Modular Construction of Modal Logics

Corina Cîrstea[1] and Dirk Pattinson[2]

[1] School of Electronics and Computer Science, University of Southampton, UK
cc2@ecs.soton.ac.uk
[2] Institut für Informatik, LMU München, Germany
pattinso@informatik.uni-muenchen.de

Abstract. We present a modular approach to defining logics for a wide variety of state-based systems. We use coalgebras to model the behaviour of systems, and modal logics to specify behavioural properties of systems. We show that the syntax, semantics and proof systems associated to such logics can all be derived in a modular way. Moreover, we show that the logics thus obtained inherit soundness, completeness and expressiveness properties from their building blocks. We apply these techniques to derive sound, complete and expressive logics for a wide variety of probabilistic systems.

1 Introduction

Modularity has been a key concern in software engineering since the conception of the discipline [21]. This paper investigates modularity not in the context of building software systems, but in connection with specifying and reasoning about systems. Our work focuses on reactive systems, which are modelled as coalgebras over the category of sets and functions. The coalgebraic approach provides a uniform framework for modelling a wide range of state-based and reactive systems [27]. Furthermore, coalgebras provide models for a large class of probabilistic systems, as shown by the recent survey [3], which discusses the coalgebraic modelling of eight different types of probabilistic systems.

In the coalgebraic approach, a system consists of a state space C and a function $\gamma : C \to TC$, which maps every state $c \in C$ to the observations $\gamma(c)$ which can be made of c after one transition step. Different types of systems can then be represented in the by varying the type T of observations. A closer look at the coalgebraic modelling of state based and reactive systems reveals that in nearly all cases of interest, the type T of observations arises as the composition of a small number of basic constructs.

The main goal of this paper is to lift this compositionality at the level of observations to the level of specification languages and proof systems. That is, we associate a specification language and a proof system to every basic construct and show, how to obtain specification languages and proof systems for a combination of constructs in terms of the ingredients of the construction. Our main technical contribution is the study of the properties, which are preserved by a combination of languages and proof systems. On the side of languages, we isolate a property which ensures that combined languages are expressive, i.e. have the Hennessy-Milner property w.r.t. behavioural equivalence. Since this property is present in all of the basic constructs, we automatically obtain expressive specification languages for a large class of systems. Concerning proof systems,

our main interests are soundness and completeness of the resulting logical system. In order to guarantee both, we investigate conditions which ensure that soundness and completeness of a combination of logics is inherited from the corresponding properties of the ingredients of the construction. Again, we demonstrate that this property is present in all basic building blocks.

As an immediate application of our compositional approach, we obtain sound, complete and expressive specification logics for a large class of probabilistic systems. To the best of the authors' knowledge, this class contains many systems, for which neither a sound and complete axiomatisation nor the Hennessy-Milner property was previously established, e.g. the simple and general probabilistic automata of Segala [28].

Our main technical tool to establish the above results is the systematic exploitation of the fact that coalgebras model the one-step behaviour of a system, i.e. that one application of the coalgebra map allows us to extract information about one transition step. This one-step behaviour of systems is parallelled both on the level of specification languages and proof systems. Regarding specification languages, we introduce the notion of *syntax constructor*, which specifies a set of syntactic features allowing the formulation of assertions about the next transition step of a system. Similarly, a *proof system constructor* specifies how one can infer judgements about the next transition step.

These notions are then used to make assertions about the global system behaviour by viewing the behaviour as the stratification of the observations which can be made after a (finite) number of steps. This is again parallelled on the level of the languages and proof systems. Completeness, for example, can then be established by isolating the corresponding one-step notion, which we call *one-step completeness*, and then proving that this entails completeness in the ordinary sense by induction on the number of transition steps. Expressiveness and soundness are treated similarly by considering the associated notions of one-step expressiveness and one-step soundness. When combining the logics, we combine both the syntax constructors and the proof system constructors, and show, that such combinations preserve one-step soundness, completeness and expressiveness.

The combination of logics and specification languages has been previously studied in different contexts. In the area of algebraic specification [30], structured specifications are used to combine already existing specifications along with their proof systems, see [4, 6]. The main technique is the use of colimits in a category of algebraic signatures and corresponding constructions on the level of models and proof systems. Since the coalgebraic approach uses endofunctors to describe the behaviour of systems, our notion of signature is much richer, and we can accordingly investigate more constructions, with functor composition being the prime example. Furthermore, the coupling of the language and its semantics is much stronger in the algebraic approach, due to the particular notions of signature and model (there is a 1-1 correspondence between function symbols on the syntactical side and functions on the level of models), so the (dual) notion of expressiveness does not play a role there.

