final coalgebra coincidence in $\mathcal{K}\ell(T)$. This is done by reversing all the arrows in (2.17) and transforming the diagram into the one of the final sequence and its limits.

We notice (Lemma 2.3.6) that each arrow $\overline{F}^n\,\text{¡}$ in the initial sequence is an embedding (in the sense of *embedding-projection pairs*; see Definition A.2.1). Hence the limit-colimit coincidence Theorem A.2.5 says that every arrow in the diagram is an embedding. Note that $J\alpha$ and $J\alpha^{-1}$, inverse to each other, form an embedding-projection pair.

By taking the corresponding projections—they are uniquely determined (Lemma A.2.2) and are denoted by $(\_)^P$—we obtain the next diagram. The limit-colimit coincidence Theorem A.2.5 says that the two resulting cones are both limits. It is also obvious that the whole diagram commutes.

$$\begin{tikzcd}
\underline{\text{In } \mathcal{K}\ell(T)} & & & & A \arrow[dllll, bend right, "(J\alpha_n)^P"'] \arrow[dll, "(J\alpha_{n+1})^P"'] \arrow[dd, bend left, "(J\alpha)^P"] & \text{(limit)} \\
\cdots & \overline{F}^n 0 \arrow[l, "(\overline{F}^{n-1}\,\text{¡})^P"'] & \overline{F}^{n+1}0 \arrow[l] & \cdots \arrow[l] & \cong & \\
 & & & & \overline{F}A \arrow[ulll, bend left, "(\overline{F}J\alpha_{n-1})^P"] \arrow[ull, "(\overline{F}J\alpha_n)^P"] \arrow[uu, bend right, "(J\alpha^{-1})^P"'] & \text{(limit)}
\end{tikzcd}\tag{2.18}$$

The ω^{op}-chain here is indeed a final sequence: Lemma 2.3.5 shows—using the assumption (1) on left-strictness—that 0 is also final in $\mathcal{K}\ell(T)$, and according to Lemma 2.3.6 we have $(\overline{F}^n \text{¡})^P = \overline{F}^n\, !$ where $! : X \to 0$ is the unique arrow to the final object 0 in $\mathcal{K}\ell(T)$. As to the lower cone we have $\big(\overline{F} J\alpha_n\big)^P = \overline{F}\big((J\alpha_n)^P\big)$ by Lemma 2.3.7.

Hence the diagram (2.18) is equal to the following one, showing the final sequence for $\overline{F}$, its limit (the upper one) and that limit mapped by $\overline{F}$ (the lower one) which is again a limit.

$$\begin{tikzcd}
\underline{\text{In } \mathcal{K}\ell(T)} & & & & A \arrow[llld, bend right, "(J\alpha_n)^P"'] \arrow[lld, "(J\alpha_{n+1})^P"] \arrow[dd, bend right, "J\alpha^{-1}"'] \arrow[dd, phantom, "\cong"] & \text{(limit)} \\
\cdots & \overline{F}^n 0 \arrow[l, "\overline{F}^{n-1}\, !"'] & \overline{F}^{n+1} 0 \arrow[l] & \cdots \arrow[l] & & \\
& & & & \overline{F}A \arrow[lllu, bend left, "\overline{F}(J\alpha_{n-1})^P"] \arrow[llu, "\overline{F}(J\alpha_n)^P"'] \arrow[uu, bend right, "J\alpha"'] & \text{(limit)}
\end{tikzcd} \tag{2.19}$$

By Proposition A.1.2 we conclude that $J\alpha^{-1}$ is a final $\overline{F}$-coalgebra. □

In the remainder of this section the lemmas used in the above proof are presented. We work with a trace situation (F, T, λ), i.e. those which satisfy the assumptions as in Theorem 2.3.3.

2.3.4 Lemma. *The ω-chain in the diagram (2.16) is indeed the initial sequence for $\overline{F}$. That is, we have for each $n < \omega$,*

$$JF^n\big(\,\text{¡}^{\mathbf{Sets}}\,\big) = \overline{F}^n\big(\,\text{¡}^{\mathcal{K}\ell(T)}\,\big) \;:\; JF^n 0 \longrightarrow JF^{n+1} 0 \quad \textit{in } \mathcal{K}\ell(T),$$

where $\text{¡}^{\mathbf{Sets}} : 0 \to F0$ *in* **Sets** *and* $\text{¡}^{\mathcal{K}\ell(T)} : 0 \to F0$ *in* $\mathcal{K}\ell(T)$ *denote the unique maps.*

Proof. By induction on n. For $n = 0$ the two maps are equal due to initiality of $J0 = 0$ in $\mathcal{K}\ell(T)$. For the step case we use the commutativity $JF = \overline{F}J$ of (2.11). □

2.3.5 Lemma. *The empty set 0 is both an initial and a final object in $\mathcal{K}\ell(T)$.*

In particular, this implies that the object $T0$ is final in **Sets**, hence $T0 \cong 1$.

Proof. The functor $J : \mathbf{Sets} \to \mathcal{K}\ell(T)$ preserves initial objects since it is a left adjoint. Therefore $0 = J0$ is initial in $\mathcal{K}\ell(T)$. Finality follows essentially from the left-strictness assumption: for each set X there exists at least one arrow $X \to 0$ in $\mathcal{K}\ell(T)$, for example $\perp_{X,0}$. To show the uniqueness of such an arrow, take an arbitrary arrow $f : X \to 0$ in $\mathcal{K}\ell(T)$. Recalling that the bottom map $\perp_{0,0} : 0 \to 0$ is also the identity arrow in $\mathcal{K}\ell(T)$ because of initiality, we obtain

$$f \;=\; \mathrm{id} \circ f \;=\; \perp_{0,0} \circ f \;\overset{(*)}{=}\; \perp_{X,0} \;,$$

where the compositions are taken in $\mathcal{K}\ell(T)$ and the equality marked by $(*)$ holds by left-strictness of composition. □

2.3.6 Lemma. *Each arrow $\overline{F}^n\,\text{¡}$ in the initial sequence for $\overline{F}$, as in the diagram (2.17), is an embedding. Its corresponding projection is given by*

$$\left(\overline{F}^n\,\text{¡}\right)^P = \overline{F}^n\,! \qquad \textit{in} \quad \begin{tikzcd} F^n0 \arrow[r, tail, bend right, "\overline{F}^n\,\text{¡}"'] & F^{n+1}0 \arrow[l, two heads, bend right, "\overline{F}^n\,!"'] \end{tikzcd} \; .$$

Proof. We show that $(\overline{F}^n\,\text{¡}\,,\,\overline{F}^n\,!\,)$ is an embedding-projection pair for all $n < \omega$. We have $\overline{F}^n\,! \circ \overline{F}^n\,\text{¡} = \mathrm{id}$ because $! \circ \text{¡} = \mathrm{id}$. For the other half we have

$$\begin{aligned} \overline{F}^n\,\text{¡} \circ \overline{F}^n\,! &= \overline{F}^n(\,\text{¡} \circ !\,) \\ &= \overline{F}^n(\bot_{0,F0} \circ !\,) && \text{initiality of 0 in } \mathcal{K}\ell(T) \\ &= \overline{F}^n(\bot_{F0,F0}) && \text{composition is left-strict} \\ &\sqsubseteq \overline{F}^n(\mathrm{id}) = \mathrm{id} && \overline{F} \text{ is locally monotone.} \qquad \square \end{aligned}$$

2.3.7 Lemma. *We have $\left(\overline{F}J\alpha_n\right)^P = \overline{F}\big((J\alpha_n)^P\big)$. Hence the lower cone in the diagram (2.18) is the image of the upper cone under $\overline{F}$.*

Proof. It is easy to check that $\left(\overline{F}J\alpha_n,\, \overline{F}\big((J\alpha_n)^P\big)\right)$ indeed form an embedding-projection pair. Therein we use the monotonicity of $\overline{F}$'s action on arrows. □

2.3.2 Simpler proof in $\mathcal{K}\ell(\mathcal{P}) \cong$ Rel

When $T = \mathcal{P}$ we have the self-duality

$$Op \quad : \quad \mathcal{K}\ell(\mathcal{P})^{\mathrm{op}} \xrightarrow{\cong} \mathcal{K}\ell(\mathcal{P}) \; .$$

This is because of the following bijective correspondence between functions

$$\begin{array}{c} X \xrightarrow{\quad f \quad} \mathcal{P}Y \;\; \text{in } \mathbf{Sets} \\ \hline\hline Y \xrightarrow{\quad f^{\vee} \quad} \mathcal{P}X \;\; \text{in } \mathbf{Sets} \end{array}$$

given by $f^{\vee}(y) = \{x \in X \mid y \in f(x)\}$. Recalling $\mathcal{K}\ell(\mathcal{P}) \cong \mathbf{Rel}$, this mapping $f \mapsto f^{\vee}$ carries a relation to its opposite relation.

Due to this "global" duality $\mathcal{K}\ell(\mathcal{P}) \cong \mathcal{K}\ell(\mathcal{P})^{\mathrm{op}}$, the proof of Theorem 2.3.3 is drastically simplified for $T = \mathcal{P}$. It essentially relies on the lifted self duality $\mathbf{Alg}_{\overline{F}} \cong \mathbf{Alg}_{\overline{F}^{\mathrm{op}}}$, where the latter is isomorphic to $(\mathbf{Coalg}_{\overline{F}})^{\mathrm{op}}$. We do not need here an order structure of $\mathcal{K}\ell(\mathcal{P})$ nor local monotonicity of $\overline{F}$.

2.3.8 Theorem. *Let F : **Sets** $\to$ **Sets** be a functor which preserves weak pullbacks, and $\overline{F} : \mathcal{K}\ell(\mathcal{P}) \to \mathcal{K}\ell(\mathcal{P})$ be its lifting induced by relation lifting (Lemma 2.2.4). Then an initial F-algebra in* **Sets** *yields a final $\overline{F}$-coalgebra in $\mathcal{K}\ell(\mathcal{P})$.*

Proof. We have the following situation because of the self-duality of $\mathcal{K}\ell(\mathcal{P})$.

$$\begin{tikzcd}
\mathbf{Sets} \arrow[r, bend left, "J"] \arrow[r, phantom, "\perp"] \arrow["F"', loop below] & \mathcal{K}\ell(\mathcal{P}) \arrow[l, bend left, "K"] \arrow[r, "Op^{\mathrm{op}}", "\cong"'] \arrow["\overline{F}"', loop below] & \mathcal{K}\ell(\mathcal{P})^{\mathrm{op}} \arrow["\overline{F}^{\mathrm{op}}"', loop below]
\end{tikzcd}$$

The adjunction $J \dashv K$ and the isomorphism $Op : \mathcal{K}\ell(\mathcal{P})^{\mathrm{op}} \overset{\cong}{\Rightarrow} \mathcal{K}\ell(\mathcal{P})$ lift to those between the categories of algebras.

$$\begin{tikzcd}
\mathbf{Alg}_F \arrow[r, bend left, "J'"] \arrow[r, phantom, "\perp"] \arrow[d] & \mathbf{Alg}_{\overline{F}} \arrow[l, bend left] \arrow[r, "(Op')^{\mathrm{op}}", "\cong"'] \arrow[d] & \mathbf{Alg}_{\overline{F}^{\mathrm{op}}} \arrow[r, "\cong"] \arrow[d] & (\mathbf{Coalg}_{\overline{F}})^{\mathrm{op}} \\
\mathbf{Sets} \arrow[r, bend left, "J"] \arrow[r, phantom, "\perp"] \arrow["F"', loop below] & \mathcal{K}\ell(\mathcal{P}) \arrow[l, bend left, "K"] \arrow[r, "Op^{\mathrm{op}}", "\cong"'] \arrow["\overline{F}"', loop below] & \mathcal{K}\ell(\mathcal{P})^{\mathrm{op}} \arrow["\overline{F}^{\mathrm{op}}"', loop below] &
\end{tikzcd}$$

Indeed, $J \dashv K$ lifts due to Proposition 2.3.2; the lifted isomorphism $Op' : \mathbf{Alg}_{\overline{F}} \overset{\cong}{\Rightarrow} \mathbf{Alg}_{\overline{F}^{\mathrm{op}}}$ is because of the following commutativity:

$$\begin{tikzcd}
\mathcal{K}\ell(\mathcal{P})^{\mathrm{op}} \arrow[r, "Op"] \arrow[d, "\overline{F}^{\mathrm{op}}"'] & \mathcal{K}\ell(\mathcal{P}) \arrow[d, "\overline{F}"] \\
\mathcal{K}\ell(\mathcal{P})^{\mathrm{op}} \arrow[r, "Op"'] & \mathcal{K}\ell(\mathcal{P})
\end{tikzcd} \tag{2.20}$$

which is because: $\overline{F}R = \mathrm{Rel}_F(R)$ (see (2.12)); and taking relation liftings is compatible with opposite relations (i.e. $\mathrm{Rel}_F(R^{\mathrm{op}}) = (\mathrm{Rel}_F R)^{\mathrm{op}}$, see [58]). Moreover the category $\mathbf{Alg}_{\overline{F}^{\mathrm{op}}}$ is obviously isomorphic to $(\mathbf{Coalg}_{\overline{F}})^{\mathrm{op}}$.

Therefore an initial object in $\mathbf{Alg}_F$ is carried to an initial object in $(\mathbf{Coalg}_{\overline{F}})^{\mathrm{op}}$, hence to a final object in $\mathbf{Coalg}_{\overline{F}}$. □

For monads such as $T = \mathcal{D}$ a "global" self-duality $\mathcal{K}\ell(T) \cong \mathcal{K}\ell(T)^{\mathrm{op}}$ is not available. Instead, in the proof of Theorem 2.3.3, we exploit the "partial" duality which holds between the colimit/limit of the initial/final sequence.

2.3.3 Related work: axiomatic domain theory

The initial algebra-final coalgebra coincidence is heavily exploited in the field of *axiomatic domain theory*, e.g. in [31, 34, 35, 125]. There, categories which have an initial algebra and a final coalgebra coinciding with each other, for each endofunctor, are called *algebraically compact categories.* They draw special attention as suitable "categories of domains" for denotational semantics of datatype construction. The relevance comes as follows.

Let $\mathbb{C}$ be a "category of domains." One thinks of an object of the category $\mathbb{C}$ as a type. A "recursive datatype constructor" is presented as a bifunctor $G : \mathbb{C}^{\mathrm{op}} \times \mathbb{C} \to \mathbb{C}$.

Note the presence of both covariance and contravariance. One expects that such a category $\mathbb{C}$ has a canonical fixed point $\mathrm{Fix}\, G$ such that

$$G(\mathrm{Fix}\, G, \mathrm{Fix}\, G) \xrightarrow{\cong} \mathrm{Fix}\, G \enspace ,$$

which represents the recursive type determined by the datatype constructor G. Freyd [34] showed that if $\mathbb{C}$ is algebraically compact, then one can construct such a fixed point as a suitable initial algebra; moreover this fixed point is shown by Fiore [31] to be a canonical one in a suitable sense. The rough idea here is that the covariant part of G is taken care of by an initial algebra; the contravariant part is by a final coalgebra; the initial algebra-final coalgebra coincidence yields a fixed point of overall G.

Typical examples of algebraically compact categories are enriched over **Cppo** or one of its variants. This conforms the traditional use of the word "domain" for certain cpo's (e.g. in [2]).

Although in this chapter we utilize the initial algebra-final coalgebra coincidence result in $\mathcal{K}\ell(T)$, we are not so much interested in algebraic compactness of $\mathcal{K}\ell(T)$. This is because our motivation is different from that of axiomatic domain theory. In studying trace semantics for coalgebras, we need not deal with *every* endofunctor on $\mathcal{K}\ell(T)$, but only such an endofunctor $\overline{F}$ which is a lifting of F : **Sets** → **Sets**.

Local continuity vs. local monotonicity In axiomatic domain theory, **Cppo**-enriched categories are said to be algebraically compact because, "in a 2-category setting" [35], every endofunctor has an initial algebra and a final coalgebra. Concretely this means: "every locally continuous functor."

In this spirit, we could have made a stronger assumption of $\overline{F}$'s local continuity in Theorem 2.3.3 instead of local monotonicity. If we do so, in fact, the proof of Theorem 2.3.3 becomes much simpler: the following proposition (Lemma in [35, p.98]) immediately yields the initial algebra-final coalgebra coincidence for a locally continuous $\overline{F}$.

2.3.9 Proposition ([35]). *Let $\mathbb{D}$ be a* **Cppo***-enriched category whose composition is left-strict, and $G : \mathbb{D} \to \mathbb{D}$ be a locally continuous endofunctor. An initial algebra $\beta : GB \xrightarrow{\cong} B$, if it exists, yields a final coalgebra $\beta^{-1} : B \xrightarrow{\cong} GB$.*

Proof. Given a coalgebra $d : Y \to GY$, the function

$$\Phi \;:\; \mathbb{D}(Y, B) \longrightarrow \mathbb{D}(Y, B) \enspace , \qquad f \longmapsto \beta \circ Gf \circ d$$

is continuous due to the local continuity of G. Hence by the Knaster-Tarski theorem it has the least fixed point $\bigsqcup_{n<\omega} \Phi^n(\bot)$; this proves existence of a morphism from d to β^{-1}.

$$\begin{tikzcd}
GY \arrow[r] & GB \\
Y \arrow[u, "d"] \arrow[r] & B \arrow[u, "\cong"', "\beta^{-1}"]
\end{tikzcd}$$

Now we shall show its uniqueness. Assume that $g : Y \to B$ is a morphism of coalgebras as above, that is, $\Phi(g) = g$. Similarly to Φ, we define a function

$\Psi : \mathbb{D}(B,B) \to \mathbb{D}(B,B)$ as the one which carries $h : B \to B$ to $\beta \circ Gh \circ \beta^{-1}$. We have

$$\begin{aligned}
\textstyle\bigsqcup_n \Phi^n(\bot) &= \textstyle\bigsqcup_n \Phi^n(Y \xrightarrow{g} B \xrightarrow{\bot} B) && \text{composition is left-strict, so } \bot \circ g = \bot \\
&= \textstyle\bigsqcup_n \big(\Psi^n(\bot) \circ \Phi^n(g)\big) && \Phi^n(\bot \circ g) = \Psi^n(\bot) \circ \Phi^n(g)\text{, by induction} \\
&= \big(\textstyle\bigsqcup_n \Psi^n(\bot)\big) \circ \big(\textstyle\bigsqcup_n \Phi^n(g)\big) && \text{composition is continuous} \\
&= \textstyle\bigsqcup_n \Phi^n(g) && \textstyle\bigsqcup_n \Psi^n(\bot) = \mathrm{id},\ (*) \\
&= g && \Phi(g) = g \text{ by assumption.}
\end{aligned}$$

Here $(*)$ holds because $\bigsqcup_n \Psi^n(\bot)$, being a fixed point for Ψ, is the unique morphism of algebras from β to β. This shows that the morphism g must be the least fixed point of Φ, which is unique. □

For our main Theorem 2.3.3 of this chapter we can do with only local monotonicity of the lifted functor $\overline{F}$, by taking a closer look at the initial/final sequences. However at this stage it is not clear how much we gain from this generality: up to now we have not found an example where the functor $\overline{F}$ is only locally monotone (and not locally continuous).

2.3.4 Related work: functional programming

Datatypes and effects In a different context of functional programming, the work [106] also studies initial algebras and final coalgebras in a Kleisli category. The motivation there is to combine *datatypes* and *effects*. More specifically, an initial algebra and a final coalgebra support the *fold* and the *unfold* operators, respectively, used in recursive programs over datatypes. A computational effect is presented as a monad, and its Kleisli category is the category of effectful computations.

The difference between [106] and the current work is as follows. In [106], the original, base category of pure functions is already assumed to be algebraically compact; the paper studies the conditions for the base category's algebraic compactness to be carried over to a Kleisli category. In contrast, in the current work, it is a monad—with a suitable order structure, embodying the essence of "branching"—which yields the initial algebra-final coalgebra coincidence in a Kleisli category; the coincidence is not present in the original category **Sets**.

Relational refinement calculus In a yet another context, initial algebras in the category **Rel** of sets and relations are investigated in [16, 101]. The category **Rel** is isomorphic to the Kleisli category $\mathcal{K}\ell(\mathcal{P})$, an instance of our general Kleisli framework.

The aim in [16, 101] is to develop a refinement calculus which involves algebraic datatypes: starting from a specification of a desired function, such a calculus allows one to derive an implementation of—i.e. an algorithm which realizes—the desired function, by repeatedly applying derivation rules of the calculus (such as "fusion laws"). The desired *function* is an arrow in **Sets**; however one can make intermediate derivations in the extended domain of **Rel** and come back at the very last moment to **Sets**. Working in **Rel** gives one much more freedom. For example one can take the inverse of every arrow in **Rel**.

Since their main concern is in initial algebras in **Rel**, the lifting result (our Proposition 2.3.2) is fundamental in their investigation. Moreover, as we have seen in Section 2.3.2, the initial algebra-final coalgebra coincidence is much easier to see in the special case of **Rel** $\cong \mathcal{K}\ell(\mathcal{P})$. In fact identification of final coalgebras in **Rel** is almost implicit in [16]: one important technical ingredient in Section 2.3.2 is the compatibility diagram (2.20) which is present in [16, pp.112] (in a more general setting of allegories). Nevertheless the crucial initial algebra-final coalgebra coincidence in **Rel** is not explicitly presented in [16].

The main result of [101] (Theorem 5) does explicitly mention final coalgebras, but in a category different from **Rel**. The result is about final coalgebras in the category **Tran** of sets and monotone predicate transformers between them. Up to now we are yet to find a direct connection between our result and the one in [101].

2.4 Finite trace semantics via coinduction

In this section we shall further illustrate the observation that the principle of coinduction, when employed in $\mathcal{K}\ell(T)$, captures trace semantics of state-based systems. As we have shown in the previous section, an initial algebra in **Sets** constitutes the semantic domain, i.e. is a final coalgebra in $\mathcal{K}\ell(T)$. Viewing an initial algebra as the set of well-founded terms (such as finite words or finite-depth parse trees), this fact means that the "trace semantics" induced by coinduction is inevitably *finite*, in the sense that it captures only finite-depth behaviors. Here we will elaborate on this finiteness issue as well.

2.4.1 Trace semantics by coinduction

As we have seen in Section 2.2.2 various types of state-based systems allow their presentation as coalgebras $X \to \overline{F}X$ in a Kleisli category $\mathcal{K}\ell(T)$. For example,

- LTSs with explicit termination, with $T = \mathcal{P}$ and $F = 1 + \Sigma \times \_$;
- probabilistic LTSs (also called *generative probabilistic transition systems* in [40, 128]) with explicit termination, with $T = \mathcal{D}$ and $F = 1 + \Sigma \times \_$;
- context-free grammars with $T = \mathcal{P}$ and $F = (\Sigma + \_)^*$.

The main observation underlying this work is the following. If we instantiate the parameters

$$T \text{ for branching type} \qquad \text{and} \qquad F \text{ for transition type}$$

in the coinduction diagram

$$\begin{tikzcd}
\overline{F}X \arrow[r, dashed, "\overline{F}(\mathsf{tr}(c))"] & \overline{F}A \\
X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{tr}(c)"'] & A \arrow[u, "\cong"', "J\alpha^{-1}"]
\end{tikzcd} \qquad \text{in } \mathcal{K}\ell(T) \tag{2.21}$$

with one of the above choices, then the commutativity of the diagram is equivalent to the corresponding (conventional) definition of trace semantics in Section 2.1.1. Therefore we claim that the diagram (2.21) is the mathematical principle underlying various "trace semantics," no matter if it is "trace set" (non-deterministic) or "trace distribution" (probabilistic).

2.4.1 Definition (Trace semantics for coalgebras)**.** Let (F, T, λ) be a trace situation (Theorem 2.3.3), and $\alpha : FA \xrightarrow{\cong} A$ is an initial F-algebra in **Sets**. Given a coalgebra $c : X \to TFX$ in **Sets**, we can assign a function

$$\mathsf{tr}(c) \ : \ X \longrightarrow TA \qquad \text{in } \mathbf{Sets}$$

which is, as an arrow $X \to A$ in $\mathcal{K}\ell(T)$, the unique one making the diagram (2.21) commute. We shall call this function $\mathsf{tr}(c)$ the *(finite) trace semantics* for the coalgebra c. □

2.4.2 Example. As further illustration we give details for the choice of parameters $T = \mathcal{P}$ and $F = 1 + \Sigma \times \_$. This is the suitable choice of parameters to deal with the first system in (2.1).

Now the coinduction diagram looks as follows. Recall that an initial F-algebra is carried by the set Σ^* of finite words.

$$\begin{tikzcd}
1 + \Sigma \times X \arrow[r, dashed, "1 + \Sigma \times \mathsf{tr}(c)"] & 1 + \Sigma \times \Sigma^* \\
X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{tr}(c)"'] & \Sigma^* \arrow[u, "\cong"', "J([\mathsf{nil}, \mathsf{cons}])^{-1}"]
\end{tikzcd} \qquad \text{in } \mathcal{K}\ell(\mathcal{P}) \tag{2.22}$$

It assigns, to a system c, a function $\mathsf{tr}(c) : X \to \mathcal{P}(\Sigma^*)$ which carries a state $x \in X$ to the set of finite words on Σ which can possibly arise as an "execution trace" of c starting from x. The commutativity states equality of two arrows $X \rightrightarrows 1 + \Sigma \times \Sigma^*$ in $\mathcal{K}\ell(\mathcal{P})$, that is, functions $X \rightrightarrows \mathcal{P}(1 + \Sigma \times \Sigma^*)$. Let us denote these functions by

$$u = (1{+}\Sigma{\times}\mathsf{tr}(c)) \circ c \quad \text{(up, then right)}, \qquad v = J([\mathsf{nil}, \mathsf{cons}])^{-1} \circ \mathsf{tr}(c) \quad \text{(right, then up)}.$$

For each $x \in X$, the following conditions—derived straightforwardly by definition of composition of $\mathcal{K}\ell(\mathcal{P})$, lifting of the functor $1 + \Sigma \times \_$, etc.—specify u and v's value at x, as a subset of $1 + \Sigma \times \Sigma^*$.

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

Commutativity of (2.22) amounts to $u = v$; this leads to the condition (2.3).

From a different point of view we can also say as follows: finality of the coalgebra $\Sigma^* \xrightarrow{\cong} 1 + \Sigma \times \Sigma^*$ in (2.22) ensures that the conventional recursive definition (2.3) uniquely determines a function $\mathsf{tr}(c) : X \to \mathcal{P}(\Sigma^*)$. That is, $\mathsf{tr}(c)$ is *well-defined.*

An easy consequence of the recursive definition (2.3) is

$$a_1 \dots a_n \in \mathsf{tr}(c)(x) \quad \Longleftrightarrow \quad \exists x_1, \dots, x_n \in X. \quad x \overset{a_1}{\to} \cdots \overset{a_n}{\to} x_n \to \checkmark \ .$$

Therefore every trace $a_1 \dots a_n \in \mathsf{tr}(c)(x)$ has termination $\checkmark$ implicit at its tail. In particular, the set $\mathsf{tr}(c)(x)$ is not necessarily prefix-closed: $a_1 \dots a_n a_{n+1} \cdots a_{n+m} \in \mathsf{tr}(c)(x)$ does not imply $a_1 \dots a_n \in \mathsf{tr}(c)(x)$.

2.4.3 Example. Let us take $T = \mathcal{L}$ (the lift monad) and $F = 1 + \Sigma \times \_$. In this case a coalgebra $X \overset{c}{\to} \mathcal{L}(1 + \Sigma \times X)$ in **Sets** is a system which can

- get into a deadlock ($c(x) = \bot$ with $\mathcal{L} = \{\bot\} + \_$),
- successfully terminate ($c(x) = \checkmark$ with $F = \{\checkmark\} + \Sigma \times \_$), or
- output a letter from Σ and move to the next state ($c(x) = (a, x')$).

By examining trace semantics for such systems, we shall formally put the difference between the computational meanings of the two elements, $\bot$ and $\checkmark$.

The coinduction diagram (2.21) instantiates to the same diagram as (2.22), but now in the category $\mathcal{K}\ell(\mathcal{L})$. Easy calculation shows that its commutativity amounts to the following condition. The function

$$X \xrightarrow{\mathsf{tr}(c)} \mathcal{L}(\Sigma^*) = \{\bot\} + \Sigma^* \qquad \text{in } \mathbf{Sets}$$

satisfies, for each $x \in X$,

$$\begin{array}{lll} \mathsf{tr}(c)(x) = \langle\rangle & \Longleftrightarrow & c(x) = \checkmark \ , \\ \mathsf{tr}(c)(x) = a \cdot \sigma & \Longleftrightarrow & \exists x' \in X. \ \big(c(x) = (a, x') \ \wedge \ \mathsf{tr}(c)(x') = \sigma\big) \ , \\ \mathsf{tr}(c)(x) = \bot & \Longleftrightarrow & c(x) = \bot \quad \text{or} \quad \exists x' \in X. \ \big(c(x) = (a, x') \ \wedge \ \mathsf{tr}(c)(x') = \bot\big) \ . \end{array} \tag{2.23}$$

Here $\sigma \in \Sigma^*$ is a word in Σ.

For the systems under consideration, we can think of the following three different kinds of possible executions.

- An execution eventually hitting $\checkmark$, that is, $x \overset{a_1}{\to} \cdots \overset{a_n}{\to} x_n \to \checkmark$. By the condition (2.23) it yields a word $\mathsf{tr}(c)(x) = a_1 \dots a_n$ as its trace.
- An execution eventually hitting $\bot$, that is, $x \overset{a_1}{\to} \cdots \overset{a_n}{\to} x_n \to \bot$. By the third line of (2.23) we see that $\mathsf{tr}(c)(x_n) = \bot$; moreover

  $$\mathsf{tr}(c)(x_{n-1}) = \mathsf{tr}(c)(x_{n-2}) = \cdots = \mathsf{tr}(c)(x) = \bot \ .$$

  It properly reflects our intuition that a state x that eventually goes into deadlock does not yield a finite (i.e. "terminating") trace.

- An execution not hitting ✓ nor $\perp$, that is, $x \overset{a_1}{\to} x_1 \overset{a_2}{\to} \cdots$. In this case, the only possible solution of the "recursive equation" (2.23) is $\mathsf{tr}(c)(x) = \mathsf{tr}(c)(x_1) = \cdots = \perp$. The intuition here is: a state leading to *livelock* does not yield a finite trace.

2.4.2 Infinite traces

The trace semantics obtained via coinduction (Definition 2.4.1) assigns, to each state $x \in X$, "a set of" (if $T = \mathcal{P}$) or "a distribution over" (if $T = \mathcal{D}$) elements of the initial algebra A. Elements of A are thought of as possible linear behaviors of the system determined by the transition type (i.e. the functor F).

Now the intuition is that an initial F-algebra A consists of the well-founded (or finite-depth) terms and a final F-coalgebra Z consists of the possibly non-well-founded (or infinite-depth) terms. For example,

- for $F = 1 + \Sigma \times \_$, $A = \Sigma^*$ consists of all the finite words, and $Z = \Sigma^\infty = \Sigma^* + \Sigma^\omega$ is augmented with *streams*, i.e. infinite words;
- for $F = (\Sigma + \_)^*$, A is the set of finite-depth *skeletal parse trees* (see [43]), and Z additionally contains infinite-depth ones;
- for $F = \Sigma \times \_$ which models LTSs *without* explicit termination, $A = 0$ and $Z = \Sigma^\omega$.

Therefore our trace semantics $X \to TA$ takes account of only finite-depth, well-founded linear-time behaviors but not infinite-depth ones. This is why the trace set (2.2) does not contain ab^ω; it is also why we have been talking about LTSs *with* explicit termination—otherwise the finite trace semantics is always empty.

Designing a coalgebraic framework to capture possibly infinite trace semantics is the main aim of [62]. The work is done exclusively in a non-deterministic setting and the main result reads as follows.

2.4.4 Theorem (Possibly infinite trace semantics for coalgebras, [62]). *Let F be a shapely functor on* **Sets**, *and $\zeta : Z \overset{\cong}{\to} FZ$ be a final coalgebra in* **Sets**. *The coalgebra*

$$J\zeta \;:\; Z \longrightarrow \overline{F}Z \qquad \text{in } \mathcal{K}\ell(\mathcal{P})$$

is weakly final: that is, given a coalgebra $c : X \to \overline{F}X$, there is a morphism from c to $J\zeta$ but the morphism is not necessarily unique.

$$\begin{tikzcd}
\overline{F}X \arrow[r, squiggly, "\overline{F}(\mathsf{tr}^\infty(c))"] & \overline{F}Z & \text{in } \mathcal{K}\ell(\mathcal{P}) \\
X \arrow[u, "c"] \arrow[r, squiggly, "\mathsf{tr}^\infty(c)"'] & Z \arrow[u, "J\zeta"', "\cong"]
\end{tikzcd} \tag{2.24}$$

*Still there is a canonical choice $\mathsf{tr}^\infty(c)$ among such morphisms, namely the one which is maximal with respect to the inclusion order. We shall call the function $\mathsf{tr}^\infty(c) : X \to \mathcal{P}Z$ the **possibly-infinite trace semantics** for c.* □

Note here that, when we take $F = 1 + \Sigma \times \_$ and $T = \mathcal{P}$ (the choice for LTSs with termination), commutativity of (2.24) boils down to exactly the same conditions as (2.3):

$$\begin{array}{rcl} \langle\rangle \in \mathsf{tr}^{\infty}(c)(x) & \iff & x \to \checkmark , \\ a \cdot \sigma \in \mathsf{tr}^{\infty}(c)(x) & \iff & \exists y. \, (\, x \xrightarrow{a} y \ \wedge \ \sigma \in \mathsf{tr}^{\infty}(c)(y) \,) . \end{array} \tag{2.25}$$

Weak finality of $\Sigma^{\infty} \xrightarrow{\cong} 1 + \Sigma \times \Sigma^{\infty}$ (corresponding to $Z \xrightarrow{\cong} \overline{F}Z$ in (2.24)) means the following. The recursive definition (2.25)—although it looks valid at the first sight—does *not* uniquely determine the infinite trace map $\mathsf{tr}^{\infty}(c) : X \to \mathcal{P}(\Sigma^{\infty})$. Instead, the map $\mathsf{tr}^{\infty}(c)$ we want is the maximal one among those which satisfy (2.25).

As an example take the first system in (2.1). We expect its possibly-infinite trace map $X \to \mathcal{P}(\Sigma^{\infty})$ to be such that $x \mapsto ab^* + ab^{\omega}$ and $y \mapsto b^* + b^{\omega}$. Indeed this satisfies (2.25) and is moreover the maximal. However, the function $x \mapsto ab^*$ and $y \mapsto b^*$—this is actually the finite trace $X \to \mathcal{P}(\Sigma^*)$ embedded along $\Sigma^* \hookrightarrow \Sigma^{\infty}$—also satisfies (2.25). In fact, [43, Section 5] shows a general fact that such an embedding of the finite trace map is the minimal one among those morphisms which make the diagram (2.24) commute.

The coalgebraic characterization (Theorem 2.4.4) of possibly-infinite trace semantics is not yet fully developed. In particular the current proof of Theorem 2.4.4 (in [62]) is fairly concrete and a categorical principle behind it is less clear than the one behind finite traces. Consequently the result's applicability is limited: we do not know whether the result holds in a probabilistic setting; or whether it holds for any weak-pullback-preserving functor F.

2.4.3 Accepted languages via determinization

In Example 2.4.2 we illustrated that our coalgebraic framework successfully captures trace semantics for (non-deterministic) labeled transition system with explicit termination, that is, accepted languages of non-deterministic automata (see Remark 2.2.2). There is another standard way to capture accepted language by coinduction, namely via *determinization* of non-deterministic automata. It appears for example in [116]; here we briefly describe the construction.

The construction is in fact the standard determinization procedure—which transforms a non-deterministic automaton (NDA) with a state space X into a deterministic one (DA) with a state space $\mathcal{P}X$—described coalgebraically. Deterministic automata are modeled as coalgebras for the functor $G = 2 \times (\_)^{\Sigma}$ to which, as easy calculation shows, coinduction assigns accepted languages. Therefore what we do is to transform an NDA to a DA, and apply coinduction to the latter. In its course we will utilize the isomorphism (2.10) without the finiteness assumption. That is,

$$2 \times \mathcal{P}(\Sigma \times X) \xrightarrow{\cong} \mathcal{P}(1 + \Sigma \times X) \quad . \tag{2.26}$$

In the sequel we put $F = 1 + \Sigma \times \_$ and $G = 2 \times (\_)^{\Sigma}$. Hence (2.26) establishes an isomorphism $G\mathcal{P} \overset{\cong}{\Longrightarrow} \mathcal{P}F$. It is easy to see its naturality.

Given a NDA $X \xrightarrow{c} \mathcal{P}FX$, we transform it to a DA (i.e. a G-coalgebra) in the following way.

$$\mathcal{P}X \xrightarrow{\mathcal{P}c} \mathcal{P}^2FX \xrightarrow{\mu^{\mathcal{P}}_{FX}} \mathcal{P}FX \xrightarrow{\cong} G\mathcal{P}X$$

We denote this composite by $\mathsf{det}(c)$. Now it is standard that a final G-coalgebra is carried by the set $\mathcal{P}(\Sigma^*)$ of *languages* over Σ, with its structure given as follows.

$$\begin{tikzcd} 2 \times \big(\mathcal{P}(\Sigma^*)\big)^{\Sigma} \\ \mathcal{P}(\Sigma^*) \arrow[u, "{\langle \epsilon, \delta \rangle}"'] \end{tikzcd} \qquad \text{with} \qquad \begin{array}{lcl} \epsilon(L) = \mathbf{T} & \iff & \langle\rangle \in L \ , \\ \delta(L)(a) = \{\sigma \mid a \cdot \sigma \in L\} & & . \end{array}$$

The operation δ is called the *Brzozowski derivative*. For more details see [67, 115].

Now by coinduction—remember that we are in **Sets** now—a G-coalgebra $\mathsf{det}(c)$ induces the unique map below. It assigns, to each "state" $S \in \mathcal{P}X$, its accepted language.

$$\begin{tikzcd} & G\mathcal{P}X \arrow[r, dashed] & G\mathcal{P}(\Sigma^*) \\ X \arrow[r, "{\eta^{\mathcal{P}} = \{\_\}}"'] & \mathcal{P}X \arrow[u, "\mathsf{det}(c)"] \arrow[r, dashed] & \mathcal{P}(\Sigma^*) \arrow[u, "\cong"', "{\langle \epsilon, \delta \rangle}"] \end{tikzcd}$$

Our original interest is the accepted language of each single state $x \in X$. This is obtained by the above map, by pre-composing $\eta^{\mathcal{P}} : X \longrightarrow \mathcal{P}X$ which yields singletons.

This much said, its relation to our general Kleisli framework is not yet clear. The above determinization construction relies on the following ingredients:

- a natural isomorphism $G\mathcal{P} \overset{\cong}{\Longrightarrow} \mathcal{P}F$;
- a final G-coalgebra is carried by $\mathcal{P}A$ where A carries an initial F-algebra.

These assumptions already look quite strong and we are yet to see how to generalize them.

2.5 Trace semantics as testing equivalence

In this section we will observe that, in a non-deterministic setting, the coalgebraic finite trace semantics (i.e. coinduction in $\mathcal{K}\ell(\mathcal{P})$) gives rise to a canonical *testing situation* in which a test is an element of an initial F-algebra A in **Sets**. Here F specifies the transition type, just as before. The notion of testing situations (Definition 2.5.1) and its variants have attracted many authors' attention in the context of coalgebraic modal logic; our aim here is to demonstrate genericity and pervasiveness of the notion of testing situations by presenting an example which is not much like modal logic (that is, propositional logic plus modality).

In Section 2.5.1 we introduce the notion of testing situations and investigate some of their general properties. Our main concern there is the comparison between two process equivalences, namely *testing equivalence* and *equivalence modulo final coalgebra semantics*. We present the equivalences categorically as suitable kernel pairs; this makes the arguments simple and clean. In Section 2.5.2 we present the canonical testing situation for trace semantics. Moreover we show that it is *expressive*: the testing captures final coalgebra semantics, which is now trace semantics.

2.5.1 Testing situations

Recent studies [17, 18, 80, 84, 87, 109] on coalgebra and modal logic have identified (variants of) the following categorical situation as the essential underlying structure. Following [109], we prefer using a more general term "testing": it subsumes "modal logic" in the following sense. We learn properties of a system through pass or failure of *tests*; modal logic constitutes a special case where tests are modal formulas.

2.5.1 Definition. A *testing situation* is the following situation of a contravariant adjunction $S^{\mathrm{op}} \dashv P$ and two endofunctors F, M

$$\begin{tikzcd}
\mathbb{C}^{\mathrm{op}} \arrow[loop left, "F^{\mathrm{op}}"] \arrow[r, bend left, "P"] \arrow[r, phantom, "\top"] & \mathbb{A} \arrow[l, bend left, "S^{\mathrm{op}}"] \arrow[loop right, "M"]
\end{tikzcd} \tag{2.27}$$

plus a "denotation" natural transformation $\delta : MP \Rightarrow PF^{\mathrm{op}} : \mathbb{C}^{\mathrm{op}} \to \mathbb{A}$, which consists of arrows $MPX \xrightarrow{\delta_X} PFX$ in $\mathbb{A}$.

Note that the denotation δ is a parameter: the same "syntax for tests" $M : \mathbb{A} \to \mathbb{A}$ can have different interpretations with different δ.

The requirements in Definition 2.5.1 are the same as in [80, 109]. They are what we need to compare two process semantics, namely *testing equivalence*—which arises naturally from the concept of testing—and final coalgebra semantics.[8] We shall explain each ingredient's role, using the well-established terminology of modal logic.

- The endofunctor $F : \mathbb{C} \to \mathbb{C}$ makes $\mathbf{Coalg}_F$ the category of "systems," or "Kripke models" in modal logic.
- The category $\mathbb{A}$—typical examples being **Bool** of Boolean algebras or **Heyt** of Heyting algebras—is that of "propositional logic." The functor M specifies "modality": modal operators and axioms. Then $\mathbf{Alg}_M$ is the category of "modal algebras"; an initial M-algebra $ML \overset{\cong}{\to} L$ is a "modal logic" consisting of modal formulas, modulo logical equivalence.

[8] In fact we can be even more liberal: existence of a denotation δ can be replaced by existence of a lifting $\hat{P} : \mathbf{Coalg}_F^{\mathrm{op}} \to \mathbf{Alg}_M$ of P. The results in this section nevertheless hold in that case. The latter condition (there is a lifting $\hat{P}$) is strictly weaker than the former (there is a natural transformation δ): obviously δ induces $\hat{P}$ but not the other way round. Let $\mathbb{C} = \omega^{\mathrm{op}}, \mathbb{A} = \omega, P = \mathrm{id}, F = (1 + \_)^{\mathrm{op}}$ and $M = 2 + \_$. Then both $\mathbf{Coalg}_F$ and $\mathbf{Alg}_M$ are the empty category hence P has the trivial lifting. However there is no natural transformation $MPX \to PF^{\mathrm{op}}X$.

- The denotation δ specifies how the modality M is interpreted via transitions of type F. This allows to give “Kripke semantics” for the modal logic: given a coalgebra (or a “Kripke model”) $c : X \to FX$, interpretation $[\![\_]\!]_c$ of modal formulas therein is given by the following induction.

$$\begin{tikzcd}
ML \arrow[r, dashed] \arrow[dd, "\text{initial}"', "\cong"] & MPX \arrow[d, "\delta_X"] \\
 & PFX \arrow[d, "Pc"] \\
L \arrow[r, dashed, "{[\![\_]\!]_c}"'] & PX
\end{tikzcd} \tag{2.28}$$

- Why a right adjoint S of P^{op}? It allows us, via transposition, to assign a modal “theory” to each state of a Kripke model.

$$\frac{L \xrightarrow{[\![\_]\!]_c} PX \quad \text{in } \mathbb{A}}{X \xrightarrow{\mathsf{th}_c} SL \quad \text{in } \mathbb{C}} \; (S^{\mathrm{op}} \dashv P) \tag{2.29}$$

  The theory $\mathsf{th}_c(x)$ associated with a state x contains precisely the modal formulas that hold at x.

Following the above intuition, we define the categorical notion of testing equivalence: two states are testing-equivalent if they have the same modal theory.

2.5.2 Definition. Assume that we have a testing situation (2.27), and that $\mathbb{C}$ has finite limits. On a coalgebra $X \overset{c}{\to} FX$, the *testing equivalence* TestEq_c is the kernel pair of the theory map th_c defined by (2.28) and (2.29). Equivalently,

$$\begin{tikzcd}
\mathsf{TestEq}_c \arrow[r, tail, "{\langle p_1, p_2 \rangle}"] & X \times X \arrow[r, shift left, "\mathsf{th}_c \circ \pi_1"] \arrow[r, shift right, "\mathsf{th}_c \circ \pi_2"'] & SL
\end{tikzcd} \tag{2.30}$$

is an equalizer.

Similarly, we introduce the categorical notion of “equivalence modulo final coalgebra semantics”; we shall call it *FCS-equivalence* for short.

2.5.3 Definition. Assume that there is a final F-coalgebra $\zeta : Z \overset{\cong}{\to} FZ$, and that $\mathbb{C}$ has finite limits. On a coalgebra $X \overset{c}{\to} FX$, the *FCS-equivalence* FCSEq_c is the kernel pair of the unique map $\mathsf{beh}(c) : X \to Z$ induced by finality. Equivalently,

$$\begin{tikzcd}
\mathsf{FCSEq}_c \arrow[r, tail, "{\langle q_1, q_2 \rangle}"] & X \times X \arrow[r, shift left, "\mathsf{beh}(c) \circ \pi_1"] \arrow[r, shift right, "\mathsf{beh}(c) \circ \pi_2"'] & Z
\end{tikzcd} \tag{2.31}$$

is an equalizer.

It is easily seen that the two "relations" TestEq_c and FCSEq_c on X are *equivalence relations* in the sense of [61, Section 1.3]. That is, they satisfy the reflexivity, symmetry, and transitivity conditions when the conditions are suitably formulated in categorical terms.

Now our concern is the comparison between two process semantics TestEq_c and FCSEq_c, as subobjects of $X \times X$. The following lemma is crucial for our investigation; in fact it is important for coalgebraic modal logic in general and appears e.g. as [80, Theorem 3.3].

2.5.4 Lemma. *A morphism of F-coalgebras preserves theory maps. That is,*

$$\begin{tikzcd} FX \arrow[r, "Ff"] & FY \\ X \arrow[u, "c"] \arrow[r, "f"'] & Y \arrow[u, "d"'] \end{tikzcd} \quad \textit{implies} \quad \begin{tikzcd} X \arrow[d, "f"'] \arrow[rd, "\mathsf{th}_c"] & \\ Y \arrow[r, "\mathsf{th}_d"'] & SL \end{tikzcd} \quad .$$

Proof. The following induction diagram proves $Pf \circ [\![\_]\!]_d = [\![\_]\!]_c$. Naturality of δ plays an important role there.

$$\begin{tikzcd} ML \arrow[r, dashed, "Ff"] \arrow[dd, "\text{initial}"', "\cong"] & MPY \arrow[r, "MPf"] \arrow[d, "\delta_Y"] & MPX \arrow[d, "\delta_X"] \\ & PFY \arrow[r, "PFf"'] \arrow[d, "Pd"] & PFX \arrow[d, "Pc"] \\ L \arrow[r, dashed, "{[\![\_]\!]_d}"'] & PY \arrow[r, "Pf"'] & PX \end{tikzcd} \quad .$$

Then the claim follows from naturality of the transposition (2.29). □

We show that in a testing situation like (2.27), tests respect final coalgebra semantics. That is, testing does not distinguish two FCS-equivalent states.

2.5.5 Proposition. *Consider such a testing situation and equivalence relations as in Definitions 2.5.2 and 2.5.3. For any coalgebra $X \xrightarrow{c} FX$ we have an inclusion*

$$\mathsf{FCSEq}_c \leq \mathsf{TestEq}_c$$

of subobjects of $X \times X$.

Proof. It suffices to show that the arrow $\langle q_1, q_2 \rangle$ in (2.31) equates the parallel arrows in (2.30); then the claim follows from universality of an equalizer.

$$\begin{aligned} \mathsf{th}_c \circ \pi_1 \circ \langle q_1, q_2 \rangle &= \mathsf{th}_c \circ q_1 \\ &= \mathsf{th}_\zeta \circ \mathsf{beh}(c) \circ q_1 && (*) \\ &= \mathsf{th}_\zeta \circ \mathsf{beh}(c) \circ q_2 && \text{due to (2.31)} \\ &= \mathsf{th}_c \circ q_2 && (*) \\ &= \mathsf{th}_c \circ \pi_2 \circ \langle q_1, q_2 \rangle \ . \end{aligned}$$

Here $(*)$ is an instance of Lemma 2.5.4: $\mathsf{beh}(c)$ is a morphism of coalgebras from c to the final ζ. □

The converse $\mathsf{TestEq}_c \leq \mathsf{FCSEq}_c$ does not hold in general. For a fixed type of systems (i.e. for fixed $F : \mathbb{C} \to \mathbb{C}$), we can think of logics with varying degree of expressive power; this results in process equivalences with varying granularity. This view is systematically presented by van Glabbeek in [39] as the *linear time-branching time spectrum*—a categorical version of which we consider as an important direction of future work.

It is when we have $\mathsf{FCSEq}_c \overset{\cong}{\Rightarrow} \mathsf{TestEq}_c$ that a modal logic (considered as a testing situation) is said to be *expressive.* Recall that FCSEq_c usually coincides with bisimilarity if $\mathbb{C}$ is **Sets**: in this case an expressive logic *captures* bisimilarity.

The following proposition states a (rather trivial) equivalent condition for a testing situation to be expressive. For more ingenious sufficient conditions—which essentially rely on the transpose of δ being monic—see e.g. [80].

2.5.6 Proposition. *Consider a testing situation as in Definitions 2.5.2 and 2.5.3. The testing is expressive, that is, for any coalgebra $X \xrightarrow{c} FX$ we have*

$$\mathsf{TestEq}_c \overset{\cong}{\Rightarrow} \mathsf{FCSEq}_c$$

as subobjects of $X \times X$, if and only if the theory map $\mathsf{th}_\zeta : Z \to SL$ for the final coalgebra is a mono.