The combination of logical systems has also been studied in its own right, based on Gabbay's notion of fibring logics [11]. The result of fibring two logics is a logic, which freely combines the connectives and proof rules from both logics. One is interested in the preservation of soundness and, in particular, completeness [32, 7]. Our approach differs from fibring in that we consider a set of particular combinations of logical oper-

ators. These combinations are also of a very specific nature, since they allow to specify information about one transition step of the system. This makes our approach specific to coalgebras and modal logics, and allows us to use induction on the number of transition steps as a proof tool.

Finally, modal logics for coalgebras have been investigated by a number of authors, starting with Moss [20], who describes an abstract syntax for a large class of systems, but there is no general completeness result. Concrete logics for coalgebras and complete proof systems are described in [20, 16, 26, 13]. This approach applies to an inductively defined class of systems, which is strictly subsumed by our approach, since we also obtain logics for probabilistic systems. Furthermore, thanks to the modularity of our construction, our logics are easily extensible to accommodate more features of transition systems, whereas it is *a priori* difficult to extend the approach of *loc. cit.* as one would have to work through one large inductive proof.

Regarding further work, we plan to extend our approach to more expressive logics, in particular to a coalgebraic version of CTL [9] and the modal μ calculus [15]. Also, it remains to be explored in what way our setup induces logics for programming languages with coalgebraically defined semantics [29, 14, 2].

2 Preliminaries and Notation

We denote the category of sets and functions by Set and pick a final object $1 = \{*\}$. Binary products (coproducts) in Set are written $X_1 \times X_2$ ($X_1 + X_2$), with canonical projections $\pi_i : X_1 \times X_2 \to X_i$ (canonical injections $\iota_i : X_i \to X_1 + X_2$). Finally, X^Y denotes the set of functions $Y \to X$.

We write Σ_{BA} for the algebraic signature specifying the boolean operators $\mathsf{ff}, \mathsf{tt}, \neg, \to, \vee, \wedge$. For any set X, its power set $\mathcal{P}X$ carries the structure of a Σ_{BA}-algebra. Then, for a set L and a function $d : L \to \mathcal{P}X$, we write $\overline{L}$ for the carrier of the free Σ_{BA}-algebra over L, and $\overline{d} : \overline{L} \to \mathcal{P}X$ for the induced Σ_{BA}-morphism.

A *boolean preorder* $(L, \vdash)$ is a Σ_{BA}-algebra L together with a preorder $\vdash \subseteq L \times L$ which is closed under the axioms and rules of propositional logic. The category of boolean preorders and order-preserving maps is denoted by $\mathsf{Preord}_{\mathsf{BA}}$; the objects of $\mathsf{Preord}_{\mathsf{BA}}$ are boolean preorders $(L, \vdash)$, while arrows from $(L, \vdash)$ to $(L', \vdash')$ are given by order-preserving Σ_{BA}-morphisms from L to L'.

We use endofunctors $T : \mathsf{Set} \to \mathsf{Set}$ to specify particular system types, and we refer to T sometimes as *signature functor*. More exactly, T specifies how the information which can be observed of the system states *in one step* is structured. Systems themselves are then modelled as T-coalgebras.

Definition 1 (Coalgebras, morphisms). *A* T-coalgebra *is a pair* (C, γ) *where* C *is a set (the* carrier*, or* state space *of the coalgebra) and* $\gamma : C \to TC$ *a function (the* coalgebra map*, or* transition structure*). A* coalgebra morphism $f : (C, \gamma) \to (D, \delta)$ *is a function* $f : C \to D$ *such that* $Tf \circ \gamma = \delta \circ f$*. The category of* T*-coalgebras is denoted by* $\mathsf{CoAlg}(T)$.

For $(C, \gamma) \in \mathsf{CoAlg}(T)$, the transition structure determines the observations $\gamma(c) \in TC$ which can be made from a state $c \in C$ in one transition step. Morphisms between

coalgebras preserve this one-step behaviour. The next example shows, that coalgebras can be used to model a wide variety of state-based and probabilistic systems:

Example 1. We use $\mathcal{P}$ to denote the covariant powerset functor and $\mathcal{D}$ for the probability distribution functor, given by

$$\mathcal{D}X = \{\mu : X \to [0,1] \mid \mu(x) = 0 \text{ for all but finitely many } x \in X \text{ and } \sum_{x \in X} \mu(x) = 1\}.$$

(i) For $TX = \mathcal{P}(A \times X) \cong \mathcal{P}(X)^A$, it is easy to see that T-coalgebras $\gamma : C \to \mathcal{P}(A \times C)$ are in 1-1 correspondence with labelled transition systems (C, R) where $R \subseteq C \times A \times C$ is defined by $(c, a, c') \in R \iff (a, c') \in \gamma(c)$. Similarly, every $\mathcal{P}$-coalgebra determines a Kripke frame and vice versa.