Proof. We first prove the "if" direction. In view of Proposition 2.5.5, it suffices to show that $\langle p_1, p_2\rangle$ in (2.30) equalizes $\mathsf{beh}(c) \circ \pi_1$ and $\mathsf{beh}(c) \circ \pi_2$ (which proves $\mathsf{TestEq}_c \leq \mathsf{FCSEq}_c$).

$$\begin{aligned}
\mathsf{th}_\zeta \circ \mathsf{beh}(c) \circ p_1 &= \mathsf{th}_c \circ p_1 && \text{by Lemma 2.5.4}\\
&= \mathsf{th}_c \circ p_2 && \text{due to (2.30)}\\
&= \mathsf{th}_\zeta \circ \mathsf{beh}(c) \circ p_2 && \text{by Lemma 2.5.4}
\end{aligned}$$

We have $\mathsf{beh}(c) \circ p_1 = \mathsf{beh}(c) \circ p_2$ since th_ζ is a mono.

To prove the "only if" direction, first we observe that the FCS-equivalence on the final coalgebra $\zeta : Z \overset{\cong}{\Rightarrow} FZ$ is the diagonal relation: that is,

$$\begin{tikzcd}
\mathsf{FCSEq}_\zeta \arrow[r, tail] \arrow[d, "\cong"'] & Z \times Z \arrow[r, shift left, "\mathsf{beh}(\zeta)\circ\pi_1"] \arrow[r, shift right, "\mathsf{beh}(\zeta)\circ\pi_2"'] & Z \\
Z \arrow[ur, tail, "\langle \mathrm{id},\mathrm{id}\rangle"'] & &
\end{tikzcd} .$$

This is because $\mathsf{beh}(\zeta) = \mathrm{id} : Z \to Z$. Now assume that $\mathsf{th}_\zeta \circ k = \mathsf{th}_\zeta \circ l$ for $k, l : Y \rightrightarrows Z$. Universality of an equalizer TestEq_ζ induces a mediating arrow m in the following diagram.

$$\begin{tikzcd}
Y \arrow[rr, "\langle k, l\rangle"] \arrow[d, dashed, "m"'] & & Z \times Z \arrow[r, shift left, "\mathsf{beh}(\zeta) \circ \pi_1"] \arrow[r, shift right, "\mathsf{beh}(\zeta) \circ \pi_2"'] & Z \\
\mathsf{TestEq}_\zeta \arrow[urr, tail] \arrow[r, tail, "\cong"'] & \mathsf{FCSEq}_\zeta \arrow[r, "\cong"'] & Z \arrow[u, tail, "\langle \mathrm{id}, \mathrm{id}\rangle"] &
\end{tikzcd}$$

The whole diagram commutes since $\mathsf{TestEq}_\zeta \cong \mathsf{FCSEq}_\zeta$ (by assumption) and $\mathsf{FCSEq}_\zeta \cong Z$ (by the above observation), both as subobjects of $Z \times Z$. This proves $k = l$. □

2.5.7 Remark. The literature [17, 18] considers more restricted settings than the testing situations in Definition 2.5.1. There an adjunction $S^{\mathrm{op}} \dashv P$ is replaced by a dual equivalence of categories, and a denotation δ is required to be a natural isomorphism. These additional restrictions allow one to say more about the situations: logics are always expressive; the main concern of [18] is how to present an abstract modal logic $M : \mathbb{A} \to \mathbb{A}$ by concrete syntax. However, for our purpose in Section 2.5.2 the greater generality of our notion of testing situations is needed.

2.5.2 Canonical testing for trace semantics in $\mathcal{K}\ell(\mathcal{P})$

In this section we shall present a canonical testing situation for coalgebras in $\mathcal{K}\ell(\mathcal{P})$. We shall also show that the testing is "expressive," in the sense that the testing captures final coalgebra semantics. The intuition is as follows.

Trace semantics for non-deterministic systems assigns to each system c its "(finite) trace set" map $\mathsf{tr}(c) : X \to \mathcal{P}A$, where A carries an initial algebra in **Sets**. This suggests a natural testing framework where: an element t of A is a test; a state $x \in X$ of a system passes a test t if and only if the trace set of x includes t (i.e. $x \models t \iff t \in \mathsf{tr}(c)(x)$). An important point here is that A, carrying an initial algebra in **Sets**, usually gives a *well-founded syntax* for tests.[9]

We focus on a non-deterministic setting (i.e. $T = \mathcal{P}$) in this section and leave a probabilistic one as future work. Although the above intuition is true in probabilistic settings as well—where the 2-valued (pass/failure) observation scheme is replaced by the refined $[0,1]$-valued one—we do not know yet how to extend the current material to probabilistic settings. The difficulty is that the category $\mathcal{K}\ell(\mathcal{D})$ is not self-dual, as opposed to $\mathcal{K}\ell(\mathcal{P})$; see (2.32) below.

The canonical testing situation which captures finite trace semantics is the following one.

$$\begin{tikzcd}
\mathcal{K}\ell(\mathcal{P})^{\mathrm{op}} \arrow[loop below, "\overline{F}^{\mathrm{op}}"] \arrow[r, bend left, "Op"] \arrow[r, phantom, "\cong"] & \mathcal{K}\ell(\mathcal{P}) \arrow[l, bend left, "Op^{\mathrm{op}}"] \arrow[r, bend left, "K"] \arrow[r, phantom, "\top"] & \mathbf{Sets} \arrow[l, bend left, "J"] \arrow[loop above, "F"]
\end{tikzcd} \tag{2.32}$$

Here $J \dashv K$ is the canonical Kleisli adjunction. Recall the self duality $Op : \mathcal{K}\ell(\mathcal{P})^{\mathrm{op}} \overset{\cong}{\Rightarrow} \mathcal{K}\ell(\mathcal{P})$ from Section 2.3.2. The denotation is given by (the components of) the distributive law $\lambda : F\mathcal{P} \Rightarrow \mathcal{P}F$. The following lemma establishes that the denotation thus defined is indeed a natural transformation.

[9] Recall the construction of an initial algebra in **Sets** via the initial sequence (Proposition A.1.1). The set A is the colimit (*union* in **Sets**) of the initial sequence $0 \to F0 \to F^2 0 \to \cdots$. Each $F^n 0$ can be thought of as the set of terms with depth $\leq n$.

2.5.8 Lemma. *Let $F : \mathbf{Sets} \to \mathbf{Sets}$ be a functor which preserves weak pullbacks, and $\overline{F}$ be its lifting induced by the relation lifting (Lemma 2.2.4). Then the components $F\mathcal{P}X \overset{\lambda_X}{\to} \mathcal{P}FX$ of the corresponding distributive law λ also form a natural transformation*

$$F \circ K \circ Op \Longrightarrow K \circ Op \circ \overline{F}^{\mathrm{op}} \ :\ \mathcal{K}\ell(\mathcal{P})^{\mathrm{op}} \longrightarrow \mathbf{Sets}\ .$$

Proof. The desired natural transformation is obtained from another natural transformation

$$\lambda' \ :\ FK \Longrightarrow K\overline{F} \ :\ \mathcal{K}\ell(\mathcal{P}) \longrightarrow \mathbf{Sets}$$

which we describe in a moment, by post-composing the functor Op. That is, the desired one is the composite

$$FKOp \overset{\lambda' \circ Op}{\Longrightarrow} K\overline{F}Op \overset{(*)}{=} KOp\overline{F}^{\mathrm{op}}\ ,$$

or equivalently, in a 2-categorical presentation,

$$\begin{tikzcd}
\mathcal{K}\ell(\mathcal{P})^{\mathrm{op}} \arrow[r, "Op"] \arrow[d, "\overline{F}^{\mathrm{op}}"'] \arrow[dr, phantom, "(*)/\!/"] & \mathcal{K}\ell(\mathcal{P}) \arrow[r, "K"] \arrow[d, "\overline{F}"'] \arrow[dr, phantom, "\Swarrow\lambda'"] & \mathbf{Sets} \arrow[d, "F"] \\
\mathcal{K}\ell(\mathcal{P})^{\mathrm{op}} \arrow[r, "Op"'] & \mathcal{K}\ell(\mathcal{P}) \arrow[r, "K"'] & \mathbf{Sets}\ .
\end{tikzcd}$$

Here the equality $(*)$ is the one in (2.20).

Now we describe the natural transformation λ'. Its components are given by those of λ; naturality of λ' is an easy consequence of λ's being a distributive law. Indeed, given an arrow $f : X \to Y$ in $\mathcal{K}\ell(\mathcal{P})$, the following shows that the naturality square commutes.

$$\begin{aligned}
K\overline{F}f \circ \lambda_X &= \mu^{\mathcal{P}}_{FY} \circ \mathcal{P}\overline{F}f \circ \lambda_X && \text{definition of } K \\
&= \mu^{\mathcal{P}}_{FY} \circ \mathcal{P}\lambda_Y \circ \mathcal{P}Ff \circ \lambda_X && \text{definition of } \overline{F} \\
&= \mu^{\mathcal{P}}_{FY} \circ \mathcal{P}\lambda_Y \circ \lambda_{\mathcal{P}Y} \circ F\mathcal{P}f && \text{naturality of } \lambda \\
&= \lambda_Y \circ F\mu^{\mathcal{P}}_Y \circ F\mathcal{P}f && \lambda \text{ is compatible with multiplication } \mu^{\mathcal{P}} \\
&= \lambda_Y \circ FKf && \text{definition of } K
\end{aligned}$$

□

The previous lemma establishes that the situation (2.32) is indeed a testing situation as defined in Definition 2.5.1.

In the previous Section 2.5.1, the use of testing situations is demonstrated through comparing testing equivalence and final coalgebra semantics, both described as suitable kernel pairs. Unfortunately this argument is not valid in the current situation (2.32), since the category $\mathcal{K}\ell(\mathcal{P})$ does not have kernel pairs.

Still, we shall claim that the situation (2.32) is "expressive," in the sense that final coalgebra semantics is captured by testing. This claim is supported by the following fact: in the current situation the two arrows $\mathsf{tr}(c)$ and th_c simply coincide. Therefore their kernel relations—in any reasonable formalization—should coincide as well.

2.5.9 Proposition. *Let $X \overset{c}{\to} \overline{F}X$ be a coalgebra in $\mathcal{K}\ell(\mathcal{P})$. In the testing situation (2.32), the following arrows in $\mathcal{K}\ell(\mathcal{P})$ coincide.*

- $\mathsf{tr}(c) : X \to A$, *giving the final coalgebra (trace) semantics for c.*
- $\mathsf{th}_c : X \to A$, *giving the testing semantics, i.e. the set of passed tests.*

Therefore the testing is "expressive": tests from an initial F-algebra captures trace semantics (which is via a final $\overline{F}$-coalgebra).

Here A is the carrier of an initial F-algebra, hence that of a final $\overline{F}$-coalgebra. Note that, in the general setting in Section 2.5.1, the codomains of $\mathsf{tr}(c)$ and th_c need not coincide.

Proof. We shall show that the transpose

$$\mathsf{tr}(c)^\vee \;:\; A \longrightarrow \mathcal{P}X \qquad \text{in } \mathbf{Sets}$$

of $\mathsf{tr}(c)$ under the adjunction in (2.32) makes the diagram (2.28)—which defines $[\![\_]\!]_c$—commute. This proves $\mathsf{tr}(c)^\vee = [\![\_]\!]_c$, hence $\mathsf{tr}(c) = {[\![\_]\!]_c}^\vee = \mathsf{th}_c$.

First note that the transpose $\mathsf{tr}(c)^\vee : A \to \mathcal{P}X$, when it is thought of as an arrow in $\mathcal{K}\ell(\mathcal{P})$, coincides with the arrow $\mathit{Op}(\mathsf{tr}(c)) : A \to X$. In the sequel we shall write $\mathit{Op}(\mathsf{tr}(c))$ for $\mathsf{tr}(c)^\vee$.

Commutativity of the diagram (2.21)—defining $\mathsf{tr}(c)$—yields the following equality.

$$\mathit{Op}(\mathsf{tr}(c)) \circ \mathit{Op}(J\alpha^{-1}) = \mathit{Op}(c) \circ \mathit{Op}(\overline{F}^{\mathrm{op}}\mathsf{tr}(c)) \qquad \text{in } \mathcal{K}\ell(\mathcal{P}).$$

By the definition of composition in $\mathcal{K}\ell(\mathcal{P})$, it reads as follows in **Sets**.

$$\mu_X \circ \mathcal{P}(\mathit{Op}(\mathsf{tr}(c))) \circ \mathit{Op}(J\alpha^{-1}) = \mu_X \circ \mathcal{P}(\mathit{Op}(c)) \circ \mathit{Op}(\overline{F}^{\mathrm{op}}\mathsf{tr}(c)) \tag{2.33}$$

We use this equality in showing that $\mathit{Op}(\mathsf{tr}(c))$ makes the diagram (2.28) commute.

$$\begin{aligned}
\mathit{Op}(\mathsf{tr}(c)) \circ \alpha &= \mu_X \circ \eta_X \circ \mathit{Op}(\mathsf{tr}(c)) \circ \alpha && \text{unit law} \\
&= \mu_X \circ \mathcal{P}(\mathit{Op}(\mathsf{tr}(c))) \circ \eta_A \circ \alpha && \text{naturality of } \eta \\
&= \mu_X \circ \mathcal{P}(\mathit{Op}(\mathsf{tr}(c))) \circ \mathit{Op}(J\alpha^{-1}) && \mathit{Op}(J\alpha^{-1}) = J\alpha = \eta_A \circ \alpha \\
&= \mu_X \circ \mathcal{P}(\mathit{Op}(c)) \circ \mathit{Op}(\overline{F}^{\mathrm{op}}\mathsf{tr}(c)) && \text{by (2.33)} \\
&= \mu_X \circ \mathcal{P}(\mathit{Op}(c)) \circ \overline{F}\mathit{Op}(\mathsf{tr}(c)) && \mathit{Op}\overline{F}^{\mathrm{op}} = \overline{F}\mathit{Op},\ (2.20) \\
&= \mu_X \circ \mathcal{P}(\mathit{Op}(c)) \circ \lambda_X \circ F\mathit{Op}(\mathsf{tr}(c)) && \text{definition of } \overline{F} \\
&= K\mathit{Op}(c) \circ \lambda_X \circ F\mathit{Op}(\mathsf{tr}(c)) \;.
\end{aligned}$$

Recall that M in (2.28) is now F; P in (2.28) is now $K\mathit{Op}$. This concludes the proof. □

The proposition establishes a connection between two semantics for $\overline{F}$-coalgebras in $\mathcal{K}\ell(\mathcal{P})$, namely: $\mathsf{tr}(c)$ via a final $\overline{F}$-coalgebra, and th_c via an initial F-algebra. One may well say that it is a "degenerate" case because, as we have shown in Section 2.3, coinduction in $\mathcal{K}\ell(\mathcal{P})$ and induction in **Sets** are essentially the same principle. Our emphasis is more on the fact that the coincidence of induction and coinduction yields a rather uncommon example of testing situations. Testing situations are of interest in modal logic—where the underlying contravariant adjunction $S^{\mathrm{op}} \dashv P : \mathbb{A} \to \mathbb{C}^{\mathrm{op}}$ in (2.27) is often the Stone duality or one of its variants. Our example $\mathcal{K}\ell(\mathcal{P})^{\mathrm{op}} \leftrightarrows \mathbf{Sets}$ here does not look like one of those familiar examples.

2.6 Summary and future work

We have developed a mathematical principle underlying "trace semantics" for various kinds of branching systems, namely coinduction in a Kleisli category. This general view is supported by a technical result that a final coalgebra in a Kleisli category is induced by an initial algebra in **Sets**.

The possible instantiations of our generic framework include non-deterministic systems and probabilistic systems, but do *not* yet include systems with both non-deterministic and probabilistic branching. The importance of having both of these branchings in system verification has been claimed by many authors e.g. [120, 137], with an intuition that probabilistic branching models the choices "made by the system, i.e. on *our* side," while (coarser) non-deterministic choices are "made by the (unknown) environment of the system, i.e. on the *adversary's* side." A typical example of such systems is given by *probabilistic automata* introduced by Segala [120].

In fact this combination of non-deterministic and probabilistic branching is a notoriously difficult one from a theoretical point of view [26,132,136]: many mathematical tools that are useful in a purely non-deterministic or probabilistic setting cease to work in the presence of both. For our framework of generic trace semantics, the problem is that we could not find a suitable monad T with an order structure.

We have used the order-enriched structure of a Kleisli category (expressing "more possibilities") to obtain the initial algebra-final coalgebra coincidence result. However, an order structure is not the only one that can yield such coincidence: other examples include metric, quasi-metric and quantale-enriched structures (in increasing generality). See e.g. [32, 135] for the potential use of such enriched structures in a coalgebraic setting. The relation of the current work to such structures is yet to be investigated.

In the discipline of process algebra, a system is represented by an *algebraic* term (such as $a.P \parallel a.Q$) and a structural operational semantics (SOS) rule determines its dynamics, that is, its *coalgebraic* structure. This is where "algebra meets coalgebra" and the interaction is studied e.g. in [12, 78, 134]. Later in chapter 5 we will claim the importance of the *microcosm principle* in this context and provide a "general compositionality theorem": under certain assumptions, final coalgebra semantics is "compositional" in a suitable sense. The results of the current chapter say that final coalgebra semantics can be interpreted as finite trace semantics (besides bisimilarity); hence the result in Chapter 5 instantiates to a compositionality result for trace semantics.

We have included some material—on possibly-infinite traces and testing situations—which, unfortunately, we have worked out only in a non-deterministic setting. A fully general account of these topics is left as future work.

Finally, there are so many different process semantics for branching systems, between two edges of bisimilarity and trace equivalence in the linear time-branching time spectrum [39]. How to capture them in a coalgebraic setting is, we believe, an important and challenging question.

Chapter 3

Generic forward and backward simulations

The technique of forward/backward simulations has been applied successfully in verification of distributed/concurrent systems. In this chapter, however, we claim that the technique has more potential generality and mathematical clarity. We do so by identifying forward/backward simulations as lax/oplax morphisms of coalgebras. Following 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 systems and probabilistic systems. In particular we prove soundness and completeness results for simulations with respect to trace inclusion: the proof is by coinduction using the generic theory of traces developed in the previous chapter. By suitably instantiating our generic framework, one obtains an appropriate definition of forward/backward simulations for a variety of systems, for which soundness and completeness come *for free*.

3.1 Overview

The theory of forward/backward simulations for non-deterministic automata has been extensively studied due to its significance in system verification (as described in Chapter 1). A notable example is a systematic study by Lynch and Vaandrager [94]. The technique of forward/backward simulations has been applied successfully in many distributed and concurrent applications, described as transition systems. For example, in [71] 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 [120].

We claim that the theory of forward/backward simulations has more potential generality 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: *a forward/backward simulation is a lax/oplax morphism of* *coalgebras in a Kleisli category.*

In $\mathcal{K}\ell(T)$

$$\begin{tikzcd}
FX \arrow[r, "Ff"] & FY & & FX \arrow[r, "Ff"] & FY \\
X \arrow[u, "c"] \arrow[r, "f"'] \arrow[ur, phantom, "\sqsupseteq"] & Y \arrow[u, "d"'] & & X \arrow[u, "c"] \arrow[r, "f"'] \arrow[ur, phantom, "\sqsubseteq"] & Y \arrow[u, "d"']
\end{tikzcd} \tag{3.1}$$

lax coalgebra morphism
= forward simulation

oplax coalgebra morphism
= backward simulation

An arrow f in (3.1) is called a *lax/oplax* morphism of coalgebras because it satisfies a condition which is similar to the one for a morphism of coalgebras, but has an inequality in the square (instead of an equality—which we usually do not put explicit).

$$\begin{tikzcd}
FX \arrow[r, "Ff"] & FY & FX \arrow[r, "Ff"] & FY & FX \arrow[r, "Ff"] & FY \\
X \arrow[u, "c"] \arrow[r, "f"'] \arrow[ur, phantom, "="] & Y \arrow[u, "d"'] & X \arrow[u, "c"] \arrow[r, "f"'] \arrow[ur, phantom, "\sqsupseteq"] & Y \arrow[u, "d"'] & X \arrow[u, "c"] \arrow[r, "f"'] \arrow[ur, phantom, "\sqsubseteq"] & Y \arrow[u, "d"']
\end{tikzcd}$$

morphism　　lax morphism　　oplax morphism

The use of categorical terms allows us to put various constructions in a mathematically clearer way. For example, the informal "duality" between forward and backward simulations is now put explicit, in the form of reversed inequalities ($\sqsupseteq$ vs. $\sqsubseteq$) in (3.1).

Based on this observation, we aim at presenting a generic version of the systematic study [94]. We employ the generic theory of traces in Chapter 2 and show:

- *Soundness.* Existence of a forward/backward/hybrid simulation from a system $\mathcal{S}$ to $\mathcal{T}$ implies trace inclusion $\mathsf{tr}(\mathcal{S}) \sqsubseteq \mathsf{tr}(\mathcal{T})$.
- *Completeness.* Trace inclusion implies existence of a certain kind of hybrid simulation, namely a backward-forward simulation.

Our generic, coalgebraic theory of forward/backward simulations in this chapter is built on top of the theory of trace semantics in Chapter 2. The mathematical setting stays the same: we work with $\overline{F}$-coalgebras in the category $\mathcal{K}\ell(T)$, where a functor $\overline{F} : \mathcal{K}\ell(T) \to \mathcal{K}\ell(T)$ is a lifting of $F : \mathbf{Sets} \to \mathbf{Sets}$. Just like in Chapter 2, we understand the functor F as a specification of "transition type"; the monad T as a specification of "branching type." Moreover, the order $\sqsubseteq$ in (3.1) refers to the **Cppo**-enriched structure of the Kleisli category, as the "more possibilities" order that we exploited in Chapter 2. The conditions imposed on these F and T remain the same. This results in the same (wide) variety of possible choices of parameters F and T, hence in the same applicability. In particular, our generic theory of simulations covers both non-deterministic systems as well as purely probabilistic systems (but not systems with both non-determinism and probability—as in Chapter 2).

For a variety of systems, our generic theory of simulations gives a definition of forward/backward simulations tailored to the specific kind of systems, as an instantiation of the coalgebraic definition (3.1). The resulting definition is (at least) a "useful" one, since for these simulations a soundness theorem comes *for free.* Soundness plays a crucial role in the simulation-based verification technique of trace-based properties (see Chapter 1); we will prove generic soundness theorems parametrized by F and T, once for all.

Now let us take the viewpoint of a coalgebra theorist. This chapter continues investigation of coalgebras in a Kleisli category which we started in Chapter 2, filling in the lower middle cell in the following table.

	morphism of coalgebras	coinduction gives
In **Sets**	functional bisimulation	bisimilarity
In $\mathcal{K}\ell(T)$	lax $\cdots$ forward simulation oplax $\cdots$ backward simulation [this chapter]	trace semantics [Chapter 2]

3.1.1 Bibliographical remarks

The novelty in this chapter is to identify lax/oplax coalgebra morphisms in a Kleisli category as forward/backward simulations. This view—with three keywords *lax morphism*, *Kleisli* and *simulation*—seems new to the best of our knowledge. However, a close relationship between lax morphisms and simulations (not necessarily in a Kleisli category) has been pointed out or hinted by many authors.

The work [139] uses lax/oplax coalgebra morphisms in obtaining a coalgebraic representation of the *Cone of Influence* reduction. The key observation there is, for coalgebras in **Sets** with their signature $F = \mathcal{P}$, a *graph* of a lax morphism is a simulation. The work [32, 135] formulate what they call *ordered categorical bisimulations* using lax coalgebra morphisms. The notion of ordered categorical bisimulations "resembles the notion of *bisimulation up to*" [32]; this comes close to simulations. Additionally, the work [21] describes users' visit to a website in two ways—namely as a suitable simulation and as a lax coalgebra morphism—hinting conceptual closeness between the two notions.

Besides, there are work that we are aware of which focuses on one of the three keywords listed above. For example, the work [58, 64] presents a coalgebraic formulation of simulations which is different from ours, starting from a given order structure in a signature functor F. The work [56, 76] investigates use of lax categorical structures in computer science. Details about these references, together with comparison of our notion of probabilistic simulations to some existing definitions, are found in Section 3.4.

3.2 Coalgebraic forward and backward simulations

In this section we introduce our coalgebraic presentations of forward and backward simulations. It turns out that, in defining these coalgebraic simulations, it is more appropriate to make *start states* explicit, although start states are usually left implicit in the study of coalgebras.

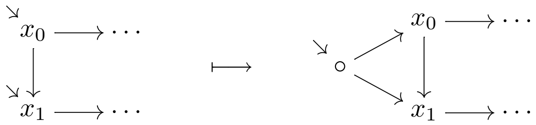
Therefore in Section 3.2.1 we start with introducing the notion of coalgebras in a Kleisli category accompanied with their (explicit) start states; this notion we shall call (T, F)*-systems.* In Section 3.2.2 the coalgebraic notion of (finite) trace semantics from Chapter 2 is extended to (T, F)-systems in a natural way. Section 3.2.3 presents our core definition of coalgebraic forward/backward simulations; in Section 3.2.4 we explain why we need to make start states explicit.

3.2.1 (T, F)-systems

For the reason we will discuss later in Section 3.2.4, in this chapter we shall work with coalgebras in $\mathcal{K}\ell(T)$ which additionally have explicit start states. When one think of "(a) start state(s)" of a branching system, there are two different possible approaches.

1. One single state of the system is its starting state. That is, there is no branching in choosing a start state.
2. There *is* branching in choosing a start state. If the system in question is non-deterministic, the notion of "start states" should be given by the set of those states which can possibly be chosen as an initial state of an execution. If the system is probabilistic then the notion of "start states" is given by a probability (sub)distribution over the state space—describing with what probability a state is chosen as an initial state of an execution.

These two approaches are equivalent in practice. The first (non-branching) "start state" is turned into the second kind by regarding it as a trivial branching; conversely a system with the second (branching) kind of "start states" is turned into a system with the first kind of "start states" by adding a fresh "start state" (∘ below) and transitions from it.



We shall take the second approach ("branching start state") since it is better accommodated in our setting in a Kleisli category. Specifically, let X be the set of states. A (branching) *start state map* is given by an arrow

$$1 \xrightarrow{s} X \quad \text{in } \mathcal{K}\ell(T) \ ,$$

which is the same thing as an arrow $1 \xrightarrow{s} TX$ in **Sets**, hence as an element $s \in TX$. When our system is non-deterministic ($T = \mathcal{P}$) the arrow s is hence a subset of X; when it is probabilistic ($T = \mathcal{D}$) the arrow s gives us a probability subdistribution over the set X. This matches the intuition described above.

This notion of start state maps $1 \xrightarrow{s} X$ is combined with coalgebras to give our presentation of state-based systems.

3.2.1 Definition ((T, F)-systems)**.** Let T be a monad and F be an endofunctor, both on **Sets**. Assume further that $F : \mathbf{Sets} \to \mathbf{Sets}$ has a lifting $\overline{F} : \mathcal{K}\ell(T) \to \mathcal{K}\ell(T)$ in the sense of the diagram (2.11).

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

$$\begin{tikzcd} \overline{F}X \\ X \arrow[u, "c"'] \\ 1 \arrow[u, "s"'] \end{tikzcd} \qquad \text{in the Kleisli category } \mathcal{K}\ell(T) \enspace ,$$

which is the same thing as a pair of functions

$$\left(\begin{array}{c} TX \\ \uparrow s \\ 1 \end{array} , \quad \begin{array}{c} TFX \\ \uparrow c \\ X \end{array}\right) \qquad \text{in } \mathbf{Sets} \enspace ,$$

recalling that $\overline{F}X = FX$ for a lifting $\overline{F}$. The arrow s is called the *start state 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 is denoted by $*$.[1]

3.2.2 Example (LTSs with explicit termination)**.** Let us take $T = \mathcal{P}$ and $F = 1 + \Sigma \times \_$, in which case an $\overline{F}$-coalgebra in $\mathcal{K}\ell(T)$ is a function

$$X \xrightarrow{c} \mathcal{P}(1 + \Sigma \times X) \qquad \text{in } \mathbf{Sets}$$

modeling an LTS with explicit termination such as the leftmost one in (2.1). The monad $T = \mathcal{P}$ models non-deterministic branching; the endofunctor $F = 1 + \Sigma \times \_$ sets the transition type of "terminate, or (output and proceed)."

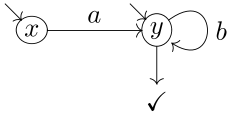
In this setting, a (T, F)-system—i.e. $(\mathcal{P}, 1+\Sigma\times\_)$-system—is an LTS with explicit termination

$$X \xrightarrow{c} \mathcal{P}(1 + \Sigma \times X) \qquad \text{in } \mathbf{Sets}$$

plus an explicit start state map

$$\begin{array}{cccl} 1 & \xrightarrow{s} & X & \text{in } \mathcal{K}\ell(\mathcal{P}) \\ \hline\hline 1 & \xrightarrow{s} & \mathcal{P}X & \text{in } \mathbf{Sets} \\ \hline\hline & s \subseteq X & . & \end{array}$$

When the leftmost system in (2.1) is accompanied by a start state map $s(*) = \{x, y\}$, it means that both states x and y can potentially become an initial state of an execution.



[1] Throughout Chapters 2 and 3 we have three different singletons with different meanings: $\{\checkmark\}$ (in $F = 1 + \Sigma \times \_$) for successful termination; $\{\perp\}$ (in $T = \mathcal{L} = 1 + \_$) for deadlock; and $\{*\}$ to denote start states.

3.2.3 Example (Probabilistic LTSs with explicit termination)**.** Let us replace $T = \mathcal{P}$ in the previous example by $T = \mathcal{D}$, the subdistribution monad, in which case we have probabilistic branching. A (T, F)-system is an $\overline{F}$-coalgebra

$$X \xrightarrow{c} \mathcal{D}(1 + \Sigma \times X) \qquad \text{in } \mathbf{Sets}$$

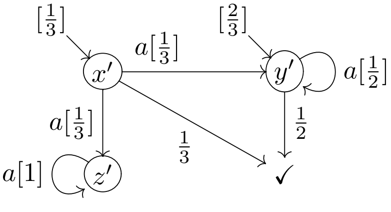
such as the probabilistic system in the middle of (2.1), augmented with a start state map

$$\frac{\dfrac{1 \xrightarrow{s} X \quad \text{in } \mathcal{K}\ell(\mathcal{D})}{1 \xrightarrow{s} \mathcal{D}X \quad \text{in } \mathbf{Sets}}}{s \in \mathcal{D}X} \; .$$

An example of such s is

$$s(*) = \begin{bmatrix} x' & \mapsto & 1/3 \\ y' & \mapsto & 2/3 \end{bmatrix} \;,$$

which we can depict as follows.



3.2.4 Remark. In category theory, an arrow in a category from a terminal object to an object X is often called a *point* in X. It is so because in **Sets** (where a singleton 1 is terminal) we have the following correspondence.

$$\frac{1 \xrightarrow{x} X \quad \text{in } \mathbf{Sets}}{\text{an element} \quad x \in X}$$

In the current setting, a start state map $1 \xrightarrow{s} X$ is *not* a point in this sense, since its domain 1 is not a terminal object in $\mathcal{K}\ell(T)$.[2]

Nevertheless the object $1 \in \mathcal{K}\ell(T)$ has a special status as a *unit* of a symmetric monoidal category $\mathcal{K}\ell(T)$. Specifically, when the monad T in question is commutative (Section 2.2), its Kleisli category $\mathcal{K}\ell(T)$ has a canonical commutative tensor (see [60, Theorem 4.3] or an abstract treatment later in Section 5.5.1). This tensor acts on objects as products hence has 1 as its neutral object. Consequences of this status of 1 in $\mathcal{K}\ell(T)$ are yet to be investigated.

[2] The empty set 0 is terminal in a setting of our interest, that is, when $\mathcal{K}\ell(T)$ is **Cppo**-enriched (Lemma 2.3.5).

3.2.2 Trace semantics for (T, F)-systems

In Chapter 2 we introduced the scheme of (finite) trace semantics for $\overline{F}$-coalgebras in a Kleisli category $\mathcal{K}\ell(T)$. It extends naturally to (T, F)-systems.

Let us make a brief review. Given a coalgebra $c : X \to \overline{F}X$ in $\mathcal{K}\ell(T)$, where the functor $\overline{F}$ is a lifting of F : **Sets** → **Sets**, we obtain its (finite) trace semantics via coinduction.

$$\begin{tikzcd}
\overline{F}X \arrow[r, dashed, "\overline{F}(\mathsf{tr}(c))"] & \overline{F}A & \text{in } \mathcal{K}\ell(T) \\
X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{tr}(c)"'] & A \arrow[u, "\cong"', "J\alpha^{-1}"]
\end{tikzcd}$$

Specifically, a coalgebra c is assigned the unique morphism $\mathsf{tr}(c)$ going into a final $\overline{F}$-coalgebra $J\alpha^{-1}$ (which is given by an initial F-algebra in **Sets**). The resulting arrow is, as a function, of the type

$$\mathsf{tr}(c) \ : \ X \longrightarrow TA \qquad \text{in } \mathbf{Sets}.$$

This (set-theoretic) function carries each state $x \in X$ to a "set of" (or "probability subdistribution on," etc. depending on the choice of T) linear-time behaviors which are elements of A.

Now that we have an explicit start state map $s : 1 \to X$ (in $\mathcal{K}\ell(T)$), the trace semantics of a (T, F)-system $1 \xrightarrow{s} X \xrightarrow{c} \overline{F}X$ is naturally defined as the composition of $\mathsf{tr}(c) : X \to A$ after $s : 1 \to X$, taken in $\mathcal{K}\ell(T)$.

3.2.5 Definition (Finite trace semantics of (T, F)-systems)**.** Let (F, T, λ) be a trace situation (Theorem 2.3.3). Given a (T, F)-system $1 \xrightarrow{s} X \xrightarrow{c} \overline{F}X$ in $\mathcal{K}\ell(T)$, its *(finite) trace semantics* $\mathsf{tr}(s, c)$ is the following composite.

$$\begin{tikzcd}
\underline{\text{In } \mathcal{K}\ell(T)} & \overline{F}X \arrow[r, dashed, "\overline{F}(\mathsf{tr}(c))"] & \overline{F}A \\
& X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{tr}(c)"] & A \arrow[u, "\cong"', "J\alpha^{-1}"] \\
& 1 \arrow[u, "s"] \arrow[ur, bend right, "\mathsf{tr}(s,c)"] &
\end{tikzcd}$$

Here $FA \xrightarrow{\alpha} A$ is an initial algebra in **Sets**; J : **Sets** → $\mathcal{K}\ell(T)$ is the standard Kleisli left adjoint.

This definition indeed gives what we expect from "trace semantics taking start states into account," as the following examples illustrate.

3.2.6 Example. Let us take $T = \mathcal{P}$ modeling non-deterministic branching. As we saw in Example 3.2.2, a start state map $s : 1 \to X$ (in $\mathcal{K}\ell(\mathcal{P})$) is identified as a subset $s(*)$ of X; it is seen as the set of possible initial states.

In this case, a natural interpretation of "trace semantics of the system $X \xrightarrow{c} \overline{F}X$ starting from s" is the union

$$\bigcup_{x\in s(*)} \mathsf{tr}(c)(x)$$

of the possible linear-time behaviors $\mathsf{tr}(c)(x)$, taken over all the possible initial states $x \in s(*)$. The composite $\mathsf{tr}(s,c) = \mathsf{tr}(c) \circ s$ in Definition 3.2.5 indeed yields this union:

$$\begin{array}{lcl} 1 \xrightarrow{s} X \xrightarrow{\mathsf{tr}(c)} A \quad \text{in } \mathcal{K}\ell(\mathcal{P}) & = & 1 \xrightarrow{s} \mathcal{P}X \xrightarrow{\mathcal{P}(\mathsf{tr}(c))} \mathcal{P}\mathcal{P}A \xrightarrow{\mu^{\mathcal{P}}_A} \mathcal{P}A \quad \text{in } \mathbf{Sets} \\ & & \text{which carries} \quad * \in 1 \quad \text{to} \quad \displaystyle\bigcup_{x\in s(*)} \mathsf{tr}(c)(x) \ . \end{array}$$

Recall that the multiplication $\mu^{\mathcal{P}}$ for the powerset monad is given by unions.

3.2.7 Example. Let us take $T = \mathcal{D}$, the subdistribution monad, to model probabilistic branching. A start state map $s : 1 \to X$ (in $\mathcal{K}\ell(\mathcal{D})$) is identified with a probability subdistribution $s(*) \in \mathcal{D}X$ such as in Example 3.2.3.

It is easy to see that the trace semantics of a (T,F)-system

$$\begin{array}{rcccl} \mathsf{tr}(s,c): & 1 & \longrightarrow & A & \text{in } \mathcal{K}\ell(\mathcal{D}) \\ \hline\hline \mathsf{tr}(s,c): & 1 & \longrightarrow & \mathcal{D}A & \text{in } \mathbf{Sets} \end{array}$$

carries $* \in 1$ to the probability subdistribution

$$\left[u \in A \quad \longmapsto \quad \sum_{x\in X} s(x) \cdot \mathsf{tr}(c)(x)(u) \right] ,$$

that is,

$$\left[u \in A \quad \longmapsto \quad \sum_{x\in X} \mathsf{Pr}(\text{init. state is } x) \cdot \mathsf{Pr}\begin{pmatrix}\text{exec. of } c\text{, starting from } x\text{, yields } u \\ \text{as its linear-time behavior}\end{pmatrix} \right] .$$

This probability assigned to a behavior $u \in A$ can be justifiably called the probability with which "the system (s,c) yields u as its linear-time behavior."

3.2.3 Forward and backward simulations

Now let us present the key observation in this chapter, namely the identification of

$$\boxed{\text{lax/oplax morphisms of coalgebras} \quad \text{as} \quad \text{forward/backward simulations.}}$$

Throughout the rest of this chapter, we assume that (F,T,λ) is a trace situation, that is, it satisfies the conditions in Theorem 2.3.3 on trace semantics.

3.2.8 Definition (Forward simulations)**.** Let $1 \xrightarrow{s} X \xrightarrow{c} \overline{F}X$ and $1 \xrightarrow{t} Y \xrightarrow{d} \overline{F}Y$ 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:

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

that is,

$$\begin{tikzcd}
\underline{\text{In } \mathcal{K}\ell(T)} & \overline{F}X \arrow[r, "\overline{F}f"] & \overline{F}Y \\
& X \arrow[u, "c"] \arrow[r, "f"] \arrow[ur, phantom, "\sqsupseteq"] & Y \arrow[u, "d"'] \\
& 1 \arrow[u, "s"] \arrow[r, equal] \arrow[ur, phantom, "\sqsupseteq"] & 1 \arrow[u, "t"']
\end{tikzcd} \ , \tag{3.2}$$

where $\sqsubseteq$ refers to the $\mathbf{DCpo}_\perp$-enriched structure of the Kleisli category $\mathcal{K}\ell(T)$ which we assume to be present; see Section 2.2.3.

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

Therefore, to be precise, a forward simulation is a lax morphism of coalgebras

$$\begin{tikzcd}
\overline{F}X \arrow[r, "\overline{F}f"] & \overline{F}Y \\
X \arrow[u, "c"] \arrow[r, "f"'] \arrow[ur, phantom, "\sqsupseteq"] & Y \arrow[u, "d"']
\end{tikzcd}$$

which satisfies an additional inequality regarding start state maps (the lower square in (3.2)).

A "dual" notion of forward simulations, with the inequalities reversed (but not the directions of arrows), is that of backward simulations.

3.2.9 Definition (Backward simulation)**.** Let $1 \xrightarrow{s} X \xrightarrow{c} \overline{F}X$ and $1 \xrightarrow{t} Y \xrightarrow{d} \overline{F}Y$ 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 $\overline{F}f \circ c \sqsubseteq d \circ f$. That is,

$$\begin{tikzcd}
\underline{\text{In } \mathcal{K}\ell(T)} & \overline{F}X \arrow[r, "\overline{F}f"] & \overline{F}Y \\
& X \arrow[u, "c"] \arrow[r, "f"] \arrow[ur, phantom, "\sqsubseteq"] & Y \arrow[u, "d"'] \\
& 1 \arrow[u, "s"] \arrow[r, equal] \arrow[ur, phantom, "\sqsubseteq"] & 1 \arrow[u, "t"']
\end{tikzcd} \ . \tag{3.3}$$

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

In the context of data refinement, lax natural transformations are used to formulate *simulations* between interpretations of a programming language [56, 76]—note that those “simulations” are not the ones (between state-based systems) that we are currently investigating. In a coalgebraic setting, [32] uses lax morphisms to investigate order-enriched version of bisimulations. However, to the best of our knowledge, the significance of lax/oplax morphisms in Kleisli categories is first emphasized here.

When an arrow $f : X \to Y$ makes the diagram (3.2) or (3.3) commute—i.e. equalities hold in place of inequalities—the arrow f will be called a *morphism of systems.*

3.2.10 Definition. Let $1 \xrightarrow{s} X \xrightarrow{c} \overline{F}X$ and $1 \xrightarrow{t} Y \xrightarrow{d} \overline{F}Y$ be (T, F)-systems, presented in $\mathcal{K}\ell(T)$. An arrow $f : X \to Y$ is a *morphism of* (T, F)*-systems* from (s, c) to (t, d), if the following diagram commutes.

$$\underline{\text{In } \mathcal{K}\ell(T)} \qquad \begin{tikzcd} \overline{F}X \arrow[r, "\overline{F}f"] & \overline{F}Y \\ X \arrow[u, "c"] \arrow[r, "f"] & Y \arrow[u, "d"'] \\ 1 \arrow[u, "s"] \arrow[r, equal] & 1 \arrow[u, "t"'] \end{tikzcd} \quad . \qquad (3.4)$$

3.2.11 Remark. Note the directions of arrows and inequalities in the definition of forward/backward simulations. Specifically, in (3.2) the system (t, d) is simulated by (s, c) via f. In (3.3)—where the “simulation” arrow f has the same direction from X to Y but the inequalities are reversed—the system (s, c) is simulated by (t, d) via f, with the roles of the two systems exchanged.

A general rule here is that the system appearing on the smaller side of the inequalities is the system which is simulated by the other system. The coming examples will clarify this intuition.

Let us justify our coalgebraic definition of forward/backward simulations. We shall do so by showing that the coalgebraic definition instantiates to the conventional definition of (non-deterministic) simulations, when the parameters T and F are suitably chosen.

3.2.12 Example (LTSs with explicit termination)**.** In the setting of Example 3.2.2, namely $T = \mathcal{P}$ and $F = 1 + \Sigma \times \_$, an arrow $X \to Y$ in $\mathcal{K}\ell(T)$ is identified with a binary relation R from X to Y.

$$\begin{array}{c} X \quad \longrightarrow \quad Y \qquad \text{in } \mathcal{K}\ell(\mathcal{P}) \\ \hline\hline X \quad \longrightarrow \quad \mathcal{P}Y \quad \text{in } \mathbf{Sets} \\ \hline\hline R \subseteq X \times Y, \ \text{binary relation} \end{array}$$

In this case, Definition 3.2.8 is instantiated as follows. A relation $R \subseteq X \times Y$ is a forward simulation from (t, d) to (s, c), if and only if it satisfies the following conditions.

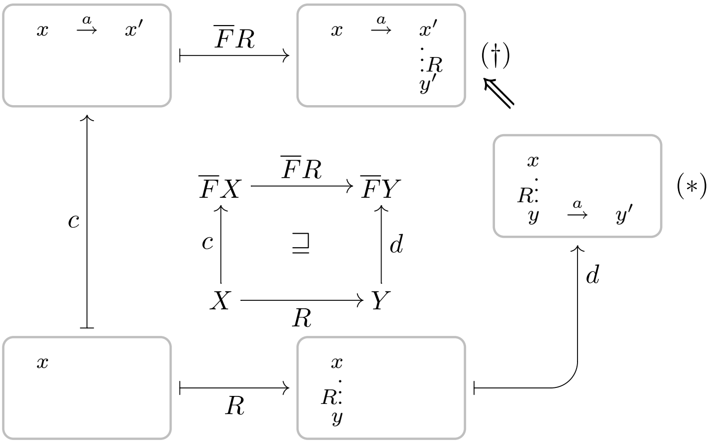
$$\begin{array}{rcl} y \in \mathsf{start}_{(t,d)} & \Longrightarrow & \exists x \in \mathsf{start}_{(s,c)}.\quad xRy\ , \\ xRy \ \wedge\ y \rightarrow_d \checkmark & \Longrightarrow & x \rightarrow_c \checkmark\ , \\ xRy \ \wedge\ y \xrightarrow{a}_d y' & \Longrightarrow & \exists x' \in X.\quad \big(\,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

Here $\mathsf{start}_{(s,c)}$ denotes the set $s(*)$ of the possible initial states of the system (s, c); and $x \xrightarrow{a}_c x'$ denotes the fact that $(a, x') \in c(x)$, i.e. $x \in X$ can make an a-move to x' in the system c. Hence the first line is concerned with initial states; the second line with termination; and the last line with "output and continue" type transitions. Among them we consider the third one as a core condition for simulations, and it is exactly the same as the one in the standard literature [94]. So is the first one on initial states.[3]

The third line draws special attention because it is the conventional definition of forward simulations between LTSs (without explicit start states or explicit termination). It works indeed "forwards," as illustrated below.

$$\begin{tikzcd} x \arrow[d, dotted, no head, "R"'] & & & x \arrow[r, "a"] \arrow[d, dotted, no head, "R"'] & \exists x' \arrow[d, dotted, no head, "R"] & \text{in } X \xrightarrow{c} \overline{F}X \\ y \arrow[r, "a"] & y' & \Longrightarrow & y \arrow[r, "a"] & y' & \text{in } Y \xrightarrow{d} \overline{F}Y \end{tikzcd} \tag{3.5}$$

Let us further illustrate how this well-known condition arises from the diagram (3.2).



(3.6)

First imagine that we take the path $d \circ R$ from X, i.e. first to the right then upwards. By going along R we find a state $y \in Y$ such that xRy; then by going along d, the state y makes a transition in d namely $y \xrightarrow{a} y'$. This yields the situation $(*)$ above.

[3] The three conditions here differ from the definition in [94] in the following respects: termination and internal actions. More specifically, the second line here is missing in [94] since explicit termination is not considered there. The condition we have here is a natural one since, intuitively, xRy means "x can simulate what y can do." We do not consider internal τ-actions (while [94] does); coalgebraic treatment of internal actions is known to be hard and we leave it as future work.

Look at the other path $\overline{F}R \circ c$ (upwards, then to the right). First the state $x \in X$ makes a transition $x \xrightarrow{a} x'$ in c; then by going along $\overline{F}R$ (the "lifting" of the relation R) the successor x' is related to some $y' \in Y$ via R. This results in the situation (†).

Finally, the inequality in the diagram postulates $d \circ R \sqsubseteq \overline{F}R \circ c$, that is, *in case we have the situation* $(*)$ *we need to have* (†) *as well.* Hence the implication $\Longrightarrow$ in the above diagram (3.6); it requires us to be able to find a state x' in the situation (†). This is precisely the conventional definition (3.5).

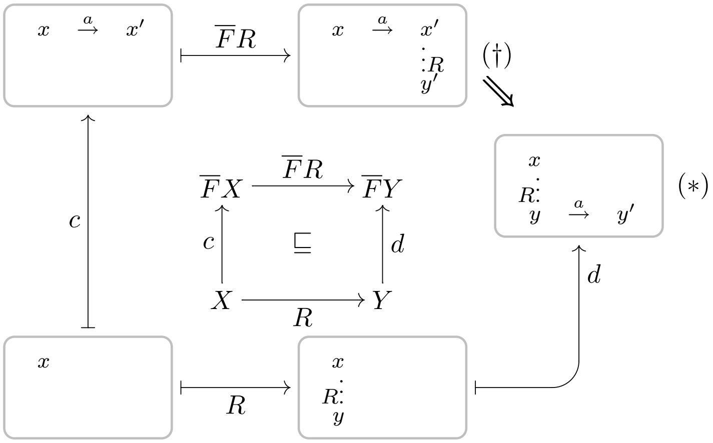
Similarly, a relation R from X to Y is a backward simulation (Definition 3.2.9) from (s, c) to (t, d) if and only if:

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

The third condition here works "backwards":

$$\begin{tikzcd} x \arrow[r, "a"] & x' \arrow[d, dash, dotted, "R"] & & x \arrow[r, "a"] \arrow[d, dash, dotted, "R"'] & x' \arrow[d, dash, dotted, "R"] & \text{in } X \xrightarrow{c} \overline{F}X \\ & y' & \Longrightarrow & \exists y \arrow[r, "a"] & y' & \text{in } Y \xrightarrow{d} \overline{F}Y \end{tikzcd} \tag{3.7}$$

which arises from the diagram (3.3) in the following way.



The only differences from the diagram (3.6) is that the inequality is reversed and so is the implication $\Longrightarrow$. The reversed implication (†) $\Longrightarrow$ $(*)$ yields (3.7).

3.2.13 Example (Probabilistic LTSs with explicit termination)**.** Let us see what arises from the generic definition (Definitions 3.2.8 and 3.2.9), if the systems have probabilistic branching (by taking $T = \mathcal{D}$). We consider the same transition-type $F = 1 + \Sigma \times \_$. Recall from Example 3.2.3 that (T, F)-systems, say (s, c) and (t, d), are thought of as probabilistic LTSs with explicit termination ✓ and an explicit start state distribution.

A forward simulation from (t, d) to (s, c) is a function $f : X \to \mathcal{D}Y$ in **Sets** which satisfies the following conditions.

$$\begin{aligned} t(*)(y) &\leq \sum_{x \in X} s(*)(x) \cdot f(x)(y) \\ \sum_{y \in Y} f(x)(y) \cdot d(y)(\checkmark) &\leq c(x)(\checkmark) \\ \sum_{y \in Y} f(x)(y) \cdot d(y)(a, y') &\leq \sum_{x' \in X} c(x)(a, x') \cdot f(x')(y') \end{aligned} \tag{3.8}$$

The third line can be rephrased as follows. It can be a bit more intuitive.

$$\sum_{y \in Y} \mathsf{Pr}(x \rightsquigarrow y) \cdot \mathsf{Pr}(y \xrightarrow{a} y') \leq \sum_{x' \in X} \mathsf{Pr}(x \xrightarrow{a} x') \cdot \mathsf{Pr}(x' \rightsquigarrow y') \qquad \left(\text{cf.} \quad \begin{tikzcd} x \arrow[r, "a"] \arrow[d, rightsquigarrow] & x' \arrow[d, rightsquigarrow] \\ y \arrow[r, "a"] & y' \end{tikzcd} \right)$$

where a function $f : X \to \mathcal{D}Y$ is thought of as a "probabilistic transition" from X to Y and the value $f(x)(y)$ is denoted by $\mathsf{Pr}(x \rightsquigarrow y)$. The generic definition of backward simulations can be instantiated in a similar manner.

One may wonder why we can call such a function $f : X \to \mathcal{D}Y$ a "forward simulation." In particular, our "probabilistic simulations" are no longer binary relations between state spaces. We have two justifications for the notion.

- The authenticity of our "probabilistic simulations" is supported by the way it arises: it is an instantiation of the generic, coalgebraic simulations which yields conventional simulations in a non-deterministic setting.
- Our view on forward/backward simulations in this thesis is that they are *tools* for establishing trace inclusion, hence for showing a system's satisfaction of some trace-based property. Such a scenario has been demonstrated in Section 1.4.

  Therefore we do not care much if our "simulation" is a binary relation or not. Our emphasis is instead on a soundness property which ensures usefulness of our "simulation." Indeed, we will later prove general soundness results (Theorem 3.3.2) which instantiates to the soundness property for our "probabilistic simulations." Moreover we will use them in verifying probabilistic trace-based (anonymity) properties later in Chapter 4.

Later in Section 3.4.3 our notion of "probabilistic simulations" (as an instantiation of the coalgebraic simulations) is compared with existing notions of (bi)simulations for probabilistic systems.

Although forward and backward simulations will be shown to be sound (Theorem 3.3.2), they fail to be complete in general.

$$\begin{tikzcd}
\boxed{\exists\,(\text{forward or backward simulation from } \mathcal{X} \text{ to } \mathcal{Y})} \arrow[d, Rightarrow, shift right=6ex, "\text{soundness}"'] \\
\boxed{(\text{trace sem. for } \mathcal{X}) \sqsubseteq (\text{trace sem. for } \mathcal{Y})} \arrow[u, Rightarrow, shift right=6ex, "\times" description, "\text{completeness}"']
\end{tikzcd}$$

A completeness result will be proved for a certain combination of forward and backward simulations. In [94] various combinations of forward and backward simulations are called *hybrid* simulations and their completeness properties are proved. Here we shall focus on the following kind of hybrid simulations.

3.2.14 Definition (Backward-forward simulations)**.** Let $1 \xrightarrow{s} X \xrightarrow{c} \overline{F}X$ and $1 \xrightarrow{t} Y \xrightarrow{d} \overline{F}Y$ be (T, F)-systems. A *backward-forward simulation* from (s, c) to (t, d) consists of

- an *intermediate system* $1 \xrightarrow{r} U \xrightarrow{b} \overline{F}U$;
- a backward simulation f from (s, c) to (r, b), and
- a forward simulation g from (r, b) to (t, d).

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

$$\begin{tikzcd}
\overline{F}X \arrow[r, "\overline{F}f"] & \overline{F}U & \overline{F}Y \arrow[l, "\overline{F}g"'] \\
X \arrow[u, "c"] \arrow[r, "f"] \arrow[ur, phantom, "\sqsubseteq"] & U \arrow[u, "b"] \arrow[ur, phantom, "\sqsubseteq"] & Y \arrow[u, "d"'] \arrow[l, "g"'] \\
1 \arrow[u, "s"] \arrow[r, equal] \arrow[ur, phantom, "\sqsubseteq"] & 1 \arrow[u, "r"] \arrow[r, equal] \arrow[ur, phantom, "\sqsubseteq"] & 1 \arrow[u, "t"']
\end{tikzcd} \quad . \tag{3.9}$$

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

$$(s, c) \sqsubseteq_{\mathbf{BF}} (t, d) \qquad \Longleftrightarrow \qquad \exists (r, b).\ \big(\,(s, c) \sqsubseteq_{\mathbf{B}} (r, b)\ \wedge\ (r, b) \sqsubseteq_{\mathbf{F}} (t, d)\,\big)\ . \tag{3.10}$$

3.2.4 Why explicit start states?

We shall explain the reason why we need an explicit start state map $1 \xrightarrow{s} X$ incorporated in the notion of (T, F)-systems. It has much to do with the general phenomenon that

a morphism of coalgebras preserves a behavior (given by coinduction), an idea we informally introduced in Section 1.3.1.

Specifically, assume that an endofunctor $F : \mathbb{C} \to \mathbb{C}$ has a final coalgebra $\zeta : Z \xrightarrow{\cong} FZ$. If there is a morphism of coalgebras

$$\begin{tikzcd} FX \arrow[r, "Ff"] & FY \\ X \arrow[u, "c"] \arrow[r, "f"'] & Y \arrow[u, "d"'] \end{tikzcd}$$

then f preserves a behavior given by coinduction. That is,

$$\mathsf{beh}(d) \circ f = \mathsf{beh}(c) \ . \tag{3.11}$$

This fact follows immediately from coinduction (proof principle). Soundness results (Theorem 3.3.2) are in the same spirit: it is about relating (possibly relaxed versions of) morphisms with behaviors by coinduction.

The resulting "behavior preservation" (3.11) has an immediate interpretation in the category **Sets**. It means: for each state $x \in X$ we have

$$\mathsf{beh}(d)\bigl(f(x)\bigr) = \mathsf{beh}(c)(x) \ ,$$

that is, the state $f(x) \in Y$ has the same behavior (modulo bisimilarity) as $x \in X$.

However, when we work in a Kleisli category $\mathcal{K}\ell(\mathcal{P})$, the equality (3.11) is no longer intuitive. For illustration let us take $T = \mathcal{P}$, the powerset monad. In (3.11) the arrows $\mathsf{beh}(c)$ and $\mathsf{beh}(d)$ give trace (set) semantics; the arrow f has, as a function, the type $f : X \to \mathcal{P}Y$. Equality means: for each $x \in X$,

$$\bigcup_{y \in f(x)} \mathsf{beh}(d)(y) = \mathsf{beh}(c)(x) \ . \tag{3.12}$$

That is, the state x has more behavior than any state $y \in f(x)$, but at the same time any behavior of x is simulated by some $y \in f(x)$. It is not easy to see significance of this equality.

The clumsiness here lies in comparing "one x vs. many y's." Introducing explicit start states is how we mend it: the following observation is an adaptation of the previous "behavior preservation" result, so that it takes start states into account.

3.2.15 Proposition. *Let $1 \xrightarrow{s} X \xrightarrow{c} \overline{F}X$ and $1 \xrightarrow{t} Y \xrightarrow{d} \overline{F}Y$ be (T, F)-systems, presented in $\mathcal{K}\ell(T)$.*

If $f : X \to Y$ is a morphism of (T, F)-systems from (s, c) to (t, d) (Definition 3.2.10), then trace semantics for the two systems is the same:

$$\mathsf{tr}(s, c) = \mathsf{tr}(t, d) \ . \tag{3.13}$$

Proof. We have the following situation.

$$\begin{tikzcd}
\overline{F}X \arrow[r, "\overline{F}f"] & \overline{F}Y \arrow[r, dashed, "\overline{F}(\mathsf{tr}(d))"] & \overline{F}A \\
X \arrow[u, "c"] \arrow[r, "f"] & Y \arrow[u, "d"] \arrow[r, dashed, "\mathsf{tr}(d)"] & A \arrow[u, "\cong", "J\alpha^{-1}"'] \\
1 \arrow[u, "s"] \arrow[r, equal] & 1 \arrow[u, "t"] \arrow[ur, bend right, "\mathsf{tr}(t,d)"'] &
\end{tikzcd}$$

The claim is shown by an easy calculation.

$$\mathsf{tr}(t,d) = \mathsf{tr}(d) \circ t = \mathsf{tr}(d) \circ f \circ s \overset{(*)}{=} \mathsf{tr}(c) \circ s = \mathsf{tr}(s,c) \enspace ,$$

where the equality $(*)$ is due to coinduction. □

The meaning of the equality (3.13) is intuitive and clear: when $T = \mathcal{P}$ it means

$$\bigcup_{x \in s(*)} \mathsf{beh}(c)(x) = \bigcup_{y \in t(*)} \mathsf{beh}(d)(y) \enspace .$$

The difference from (3.12) is that we are now comparing "many x's vs. many y's," with the help of explicit start states.

3.3 Soundness and completeness theorems

In this section we present the main results in this chapter, namely soundness and completeness theorems which relate forward/backward simulations (as lax/oplax morphisms of coalgebras) and trace semantics (as given by coinduction). The soundness results possibly reduce the task of proving trace inclusion to finding a simulation—the former involves looking ahead arbitrary many steps, while the latter involves only one-step transitions. Hence soundness plays a central role in simulation-based verification methods for trace-based properties, some examples of which will be presented later in Chapter 4.

3.3.1 Order-theoretic universality of a final coalgebra in $\mathcal{K}\ell(T)$

We have already observed that a morphism of (T, F)-systems preserves trace semantics (Proposition 3.2.15); its proof was essentially by coinduction. In the soundness results we will relate trace semantics (by coinduction) with forward/backward simulations—which are lax/oplax morphisms and hence involve the order structure of $\mathcal{K}\ell(T)$. Hence in the proofs (by coinduction) we will need an *order-theoretic* universal property of a final coalgebra in $\mathcal{K}\ell(T)$ besides its finality in a usual sense; it is given by the following result.

3.3.1 Proposition (Trace semantics as the largest lax morphism)**.** *Assume that (F, T, λ) is a trace situation (Theorem 2.3.3). An arbitrary $\overline{F}$-coalgebra $c : X \to \overline{F}X$ gives rise to its trace semantics $\mathsf{tr}(c) : X \to A$ given by the following coinduction.*

$$\begin{tikzcd}
\underline{\text{In } \mathcal{K}\ell(T)} & \overline{F}X \arrow[r, dashed, "\overline{F}(\mathsf{tr}(c))"] & \overline{F}A \\
& X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{tr}(c)"'] & A \arrow[u, "\cong"', "J\alpha^{-1}"]
\end{tikzcd}$$

Note that an equality $=$ is implicit in the middle of the diagram. Hence the arrow $\mathsf{tr}(c)$ is both a lax morphism and an oplax morphism from c to $J\alpha^{-1}$.

- *The trace arrow $\mathsf{tr}(c)$ is the largest lax morphism from c to $J\alpha^{-1}$. That is, for any $f : X \to A$ in $\mathcal{K}\ell(T)$,*

$$\begin{tikzcd}
FX \arrow[r, "\overline{F}f"] \arrow[dr, phantom, "\sqsupseteq"] & FA \\
X \arrow[u, "c"] \arrow[r, "f"'] & A \arrow[u, "\cong"', "J\alpha^{-1}"]
\end{tikzcd}
\quad \textit{implies} \quad
\begin{tikzcd}
X \arrow[r, bend left, "f"] \arrow[r, bend right, "\mathsf{tr}(c)"'] \arrow[r, phantom, "\rotatebox{90}{$\sqsubseteq$}"] & A
\end{tikzcd} .$$

- *Dually, the trace semantics $\mathsf{tr}(c)$ is the smallest oplax morphisms from c to $J\alpha^{-1}$. That is, for any $f : X \to A$ in $\mathcal{K}\ell(T)$,*

$$\begin{tikzcd}
FX \arrow[r, "\overline{F}f"] \arrow[dr, phantom, "\sqsubseteq"] & FA \\
X \arrow[u, "c"] \arrow[r, "f"'] & A \arrow[u, "\cong"', "J\alpha^{-1}"]
\end{tikzcd}
\quad \textit{implies} \quad
\begin{tikzcd}
X \arrow[r, bend left, "\mathsf{tr}(c)"] \arrow[r, bend right, "f"'] \arrow[r, phantom, "\rotatebox{90}{$\sqsubseteq$}"] & A
\end{tikzcd} .$$

Proof. Although the claim follows from a general result [32, Proposition 6.7], in this specific setting of a Kleisli category we can give another proof. Our proof depends only on the local monotonicity of $\overline{F}$ and not on its local continuity. It avoids the use of local continuity in a similar way to the proof of Theorem 2.3.3, namely by looking into the final sequence for the functor $\overline{F}$.

Let us consider the final sequence $1 \xleftarrow{!} \overline{F}1 \xleftarrow{\overline{F}!} \cdots$ for $\overline{F}$ in $\mathcal{K}\ell(T)$.[4] The following facts are standard about the final sequence; they are in the proof of Proposition A.1.2.

- A final $\overline{F}$-coalgebra (which by the way coincides with an F-initial algebra in **Sets**, Theorem 2.3.3) is an ω-limit of this final sequence. We denote this limit by $(\zeta_n : A \to F^n 1)_{n<\omega}$.
- An $\overline{F}$-coalgebra $c : X \to \overline{F}X$ yields a cone $(\gamma_n : X \to F^n 1)_{n<\omega}$ over the final sequence, in an inductive manner.

[4] For simplicity of presentation, we employ different notations from those in the proof of Theorem 2.3.3. For example, a final object 1 in $\mathcal{K}\ell(T)$ is given by the empty set, so it was denoted by 0 before.

– The unique coalgebra morphism $\mathsf{tr}(c)$ from c to the final coalgebra coincides with the unique mediating arrow $X \to A$ from the cone (γ_n) to the limit (ζ_n).

Hence we have the following situation in $\mathcal{K}\ell(T)$.

$$\begin{tikzcd}
{} & {} & {} & {} & A \text{ (limit)} \arrow[dlll, bend right, "\zeta_n"'] \arrow[dll, "\zeta_{n+1}"'] \\
\cdots & F^n 1 \arrow[l] & F^{n+1} 1 \arrow[l, "F^n!"'] & \cdots \arrow[l] & {} \\
{} & {} & {} & {} & X \arrow[uu, dashed, "\mathsf{tr}(c)"'] \arrow[ulll, bend left, "\gamma_n"] \arrow[ull, "\gamma_{n+1}"]
\end{tikzcd}$$

In the proof of Theorem 2.3.3, it was crucial that the limit (ζ_n) can be also characterized by the order-theoretical notion of **O**-limit (Definition A.2.3). In particular we can take the corresponding embedding $\zeta_n^E : F^n 1 \to A$ of each $\zeta_n : A \to F^n 1$; moreover we have $\mathrm{id}_A = \bigsqcup_{n<\omega}(\zeta_n^E \circ \zeta_n)$.

Now we prove the first statement of the proposition.

$$\begin{aligned}
\mathsf{tr}(c) &= \bigl(\textstyle\bigsqcup_{n<\omega} \zeta_n^E \circ \zeta_n\bigr) \circ \mathsf{tr}(c) && \text{by } \mathrm{id}_A = \textstyle\bigsqcup_{n<\omega}(\zeta_n^E \circ \zeta_n) \\
&= \bigl(\textstyle\bigsqcup_{n<\omega} \zeta_n^E \circ \zeta_n \circ \mathsf{tr}(c)\bigr) && \text{composition is continuous} \\
&= \bigl(\textstyle\bigsqcup_{n<\omega} \zeta_n^E \circ \gamma_n\bigr) && \mathsf{tr}(c) \text{ is a mediating arrow} \\
&\sqsupseteq \bigl(\textstyle\bigsqcup_{n<\omega} \zeta_n^E \circ \zeta_n \circ f\bigr) && \zeta_n \circ f \sqsubseteq \gamma_n \text{ for each } n,\ (\dagger) \\
&= \bigl(\textstyle\bigsqcup_{n<\omega} \zeta_n^E \circ \zeta_n\bigr) \circ f \quad = f \ , && \text{composition is continuous}
\end{aligned}$$

where the inequality (†) is proved by induction on n, using the assumption that f is a lax morphism. In this proof, local monotonicity of $\overline{F}$ has been implicitly used in showing that the limit (ζ_n) is also an **O**-limit.

The dual statement is proved in a similar way. □

3.3.2 Soundness of forward/backward simulations

3.3.2 Theorem (Soundness of simulations)**.** *Let* $1 \xrightarrow{s} X \xrightarrow{c} \overline{F}X$ *and* $1 \xrightarrow{t} Y \xrightarrow{d} \overline{F}Y$ *be* (T, F)*-systems. Existence of a forward/backward/backward-forward simulation from* (s, c) *to* (t, d) *implies trace inclusion* $\mathsf{tr}(s, c) \sqsubseteq \mathsf{tr}(t, d)$*. That is,*

$$\begin{array}{rrcl}
\textit{1.} & (s,c) \sqsubseteq_{\mathbf{F}} (t,d) & \Longrightarrow & \mathsf{tr}(s,c) \sqsubseteq \mathsf{tr}(t,d) \ , \\
\textit{2.} & (s,c) \sqsubseteq_{\mathbf{B}} (t,d) & \Longrightarrow & \mathsf{tr}(s,c) \sqsubseteq \mathsf{tr}(t,d) \ , \\
\textit{3.} & (s,c) \sqsubseteq_{\mathbf{BF}} (t,d) & \Longrightarrow & \mathsf{tr}(s,c) \sqsubseteq \mathsf{tr}(t,d) \ .
\end{array}$$

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

$$\begin{tikzcd}
\overline{F}Y \arrow[r, "\overline{F}f"] & \overline{F}X \arrow[r, dashed, "\overline{F}(\mathsf{tr}(c))"] & \overline{F}A \\
Y \arrow[u, "d"] \arrow[r, "f"'] \arrow[ur, phantom, "\sqsupseteq"] & X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{tr}(c)"'] \arrow[ur, phantom, "="] & A \arrow[u, "\cong"', "J\alpha^{-1}\text{ (final)}"]
\end{tikzcd}$$

The square on the right is a coinduction diagram. This shows that the arrow $\mathsf{tr}(c) \circ f$ is a lax coalgebra morphism from d to the final coalgebra, that is,

$$J\alpha^{-1} \circ \big(\mathsf{tr}(c) \circ f\big) \sqsubseteq \overline{F}\big(\mathsf{tr}(c) \circ f\big) \circ d \qquad \text{in } \mathcal{K}\ell(T).$$

Since the trace map is the biggest lax coalgebra morphism (Proposition 3.3.1), we have $\mathsf{tr}(c) \circ f \sqsubseteq \mathsf{tr}(d)$. This inequality is combined with f's condition on start state maps.

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

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 $1 \rightrightarrows A$. □

3.3.3 Completeness of hybrid simulations

Completeness (the converse of soundness) does not hold for $\sqsubseteq_{\mathbf{F}}, \sqsubseteq_{\mathbf{B}}$ but does hold for the more strict notion of $\sqsubseteq_{\mathbf{BF}}$. For a restricted class of non-deterministic systems the completeness result is shown in [77, 94].

3.3.3 Theorem (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. Given a (T,F)-system (s,c), we construct the following system which we refer to as a *canonical system.*

$$1 \xrightarrow{\mathsf{tr}(s,c)} A \xrightarrow[\cong]{J\alpha^{-1}} \overline{F}A \qquad \text{in } \mathcal{K}\ell(T)$$

Its dynamics is a final $\overline{F}$-coalgebra; its start states map is given by the trace semantics of the system (s,c) in consideration.

By definition of trace semantics, the arrow $\mathsf{tr}(c) : X \to A$ is a morphism of systems from (s,c) to this canonical system. This establishes the left side of the diagram (3.14) below. We apply the same construction to (t,d) and obtain the right side of the diagram. Now the assumption $\mathsf{tr}(s,c) \sqsubseteq \mathsf{tr}(t,d)$ fits in the lower middle of the diagram.

$$\begin{tikzcd}
FX \arrow[r, dashed, "\overline{F}(\mathsf{tr}(c))"] & FA & FY \arrow[l, dashed, "\overline{F}(\mathsf{tr}(d))"'] \\
X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{tr}(c)"] & A \arrow[u, "\cong", "J\alpha^{-1}"'] & Y \arrow[u, "d"'] \arrow[l, dashed, "\mathsf{tr}(d)"'] \\
1 \arrow[u, "s"] \arrow[r, equal] & 1 \arrow[u, bend left, "\mathsf{tr}(s,c)"] \arrow[u, bend right, "\mathsf{tr}(t,d)"'] \arrow[u, phantom, "\sqsubseteq"] \arrow[r, equal] & 1 \arrow[u, "t"']
\end{tikzcd} \tag{3.14}$$

From this we obtain the following two diagrams of backward-forward simulations—like the diagram (3.9) in Definition 3.2.14—depending on our choice of the intermediate system.

$$\begin{tikzcd}
FX \arrow[r, dashed] & FA & FY \arrow[l, dashed] \\
X \arrow[u, "c"] \arrow[r, dashed] & A \arrow[u, "\cong"] & Y \arrow[u, "d"'] \arrow[l, dashed] \\
1 \arrow[u, "s"] \arrow[r, equal] & 1 \arrow[u, "\mathsf{tr}(s,c)"] \arrow[r, equal] \arrow[ur, phantom, "\sqsubseteq"] & 1 \arrow[u, "t"']
\end{tikzcd}
\qquad
\begin{tikzcd}
FX \arrow[r, dashed] & FA & FY \arrow[l, dashed] \\
X \arrow[u, "c"] \arrow[r, dashed] & A \arrow[u, "\cong"] & Y \arrow[u, "d"'] \arrow[l, dashed] \\
1 \arrow[u, "s"] \arrow[r, equal] \arrow[ur, phantom, "\sqsubseteq"] & 1 \arrow[u, "\mathsf{tr}(t,d)"'] \arrow[r, equal] & 1 \arrow[u, "t"']
\end{tikzcd}$$

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

3.3.4 Similarity relations are preorders

We shall show that three similarity relations $\sqsubseteq_{\mathbf{F}}, \sqsubseteq_{\mathbf{B}}$ and $\sqsubseteq_{\mathbf{BF}}$—corresponding to forward, backward, and backward-forward simulations, respectively—are indeed preorders. Being a preorder is a basic property that we would expect from such a relation. The proofs have been postponed until now, because in showing transitivity of $\sqsubseteq_{\mathbf{BF}}$ we use completeness (Theorem 3.3.3).

For $\sqsubseteq_{\mathbf{F}}$ and $\sqsubseteq_{\mathbf{B}}$ the proof is straightforward.

3.3.4 Proposition ($\sqsubseteq_{\mathbf{F}}, \sqsubseteq_{\mathbf{B}}$ are preorders)**.** *The forward and backward similarity relations $\sqsubseteq_{\mathbf{F}}$ and $\sqsubseteq_{\mathbf{B}}$ are preorders. That is, they are reflexive and transitive.*

Proof. Reflexivity is obvious: take the identity arrow in the Kleisli category as a forward (or backward) simulation. To show transitivity, assume $(s,c) \sqsubseteq_{\mathbf{F}} (t,d) \sqsubseteq_{\mathbf{F}} (r,b)$. There exist forward simulations f and g such that

$$\begin{tikzcd}
FU \arrow[r, "\overline{F}f"] & FY \arrow[r, "\overline{F}g"] & FX \\
U \arrow[u, "b"] \arrow[r, "f" description] \arrow[ur, phantom, "\sqsupseteq"] & Y \arrow[u, "d"] \arrow[r, "g" description] \arrow[ur, phantom, "\sqsupseteq"] & X \arrow[u, "c"'] \\
1 \arrow[u, "r"] \arrow[r, equal] \arrow[ur, phantom, "\sqsupseteq"] & 1 \arrow[u, "t"] \arrow[r, equal] \arrow[ur, phantom, "\sqsupseteq"] & 1 \arrow[u, "s"']
\end{tikzcd}
, \quad \text{hence} \quad
\begin{tikzcd}
FU \arrow[r, "\overline{F}(g \circ f)"] & FX \\
U \arrow[u, "b"] \arrow[r, "g \circ f" description] \arrow[ur, phantom, "\sqsupseteq"] & X \arrow[u, "c"'] \\
1 \arrow[u, "r"] \arrow[r, equal] \arrow[ur, phantom, "\sqsupseteq"] & 1 \arrow[u, "s"']
\end{tikzcd} .$$

This shows $(s,c) \sqsubseteq_{\mathbf{F}} (r,b)$. Transitivity of $\sqsubseteq_{\mathbf{B}}$ is proved in a similar way. □

3.3.5 Proposition ($\sqsubseteq_{\mathbf{BF}}$ is a preorder)**.** *The backward-forward similarity relation $\sqsubseteq_{\mathbf{BF}}$ is a preorder.*

Proof. Reflexivity is trivial by taking the system itself as an intermediate one. Assume $(s,c) \sqsubseteq_{\mathbf{BF}} (t,d) \sqsubseteq_{\mathbf{BF}} (r,b)$. By soundness of $\sqsubseteq_{\mathbf{BF}}$ (Theorem 3.3.2) we have $\mathsf{tr}(s,c) \sqsubseteq \mathsf{tr}(t,d) \sqsubseteq \mathsf{tr}(r,b)$; transitivity of $\sqsubseteq$ implies $\mathsf{tr}(s,c) \sqsubseteq \mathsf{tr}(r,b)$. This in turn yields $(s,c) \sqsubseteq_{\mathbf{BF}} (r,b)$ by completeness of $\sqsubseteq_{\mathbf{BF}}$ (Theorem 3.3.3). Hence we have shown transitivity of $\sqsubseteq_{\mathbf{BF}}$. □

3.4 Related work

3.4.1 Simulations in coalgebra

Hughes and Jacobs [58, 64] also study simulations in a coalgebraic setting. Their coalgebraic modeling of simulations—and their motivation which leads to the modeling—is quite different from ours. They introduce their notion of simulations (which, by the way, work "forwards") as a relaxed version of bisimulations. Based on this definition they proceed to observe how these simulations are related to coalgebraic bisimulations, hence to final coalgebras in **Sets**. This marks a contrast between their investigation and ours; in the current work simulations are related to trace semantics (not to bisimulations), hence to final coalgebras in $\mathcal{K}\ell(T)$.

More specifically, the definition of coalgebraic bisimulations which is used in [58, 64]—and which is relaxed to yield coalgebraic simulations—is the following one, using *relation liftings*. (We have used relation liftings in Lemma 2.2.4.) Given coalgebras $X \xrightarrow{c} FX$ and $Y \xrightarrow{d} FY$ in **Sets**, a *bisimulation* between them is a binary relation $R \subseteq X \times Y$ such that

$$(x, y) \in R \quad\Longrightarrow\quad \bigl(c(x), d(y)\bigr) \in \mathrm{Rel}_F(R) \ . \tag{3.15}$$

The relation $\mathrm{Rel}_F(R) \subseteq FX \times FY$ is the F-relation lifting of R. For illustration, let us take $F = \mathcal{P}_{\mathrm{fin}}(\Sigma \times \_)$ for which coalgebras are (finitely branching) LTSs. For this F, the relation lifting $\mathrm{Rel}_F(R)$ is the following relation.

$$\begin{array}{rcl}
\mathrm{Rel}_F(R) & \subseteq & \mathcal{P}_{\mathrm{fin}}(\Sigma \times X) \times \mathcal{P}_{\mathrm{fin}}(\Sigma \times Y) \ , \\
(u, v) \in \mathrm{Rel}_F(R) & \Longleftrightarrow & \left[\begin{array}{lcl} (a, x') \in u & \Longrightarrow & \exists y' \in Y.\ \bigl(\ (a, y') \in v \ \wedge\ (x', y') \in R\ \bigr) \\ (a, y') \in v & \Longrightarrow & \exists x' \in X.\ \bigl(\ (a, x') \in u \ \wedge\ (x', y') \in R\ \bigr) \end{array}\right]
\end{array}$$

In this case it is obvious that a coalgebraic bisimulation (3.15) is the same thing as a conventional bisimulation between LTSs. More generally, *bisimilarity* defined in terms of these coalgebraic bisimulations (using relation liftings) coincides with the bisimilarity via coinduction (which we discussed in Section 1.3.2), if the functor F preserves weak pullbacks.

The core assumption made in [58, 64] is that the functor F comes with a preorder structure. Specifically, the functor $F : \mathbf{Sets} \to \mathbf{Sets}$ factors through the forgetful functor $\mathbf{PreOrd} \to \mathbf{Sets}$ from the categories of preordered sets and monotone functions.

$$\begin{tikzcd}
 & \mathbf{PreOrd} \arrow[d] \\
\mathbf{Sets} \arrow[ur, "\sqsubseteq"] \arrow[r, "F"'] & \mathbf{Sets}
\end{tikzcd} \tag{3.16}$$

In particular, each set of the form FX comes with a preorder $\sqsubseteq$.

This preorder structure $\sqsubseteq$ on FX and FY is used to relax (3.15) and obtain coalgebraic simulations in [58, 64]. Namely, a *simulation* from $X \xrightarrow{c} FX$ to $Y \xrightarrow{d} FY$

is a binary relation $R \subseteq X \times Y$ such that

$$(x,y) \in R \quad \Longrightarrow \quad \exists u \in FX.\, \exists v \in FY. \begin{bmatrix} & c(x) \sqsubseteq_{FX} u \\ \wedge & (u,v) \in \mathrm{Rel}_F(R) \\ \wedge & v \sqsubseteq_{FY} d(y) \end{bmatrix} , \tag{3.17}$$

$$\text{that is,} \qquad \begin{tikzcd} c(x) \arrow[r, dotted, no head, "\sqsubseteq_{FX}"] & \exists u \arrow[r, dotted, no head, "\mathrm{Rel}_F(R)"] & \exists v \arrow[r, dotted, no head, "\sqsubseteq_{FY}"] & d(y) \end{tikzcd} .$$

Here $\sqsubseteq_{FX}$ refers to the preorder on FX which results from the assumption (3.16). The condition (3.17) for simulations is weaker than (3.15) for *bi*simulations because, in (3.17), $c(x)$ and $d(y)$ can use "helper" preorders $\sqsubseteq_{FX}$ and $\sqsubseteq_{FY}$ in order to be related via $\mathrm{Rel}_F(R)$.

In [58, 64] it is demonstrated that the coalgebraic simulation (3.17) gives a conventional simulation in many examples. They proceed to investigate the relationship between coalgebraic simulations and bisimulations, such as the following.

- When does bisimilarity coincide with two-way similarity? The former implies the latter but not the other way round in general.
- Similarity order makes a final coalgebra an ω-cpo. Moreover, under certain conditions it becomes an algebraic cpo.

All these observations in [58, 64] take place in **Sets**.

In contrast, we take a different view on forward/backward simulations in our current setting: simulations are tools to show trace inclusion. Hence our emphasis is on the relationship to trace semantics, in particular soundness properties. We base our observations on a Kleisli category.

Let us compare concrete definitions of simulations obtained from the framework in [58, 64] and definitions obtained from ours. For non-deterministic systems, both frameworks yield quite similar "simulations"; moreover they are similar to conventional ones such as in [94]. Indeed if we take $F = \mathcal{P}(1 + \Sigma \times \_)$ (equipped with the inclusion order) in the framework of [58, 64], their notion of simulations coincides with our forward simulations for LTSs with explicit termination (described in Example 3.2.12). Notice that in the former framework there is no distinction between branching types and transition types; hence both $\mathcal{P}$ and $1 + \Sigma \times \_$ are put into a single functor F.

However a difference emerges in a probabilistic setting. It is possible to apply the framework of [58, 64] to a functor F which contains probabilistic branching; however, still in this case, the induced simulation is a binary relation between state spaces (see (3.17)). In contrast our simulation is a function of type $X \to \mathcal{D}Y$. The latter finds its use in showing trace inclusion (see Chapter 4); comparative merits of the former notion of simulations is yet to be investigated.

3.4.2 Lax categorical structures and data refinement

Categorical notions of *lax* and *oplax* morphisms of coalgebras play central roles in our framework. As we have briefly mentioned, lax/oplax categorical constructs are also used in formalizing data refinement [56, 76]. In this context of data refinement, people use terminologies such as *upward simulations*, *downward simulations*, and so on. These "simulations" are not the same things as the simulations between state-based systems (investigated in this chapter); nevertheless we shall briefly sketch "simulations" in data refinement.

In short, an *upward simulation* in data refinement is a "relationship" between two interpretations of the same programming language. It is categorically formalized as a lax natural transformation ρ in the following situation [56].

$$\begin{tikzcd} \mathbf{Cl} \arrow[r, bend left, "F"{name=U}] \arrow[r, bend right, "G"'{name=D}] & \mathbf{Dom} \arrow[from=U, to=D, Rightarrow, "\rho"] \end{tikzcd} \tag{3.18}$$

Let us explain the situation. First, one identifies the programming language of our interest with its classifying, syntactic category $\mathbf{Cl}$.

- An object in $\mathbf{Cl}$ is a type, such as stack;
- an arrow in $\mathbf{Cl}$ is a piece of code which denotes a function with corresponding types, such as $\mathsf{pop} : \mathsf{stack} \to \mathsf{stack}$, or $(\mathsf{pop}; \mathsf{pop}) : \mathsf{stack} \to \mathsf{stack}$.

These syntactic objects are interpreted in a locally-ordered category $\mathbf{Dom}$ of "semantic domains" such as $\mathbf{Dom} = \mathbf{Cppo}$. An interpretation is given categorically as a functor $F : \mathbf{Cl} \to \mathbf{Dom}$; it carries objects (i.e. types) and arrows (i.e. functions) in $\mathbf{Cl}$ to their interpretations in $\mathbf{Dom}$, respectively.

The basic idea of data refinement is as follows. Given two interpretations F and G of the programming language $\mathbf{Cl}$ in $\mathbf{Dom}$, sometimes one would like to say 'F is a refined interpretation of G' or 'F gives a representation of a "specification" G.' Such a statement would mean that there is, for each type $X \in \mathbf{Cl}$, an arrow

$$\rho_X \ : \ FX \longrightarrow GX \qquad \text{in } \mathbf{Dom}.$$

Moreover this "translation" should be compatible with interpretations of functions: for each arrow $X \xrightarrow{f} Y$ in $\mathbf{Cl}$,

$$\begin{tikzcd} FX \arrow[r, "\rho_X"] \arrow[d, "Ff"'] & GX \arrow[d, "Gf"] \\ FY \arrow[r, "\rho_Y"'] & GY \end{tikzcd} \quad . \tag{3.19}$$

This makes $\rho : F \Rightarrow G$ a natural transformation.

However, in a "specification" G, some function f may be yet to be interpreted. Categorically it means that the arrow $Gf : GX \to GY$ in $\mathbf{Dom}$ might be the bottom $\perp$ (which is available e.g. when $\mathbf{Dom}$ is $\mathbf{Cppo}$). Nevertheless the refined interpretation F may have the function f already interpreted; in this case the diagram (3.19) does not commute but rather one obtains an inequality in the square.

$$\begin{tikzcd}
FX \arrow[r, "\rho_X"] \arrow[d, "Ff"'] \arrow[dr, phantom, "\sqsupseteq"] & GX \arrow[d, "Gf"] \\
FY \arrow[r, "\rho_Y"'] & GY
\end{tikzcd} .$$

Hence an upward simulation ρ from F to G—which witnesses that F is a refined interpretation of G—is a *lax* natural transformation as in the situation (3.18). Here the order structure of **Dom** is considered to be "degree of definedness" [76].

3.4.3 (Bi)simulations for probabilistic systems

We have obtained the notion of "probabilistic forward/backward simulations" by suitably instantiating our coalgebraic simulations (Example 3.2.13). What is notable about it is that a simulation is not a binary relation between state spaces, but instead a function of the type $X \to \mathcal{D}Y$; and that we have soundness theorems for free.

There have been definitions of "bisimulations" and "simulations" for probabilistic systems, proposed by several authors. Here we shall focus on two notable definitions among them—those introduced in [88, 120]—and compare these two with our definition.

In [120] the notion of (forward) simulations is introduced for (Segala's) *probabilistic automata* which are models of probabilistic systems introduced in [120]. It is proved to be sound with respect to trace inclusion in [120]. Moreover in [95] these simulations are shown to be also complete, with respect to *trace distribution precongruence* $\leq_{\mathrm{DC}}$— it is the coarsest preorder which is included in the trace inclusion preorder and is "compositional," i.e. compatible with parallel composition of probabilistic automata.

The biggest difference between the setting of [95, 120] and ours is that (Segala's) probabilistic automata have both non-deterministic and probabilistic branching. Coalgebraically, (Segala's) probabilistic automata are functions of the type

$$X \longrightarrow \mathcal{P}\mathcal{D}FX$$

where F is a suitable transition type. See [128] for detailed treatment of coalgebraic formulation of various models of probabilistic systems. Our coalgebraic theory of traces and simulations (in $\mathcal{K}\ell(T)$) has not been applied to this combined branching yet (see Section 2.6).

Still the following fact suggests possible applicability of our coalgebraic framework. A "simulation" in [120] is not a binary relation between states, but it is a relation between a state and a probability distribution over states, which can be identified with a function of the type $X \to \mathcal{P}\mathcal{D}Y$.

$$\frac{R \subseteq X \times \mathcal{D}Y, \quad \text{a relation}}{X \longrightarrow \mathcal{P}\mathcal{D}Y \quad \text{in } \mathbf{Sets}}$$

Hence this simulation might be an arrow $X \to Y$ in a suitable Kleisli category like $\mathcal{K}\ell(\mathcal{PD})$;[5] if so, it matches our framework where a simulation is an arrow $X \to Y$ in $\mathcal{K}\ell(T)$.

Another point to note is that the notion of "trace inclusion" in [95, 120] between (Segala's) probabilistic automata is quite different from ours. Specifically, trace semantics for a (Segala's) probabilistic automaton is given by a set of probabilistic distributions over the set of linear-time behavior, that is, a function

$$\mathsf{tr}(s,c) \ : \ 1 \longrightarrow \mathcal{PD}A$$

where A is the set of all linear-time behavior. Now the "trace inclusion" order $\mathsf{tr}(s,c) \sqsubseteq \mathsf{tr}(t,d)$ is defined by:

$$\mathsf{tr}(s,c) \sqsubseteq \mathsf{tr}(t,d) \quad \Longleftrightarrow \quad \forall d \in \mathsf{tr}(s,c).\ d \in \mathsf{tr}(t,d) \ .$$

Hence "trace inclusion" for (Segala's) probabilistic automata in [95, 120] refers to (pure) set inclusion; it does not say anything about the comparative quantity of probability with which a linear-time behavior would occur. In contrast, in our setting (which is purely probabilistic), trace inclusion—defined in terms of the **Cppo**-enriched structure of $\mathcal{K}\ell(\mathcal{D})$—is based on comparative quantity of probability.

The other definition that interests us is the notion of *probabilistic bisimulation* in [88]. A probabilistic bisimulation in [88] is defined between purely probabilistic systems (i.e. no non-deterministic branching) so the setting is similar to ours; still it is given by a binary relation, in contrast to ours which are functions of the type $X \to \mathcal{D}Y$. Roughly speaking, an equivalence relation $R \subseteq X \times X$ on a state space of a probabilistic LTS is said to be a *probabilistic bisimulation* if

$$xRy \quad \Longrightarrow \quad \forall a \in \Sigma.\ \forall S \in X/R. \quad \mathsf{Pr}(x \xrightarrow{a} S) = \mathsf{Pr}(y \xrightarrow{a} S) \ . \tag{3.20}$$

Here Σ is the set of actions and $\mathsf{Pr}(x \xrightarrow{a} S)$ is the probability with which x makes an a-move to one of the states in S. That is,

$$\mathsf{Pr}(x \xrightarrow{a} S) = \sum_{x' \in S} \mathsf{Pr}(x \xrightarrow{a} x') \ .$$

The problem is that this definition is not immediately transformable into a definition of simulations. Specifically, the relation R's being an equivalence is crucial in writing down the condition (3.20); however we expect a simulation to be a preorder rather than an equivalence. Additionally, the above definition of probabilistic bisimulations works only within the same system. It does not easily generalize to bisimulations between two different systems; neither to simulations from one system to another.

[5] It is known that the combination $\mathcal{PD}$ is *not* a monad [136]. To make it a monad a small modification is needed for either $\mathcal{P}$ or $\mathcal{D}$.

3.5 Summary and future work

Continuing from Chapter 2, we have developed a generic theory of traces and simulations 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 implications of this theory in practice.

As mentioned in Section 2.6, systems with both non-deterministic and probabilistic branching (such as Segala's probabilistic automata) do not fit in our general framework. Some observations regarding the potential applicability of our framework have been made in Section 3.4.3.

Later in Chapter 5 we will introduce a generic coalgebraic framework to describe parallel composition of systems and compositionality of process semantics. As mentioned in Section 2.6, the framework works for all the process semantics given by coinduction, hence including trace semantics (by coinduction in $\mathcal{K}\ell(T)$). The relationship between this composition framework and the material in this chapter (on simulations) is investigated as well; we will obtain compositionality results for similarity relations $\sqsubseteq_{\mathbf{F}}$, $\sqsubseteq_{\mathbf{B}}$ and $\sqsubseteq_{\mathbf{BF}}$ (Propositions 5.5.5 and 5.5.6).

Often a process equivalence $\sim$ (or a process preorder $\sqsubseteq$) is characterized by means of some modal logic: $\mathcal{C} \sim \mathcal{D}$ (or $\mathcal{C} \sqsubseteq \mathcal{D}$) if, for any formula φ in the logic,

$$\mathcal{C} \models \varphi \quad \Longleftrightarrow \quad \mathcal{D} \models \varphi \qquad (\text{or} \quad \mathcal{C} \models \varphi \quad \Longrightarrow \quad \mathcal{D} \models \varphi),$$

respectively. A classic result of this kind is expressivity of the Hennessy-Milner logic with respect to bisimilarity [52]; other than that a variety of process equivalences/preorders have been given such characterizations (see e.g. [39]). The canonical testing situation in Section 2.5 is also considered to be a "logical characterization" of coalgebraic trace equivalence (Proposition 2.5.9); logical characterizations of the coalgebraic similarity relations $\sqsubseteq_{\mathbf{F}}$ and $\sqsubseteq_{\mathbf{B}}$ in this chapter is yet to be investigated.

IOA Toolset [38] 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 contemplate on a generic version of the toolset as well.

Chapter 4

Case study: probabilistic anonymity

There is a growing concern about anonymity and privacy on the Internet, resulting in lots of work on formalization and verification of anonymity. Especially, importance of probabilistic aspects of anonymity is claimed recently by many authors. Several different notions of “probabilistic anonymity” have been studied so far, but proof methods for such probabilistic notions are not yet much elaborated. In this chapter we introduce a simulation-based proof method for one notion of probabilistic anonymity introduced by Bhargava and Palamidessi [15], called *strong probabilistic anonymity.* The method is a probabilistic adaptation of the one by Kawabe, Sakurada et al. for non-deterministic anonymity [71, 72]: anonymity of a protocol is proved by finding out a forward/backward simulation between certain automata. For the jump from non-determinism to probability we exploit the generic, coalgebraic theory of traces and simulations developed in Chapters 2 and 3. In particular, an appropriate notion of probabilistic simulations is obtained as an instantiation of the generic definition, for which a soundness theorem comes for free. Additionally, we show how we can use a similar idea to verify a weaker notion of probabilistic anonymity called *probable innocence.*

4.1 Overview

4.1.1 Online anonymity

Nowadays most human activities rely on communication over the Internet, hence on communication protocols. This has made verification of communication protocols a trend in computer science. At the same time, varying purposes of communication protocols have identified new verification goals—or *security properties*—such as anonymity, in addition to rather traditional ones like secrecy and authentication.

Anonymity properties have attracted growing concern from the public. There are emerging threats as well: for example, the European Parliament in December

2005 approved rules forcing ISPs to retain access records. Consequently an increasing extent of research activities—especially from the formal methods community—are aiming at verification of anonymity properties (see [6]).

Formal verification of anonymity properties is relatively young compared to authentication or secrecy. The topic still allows for definitional work (such as [15, 22, 37, 41, 57, 110]) pointing out many different aspects of anonymity notions. Notably many authors [15,22,41,123,124] claim the significant role of *probability* in anonymity notions. This is the focus of the current work.

4.1.2 Probabilistic anonymity

There have been several different notions of anonymity proposed in probabilistic settings, e.g. [15, 41, 112]. There are also some case studies which analyze existing anonymizing protocols and examine whether they satisfy the notions of probabilistic anonymity. However, generic verification methods for probabilistic anonymity have not yet much elaborated.

In this chapter we introduce simulation-based proof methods for two different notions of probabilistic anonymity. The first one is *strong probabilistic anonymity* (which will be also called *strong anonymity*) introduced by Bhargava and Palamidessi [15]. The definition requires that, by observing an execution of the protocol, the adversary should gain absolutely no information on who is the person to be blamed—the *culprit*. The notion is a natural probabilistic adaptation of the non-deterministic notion of *trace anonymity*.[1]

What we shall do in this chapter is to adapt a proof method as well, from the one for (non-deterministic) trace anonymity to the one for probabilistic strong anonymity. The proof method we start with is the simulation-based one which is introduced in [71, 72]. Its basic scenario is as follows.

1. First we model an anonymizing protocol as a certain kind of automaton $\mathcal{X}$.
2. Second we construct the *anonymized* version $\mathsf{an}(\mathcal{X})$ of $\mathcal{X}$. The automaton $\mathsf{an}(\mathcal{X})$ satisfies the appropriate notion of anonymity because of the way it is constructed.
3. We prove that

$$\text{(trace semantics of } \mathcal{X}) = \text{(trace semantics of } \mathsf{an}(\mathcal{X})) \enspace .$$

   Then, since the notion of anonymity is defined in terms of traces, anonymity of $\mathsf{an}(\mathcal{X})$ yields anonymity of $\mathcal{X}$. The equality is proved by showing that the (appropriate notion of) inclusion order $\sqsubseteq$ holds in both directions.

   - One direction $\sqsubseteq$ holds because of the construction of $\mathsf{an}(\mathcal{X})$.

[1] The notion of trace anonymity is originally introduced in [119] under the name of *strong anonymity*. In this chapter we follow [71, 72] and call it trace anonymity. This is for the sake of terminological convenience.

- The other direction $\sqsupseteq$ is proved by finding a (forward or backward) *simulation* from $\mathsf{an}(\mathcal{X})$ to $\mathcal{X}$. Here we appeal to the soundness theorem for simulations—existence of a simulation implies trace inclusion.

Hence the anonymity proof of $\mathcal{X}$ is reduced to finding a suitable forward/backward simulation.

4.1.3 Converting non-determinism to probability

The basic scenario remains the same for strong (probabilistic) anonymity. However, there is an obvious difficulty in conducting the scenario in a probabilistic setting. The theory of traces and simulations in a non-deterministic setting is well studied e.g. in [94]; however appropriate definitions of probabilistic traces and simulations are far from trivial.

This jump from non-determinism to probability is where we exploit the generic, coalgebraic theory of traces and simulations developed in the previous chapters. In the generic theory, fundamental notions such as systems (or automata), trace semantics and forward/backward simulations are identified as certain kinds of coalgebraic constructs. On this level of abstraction the general soundness theorem—existence of a (coalgebraic) simulation yields (coalgebraic) trace inclusion—is proved by categorical arguments.

The generic theory has two parameters T and F appearing in it; by making different choices of these parameters the theory can cover a wide variety of systems. In particular, according to the choice of one parameter T, systems can be non-deterministic or probabilistic. In this chapter we obtain a complex definition of probabilistic simulations as an instance of the general, coalgebraic definition. Moreover, this definition is a *useful* one: a soundness theorem comes for free from the general soundness theorem.

After a simulation-based proof method for strong (probabilistic) anonymity, we use a similar idea to verify a weaker anonymity notion called *probable innocence*. The notion appears in [41, 112] and is extensively studied in [23]. The significance of this weaker notion is due to the fact that many real-world anonymizing protocols—which include Crowds [112]—do not satisfy strong anonymity but do satisfy probable innocence. Intuitively, probable innocence does allow the adversary to learn some information about the culprit; but it requires that this information leak is only up to a certain bound.

We take the definition in [23]—which generalizes the ones in [41,112]—and present a simulation-based proof method for this notion. Although the basic scenario is not exactly the same as before, the idea of using simulations is quite similar.

4.1.4 Outline of the chapter

In Section 4.2 we illustrate notions of probabilistic anonymity using a couple of examples. They include the Dining Cryptographers protocol and the Crowds protocol. The formal definitions of anonymity notions are presented in 4.3, where we also introduce our models of anonymizing protocols called *anonymity automata*. In Section 4.4

we describe our simulation-based proof method for strong anonymity and prove its correctness. In Section 4.5 the proof method for probable innocence is described. Relation to other work with a similar interest is discussed in Section 4.6.

Notations The set of lists in an alphabet X with length ≥ 1 is denoted by X^+, that is, $X^+ = X^* \cdot X$ in a regular-expression-like notation. This appears later as a domain of trace semantics for anonymity automata.

4.2 Motivating examples

In this section we motivate

- the probabilistic aspect of anonymity, and
- possible candidates for a formal notion of "probabilistic anonymity,"

by presenting concrete examples of anonymizing protocols. The first example is the Dining Cryptographers (DC) protocol [25]: this common example in the formal study of anonymity protocols will illustrate the important role of probability in anonymizing protocols. The second example of simple and artificial protocols for anonymous donation will clarify the idea behind the notion of strong probabilistic anonymity. In the last place we present an example of Crowds [112]—which does not satisfy strong probabilistic anonymity—to motivate the weaker notion of probable innocence.

4.2.1 The Dining Cryptographers (DC) protocol

Here we follow [15] to illustrate the role of probability in anonymity, using the Dining Cryptographers (DC) protocol.

There are three cryptographers (or "users") dining together. The payment will be made either by one of the cryptographers, or by NSA (U.S. National Security Agency) which organizes the dinner. Who is paying has been determined by NSA; if one of the cryptographers is paying, the payer has been told so beforehand.

The goal of the DC protocol is as follows. The three cryptographers

- reveal whether there is a payer among them, but
- not revealing which of them is the payer (if any),

to the observer (called the *adversary* in the sequel) and also to the cryptographers who are not paying.