(ii) Coalgebras for $TX = (1 + \mathcal{D}X)^A$ are A-labelled probabilistic transition systems (see [10] for details).

(iii) The simple probabilistic automata and general probabilistic automata of [28] can be modelled as coalgebras for $TX = \mathcal{P}(A \times \mathcal{D}X)$ and $TX = \mathcal{P}(\mathcal{D}(A \times X))$.

Note that the endofunctors in the above examples are combinations of a small number of simple functors (constant, identity, powerset and probability distribution functor) using products, coproducts, exponentiation with finite exponents, and composition. In the sequel, we don't treat exponentiation with finite exponents explicitly, as it can be expressed using finite products. A recent survey of systems used in probabilistic modelling [3] identified no less than eight probabilistic system types of interest, all of which can be written as such a combination. Our goal is to derive languages and proof systems for these systems, using similar combinations on the logical level.

Apart from making this kind of compositionality explicit, the coalgebraic approach also allows for a uniform definition of behavioural equivalence, which specialises to standard notions of equivalence in many important examples.

Definition 2 (Behavioural equivalence). *Given T-coalgebras (C, γ) and (D, δ), two states $c \in C$ and $d \in D$ are called* behaviourally-equivalent *(written $c \simeq d$) if there exist T-coalgebra morphisms f and g such that $f(c) = g(d)$.*

Two states c and d are ω-behaviourally equivalent (denoted by $c \simeq_\omega d$), if $\gamma_n(c) = \delta_n(d)$ for all $n \in \omega$, where, for $(E, \epsilon) \in \mathsf{CoAlg}(T)$, $\epsilon_0 : E \to 1$ is the unique map and $\epsilon_{n+1} = T\epsilon_n \circ \epsilon$.

The notion of ω-behavioural equivalence only takes finitely observable behaviour into account and is strictly weaker than behavioural equivalence. It can be shown that for ω-accessible T, both notions coincide [17]. It is often possible to define finitary logics for which logical equivalence coincides with ω-behavioural equivalence. On the other hand, we can not in general hope to characterise behavioural equivalence by a logic with finitary syntax.

It can be shown that for weak pullback preserving endofunctors, the notion of behavioural equivalence coincides with coalgebraic bisimulation, introduced by Aczel and Mendler [1] and studied by Rutten [27]. All functors considered in the sequel are weak pullback preserving. In the examples, the situation is as follows:

Example 2. We consider some of the systems introduced in Example 1.

(i) For labelled transition systems, i.e. coalgebras for $TX = \mathcal{P}(X)^A$, behavioural equivalence coincides with Park-Milner bisimulation [22, 19].

(ii) The notion of behavioural equivalence for coalgebras for $TX = (1 + \mathcal{D}X)^A$, that is, probabilistic transition systems, coincides with the notion of probabilistic bisimulation considered in [18]. (This is proved in [10].)

A more detailed analysis of probabilistic systems from a coalgebraic point of view can be found in [3].

3 Modular Construction of Modal Languages

In this section we introduce *syntax constructors* and the modal languages they define. If we consider a modal language $\mathcal{L}$ as an extension of propositional logic, the idea of a syntax constructor is that it describes what we need to add to the propositional language in order to obtain $\mathcal{L}$. The important feature of syntax constructors is, that they can be combined like the signature functors which define the particular shape of the systems under consideration. After introducing the abstract concept, we give examples of syntax constructors for some basic functors and show how they can be combined in order to obtain more structured modal languages.

Definition 3 (Syntax constructor and induced language).

(i) A syntax constructor *is an ω-accessible endofunctor $\mathsf{S} : \mathsf{Set} \to \mathsf{Set}$, which preserves inclusions, i.e. $\mathsf{S}X \subseteq \mathsf{S}Y$ for all $X \subseteq Y$.*

(ii) The language $\mathcal{L}(\mathsf{S})$ *associated with a syntax constructor is the least set of formulas containing*

- ff *and $\varphi \to \psi$ whenever $\varphi, \psi \in \mathcal{L}(\mathsf{S})$*
- *all $\varphi \in \mathsf{S}(\Phi)$ whenever $\Phi \subseteq \mathcal{L}(\mathsf{S})$.*

The requirement that syntax constructors preserve inclusions is mainly for ease of exposition, since in this case they define a monotone operator on sets, and languages can be constructed as least fixed points in the usual way. Equivalently, one could drop the requirement of inclusion-preservation at the expense of having to work with abstract (first oder) syntax, that is, constructing the language associated with a syntax constructor as the initial algebra of the functor $LX = 1 + X^2 + \mathsf{S}X$.