The protocol proceeds in the following way. Three cryptographers Crypt_i for $i = 0, 1, 2$ sit in a circle, each with a coin Coin_i. The coins are held in such a way that they can be seen by the owner and one of the other two: the arrows → in the following figure denote the "able-to-see-her-coin" relation.

$$\begin{tikzcd}
 & \mathsf{Crypt}_0 \arrow[dl] & \\
\mathsf{Crypt}_1 \arrow[rr] & & \mathsf{Crypt}_2 \arrow[ul]
\end{tikzcd}$$

Hence Crypt_0 can see the coin held by Crypt_1 as well as her own. Then the coins are flipped; each cryptographer, comparing the two coins she can see, announces to the public whether they *agree* (showing the same side) or *disagree*. The trick is that the one who is paying—if there is—must lie about the announcement. For example, given that Crypt_0 is paying, the configuration of coins

$$(\mathsf{h}, \mathsf{t}, \mathsf{h}) \qquad \text{that is} \qquad \begin{tikzcd} & \mathsf{h} \arrow[dl, bend right] & \\ \mathsf{t} \arrow[rr, bend right] & & \mathsf{h} \arrow[ul, bend right] \end{tikzcd} ,$$

results in the announcement

$$(\mathsf{a}, \mathsf{d}, \mathsf{a}) \qquad \text{that is} \qquad \begin{tikzcd} & \mathsf{a} \arrow[dl, bend right] & \\ \mathsf{d} \arrow[rr, bend right] & & \mathsf{a} \arrow[ul, bend right] \end{tikzcd} .$$

This announcement is the only thing the adversary can observe; occurrence of an odd number of d's reveals the presence of a liar, hence the presence of a payer among the cryptographers.

Can the adversary tell which cryptographer is paying? No. In fact, given any announcement with an odd number of d's, one can construct a coin configuration which yields the given announcement under an arbitrary choice of the payer. For example, the announcement $(\mathsf{a}, \mathsf{d}, \mathsf{a})$ above can be obtained from any of the following configurations.

Crypt_0 pays, and coins are $(\mathsf{h}, \mathsf{t}, \mathsf{h})$ or $(\mathsf{t}, \mathsf{h}, \mathsf{t})$
Crypt_1 pays, and coins are $(\mathsf{h}, \mathsf{h}, \mathsf{h})$ or $(\mathsf{t}, \mathsf{t}, \mathsf{t})$
Crypt_2 pays, and coins are $(\mathsf{h}, \mathsf{h}, \mathsf{t})$ or $(\mathsf{t}, \mathsf{t}, \mathsf{h})$

4.2.2 Probabilistic anonymity in DC

Up to now our arguments have been non-deterministic; now we shall explain how probabilistic aspects in DC can emerge. Assume that the coins are biased: each of three Coin_i's gives head with the probability 9/10. Provided that Crypt_0 is paying, the announcement $(\mathsf{a}, \mathsf{d}, \mathsf{a})$ occurs with the probability $(9 \cdot 1 \cdot 9 + 1 \cdot 9 \cdot 1)/10^3$, because it results from $(\mathsf{h}, \mathsf{t}, \mathsf{h})$ or $(\mathsf{t}, \mathsf{h}, \mathsf{t})$. Similar calculations lead to the following table of conditional probabilities.

	$(\mathsf{d}, \mathsf{a}, \mathsf{a})$	$(\mathsf{a}, \mathsf{d}, \mathsf{a})$	$(\mathsf{a}, \mathsf{a}, \mathsf{d})$	$(\mathsf{d}, \mathsf{d}, \mathsf{d})$
Crypt_0 pays	0.73	0.09	0.09	0.09
Crypt_1 pays	0.09	0.73	0.09	0.09
Crypt_2 pays	0.09	0.09	0.73	0.09

Are the cryptographers still "anonymous"? We would not say so. For example, if the adversary observes an announcement $(\mathsf{d}, \mathsf{a}, \mathsf{a})$, it is reasonable for her to suspect Crypt_0 more than the other two.

Nevertheless, if the coins are not biased we cannot find any symptom of broken anonymity. Therefore we want to come up with the following two things.

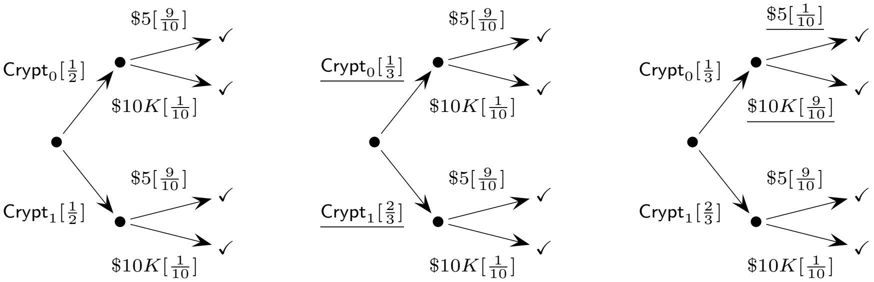
The first is an appropriate notion of "probabilistic anonymity" which holds with fair coins but is violated with biased coins. This is done in [15] and the definition will be explained using the next simpler example.

The second is an effective proof method to verify this notion of anonymity. This is what we aim at in the current work.

4.2.3 Strong probabilistic anonymity

In this section we present toy examples of probabilistic anonymizing protocols to motivate the notion of strong (probabilistic) anonymity [15].

Let us think of a situation of *anonymous donation*: one of two cryptographers Crypt_0 and Crypt_1 tries to donate some money, without revealing the donor's identity. Consider the following three protocols, $\mathcal{X}_0, \mathcal{X}_1, \mathcal{X}_2$ from left to right.



It is assumed that the labels Crypt_i—deciding which cryptographer is the donor—are invisible for the observer. Only the labels $\$n$—for money transactions—are visible.

The first protocol $\mathcal{X}_0$ works as follows. First the donor is chosen with the uniform probability distribution; no matter which is chosen, the donor donates \$5 with the probability 9/10 and \$10K with 1/10. In the second protocol $\mathcal{X}_1$, it is in the specification of the protocol (hence is known to the adversary) that Crypt_1 is more likely to be the donor—the culprit. In $\mathcal{X}_2$ (in comparison to $\mathcal{X}_1$), Crypt_0 is known, so to say, to be rich: when she is chosen as the donor more often than not she gives away \$10K.

Are these protocols "anonymous"? That is, by observing money transactions, can the adversary tell who is the culprit? Intuitively it is clear that $\mathcal{X}_0$ should be "anonymous" and $\mathcal{X}_2$ should not be. $\mathcal{X}_1$ can be debatable but we want to claim it to be anonymous. It is true that Crypt_1 is inherently more suspicious than Crypt_0 in $\mathcal{X}_1$ (the *a-priori* distribution of suspicion is not uniform). However, after observing any execution of the protocol, from the adversary's viewpoint the cryptographers look exactly as suspicious as before. In other words, in $\mathcal{X}_1$, *an observation of transaction of* \$5 *or* \$10K *does not carry any information on who is the culprit.*

This intuition leads to the notion of strong anonymity introduced in [15]. Here we give an informal description; a formal definition is found in Definition 4.3.8.

4.2.1 Definition (Strong anonymity). An anonymizing protocol $\mathcal{X}$ satisfies *strong (probabilistic) anonymity* if, for any observation o and users i and j, we have

$$P_{\mathcal{X}}(o \mid i \text{ is a culprit}) = P_{\mathcal{X}}(o \mid j \text{ is a culprit}) \ .$$

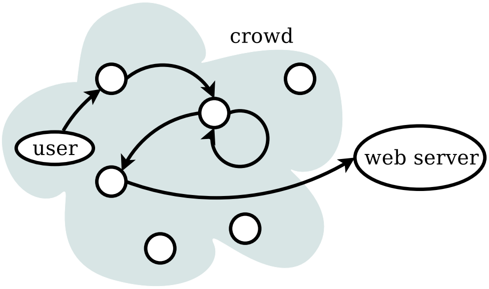
Here $P_{\mathcal{X}}(o \mid i \text{ is a culprit})$ denotes the conditional probability for the event "given that the user i is the culprit, an execution of $\mathcal{X}$ yields the observation o."

The intuition behind this definition is quite similar to the one behind the notion of *conditional anonymity* [41] whose formal definition is given later in Definition 4.3.9. In fact, it is shown in [15] that under reasonable assumptions these two notions of anonymity coincide.

4.2.4 Crowds and probable innocence

Although the DC protocol with fair coins (as well as the anonymous donation protocols $\mathcal{X}_0$ and X_1 in Section 4.2.3) satisfies strong anonymity, there are a large body of real-world protocols which do not satisfy the property. Crowds [112] is one of them.

The Crowds protocol aims at anonymous web browsing, so that a user can send a request to a web server without revealing the user's identity. In essence the identity is concealed by relaying the request among a set ("crowd") of relays.



- A user, who has already joined a group ("crowd") of hosts, picks one relay out of the crowd with a uniform probability and forwards a request to the chosen relay. The relay can be the initiating user herself.
- A relay in a crowd, on receiving a request, flips a (biased) coin to decide whether
  - to deliver the request to its intended recipient, i.e. the web server (this happens with the probability $1 - p_f$), or
  - to forward it to another relay (with the probability p_f). The next relay is chosen from the crowd with the uniform probability, including the current relay itself.

Here the probability p_f is a system-wide parameter. This way the web server only sees a request coming from a relay, hence it is not sure who initiated the request.

It is obvious that Crowds gives no anonymity guarantee against a *global eavesdropper*—an adversary capable of observing all the network traffic—because a global eavesdropper can see the initiator sending out a message without receiving any message beforehand. A more realistic attack scenario is that *some of the relays are corrupt* *and they cooperate with the adversary.* A short calculation shows the following. The fact that a certain host i has sent a request to a corrupt relay—although the adversary cannot tell for sure whether the host i is the initiator of the request or is just forwarding it—makes the host i probabilistically more suspicious than the other hosts. This makes the strong anonymity fail: the observation "i has sent a request to me" does carry some information on who is the initiator.[2]

Still, when the number of relays (i.e. the cardinality of the crowd) is sufficiently bigger than that of corrupt relays, we want to say that Crowds is "anonymous," that is, Crowds satisfies some weaker notion of anonymity that allows information leak up to a certain amount. The notion of *probable innocence* is one candidate of such an anonymity notion.

Intuitively, probable innocence is satisfied if *after any observation, the adversary can never have confidence exceeding* $1/2$ *that someone is the culprit.* In [112] it is shown that Crowds satisfies probable innocence if $p_f > 1/2$ and the number n of relays and the number c of corrupt relays satisfy the inequality $n \geq \frac{p_f}{p_f - 1/2}(c+1)$.[3]

However, the above statement in [112] is only true if every user in the crowd is equally suspicious a-priori, i.e. before an execution of Crowds. Assume there is a user who is known to be much more frequently an initiator of a request than other users; then the above "confidence is $\leq 1/2$" definition of probable innocence is more easily violated. The situation is similar to the comparison between $\mathcal{X}_0$ and $\mathcal{X}_1$ in Section 4.2.3; we want our definition robust to such a change of the a-priori distribution of suspicion. A user who is a-priori more suspicious than the others can still look more suspicious than the others after an execution of the protocol; a bit more than it did before an execution, because now we allow some marginal information leak; but not much more.

This idea led Chatzikokolakis and Palamidessi [23] to the following definition of probable innocence, which we formally present later in Definition 4.3.10. It is also shown to coincide with the original definitions in [41, 112] under suitable assumptions.

4.2.2 Definition (Probable innocence)**.** An anonymizing protocol $\mathcal{X}$ satisfies *probable innocence* if, for any observation o and user i we have

$$(n-1)\frac{P_{\mathcal{X}}(i \text{ is a culprit})}{\sum_{j\neq i} P_{\mathcal{X}}(j \text{ is a culprit})} \geq \frac{P_{\mathcal{X}}(i \text{ is a culprit} \mid o)}{\sum_{j\neq i} P_{\mathcal{X}}(j \text{ is a culprit} \mid o)} \ . \qquad (4.1)$$

Here n is the number of users who can possibly be a culprit; we are assuming that two different users cannot be culprits at the same time.

[2] Strong anonymity is satisfied if no relay in the crowd is corrupt. In this case, the adversary is a web server outside the crowd; when it receives a request from a host i, it can tell for sure that i is forwarding the request (recall that at least one forwarding is mandated). Hence the host i is no more suspicious than the other hosts.

[3] In fact, the definition of probable innocence used in [112] is slightly different from this one which is based on the definition in [41]. These two are shown in [23] to coincide under some mild conditions.

The left-hand side of (4.1) is about the a-priori distribution of suspicion. To get an idea let us assume that the a-priori distribution is uniform. Then (4.1) becomes

$$1 \geq \frac{P_{\mathcal{X}}(i \text{ is a culprit} \mid o)}{\sum_{j \neq i} P_{\mathcal{X}}(j \text{ is a culprit} \mid o)} \quad .$$

If we moreover assume that the observation o reveals *existence* of a culprit, that is,

$$\sum_{i} P_{\mathcal{X}}(i \text{ is a culprit} \mid o) = 1 \quad ,$$

the condition is equivalent to $1/2 \geq P_{\mathcal{X}}(i \text{ is a culprit} \mid o)$. In this way we can recover the original definition: "the adversary's confidence does not exceed 1/2."

4.3 Formalizing notions for probabilistic anonymity

4.3.1 Anonymity automata: models of anonymizing protocols

In this chapter anonymizing protocols are formalized as a specific kind of probabilistic systems which we shall call (probabilistic) *anonymity automata.* The notion is to some extent similar to probabilistic automata [121]. However, in anonymity automata, branching is purely probabilistic without any non-determinism. This modification, together with other minor ones, is made so that the coalgebraic framework of the previous chapters applies.

The features of an anonymity automaton are as follows.

- By making a transition it can either
  - execute an action and successfully terminate ($x \xrightarrow{a} \checkmark$), or
  - execute an action and move to another state ($x \xrightarrow{a} y$).

  For technical simplicity we leave out internal actions (denoted by τ).
- An action a can be either
  - an *observable action* o which can be seen by the adversary, or
  - an *actor action* $\mathsf{act}(i)$ which means the choice of a user i as the culprit.
- Each state comes with a probability subdistribution over the set of possible transitions. Recall that, by "sub"distribution it is meant that the sum of all the probabilities is ≤ 1 rather than $= 1$. The missing probability is understood as the probability for deadlock.

Here is a formal definition.

4.3.1 Definition (Anonymity automata)**.** An *anonymity automaton* is a 5-tuple $(X, \mathcal{U}, \mathcal{O}, c, s)$ where:

- X is a non-empty set called the *state space.*
- $\mathcal{U}$ is a non-empty set of *users.*[4]
- $\mathcal{O}$ is a non-empty set of *observable actions.*
- $c : X \to \mathcal{D}\big(\mathcal{A} \times \{\checkmark\} + \mathcal{A} \times X\big)$ is a function which assigns to each state $x \in X$ a probability subdistribution $c(x)$ over possible transitions. The set $\mathcal{A}$ is the set of *actions* and defined by

  $$\mathcal{A} = \mathcal{O} + \{\, \mathsf{act}(i) \mid i \in \mathcal{U} \,\} \enspace .$$

  The operation $\mathcal{D}$ is the subdistribution monad; its action on a set Y is given by

  $$\mathcal{D}Y = \big\{ d : Y \to [0,1] \mid \sum_{y \in Y} d(y) \leq 1 \big\} \enspace . \tag{4.2}$$

  For example, the value $c(x)(a, \checkmark)$[5] in $[0,1]$ is the probability with which a state x executes a and then successfully terminate (i.e. $x \xrightarrow{a} \checkmark$).
- $s \in \mathcal{D}X$ is a probability subdistribution over the state space X. This specifies which state would be a *starting* (or *initial*) one.

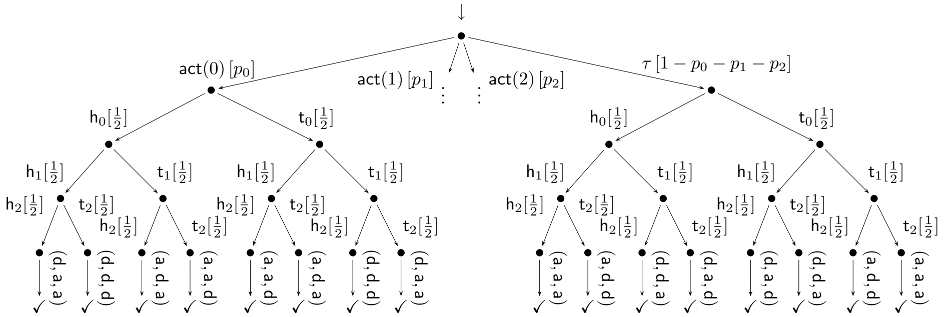
A major difference between (probabilistic) anonymity automata and probabilistic LTS with explicit termination (Example 3.2.3) is that the former executes an action even when it terminates ($x \xrightarrow{a} \checkmark$); while the latter does not ($x \to \checkmark$).

4.3.2 Example (Anonymity automaton $\mathcal{X}_{\mathrm{DC}}$ for DC)**.** To model the DC protocol, we take

$$\mathcal{U} = \{0, 1, 2\} \enspace , \qquad \mathcal{O} = \{\mathsf{a}, \mathsf{d}\} \times \{\mathsf{a}, \mathsf{d}\} \times \{\mathsf{a}, \mathsf{d}\} = \big\{ (\mathsf{x}, \mathsf{y}, \mathsf{z}) \mid \mathsf{x}, \mathsf{y}, \mathsf{z} \in \{\mathsf{a}, \mathsf{d}\} \big\} \enspace .$$

We need to fix the a-priori probability distribution on who will make a payment. Let us denote by p_i the probability with which the user i pays.

The following is a natural representation of the DC protocol (with its a-priori probability distribution given by p_i's). It makes explicit use of internal actions which we still have to get rid of. Probability for each transition is presented in square brackets; otherwise the transition occurs with probability 1.

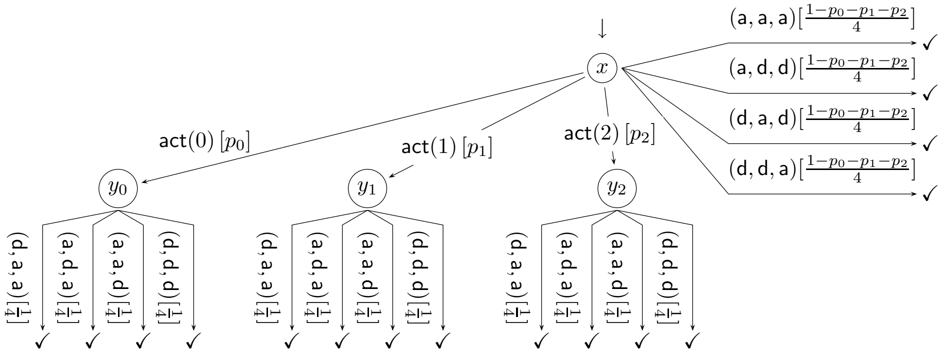


[4] A user is called an *anonymous user* in [15].

[5] To be precise this should be written as $c(x)\big(\kappa_1(a, \checkmark)\big)$, where $\kappa_1 : \mathcal{A} \times \{\checkmark\} \to \mathcal{A} \times \{\checkmark\} + \mathcal{A} \times X$ is the inclusion map.

Here τ denotes an internal action with the intention of “NSA pays.”

However, the coin-flip actions h_i and t_i—with their obvious meanings—must not be present because they are not observable by the adversary nor are actor actions. These actions are replaced by τ's. Moreover, for technical simplicity we do not allow τ's to appear in an anonymity automaton. Hence we take the “closure” of the above automaton in an obvious way, and obtain the following.


x
act(0) $[p_0]$
act(1) $[p_1]$
act(2) $[p_2]$
y_0
y_1
y_2
$(\mathsf{a},\mathsf{a},\mathsf{a})[\frac{1-p_0-p_1-p_2}{4}]$
$(\mathsf{a},\mathsf{d},\mathsf{d})[\frac{1-p_0-p_1-p_2}{4}]$
$(\mathsf{d},\mathsf{a},\mathsf{d})[\frac{1-p_0-p_1-p_2}{4}]$
$(\mathsf{d},\mathsf{d},\mathsf{a})[\frac{1-p_0-p_1-p_2}{4}]$
$(\mathsf{d},\mathsf{a},\mathsf{a})[\frac{1}{4}]$
$(\mathsf{a},\mathsf{d},\mathsf{a})[\frac{1}{4}]$
$(\mathsf{a},\mathsf{a},\mathsf{d})[\frac{1}{4}]$
$(\mathsf{d},\mathsf{d},\mathsf{d})[\frac{1}{4}]$
✓

The start state distribution s is: $x \mapsto 1$. This anonymity automaton we shall refer to as $\mathcal{X}_{\mathrm{DC}}$.

4.3.2 Anonymity automata reconciled as coalgebras

The generic, coalgebraic theory of traces and simulations (Chapters 2 and 3) applies to anonymity automata. Recall that the generic theory is developed with two parameters T and F:

- a monad T on **Sets** specifies the *branching type*, such as non-determinism or probability;
- a functor F on **Sets** specifies the *transition type*, i.e., what a system can do by making a transition.

Systems for which traces/simulations are defined are called (T, F)*-systems* in the generic theory, making the parameters explicit (Definition 3.2.1). The theory is coalgebraic because a (T, F)-system is essentially a coalgebra in a Kleisli category $\mathcal{K}\ell(T)$.

Anonymity automata fit in this generic framework. They are (T, F)-systems with the following choice of parameters T and F.

- T is the subdistribution monad $\mathcal{D}$, modeling purely probabilistic branching.
- $FX = \mathcal{A}\times\{\checkmark\}+\mathcal{A}\times X$, modeling the transition-type of “(action and terminate) or (action and next state).”

It is immediately seen that for this choice of F, the set $\mathcal{A}^+$ (of lists in $\mathcal{A}$ with length ≥ 1) carries the following initial algebra in **Sets**. We denote its structure map by α.

$$\begin{tikzcd}
\mathcal{A}\times\{\checkmark\}+\mathcal{A}\times\mathcal{A}^+ \arrow[d, "\alpha"', "\cong"] & \kappa_1(a,\checkmark) \arrow[d, mapsto] & \kappa_2(a,\overrightarrow{a}) \arrow[d, mapsto] \\
\mathcal{A}^+ & \langle a\rangle & a\cdot\overrightarrow{a}
\end{tikzcd} ,$$

where $\langle a\rangle$ denotes a list of length 1, and $a\cdot\overrightarrow{a}$ is what would be written as (`cons` a $\overrightarrow{a}$) in Lisp. Therefore Definition 2.4.1 suggests that the set $\mathcal{A}^+$ is the appropriate domain of (finite) trace semantics for anonymity automata; this is indeed the case later in Definition 4.3.3.

4.3.3 Trace semantics for anonymity automata

Trace semantics for anonymity automata is used in defining probabilistic notions of anonymity. In a non-deterministic setting, trace semantics yields a *set* of lists ("traces") of actions which can possibly occur during an execution. In contrast, trace semantics of a probabilistic system is given by a *probability subdistribution* over lists.

4.3.3 Definition (Trace semantics for anonymity automata)**.** Given an anonymity automaton $\mathcal{X}=(X,\mathcal{U},\mathcal{O},c,s)$, its *trace semantics*

$$P_{\mathcal{X}}\in\mathcal{D}(\mathcal{A}^+)$$

is defined as follows. For a list of actions $\langle a_0,a_1,\dots,a_n\rangle$ with a finite length $n\geq 1$,

$$P_{\mathcal{X}}(\langle a_0,a_1,\dots,a_n\rangle)=\sum_{x_0,x_1,\dots,x_n\in X} P_{\mathcal{X}}(x_0\overset{a_0}{\to}x_1\overset{a_1}{\to}\cdots\overset{a_{n-1}}{\to}x_n\overset{a_n}{\to}\checkmark) ,$$

where the probability

$$\begin{aligned}
&P_{\mathcal{X}}(x_0\overset{a_0}{\to}x_1\overset{a_1}{\to}\cdots\overset{a_{n-1}}{\to}x_n\overset{a_n}{\to}\checkmark)\\
&=s(x_0)\,\cdot\,c(x_0)(a_0,x_1)\,\cdot\,\cdots\,\cdot\,c(x_{n-1})(a_{n-1},x_n)\,\cdot\,c(x_n)(a_n,\checkmark)
\end{aligned}$$

is for the event that an execution of $\mathcal{X}$ starts at x_0, follows the path $\overset{a_0}{\to}x_1\overset{a_1}{\to}\cdots\overset{a_{n-1}}{\to}x_n$ and finally terminates with $\overset{a_n}{\to}\checkmark$.

Intuitively the value $P_{\mathcal{X}}(\overrightarrow{a})\in[0,1]$ for a list $\overrightarrow{a}\in\mathcal{A}^+$ is the probability with which the system $\mathcal{X}$ executes actions in $\overrightarrow{a}$ successively and then terminates. Our concern is about actions (observable actions or actor actions) the system makes but not about the (internal) states it exhibits.

The following alternative characterization allows us to apply the generic, coalgebraic theory of traces (Chapters 2 and 3).

4.3.4 Lemma (Trace semantics via the generic theory)**.** *Given an anonymity automaton $\mathcal{X}$, let (s,c) be a (T,F)-system identified with $\mathcal{X}$ as in Section 4.3.2.*

The trace semantics $P_{\mathcal{X}}$ of $\mathcal{X}$ coincides with the coalgebraic trace semantics $\mathsf{tr}(s,c)$ *defined for (s,c) (Definition 3.2.5).* □

4.3.5 Example (Dining cryptographers)**.** For the anonymity automaton $\mathcal{X}_{\mathrm{DC}}$ in Example 4.3.2, its trace semantics $P_{\mathcal{X}_{\mathrm{DC}}}$ is the following probability subdistribution.

$$\begin{array}{lcl@{\qquad}lcl}
\langle\, \mathsf{act}(i), (\mathsf{d}, \mathsf{a}, \mathsf{a})\,\rangle & \mapsto & p_i/4 & \langle\, (\mathsf{a}, \mathsf{a}, \mathsf{a})\,\rangle & \mapsto & (1-p_0-p_1-p_2)/4 \\
\langle\, \mathsf{act}(i), (\mathsf{a}, \mathsf{d}, \mathsf{a})\,\rangle & \mapsto & p_i/4 & \langle\, (\mathsf{a}, \mathsf{d}, \mathsf{d})\,\rangle & \mapsto & (1-p_0-p_1-p_2)/4 \\
\langle\, \mathsf{act}(i), (\mathsf{a}, \mathsf{a}, \mathsf{d})\,\rangle & \mapsto & p_i/4 & \langle\, (\mathsf{d}, \mathsf{a}, \mathsf{d})\,\rangle & \mapsto & (1-p_0-p_1-p_2)/4 \\
\langle\, \mathsf{act}(i), (\mathsf{d}, \mathsf{d}, \mathsf{d})\,\rangle & \mapsto & p_i/4 & \langle\, (\mathsf{d}, \mathsf{d}, \mathsf{a})\,\rangle & \mapsto & (1-p_0-p_1-p_2)/4 \\
 & & (\text{for } i = 0, 1, 2) & & &
\end{array}$$

The other lists in $\mathcal{A}^+$ have probability 0.

In this work we assume that in each execution of an anonymizing protocol there appears at most one actor action. This is the same assumption as [15, Assumption 1] and is true in all the examples in this chapter.

4.3.6 Assumption (At most one actor action)**.** Let $\mathcal{X} = (X, \mathcal{U}, \mathcal{O}, c, s)$ be an anonymity automaton and $\overrightarrow{a} \in \mathcal{A}^+$. If $\overrightarrow{a}$ contains more than one actor actions, then we have $P_{\mathcal{X}}(\overrightarrow{a}) = 0$.

4.3.4 Notions of probabilistic anonymity

In this section we formalize two different notions of probabilistic anonymity, namely strong (probabilistic) anonymity [15] and probable innocence [23, 41, 112].

First, for the sake of simplicity of presentation, we shall introduce the following notations for predicates (i.e. subsets) on $\mathcal{A}^+$.

4.3.7 Definition (Predicates $[\mathsf{act}(i)]$ and $[\overrightarrow{o}]$)**.**

- For each $i \in \mathcal{U}$, a predicate $[\mathsf{act}(i)]$ on $\mathcal{A}^+$ is defined as follows. In a regular-expression-like notation,

  $$[\mathsf{act}(i)] = \mathcal{O}^* \cdot \mathsf{act}(i) \cdot \mathcal{O}^* \enspace ,$$

  that is, it is the set of lists which contains only one actor action which is $\mathsf{act}(i)$. Obviously we have $[\mathsf{act}(i)] \cap [\mathsf{act}(j)] = \emptyset$ if $i \neq j$, by Assumption 4.3.6.
- For each $\overrightarrow{o} \in \mathcal{O}^*$, a predicate $[\overrightarrow{o}]$ on $\mathcal{A}^+$ is defined as follows.

  $$[\overrightarrow{o}] = \{\overrightarrow{a} \in \mathcal{A}^+ \mid \mathsf{removeActor}(\overrightarrow{a}) = \overrightarrow{o}\} \enspace ,$$

  where the function $\mathsf{removeActor} : \mathcal{A}^+ \to \mathcal{O}^*$—which is defined by a suitable induction—removes actor actions appearing in a list. Hence the set $[\overrightarrow{o}] \subseteq \mathcal{A}^+$ consists of those lists which yield $\overrightarrow{o}$ as the adversary's observation. It is emphasized that $[\overrightarrow{o}]$ is *not* the set of lists which contain $\overrightarrow{o}$ as sublists: we remove only actor actions, but do not remove observable actions.

Note that we are overriding the notation [_]: no confusion would arise since the arguments are of different types. Values such as $P_{\mathcal{X}}(\,[\mathsf{act}(i)]\,)$ are defined in a straightforward manner:

$$P_{\mathcal{X}}(\,[\mathsf{act}(i)]\,) = \sum_{\overrightarrow{a} \in [\mathsf{act}(i)]} P_{\mathcal{X}}(\,\overrightarrow{a}\,) \ .$$

This is the probability with which $\mathcal{X}$ yields an execution in which the user i is a culprit.

Strong (probabilistic) anonymity

Based on anonymity automata as models of anonymity protocols, we shall formalize the notion of strong anonymity which is informally introduced in Definition 4.2.1.

4.3.8 Definition (Strong anonymity [15]). We say an anonymity automaton $\mathcal{X}$ satisfies *strong anonymity* if, for each $i, j \in \mathcal{U}$ and $\overrightarrow{o} \in \mathcal{O}^*$,

$$\begin{aligned} & P_{\mathcal{X}}(\,[\mathsf{act}(i)]\,) > 0 \quad \wedge \quad P_{\mathcal{X}}(\,[\mathsf{act}(j)]\,) > 0 \\ \Longrightarrow \quad & P_{\mathcal{X}}(\,[\overrightarrow{o}\,] \mid [\mathsf{act}(i)]\,) = P_{\mathcal{X}}(\,[\overrightarrow{o}\,] \mid [\mathsf{act}(j)]\,) \ . \end{aligned}$$

Here $P_{\mathcal{X}}(\,[\overrightarrow{o}\,] \mid [\mathsf{act}(i)]\,)$ is a conditional probability: it is given by

$$P_{\mathcal{X}}(\,[\overrightarrow{o}\,] \mid [\mathsf{act}(i)]\,) = \frac{P_{\mathcal{X}}(\,[\overrightarrow{o}\,] \cap [\mathsf{act}(i)]\,)}{P_{\mathcal{X}}(\,[\mathsf{act}(i)]\,)} \ .$$

It is shown in [15] that under reasonable assumptions this notion coincides with *conditional anonymity* [41], whose formal definition we present now.

4.3.9 Definition (Conditional anonimity [41]). An anonymity automaton $\mathcal{X}$ satisfies *conditional anonymity* if for each $i \in \mathcal{U}$ and $\overrightarrow{o} \in \mathcal{O}^*$,

$$\begin{aligned} & P_{\mathcal{X}}(\,[\mathsf{act}(i)] \cap [\overrightarrow{o}\,]\,) > 0 \\ \Longrightarrow \quad & P_{\mathcal{X}}(\,[\mathsf{act}(i)] \mid [\overrightarrow{o}\,]\,) = P_{\mathcal{X}}(\,[\mathsf{act}(i)] \mid \bigcup_{j \in \mathcal{U}} [\mathsf{act}(j)]\,) \ . \end{aligned}$$

The notion in Definition 4.3.8 is a natural probabilistic adaptation of *trace anonymity* in [119]. It is emphasized that these anonymity notions are based on trace semantics which is at the coarsest end in the linear time-branching time spectrum [39]. Hence our adversary has less observation power than the one in [1], for example, where security notions are bisimulation-based.

Our choice of such a weak adversary is justified as long as we think of a passive adversary—or an "eavesdropper"—which passively observe an execution of the protocol without actively taking part in it. In fact, each process semantics in the linear time-branching time spectrum [39] has a corresponding *testing scenario*, an informal description about what kind of observation will have the distinguishing power prescribed by the process semantics. The ability of a passive adversary coincides with the testing scenario for the weakest process semantics in the spectrum, namely trace semantics.

Probable innocence

The weaker notion of *probable innocence*—informally introduced in Definition 4.2.2—is also formalized based on anonymity automata and their trace semantics. It is based on the definition in [23].

4.3.10 Definition (Probable innocence [23])**.** We say an anonymity automaton $\mathcal{X}$ satisfies *probable innocence* if, for each $i \in \mathcal{U}$ and $\overrightarrow{o} \in \mathcal{O}^*$,

- Neither of the probabilities $P_{\mathcal{X}}(\bigcup_{j\neq i}[\mathsf{act}(j)])$ and $P_{\mathcal{X}}(\,\bigcup_{j\neq i}[\mathsf{act}(j)] \mid [\overrightarrow{o}]\,)$ is 0. These appear as denominators in the following inequality.
- We have

$$(n-1)\frac{P_{\mathcal{X}}(\,[\mathsf{act}(i)]\,)}{P_{\mathcal{X}}(\,\bigcup_{j\neq i}[\mathsf{act}(j)]\,)} \geq \frac{P_{\mathcal{X}}(\,[\mathsf{act}(i)] \mid [\overrightarrow{o}]\,)}{P_{\mathcal{X}}(\,\bigcup_{j\neq i}[\mathsf{act}(j)] \mid [\overrightarrow{o}]\,)} \quad . \tag{4.3}$$

Here $n = |\mathcal{U}|$ is the number of users.

4.4 Verifying strong anonymity with probabilistic simulations

In this section we adapt the proof method [71, 72] for (non-deterministic) trace anonymity to the probabilistic setting for strong (probabilistic) anonymity. In Section 4.1.2 we have presented its basic scenario. Now we shall describe its details, with all the notions therein (traces, simulations, etc.) interpreted probabilistically.

4.4.1 Anonymized automaton $\mathsf{an}(\mathcal{X})$

We start with the definition of $\mathsf{an}(\mathcal{X})$, the *anonymized version* of an anonymity automaton $\mathcal{X}$. Recall that the notion of strong anonymity is conditional: the adversary has a-priori knowledge on who is more suspicious. In an anonymity automaton $\mathcal{X}$, the a-priori probability with which a user i is a culprit is given by $P_{\mathcal{X}}(\,[\mathsf{act}(i)]\,)$. Its normalized, conditional version

$$\begin{aligned} r_i \overset{\text{def.}}{=} P_{\mathcal{X}}(\,[\mathsf{act}(i)] \mid \bigcup_{j\in\mathcal{U}}[\mathsf{act}(j)]\,) &= \frac{P_{\mathcal{X}}(\,[\mathsf{act}(i)] \cap \bigcup_{j\in\mathcal{U}}[\mathsf{act}(j)]\,)}{P_{\mathcal{X}}(\,\bigcup_{j\in\mathcal{U}}[\mathsf{act}(j)]\,)} \\ &= \frac{P_{\mathcal{X}}(\,[\mathsf{act}(i)]\,)}{\sum_{j\in\mathcal{U}} P_{\mathcal{X}}(\,[\mathsf{act}(j)]\,)} \end{aligned} \tag{4.4}$$

(the equalities are due to Assumption 4.3.6) plays an important role in the following definition of $\mathsf{an}(\mathcal{X})$. The value r_i is the conditional probability with which a user i is a culprit, given that there is any culprit; we have $\sum_{i\in\mathcal{U}} r_i = 1$. Of course, for the values r_i to be well-defined, the anonymity automaton $\mathcal{X}$ needs to satisfy the following reasonable assumption.

4.4.1 Assumption (There must be a culprit)**.** For an anonymity automaton $\mathcal{X}$,

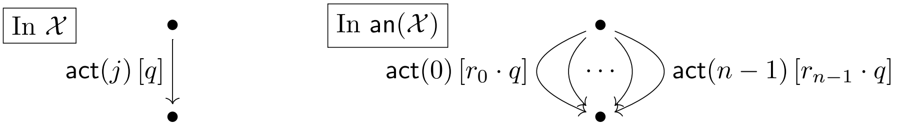
$$\sum_{j \in \mathcal{U}} P_{\mathcal{X}}(\,[\mathsf{act}(j)]\,) \neq 0 \enspace .$$

Intuitively, $\mathsf{an}(\mathcal{X})$ is obtained from $\mathcal{X}$ by distributing the probability for an actor action $\mathsf{act}(i)$ to each user j in proportion to r_j.

4.4.2 Definition (Anonymized anonymity automaton $\mathsf{an}(\mathcal{X})$)**.** Given an anonymity automaton $\mathcal{X} = (X, \mathcal{U}, \mathcal{O}, c, s)$, its *anonymized anonymity automaton* $\mathsf{an}(\mathcal{X})$ is a quintuple $(X, \mathcal{U}, \mathcal{O}, c^{\mathsf{an}}, s)$, where c^{an} is defined as follows. For each $x \in X$,

$$\begin{array}{lcll} c^{\mathsf{an}}(x)(\mathsf{act}(i), u) & = & \sum_{j \in \mathcal{U}} r_i \cdot c(x)(\mathsf{act}(j), u) & \text{for } i \in \mathcal{U} \text{ and } u \in \{\checkmark\} + X, \\ c^{\mathsf{an}}(x)(o, u) & = & c(x)(o, u) & \text{for } o \in \mathcal{O} \text{ and } u \in \{\checkmark\} + X. \end{array}$$

In the first equation, the summand $r_i \cdot c(x)(\mathsf{act}(j), u)$ results from distributing the probability $c(x)(\mathsf{act}(j), u)$ for a transition $x \xrightarrow{\mathsf{act}(j)} u$, to a user i. This is illustrated in the following figure: here $\mathcal{U} = \{0, 1, \ldots, n-1\}$ and $q = c(x)(\mathsf{act}(j), u)$.


In $\mathcal{X}$
$\mathsf{act}(j)\ [q]$
In $\mathsf{an}(\mathcal{X})$
$\mathsf{act}(0)\ [r_0 \cdot q]$
$\cdots$
$\mathsf{act}(n-1)\ [r_{n-1} \cdot q]$
(4.5)

The automaton $\mathsf{an}(\mathcal{X})$ is "anonymized" in the sense of the following lemmas.

4.4.3 Lemma. *Let $\mathcal{X}$ be an anonymity automaton. In its anonymized version $\mathsf{an}(\mathcal{X}) = (X, \mathcal{U}, \mathcal{O}, c^{\mathsf{an}}, s)$ we have*

$$r_j \cdot c^{\mathsf{an}}(x)(\mathsf{act}(i), u) = r_i \cdot c^{\mathsf{an}}(x)(\mathsf{act}(j), u)$$

for any $i, j \in \mathcal{U}$, $x \in X$ and $u \in \{\checkmark\} + X$.

Proof. Obvious from the definition of c^{an}. □

4.4.4 Lemma ($\mathsf{an}(\mathcal{X})$ satisfies strong anonymity)**.** *Given an anonymity automaton $\mathcal{X}$, its anonymized version $\mathsf{an}(\mathcal{X})$ satisfies strong anonymity in the sense of Definition 4.3.8.*

Proof. Let $\overrightarrow{o} = \langle o_1, o_2, \ldots, o_n \rangle \in \mathcal{O}^*$ and $i, j \in \mathcal{U}$. Moreover, assume

$$P_{\mathsf{an}(\mathcal{X})}(\,[\mathsf{act}(i)]\,) \neq 0 \quad \text{and} \quad P_{\mathsf{an}(\mathcal{X})}(\,[\mathsf{act}(j)]\,) \neq 0 \enspace ,$$

hence $r_i \neq 0$ and $r_j \neq 0$. Then

$$\begin{aligned}
&P_{\mathsf{an}(\mathcal{X})}(\,[\overrightarrow{o}] \cap [\mathsf{act}(i)]\,) \\
&= P_{\mathsf{an}(\mathcal{X})}(\,\langle \mathsf{act}(i), o_1, o_2, \ldots, o_n \rangle\,) \\
&\quad + P_{\mathsf{an}(\mathcal{X})}(\,\langle o_1, \mathsf{act}(i), o_2, \ldots, o_n \rangle\,) \\
&\quad + \cdots \\
&\quad + P_{\mathsf{an}(\mathcal{X})}(\,\langle o_1, o_2, \ldots, o_n, \mathsf{act}(i) \rangle\,) \\
&= \sum_{x_0, x_1, \ldots, x_n \in X} s(x_0) \cdot c^{\mathsf{an}}(x_0)(\mathsf{act}(i), x_1) \cdot c^{\mathsf{an}}(x_1)(o_1, x_2) \cdot \cdots \cdot c^{\mathsf{an}}(x_n)(o_n, \checkmark) \\
&\quad + \sum_{x_0, x_1, \ldots, x_n \in X} s(x_0) \cdot c^{\mathsf{an}}(x_0)(o_1, x_1) \cdot c^{\mathsf{an}}(x_1)(\mathsf{act}(i), x_2) \cdot \cdots \cdot c^{\mathsf{an}}(x_n)(o_n, \checkmark) \\
&\quad + \cdots \\
&\quad + \sum_{x_0, x_1, \ldots, x_n \in X} s(x_0) \cdot c^{\mathsf{an}}(x_0)(o_1, x_1) \cdot c^{\mathsf{an}}(x_1)(o_2, x_2) \cdot \cdots \cdot c^{\mathsf{an}}(x_n)(\mathsf{act}(i), \checkmark) \;.
\end{aligned}$$

We have the same equation for j instead of i. Hence by Lemma 4.4.3 we have

$$r_j \cdot P_{\mathsf{an}(\mathcal{X})}(\,[\overrightarrow{o}] \cap [\mathsf{act}(i)]\,) = r_i \cdot P_{\mathsf{an}(\mathcal{X})}(\,[\overrightarrow{o}] \cap [\mathsf{act}(j)]\,) \;. \tag{4.6}$$

This is used to show the equality of two conditional probabilities.

$$\begin{aligned}
P_{\mathsf{an}(\mathcal{X})}(\,[\overrightarrow{o}] \mid [\mathsf{act}(i)]\,) &= \frac{P_{\mathsf{an}(\mathcal{X})}(\,[\overrightarrow{o}] \cap [\mathsf{act}(i)]\,)}{P_{\mathsf{an}(\mathcal{X})}(\,[\mathsf{act}(i)]\,)} \\
&= \frac{r_i}{r_j} \cdot \frac{P_{\mathsf{an}(\mathcal{X})}(\,[\overrightarrow{o}] \cap [\mathsf{act}(j)]\,)}{P_{\mathsf{an}(\mathcal{X})}(\,[\mathsf{act}(i)]\,)} && \text{by (4.6)} \\
&= \frac{P_{\mathsf{an}(\mathcal{X})}(\,[\overrightarrow{o}] \cap [\mathsf{act}(j)]\,)}{P_{\mathsf{an}(\mathcal{X})}(\,[\mathsf{act}(j)]\,)} && \text{by definition of } r_i, r_j \\
&= P_{\mathsf{an}(\mathcal{X})}(\,[\overrightarrow{o}] \mid [\mathsf{act}(j)]\,) \;.
\end{aligned}$$

□

4.4.2 Forward/backward simulations for anonymity automata

We proceed to introduce appropriate notions of forward and backward simulations. The (tedious) definition and a *soundness theorem*—existence of a forward/backward simulation implies trace inclusion—come for free from the generic theory in Chapter 3. This forms a crucial part of our simulation-based proof method.

4.4.5 Definition (Forward/backward simulations for anonymity automata)**.** Let $\mathcal{X} = (X, \mathcal{U}, \mathcal{O}, c, s)$ and $\mathcal{Y} = (Y, \mathcal{U}, \mathcal{O}, d, t)$ be anonymity automata which have the same sets of users and observable actions.

A *forward simulation* from $\mathcal{X}$ to $\mathcal{Y}$—through which $\mathcal{Y}$ *forward-simulates* $\mathcal{X}$—is a function

$$f : Y \longrightarrow \mathcal{D}X$$

which satisfies the following inequalities in $[0, 1]$.

$$\begin{array}{rcll} s(x) & \leq & \sum_{y \in Y} t(y) \cdot f(y)(x) & \text{for any } x \in X, \\ \sum_{x \in X} f(y)(x) \cdot c(x)(e, \checkmark) & \leq & d(y)(e, \checkmark) & \text{for any } y \in Y \text{ and } e \in \mathcal{A}, \\ \sum_{x \in X} f(y)(x) \cdot c(x)(e, x') & \leq & \sum_{y' \in Y} d(y)(e, y') \cdot f(y')(x') & \\ & & & \text{for any } y \in Y,\ e \in \mathcal{A} \text{ and } x' \in X. \end{array}$$

A *backward simulation* from $\mathcal{X}$ to $\mathcal{Y}$—through which $\mathcal{Y}$ *backward-simulates* $\mathcal{X}$—is a function

$$b : X \longrightarrow \mathcal{D}Y$$

which satisfies the following inequalities in $[0, 1]$.

$$\begin{array}{rcll} \sum_{x \in X} s(x) \cdot b(x)(y) & \leq & t(y) & \text{for any } y \in Y, \\ c(x)(e, \checkmark) & \leq & \sum_{y \in Y} b(x)(y) \cdot d(y)(e, \checkmark) & \text{for any } x \in X \text{ and } e \in \mathcal{A}, \\ \sum_{x' \in X} c(x)(e, x') \cdot b(x')(y') & \leq & \sum_{y \in Y} b(x)(y) \cdot d(y)(e, y') & \\ & & & \text{for any } x \in X,\ e \in \mathcal{A} \text{ and } y' \in Y. \end{array}$$

This definition (similar to the "probabilistic simulations" in Example 3.2.13) is an instantiation of the general, coalgebraic notions of forward/backward simulations (Definitions 3.2.8 and 3.2.9). More specifically, the two parameters T and F in the generic definition are instantiated as in Section 4.3.2.

4.4.6 Theorem (Soundness of forward/backward simulations). *Assume there is a forward (or backward) simulation from one anonymity automaton $\mathcal{X}$ to another $\mathcal{Y}$. Then we have trace inclusion*

$$P_{\mathcal{X}} \sqsubseteq P_{\mathcal{Y}} \ ,$$

where the order $\sqsubseteq$ is defined to be the pointwise order: for each $\overrightarrow{a} \in \mathcal{A}^{+}$,

$$P_{\mathcal{X}}(\overrightarrow{a}) \leq P_{\mathcal{Y}}(\overrightarrow{a}) \ .$$

Proof. We know (Lemma 4.3.4) that the notions of traces and simulations for anonymity automata are instantiations of the general, coalgebraic notions in the previous chapters. Therefore we can appeal to the general soundness theorem (Theorem 3.3.2). □

4.4.3 Probabilistic anonymity via simulations

We shall use the material in Section 4.4.1 and 4.4.2 to prove the validity of our simulation-based proof method (Theorem 4.4.11).

The following lemma—which essentially says $P_{\mathcal{X}} \sqsubseteq P_{\mathsf{an}(\mathcal{X})}$—relies on the way the automaton $\mathsf{an}(\mathcal{X})$ is constructed. The proof is a bit more complicated here than in the non-deterministic setting [71, 72].

4.4.7 Lemma. *Let $\mathcal{X}$ be an anonymity automaton. Assume there exists a forward or backward simulation from $\mathsf{an}(\mathcal{X})$ to $\mathcal{X}$—through which $\mathcal{X}$ simulates $\mathsf{an}(\mathcal{X})$. Then their trace semantics are equal:*

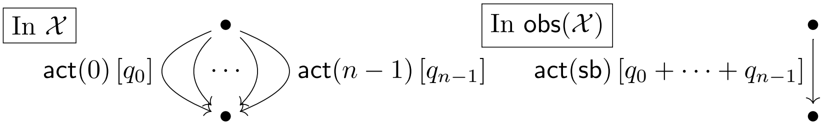
$$P_{\mathcal{X}} = P_{\mathsf{an}(\mathcal{X})} \ .$$

Proof. By the soundness theorem (Theorem 4.4.6) we have

$$P_{\mathcal{X}} \sqsupseteq P_{\mathsf{an}(\mathcal{X})} \ , \tag{4.7}$$

where $\sqsupseteq$ refers to the pointwise order between functions $\mathcal{A}^+ \rightrightarrows [0,1]$. We shall show that this inequality is in fact an equality.

First we introduce an operation obs which acts on anonymity automata. Intuitively, $\mathsf{obs}(\mathcal{Y})$ is obtained from $\mathcal{Y}$ by replacing all the different actor actions $\mathsf{act}(i)$ with single $\mathsf{act}(\mathsf{sb})$—$\mathsf{sb}$ is for "somebody." This conceals actor actions in $\mathcal{Y}$; hence $\mathsf{obs}(\mathcal{Y})$ only carries information on the observable actions of $\mathcal{Y}$.


In $\mathcal{X}$
$\mathsf{act}(0)\ [q_0]$
$\cdots$
$\mathsf{act}(n-1)\ [q_{n-1}]$
In $\mathsf{obs}(\mathcal{X})$
$\mathsf{act}(\mathsf{sb})\ [q_0 + \cdots + q_{n-1}]$

(4.8)

Formally,

4.4.8 Definition (Anonymity automaton $\mathsf{obs}(\mathcal{Y})$). Given an anonymity automaton $\mathcal{Y} = (Y, \mathcal{U}, \mathcal{O}, d, t)$, we define an automaton $\mathsf{obs}(\mathcal{Y})$ as the tuple $(Y, \{\mathsf{sb}\}, \mathcal{O}, d^{\mathsf{obs}}, t)$ where:

- sb is a fresh entity,
- d^{obs} is a function

$$d^{\mathsf{obs}} : Y \longrightarrow \mathcal{D}\big(\mathcal{A}^{\mathsf{obs}} \times \{\checkmark\} + \mathcal{A}^{\mathsf{obs}} \times Y\big)$$

where $\mathcal{A}^{\mathsf{obs}} = \mathcal{O} + \{\mathsf{act}(\mathsf{sb})\}$, defined by:

$$\begin{array}{lcll} d^{\mathsf{obs}}(y)(\mathsf{act}(\mathsf{sb}), u) & = & \sum_{i \in \mathcal{U}} d(y)(\mathsf{act}(i), u) & \text{for } y \in Y \text{ and } u \in \{\checkmark\} + Y, \\ d^{\mathsf{obs}}(y)(o, u) & = & d(y)(o, u) & \text{for } y \in Y,\ o \in \mathcal{O} \text{ and } u \in \{\checkmark\} + Y. \end{array}$$

The following fact is obvious.

4.4.9 Sublemma. For an anonymity automaton $\mathcal{X}$, $\mathsf{obs}(\mathcal{X})$ and $\mathsf{obs}(\mathsf{an}(\mathcal{X}))$ are identical. □

The following fact is crucial in the proof of Lemma 4.4.7. Two automata $\mathcal{Y}$ and $\mathsf{obs}(\mathcal{Y})$, although their trace semantics distributes over different sets, have the same sum of probabilities taken over all executions.

4.4.10 Sublemma. For an anonymity automaton $\mathcal{Y}$,

$$\sum_{\overrightarrow{a} \in \mathcal{A}^+} P_{\mathcal{Y}}(\overrightarrow{a}) = \sum_{\overrightarrow{a'} \in (\mathcal{A}^{\mathsf{obs}})^+} P_{\mathsf{obs}(\mathcal{Y})}(\overrightarrow{a'}) \ .$$

Recall that $\mathcal{A} = \mathcal{O} + \{\mathsf{act}(i) \mid i \in \mathcal{U}\}$ and $\mathcal{A}^{\mathsf{obs}} = \mathcal{O} + \{\mathsf{act}(\mathsf{sb})\}$.

Proof. From the definition of trace semantics (Definition 4.3.3), the sublemma is proved by easy calculation. □

We turn back to the proof of Lemma 4.4.7. We argue by contradiction—assume that the inequality in (4.7) is strict. That is, there exists $\overrightarrow{a_0} \in \mathcal{A}^+$ such that $P_{\mathcal{X}}(\overrightarrow{a_0}) \gneq P_{\mathsf{an}(\mathcal{X})}(\overrightarrow{a_0})$. Then, by (4.7) we have $\sum_{\overrightarrow{a} \in \mathcal{A}^+} P_{\mathcal{X}}(\overrightarrow{a}) \gneq \sum_{\overrightarrow{a} \in \mathcal{A}^+} P_{\mathsf{an}(\mathcal{X})}(\overrightarrow{a})$. However,

$$\begin{aligned} \textstyle\sum_{\overrightarrow{a} \in \mathcal{A}^+} P_{\mathcal{X}}(\overrightarrow{a}) &= \textstyle\sum_{\overrightarrow{a'} \in (\mathcal{A}^{\mathsf{obs}})^+} P_{\mathsf{obs}(\mathcal{X})}(\overrightarrow{a'}) && \text{by Sublemma 4.4.10} \\ &= \textstyle\sum_{\overrightarrow{a'} \in (\mathcal{A}^{\mathsf{obs}})^+} P_{\mathsf{obs}(\mathsf{an}(\mathcal{X}))}(\overrightarrow{a'}) && \text{by Sublemma 4.4.9} \\ &= \textstyle\sum_{\overrightarrow{a} \in \mathcal{A}^+} P_{\mathsf{an}(\mathcal{X})}(\overrightarrow{a}) && \text{by Sublemma 4.4.10.} \end{aligned}$$

This contradiction concludes the proof of Lemma 4.4.7. □

Now we are ready to state our main result for verifying strong anonymity.

4.4.11 Theorem (Strong anonymity via simulations)**.** *If there exists a forward or backward simulation from* $\mathsf{an}(\mathcal{X})$ *to* $\mathcal{X}$*, then* $\mathcal{X}$ *satisfies strong anonymity.*

Proof. By Lemma 4.4.7 we have $P_{\mathcal{X}} = P_{\mathsf{an}(\mathcal{X})}$. Moreover, by Lemma 4.4.4, $\mathsf{an}(\mathcal{X})$ is strongly anonymous. This proves strong anonymity of $\mathcal{X}$: recall that strong anonymity is a property defined in terms of traces (Definition 4.3.8). □

4.4.12 Example (Dining cryptographers)**.** We demonstrate our simulation-based proof method by applying it to the DC protocol.

Let $X = \{x, y_0, y_1, y_2\}$ be the state space of $\mathcal{X}_{\mathrm{DC}}$. Its anonymized version $\mathsf{an}(\mathcal{X}_{\mathrm{DC}})$ has the same state space; for notational convenience the state space of $\mathsf{an}(\mathcal{X}_{\mathrm{DC}})$ is denoted by $X' = \{x', y_0{}', y_1{}', y_2{}'\}$. It is verified by easy calculation that the following function $f : X \to \mathcal{D}(X')$ is a forward simulation from $\mathsf{an}(\mathcal{X}_{\mathrm{DC}})$ to $\mathcal{X}_{\mathrm{DC}}$.

$$f(x) = [x' \mapsto 1] \qquad f(y_0) = f(y_1) = f(y_2) = \begin{bmatrix} y_0{}' & \mapsto & \frac{p_0}{p_0+p_1+p_2} \\ y_1{}' & \mapsto & \frac{p_1}{p_0+p_1+p_2} \\ y_2{}' & \mapsto & \frac{p_2}{p_0+p_1+p_2} \end{bmatrix}$$

By Theorem 4.4.11 this proves (probabilistic) strong anonymity of $\mathcal{X}_{\mathrm{DC}}$, hence of the DC protocol.

4.5 Verifying probable innocence with probabilistic simulations

In this section we present a simulation-based proof method for the weaker notion of probable innocence. Although the basic idea is similar to that in the last section for strong anonymity, there are certain differences as well. Most notably, now we have to find a simulation from $\mathcal{X}$ to its "innocent version" $\mathsf{inno}(\mathcal{X})$; this is in the opposite direction of what we do for strong anonymity.

The basic scenario is as follows.

1. We model an anonymizing protocol as an anonymity automaton $\mathcal{X}$.
2. We construct the *innocent version* $\mathsf{inno}(\mathcal{X})$ of $\mathcal{X}$. Here it is not much of our concern whether $\mathsf{inno}(\mathcal{X})$ satisfies probable innocence or not. Rather $\mathsf{inno}(\mathcal{X})$ is thought of as the automaton describing the upper bound of admissible information leakage on who is the culprit. In fact, by examining the construction of $\mathsf{inno}(\mathcal{X})$, we can prove that
$$P_{\mathcal{X}} \sqsubseteq P_{\mathsf{inno}(\mathcal{X})} \quad\Longrightarrow\quad \mathcal{X} \text{ satisfies probable innocence.} \tag{4.9}$$
3. We find a simulation from $\mathcal{X}$ to $\mathsf{inno}(\mathcal{X})$. By the soundness theorem this yields trace inclusion $P_{\mathcal{X}} \sqsubseteq P_{\mathsf{inno}(\mathcal{X})}$, which by (4.9) proves that $\mathcal{X}$ satisfies probable innocence.

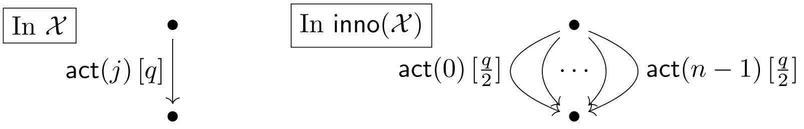
Although this scenario is somewhat different from the one for strong anonymity (or for non-deterministic trace anonymity [71, 72]), they are the same in that:

- we model a (trace-based) anonymity property as the "idealized version" of $\mathcal{X}$ (such as $\mathsf{an}(\mathcal{X})$ or $\mathsf{inno}(\mathcal{X})$);
- by finding a simulation and then appealing to the soundness theorem, we show that the property is transferred from the idealized version to the original $\mathcal{X}$.

Therein the construction of the "idealized version of $\mathcal{X}$" must be much tailored to the specific property to be verified.

4.5.1 "Innocent" automaton $\mathsf{inno}(\mathcal{X})$

We describe the construction of the innocent version $\mathsf{inno}(\mathcal{X})$ of a given anonymity automaton $\mathcal{X}$. To illustrate the intuition, let us assume that the a-priori suspicion is uniformly distributed, that is, $r_i = 1/n$ where $\mathcal{U} = \{0, 1, \ldots, n-1\}$. The automaton $\mathsf{inno}(\mathcal{X})$ is obtained by replacing actor actions in $\mathcal{X}$ as follows: compare it with (4.5).


In $\mathcal{X}$
$\mathsf{act}(j)$ $[q]$
In $\mathsf{inno}(\mathcal{X})$
$\mathsf{act}(0)$ $[\frac{q}{2}]$
$\cdots$
$\mathsf{act}(n-1)$ $[\frac{q}{2}]$

(4.10)

There is an obvious problem in this definition: the resulting $\mathsf{inno}(\mathcal{X})$ may not be an anonymity automaton, since the probabilities $q/2$ appearing n times can add up to more than 1. We shall call such an automaton an *extended anonymity automaton.*

4.5.1 Definition (Extended anonymity automata)**.** An *extended anonymity automaton* $(X, \mathcal{U}, \mathcal{O}, c, s)$ is the same thing as an anonymity automaton (Definition 4.3.1), except for the fact that c is a function of the type

$$X \longrightarrow \mathcal{V}\big(\mathcal{A} \times \{\checkmark\} + \mathcal{A} \times X\big)$$

rather than $X \to \mathcal{D}(\mathcal{A} \times \{\checkmark\} + \mathcal{A} \times X)$. Here the operation $\mathcal{V}$ is such that[6]

$$\mathcal{V}Y = [0,\infty]^Y = \bigl\{d : Y \to [0,\infty]\bigr\} \ . \tag{4.11}$$

The operation $\mathcal{V}$ canonically extends to a monad, which we shall call the *valuation monad.*

Given an extended anonymity automaton $\mathcal{X}$, its *trace semantics*

$$P_{\mathcal{X}} \in \mathcal{V}(A^+) \tag{4.12}$$

is defined exactly in the same way as for anonymity automata (Definition 4.3.3).

Given an anonymity automaton $\mathcal{X}$, it can be thought of as an extended anonymity automaton because there is an inclusion map $\mathcal{D}Y \hookrightarrow \mathcal{V}Y$:

$$X \ \overset{c}{\to} \ \mathcal{D}(\mathcal{A} \times \{\checkmark\} + \mathcal{A} \times X) \ \hookrightarrow \ \mathcal{V}(\mathcal{A} \times \{\checkmark\} + \mathcal{A} \times X) \ . \tag{4.13}$$

Therefore we are overriding the notation $P_{\mathcal{X}}$ in (4.12). This is not a problem, since for an anonymity automaton $\mathcal{X}$ the following two trace semantics obviously coincide.

- Its trace semantics (Definition 4.3.3) as an anonymity automaton, and
- the trace semantics of the extended anonymity automaton induced by (4.13).

Before giving a formal definition of $\mathsf{inno}(\mathcal{X})$, note that the core inequality (4.3) in the definition of probable innocence is equivalent to the following one, which uses the a-priori distribution of suspicion $(r_i)_{i\in\mathcal{U}}$ from (4.4):

$$P_{\mathcal{X}}(\,[\mathsf{act}(i)] \cap [\overrightarrow{o}]\,) \quad \le \quad \frac{(n-1)r_i}{1+(n-2)r_i} \cdot P_{\mathcal{X}}\Bigl(\,\bigl(\bigcup_{j\in\mathcal{U}} [\mathsf{act}(j)]\bigr) \cap [\overrightarrow{o}]\,\Bigr) \ . \tag{4.14}$$

The equivalence is shown by the following easy calculation:

$$\begin{array}{lrcl}
 & (n-1)\dfrac{P_{\mathcal{X}}(\,[\mathsf{act}(i)]\,)}{P_{\mathcal{X}}(\,\bigcup_{j\neq i}[\mathsf{act}(j)]\,)} & \ge & \dfrac{P_{\mathcal{X}}(\,[\mathsf{act}(i)] \mid [\overrightarrow{o}]\,)}{P_{\mathcal{X}}(\,\bigcup_{j\neq i}[\mathsf{act}(j)] \mid [\overrightarrow{o}]\,)} \\
\iff & \dfrac{(n-1)r_i}{\sum_{j\neq i} r_j} & \ge & \dfrac{P_{\mathcal{X}}(\,[\mathsf{act}(i)] \mid [\overrightarrow{o}]\,)}{P_{\mathcal{X}}(\,\bigcup_{j\neq i}[\mathsf{act}(j)] \mid [\overrightarrow{o}]\,)} \\
\iff & \dfrac{(n-1)r_i}{1-r_i} & \ge & \dfrac{P_{\mathcal{X}}(\,[\mathsf{act}(i)] \mid [\overrightarrow{o}]\,)}{P_{\mathcal{X}}(\,\bigcup_{j\neq i}[\mathsf{act}(j)] \mid [\overrightarrow{o}]\,)} \\
\iff & P_{\mathcal{X}}(\,[\mathsf{act}(i)] \mid [\overrightarrow{o}]\,) & \le & \dfrac{(n-1)r_i}{1-r_i} \cdot P_{\mathcal{X}}(\,\bigcup\limits_{j\neq i}[\mathsf{act}(j)] \mid [\overrightarrow{o}]\,) \\
\iff & \left(1+\dfrac{(n-1)r_i}{1-r_i}\right) P_{\mathcal{X}}(\,[\mathsf{act}(i)] \mid [\overrightarrow{o}]\,) & \le & \dfrac{(n-1)r_i}{1-r_i} \cdot P_{\mathcal{X}}(\,\bigcup\limits_{j\in\mathcal{U}}[\mathsf{act}(j)] \mid [\overrightarrow{o}]\,) \ ,
\end{array}$$

from which (4.14) follows immediately.

[6] The range of $d \in \mathcal{V}Y$ is $[0,\infty]$ including ∞. This choice is made because, in order to apply the coalgebraic theory of traces and simulations, $\mathcal{V}Y$ needs to carry a **Cppo**-structure. In particular, any increasing ω-chain in $\mathcal{V}Y$ must have its upper bound in $\mathcal{V}Y$.

4.5.2 Definition (Anonymity automaton $\mathsf{inno}(\mathcal{X})$)**.** Given an anonymity automaton $\mathcal{X} = (X, \mathcal{U}, \mathcal{O}, c, s)$, its *innocent version* $\mathsf{inno}(\mathcal{X})$ is an extended anonymity automaton given by the 5-tuple $(X, \mathcal{U}, \mathcal{O}, c^{\mathsf{inno}}, s)$, where c^{inno} is defined as follows. For each $x \in X$,

$$\begin{aligned} c^{\mathsf{inno}}(x)(\mathsf{act}(i), u) &= \frac{(n-1)r_i}{1+(n-2)r_i} \cdot \sum_{j \in \mathcal{U}} c(x)(\mathsf{act}(j), u) \\ &\qquad\qquad\qquad\quad \text{for } i \in \mathcal{U} \text{ and } u \in \{\checkmark\} + X, \\ c^{\mathsf{inno}}(x)(o, u) &= c(x)(o, u) \qquad \text{for } o \in \mathcal{O} \text{ and } u \in \{\checkmark\} + X. \end{aligned}$$

The coefficient $(n-1)r_i/(1+(n-2)r_i)$ comes from the inequality (4.14). The definition agrees with the informal description (4.10) when the a-priori suspicion $(r_i)_{i \in \mathcal{U}}$ is uniform.

4.5.2 Probable innocence via simulations

The intuition about the automaton $\mathsf{inno}(\mathcal{X})$ is that it represents the upper bound of admissible information leak. This intuition is put precise in the following lemma, which is a crucial step in our simulation-based proof method for probable innocence.

4.5.3 Lemma. *If we have*

$$P_{\mathcal{X}} \sqsubseteq P_{\mathsf{inno}(\mathcal{X})} \qquad \textit{that is,} \qquad \forall \overrightarrow{a} \in \mathcal{A}^{+}.\quad P_{\mathcal{X}}(\overrightarrow{a}) \le P_{\mathsf{inno}(\mathcal{X})}(\overrightarrow{a}) \enspace ,$$

then the anonymity automaton $\mathcal{X}$ satisfies probable innocence (Definition 4.3.10).

Proof. By calculation much like the one which shows the equality (4.6) in the proof of Lemma 4.4.4, we obtain the following.

$$P_{\mathsf{inno}(\mathcal{X})}(\,[\overrightarrow{o}] \cap [\mathsf{act}(i)]\,) = \frac{(n-1)r_i}{1+(n-2)r_i} \cdot \sum_{j \in \mathcal{U}} P_{\mathcal{X}}(\,[\overrightarrow{o}] \cap [\mathsf{act}(j)]\,) \tag{4.15}$$

By assumption, we have

$$P_{\mathcal{X}}(\,[\overrightarrow{o}] \cap [\mathsf{act}(i)]\,) \le P_{\mathsf{inno}(\mathcal{X})}(\,[\overrightarrow{o}] \cap [\mathsf{act}(i)]\,) \enspace ,$$

which, together with (4.15), derives (4.14). □

The assumption of Lemma 4.5.3—trace inclusion $P_{\mathcal{X}} \sqsubseteq P_{\mathsf{inno}(\mathcal{X})}$—shall be shown by finding a probabilistic simulation. Now that $\mathsf{inno}(\mathcal{X})$ is an *extended* anonymity automaton, we have to adapt the notion of probabilistic simulations and its soundness result to the extended setting. In terms of the generic theory in the previous chapters, this corresponds to changing the parameter T from $\mathcal{D}$ to $\mathcal{V}$.

4.5.4 Definition (Forward/backward simulations, extended)**.** Let $\mathcal{X}$ and $\mathcal{Y}$ be extended anonymity automata. A *forward (or backward) simulation* from $\mathcal{X}$ to $\mathcal{Y}$ is the same thing as a forward (or backward) simulation for anonymity automata (Definition 4.4.5), except that

- we have now $\mathcal{V}$ in place of $\mathcal{D}$, and
- the inequalities are interpreted in $[0,\infty]$ rather than in $[0,1]$.

For example, a forward simulation from $\mathcal{X}$ to $\mathcal{Y}$ is a function

$$f : Y \longrightarrow \mathcal{V}X$$

which satisfies suitable inequalities as in Definition 4.4.5.

4.5.5 Theorem (Soundness of simulations, extended)**.** *If there is a forward or backward simulation from an extended anonymity automaton $\mathcal{X}$ to another $\mathcal{Y}$, then we have trace inclusion*

$$P_{\mathcal{X}} \sqsubseteq P_{\mathcal{Y}} \ , \qquad \textit{that is,} \qquad \forall \overrightarrow{a} \in \mathcal{A}^{+}. \quad P_{\mathcal{X}}(\overrightarrow{a}) \leq P_{\mathcal{Y}}(\overrightarrow{a}) \ .$$

Proof. The general, coalgebraic theory of traces in the previous chapters applies to the choice of parameters $T = \mathcal{V}$ and $F = \mathcal{A} \times \{\checkmark\} + \mathcal{A} \times \_$, in which case the coalgebraic notion of (T, F)-systems instantiates to extended anonymity automata. Moreover, the coalgebraic notions of trace semantics and simulations instantiate to the corresponding notions for extended anonymity automata (Definitions 4.5.1 and 4.5.4). Therefore the statement is an instance of the general soundness theorem (Theorem 3.3.2). □

4.5.6 Theorem (Probable innocence via simulations)**.** *If there exists a forward or backward simulation from $\mathcal{X}$ to* $\mathsf{inno}(\mathcal{X})$*, then $\mathcal{X}$ satisfies probable innocence.*

Proof. By the soundness theorem (Theorem 4.5.5) we have $P_{\mathcal{X}} \sqsubseteq P_{\mathsf{inno}(\mathcal{X})}$; by Lemma 4.5.3 this implies $\mathcal{X}$'s probable innocence. □

4.6 Related work

4.6.1 Probabilistic non-interference

In the theory of programming languages, a property called *non-interference* [97] has attracted much attention. Intuitively, a program C satisfies non-interference if there is no "insecure" information flow from variables with high confidentiality (*high variables*) to those with low confidentiality (*low variables*).

Let us put it slightly more formally. By a memory state we mean a list of pairs of a variable name and a value (such as `[H=0,L=0]`). Assume that two memory states μ and ν differ only in high variables ($\mu =_L \nu$). Non-interference of a program C requires that the execution of C with the initial memory state μ is observationally indistinguishable from that with ν. In particular, if μ' and ν' are the memory states after these executions, respectively, then they have to agree on low variables:

$$\mu =_L \nu, \quad \mu \overset{C}{\rightsquigarrow} \mu', \quad \nu \overset{C}{\rightsquigarrow} \nu' \quad \Longrightarrow \quad \mu' =_L \nu'.$$

A possible variant of the formal definition concerns C's termination as well.

Volpano and Smith [138] claim the importance of probabilistic aspects in non-interference. The leading example is in a simple programming language which has the *multi-threading* construct $C \mid C'$ which denotes running two threads C and C' in parallel, in an interleaving manner. Even though there is no explicit probabilistic construct in the language, a probabilistic *scheduler*—a mechanism that determines which thread to execute first—introduces probability in an operational model. In [138] a simple example is presented in which insecure information flow emerges only in a probabilistic setting (but not in a non-deterministic setting).

A definition of probabilistic non-interference (disregarding termination for the sake of simplicity) can be presented as follows. Let μ_H, ν_H be memory states of high variables; μ_L, μ'_L be states of low variables. A program C satisfies probabilistic non-interference if, for every μ_H, ν_H, μ_L and μ'_L,

$$\begin{aligned}&\mathsf{Pr}(\mu_L \overset{C}{\rightsquigarrow} \mu'_L \mid \text{initial high-memory is } \mu_H)\\ &= \mathsf{Pr}(\mu_L \overset{C}{\rightsquigarrow} \mu'_L \mid \text{initial high-memory is } \nu_H)\ .\end{aligned}$$

Notice similarity to the notion of strong anonymity (Definition 4.2.1). One can think of the change of low-memory $\mu_L \overset{C}{\rightsquigarrow} \mu'_L$ as an “observation”; and the initial high-memory as a “culprit,” i.e. the information that we want to disguise. This similarity between the two notions—(strong) probabilistic anonymity and probabilistic non-interference—suggests that a technique to ensure one of these properties can be used to ensure the other property as well.

Use of *type systems* is a standard technique to ensure that a certain piece of program satisfies a certain desirable property, such as non-interference. A type system typically has *typing rules*, and a *soundness theorem* stating: if a program C is *typable*—meaning that we can derive a typing judgment for C using the given rules—then C satisfies the desired property. One big advantage of the type-based approach is that type systems are often accompanied with *typing algorithms* that effectively determine whether a program is typable or not.

For probabilistic non-interference, a number of type systems have been proposed [117, 138]. They are in principle extensions of the type system in [126] that ensures non-deterministic notion of non-interference. For example, the following program with an obvious (but indirect) insecure information flow

```
if H=0 then L=0 else L=1;                    (4.16)
```

is typable in none of the type systems in [117, 126, 138]. Moreover, any program containing this piece is not typable either, because of the compositional way of type derivation in these type systems.

Because of this fact the type systems from [117, 138] do not seem widely applicable to verification of anonymizing protocols. Consider the example of Dining Cryptographers: a natural representation of the protocol as a program should include a piece like (4.16) expressing “if a cryptographer is the payer, then she lies about her announcement.” Therefore such a program for DC is not typable.

In the DC protocol each cryptographer's behavior is certainly influenced by the information to be hidden; this is why the type systems from [117, 138] do not seem to apply. Nevertheless, when three cryptographers are "composed" in the way prescribed by the protocol, they manage to hide who is the payer. In this sense, *broken anonymity is not compositional.* A type system, if one wants to use it for verification of anonymizing protocols, needs to address such a non-compositional phenomenon. However potential complexity of such a type system would make it harder to come up with a useful typing algorithm.

4.6.2 Process algebraic techniques

In the current work we have presented anonymizing protocols in the form of automata, or transition systems. Using *process algebraic terms* or *process terms*—such as $P + Q$ for "non-deterministic choice between P and Q"—is another way of expressing processes such as anonymizing protocols. In fact, a process term induces a transition system via the structural operational semantics of the process algebra [3], hence a process algebra can be seen as a "programming language" which denotes transition systems.

One merit of using process algebras is that an algebraic structure in process terms often aids reasoning about properties of the term (more precisely, properties of the automaton denoted by the term). For example, one can employ inductive reasoning on the (inductive) construction of process terms. Many tools are available for analyzing properties of process terms.

Process algebraic techniques have been successfully applied in many anonymity applications. In [27], the MUTE anonymous file-sharing system [113] is analyzed by representing the protocol as a π-calculus term. The analysis led to discovery of a flaw, which is detected by the ABC bisimulation checker [20] for the π-calculus. A detailed account of the flaw is in [28]. In [29] a general framework for automatic checking of anonymity is proposed; the framework is based on the process algebra μCRL. However, all the work mentioned here is done in non-deterministic settings. A probabilistic framework of such a kind which would aid automatic error-detection/verification of anonymity is one interesting direction of future work.

The *probabilistic π-calculus* [53, 105] is a process algebra that can model probabilistic choices. A process term in that calculus yields a probabilistic automaton [121] as its operational model. It is used to model anonymity protocols in [15]. In a similar direction, recent work [24] uses a variant of CCS with an additional operator for probabilistic choices.

4.7 Summary and future work

We have extended the simulation-based proof method [71, 72] for non-deterministic anonymity to apply to the notion of strong probabilistic anonymity [15]. For the move we have exploited a generic theory of traces and simulations developed in Chapters 2 and 3, in which the difference between non-determinism and probability is just a different choice of a parameter. Additionally a simulation-based proof method for the weaker notion of probable innocence is introduced.

The DC example in this chapter fails to demonstrate the usefulness of our proof method. For this small example direct calculation of trace distribution is not hard. A real benefit would arise in theorem-proving anonymity of an unboundedly large system (which we cannot model-check). In fact, the non-deterministic version of our proof method is used to theorem-prove anonymity of a voting protocol with arbitrary many voters [71]. A probabilistic case study of such kind is currently missing.

Chapter 5

Foundational study: concurrency and the microcosm principle

Coalgebras are categorical presentations of state-based systems. In investigating parallel composition of coalgebras (realizing *concurrency*), we observe that the same algebraic theory is interpreted in two different domains in a nested manner, namely: in the category of coalgebras, and in the final coalgebra as an object in it. This phenomenon is what Baez and Dolan have called the *microcosm principle*, a prototypical example of which is "a monoid in a monoidal category." In the current work we obtain a formalization of the microcosm principle in which such a nested model is expressed categorically as a suitable lax natural transformation. An application of this account is a general compositionality result which supports modular verification of complex systems.

5.1 Overview

Design of systems with *concurrency* is nowadays one of the mainstream challenges in computer science [91]. Concurrency is everywhere: with the Internet being the biggest example and multi-core processors probably the smallest; also in a modular, component-based architecture of a complex system, its components collaborate in a concurrent manner. However numerous difficulties have been identified in getting concurrency right. For example, a system's exponentially growing complexity is one of the main obstacles. One way to cope with it is a *modular* verification method in which correctness of the whole system $\mathcal{C}_1 \parallel \cdots \parallel \mathcal{C}_n$ is established using correctness of each component $\mathcal{C}_i$. *Compositionality*—meaning that the behavior of $\mathcal{C} \parallel \mathcal{D}$ is determined by the behavior of $\mathcal{C}$ and that of $\mathcal{D}$—is an essential property for such a modular method to work.

5.1.1 Coalgebras as systems

The work described in this chapter is a starting point of our research program aimed at better understanding of the mathematical nature of concurrency. In its course we shall use *coalgebras* as presentations of systems to be run in parallel. The use of coalgebras as an appropriate abstract model of state-based systems is increasingly established (see Chapter 1); the notion's mathematical simplicity and clarity provide us with a sound foundation for our exploration. We recall the following table which summarizes how ingredients of the theory of systems are presented as coalgebraic constructs. The same table appeared before as (1.9)

	system	behavior-preserving map	behavior
coalgebraically	coalgebra $\begin{tikzcd} FX \\ X \arrow[u] \end{tikzcd}$	morphism of coalgebras $\begin{tikzcd} FX \arrow[r, "Ff"] & FY \\ X \arrow[u] \arrow[r, "f"'] & Y \arrow[u] \end{tikzcd}$	by coinduction $\begin{tikzcd} FX \arrow[r, dashed] & FZ \\ X \arrow[u, "c"] \arrow[r, dashed, "\mathsf{beh}(c)"'] & Z \arrow[u, "\mathrm{final}"'] \end{tikzcd}$

(5.1)

This view of "coalgebras as systems" has been successfully applied in the category **Sets** of sets and functions, in which case the word "behavior" in (5.1) refers (roughly) to the one modulo bisimilarity. Additionally, in Chapters 2 and 3 we have shown that "behavior" in the table can also refer to trace semantics, by moving from **Sets** to a suitable Kleisli category.

5.1.2 Compositionality in coalgebras

We start with the following question: what is "compositionality" in this coalgebraic setting? Conventionally compositionality is expressed as:

$$\mathcal{C}_1 \sim \mathcal{C}_2 \quad \text{and} \quad \mathcal{D}_1 \sim \mathcal{D}_2 \quad \Longrightarrow \quad \mathcal{C}_1 \parallel \mathcal{D}_1 \ \sim \ \mathcal{C}_2 \parallel \mathcal{D}_2 \ ,$$

where the relation $\sim$ denotes the behavioral equivalence of interest. If this is the case the relation $\sim$ is said to be a *congruence*, with its oft-heard instance being "bisimilarity is a congruence."

When we interpret "behavior" in compositionality as the coalgebraic behavior induced by coinduction (see (5.1)), the following equation comes natural as a coalgebraic presentation of compositionality.

$$\mathsf{beh}\left(\begin{array}{c} FX \\ c\uparrow \\ X \end{array} \middle\| \begin{array}{c} FY \\ d\uparrow \\ Y \end{array}\right) = \mathsf{beh}\left(\begin{array}{c} FX \\ c\uparrow \\ X \end{array}\right) \Big\| \, \mathsf{beh}\left(\begin{array}{c} FY \\ d\uparrow \\ Y \end{array}\right) \tag{5.2}$$

But a closer look reveals that the two "parallel composition operators" $\parallel$ in the equation have in fact different types: the one on the left has a type

$$\mathbf{Coalg}_F \times \mathbf{Coalg}_F \xrightarrow{\parallel} \mathbf{Coalg}_F \ , \quad \text{composing coalgebras as systems;}$$

while the one on the right has a type[1]

$$Z \times Z \xrightarrow{\|} Z \ , \quad \text{composing states of the final coalgebra } Z \text{ as behaviors.}$$

Moreover, the two domains are actually nested: the latter one $Z \xrightarrow{\cong} FZ$ is an object of the former one $\mathbf{Coalg}_F$.

5.1.3 The microcosm principle

What we have just observed is one instance—which is not often claimed explicitly in computer science—of the *microcosm principle* as it is called by Baez and Dolan [9]. It refers to a phenomenon that the same algebraic theory (or algebraic "specification," consisting of operations and equations) is interpreted twice in a nested manner, once in a category $\mathbb{C}$ and the other time in its object $X \in \mathbb{C}$. This is not something very unusual, because "a monoid in a monoidal category" constitutes a prototypical example.

monoidal category $\mathbb{C}$		monoid $X \in \mathbb{C}$
$\otimes : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$	multiplication	$X \otimes X \xrightarrow{\mu} X$
$I \in \mathbb{C}$	unit	$I \xrightarrow{\eta} X$
$I \otimes X \cong X \cong X \otimes I$	unit law	$\begin{tikzcd} X \arrow[r] \arrow[dr, equal] & X \otimes X \arrow[d] & X \arrow[l] \arrow[dl, equal] \\ & X & \end{tikzcd}$
$X \otimes (Y \otimes Z) \cong (X \otimes Y) \otimes Z$	associativity law	$\begin{tikzcd} X \otimes X \otimes X \arrow[r] \arrow[d] & X \otimes X \arrow[d] \\ X \otimes X \arrow[r] & X \end{tikzcd}$

(5.3)

Notice here that the outer operation $\otimes$ appears in the formulation of the inner operation μ. Moreover, to be precise, in the inner "equations" the outer isomorphisms should be present in suitable places. Hence this monoid example demonstrates that, in such nested algebraic structures, the inner structure depends on the outer. What is a mathematically precise formalization of such nested models? Answering this question is a main goal of this chapter.

Such a formalization has been done in [9] when algebraic theories are specified in the form of *opetopes*. Here instead we shall formalize the microcosm principle for *Lawvere theories* [90], whose role as categorical representation of algebraic theories has been recognized in theoretical computer science.

As it turns out, our formalization looks like the situation on the right. Here $\mathbb{L}$ is a category (a Lawvere theory) representing an algebraic theory; an outer model $\mathbb{C}$ is a product-preserving functor $\mathbb{L} \to \mathbf{CAT}$; and an inner model X is a lax natural transformation. The whole setting is 2-categorical: 2-categories (categories in categories) serve as an appropriate basis for the microcosm principle (algebras in algebras).

$$\begin{tikzcd} \mathbb{L} \arrow[r, bend left=50, "\mathbf{1}", ""{name=U, below}] \arrow[r, "\mathbb{C}"', ""{name=D}] & \mathbf{CAT} \arrow[from=U, to=D, phantom, "\Downarrow X"] \end{tikzcd}$$

[1] At this stage the presentation remains sloppy for the sake of simplicity. Later in technical sections the first composition operator will be denoted by $\circledast$; and the second composition operator will have the type $Z \circledast Z \to Z$ instead of $Z \times Z \to Z$.

5.1.4 Applications to coalgebras: parallel composition via sync

The categorical account we have sketched above shall be applied to our original question about parallel composition of coalgebras. As a main application we prove a *generic compositionality theorem*. For an arbitrary algebraic theory $\mathbb{L}$, compositionality like (5.2) is formulated as follows: the "behavior" functor $\mathsf{beh} : \mathbf{Coalg}_F \to \mathbb{C}/Z$ (via coinduction) preserves an $\mathbb{L}$-structure. This general form of compositionality holds under the following assumptions: $\mathbb{C}$ has an $\mathbb{L}$-structure and $F : \mathbb{C} \to \mathbb{C}$ *lax*-preserves the $\mathbb{L}$-structure.

Turning back to the original setting (5.2) of "coalgebraic compositionality," these general assumptions read roughly as follows: the base category $\mathbb{C}$ has a binary operation $\|$; and the endofunctor F comes with a natural transformation $\mathsf{sync} : FX \parallel FY \to F(X \parallel Y)$. Essentially, this sync is what lifts $\|$ on $\mathbb{C}$ to $\|$ on $\mathbf{Coalg}_F$, hence "parallel composition via sync." It is called a *synchronization* because it specifies the way two systems synchronize with each other. In fact, for a fixed functor F there can be different choices of sync (such as CSP-style vs. CCS-style), which in turn yield different "parallel composition" operators on the category $\mathbf{Coalg}_F$.

5.1.5 Organization of the chapter

We shall not dive into our 2-categorical exploration from the beginning. In Section 5.2, we instead focus on one specific algebraic theory, namely the one for parallel composition of systems. Our emphasis there is on the fact that the sync natural transformation essentially gives rise to parallel composition $\|$, and the fact that equational properties of $\|$ (such as associativity) can be reduced to the corresponding equational properties of sync.

These concrete observations will provide us with intuition for abstract categorical constructs in Section 5.3, where we formalize the microcosm principle for an arbitrary Lawvere theory $\mathbb{L}$. In Section 5.4 this formalization is applied to our original question on concurrency; here results on coalgebras such as compositionality are proved in their full generality and abstraction. Additionally in Section 5.5 we address some specific issues which arise in a Kleisli category, where coinduction captures trace semantics.

In the current work we shall focus on *strict* algebraic structures on categories in order to avoid complicated coherence issues. This means for example that we only consider *strictly* monoidal categories for which the isomorphisms in (5.3) are in fact equalities. Investigation of "pseudo" or "strong" algebraic structures (such as not necessarily strictly monoidal categories) is left as future work: some preliminary observations are presented in Section 5.3.3.

5.1.6 Related work

Our interest is similar to that of studies of *bialgebraic structures* in computer science (such as [12, 67, 75, 78, 79, 116, 134]), in the sense that we are also concerned with algebraic structures on coalgebras as systems. Our current framework is distinguished in the following aspects.

First, we handle *equations* in an algebraic theory as an integral part of our approach. Equations such as associativity and commutativity appear explicitly as commutative diagrams in a Lawvere theory $\mathbb{L}$. We benefit from this explicitness in e.g. spelling out a condition for the generic associativity result (Theorem 5.2.4). In contrast, in the bialgebraic studies an algebraic theory is presented either by an endofunctor $X \mapsto \coprod_{\sigma\in\Sigma} X^{|\sigma|}$ or by a monad T. In the former case equations are simply not present; in the latter case equations are there but not as explicitly as in a Lawvere theory.

Secondly and more importantly, by considering higher-dimensional, nested algebraic structures, we can now compose different coalgebras as well as different states of the same coalgebra. In this way the current work can be seen as a higher-dimensional extension of the existing bialgebraic studies (the existing studies focus on "inner" algebraic structures).

5.2 Parallel composition of coalgebras

5.2.1 Parallel composition via sync natural transformation

Let us start with the equation (5.2), a coalgebraic representation of compositionality. The operator $\|$ on the left is of type $\mathbf{Coalg}_F \times \mathbf{Coalg}_F \to \mathbf{Coalg}_F$. It is natural to require functoriality of this operation, making it a *bifunctor*. A bifunctor—especially an associative one which we investigate later in Section 5.2.3—plays an important role in various applications of category theory. Usually such an (associative) bifunctor is called a *tensor* and denoted by $\boldsymbol{\otimes}$, a convention that we also follow.[2] Therefore the "compositionality" statement now looks as follows.

$$\mathsf{beh}\begin{pmatrix} FX & & FY \\ c\!\uparrow & \boldsymbol{\otimes} & d\!\uparrow \\ X & & Y \end{pmatrix} = \mathsf{beh}\begin{pmatrix} FX \\ c\!\uparrow \\ X \end{pmatrix} \Bigg\| \;\mathsf{beh}\begin{pmatrix} FY \\ d\!\uparrow \\ Y \end{pmatrix} \tag{5.4}$$

The first question is: when do we have such a tensor $\boldsymbol{\otimes}$ on $\mathbf{Coalg}_F$? In many applications of coalgebras, it is obtained by lifting a tensor $\otimes$ on the base category $\mathbb{C}$ to $\mathbf{Coalg}_F$.[3] Such a lifting is possible in presence of a natural transformation

$$\mathsf{sync}_{X,Y} \;:\; FX \otimes FY \longrightarrow F(X \otimes Y) \;,$$

used in

$$\begin{array}{ccc} FX & & FY \\ c\!\uparrow & \boldsymbol{\otimes} & d\!\uparrow \\ X & & Y \end{array} \quad := \quad \begin{array}{l} F(X\otimes Y) \\ \quad\;\uparrow \mathsf{sync}_{X,Y} \\ FX\otimes FY \\ \quad\;\uparrow c\otimes d \\ \;X\otimes Y \end{array} \quad . \tag{5.5}$$

[2] Hence what is called a tensor here in Section 5.2.1 is just a bifunctor, possibly satisfying equational axioms such as associativity. Later in Section 5.2.3 we investigate its equational properties.

[3] We use boldface $\boldsymbol{\otimes}$ for a tensor on $\mathbf{Coalg}_F$ to distinguish it from $\otimes$ on $\mathbb{C}$.

We shall call this sync a *synchronization* because its computational meaning is indeed a specification of the way two systems synchronize. This will be illustrated in the coming examples.

Once we have an outer parallel composition $\otimes$ in the form of a tensor, an inner operator $\|$ which composes behaviors (i.e. states of the final coalgebra) is also obtained immediately by the following coinduction.

$$\begin{tikzcd}
F(Z \otimes Z) \arrow[r, dashed] & FZ \\
Z \otimes Z \arrow[u, "\zeta\otimes\zeta"] \arrow[r, dashed, "\|"'] & Z \arrow[u, "\text{final}", "\zeta"']
\end{tikzcd} \tag{5.6}$$

Compositionality (5.4) is straightforward by finality: both sides of the equation are the unique coalgebra morphism from $c \otimes d$ to the final ζ. The following theorem summarizes the observations so far.

5.2.1 Theorem (Coalgebraic compositionality)**.** *Assume that a category $\mathbb{C}$ has a tensor $\otimes : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$ and an endofunctor $F : \mathbb{C} \to \mathbb{C}$ has a natural transformation* $\mathsf{sync}_{X,Y} : FX \otimes FY \to F(X \otimes Y)$*. If moreover there exists a final F-coalgebra $\zeta : Z \xrightarrow{\cong} FZ$, then:*

1. *The tensor $\otimes$ on $\mathbb{C}$ lifts to an "outer" tensor*

$$\otimes \;:\; \mathbf{Coalg}_F \times \mathbf{Coalg}_F \longrightarrow \mathbf{Coalg}_F \;,$$

   which we understand as an "outer" composition operator.

2. *We obtain an "inner" composition operator $\|: Z \otimes Z \to Z$ by coinduction (5.6).*
3. *Between the two composition operators the compositionality property (5.4) holds.*

□

We can put the compositionality property (5.4) in more abstract terms as "the functor $\mathsf{beh} : \mathbf{Coalg}_F \to \mathbb{C}/Z$ preserves the tensor," meaning that the following diagram commutes.

$$\begin{tikzcd}
\mathbf{Coalg}_F \times \mathbf{Coalg}_F \arrow[r, "\mathsf{beh}\times\mathsf{beh}"] \arrow[d, "\otimes"'] & \mathbb{C}/Z \times \mathbb{C}/Z \arrow[d, "\underline{\otimes}"] \\
\mathbf{Coalg}_F \arrow[r, "\mathsf{beh}"'] & \mathbb{C}/Z
\end{tikzcd}$$

Here the tensor $\underline{\otimes}$ on the slice category $\mathbb{C}/Z$ is given as follows, using the inner composition $\|$.

$$\left(\begin{array}{c}X\\ \downarrow f\\ Z\end{array}, \begin{array}{c}Y\\ \downarrow g\\ Z\end{array}\right) \overset{\underline{\otimes}}{\longmapsto} \begin{array}{c}X \otimes Y\\ \downarrow f \otimes g\\ Z \otimes Z\\ \downarrow \|\\ Z\end{array} \tag{5.7}$$

The point of Theorem 5.2.1 is as follows. Those parallel composition operators which are induced by sync are well-behaved ones: good properties like compositionality come for free. We shall present some examples in Section 5.2.2.

5.2.2 Remark. The view of parallel composition of systems as a tensor structure on $\mathbf{Coalg}_F$ has been previously presented in [70]. The interest there is in categorical structures on $\mathbf{Coalg}_F$ and not so much in properties of parallel composition such as compositionality. In [70] and other references an endofunctor F with sync (equipped with some additional compatibility) is called a *monoidal endofunctor*.[4]

5.2.2 Examples

In Sets: bisimilarity is a congruence

We shall focus on (finitely branching) LTSs, and bisimilarity as their process semantics. For this purpose it is appropriate to take **Sets** as our base category $\mathbb{C}$ and $\mathcal{P}_{\mathrm{fin}}(\Sigma \times \_)$ as the functor F (see Section 1.3.2). We use Cartesian products as a tensor on **Sets**. This means that a composition of two coalgebras has the product of the two state spaces as its state space, which matches our intuition. The functor $\mathcal{P}_{\mathrm{fin}}$ in F is the finite powerset functor; the finiteness assumption is needed for existence of a final F-coalgebra. In Section 1.3.2 we sketched that a final F-coalgebra captures bisimilarity via coinduction.

In considering parallel composition of LTSs, the following three examples are well-known ones.[5]

- *CSP-style* [55]: $a.P \parallel a.Q \xrightarrow{a} P \parallel Q$. For the whole system to make an a-action, each component has to make an a-action.
- *CCS-style* [99]: $a.P \parallel \overline{a}.Q \xrightarrow{\tau} P \parallel Q$, assuming $\Sigma = \{a, b, \dots\} \cup \{\overline{a}, \overline{b}, \dots\} \cup \{\tau\}$. When one component outputs on a channel a and the other inputs from a, then the whole system makes an internal τ move.
- *ACP-style* [13]: $a.P \parallel b.Q \xrightarrow{a|b} P \parallel Q$. The way processes interact with each other is parametrized by a partial binary operation $|$ on Σ. In fact, suitable choices of $|$ realize CSP- and CCS-style synchronization.

In fact, each of these different ways of synchronization can be represented by a suitable

[4] Later in Section 5.3 we will observe that a functor F with sync is a special case of a *lax* $\mathbb{L}$-functor. Such a functor F with sync is usually called a monoidal functor (as opposed to a *lax* monoidal functor), probably because it preserves (inner) monoid objects; see Proposition 5.3.13.1.

[5] Here we focus on synchronous interaction. All the process algebras mentioned here have an additional kind of interaction, namely an "interleaving" one; see Remark 5.2.3.

sync natural transformation.

$$\begin{array}{ccl}
\mathcal{P}_{\mathrm{fin}}(\Sigma \times X) \times \mathcal{P}_{\mathrm{fin}}(\Sigma \times Y) & \longrightarrow & \mathcal{P}_{\mathrm{fin}}\big(\Sigma \times (X \times Y)\big) \\
(u,v) & \overset{\mathsf{sync}^{\mathrm{CSP}}_{X,Y}}{\longmapsto} & \big\{\, (a,(x,y)) \mid (a,x) \in u \,\wedge\, (a,y) \in v \,\big\} \\
(u,v) & \overset{\mathsf{sync}^{\mathrm{CCS}}_{X,Y}}{\longmapsto} & \big\{\, (\tau,(x,y)) \mid (a,x) \in u \,\wedge\, (\overline{a},y) \in v \,\big\} \\
(u,v) & \overset{\mathsf{sync}^{\mathrm{ACP}}_{X,Y}}{\longmapsto} & \big\{\, (a|b,(x,y)) \mid \\
 & & \qquad (a,x) \in u \,\wedge\, (b,y) \in v \,\wedge\, a|b \text{ is defined} \,\big\}
\end{array}$$

By Theorem 5.2.1, each of these gives (different) $\otimes$ on $\mathbf{Coalg}_F$, and $\|$ on Z; moreover the behavior functor beh satisfies compositionality (5.4). In other words: bisimilarity is a congruence with respect to CSP-, CCS- and ACP-style parallel composition.

5.2.3 Remark. As mentioned in Section 5.1, in some ways the current work can be seen as an extension of the bialgebraic studies started in [134]. However there is also a drawback, namely the limited expressive power of $\mathsf{sync} : FX \otimes FY \to F(X \otimes Y)$.

Our sync specifies the way an algebraic structure interacts with a coalgebraic one. In this sense it is a counterpart of a distributive law $\Sigma F \Rightarrow F\Sigma$ in [134] representing operational rules, where Σ is a functor induced by an algebraic signature. However there are many common operational rules which do not allow representation of the form $\Sigma F \Rightarrow F\Sigma$; therefore in [134] the type of such a distributive law is eventually extended to $\Sigma(F \times \mathrm{id}) \Rightarrow F\Sigma^*$ for enhanced expressive power. The class of rules representable in this extended form coincides with the class of so-called *GSOS-rules*.

At present it is not clear how we can make a similar extension for our sync; consequently there are some operational rules which we cannot model by sync. One important example is an *interleaving* kind of interaction—such as $a.P \parallel Q \xrightarrow{a} P \parallel Q$ which leaves the second component unchanged. This is taken care of in [134] by the identity functor (id) appearing on the left-hand side of $\Sigma(F \times \mathrm{id}) \Rightarrow F\Sigma^*$. For our sync to be able to model such interleaving, we can replace F by the cofree comonad on it, as is done in [70, Example 3.11]. This extension should not be hard but detailed treatment is left as future work.

In $\mathcal{K}\ell(T)$: trace equivalence is a congruence

In Chapter 2 we have shown that trace semantics—including trace *set* semantics for non-deterministic systems and trace *distribution* semantics for probabilistic systems—is also captured by coinduction employed in a Kleisli category $\mathcal{K}\ell(T)$.

Our compositionality result (Theorem 5.2.1) applies to this Kleisli setting all the same: in presence of a suitable tensor $\otimes$ in $\mathcal{K}\ell(T)$ and a synchronization natural transformation, we can compose coalgebras and trace semantics (via coinduction in $\mathcal{K}\ell(T)$) is automatically compositional.

Recall that, in this setting, systems are modeled as $\overline{F}$-coalgebras in $\mathcal{K}\ell(T)$ where the functor $\overline{F} : \mathcal{K}\ell(T) \to \mathcal{K}\ell(T)$ is a lifting of some $F : \mathbf{Sets} \to \mathbf{Sets}$. Following the same spirit, it is possible to reduce the relevant ingredients (namely a tensor $\otimes$ and a synchronization, both in $\mathcal{K}\ell(T)$) to certain constructs in **Sets**. This procedure will be described later in Section 5.5.

5.2.3 Equational properties of parallel composition operators

Now we shall investigate equational properties—associativity, commutativity, and so on—of an outer parallel composition operator $\boldsymbol{\otimes}$ and an inner one $\|$, which we have ignored deliberately for simplicity of arguments. We present our results in terms of associativity; it is straightforward to transfer the results to other properties like commutativity.

First we present a result on an outer composition $\boldsymbol{\otimes}$ (which arises from $\otimes$ and sync; see Section 5.2.1). The main point is as follows: if $\otimes$ is associative and sync is "associative," then the lifting $\boldsymbol{\otimes}$ is associative. The proof is straightforward.

5.2.4 Theorem. *Let $\mathbb{C}$ be a category with a strictly associative tensor $\otimes$,*[6] *and $F : \mathbb{C} \to \mathbb{C}$ be a functor with $\mathsf{sync} : FX \otimes FY \to F(X \otimes Y)$. If the diagram*

$$\begin{tikzcd}
FX \otimes (FY \otimes FZ) \arrow[r, "FX \otimes \mathsf{sync}"] \arrow[d, "\mathrm{id}"] & FX \otimes F(Y \otimes Z) \arrow[r, "\mathsf{sync}"] & F(X \otimes (Y \otimes Z)) \arrow[d, "\mathrm{id}"] \\
(FX \otimes FY) \otimes FZ \arrow[r, "\mathsf{sync} \otimes FZ"'] & F(X \otimes Y) \otimes FZ \arrow[r, "\mathsf{sync}"'] & F((X \otimes Y) \otimes Z)
\end{tikzcd} \tag{5.8}$$

commutes, then the lifted tensor $\boldsymbol{\otimes}$ on $\mathbf{Coalg}_F$ is strictly associative. □

The two (vertical) identity arrows in (5.8) are available due to strict associativity of $\otimes$. Later in Section 5.4 we shall reveal the generic principle behind the commutativity condition of (5.8), namely a coherence condition of a lax natural transformation.

Once we have an associative outer composition $\boldsymbol{\otimes}$, associativity of an inner composition $\|: Z \otimes Z \to Z$ is straightforward by coinduction.

5.2.5 Theorem. *Assume that Theorem 5.2.4 holds and hence we have a strictly associative tensor $\boldsymbol{\otimes}$ on $\mathbf{Coalg}_F$. Then the inner composition operator $\|: Z \otimes Z \to Z$ induced by coinduction (5.6) is associative in the following sense.*

$$\begin{tikzcd}
Z \otimes (Z \otimes Z) \arrow[r, "\mathrm{id}"] \arrow[d, "Z \otimes \|"'] & (Z \otimes Z) \otimes Z \arrow[r, "\| \otimes Z"] & Z \otimes Z \arrow[d, "\|"] \\
Z \otimes Z \arrow[rr, "\|"'] & & Z
\end{tikzcd}$$

The identity arrow in the diagram is due to the strict associativity of $\otimes$. □

As an example, $\mathsf{sync}^{\mathrm{CSP}}$ and $\mathsf{sync}^{\mathrm{CCS}}$ in Section 5.2.2 are easily seen to be "associative" in the sense of the diagram (5.8). Therefore the resulting tensors $\boldsymbol{\otimes}$ are strictly associative; so are the corresponding inner composition operators $\|$.

[6] As mentioned already, in the current work we stick to *strict* algebraic structures.

5.3 Formalizing the microcosm principle

In this section we shall formalize the microcosm principle for an arbitrary algebraic theory presented as a Lawvere theory $\mathbb{L}$. The formalization will be used later in Section 5.4 where we give a general account of algebraic structures on coalgebras.

In our formalization (which we sketched in Section 5.1) an outer model will be a product-preserving functor $\mathbb{C} : \mathbb{L} \to \mathbf{CAT}$; an inner model inside will be a lax natural transformation $X : \mathbf{1} \Rightarrow \mathbb{C}$. Here $\mathbf{1} : \mathbb{L} \to \mathbf{CAT}$ is the constant functor which maps everything to the category $\mathbf{1}$ with one object and one arrow.[7] Mediating 2-cells for the *lax* natural transformation X play a crucial role as inner interpretations of algebraic operations. In this section we heavily rely on 2-categorical notions, of which detailed accounts can be found in [19].

$$\begin{tikzcd}
\mathbb{L} \arrow[r, bend left=50, "\mathbf{1}", ""{name=U, below}] \arrow[r, "\mathbb{C}"', ""{name=D, above}] & \mathbf{CAT} \arrow[from=U, to=D, phantom, "\Downarrow X"]
\end{tikzcd}$$

5.3.1 Lawvere theories

Lawvere theories are categorical presentations of algebraic theories. The notion is introduced in [90] (not under this name, though) aiming at a categorical formulation of "theories" and "semantics." An accessible introduction to the notion can be found in [82]. Lawvere theories are known to be equivalent to *finitary monads.* These two ways of presenting algebraic theories have been widely used in theoretical computer science, e.g. for modeling computation with effect [59, 100]. Recent developments (such as [104]) utilize the increased expressive power of *enriched* Lawvere theories.

In the sequel, by an *FP-category* we refer to a category with (a choice of) finite products. An *FP-functor* is a functor between FP-categories which preserves finite products "on-the-nose," that is, up-to-equality instead of up-to-isomorphism.

5.3.1 Definition (The category **Nat**)**.** By **Nat** we denote the category of natural numbers (as sets) and functions between them. Therefore every arrow in **Nat** is a (cotuple of) coprojection: an arrow $f : k \to n$ can be written as a cotuple $[\kappa_{f(1)}, \ldots, \kappa_{f(k)}]$ where $\kappa_i : 1 \to n$ is the coprojection into the i-th summand of $n = 1 + \cdots + 1$ (n times). Dually, every arrow in $\mathbf{Nat}^{\mathrm{op}}$ is a (tuple of) projection.

5.3.2 Definition (Lawvere theory)**.** A *Lawvere theory* is a small FP-category $\mathbb{L}$ equipped with an FP-functor $H : \mathbf{Nat}^{\mathrm{op}} \to \mathbb{L}$ which is bijective on objects. We shall denote an object of $\mathbb{L}$ by a natural number k, identifying $k \in \mathbf{Nat}^{\mathrm{op}}$ and $Hk \in \mathbb{L}$.

The category $\mathbf{Nat}^{\mathrm{op}}$—which is a free FP-category on the trivial category $\mathbf{1}$—is there in order to specify the choice of finite products in $\mathbb{L}$. For illustration, we make some remarks on $\mathbb{L}$'s objects and arrows.

- An object $k \in \mathbb{L}$ is a k-fold product $1 \times \cdots \times 1$ of $1 \in \mathbb{L}$. Note that $1 \in \mathbb{L}$ is *not* a terminal object; instead $0 \in \mathbb{L}$ is terminal since 0 is terminal in $\mathbf{Nat}^{\mathrm{op}}$.

[7] The functor $\mathbf{1} : \mathbb{L} \to \mathbf{CAT}$ is a special case of an outer model of $\mathbb{L}$.

- An algebraic operation appears in $\mathbb{L}$ in the form of an arrow. That is, a k-ary operation as an arrow $k \to 1$ in $\mathbb{L}$; an arrow $k \to n$ is then understood as an n-tuple $\langle \mathsf{f}_1, \dots, \mathsf{f}_n \rangle$ of k-ary operations. To be precise, arrows in $\mathbb{L}$ also include projections (such as $2 \xrightarrow{\pi_1} 1$) and *terms* made up of operations and projections (such as $3 \xrightarrow{\mathsf{m} \circ \langle \pi_1, \pi_2 \rangle} 1$).

Conventionally in universal algebra, an algebraic theory is presented by an *algebraic specification* (Σ, E)—a pair of a set Σ of operations and a set E of equations. A Lawvere theory $\mathbb{L}$ arises from such (Σ, E) as its so-called *classifying category* (see e.g. [61, 90]). An arrow $k \to n$ in the resulting Lawvere theory $\mathbb{L}$ is an n-tuple $([t_1(\overrightarrow{x})], \dots, [t_n(\overrightarrow{x})])$ of Σ-terms with k variables $\overrightarrow{x}$, where $[\_]$ denotes taking an equivalence class modulo equations in E. An equivalent way to describe this construction is via *sketches*: (Σ, E) is identified with an FP-sketch, which in turn induces $\mathbb{L}$ as a free FP-category. See [11] for details.

5.3.3 Example. Our leading example is the Lawvere theory **Mon** for monoids.[8] It arises as a classifying category from the well-known algebraic specification of monoids. This specification has a nullary operation e and a binary one m; subject to the equations

$$\mathsf{m}(x, \mathsf{e}) = x \ , \quad \mathsf{m}(\mathsf{e}, x) = x \quad \text{and} \quad \mathsf{m}(x, \mathsf{m}(y, z)) = \mathsf{m}(\mathsf{m}(x, y), z) \ .$$

Equivalently, **Mon** is the freely generated FP-category by arrows $0 \xrightarrow{\mathsf{e}} 1$ and $2 \xrightarrow{\mathsf{m}} 1$ subject to the following commutativity.

$$\begin{tikzcd}
1 \arrow[r, "{\langle \mathrm{id}, \mathsf{e} \rangle}"] \arrow[dr, bend right, "\mathrm{id}"'] & 2 \arrow[d, "\mathsf{m}"] & 1 \arrow[l, "{\langle \mathsf{e}, \mathrm{id} \rangle}"'] \arrow[dl, bend left, "\mathrm{id}"] \\
& 1 &
\end{tikzcd}
\qquad
\begin{tikzcd}
3 \arrow[r, "\mathsf{m} \times \mathrm{id}"] \arrow[d, "\mathrm{id} \times \mathsf{m}"'] & 2 \arrow[d, "\mathsf{m}"] \\
2 \arrow[r, "\mathsf{m}"'] & 1
\end{tikzcd}
\tag{5.9}$$

These data (arrows and commutative diagrams) form an FP-sketch (see [11]).

5.3.2 Outer models: $\mathbb{L}$-categories

We start by formalizing an outer model. It is a category with an $\mathbb{L}$-structure, hence we call it an $\mathbb{L}$-*category*.

By the way, it is standard that a (set-theoretic) model of $\mathbb{L}$—a *set* with an $\mathbb{L}$-structure—is identified with an FP-functor $\mathbb{L} \xrightarrow{X} \mathbf{Sets}$. Concretely, let $X = X1$ be the image of $1 \in \mathbb{L}$; this is the carrier set. Then $k \in \mathbb{L}$ must be sent to $X^k = X \times \cdots \times X$ (k times) due to preservation of finite products. Now the functor's action on arrows

[8] The Lawvere theory **Mon** for the theory of monoids should not be confused with the category of (set-theoretic) monoids and monoid homomorphisms (which is often denoted by **Mon** as well).

is what interprets $\mathbb{L}$'s operations in X.

$$\begin{tikzcd}
\mathbb{L} \arrow[r, "X"] & \mathbf{Sets} \\
2 \arrow[d, "\mathsf{m}"] & X^2 \arrow[d, "{[\![\mathsf{m}]\!]}"] \\
1 \arrow[r, phantom, "\longmapsto" near start, shift left=6ex] & X
\end{tikzcd}$$

Equations (expressed as commutative diagrams in $\mathbb{L}$) are satisfied because a functor preserves commutative diagrams.

Turning back to $\mathbb{L}$-*categories*, what we have to do now is to just replace **Sets** by the category **CAT** of (possibly large but locally small) categories.

5.3.4 Definition ($\mathbb{L}$-categories)**.** A (strict) $\mathbb{L}$-*category* is an FP-functor $\mathbb{L} \xrightarrow{\mathbb{C}} \mathbf{CAT}$. In the sequel we denote the image $\mathbb{C}1$ of $1 \in \mathbb{L}$ by $\mathbb{C}$; and the image $\mathbb{C}\mathsf{a}$ of an arrow a by $[\![\mathsf{a}]\!]$.

A morphism of $\mathbb{L}$-categories—formalized as follows—is understood as a functor which preserves an $\mathbb{L}$-structure.

5.3.5 Definition ($\mathbb{L}$-functors)**.** Let $\mathbb{C}$ and $\mathbb{D}$ be $\mathbb{L}$-categories. An $\mathbb{L}$-*functor* $F : \mathbb{C} \to \mathbb{D}$ is a natural transformation

$$\begin{tikzcd}
\mathbb{L} \arrow[r, bend left, "\mathbb{C}"] \arrow[r, bend right, "\mathbb{D}"'] \arrow[r, phantom, "\Downarrow F"] & \mathbf{CAT}
\end{tikzcd} .$$

Indeed, the 1-component of such a natural transformation is a functor $F_1 : \mathbb{C} \to \mathbb{D}$; this is the "underlying" functor.

Another way to look at the previous definitions is to view an $\mathbb{L}$-structure as "factorization through $\mathbf{Nat}^{\mathrm{op}} \to \mathbb{L}$." Specifically, we can identify a category $\mathbb{C} \in \mathbf{CAT}$ with a functor $\mathbf{1} \to \mathbf{CAT}$, which is in turn identified with an FP-functor $\mathbf{Nat}^{\mathrm{op}} \to \mathbf{CAT}$, because $\mathbf{Nat}^{\mathrm{op}}$ is the free FP-category on $\mathbf{1}$.

$$\begin{array}{c}
\mathbb{C}, \quad \text{a category} \\
\hline\hline
\mathbb{C} : \mathbf{1} \longrightarrow \mathbf{CAT}, \quad \text{a functor} \\
\hline\hline
\mathbb{C} : \mathbf{Nat}^{\mathrm{op}} \longrightarrow \mathbf{CAT}, \quad \text{an FP-functor}
\end{array}$$

We say that $\mathbb{C}$ has an $\mathbb{L}$-structure, if the last FP-functor $\mathbb{C} : \mathbf{Nat}^{\mathrm{op}} \to \mathbf{CAT}$ factors through $H : \mathbf{Nat}^{\mathrm{op}} \to \mathbb{L}$.

$$\begin{tikzcd}
\mathbf{Nat}^{\mathrm{op}} \arrow[r, "H"] \arrow[dr, "\mathbb{C}"'] & \mathbb{L} \arrow[d] \\
& \mathbf{CAT}
\end{tikzcd}$$

Note that the factorization is not necessarily unique, because there can be different ways of interpreting the algebraic theory $\mathbb{L}$ in $\mathbb{C}$.

This factorization view applies also to $\mathbb{L}$-functors. A functor $\mathbb{C} \xrightarrow{F} \mathbb{D}$ is identified with a natural transformation $\begin{tikzcd} \mathbf{1} \arrow[r, bend left] \arrow[r, bend right] \arrow[r, phantom, "\Downarrow F"] & \mathbf{CAT} \end{tikzcd}$; and then with $\begin{tikzcd} \mathbf{Nat}^{\mathrm{op}} \arrow[r, bend left] \arrow[r, bend right] \arrow[r, phantom, "\Downarrow F"] & \mathbf{CAT} \end{tikzcd}$ due to the 2-universality of $\mathbf{Nat}^{\mathrm{op}}$ as a free object. We say that this F preserves an $\mathbb{L}$-structure, if the natural transformation $\begin{tikzcd}\mathbf{Nat}^{\mathrm{op}} \arrow[r, bend left] \arrow[r, bend right] \arrow[r, phantom, "\Downarrow F"] & \mathbf{CAT}\end{tikzcd}$ factors through $H : \mathbf{Nat}^{\mathrm{op}} \to \mathbb{L}$.

$$\begin{tikzcd}
\mathbf{Nat}^{\mathrm{op}} \arrow[r, "H"] \arrow[dr] \arrow[dr, bend right] \arrow[dr, phantom, "\swarrow F"] & \mathbb{L} \arrow[d, bend right] \arrow[d, bend left] \arrow[d, phantom, "\Leftarrow"] \\
& \mathbf{CAT}
\end{tikzcd}$$

5.3.6 Example. The usual notion of strictly monoidal categories coincides with $\mathbb{L}$-categories for $\mathbb{L} = \mathbf{Mon}$. A tensor $\otimes$ and a unit I on a category arise as interpretation of the operations $2 \xrightarrow{\mathsf{m}} 1$ and $0 \xrightarrow{\mathsf{e}} 1$; commuting diagrams in $\mathbf{Mon}$ such as $\mathsf{m} \circ \langle \mathrm{id}, \mathsf{e} \rangle = \mathrm{id}$ yield equational properties of $\otimes$ and I.

5.3.3 Remarks on "pseudo" algebraic structures

As we mentioned already, in the current work we focus on *strict* algebraic structures. This means that monoidal categories (in which associativity holds only up-to-isomorphism, for example) fall out of our consideration. Extending our current framework to such "pseudo" algebraic structures is one important direction of our future work. Such an extension is not entirely obvious; we shall sketch some preliminary observations in this direction.

The starting point is to relax the definition of $\mathbb{L}$-categories from (strict) functors $\mathbb{L} \to \mathbf{CAT}$ to *pseudo* functors, meaning that composition and identities are preserved only up-to-isomorphism. Then it is not hard to see that a pseudo functor $\mathbf{Mon} \xrightarrow{\mathbb{C}} \mathbf{CAT}$ (which preserves finite products in a suitable sense) gives rise to a monoidal category.

Let us take a detailed look. A pseudo functor $\mathbf{Mon} \xrightarrow{\mathbb{C}} \mathbf{CAT}$ preserves composition up-to-isomorphism; let us denote the mediating natural isomorphism by $\mathbb{C}_{\mathsf{b},\mathsf{a}}$ as below.

$$\begin{tikzcd}
\underline{\text{in } \mathbf{Mon}} & j \arrow[d, "\mathsf{a}"] \\
& k \arrow[d, "\mathsf{b}"] \\
& n
\end{tikzcd}
\qquad
\begin{tikzcd}
\underline{\text{in } \mathbf{CAT}} & & \mathbb{C}^j \arrow[dl, "{[\![\mathsf{a}]\!]}"'] \arrow[dd, bend left=60, "{[\![\mathsf{b}\circ\mathsf{a}]\!]}"] \\
& \mathbb{C}^k \arrow[dr, "{[\![\mathsf{b}]\!]}"'] \arrow[r, phantom, "\overset{\mathbb{C}_{\mathsf{b},\mathsf{a}}}{\underset{\cong}{\Longrightarrow}}"] & \\
& & \mathbb{C}^n
\end{tikzcd}
\tag{5.10}$$

Then a commuting diagram in $\mathbf{Mon}$ (below left) gives rise to the two iso-2-cells on the right.

$$\begin{tikzcd}
\underline{\text{in } \mathbf{Mon}} & & 3 \arrow[dl, "\mathrm{id}\times\mathsf{m}"'] \arrow[dr, "\mathsf{m}\times\mathrm{id}"] & \\
& 2 \arrow[dr, "\mathsf{m}"'] & & 2 \arrow[dl, "\mathsf{m}"] \\
& & 1 &
\end{tikzcd}
\qquad
\begin{tikzcd}
\underline{\text{in } \mathbf{CAT}} & & \mathbb{C}^3 \arrow[dl, bend right, "{[\![\mathrm{id}\times\mathsf{m}]\!]}"'] \arrow[dr, bend left, "{[\![\mathsf{m}\times\mathrm{id}]\!]}"] \arrow[dd, "{[\![\mathsf{m}\circ(\mathsf{m}\times\mathrm{id})]\!] = [\![\mathsf{m}\circ(\mathrm{id}\times\mathsf{m})]\!]}" description] & \\
& \mathbb{C}^2 \arrow[dr, bend right, "{[\![\mathsf{m}]\!]}"'] \arrow[r, phantom, "\overset{\mathbb{C}_{\mathsf{m},\mathrm{id}\times\mathsf{m}}}{\Rightarrow}"] & & \mathbb{C}^2 \arrow[dl, bend left, "{[\![\mathsf{m}]\!]}"] \arrow[l, phantom, "\overset{\mathbb{C}_{\mathsf{m},\mathsf{m}\times\mathrm{id}}}{\Leftarrow}"] \\
& & \mathbb{C} &
\end{tikzcd}
\tag{5.11}$$

The composite $[\![\mathsf{m}]\!] \circ [\![\mathrm{id} \times \mathsf{m}]\!]$ on the left edge is a functor such that

$$(X_1, X_2, X_3) \quad \longmapsto \quad X_1 \otimes (X_2 \otimes X_3) \ ;$$

the one $\llbracket \mathsf{m} \rrbracket \circ \llbracket \mathsf{m} \times \mathrm{id} \rrbracket$ on the right edge is such that

$$(X_1, X_2, X_3) \quad \longmapsto \quad (X_1 \otimes X_2) \otimes X_3 \ .$$

Now the composite $\mathbb{C}^{-1}_{\mathsf{m},\mathsf{m}\times\mathrm{id}} \bullet \mathbb{C}_{\mathsf{m},\mathrm{id}\times\mathsf{m}}$ of the iso-2-cells is what gives us a natural isomorphism $\alpha : X_1 \otimes (X_2 \otimes X_3) \overset{\cong}{\Rightarrow} (X_1 \otimes X_2) \otimes X_3$.

Moreover, the coherence condition on such isomorphisms in a monoidal category (see [14, 96]) requires that "any two compositions of such natural isomorphisms are identical." A well-known instance is commutativity of the following "pentagon diagram."

$$\begin{tikzcd}
X_1 \otimes (X_2 \otimes (X_3 \otimes X_4)) \arrow[r, "\alpha"] \arrow[d, "X_1 \otimes \alpha"'] & (X_1 \otimes X_2) \otimes (X_3 \otimes X_4) \arrow[dd, "\alpha"'] \\
X_1 \otimes ((X_2 \otimes X_3) \otimes X_4) \arrow[d, "\alpha"'] & \\
(X_1 \otimes (X_2 \otimes X_3)) \otimes X_4 \arrow[r, "\alpha \otimes X_4"'] & ((X_1 \otimes X_2) \otimes X_3) \otimes X_4
\end{tikzcd}$$

Indeed, commutativity of this pentagon is derived from the fact that the following two composed iso-2-cells are identical.

$$\begin{tikzcd}
\mathbb{C}^4 \arrow[rr, "\llbracket 1\cdot 2,3,4 \rrbracket"] \arrow[dr, "\llbracket 1,2\cdot 3,4 \rrbracket"] \arrow[dd, "\llbracket 1,2,3\cdot 4 \rrbracket"'] & \arrow[d, phantom, "\alpha\times\mathbb{C}\ \Swarrow"] & \mathbb{C}^3 \arrow[dr, "\llbracket 1\cdot 2,3 \rrbracket"] & \\
\arrow[r, phantom, "\mathbb{C}\times\alpha\ \Swarrow"] & \mathbb{C}^3 \arrow[rr, "\llbracket 1\cdot 2,3 \rrbracket"] \arrow[dd, "\llbracket 1,2\cdot 3 \rrbracket"] & & \mathbb{C}^2 \arrow[dd, "\llbracket 1\cdot 2 \rrbracket"] \\
\mathbb{C}^3 \arrow[dr, "\llbracket 1,2\cdot 3 \rrbracket"'] & & \arrow[r, phantom, "\Swarrow_{\alpha}"] & \\
& \mathbb{C}^2 \arrow[rr, "\llbracket 1\cdot 2 \rrbracket"'] & & \mathbb{C}
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathbb{C}^4 \arrow[rr, "\llbracket 1\cdot 2,3,4 \rrbracket"] \arrow[dd, "\llbracket 1,2,3\cdot 4 \rrbracket"'] & & \mathbb{C}^3 \arrow[dr, "\llbracket 1\cdot 2,3 \rrbracket"] \arrow[dd, "\llbracket 1,2\cdot 3 \rrbracket"] & \\
& \arrow[r, phantom, "\mathrm{id}\ \Swarrow"] & \arrow[r, phantom, "\alpha\ \Swarrow"] & \mathbb{C}^2 \arrow[dd, "\llbracket 1\cdot 2 \rrbracket"] \\
\mathbb{C}^3 \arrow[rr, "\llbracket 1\cdot 2,3 \rrbracket"] \arrow[dr, "\llbracket 1,2\cdot 3 \rrbracket"'] & \arrow[d, phantom, "\alpha\ \Swarrow"] & \mathbb{C}^2 \arrow[dr, "\llbracket 1\cdot 2 \rrbracket"'] & \\
& \mathbb{C}^2 \arrow[rr, "\llbracket 1\cdot 2 \rrbracket"'] & & \mathbb{C}
\end{tikzcd}$$

These two composites are identical due to the coherence condition on the mediating 2-cells $\mathbb{C}_{\mathsf{b},\mathsf{a}}$ of a pseudo functor (see [19]). Here the notation $1 \cdot 2, 3, 4$—with its intention being $x_1, x_2, x_3, x_4 \vdash x_1 \cdot x_2, x_3, x_4$ following the categorical logic tradition—denotes the arrow $\langle \mathsf{m} \circ \langle \pi_1, \pi_2 \rangle, \pi_3, \pi_4 \rangle : 4 \to 3$ in **Mon**.

So far so good. However, at this moment it is not clear what is a canonical construction the other way round, i.e. from a monoidal category to a pseudo functor.[9] In the present paper we side-step these 2-categorical subtleties by restricting ourselves to strict, non-pseudo algebraic structures.

5.3.4 Inner models: $\mathbb{L}$-objects

We proceed to formalize an inner model. It is an object in an $\mathbb{L}$-category which itself carries an (inner) $\mathbb{L}$-structure, hence is called an $\mathbb{L}$-*object*. A monoid object in a

[9] For example, given a monoidal category $\mathbb{C}$, we need to define a functor $\llbracket \mathsf{m} \circ (\mathsf{m} \times \mathrm{id}) \rrbracket = \llbracket \mathsf{m} \circ (\mathrm{id} \times \mathsf{m}) \rrbracket$ in (5.11). It's not clear whether it should carry (X, Y, Z) to $X \otimes (Y \otimes Z)$, or to $(X \otimes Y) \otimes Z$.

monoidal category is a prototypical example. We first present an abstract definition; some illustration follows afterwards.

5.3.7 Definition ($\mathbb{L}$-objects)**.** An $\mathbb{L}$-*object* X in an $\mathbb{L}$-category $\mathbb{C}$ is a lax natural transformation $X : \mathbf{1} \Rightarrow \mathbb{C}$ (below left) which is "product-preserving": this means that the composition $X \circ H$ (below right) is strictly, non-lax natural. Here $\mathbf{1} : \mathbb{L} \to \mathbf{CAT}$ denotes the constant functor to the trivial one-object category $\mathbf{1}$.

$$\begin{tikzcd}
\mathbb{L} \arrow[r, bend left=60, "\mathbf{1}"] \arrow[r, "\mathbb{C}"'] \arrow[r, phantom, shift left=2, "\Downarrow X"] & \mathbf{CAT}
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathbf{Nat}^{\mathrm{op}} \arrow[r, "H"] & \mathbb{L} \arrow[r, bend left=60, "\mathbf{1}"] \arrow[r, "\mathbb{C}"'] \arrow[r, phantom, shift left=2, "\Downarrow X"] & \mathbf{CAT}
\end{tikzcd}$$

Such a nested algebraic structure—formalized as an $\mathbb{L}$-object in an $\mathbb{L}$-category—shall be called a *microcosm model* for $\mathbb{L}$.

Let us now illustrate the definition. First, X's component at $1 \in \mathbb{L}$ is a functor $\mathbf{1} \xrightarrow{X_1} \mathbb{C}$, which is identified with an object $X \in \mathbb{C}$. This is the carrier object of this inner algebra. Moreover, any component $\mathbf{1} \xrightarrow{X_k} \mathbb{C}^k$ must be the k-tuple $(X, \dots, X) \in \mathbb{C}^k$ of X's. This is because X is "product-preserving": for any $i \in [1, k]$ we have the following naturality diagram of $X \circ H$ which requires the composite $\pi_i \circ X_k$ to be $X_1 = X$.

$$\begin{tikzcd}
\underline{\text{in } \mathbf{Nat}^{\mathrm{op}}} & k \arrow[d, "\pi_i"] & \underline{\text{in } \mathbf{CAT}} & \mathbf{1} \arrow[r, "X_k"] \arrow[d, equal] \arrow[dr, phantom, "//"] & \mathbb{C}^k \arrow[d, "{[\![H\pi_i]\!]=\pi_i}"] \\
& 1 & & \mathbf{1} \arrow[r, "X_1=X"'] & \mathbb{C}
\end{tikzcd}$$

The (inner) algebraic structure on the object X arises in the form of mediating 2-cells of the *lax* natural transformation X. For each arrow $k \xrightarrow{\mathsf{a}} n$ in $\mathbb{L}$, lax naturality of X requires existence of a mediating 2-cell $X_{\mathsf{a}} : [\![\mathsf{a}]\!] \circ X_k \Rightarrow X_n$. The following diagram shows the situation when we set $\mathsf{a} = \mathsf{m}$, a binary operation.

$$\begin{tikzcd}
\underline{\text{in } \mathbb{L}} & 2 \arrow[d, "\mathsf{m}"] & \underline{\text{in } \mathbf{CAT}} & \mathbf{1} \arrow[r, "{X_2=(X,X)}"] \arrow[d, equal] \arrow[dr, phantom, "\swarrow X_{\mathsf{m}}"] & \mathbb{C}^2 \arrow[d, "{[\![\mathsf{m}]\!]=\otimes}"] \\
& 1 & & \mathbf{1} \arrow[r, "X"'] & \mathbb{C}
\end{tikzcd} \tag{5.12}$$

The natural transformation X_{m} can be identified with an arrow $X \otimes X \xrightarrow{\mu} X$ in $\mathbb{C}$, which gives an inner binary operation on the carrier X.

How do such inner operations on X satisfy equations as specified in $\mathbb{L}$? The key is the coherence condition[10] on mediating 2-cells: it requires $X_{\mathrm{id}} = \mathrm{id}$ concerning identities; and $X_{\mathsf{b} \circ \mathsf{a}} = X_{\mathsf{b}} \bullet ([\![\mathsf{b}]\!] \circ X_{\mathsf{a}})$ concerning composition (as in the following

[10] This is part of the notion of lax natural transformations; see [19].

diagram).

$$\begin{tikzcd}
\mathbf{1} \arrow[r] \arrow[dd, equal] & \mathbb{C}^l \arrow[dd, "{[\![\mathsf{b}\circ\mathsf{a}]\!]}"] \arrow[ddl, phantom, "\swarrow X_{\mathsf{b}\circ\mathsf{a}}"] \\
 & \\
\mathbf{1} \arrow[r] & \mathbb{C}^n
\end{tikzcd}
\quad = \quad
\begin{tikzcd}
\mathbf{1} \arrow[r] \arrow[d, equal] & \mathbb{C}^l \arrow[d, "{[\![\mathsf{a}]\!]}"] \arrow[dl, phantom, "\swarrow X_{\mathsf{a}}"] \\
\mathbf{1} \arrow[r] \arrow[d, equal] & \mathbb{C}^k \arrow[d, "{[\![\mathsf{b}]\!]}"] \arrow[dl, phantom, "\swarrow X_{\mathsf{b}}"] \\
\mathbf{1} \arrow[r] & \mathbb{C}^n
\end{tikzcd}
\tag{5.13}$$

The following example illustrates how such coherence induces equational properties.

5.3.8 Example. A monoid object in a strictly monoidal category is an example of an $\mathbb{L}$-object in an $\mathbb{L}$-category. Here we take $\mathbb{L} = \mathbf{Mon}$, the theory of monoids.

For illustration, let us here derive associativity of (inner) multiplication $X \otimes X \xrightarrow{\mu} X$. In the current setting a tensor $\otimes$ is identified with $[\![\mathsf{m}]\!] = \mathbb{C}\mathsf{m} : \mathbb{C}^2 \to \mathbb{C}$; the inner multiplication μ is identified with a mediating 2-cell X_{m} as in (5.12) above. Now the coherence condition (5.13) yields the two equalities $(*)$ below.

<u>in **Mon**</u> <u>in **CAT**</u>

$$\begin{tikzcd}
 & 3 \arrow[dl, "\mathsf{id}\times\mathsf{m}"'] \arrow[dr, "\mathsf{m}\times\mathsf{id}"] & \\
2 \arrow[dr, "\mathsf{m}"'] & & 2 \arrow[dl, "\mathsf{m}"] \\
 & 1 &
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathbf{1} \arrow[r] \arrow[d, equal] & \mathbb{C}^3 \arrow[d, "{[\![\mathsf{id}\times\mathsf{m}]\!]}"] \arrow[dl, phantom, "\swarrow X_{\mathsf{id}\times\mathsf{m}}"] \\
\mathbf{1} \arrow[r] \arrow[d, equal] & \mathbb{C}^2 \arrow[d, "{[\![\mathsf{m}]\!]}"] \arrow[dl, phantom, "\swarrow X_{\mathsf{m}}"] \\
\mathbf{1} \arrow[r] & \mathbb{C}
\end{tikzcd}
\overset{(*)}{=}
\begin{tikzcd}
\mathbf{1} \arrow[r] \arrow[dd, equal] & \mathbb{C}^3 \arrow[dd] \arrow[ddl, phantom, "{\swarrow X_{\mathsf{m}\circ(\mathsf{id}\times\mathsf{m})} = X_{\mathsf{m}\circ(\mathsf{m}\times\mathsf{id})}}"] \\
 & \\
\mathbf{1} \arrow[r] & \mathbb{C}^1
\end{tikzcd}
\overset{(*)}{=}
\begin{tikzcd}
\mathbf{1} \arrow[r] \arrow[d, equal] & \mathbb{C}^3 \arrow[d, "{[\![\mathsf{m}\times\mathsf{id}]\!]}"] \arrow[dl, phantom, "\swarrow X_{\mathsf{m}\times\mathsf{id}}"] \\
\mathbf{1} \arrow[r] \arrow[d, equal] & \mathbb{C}^2 \arrow[d, "{[\![\mathsf{m}]\!]}"] \arrow[dl, phantom, "\swarrow X_{\mathsf{m}}"] \\
\mathbf{1} \arrow[r] & \mathbb{C}
\end{tikzcd}
\tag{5.14}$$

Now it is not hard to see that: the composed 2-cell on the left

$$\begin{tikzcd}
\mathbf{1} \arrow[r] \arrow[d, equal] & \mathbb{C}^3 \arrow[d, "{[\![\mathsf{id}\times\mathsf{m}]\!]}"] \arrow[dl, phantom, "\swarrow X_{\mathsf{id}\times\mathsf{m}}"] \\
\mathbf{1} \arrow[r] \arrow[d, equal] & \mathbb{C}^2 \arrow[d, "{[\![\mathsf{m}]\!]}"] \arrow[dl, phantom, "\swarrow X_{\mathsf{m}}"] \\
\mathbf{1} \arrow[r] & \mathbb{C}
\end{tikzcd}
\quad \text{corresponds to} \quad X \otimes X \otimes X \xrightarrow{X\otimes\mu} X \otimes X \xrightarrow{\mu} X \;;
\tag{5.15}$$

and the one on the right

$$\begin{tikzcd}
\mathbf{1} \arrow[r] \arrow[d, equal] & \mathbb{C}^3 \arrow[d, "{[\![\mathsf{m}\times\mathsf{id}]\!]}"] \arrow[dl, phantom, "\swarrow X_{\mathsf{m}\times\mathsf{id}}"] \\
\mathbf{1} \arrow[r] \arrow[d, equal] & \mathbb{C}^2 \arrow[d, "{[\![\mathsf{m}]\!]}"] \arrow[dl, phantom, "\swarrow X_{\mathsf{m}}"] \\
\mathbf{1} \arrow[r] & \mathbb{C}
\end{tikzcd}
\quad \text{corresponds to} \quad X \otimes X \otimes X \xrightarrow{\mu\otimes X} X \otimes X \xrightarrow{\mu} X \;.$$

Therefore the equalities in (5.14) prove associativity of the inner multiplication $\mu : X \otimes X \to X$.

$$\begin{tikzcd}
X \otimes X \otimes X \arrow[r, "\mu \otimes X"] \arrow[d, "X \otimes \mu"'] & X \otimes X \arrow[d, "\mu"] \\
X \otimes X \arrow[r, "\mu"'] & X
\end{tikzcd}$$

For further illustration, let us elaborate the correspondence (5.15). The only nontrivial step in the correspondence is showing

$$X_{\mathrm{id}\times\mathsf{m}} = \langle \mathrm{id}_X, X_{\mathsf{m}}\rangle \ , \tag{5.16}$$

that is, the (upper) 2-cell $X_{\mathrm{id}\times\mathsf{m}}$ in (5.15) is identical to the tuple $\langle \mathrm{id}_X, X_{\mathsf{m}}\rangle$ (depicted below) which obviously corresponds to the arrow $X\otimes X\otimes X \xrightarrow{X\times\mu} X\otimes X$ in $\mathbb{C}$.

$$\begin{tikzcd}
 & \mathbb{C}^3 \arrow[dr, bend left, "{[\![\mathrm{id}\times\mathsf{m}]\!]=\mathrm{id}_{\mathbb{C}}\times\otimes}"] & \\
\mathbf{1} \arrow[ur, bend left, "{(X,X,X)}"] \arrow[rr, bend right, "{(X,X)}"'] \arrow[rr, phantom, "\Downarrow\langle\mathrm{id}_X{,}X_{\mathsf{m}}\rangle"] & & \mathbb{C}^2
\end{tikzcd}
\quad \text{that is,} \quad
\left\langle
\begin{tikzcd}
\mathbf{1} \arrow[r, bend left, "X"] \arrow[r, bend right, "X"'] \arrow[r, phantom, "\Downarrow\mathrm{id}_X"] & \mathbb{C}
\end{tikzcd}
\ ,\
\begin{tikzcd}
\mathbf{1} \arrow[r, bend left, "X\otimes X"] \arrow[r, bend right, "X"'] \arrow[r, phantom, "\Downarrow X_{\mathsf{m}}"] & \mathbb{C}
\end{tikzcd}
\right\rangle$$

The equality $X_{\mathrm{id}\times\mathsf{m}} = \langle \mathrm{id}_X, X_{\mathsf{m}}\rangle$ (5.16) holds essentially because the lax natural transformation X is "product-preserving" (Definition 5.3.7). Product-preservation in this context means that a mediating 2-cell X_{π_i} for a projection π_i is an identity. Indeed, the following calculation shows that $\pi_2\circ X_{\mathrm{id}\times\mathsf{m}} = X_{\mathsf{m}}$.

$$\begin{tikzcd}
\mathbf{1} \arrow[r] \arrow[d, equal] \arrow[dr, phantom, "\swarrow X_{\mathrm{id}\times\mathsf{m}}"] & \mathbb{C}^3 \arrow[d, "{[\![\mathrm{id}\times\mathsf{m}]\!]}"] \\
\mathbf{1} \arrow[r] & \mathbb{C}^2 \arrow[d, "{[\![\pi_2]\!]=\pi_2}"] \\
 & \mathbb{C}
\end{tikzcd}
=
\begin{tikzcd}
\mathbf{1} \arrow[r, "{(X,X,X)}"] \arrow[d, equal] \arrow[dr, phantom, "\swarrow X_{\mathrm{id}\times\mathsf{m}}"] & \mathbb{C}^3 \arrow[d, "{[\![\mathrm{id}\times\mathsf{m}]\!]}"] \\
\mathbf{1} \arrow[r, "{(X,X)}"] \arrow[d, equal] \arrow[dr, phantom, "\swarrow \mathrm{id}"] & \mathbb{C}^2 \arrow[d, "{[\![\pi_2]\!]=\pi_2}"] \\
\mathbf{1} \arrow[r, "X"'] & \mathbb{C}
\end{tikzcd}
\overset{(*)}{=}
\begin{tikzcd}
\mathbf{1} \arrow[r, "{(X,X,X)}"] \arrow[d, equal] \arrow[dr, phantom, "\swarrow X_{\mathrm{id}\times\mathsf{m}}"] & \mathbb{C}^3 \arrow[d, "{[\![\mathrm{id}\times\mathsf{m}]\!]}"] \\
\mathbf{1} \arrow[r, "{(X,X)}"] \arrow[d, equal] \arrow[dr, phantom, "\swarrow X_{\pi_2}"] & \mathbb{C}^2 \arrow[d, "{[\![\pi_2]\!]=\pi_2}"] \\
\mathbf{1} \arrow[r, "X"'] & \mathbb{C}
\end{tikzcd}$$

$$\overset{(\dagger)}{=}
\begin{tikzcd}
\mathbf{1} \arrow[r] \arrow[d, equal] \arrow[dr, phantom, "{\swarrow X_{\pi_2\circ(\mathrm{id}\times\mathsf{m})}=X_{\mathsf{m}\circ\pi_2}}"] & \mathbb{C}^3 \arrow[d] \\
\mathbf{1} \arrow[r] & \mathbb{C}
\end{tikzcd}
\overset{(\dagger)}{=}
\begin{tikzcd}
\mathbf{1} \arrow[r, "{(X,X,X)}"] \arrow[d, equal] \arrow[dr, phantom, "\swarrow X_{\pi_2}"] & \mathbb{C}^3 \arrow[d, "{[\![\pi_2]\!]=\pi_2}"] \\
\mathbf{1} \arrow[r, "{(X,X)}"] \arrow[d, equal] \arrow[dr, phantom, "\swarrow X_{\mathsf{m}}"] & \mathbb{C}^2 \arrow[d, "{[\![\mathsf{m}]\!]}"] \\
\mathbf{1} \arrow[r, "X"'] & \mathbb{C}
\end{tikzcd}
\overset{(*)}{=}
\begin{tikzcd}
\mathbf{1} \arrow[r, "{(X,X,X)}"] \arrow[d, equal] \arrow[dr, phantom, "\swarrow \mathrm{id}"] & \mathbb{C}^3 \arrow[d, "{[\![\pi_2]\!]=\pi_2}"] \\
\mathbf{1} \arrow[r, "{(X,X)}"] \arrow[d, equal] \arrow[dr, phantom, "\swarrow X_{\mathsf{m}}"] & \mathbb{C}^2 \arrow[d, "{[\![\mathsf{m}]\!]}"] \\
\mathbf{1} \arrow[r, "X"'] & \mathbb{C}
\end{tikzcd}
= X_{\mathsf{m}}$$

Here $(*)$ holds since $X_{\pi_i} = \mathrm{id}$; $(\dagger)$ refers to the coherence condition (5.13) on mediating 2-cells on X. Similar calculation shows $\pi_1\circ(X_{\mathrm{id}\times\mathsf{m}}) = \mathrm{id}_X$; this proves the equality $X_{\mathrm{id}\times\mathsf{m}} = \langle \mathrm{id}_X, X_{\mathsf{m}}\rangle$ in (5.16).

5.3.5 Basic observations on microcosm models

We shall establish some basic facts about microcosm models. Some of them will be used later in our main result of general compositionality (but many others will not).

Lifting outer $\mathbb{L}$-structures

Let $\mathbb{C}$ be an $\mathbb{L}$-category, and $F : \mathbb{C}\to\mathbb{C}$ be a functor. Later in Section 5.4 we want to lift the $\mathbb{L}$-structure on $\mathbb{C}$ to the one on the category $\mathbf{Coalg}_F$ of F-coalgebras. We can imagine that, for us to be able to do so, the functor F needs to be somehow compatible with $\mathbb{L}$. It turns out that F's being a *lax* $\mathbb{L}$*-functor* is sufficient. It is weaker than F's being an $\mathbb{L}$-functor (Definition 5.3.5).

5.3.9 Definition (Lax $\mathbb{L}$-functors)**.** A functor $F : \mathbb{C} \to \mathbb{D}$ between $\mathbb{L}$-categories is said to be a *lax* $\mathbb{L}$*-functor* if it is identified with[11] some lax natural transformation $\begin{tikzcd} \mathbb{L} \arrow[r, bend left, "\mathbb{C}"{name=U}] \arrow[r, bend right, "\mathbb{D}"'{name=D}] \arrow[Rightarrow, from=U, to=D, "F"] & \mathbf{CAT} \end{tikzcd}$ which is product-preserving (meaning $F \circ H$ is strictly natural; see Definition 5.3.7).

Lax naturality means that, for each arrow $\mathsf{a} : k \to n$ in $\mathbb{L}$, we have the following mediating 2-cell. We shall denote it by F_{a}.

$$\begin{tikzcd} \underline{\text{in } \mathbb{L}} & k \arrow[d, "\mathsf{a}"] & \underline{\text{in } \mathbf{CAT}} & \mathbb{C}^k \arrow[r, "F_k = F^k"] \arrow[d, "{[\![\mathsf{a}]\!]}"'] \arrow[dr, phantom, "\swarrow F_{\mathsf{a}}"] & \mathbb{D}^k \arrow[d, "{[\![\mathsf{a}]\!]}"] \\ & n & & \mathbb{C}^n \arrow[r, "F_n = F^n"'] & \mathbb{D}^n \end{tikzcd} \tag{5.17}$$

Here the k-component F_k of the lax natural transformation F is identical to

$$F^k \;:\; \mathbb{C}^k \longrightarrow \mathbb{C}^k \;, \qquad (X_1, \dots, X_k) \longmapsto (FX_1, \dots, FX_k) \;, \tag{5.18}$$

because F is "product-preserving": for each $i \in [1, k]$ we have the following (non-lax) naturality diagram.

$$\begin{tikzcd} \underline{\text{in } \mathbf{Nat}^{\mathrm{op}}} & k \arrow[d, "\pi_i"] & \underline{\text{in } \mathbf{CAT}} & \mathbb{C}^k \arrow[r, "F_k"] \arrow[d, "{[\![\pi_i]\!] = \pi_i}"'] \arrow[dr, phantom, "/\!/"] & \mathbb{D}^k \arrow[d, "{[\![\pi_i]\!] = \pi_i}"] \\ & 1 & & \mathbb{C} \arrow[r, "F_1 = F"'] & \mathbb{D} \end{tikzcd}$$

Dually, an *oplax* $\mathbb{L}$*-functor* is a functor $F : \mathbb{C} \to \mathbb{D}$ which can be identified with an *oplax* natural transformation $\begin{tikzcd} \mathbb{L} \arrow[r, bend left, "\mathbb{C}"{name=U}] \arrow[r, bend right, "\mathbb{D}"'{name=D}] \arrow[Rightarrow, from=U, to=D, "F"] & \mathbf{CAT} \end{tikzcd}$ which is "product-preserving." Its mediating 2-cells—as in (5.17), but in the opposite direction—shall be denoted by F_{a} as well.

5.3.10 Proposition. *1. Let $\mathbb{C}$ be an $\mathbb{L}$-category and $F : \mathbb{C} \to \mathbb{C}$ be a lax $\mathbb{L}$-functor. Then $\mathbf{Coalg}_F$ is an $\mathbb{L}$-category; moreover the forgetful functor $\mathbf{Coalg}_F \xrightarrow{U} \mathbb{C}$ is a (strict, non-lax) $\mathbb{L}$-functor.*

2. Dually, if $F : \mathbb{C} \to \mathbb{C}$ is an oplax $\mathbb{L}$-functor, then the category $\mathbf{Alg}_F$ of F-algebras is an $\mathbb{L}$-category. Moreover the forgetful functor $\mathbf{Alg}_F \to \mathbb{C}$ is an $\mathbb{L}$-functor.

3. Given a microcosm model $X \in \mathbb{C}$ for $\mathbb{L}$, the slice category $\mathbb{C}/X$ is an $\mathbb{L}$-category; moreover the functor $\mathbb{C}/X \xrightarrow{\mathrm{dom}} \mathbb{C}$ is an $\mathbb{L}$-functor. □

These results can be derived from a more general result (Lemma 5.3.12) concerning *inserters*. The notion of inserters can be defined in any 2-category; here for simplicity we focus on inserters in the specific 2-category $\mathbf{CAT}$.

[11] Meaning: $F : \mathbb{C} \to \mathbb{D}$ is the 1-component of such a lax natural transformation $\mathbb{C} \Rightarrow \mathbb{D}$.

5.3.11 Definition (Inserters)**.** Let $F, G : \mathbb{C} \rightrightarrows \mathbb{D}$ be two functors with the same domain and codomain. An *inserter* $\mathit{Ins}(F, G)$ for F and G consists of

- a category $\mathit{Ins}(F, G)$;
- a functor $\mathit{Ins}(F, G) \xrightarrow{R} \mathbb{C}$; and
- a natural transformation $\rho : FR \Rightarrow GR$ (below left) which is universal: each triple $(\mathbb{B}, B, \beta)$ such as below right

$$\begin{tikzcd} & \mathbb{C} \arrow[dr, "F"] & \\ \mathit{Ins}(F,G) \arrow[ur, "R"] \arrow[dr, "R"'] & \Swarrow \rho & \mathbb{D} \\ & \mathbb{C} \arrow[ur, "G"'] & \end{tikzcd} \qquad \begin{tikzcd} & \mathbb{C} \arrow[dr, "F"] & \\ \mathbb{B} \arrow[ur, "B"] \arrow[dr, "B"'] & \Swarrow \beta & \mathbb{D} \\ & \mathbb{C} \arrow[ur, "G"'] & \end{tikzcd}$$

induces a unique mediating functor $\overline{B} : \mathbb{B} \to \mathit{Ins}(F, G)$ such that $B = R \circ \overline{B}$ and $\beta = \rho \circ \overline{B}$.

$$\begin{tikzcd} \mathbb{B} \arrow[rr, "B"] \arrow[rrdd, "B" near end] \arrow[dd, dashed, "\overline{B}"'] & \Downarrow \beta & \mathbb{C} \arrow[dr, "F"] & \\ & & & \mathbb{D} \\ \mathit{Ins}(F,G) \arrow[rr, "R"'] \arrow[rruu, "R" near end] & \Downarrow \rho & \mathbb{C} \arrow[ur, "G"'] & \end{tikzcd}$$

By the universality it is obvious that an inserter, if it exists, is unique up-to-isomorphism.

This 2-categorical notion of inserters comes from [74, 131]; see also [54]. Examples include slice categories, categories of algebras and of coalgebras:

$$\mathbb{C}/X = \mathit{Ins}(\,\mathbb{C} \xrightarrow{\mathrm{id}} \mathbb{C}\,,\ \mathbb{C} \xrightarrow{!} \mathbf{1} \xrightarrow{X} \mathbb{C}\,)\ ,$$

$$\mathbf{Alg}_F = \mathit{Ins}(\,\mathbb{C} \xrightarrow{F} \mathbb{C}\,,\ \mathbb{C} \xrightarrow{\mathrm{id}} \mathbb{C}\,)\ , \qquad \mathbf{Coalg}_F = \mathit{Ins}(\,\mathbb{C} \xrightarrow{\mathrm{id}} \mathbb{C}\,,\ \mathbb{C} \xrightarrow{F} \mathbb{C}\,)\ .$$

These description of the three categories as inserters allows us to instantiate the following lemma to Proposition 5.3.10.

5.3.12 Lemma. *Let $\mathbb{C}$ and $\mathbb{D}$ be $\mathbb{L}$-categories, and $F, G : \mathbb{C} \rightrightarrows \mathbb{D}$ be two functors. If F is an oplax $\mathbb{L}$-functor and G is a lax $\mathbb{L}$-functor, then the inserter $\mathit{Ins}(F, G)$ is an $\mathbb{L}$-category. Moreover the functor $\mathit{Ins}(F, G) \xrightarrow{R} \mathbb{C}$ is an $\mathbb{L}$-functor; in this sense the $\mathbb{L}$-structure on $\mathit{Ins}(F, G)$ is a lifting of the one on $\mathbb{C}$.*

Proof. We have to define, for each arrow $k \xrightarrow{\mathsf{a}} n$ in $\mathbb{L}$, its interpretation $\mathit{Ins}(F, G)^k \xrightarrow{[\![\mathsf{a}]\!]} \mathit{Ins}(F, G)^n$. Note here that the codomain itself is an inserter:

$$\mathit{Ins}(F, G)^n \quad \cong \quad \mathit{Ins}(\ \mathbb{C}^n \xrightarrow{F^n} \mathbb{C}^n,\ \mathbb{C}^n \xrightarrow{G^n} \mathbb{C}^n\ )\ .$$

We obtain such an interpretation $\mathit{Ins}(F,G)^k \xrightarrow{[\![\mathsf{a}]\!]} \mathit{Ins}(F^n,G^n)$ by the following 2-cell, via universality of the inserter $\mathit{Ins}(F^n,G^n)$.

$$\begin{tikzcd}
 & \mathbb{C}^k \arrow[r, "{[\![\mathsf{a}]\!]_{\mathbb{C}}}"] \arrow[dr, "F^k"'] & \mathbb{C}^n \arrow[dr, "F^n"] \arrow[d, phantom, "\swarrow F_{\mathsf{a}}"] & \\
\mathit{Ins}(F,G)^k \arrow[ur, "R^k"] \arrow[dr, "R^k"'] & \swarrow \rho^k & \mathbb{D}^k \arrow[r, "{[\![\mathsf{a}]\!]_{\mathbb{D}}}"] \arrow[d, phantom, "\searrow G_{\mathsf{a}}"] & \mathbb{D}^n \\
 & \mathbb{C}^k \arrow[r, "{[\![\mathsf{a}]\!]_{\mathbb{C}}}"'] \arrow[ur, "G^k"] & \mathbb{C}^n \arrow[ur, "G^n"'] &
\end{tikzcd}$$

Here the 2-cell ρ is the one accompanying $\mathit{Ins}(F,G)$; F_{a} comes from oplax naturality of F; and G_{a} comes from lax naturality of G.

Functoriality of the operation $\mathit{Ins}(F,G) : \mathbb{L} \to \mathbf{CAT}$ thus induced follows from universality of inserters. Moreover, $\mathbb{L}$-functoriality of $\mathit{Ins}(F,G) \overset{R}{\to} \mathbb{C}$ (i.e. naturality of $\mathit{Ins}(F,G) \overset{R}{\Rightarrow} \mathbb{C} : \mathbb{L} \to \mathbf{CAT}$) holds because $[\![\mathsf{a}]\!]$ arises as a "mediating" functor. □

Facts on $\mathbb{L}$-objects

5.3.13 Proposition. *1. A lax $\mathbb{L}$-functor preserves $\mathbb{L}$-objects. Hence so does an $\mathbb{L}$-functor.*

2. A final object of an $\mathbb{L}$-category $\mathbb{C}$, if it exists, is an $\mathbb{L}$-object. The inner $\mathbb{L}$-structure is induced by finality.

Proof. 1. A vertical composition of lax natural transformations is again a lax natural transformation. Hence the following vertical composition gives rise to an $\mathbb{L}$-object FX in an $\mathbb{L}$-category $\mathbb{D}$.

$$\begin{tikzcd}
\mathbb{L} \arrow[rr, bend left=40, "\mathbf{1}"] \arrow[rr, bend right=40, "\mathbb{D}"'] \arrow[rr, "\mathbb{C}" description] \arrow[rr, phantom, bend left=20, "\Downarrow X"] \arrow[rr, phantom, bend right=20, "\Downarrow F"] & & \mathbf{CAT}
\end{tikzcd}$$

2. Straightforward. □

When we fix an $\mathbb{L}$-category $\mathbb{C}$, the notion of *morphisms* between $\mathbb{L}$-objects in $\mathbb{C}$ is formalized as suitable *modifications* (see [19]).

5.3.14 Definition (Morphisms of $\mathbb{L}$-objects)**.** Let $\mathbb{C}$ be an $\mathbb{L}$-category, and X, Y be $\mathbb{L}$-objects in $\mathbb{C}$. A *morphism* f from X to Y is a modification of the following type.

$$\begin{tikzcd}
\mathbb{L} \arrow[rr, bend left=50, "\mathbf{1}"] \arrow[rr, "\mathbb{C}"'] \arrow[rr, phantom, bend left=20, "X \Downarrow \overset{f}{\Rrightarrow} \Downarrow Y"] & & \mathbf{CAT}
\end{tikzcd}$$

$\mathbb{L}$-objects in $\mathbb{C}$ and morphisms between them form a category; we shall denote it by $\mathbb{L}\text{-}\mathbf{obj}_{\mathbb{C}}$.

Let us spell out the definition. Such a modification f consists of 2-cells $f_k : X_k \Rightarrow Y_k$ in **CAT**, for each $k \in \mathbb{L}$, from X's k-component to Y's.

$$\begin{tikzcd}
\mathbf{1} \arrow[r, bend left, "{X_k = (X,\dots,X)}"] \arrow[r, bend right, "{Y_k = (Y,\dots,Y)}"'] \arrow[r, phantom, "\Downarrow f_k"] & \mathbb{C}^k
\end{tikzcd}$$

Additionally, these components f_k of f are required to be compatible with lax naturality of X and Y, in the following sense.

$$\begin{tikzcd}
\underline{\text{in } \mathbb{L}} & k \arrow[d, "\mathsf{a}"] & \underline{\text{in } \mathbf{CAT}} & \mathbf{1} \arrow[r, bend left, "X_k"] \arrow[d, equal] \arrow[dr, phantom, "\swarrow X_{\mathsf{a}}"] & \mathbb{C}^k \arrow[d, "{[\![\mathsf{a}]\!]}"] & {} \arrow[d, phantom, "="] & \mathbf{1} \arrow[r, bend left, "X_k"] \arrow[r, bend right, "Y_k"'] \arrow[r, phantom, "\Downarrow f_k"] \arrow[d, equal] \arrow[dr, phantom, "\swarrow Y_{\mathsf{a}}"] & \mathbb{C}^k \arrow[d, "{[\![\mathsf{a}]\!]}"] & (5.19) \\
& n & & \mathbf{1} \arrow[r, bend left, "X_n"] \arrow[r, bend right, "Y_n"'] \arrow[r, phantom, "\Downarrow f_n"] & \mathbb{C}^n & {} & \mathbf{1} \arrow[r, bend right, "Y_n"'] & \mathbb{C}^n &
\end{tikzcd}$$

A special case is when an arrow a is a projection $\pi_i : k \to 1$; in this case the mediating 2-cells X_{π_i} and Y_{π_i} are in fact identities because X and Y are product-preserving (Definition 5.3.7). The equality (5.19) in this special case proves that each component f_k of f is identified with a tuple

$$(X,\dots,X) \xrightarrow{f_k = (f,\dots,f)} (Y,\dots,Y) \qquad \text{in } \mathbb{C}^k,$$

where we identify f's 1-component with an arrow $f : X \to Y$ in $\mathbb{C}$.

When the arrow a in (5.19) is thought of as an algebraic operation, the equality requires that the arrow $f : X \to Y$ in $\mathbb{C}$ should be compatible with the inner interpretations X_{a} and Y_{a} of a. For illustration let us take $\mathsf{a} = \mathsf{m} : 2 \to 1$, a binary operation. The 2-cell on the left in (5.19) corresponds to the arrow $X \otimes X \overset{\mu^X}{\to} X \overset{f}{\to} Y$; the one on the right corresponds to $X \otimes X \overset{f \otimes f}{\to} Y \otimes Y \overset{\mu^Y}{\to} Y$. Therefore the equality in (5.19) instantiates to the following oft-seen diagram for a "morphism of algebras."

$$\begin{tikzcd}
X \otimes X \arrow[r, "f \otimes f"] \arrow[d, "\mu^X"'] & Y \otimes Y \arrow[d, "\mu^Y"] \\
X \arrow[r, "f"'] & Y
\end{tikzcd}$$

This diagram as an instance of (5.19) justifies Definition 5.3.14 of morphisms of $\mathbb{L}$-objects, that is, morphisms of (inner) $\mathbb{L}$-algebras.

It is standard that a limit $\lim(\mathbb{J} \to \mathbf{Sets})$ in **Sets** is presented as the set of "coherent elements" [96, Theorem V.1.1]; this is essentially due to the isomorphism $S \cong \mathbf{Sets}(1, S)$. Similar arguments, but using $\mathbb{C} \cong \mathbf{CAT}(\mathbf{1}, \mathbb{C})$ instead, lead to the following result. Its proof is straightforward.

5.3.15 Proposition. *The category* $\mathbb{L}$-$\mathbf{obj}_{\mathbb{C}}$ *is a lax FP-limit of the diagram* $\mathbb{C} : \mathbb{L} \to \mathbf{CAT}$. *That is, the canonical cone from* $\mathbb{L}$-$\mathbf{obj}_{\mathbb{C}}$ *over* $\mathbb{C} : \mathbb{L} \to \mathbf{CAT}$ *is universal among the lax cones over* $\mathbb{C} : \mathbb{L} \to \mathbf{CAT}$

$$\begin{tikzcd} \mathbb{L} \arrow[r, bend left=50, "\Delta\mathbb{B}"] \arrow[r, "\mathbb{C}"'] \arrow[r, phantom, bend left=25, "\Downarrow\beta"] & \mathbf{CAT} \end{tikzcd}$$

which are product-preserving (meaning $\beta \circ H$ *is strictly natural; see Definition 5.3.7).* □

5.4 Microcosm structures in coalgebras

In this section we return to our original question and apply the framework we just introduced to coalgebraic settings. We present our main result in this chapter (Theorem 5.4.2), which generalizes the previous compositionality result (Theorem 5.2.1). Here the constructs in Section 5.2 (such as sync) will appear again, now in their generalized form.

Lax $\mathbb{L}$-functors generalize sync

In Section 5.2 we observed that an endofunctor F with $\mathsf{sync} : FX \otimes FY \to F(X \otimes Y)$ allows a lifting of an $\otimes$-structure on $\mathbb{C}$ to an $\otimes$-structure on $\mathbf{Coalg}_F$. In the current generalized setting where we consider an arbitrary algebraic theory $\mathbb{L}$, what corresponds to this endofunctor F with sync is a *lax* $\mathbb{L}$-endofunctor F (Definition 5.3.9). To illustrate this, look at the following lax naturality diagram for a binary operation m in $\mathbb{L} = \mathbf{Mon}$.

$$\begin{tikzcd} \underline{\text{in }\mathbf{Mon}} & 2 \arrow[d, "\mathsf{m}"] & \underline{\text{in }\mathbf{CAT}} & \mathbb{C}^2 \arrow[r, "F_2 = F \times F"] \arrow[d, "{[\![\mathsf{m}]\!] = \otimes}"'] \arrow[dr, phantom, "\swarrow F_{\mathsf{m}}"] & \mathbb{C}^2 \arrow[d, "{[\![\mathsf{m}]\!] = \otimes}"] \\ & 1 & & \mathbb{C} \arrow[r, "F"'] & \mathbb{C} \end{tikzcd} \tag{5.20}$$

The 2-component is $F_2 = F \times F$ because the lax natural transformation F is product-preserving; see (5.18). The mediating 2-cell F_{m} in (5.20) is identified with a natural transformation

$$FX \otimes FY \longrightarrow F(X \otimes Y) \ ;$$

this is what we previously called sync.

5.4.1 Remark. Moreover, F_{m} (identified with sync) becomes automatically compatible with equational properties such as associativity, in the sense of (5.8). This is because of the coherence condition (just like (5.13)) on F's mediating 2-cells. For example, using coherence we have the following equalities much like in (5.14).

<u>in **Mon**</u>

$$\begin{tikzcd} & 3 \arrow[dl, "\mathsf{id}\times\mathsf{m}"'] \arrow[dr, "\mathsf{m}\times\mathsf{id}"] & \\ 2 \arrow[dr, "\mathsf{m}"'] & & 2 \arrow[dl, "\mathsf{m}"] \\ & 1 & \end{tikzcd}$$

<u>in **CAT**</u>

$$\begin{tikzcd} \mathbb{C}^3 \arrow[r, "F\times F\times F"] \arrow[d] \arrow[dr, phantom, "\Swarrow F_{\mathsf{id}\times\mathsf{m}}"] & \mathbb{C}^3 \arrow[d, "[\![\mathsf{id}\times\mathsf{m}]\!]"] \\ \mathbb{C}^2 \arrow[r] \arrow[d] \arrow[dr, phantom, "\Swarrow F_{\mathsf{m}}"] & \mathbb{C}^2 \arrow[d, "[\![\mathsf{m}]\!]"] \\ \mathbb{C} \arrow[r, "F"'] & \mathbb{C} \end{tikzcd} = \begin{tikzcd} \mathbb{C}^3 \arrow[r] \arrow[d] \arrow[dr, phantom, "{\Swarrow F_{\mathsf{m}\circ(\mathsf{id}\times\mathsf{m})} = F_{\mathsf{m}\circ(\mathsf{m}\times\mathsf{id})}}"] & \mathbb{C}^3 \arrow[d] \\ \mathbb{C}^1 \arrow[r] & \mathbb{C}^1 \end{tikzcd} = \begin{tikzcd} \mathbb{C}^3 \arrow[r, "F\times F\times F"] \arrow[d] \arrow[dr, phantom, "\Swarrow F_{\mathsf{m}\times\mathsf{id}}"] & \mathbb{C}^3 \arrow[d, "[\![\mathsf{m}\times\mathsf{id}]\!]"] \\ \mathbb{C}^2 \arrow[r] \arrow[d] \arrow[dr, phantom, "\Swarrow F_{\mathsf{m}}"] & \mathbb{C}^2 \arrow[d, "[\![\mathsf{m}]\!]"] \\ \mathbb{C} \arrow[r, "F"'] & \mathbb{C} \end{tikzcd} \tag{5.21}$$

The composed 2-cell on the left corresponds to a natural transformation

$$FX \otimes (FY \otimes FZ) \xrightarrow{FX\otimes\mathsf{sync}} FX \otimes F(Y \otimes Z) \xrightarrow{\mathsf{sync}} F(X \otimes (Y \otimes Z)) \ ;$$

the one on the right corresponds to

$$(FX \otimes FY) \otimes FZ \xrightarrow{\mathsf{sync}\otimes FZ} F(X \otimes Y) \otimes FZ \xrightarrow{\mathsf{sync}} F((X \otimes Y) \otimes Z) \ .$$

Hence the equalities above in (5.21) prove: if F is a lax **Mon**-functor, then its sync (identified with the 2-cell F_{m}) is automatically "associative" in the sense of (5.8).

General compositionality

Among the ingredients for the previous compositionality result (Theorem 5.2.1), an endofunctor F with sync has now been generalized to a lax $\mathbb{L}$-functor F. The other ingredient—the base category $\mathbb{C}$ with a tensor $\otimes$—has an immediate generalization as an $\mathbb{L}$-category $\mathbb{C}$. These generalized notions constitute an appropriate setting for the following main result of this chapter.

5.4.2 Theorem (General compositionality)**.** *Let $\mathbb{C}$ be an $\mathbb{L}$-category and $F : \mathbb{C} \to \mathbb{C}$ be a lax $\mathbb{L}$-functor. Assume further that $\zeta : Z \xrightarrow{\cong} FZ$ is the final coalgebra. Then:*

1. *The category $\mathbf{Coalg}_F$ of F-coalgebras is an $\mathbb{L}$-category.*
2. *So is the slice category $\mathbb{C}/Z$.*
3. *The functor beh : $\mathbf{Coalg}_F \to \mathbb{C}/Z$ is a (non-lax) $\mathbb{L}$-functor. It makes the following diagram of $\mathbb{L}$-functors commute.*

$$\begin{tikzcd} \mathbf{Coalg}_F \arrow[rr, "\mathsf{beh}"] \arrow[dr, "U"'] & & \mathbb{C}/Z \arrow[dl, "\mathrm{dom}"] \\ & \mathbb{C} & \end{tikzcd}$$

Proof. 1. This is the same as Proposition 5.3.10.1.

2. $\zeta \in \mathbf{Coalg}_F$ is an $\mathbb{L}$-object (Proposition 5.3.13.2); so is $Z = U\zeta$ (Propositions 5.3.13.1 and 5.3.10.1); hence $\mathbb{C}/Z$ is an $\mathbb{L}$-category (Proposition 5.3.10.2).

3. Straightforward by finality. □

In fact, the theorem also generalizes our previous result (Theorem 5.2.4) stating: if sync is "associative," then $\otimes$ on $\mathbf{Coalg}_F$ is associative. Indeed, the statement of the above Theorem 5.4.2.1—$\mathbf{Coalg}_F$ being an $\mathbb{L}$-category—means, not only that operations are interpreted in $\mathbf{Coalg}_F$, but also that all the equational properties specified in $\mathbb{L}$ are satisfied in $\mathbf{Coalg}_F$. The assumption in the previous Theorem 5.2.4—associativity of sync—is present in the generalized Theorem 5.4.2, in the form of F's being a lax $\mathbb{L}$-functor (see Remark 5.4.1).

We describe concretely what are the lifted algebraic structures on $\mathbf{Coalg}_F$ and on $\mathbb{C}/X$ in Proposition 5.3.10; such a concrete description is helpful in seeing that Theorem 5.4.2.3 indeed implies coalgebraic compositionality expressed as the equation (5.4). An operation $\mathsf{a} : k \to 1$ in $\mathbb{L}$ is interpreted in $\mathbf{Coalg}_F$ and $\mathbb{C}/X$ as follows, respectively, using F_{a} from lax naturality of F and the inner algebraic structure X_{a} on X.

$$[\![\mathsf{a}]\!]_{\mathbf{Coalg}_F} : \left(\begin{array}{c} FX_1 \\ \uparrow c_1 \\ X_1 \end{array}, \ldots, \begin{array}{c} FX_k \\ \uparrow c_k \\ X_k \end{array}\right) \longmapsto \begin{array}{c} F[\![\mathsf{a}]\!](\overrightarrow{X}) \\ \uparrow (F_{\mathsf{a}})_{\overrightarrow{X}} \\ [\![\mathsf{a}]\!](\overrightarrow{FX}) \\ \uparrow [\![\mathsf{a}]\!](\overrightarrow{c}) \\ [\![\mathsf{a}]\!](\overrightarrow{X}) \end{array} ,$$

$$[\![\mathsf{a}]\!]_{\mathbb{C}/X} : \left(\begin{array}{c} Y_1 \\ \downarrow y_1 \\ X \end{array}, \ldots, \begin{array}{c} Y_k \\ \downarrow y_k \\ X \end{array}\right) \longmapsto \begin{array}{c} [\![\mathsf{a}]\!](\overrightarrow{Y}) \\ \downarrow [\![\mathsf{a}]\!](\overrightarrow{y}) \\ [\![\mathsf{a}]\!](\overrightarrow{X}) \\ \downarrow X_{\mathsf{a}} \\ X \end{array} .$$

These generalize the lifted binary operations $\otimes$ on $\mathbf{Coalg}_F$ (5.5) and $\underline{\otimes}$ on $\mathbb{C}/Z$ (5.7). Note the use of F_{a} (in place of sync) and X_{a} (in place of $\|$).

5.5 Parallel composition of coalgebras in $\mathcal{K}\ell(T)$

Our main results—Theorem 5.2.1 and generalized Theorem 5.4.2—apply to our coalgebraic framework for trace semantics (Chapter 2) as well. In this case we take a suitable Kleisli category as the base category $\mathbb{C}$; coinduction now captures trace semantics (as opposed to bisimilarity captured in $\mathbb{C} = \mathbf{Sets}$).

In this section we shall describe some specific issues regarding this Kleisli setting. In Section 5.5.1 the ingredients for our parallel composition framework, namely:

- an algebraic structure $\otimes$ on the base category $\mathbb{C} = \mathcal{K}\ell(T)$, and
- a synchronization natural transformation which lifts $\otimes$ on the base category to $\otimes$ on coalgebras,

are reduced to certain constructs in $\mathbf{Sets}$. In Section 5.5.2 we further investigate the relationship between parallel composition and forward/backward simulations (Chapter 3).

5.5.1 Trace equivalence is a congruence

The following is the relevant setting from Chapter 2. The functor F on **Sets** is lifted to $\overline{F}$ on $\mathcal{K}\ell(T)$ with the help of a distributive law $\lambda : FT \Rightarrow TF$. Systems are presented as $\overline{F}$-coalgebras (in $\mathcal{K}\ell(T)$) and coinduction there yields trace semantics.

$$\begin{tikzcd}
\mathbf{Coalg}_{\overline{F}} \arrow[r] & \mathcal{K}\ell(T) \arrow[loop right, "\overline{F}"] \\
& \mathbf{Sets} \arrow[u] \arrow[loop left, "T"] \arrow[loop right, "F"]
\end{tikzcd}$$

We shall show that, when we apply the parallel composition framework (Theorem 5.2.1) to this Kleisli setting, we can reduce the ingredients in $\mathcal{K}\ell(T)$—namely $\overline{F}$, $\otimes$ and $\overline{\mathsf{sync}}$—to the following constructs in **Sets**.

$\overline{F} : \mathcal{K}\ell(T) \to \mathcal{K}\ell(T)$
to $F : \mathbf{Sets} \to \mathbf{Sets}$ and $\lambda : FT \Rightarrow TF$;

$\otimes$ on $\mathcal{K}\ell(T)$
to ($\times$ on **Sets**, and) $\xi : TX \times TY \to T(X \times Y)$;

$\overline{\mathsf{sync}} : \overline{F}X \otimes \overline{F}Y \to \overline{F}(X \otimes Y)$ in $\mathcal{K}\ell(T)$
to $\mathsf{sync} : FX \times FY \to F(X \times Y)$ in **Sets**, compatible with λ and ξ.

The first item (reduction of $\overline{F}$ to F and λ) is already described in Section 2.2.

We obtain a tensor $\otimes$ on $\mathcal{K}\ell(T)$ as a lifting of Cartesian products $\times$ on **Sets**. Such a lifting is available when the monad T comes with a natural transformation

$$\xi_{X,Y} \;:\; TX \times TY \to T(X \times Y)$$

which is compatible with the monad structure in the obvious way:

$$\begin{tikzcd}
X \times Y \arrow[r, "\eta \times \eta"] \arrow[dr, "\eta"'] & TX \times TY \arrow[d, "\xi"] \\
& T(X \times Y)
\end{tikzcd}
\qquad
\begin{tikzcd}
T^2X \times T^2Y \arrow[r, "\xi"] \arrow[d, "\mu \times \mu"'] & T(TX \times TY) \arrow[r, "T\xi"] & T^2(X \times Y) \arrow[d, "\mu"] \\
TX \times TY \arrow[rr, "\xi"'] & & T(X \times Y)
\end{tikzcd} \;. \tag{5.22}$$

The lifted tensor

$$\otimes \;:\; \mathcal{K}\ell(T) \times \mathcal{K}\ell(T) \longrightarrow \mathcal{K}\ell(T)$$

acts as products on objects: $X \otimes Y = X \times Y$. Its action on arrows is defined as follows, using the natural transformation ξ.

$$\begin{array}{cc}
X \otimes Y \xrightarrow{f \otimes g} V \otimes W & \text{in } \mathcal{K}\ell(T) \\
\hline\hline
X \times Y \xrightarrow{f \times g} TV \times TW \xrightarrow{\xi_{V,W}} T(V \times W) & \text{in } \mathbf{Sets}
\end{array} \tag{5.23}$$

The natural transformation ξ's compatibility with η and μ (5.22) is needed to ensure that the lifted $\otimes$ is a bifunctor.

The monads $\mathcal{L}$, $\mathcal{P}$ and $\mathcal{D}$ of our interest—which specify types of branching, see Chapter 2—are equipped with such ξ. Indeed, those monads are commutative and their double strength dst (2.13) serves as ξ. Consequently the Kleisli category $\mathcal{K}\ell(T)$ for $T \in \{\mathcal{L}, \mathcal{P}, \mathcal{D}\}$ has a tensor $\otimes$ which arises in the way described above.

In order to further lift this tensor $\otimes$ on $\mathcal{K}\ell(T)$ to parallel composition operator $\boldsymbol{\otimes}$ on $\mathbf{Coalg}_{\overline{F}}$ (Theorem 5.2.1), we need a synchronization

$$\overline{\mathsf{sync}} \;:\; \overline{F}X \otimes \overline{F}Y \longrightarrow \overline{F}(X \otimes Y) \qquad \text{in } \mathcal{K}\ell(T).$$

This $\overline{\mathsf{sync}}$ in $\mathcal{K}\ell(T)$ can be obtained as a lifting of sync in $\mathbf{Sets}$, in case the latter is suitably compatible with λ and ξ.

5.5.1 Proposition. *Assume the following.*

- *F is an endofunctor on* $\mathbf{Sets}$ *equipped with a natural transformation*

$$\mathsf{sync}_{X,Y} \;:\; FX \times FY \longrightarrow F(X \times Y) \;;$$

- *T is a monad on* $\mathbf{Sets}$ *with a natural transformation*

$$\xi_{X,Y} \;:\; TX \times TY \longrightarrow T(X \times Y)$$

satisfying the compatibility condition (5.22); and

- *we have a distributive law $\lambda : FT \Rightarrow TF$ relating them.*

Assume further that the natural transformations sync*, ξ and λ make the following diagram commute.*

$$\begin{tikzcd}
FTX \otimes FTY \arrow[r, "\mathsf{sync}"] \arrow[d, "\lambda\otimes\lambda"'] & F(TX \otimes TY) \arrow[r, "F\xi"] & FT(X \otimes Y) \arrow[d, "\lambda"] \\
TFX \otimes TFY \arrow[r, "\xi"'] & T(FX \otimes FY) \arrow[r, "T\,\mathsf{sync}"'] & TF(X \otimes Y)
\end{tikzcd} \tag{5.24}$$

If this is the case, then

$$\overline{\mathsf{sync}}_{X,Y} := \eta \circ \mathsf{sync}_{X,Y} \quad : \quad \overline{F}X \otimes \overline{F}Y \longrightarrow \overline{F}(X \otimes Y) \qquad \textit{in } \mathcal{K}\ell(T)$$

is indeed a natural transformation.

Consequently, the category $\mathcal{K}\ell(T)$ with the tensor $\otimes$ (induced by ξ) and the endofunctor $\overline{F}$ with $\overline{\mathsf{sync}}$ *thus obtained satisfy the assumptions of Theorem 5.2.1. This ensures that behavior by coinduction (giving trace semantics) satisfies compositionality (5.4).*

Proof. We shall show the naturality of $\overline{\mathsf{sync}} = \eta \circ \mathsf{sync}_{X,Y}$. Let $f : X \to V$ and $g : Y \to W$ be arrows in $\mathcal{K}\ell(T)$.

$$
\begin{array}{ll}
\overline{F}(f \otimes_{\mathcal{K}\ell(T)} g) \circ_{\mathcal{K}\ell(T)} \overline{\mathsf{sync}}_{X,Y} & \\
\quad = \mu \circ T(\overline{F}(f \otimes_{\mathcal{K}\ell(T)} g)) \circ \overline{\mathsf{sync}} & \\
\quad = \mu \circ T\lambda \circ TF(f \otimes_{\mathcal{K}\ell(T)} g) \circ \eta \circ \mathsf{sync} & \\
\quad = \mu \circ \underline{T\lambda \circ TF\xi \circ TF(f \times g) \circ \eta} \circ \mathsf{sync} & \text{by def. of } \otimes_{\mathcal{K}\ell(T)} \\
\quad = \underline{\mu \circ \eta} \circ \lambda \circ F\xi \circ \underline{F(f \times g) \circ \mathsf{sync}} & \text{naturality of } \eta \\
\quad = \underline{\lambda \circ F\xi \circ \mathsf{sync}} \circ (Ff \times Fg) & \text{unit law, naturality of } \mathsf{sync} \\
\quad = T\,\mathsf{sync} \circ \xi \circ (\lambda \times \lambda) \circ (Ff \times Fg) & \text{assumption} \\
\quad = \mu \circ T\eta \circ T\,\mathsf{sync} \circ \xi \circ (\lambda \times \lambda) \circ (Ff \times Fg) & \text{unit law} \\
\quad = \overline{\mathsf{sync}} \circ_{\mathcal{K}\ell(T)} (\overline{F}f \otimes_{\mathcal{K}\ell(T)} \overline{F}g) \ . &
\end{array}
$$

□

5.5.2 Example. To look at more concrete examples, let us take the powerset monad $T = \mathcal{P}$ modeling non-deterministic branching, and $F = \{\checkmark\} + \Sigma \times \_$ modeling the transition-type "terminate, or output and continue." This is the leading example in Chapter 2. An $\overline{F}$-coalgebra $X \to \overline{F}X$ in $\mathcal{K}\ell(\mathcal{P})$ is an LTS with explicit termination; by coinduction it is assigned a map $X \to \mathcal{P}(\Sigma^*)$ in **Sets** which gives the terminating sequences of actions. For this choice of F, one possible choice of $\mathsf{sync} : FX \times FY \to F(X \times Y)$ is as follows.

$$
\begin{array}{rccccl}
\mathsf{sync} : & (\{\checkmark\} + \Sigma \times X) \times (\{\checkmark\} + \Sigma \times Y) & \longrightarrow & \{\checkmark\} + \Sigma \times (X \times Y) & \\
& \big((a, x)\,,\,(a, y)\big) & \longmapsto & (a, (x, y)), & \\
& (u, v) & \longmapsto & \checkmark & \text{otherwise.}
\end{array}
\tag{5.25}
$$

We can easily check that this sync is compatible with the canonical ξ and λ in the sense of (5.24). Therefore this sync on **Sets** lifts to $\overline{\mathsf{sync}}$ on $\mathcal{K}\ell(\mathcal{P})$ and Theorem 5.2.1 applies. We conclude that trace equivalence (captured by coinduction) is a congruence with respect to the parallel composition obtained from this sync.

5.5.3 Example. When we replace $T = \mathcal{P}$ in the previous Example 5.5.2 by the subdistribution monad $\mathcal{D}$, the systems have probabilistic branching and coinduction captures trace *distribution* semantics. The sync in (5.25) is well-behaved in this case too; it is lifted to $\overline{\mathsf{sync}}$ on $\mathcal{K}\ell(\mathcal{D})$ and by Theorem 5.2.1 we conclude that the equivalence modulo trace distribution semantics is a congruence.

Generalization to an arbitrary $\mathbb{L}$

We have described how we can obtain a parallel composition framework in $\mathcal{K}\ell(T)$ from suitable constructs in **Sets**. We have focused on a simple setting where we only consider one binary operation (for parallel composition) and no equational properties. Let us generalize our observation to an arbitrary algebraic theory $\mathbb{L}$. A general principle which yields the compatibility condition (5.24) will be revealed in its course.

5.5.4 Proposition. *1. Let $\mathbb{C}$ be an $\mathbb{L}$-category, and $T : \mathbb{C} \to \mathbb{C}$ be a monad such that:*

- *its functor part is a lax $\mathbb{L}$-functor; and*
- *the mediating 2-cells T_{a} (of T as a lax $\mathbb{L}$-functor) are compatible with the monad structure in an obvious way. Namely, for each arrow $\mathsf{a} : k \to n$ in $\mathbb{L}$,*

$$\begin{tikzcd}
\cdot \arrow[r, bend left=50, "\mathrm{id}"] \arrow[r, bend right=10, "T^k"'] \arrow[r, phantom, bend left=25, "\Downarrow\eta^k"] \arrow[d, "{[\![\mathsf{a}]\!]}"'] \arrow[dr, phantom, "\swarrow T_{\mathsf{a}}"] & \cdot \arrow[d, "{[\![\mathsf{a}]\!]}"] \\
\cdot \arrow[r, bend right=30, "T^n"'] & \cdot
\end{tikzcd}
=
\begin{tikzcd}
\cdot \arrow[r, bend left=30, "\mathrm{id}"] \arrow[d, "{[\![\mathsf{a}]\!]}"'] \arrow[dr, phantom, "/\!/"] & \cdot \arrow[d, "{[\![\mathsf{a}]\!]}"] \\
\cdot \arrow[r, bend left=40, "\mathrm{id}"] \arrow[r, bend right=40, "T^n"'] \arrow[r, phantom, "\Downarrow\eta^n"] & \cdot
\end{tikzcd}$$

$$\begin{tikzcd}
 & \cdot \arrow[dr, bend left, "T^k"] & \\
\cdot \arrow[ur, bend left, "T^k"] \arrow[rr, "T^k"'] \arrow[rr, phantom, bend left=30, "\Downarrow\mu^k"] \arrow[d, "{[\![\mathsf{a}]\!]}"'] \arrow[drr, phantom, "\swarrow T_{\mathsf{a}}"] & & \cdot \arrow[d, "{[\![\mathsf{a}]\!]}"] \\
\cdot \arrow[rr, "T^n"'] & & \cdot
\end{tikzcd}
=
\begin{tikzcd}
 & \cdot \arrow[dr, bend left, "T^k"] \arrow[dd, "{[\![\mathsf{a}]\!]}"] & \\
\cdot \arrow[ur, bend left, "T^k"] \arrow[dd, "{[\![\mathsf{a}]\!]}"'] \arrow[dr, phantom, "T_{\mathsf{a}}\swarrow"] & & \cdot \arrow[dd, "{[\![\mathsf{a}]\!]}"] \arrow[dl, phantom, "T_{\mathsf{a}}\swarrow"] \\
 & \cdot \arrow[dr, "T^n"] & \\
\cdot \arrow[ur, "T^n"] \arrow[rr, "T^n"'] \arrow[rr, phantom, bend left=20, "\Downarrow\mu^n"] & & \cdot
\end{tikzcd}$$

Then the Kleisli category $\mathcal{K}\ell(T)$ is an $\mathbb{L}$-category; moreover the canonical Kleisli inclusion $\mathbb{C} \to \mathcal{K}\ell(T)$ is an $\mathbb{L}$-functor.

2. Assume further that we have a lax $\mathbb{L}$-functor $F : \mathbb{C} \to \mathbb{C}$ and a distributive law $\lambda : FT \Rightarrow TF$, which are altogether compatible in the following sense. For each arrow $\mathsf{a} : k \to n$ in $\mathbb{L}$,

$$\begin{tikzcd}
 & \cdot \arrow[dr, "F^k"] \arrow[d, phantom, "\Downarrow\lambda^k"] & \\
\cdot \arrow[ur, "T^k"] \arrow[r, "F^k"'] \arrow[dd, "{[\![\mathsf{a}]\!]}"'] \arrow[ddr, phantom, "\swarrow F_{\mathsf{a}}"] & \cdot \arrow[r, "T^k"'] \arrow[dd] \arrow[ddr, phantom, "T_{\mathsf{a}}\ \swarrow"] & \cdot \arrow[dd, "{[\![\mathsf{a}]\!]}"] \\
 & & \\
\cdot \arrow[r, "F^n"'] & \cdot \arrow[r, "T^n"'] & \cdot
\end{tikzcd}
=
\begin{tikzcd}
 & \cdot \arrow[dr, "F^k"] \arrow[dd] & \\
\cdot \arrow[ur, "T^k"] \arrow[dd, "{[\![\mathsf{a}]\!]}"'] \arrow[dr, phantom, "\swarrow T_{\mathsf{a}}"] & & \cdot \arrow[dd, "{[\![\mathsf{a}]\!]}"] \arrow[dl, phantom, "F_{\mathsf{a}}\swarrow"] \\
 & \cdot \arrow[dr, "F^n"] \arrow[d, phantom, "\Downarrow\lambda^n"] & \\
\cdot \arrow[ur, "T^n"] \arrow[r, "F^n"'] & \cdot \arrow[r, "T^n"'] & \cdot
\end{tikzcd} \tag{5.26}$$

Then the lifted functor $\overline{F} : \mathcal{K}\ell(T) \to \mathcal{K}\ell(T)$ is a lax $\mathbb{L}$-functor. If moreover there is a final $\overline{F}$-coalgebra (which is the case in the setting of Chapter 2), the general compositionality result (Theorem 5.4.2) applies to $\overline{F}$-coalgebras. □

This proposition (whose proof is straightforward) generalizes the previous observations. Specifically, a monad T which satisfies the assumption of Proposition 5.5.4.1 replaces a monad T with a natural transformation $\xi : TX \otimes TY \to T(X \otimes Y)$; the resulting $\mathbb{L}$-structure on $\mathcal{K}\ell(T)$ replaces the previous $\otimes$ on $\mathcal{K}\ell(T)$. The condition (5.26) can be thought of as a "coherence condition on a modification $\lambda : FT \Rrightarrow TF$ between lax natural transformations" (see e.g. [19]); it instantiates to the previous condition (5.24).

5.5.2 Simulations are compositional

Let us continue our investigation and look at the relationship between parallel composition and forward/backward simulations. Our principal result here is: forward/backward similarity relations $\sqsubseteq_{\mathbf{F}}$ and $\sqsubseteq_{\mathbf{B}}$ (Definitions 3.2.8 and 3.2.9) are compositional, that is,

$$\begin{array}{c}\overline{F}X\\ c\uparrow\\ X\end{array} \sqsubseteq_{\mathbf{F}} \begin{array}{c}\overline{F}Y\\ d\uparrow\\ Y\end{array} \quad\Longrightarrow\quad \left(\begin{array}{ccc}\overline{F}X & & \overline{F}V\\ c\uparrow & \otimes & e\uparrow\\ X & & V\end{array}\right) \sqsubseteq_{\mathbf{F}} \left(\begin{array}{ccc}\overline{F}Y & & \overline{F}V\\ d\uparrow & \otimes & e\uparrow\\ Y & & V\end{array}\right) . \tag{5.27}$$

A sufficient condition for this compositionality is described as a simple order-theoretic property of the tensor $\otimes$ on $\mathcal{K}\ell(T)$. This condition can be further reduced to a property of ξ, a natural transformation which lifts $\times$ on **Sets** to $\otimes$ on $\mathcal{K}\ell(T)$ (Section 5.5.1).

The above compositionality (5.27) is concerned with the forward similarity $\sqsubseteq_{\mathbf{F}}$; similar results hold also for the backward similarity relation $\sqsubseteq_{\mathbf{B}}$ and the backward-forward one $\sqsubseteq_{\mathbf{BF}}$. The compositionality (5.27) also focuses on the first argument of the operator $\otimes$; compositionality with respect to the second argument is proved in the same way as well.

Throughout this section we ignore explicit start state maps which, together with $\overline{F}$-coalgebras, formed (T, F)-systems in Chapter 3. We do so for simplicity of presentation; incorporating start state maps again can be done in a straightforward way.

5.5.5 Proposition. *Let (F, T, λ) be a trace situation (Theorem 2.3.3). Assume further that*

- *the Kleisli category $\mathcal{K}\ell(T)$ has a tensor $\otimes$; and*
- *the lifted functor $\overline{F} : \mathcal{K}\ell(T) \to \mathcal{K}\ell(T)$ comes with a natural transformation $\overline{\mathsf{sync}} : \overline{F}X \otimes \overline{F}Y \to \overline{F}(X \otimes Y)$,*

so that we have a parallel composition operator $\otimes$ for $\overline{F}$-coalgebras (Theorem 5.2.1).

If the tensor

$$\otimes \;:\; \mathcal{K}\ell(T) \times \mathcal{K}\ell(T) \longrightarrow \mathcal{K}\ell(T)$$

is locally monotone, then the similarity relations $\sqsubseteq_{\mathbf{F}}$, $\sqsubseteq_{\mathbf{B}}$ and $\sqsubseteq_{\mathbf{BF}}$ are compositional as in (5.27).

Proof. Local monotonicity of $\otimes$ on $\mathcal{K}\ell(T)$ means:

$$\begin{tikzcd}[column sep=small] \underline{\text{in } \mathcal{K}\ell(T)} & X \arrow[d, bend right, "f"'] \arrow[d, bend left, "f'"] \arrow[d, phantom, "\sqsubseteq"] & , & Y \arrow[d, bend right, "g"'] \arrow[d, bend left, "g'"] \arrow[d, phantom, "\sqsubseteq"] & \Longrightarrow & X \otimes Y \arrow[d, bend right, "f \otimes g"'] \arrow[d, bend left, "f' \otimes g'"] \arrow[d, phantom, "\sqsubseteq"] & . \\ & V & & W & & V \otimes W & \end{tikzcd}$$

Let us focus on the compositionality (5.27), that is, compositionality of $\sqsubseteq_{\mathbf{F}}$ with respect to the first argument. By assumption we have a forward simulation f from c

to d.

$$\begin{tikzcd}
\underline{\text{in } \mathcal{K}\ell(T)} & \overline{F}Y \arrow[r, "\overline{F}f"] & \overline{F}X \\
& Y \arrow[u, "d"] \arrow[r, "f"'] \arrow[ur, phantom, "\sqsupseteq"] & X \arrow[u, "c"']
\end{tikzcd}$$

By local monotonicity of $\otimes$, this yields the lower inequality in the following diagram. The upper square commutes by naturality of $\overline{\mathsf{sync}}$; the whole diagram stipulates that $f \otimes V$ is a forward simulation from $c \otimes e$ to $d \otimes e$.

$$\begin{tikzcd}
\overline{F}(Y \otimes V) \arrow[r, "\overline{F}(f \otimes V)"] & \overline{F}(X \otimes V) \\
\overline{F}Y \otimes \overline{F}V \arrow[u, "\overline{\mathsf{sync}}_{Y,V}"] \arrow[r, "\overline{F}f \otimes V"] & \overline{F}X \otimes \overline{F}V \arrow[u, "\overline{\mathsf{sync}}_{X,V}"'] \\
Y \otimes V \arrow[u, "d \otimes e"] \arrow[r, "f \otimes V"'] \arrow[ur, phantom, "\sqsupseteq"] \arrow[uu, bend left=70, "d \otimes e"] & X \otimes V \arrow[u, "c \otimes e"'] \arrow[uu, bend right=70, "c \otimes e"']
\end{tikzcd}$$

Hence we have shown that $c \otimes e \sqsubseteq_{\mathbf{F}} d \otimes e$.

Compositionality of $\sqsubseteq_{\mathbf{B}}$ is proved by similar arguments. That of $\sqsubseteq_{\mathbf{BF}}$ follows from the characterization (3.10) of $\sqsubseteq_{\mathbf{BF}}$. □

Can we further interpret the condition in the previous result (namely local monotonicity of $\otimes$) in terms of more elementary constructs? We have shown that $\otimes$ on $\mathcal{K}\ell(T)$ is obtained from a natural transformation

$$\xi_{X,Y} \;:\; TX \times TY \to T(X \times Y) \qquad \text{in } \mathbf{Sets}$$

compatible with the monad structure of T. In fact, a monotone ξ induces a locally monotone $\otimes$.

5.5.6 Proposition. *Let us take the setting of Proposition 5.5.5, except for local monotonicity of $\otimes : \mathcal{K}\ell(T) \times \mathcal{K}\ell(T) \to \mathcal{K}\ell(T)$. Assume further that*

- *the tensor $\otimes : \mathcal{K}\ell(T) \times \mathcal{K}\ell(T) \to \mathcal{K}\ell(T)$ arises from $\times$ on* **Sets** *via a natural transformation ξ, as in (5.23);*
- *the* **Cppo***-enriched structure of $\mathcal{K}\ell(T)$ arises from the* **Cppo***-structures of sets TY, in a pointwise manner. That is,*

$$\begin{tikzcd}
X \arrow[r, bend left, "g"] \arrow[r, bend right, "f"'] \arrow[r, phantom, "\rotatebox{90}{$\sqsubseteq$}"] & Y
\end{tikzcd} \quad \textit{in } \mathcal{K}\ell(T) \qquad \textit{if and only if} \qquad \forall x \in X.\; f(x) \sqsubseteq_{TY} g(x) \;.$$

This is indeed the case for $T \in \{\mathcal{L}, \mathcal{P}, \mathcal{D}\}$; see Lemma 2.2.6.

If the function $\xi_{X,Y} : TX \times TY \to T(X \times Y)$ is monotone for each X and Y, then the tensor $\otimes$ on $\mathcal{K}\ell(T)$ is locally monotone. Therefore by Proposition 5.5.5 similarity relations $\sqsubseteq_{\mathbf{F}}$, $\sqsubseteq_{\mathbf{B}}$ and $\sqsubseteq_{\mathbf{BF}}$ are compositional. □

Each monad $T \in \{\mathcal{L}, \mathcal{P}, \mathcal{D}\}$ of our interest (specifying branching type) comes with a natural transformation ξ which is given by T's double strength dst; see (2.14). It is easy to see that a component of such ξ is monotone. Hence, in order for Proposition 5.5.6 to apply, what we still need is only a proper sync natural transformation.

5.6 Summary and future work

In this chapter we have observed that the microcosm principle (as called by Baez and Dolan) brings new mathematical insights into computer science. Specifically, we have looked into parallel composition of coalgebras, which would serve as a mathematical basis for the study of concurrency. As a purely mathematical expedition, we have presented a 2-categorical formalization of the microcosm principle, where an algebraic theory is presented by a Lawvere theory. Turning back to our original motivation, the formalization was applied to coalgebras and produced some general results which ensure compositionality and equational properties such as associativity.

There are many questions yet to be answered. Some of them have been already mentioned, namely: extending the expressive power of sync (Remark 5.2.3), and a proper treatment of "pseudo" algebraic structures (Section 5.3.3).

On the application side, one direction of future work is to establish a relationship between sync and *(syntactic) formats* for process algebras. Our sync represents a certain class of operational rules; formats are a more syntactic way to do the same. Formats which guarantee certain good properties (such as commutativity, see [102]) have been actively studied. Such a format should be obtained by translating e.g. a "commutative" sync into a format.

Another applicational direction is an abstract, categorical account of *testing equivalences* as introduced in [30].[12] In the testing framework parallel composition plays a crucial role: a system $\mathcal{C}$ to be tested is composed with another system $\mathcal{T}$ which represents a test; the behavior of $\mathcal{C} \parallel \mathcal{T}$ (typically outputting "OK") tells us if the system $\mathcal{C}$ passes the test $\mathcal{T}$ or not. Therefore the current work (studying parallel composition of coalgebras) is a first step towards "coalgebraic testing."

On the mathematical side, one direction is to identify more instances of the microcosm principle. Mathematics abounds with the (often implicit) idea of nested algebraic structures. To name a few: a topological space in a topos which is itself a "generalized topological space"; a category of domains which itself carries a "structure as a domain." We wish to turn such an informal statement into a mathematically rigorous one, by generalizing the current formalization of the microcosm principle. As a possible first step towards this direction, we are working on formalizing the microcosm principle for finitary monads which are known to be roughly the same thing as Lawvere theories.

Another direction is a search for n-folded nested algebraic structures. In the current work we have concentrated on two levels of interpretation. We are not (yet)

[12] The word "testing" here means something different from the one in *testing situations* (Section 2.5).

aware of examples with three or more levels; an example might be an internal category in an internal category.

The *simplicial category* Δ [96, Section VII.5] can be thought of as a "universal" microcosm model for the theory **Mon** of monoids, in the sense of [96, Proposition VII.5.1]. Further investigation is needed to make clear in which exact sense it is universal; and to generalize the construction of such a universal model to an arbitrary algebraic theory $\mathbb{L}$. The construction of Δ seems much related to the construction called *categorification* [8].

Appendix A

Preliminaries

A.1 Initial/final sequences

Here we recall the standard construction [5] of an initial algebra (or a final coalgebra) via the initial (or final) sequence. Notice that the base category need not be **Sets**.

Let $\mathbb{C}$ be a category with an initial object 0, and $F : \mathbb{C} \to \mathbb{C}$ be an endofunctor. The *initial sequence*[1] of F is a diagram

$$\begin{tikzcd}
0 \arrow[r, "¡"] & F0 \arrow[r, "F\,¡"] & \cdots \arrow[r, "F^{n-1}\,¡"] & F^n 0 \arrow[r, "F^n\,¡"] & \cdots
\end{tikzcd}$$

where $¡ : 0 \to X$ is the unique arrow.

Now assume that:

- the initial sequence has an ω-colimit[2] $(F^n 0 \xrightarrow{\alpha_n} A)_{n<\omega}$;
- the functor F preserves that ω-colimit.

Then we have two cocones $(\alpha_n)_{n<\omega}$ and $(F\alpha_{n-1})_{n<\omega}$ over the initial sequence. Moreover, the latter is again a colimit: hence we have mediating isomorphisms between these cones.

$$\begin{tikzcd}
 & & & & A \arrow[dd, bend left, "\alpha^{-1}"] & \text{(colimit)} \\
\cdots \arrow[r] & F^n 0 \arrow[r, "F^n\,¡"] \arrow[urrr, bend left, "\alpha_n"] \arrow[drrr, bend right, "F\alpha_{n-1}"'] & F^{n+1} 0 \arrow[r] \arrow[urr, "\alpha_{n+1}"] \arrow[drr, "F\alpha_n"'] & \cdots & \cong & \\
 & & & & FA \arrow[uu, bend left, "\alpha"] & \text{(colimit)}
\end{tikzcd}$$

A.1.1 Proposition. *The F-algebra $\alpha : FA \overset{\cong}{\Rightarrow} A$ is initial.*

[1] In this thesis we consider only initial/final sequences of length ω.

[2] An ω-colimit is a colimit of a diagram whose shape is the ordinal ω.

Proof. For future reference we prove the dual result: see Proposition A.1.2. □

The dual of this construction yields a final F-coalgebra. Assume that the base category $\mathbb{C}$ has a terminal object 1. The *final sequence* of F is

$$\begin{tikzcd}
1 & F1 \arrow[l, "!"'] & \cdots \arrow[l, "F!"'] & & F^n 1 \arrow[l, "F^{n-1}!"'] & \cdots \arrow[l, "F^n !"'] ,
\end{tikzcd}$$

where $! : X \to 1$ is the unique arrow. Assume that it has an ω^{op}-limit $(Z \xrightarrow{\zeta_n} F^n 1)_{n<\omega}$, and also that F preserves that ω^{op}-limit. We have the following situation.

$$\begin{tikzcd}
 & & & & Z \arrow[dlll, bend right, "\zeta_n"'] \arrow[dll, "\zeta_{n+1}"'] \arrow[dd, bend left, "\zeta"] \arrow[dd, phantom, "\cong"] & \text{(limit)} \\
\cdots & F^n 1 \arrow[l] & F^{n+1} 1 \arrow[l, "F^n !"'] & \cdots \arrow[l] & & \\
 & & & & FZ \arrow[ulll, bend left, "F\zeta_{n-1}"] \arrow[ull, "F\zeta_n"] \arrow[uu, bend left, "\zeta^{-1}"] & \text{(limit)}
\end{tikzcd}$$

A.1.2 Proposition. *The coalgebra $\zeta : Z \overset{\cong}{\to} FZ$ is final.*

Proof. Any F-coalgebra $c : X \to FX$ induces a cone $(X \xrightarrow{\gamma_n} F^n 1)_{n<\omega}$ over the final sequence in the following way.

$$\gamma_0 = \; ! : X \longrightarrow 1 \ , \qquad \gamma_{n+1} = F\gamma_n \circ c \ .$$

Now we can prove the following: for an arrow $f : X \to Z$, f is a morphism of coalgebras from c to ζ if and only if f is a mediating arrow from the cone $(\gamma_n)_{n<\omega}$ to the limit $(\zeta_n)_{n<\omega}$. Hence such a morphism of coalgebras uniquely exists. □

It is easy to see that every shapely functor in **Sets** preserves ω-colimits and ω^{op}-limits. Hence we have the following.

A.1.3 Lemma. *A shapely functor F has both an initial algebra and a final coalgebra in* **Sets**. □

A.2 Limit-colimit coincidence

We recall some relevant notions and results from [127]. The idea is that in a suitable order-enriched setting, (co)limits are equivalently described as an order-theoretic notion of **O**-*(co)limits*. Due to the inherent coincidence between **O**-limits and **O**-colimits, we also obtain the so-called *limit-colimit coincidence*.

$$\begin{tikzcd}
\text{limit} \arrow[d, equal] & \text{colimit} \arrow[d, equal] \\
\mathbf{O}\text{-limit} \arrow[r, dash, "\text{obvious coincidence}"'] & \mathbf{O}\text{-colimit}
\end{tikzcd}$$

The notions of **O**-(co)limits are stated in terms of *embedding-projection pairs* which we can define in an order-enriched category. In the sequel we assume the **Cppo**-enriched structure.

A.2.1 Definition (Embedding-projection pairs)**.** Let $\mathbb{C}$ be a **Cppo**-enriched category. A pair of arrows

$$\begin{tikzcd} X \arrow[r, tail, bend left, "e"] & Y \arrow[l, two heads, bend left, "p"] \end{tikzcd}$$

in $\mathbb{C}$ is said to be an *embedding-projection pair* if we have $p \circ e = \mathrm{id}$ and $e \circ p \sqsubseteq \mathrm{id}$. Diagrammatically presented,

$$\begin{tikzcd} X \arrow[r, tail, "e"] \arrow[dr, "\mathrm{id}"'] \arrow[dr, phantom, shift left=3, "/\!/"] & Y \arrow[d, two heads, "p"] \arrow[dr, "\mathrm{id}"] \arrow[dr, phantom, shift right=3, "\sqsubseteq"] & \\ & X \arrow[r, tail, "e"'] & Y. \end{tikzcd}$$

By $p \circ e = \mathrm{id}$ we automatically have that e is a mono and p is an epi. Both split.

A.2.2 Proposition. *Let $(e, p), (e', p') : X \rightleftarrows Y$ be two embedding-projection pairs with the same domain and codomain. Then $e \sqsubseteq e'$ holds if and only if $p' \sqsubseteq p$. As a consequence, one component of an embedding-projection pair determines the other.* □

This proposition justifies the notation e^P for the projection corresponding to a given embedding e, and p^E for the embedding corresponding to a given projection p. It is easy to check that

$$(e \circ f)^P = f^P \circ e^P \qquad \text{and} \qquad (p \circ q)^E = q^E \circ p^E \ .$$

A.2.3 Definition (**O**-(co)limits)**.** Let $X_0 \xrightarrow{f_0} X_1 \xrightarrow{f_1} \cdots$ be an ω-chain in a **Cppo**-enriched $\mathbb{C}$. A cocone $(X_n \xrightarrow{\sigma_n} C)_{n<\omega}$ over this chain is said to be an **O**-*colimit* if:

- each σ_n is an embedding;
- the sequence of arrows $(\, C \overset{\sigma_n^P}{\twoheadrightarrow} X_n \overset{\sigma_n}{\rightarrowtail} C\,)_{n<\omega}$ is increasing. Moreover its join taken in the cpo $\mathbb{C}(C, C)$ is id_C.

$$\begin{tikzcd} & C \arrow[dl, two heads, bend left=10, "\sigma_0^P"] \arrow[d, two heads, bend left, "\sigma_1^P"] & \cdots \\ X_0 \arrow[ur, tail, bend left=30, "\sigma_0"] \arrow[r, "f_0"'] & X_1 \arrow[u, tail, bend left, "\sigma_1"] \arrow[r, "f_1"'] & \cdots \end{tikzcd}$$

Dually, a cone $(C \xrightarrow{\gamma_n} Y_n)_{n<\omega}$ over an ω^{op}-chain $Y_0 \xleftarrow{g_0} Y_1 \xleftarrow{g_1} \cdots$ is an **O**-*limit* if: each γ_n is a projection, and the sequence $(\gamma_n^E \circ \gamma_n : C \to C)_{n<\omega}$ is increasing and its join is id_C.

The following proposition establishes the equivalence between (co)limits and **O**-(co)limits. For its full proof the reader is referred to [127].

A.2.4 Proposition (Propositions A–D in [127])**.** *Let $X_0 \overset{e_0}{\to} X_1 \overset{e_1}{\to} \cdots$ be an ω-chain where each e_n is an embedding.*

1. *Let $(X_n \overset{\sigma_n}{\to} C)_{n<\omega}$ be the colimit over the chain. Then each σ_n is also an embedding. Moreover, $(\sigma_n)_{n<\omega}$ is an $\mathbf{O}$-colimit.*
2. *Conversely, an $\mathbf{O}$-colimit $(X_n \overset{\sigma_n}{\to} C)_{n<\omega}$ over the chain is a colimit.*

Dually, let $X_0 \overset{p_0}{\leftarrow} X_1 \overset{p_1}{\leftarrow} \cdots$ be an ω^{op}-chain where each p_n is a projection.

3. *Let $(D \overset{\tau_n}{\to} X_n)_{n<\omega}$ be a limit over the chain. Then each τ_n is also a projection. Moreover $(\tau_n)_{n<\omega}$ is an $\mathbf{O}$-limit.*
4. *Conversely, an $\mathbf{O}$-limit $(D \overset{\tau_n}{\to} X_n)_{n<\omega}$ over the chain is a limit.*

Proof. For later reference we present the proof of (4). Let $(B \overset{\beta_n}{\to} X_n)_{n<\omega}$ be an arbitrary cone over the chain $X_0 \overset{p_0}{\leftarrow} X_1 \overset{p_1}{\leftarrow} \cdots$. First we prove the uniqueness of a mediating map $f : B \to D$.

$$\begin{aligned}
f = \mathrm{id}_D \circ f &= \big(\textstyle\bigsqcup_{n<\omega}(\tau_n^E \circ \tau_n)\big) \circ f && (\tau_n)_{n<\omega} \text{ is an } \mathbf{O}\text{-limit} \\
&= \textstyle\bigsqcup_{n<\omega}(\tau_n^E \circ \tau_n \circ f) && \text{composition is continuous} \\
&= \textstyle\bigsqcup_{n<\omega}(\tau_n^E \circ \beta_n) && f \text{ is mediating } .
\end{aligned}$$

We conclude the proof by showing that the sequence $(\tau_n^E \circ \beta_n)_{n<\omega}$ is increasing, hence such f indeed exists.

$$\tau_n^E \circ \beta_n \;=\; \tau_n^E \circ p_n \circ \beta_{n+1} \;=\; \tau_{n+1}^E \circ p_n^E \circ p_n \circ \beta_{n+1} \;\sqsubseteq\; \tau_{n+1}^E \circ \beta_{n+1}$$

The last inequality holds because $p_n^E \circ p_n \sqsubseteq \mathrm{id}$ from the definition of embedding-projection pairs. □

A.2.5 Theorem (Limit-colimit coincidence)**.** *Let $X_0 \overset{e_0}{\to} X_1 \overset{e_1}{\to} \cdots$ be an ω-chain where each e_n is an embedding, and $(X_n \overset{\sigma_n}{\to} C)_{n<\omega}$ be the colimit over the chain. Then each σ_n is an embedding, and the cone $(C \overset{\sigma_n^P}{\to} X_n)_{n<\omega}$ is a limit over the ω^{op}-chain $X_0 \overset{e_0^P}{\leftarrow} X_1 \overset{e_1^P}{\leftarrow} \cdots$.*

$$\begin{tikzcd}
 & & C & & & & & C \arrow[dll, two heads, bend right, "\sigma_0^P"'] \arrow[dl, two heads, "\sigma_1^P"] & \\
X_0 \arrow[r, tail, "e_0"'] \arrow[urr, tail, bend left, "\sigma_0"] & X_1 \arrow[r, tail, "e_1"'] \arrow[ur, tail, "\sigma_1"'] & \cdots & : \textit{colimit} & \Longrightarrow & X_0 & X_1 \arrow[l, two heads, "e_0^P"] & \cdots \arrow[l, two heads, "e_1^P"] & : \textit{limit}
\end{tikzcd}$$

Dually, the limit of an ω^{op}-chain of projections consists of projections. By taking the corresponding embeddings we obtain a colimit of an ω-chain of embeddings.

Proof. We prove the first statement. By Proposition A.2.4 each σ_n is an embedding, and moreover $(\sigma_n)_{n<\omega}$ is an **O**-colimit. Now obviously $(\sigma_n^P)_{n<\omega}$ is a cone over $X_0 \overset{e_0^P}{\leftarrow} X_1 \overset{e_1^P}{\leftarrow} \cdots$. Here we use the inherent coincidence of **O**-(co)limits: namely, the condition that $(\sigma_n)_{n<\omega}$ is an **O**-colimit is exactly the same as that $(\sigma_n^P)_{n<\omega}$ is an **O**-limit. We use Proposition A.2.4 to conclude the proof. □

Bibliography

[1] M. Abadi and A.D. Gordon. A calculus for cryptographic protocols: The Spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, pp. 36–47. ACM Press, 1997. 100

[2] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D.M. Gabbai and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, vol. 3, pp. 1–168. Oxford Univ. Press, 1994. 35, 43

[3] L. Aceto, W. Fokkink and C. Verhoef. Structural operational semantics. In J. Bergstra, A. Ponse and S. Smolka, editors, *Handbook of Process Algebra*, pp. 197–292. Elsevier, 2001. 112

[4] P. Aczel. *Non-well-founded sets.* CSLI Lecture Notes 14, Stanford, 1988. 9

[5] J. Adámek and V. Koubek. Least fixed point of a functor. *Journ. Comp. Syst. Sci*, 19(2):163–178, 1979. 37, 147

[6] Anonymity bibliography, 2007. `http://freehaven.net/anonbib`. 88

[7] N. Attrapadung, Y. Cui, D. Galindo, G. Hanaoka, I. Hasuo, H. Imai, K. Matsuura, P. Yang and R. Zhang. Relations among notions of security for identity based encryption schemes. In *Latin American Theoretical Informatics Symposium (LATIN '06)*, vol. 3887 of *Lect. Notes Comp. Sci.*, pp. 130–141. Springer, Berlin, 2006.

[8] J.C. Baez and J. Dolan. Categorification. *Contemp. Math.*, 230:1–36, 1998. 146

[9] J.C. Baez and J. Dolan. Higher dimensional algebra III: n-categories and the algebra of opetopes. *Adv. Math*, 135:145–206, 1998. 18, 117

[10] M. Barr. Terminal coalgebras in well-founded set theory. *Theor. Comp. Sci.*, 114(2):299–315, 1993. Corrigendum in *Theor. Comp. Sci.* 124:189–192, 1994. 9

[11] M. Barr and C. Wells. *Toposes, Triples and Theories.* Springer, Berlin, 1985. Available online. 28, 30, 125

[12] F. Bartels. *On generalised coinduction and probabilistic specification formats. Distributive laws in coalgebraic modelling.* PhD thesis, Free Univ. Amsterdam, 2004. 27, 58, 118

[13] J.A. Bergstra and J.W. Klop. ACP$_\tau$: A universal axiom system for process specification. In M. Wirsing and J.A. Bergstra, editors, *Algebraic Methods*, vol. 394 of *Lecture Notes in Computer Science*, pp. 447–463. Springer, 1987. 121

[14] I. Beylin and P. Dybjer. Extracting a proof of coherence for monoidal categories from a proof of normalization for monoids. In S. Berardi and M. Coppo, editors, *TYPES*, vol. 1158 of *Lect. Notes Comp. Sci.*, pp. 47–61. Springer, 1995. 128

[15] M. Bhargava and C. Palamidessi. Probabilistic anonymity. In M. Abadi and L. de Alfaro, editors, *CONCUR 2005*, vol. 3653 of *Lect. Notes Comp. Sci.*, pp. 171–185. Springer, 2005. 87, 88, 90, 92, 93, 96, 99, 100, 112

[16] R. Bird and O. de Moor. *Algebra of Programmming.* Prentice Hall Int. Series in Comput. Sci., 1996. 27, 44, 45

[17] M.M. Bonsangue and A. Kurz. Duality for logics of transition systems. In V. Sassone, editor, *FoSSaCS*, vol. 3441 of *Lect. Notes Comp. Sci.*, pp. 455–469. Springer, 2005. 26, 51, 55

[18] M.M. Bonsangue and A. Kurz. Presenting functors by operations and equations. In L. Aceto and A. Ingólfsdóttir, editors, *FoSSaCS*, vol. 3921 of *Lect. Notes Comp. Sci.*, pp. 172–186. Springer, 2006. 26, 51, 55

[19] F. Borceux. *Handbook of Categorical Algebra*, vol. 50, 51 and 52 of *Encyclopedia of Mathematics.* Cambridge Univ. Press, 1994. 19, 35, 36, 124, 128, 129, 134, 142

[20] S. Briais. ABC bisimulation checker. `http://lamp.epfl.ch/∼sbriais/abc/`, 2003. 112

[21] D. Cancila and F. Honsell. A coalgebraic description of web interactions. In C. Blundo and C. Laneve, editors, *ICTCS*, vol. 2841 of *Lecture Notes in Computer Science*, pp. 271–283. Springer, 2003. 63

[22] K. Chatzikokolakis, C. Palamidessi and P. Panangaden. Probability of error in information-hiding protocols. In *IEEE Computer Security Foundations Symposium (CSF20)*, pp. 341–354. 2007. 88

[23] K. Chatzikokolakis and C. Palamidessi. Probable innocence revisited. *Theor. Comp. Sci.*, 367(1–2):123–138, 2006. 89, 94, 99, 101

[24] K. Chatzikokolakis and C. Palamidessi. Making random choices invisible to the scheduler. In L. Caires and V.T. Vasconcelos, editors, *CONCUR*, vol. 4703 of *Lect. Notes Comp. Sci.*, pp. 42–58. Springer, 2007. 112

[25] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journ. of Cryptology*, 1(1):65–75, 1988. 90

[26] L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices.* PhD thesis, Radboud Univ. Nijmegen, 2006. 58

[27] T. Chothia. Analysing the MUTE anonymous file-sharing system using the pi-calculus. In E. Najm, J.F. Pradat-Peyre and V. Donzeau-Gouge, editors, *FORTE*, vol. 4229 of *Lect. Notes Comp. Sci.*, pp. 115–130. Springer, 2006. 112

[28] T. Chothia. Securing pseudo identities in an anonymous peer-to-peer file-sharing network. In *International Conference on Security and Privacy in Communication Networks (SecureComm 2007).* 2007. 112

[29] T. Chothia, S. Orzan, J. Pang and M.T. Dashti. A framework for automatically checking anonymity with mCRL. In *The 2nd Symposium on Trustworthy Global Computing (TGC 2006)*, Lect. Notes Comp. Sci. 2006. 112

[30] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theor. Comp. Sci.*, 34:83–133, 1984. 145

[31] M.P. Fiore. *Axiomatic Domain Theory in Categories of Partial Maps.* Distinguished Dissertations in Computer Science. Cambridge Univ. Press, 1996. 27, 42, 43

[32] M.P. Fiore. A coinduction principle for recursive data types based on bisimulation. *Inf. & Comp.*, 127(2):186–198, 1996. 58, 63, 70, 77

[33] M.M. Fokkinga. Monadic maps and folds for arbitrary datatypes. *Memoranda Informatica, University of Twente*, 94–28, 1994. 37

[34] P.J. Freyd. Algebraically complete categories. In A. Carboni, M.C. Pedicchio and G. Rosolini, editors, *Como Conference on Category Theory*, no. 1488 in Lect. Notes Math., pp. 95–104. Springer, Berlin, 1991. 27, 42, 43

[35] P.J. Freyd. Remarks on algebraically compact categories. In M.P. Fourman, P.T. Johnstone and A.M. Pitts, editors, *Applications of Categories in Computer Science*, no. 177 in LMS, pp. 95–106. Cambridge Univ. Press, 1992. 27, 42, 43

[36] D. Galindo and I. Hasuo. Security notions for identity based encryption. Cryptology ePrint Archive, Report 2005/253, 2005.

[37] F.D. Garcia, I. Hasuo, W. Pieters and P. van Rossum. Provable anonymity. In R. Küsters and J. Mitchell, editors, *3rd ACM Workshop on Formal Methods in Security Engineering (FMSE05)*, pp. 63–72. ACM Press, Alexandria , VA, U.S.A., Nov. 2005. 88

[38] S.J. Garland, N.A. Lynch and M. Vaziri. *IOA: a language for specifying, programming, and validating distributed systems.* MIT Laboratory for Computer Science, 1997. 86

[39] R.J. van Glabbeek. The linear time–branching time spectrum I; the semantics of concrete, sequential processes. In J.A. Bergstra, A. Ponse and S.A. Smolka, editors, *Handbook of Process Algebra*, chap. 1, pp. 3–99. Elsevier, 2001. Available at `http://boole.stanford.edu/pub/spectrum1.ps.gz`. 21, 26, 54, 59, 86, 100

[40] R.J. van Glabbeek, S.A. Smolka and B. Steffen. Reactive, generative, and stratified models of probabilistic processes. *Inf. & Comp.*, 121:59–80, 1995. 45

[41] J.Y. Halpern and K.R. O'Neill. Anonymity and information hiding in multiagent systems. *Journ. of Computer Security*, 13(3):483–512, 2005. 88, 89, 93, 94, 99, 100

[42] I. Hasuo and B. Jacobs. Coalgebraic trace semantics for probabilistic systems. In P. Mosses, J. Power and M. Seisenberger, editors, *CALCO-jnr Workshop.* 2005. 27

[43] I. Hasuo and B. Jacobs. Context-free languages via coalgebraic trace semantics. In J.L. Fiadeiro, N. Harman, M. Roggenbach and J.J.M.M. Rutten, editors, *International Conference on Algebra and Coalgebra in Computer Science (CALCO'05)*, vol. 3629 of *Lect. Notes Comp. Sci.*, pp. 213–231. Springer, Berlin, 2005. 14, 27, 31, 48, 49

[44] I. Hasuo, B. Jacobs and A. Sokolova. Generic trace theory. In N. Ghani and A.J. Power, editors, *International Workshop on Coalgebraic Methods in Computer Science (CMCS 2006)*, vol. 164 of *Elect. Notes in Theor. Comp. Sci.*, pp. 47–65. Elsevier, Amsterdam, 2006. 14, 27

[45] I. Hasuo, B. Jacobs and A. Sokolova. The microcosm principle and concurrency in coalgebra. In *FoSSaCS*, Lect. Notes Comp. Sci. 2008. To appear. 19

[46] I. Hasuo. Generic forward and backward simulations. In C. Baier and H. Hermanns, editors, *International Conference on Concurrency Theory (CONCUR 2006)*, vol. 4137 of *Lect. Notes Comp. Sci.*, pp. 406–420. Springer, Berlin, 2006. 16

[47] I. Hasuo, B. Jacobs and A. Sokolova. Generic trace semantics via coinduction. *Logical Methods in Comp. Sci.*, 3(4:11), 2007. 14, 27

[48] I. Hasuo, B. Jacobs and T. Uustalu. Categorical views on computations on trees (extended abstract). In L. Arge, C. Cachin, T. Jurdzinski and A. Tarlecki, editors, *ICALP*, vol. 4596 of *Lecture Notes in Computer Science*, pp. 619–630. Springer, 2007.

[49] I. Hasuo and R. Kashima. Kripke completeness of first-order constructive logics with strong negation. *Logic Journal of the IGPL*, 11(6):615–646, 2003.

[50] I. Hasuo and Y. Kawabe. Probabilistic anonymity via coalgebraic simulations. In R. De Nicola, editor, *European Symposium on Programming (ESOP 2007)*, vol. 4421 of *Lect. Notes Comp. Sci.*, pp. 379–394. Springer, 2007. 17, 157

[51] I. Hasuo, Y. Kawabe and H. Sakurada. Probabilistic anonymity via coalgebraic simulations, 2007. Extended version of [50]. Submitted for publication. 17

[52] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journ. ACM*, 32(1):137–161, 1985. 86

[53] O.M. Herescu and C. Palamidessi. Probabilistic asynchronous pi-calculus. In J. Tiuryn, editor, *FoSSaCS*, vol. 1784 of *Lect. Notes Comp. Sci.*, pp. 146–160. Springer, 2000. 112

[54] C. Hermida and B. Jacobs. Structural induction and coinduction in a fibrational setting. *Inf. & Comp.*, 145:107–152, 1998. 38, 133

[55] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985. 27, 121

[56] C.A.R. Hoare. Data refinement in a categorical setting, 1987. Unpublished manuscript. 63, 70, 83

[57] D. Hughes and V. Shmatikov. Information hiding, anonymity and privacy: A modular approach. *Journal of Computer Security*, 12(1):3–36, 2004. 88

[58] J. Hughes and B. Jacobs. Simulations in coalgebra. *Theor. Comp. Sci.*, 327(1-2):71–108, 2004. 42, 63, 81, 82

[59] M. Hyland and A.J. Power. Discrete Lawvere theories and computational effects. *Theor. Comp. Sci.*, 366(1–2):144–162, 2006. 124

[60] B. Jacobs. Semantics of weakening and contraction. *Ann. Pure & Appl. Logic*, 69(1):73–106, 1994. 66

[61] B. Jacobs. *Categorical Logic and Type Theory.* North Holland, Amsterdam, 1999. v, 53, 125

[62] B. Jacobs. Trace semantics for coalgebras. In J. Adámek and S. Milius, editors, *Coalgebraic Methods in Computer Science*, vol. 106 of *Elect. Notes in Theor. Comp. Sci.* Elsevier, Amsterdam, 2004. 26, 27, 33, 48, 49

[63] B. Jacobs and I. Hasuo. Freyd is Kleisli, for Arrows. In C. McBride and T. Uustalu, editors, *Workshop on Mathematically Structured Functional Programming (MSFP 2006)*, eWiC. 2006.

[64] B. Jacobs and J. Hughes. Simulations in coalgebra. In H.P. Gumm, editor, *Coalgebraic Methods in Computer Science*, no. 82(1) in Elect. Notes in Theor. Comp. Sci. Elsevier, Amsterdam, 2003. 63, 81, 82

[65] B. Jacobs and J.J.M.M. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:222–259, 1997. 3, 7, 19, 23, 27

[66] B. Jacobs. Introduction to coalgebra. Towards mathematics of states and observations, 2005. Draft of a book, `http://www.cs.ru.nl/B.Jacobs/PAPERS/`. 3, 4, 19, 23

[67] B. Jacobs. A bialgebraic review of deterministic automata, regular expressions and languages. In K. Futatsugi, J.P. Jouannaud and J. Meseguer, editors, *Essays Dedicated to Joseph A. Goguen*, vol. 4060 of *Lect. Notes Comp. Sci.*, pp. 375–404. Springer, 2006. 27, 50, 118

[68] B. Jacobs and I. Hasuo. Semantics and logic for security protocols. *Journ. of Computer Security*, 2007. To appear.

[69] C.B. Jay. A semantics for shape. *Science of Comput. Progr.*, 25:251–283, 1995. 34

[70] P.T. Johnstone, A.J. Power, T. Tsujishita, H. Watanabe and J. Worrell. An axiomatics for categories of transition systems as coalgebras. In *Logic in Computer Science*. IEEE, Computer Science Press, 1998. 121, 122

[71] 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. 17, 61, 87, 88, 101, 104, 107, 112, 113

[72] Y. Kawabe, K. Mano, H. Sakurada and Y. Tsukada. Theorem-proving anonymity of infinite state systems. *Inform. Process. Lett.*, 101(1):46–51, 2007. 17, 87, 88, 101, 104, 107, 112

[73] G.M. Kelly. *Basic Concepts of Enriched Category Theory.* No. 64 in LMS. Cambridge Univ. Press, 1982. 35

[74] G.M. Kelly. Elementary observations on 2-categorical limits. *Bull. Austr. Math. Soc.*, 39:301–317, 1989. 133

[75] M. Kick, A.J. Power and A. Simpson. Coalgebraic semantics for timed processes. *Inf. & Comp.*, 204(4):588–609, 2006. 25, 27, 118

[76] Y. Kinoshita and J. Power. Data refinement and algebraic structure. *Acta Informatica*, 36:693–719, 2000. 63, 70, 83, 84

[77] N. Klarlund and F.B. Schneider. Verifying safety properties using infinite-state automata. Tech. Rep. 89-1039, Department of Computer Science, Cornell University, Ithaca, New York, 1989. 79

[78] B. Klin. From bialgebraic semantics to congruence formats. In *Workshop on Structural Operational Semantics (SOS 2004)*, vol. 128 of *Elect. Notes in Theor. Comp. Sci.*, pp. 3–37. 2005. 27, 58, 118

[79] B. Klin. Bialgebraic operational semantics and modal logic. In *Logic in Computer Science*, pp. 336–345. IEEE Computer Society, 2007. 26, 27, 118

[80] B. Klin. Coalgebraic modal logic beyond **Sets**. In *MFPS XXIII*, vol. 173, pp. 177–201. Elsevier, Amsterdam, 2007. 26, 51, 53, 54

[81] A. Kock. Monads on symmetric monoidal closed categories. *Arch. Math.*, XXI:1–10, 1970. 33

[82] A. Kock and G.E. Reyes. Doctrines in categorical logic. In J. Barwise, editor, *Handbook of Mathematical Logic*, pp. 283–313. North-Holland, Amsterdam, 1977. 19, 124

[83] C. Kupke, A. Kurz and Y. Venema. Stone coalgebras. *Theor. Comp. Sci.*, 327(1-2):109–134, 2004. 26

[84] C. Kupke, A. Kurz and D. Pattinson. Algebraic semantics for coalgebraic logics. *Elect. Notes in Theor. Comp. Sci.*, 106:219–241, 2004. 51

[85] A. Kurz. *Logics for Coalgebras and Applications to Computer Science*. PhD thesis, Universität München, April 2000. 26

[86] A. Kurz. Coalgebras and modal logic. Course notes for ESSLLI, 2001. `http://www.helsinki.fi/esslli/`. 3, 19

[87] A. Kurz. Coalgebras and their logics. *SIGACT News*, 37(2):57–77, 2006. 26, 51

[88] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. & Comp.*, 94(1):1–28, 1991. 84, 85

[89] F.W. Lawvere. Metric spaces, generalized logic, and closed categories. *Seminario Matematico e Fisico. Rendiconti di Milano*, 43:135–166, 1973. Reprinted in *Theory and Applications of Categories*, 1:1–37, 2002. 35

[90] F.W. Lawvere. *Functorial Semantics of Algebraic Theories and Some Algebraic Problems in the Context of Functorial Semantics of Algebraic Theories*. PhD thesis, Columbia University, 1963. Reprints in Theory and Applications of Categories, 5 (2004) 1–121. 117, 124, 125

[91] E.A. Lee. Making concurrency mainstream. Invited talk at CONCUR 2006, 2006. 115

[92] M. Lenisa, A.J. Power and H. Watanabe. Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads. In H. Reichel, editor, *Coalgebraic Methods in Computer Science*, vol. 33 of *Elect. Notes in Theor. Comp. Sci.* Elsevier, Amsterdam, 2000. 32

[93] M. Lenisa, J. Power and H. Watanabe. Category theory for operational semantics. *Theor. Comp. Sci.*, 327(1–2):135–154, 2004. 32

[94] N. Lynch and F. Vaandrager. Forward and backward simulations. I. Untimed systems. *Inf. & Comp.*, 121(2):214–233, 1995. 15, 61, 62, 71, 74, 79, 82, 89

[95] N.A. Lynch, R. Segala and F.W. Vaandrager. Compositionality for probabilistic automata. In R.M. Amadio and D. Lugiez, editors, *CONCUR 2003*, vol. 2761 of *Lect. Notes Comp. Sci.*, pp. 204–222. Springer, 2003. 84, 85

[96] S. Mac Lane. *Categories for the Working Mathematician.* Springer, Berlin, 2nd edn., 1998. 2, 19, 28, 30, 38, 128, 135, 146

[97] J. McLean. Security models. In J. Marciniak, editor, *Encyclopedia of Software Engineering*, pp. 1136–1145. John Wiley & Sons, 1994. 110

[98] R. Milner. *A Calculus of Communicating Systems.* Lect. Notes Comp. Sci. Springer, Berlin, 1980. 9

[99] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989. 121

[100] E. Moggi. Notions of computation and monads. *Inf. & Comp.*, 93(1):55–92, 1991. 124

[101] O. de Moor. Inductive data types for predicate transformers. *Inform. Process. Lett.*, 43:113–117, 1992. 27, 44, 45

[102] M.R. Mousavi, M.A. Reniers and J.F. Groote. A syntactic commutativity format for SOS. *Inform. Process. Lett.*, 93(5):217–223, 2005. 145

[103] P.S. Mulry. Lifting theorems for Kleisli categories. In *Mathematical Foundations of Programming Semantics (MFPS IX)*, pp. 304–319. Springer-Verlag, London, UK, 1994. 32

[104] K. Nishizawa and A.J. Power. Lawvere theories enriched over a general base. *Journ. of Pure & Appl. Algebra*, 2006. To appear. 124

[105] C. Palamidessi and O.M. Herescu. A randomized encoding of the pi-calculus with mixed choice. *Theor. Comp. Sci.*, 335(2–3):373–404, 2005. 112

[106] A. Pardo. Fusion of recursive programs with computational effects. *Theor. Comp. Sci.*, 260(1–2):165–207, 2001. 27, 37, 44

[107] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proceedings 5th GI Conference on Theoretical Computer Science*, vol. 104 of *Lect. Notes Comp. Sci.*, pp. 15–32. Springer, Berlin, 1981. 9

[108] D. Pattinson. An introduction to the theory of coalgebras. Course notes for NASSLLI, 2003. `http://www.indiana.edu/~nasslli/`. 3, 4, 19

[109] D. Pavlović, M. Mislove and J.B. Worrell. Testing semantics: connecting processes and process logics. In M. Johnson and V. Vene, editors, *Algebraic Methodology and Software Technology (AMAST 2006)*, vol. 4019 of *Lect. Notes Comp. Sci.* Springer, 2006. 26, 51

[110] A. Pfitzmann and M. Köhntopp. Anonymity, unobservability, and pseudonymity: A proposal for terminology. Draft, version 0.17, July 2000. 88

[111] J. Power and D. Turi. A coalgebraic foundation for linear time semantics. In *Category Theory and Computer Science*, vol. 29 of *Elect. Notes in Theor. Comp. Sci.* Elsevier, Amsterdam, 1999. 26, 27

[112] M.K. Reiter and A.D. Rubin. Crowds: anonymity for Web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998. 88, 89, 90, 93, 94, 99

[113] J. Rohrer. MUTE technical details. `http://mutenet.sourceforge.net`, 2006. 112

[114] J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comp. Sci.*, 249:3–80, 2000. 3, 11, 19, 23, 27

[115] J.J.M.M. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theor. Comp. Sci.*, 308:1–53, 2003. 50

[116] J.J.M.M. Rutten and D. Turi. Initial algebra and final coalgebra semantics for concurrency. In J.W. de Bakker, W.P. de Roever and G. Rozenberg, editors, *A Decade of Concurrency*, no. 803 in Lect. Notes Comp. Sci., pp. 530–582. Springer, Berlin, 1994. 26, 27, 49, 118

[117] A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, pp. 200–214. 2000. 111, 112

[118] D. Sangiorgi. On the origins of bisimulation, coinduction, and fixed points. Tech. rep., Department of Computer Science, University of Bologna, 2007. 7

[119] S. Schneider and A. Sidiropoulos. CSP and anonymity. In *ESORICS '96: Proceedings of the 4th European Symposium on Research in Computer Security*, pp. 198–218. Springer-Verlag, London, UK, 1996. 17, 88, 100

[120] R. Segala. *Modeling and verification of randomized distributed real-time systems.* PhD thesis, MIT, 1995. 58, 61, 84, 85

[121] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journ. Comput.*, 2(2):250–273, 1995. 95, 112

[122] R. Segala. A compositional trace-based semantics for probabilistic automata. In *International Conference on Concurrency Theory (CONCUR '95)*, pp. 234–248. Springer-Verlag, 1995. 21

[123] A. Serjantov. *On the Anonymity of Anonymity Systems.* PhD thesis, University of Cambridge, March 2004. 88

[124] V. Shmatikov. Probabilistic model checking of an anonymity system. *Journ. of Computer Security*, 12(3):355–377, 2004. 88

[125] A.K. Simpson. Recursive types in Kleisli categories, 1992. Unpublished paper, available at `http://homepages.inf.ed.ac.uk/als/Research/`. 27, 42

[126] G. Smith and D.M. Volpano. Secure information flow in a multi-threaded imperative language. In *POPL*, pp. 355–364. 1998. 111

[127] M.B. Smyth and G.D. Plotkin. The category theoretic solution of recursive domain equations. *SIAM Journ. Comput.*, 11:761–783, 1982. 24, 27, 37, 148, 149, 150

[128] A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems.* PhD thesis, Techn. Univ. Eindhoven, 2005. 45, 84

[129] C. Stirling. Bisimulation and language equivalence. In R.J.G.B. de Queiroz, editor, *Logic for concurrency and synchronisation*, vol. 18 of *Trends in Logic*, pp. 269–284. Kluwer Academic Publishers, Norwell, MA, USA, 2003. 12

[130] M. Stoelinga and F.W. Vaandrager. A testing scenario for probabilistic automata. In J.C.M. Baeten, J.K. Lenstra, J. Parrow and G.J. Woeginger, editors, *ICALP*, vol. 2719 of *Lect. Notes Comp. Sci.*, pp. 464–477. Springer, 2003. 26

[131] R. Street. Fibrations and Yoneda's lemma in a 2-category. In G.M. Kelly, editor, *Proc. Sydney Category Theory Seminar 1972/1973*, no. 420 in Lect. Notes Math., pp. 104–133. Springer, Berlin, 1974. 133

[132] R. Tix, K. Keimel and G.D. Plotkin. Semantic domains for combining probability and non-determinism. *Elect. Notes in Theor. Comp. Sci.*, 129:1–104, 2005. 58

[133] V. Trnková. General theory of relational automata. *Fund. Informaticae*, 3:189–233, 1980. 33

[134] D. Turi and G. Plotkin. Towards a mathematical operational semantics. In *Logic in Computer Science*, pp. 280–291. IEEE, Computer Science Press, 1997. 27, 58, 118, 122

[135] D. Turi and J.J.M.M. Rutten. On the foundations of final semantics: non-standard sets, metric spaces and partial orders. *Math. Struct. in Comp. Sci.*, 8(5):481–540, 1998. 58, 63

[136] D. Varacca and G. Winskel. Distributing probabililty over nondeterminism. *Math. Struct. in Comp. Sci.*, 16(1):87–113, 2006. 58, 85

[137] M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *FOCS '85*, pp. 327–338. 1985. 58

[138] D.M. Volpano and G. Smith. Probabilistic noninterference in a concurrent language. *Journ. of Computer Security*, 7(1), 1999. 111, 112

[139] H. Watanabe, K. Nishizawa and O. Takaki. A coalgebraic representation of reduction by cone of influence. *Elect. Notes in Theor. Comp. Sci.*, 164(1):177–194, 2006. 63

Index

Summary

The aim of this thesis is to obtain better understanding of the nature of *computer systems*, that is, to develop a *mathematical theory of computer systems.* By computer systems we mean information-processing systems consisting of computers. We cannot overemphasize their important roles in the modern world. Unfortunately, however, computer systems are very much error-prone at the same time. This thesis ultimately aims at proper understanding of computer systems which will help us getting them right; we do so through extending the theory of *coalgebras.*

The use of coalgebras as a mathematical model of state-based systems has been increasingly recognized in the last couple of decades. We follow this view since, we believe, the theory of coalgebras has a right balance of (potential) applicability and mathematical simplicity. The very core of the existing theory of coalgebras that we start with can be summarized as follows.

- A system is modeled mathematically as a *coalgebra*;
- a behavior-preserving map between systems as a *morphism of coalgebras*; and
- the behavior of a system is given by the principle of *coinduction.*

These points have been established by many authors in a large body of existing work. In each chapter of this thesis, however, we try to shed new light on them.

Our most emphasis is on, among the three points listed above, the principle of coinduction which assigns to a system (i.e. a coalgebra) its behavior in the form of the unique map to a final coalgebra. In the standard setting where the base category is **Sets**, the behavior thus captured respects *bisimilarity*—which is one possible choice in the spectrum of many different notions of behavioral equivalences. In Chapter 2 we establish an appropriate mathematical framework to capture a different notion of behavior—namely *trace semantics*—also by coinduction. To put it briefly, when we move the base category from **Sets** to a suitable *Kleisli category*, then the notion of

behaviors captured by coinduction shifts from bisimilarity to trace semantics. This works equally for both (non-deterministic) trace *set* semantics and (probabilistic) trace *distribution* semantics.

Then a natural question arises: what is a morphism of coalgebras in a Kleisli category? In Chapter 3 it is given an interpretation as a *forward/backward simulation*. Use of simulations is an important technique in formal verification of systems, in which the *soundness theorem* plays a crucial role. Our categorical formulation of simulations allows us to prove a *generic soundness theorem* which works for a wide variety of systems (e.g. non-deterministic or probabilistic) and whose proof is once-for-all by simple categorical arguments. This—together with Chapter 2—establishes a *generic, coalgebraic theory of traces and simulations*.

It is this generic theory which is exploited in Chapter 4 to verify *anonymity* of network protocols. More specifically, we start with an existing simulation-based proof method for *non-deterministic* anonymity; we make it *probabilistic* to obtain a proof method for probabilistic anonymity. This technical jump from non-determinism to probability is minor for us since, in the generic theory of traces and simulations established earlier in Chapter 2–3, different types of branching (non-determinism vs. probability) are modeled exactly in the same manner, only by different parameters. In other words, what one can do in a non-deterministic setting can be also done in a probabilistic setting.

Finally, in Chapter 5 we deal with the issue of *concurrency* which is a major challenge in the study of computer systems. In our coalgebraic study of systems, *compositionality*—compatibility of behavior and concurrency—is one of our first concerns. An attempt to formulate compositionality in coalgebraic terms reveals significance of *higher-dimensional* algebraic structures, the so-called *microcosm principle*. This is how we come to a highly abstract mathematical expedition of formalizing the microcosm principle in 2-categories. It is more than just an abstract nonsense: we apply the obtained formalization to the original coalgebraic setting and derive a *generic compositionality theorem*. It automatically ensures compositionality for a certain class of concurrency situations.

Samenvatting (Dutch summary)

Het doel van dit proefschrift is om een beter inzicht in de aard van *computersystemen* te bereiken, dat wil zeggen, om een *wiskundige theorie van computersystemen* te ontwikkelen. Met computersystemen bedoelen we informatieverwerkende systemen die bestaan uit computers. Hun belangrijke rol in de moderne wereld is nauwelijks te overdrijven. Tegelijkertijd zijn computersystemen helaas echter zeer bevattelijk voor fouten. Door hun karakter goed te doorzien, draagt dit proefschrift uiteindelijk bij aan een foutloze constructie van computersystemen; we doen dit door de theorie van *coalgebras* uit te breiden.

Het gebruik van coalgebras is de afgelopen decennia opgeëist als een wiskundig model van toestandsystemen. Wij hangen deze visie aan omdat, volgens ons, de theorie van coalgebras de juiste balans vindt tussen (mogelijke) toepasbaarheid en wiskundige eenvoud. Het hart van de bestaande theorie van coalgebras waar we mee beginnen kan als volgt worden samengevat.

– Een systeem wordt wiskundig gemodelleerd als een *coalgebra*;

– een gedragsrespecterende afbeelding tussen systemen als een *morfisme van coalgebras*; en

– het gedrag van een systeem wordt vastgelegd door het principe van *coinductie.*

Deze visie is bewerkstelligd door vele auteurs, in een groot aantal bestaande werken. Desondanks probeert ieder hoofdstuk van dit proefschrift om er nieuw licht op te werpen.

Van de drie bovengenoemde punten leggen we de meeste nadruk op het principe van coinductie, dat van ieder systeem (coalgebra) haar gedrag vastlegt in de vorm van een unieke afbeelding naar een finale coalgebra. In de gebruikelijke situatie, waar de basiscategorie **Sets** is, respecteert het aldus vastgelegde gedrag *bisimilariteit*—wat trouwens slechts een van de mogelijke keuzes is uit een spectrum van gedragsequivalenties. Hoofdstuk 2 bewerkstelligt een passend wiskundig raamwerk om ook een andere notie van gedrag—namelijk *trace semantiek*—door coinductie te vangen. Kort gezegd: als we de basiscategorie van **Sets** naar een geschikte *Kleisli categorie* verplaatsen, dan verandert de notie van gedrag zoals vastgelegd door coinductie van bisimilariteit naar trace semantiek. Dit werkt net zo goed voor (non-deterministische) trace *verzamelings*semantiek als voor (probabilistische) trace *distributie*semantiek.

Een natuurlijke vraag werpt zich dan op: wat is een morfisme van coalgebras in een Kleisli categorie? In hoofdstuk 3 wordt dit ingevuld als een interpretatie van een *voorwaarts/achterwaarts simulatie.* Het gebruik van simulaties is een belangrijke techniek in formele verificatie van systemen, waar *soundnessstellingen* een cruciale rol spelen. Onze categorische formulering van simulaties stelt ons in staat om een *generieke soundnessstelling* te bewijzen, die opgaat voor een breed scala aan systemen (zoals non-deterministische en probabilistische), en het bewijs waarvan eens-en-voor-al is door eenvoudige categorische argumenten. Dit—gecombineerd met Hoofdstuk 2—geeft een *generieke, coalgebraïsche theorie van traces en simulaties.*

Het is deze generieke theorie die we inzetten in Hoofdstuk 4 om *anonimiteit* van netwerkprotocollen te verifiëren. Preciezer gezegd beginnen we met een bestaande bewijsmethode voor *non-deterministische* anonimiteit gebaseerd op simulaties, en maken haar *probabilistisch* om een bewijsmethode te krijgen voor probabilistische anonimiteit. Deze technische sprong van non-determinisme naar probabilisme is voor ons een kleine, omdat in de generieke theorie van traces en simulaties van Hoofdstuk 2–3, verschillende soorten vertakking (non-determinisme en probabilisme) op precies dezelfde manier gemodelleerd worden, op verschillende parameters na. Anders gezegd, wat met non-determinisme bereikt kan worden, kan ook probabilistisch gedaan worden.

Hoofdstuk 5 behandelt tenslotte de kwestie van *concurrency*, een grote uitdaging binnen de studie van computer systemen. In onze coalgebraïsche studie van systemen is *compositionaliteit*—compatibiliteit van gedrag en concurrency—een van de eerste aandachtspunten. Een poging om compositionaliteit in coalgebraïsche termen te formuleren onthult het belang van *hoger-dimensionale* algebraïsche structuren, het zogenaamde *microcosmos principe.* Zo komen we tot een zeer abstracte wiskundige expeditie om het microcosmos principe te formalizeren met 2-categorieen. Dit is meer dan enkel abstracte onzin: we passen de verkregen formalizatie toe op de originele coalgebraïsche situatie en leiden zo een *generieke compostionaliteitsstelling* af. Deze verzekert compositionaliteit voor een bepaalde klasse van situaties met concurrency.

Curriculum vitae

Born on 27 September 1978 in Omuta, Japan

1994–1997 La Salle High School, Kagoshima, Japan

1997–2002 BSc in Mathematics, University of Tokyo, Japan

2002–2004 MSc in Mathematical and Computing Sciences, Tokyo Institute of Technology, Japan. Thesis title: *Kripke Completeness of First-Order Constructive Logics with Strong Negation*, supervised by dr. Ryo Kashima

2004–2008 PhD student in Security of Systems group, Institute for Computing and Information Sciences, Radboud University Nijmegen, the Netherlands

2007– Assistant Professor, Research Institute for Mathematical Sciences, Kyoto University, Japan

2007– Researcher, PRESTO Research Promotion Program, Japan Science and Technology Agency

Titles in the IPA Dissertation Series since 2002

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects.* Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems.* Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity.* Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems.* Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing.* Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems.* Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing.* Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms.* Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining.* Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra.* Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL.* Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand.* Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component composition and construction.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks.* Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing.* Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions.* Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components.* Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. *The λ Abroad – A Functional Approach to Software Components.* Faculty of Mathematics and Computer Science, UU. 2003-11

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method.* Faculty of Mathematics and Computer Science, TU/e. 2004-01

G.I. Jojgov. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving.* Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. *Theory of Molecular Computing – Splicing and Membrane systems.* Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation.* Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments.* Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications.* Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications.* Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems.* Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs.* Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell.* Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation.* Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets.* Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics.* Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station.* Faculty of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents.* Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures.* Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra.* Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling.* Faculty of Mechanical Engineering, TU/e. 2004-21

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support- .* Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue.* Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms.* Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications.* Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures.* Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

I. Kurtev. *Adaptability of Model Transformations.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. *Computational Aspects of Treewidth - Lower Bounds and Network*

Reliability. Faculty of Science, UU. 2005-09

O. Tveretina. *Decision Procedures for Equality Logic with Uninterpreted Functions*. Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. *Evolution of Finite Populations in Dynamic Environments*. Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. *Data Mining using Genetic Programming: Classification and Symbolic Regression*. Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. *Top Quality Type Error Messages*. Faculty of Science, UU. 2005-13

G.F. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. Faculty of Science, Mathematics and Computer Science, RU. 2005-14

M.R. Mousavi. *Structuring Structural Operational Semantics*. Faculty of Mathematics and Computer Science, TU/e. 2005-15

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. Faculty of Mathematics and Computer Science, TU/e. 2005-16

T. Gelsema. *Effective Models for the Structure of pi-Calculus Processes with Replication*. Faculty of Mathematics and Natural Sciences, UL. 2005-17

P. Zoeteweij. *Composing Constraint Solvers*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and Rewriting*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

M.Valero Espada. *Modal Abstraction and Replication of Processes with Data*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. *Stepping through Haskell*. Faculty of Science, UU. 2005-21

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model*. Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems*. Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

M. Kyas. *Verifying OCL Specifications of UML Models: Tool Support and Compositionality*. Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. *Model Checking Timed Automata - Techniques and Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. *Böhm-Like Trees for Rewriting.* Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. *On JML: topics in tool-assisted verification of JML programs.* Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. *Towards Hybrid Molecular Simulations.* Faculty of Biomedical Engineering, TU/e. 2006-09

S.G.R. Nijssen. *Mining Structured Data.* Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. *Separation and Adaptation of Concerns in a Shared Data Space.* Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices.* Faculty of Science, Mathematics and Computer Science, RU. 2006-12

B. Badban. *Verification techniques for Extensions of Equality Logic.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2006-13

A.J. Mooij. *Constructive formal methods and protocol standardization.* Faculty of Mathematics and Computer Science, TU/e. 2006-14

T. Krilavicius. *Hybrid Techniques for Hybrid Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-15

M.E. Warnier. *Language Based Security for Java and JML.* Faculty of Science, Mathematics and Computer Science, RU. 2006-16

V. Sundramoorthy. *At Home In Service Discovery.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-17

B. Gebremichael. *Expressivity of Timed Automata Models.* Faculty of Science, Mathematics and Computer Science, RU. 2006-18

L.C.M. van Gool. *Formalising Interface Specifications.* Faculty of Mathematics and Computer Science, TU/e. 2006-19

C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols.* Faculty of Mathematics and Computer Science, TU/e. 2006-20

J.V. Guillen Scholten. *Mobile Channels for Exogenous Coordination of Distributed Systems: Semantics, Implementation and Composition.* Faculty of Mathematics and Natural Sciences, UL. 2006-21

H.A. de Jong. *Flexible Heterogeneous Software Systems.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

N.K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

M. van Veelen. *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems.* Faculty of Mathematics and Computing Sciences, RUG. 2007-03

T.D. Vu. *Semantics and Applications of Process and Program Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

L. Brandán Briones. *Theories for Model-based Testing: Real-time and*

Coverage. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05

I. Loeb. *Natural Deduction: Sharing by Presentation.* Faculty of Science, Mathematics and Computer Science, RU. 2007-06

M.W.A. Streppel. *Multifunctional Geometric Data Structures.* Faculty of Mathematics and Computer Science, TU/e. 2007-07

N. Trčka. *Silent Steps in Transition Systems and Markov Chains.* Faculty of Mathematics and Computer Science, TU/e. 2007-08

R. Brinkman. *Searching in encrypted data.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-09

A. van Weelden. *Putting types to good use.* Faculty of Science, Mathematics and Computer Science, RU. 2007-10

J.A.R. Noppen. *Imperfect Information in Software Development Processes.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-11

R. Boumen. *Integration and Test plans for Complex Manufacturing Systems.* Faculty of Mechanical Engineering, TU/e. 2007-12

A.J. Wijs. *What to do Next?: Analysing and Optimising System Behaviour in Time.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2007-13

C.F.J. Lange. *Assessing and Improving the Quality of Modeling: A Series of Empirical Studies about the UML.* Faculty of Mathematics and Computer Science, TU/e. 2007-14

T. van der Storm. *Component-based Configuration, Integration and Delivery.* Faculty of Natural Sciences, Mathematics, and Computer Science,UvA. 2007-15

B.S. Graaf. *Model-Driven Evolution of Software Architectures.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2007-16

A.H.J.Mathijssen. *Logical Calculi for Reasoning with Binding.* Faculty of Mathematics and Computer Science, TU/e. 2007-17

D. Jarnikov. *QoS framework for Video Streaming in Home Networks.* Faculty of Mathematics and Computer Science, TU/e. 2007-18

M. A. Abam. *New Data Structures and Algorithms for Mobile Data.* Faculty of Mathematics and Computer Science, TU/e. 2007-19

W.Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy.* Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS.* Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in*

Source Code. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems.* Faculty of Mechanical Engineering, TU/e. 2008-05

M. Bravenboer. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates.* Faculty of Science, UU. 2008-06

M. Torabi Dashti. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

I.S.M. de Jong. *Integration and Test Strategies for Complex Manufacturing Machines.* Faculty of Mechanical Engineering, TU/e. 2008-08

I. Hasuo. *Tracing Anonymity with Coalgebras.* Faculty of Science, Mathematics and Computer Science, RU. 2008-09