Recall that an inclusion preserving endofunctor is ω-accessible iff, for all sets X and all $x \in TX$, there is a finite $S \subseteq X$ with $x \in TS$. Hence the requirement of ω-accessibility ensures that the construction of the associated language terminates after ω steps, that is, we are dealing with finitary logics only.

Before we show how syntax constructors can be combined, we introduce syntax constructors for some simple languages.

Example 3. (i) If A is a set (of atomic propositions), then the constant functor $\mathsf{S}_A X = A$ is a syntax constructor. The associated language $\mathcal{L}(\mathsf{S})$ is the set of propositional formulas over the set A of atoms.

(ii) If M is a (possibly infinite) set of modal operators with associated (finite) arities, then S_M is a syntax constructor, where S_M maps a set X (of formulas) to the set $\mathsf{S}_\mathsf{M}(X)$ of formal expressions, given by

$$\mathsf{S}_\mathsf{M}(X) = \{m(x_1, \ldots, x_n) \mid\ m \in \mathsf{M} \text{ is } n\text{-ary}, x_1, \ldots, x_n \in X\}.$$

Viewing M as an algebraic signature, $\mathsf{S}_{\mathsf{M}}(X)$ is the set of terms with exactly one function symbol applied to variables in X. In the literature on modal logic, M is also called a modal similarity type [5]. The language of S_{M} is the set of modal formulas with modalities in M over the empty set of variables. For later reference, we let $\mathsf{S}_{\mathcal{P}} = \mathsf{S}_{\{\Box\}}$ where $\Box$ has arity one, and $\mathsf{S}_{\mathcal{D}} = \mathsf{S}_{\mathsf{M}}$ where $\mathsf{M} = \{L_p \mid p \in \mathbb{Q} \cap [0,1]\}$, each L_p having arity one, and $\mathbb{Q}$ denotes the set of rational numbers. The language associated with $\mathsf{S}_{\mathcal{P}}$ is standard modal logic over the empty set of propositional variables. The language associated with $\mathsf{S}_{\mathcal{D}}$ has a countable number of unary modalities, and will be used to describe probabilistic transition systems.

We are now ready for the first modularity issue of the present paper: the combination of syntax constructors to build more powerful languages from simple ingredients.

Definition 4 (Combinations of syntax constructors). *Consider the following operations on sets L_1, L_2 (of formulas):*

$$L_1 \otimes L_2 = \{\pi(\varphi_1, \varphi_2) \mid \varphi_i \in L_i, i = 1,2\} \qquad L_1 \oplus L_2 = \{\langle \kappa_i \rangle \varphi_i \mid \varphi_i \in L_i, i = 1,2\}.$$

For syntax constructors $\mathsf{S}_1, \mathsf{S}_2$ we let

$$(\mathsf{S}_1 \otimes \mathsf{S}_2)X = \overline{\mathsf{S}_1 X} \otimes \overline{\mathsf{S}_2 X} \quad (\mathsf{S}_1 \oplus \mathsf{S}_2)X = \overline{\mathsf{S}_1 X} \oplus \overline{\mathsf{S}_2 X} \quad (\mathsf{S}_1 \circledcirc \mathsf{S}_2)X = \mathsf{S}_1(\overline{\mathsf{S}_2 X}).$$

Note that above operations are of a purely syntactical nature, and the addition of the symbols π and κ_i serves as a way to ensure that the resulting functors are inclusion-preserving.

When combining syntax constructors, we add another layer of modal operators to already defined syntax. Closure under propositional connectives is needed to express propositional judgements also at the level on which the construction operates, e.g. to have formulas $\pi(\Box\varphi \vee \Box\psi, \Box\rho)$ in $\mathcal{L}(\mathsf{S}_{\mathcal{P}} \otimes \mathsf{S}_{\mathcal{P}})$.

The above definition is modelled after the definition of signature functors. In contrast to the logics treated in [26, 13], our syntax constructors do not deal with exponentiation. This is due to the fact that infinite exponents fail to be ω-accessible, whereas finite exponents can be simulated by finite products. The third clause dealing with the composition of syntax constructors gives rise to S_1-modal operators which are indexed by S_2-formulas. Alternatively, the composition of syntax constructors can be thought of as introducing an additional sort:

Example 4. Suppose $\mathsf{S}_i X = \{\Box_i x \mid x \in X\}$ for $i = 1,2$. Then the language $\mathcal{L} = \mathcal{L}(\mathsf{S}_1 \circledcirc \mathsf{S}_2)$ can be described by the following grammar:

$$\mathcal{L} \ni \varphi, \psi ::= \mathsf{ff} \mid \varphi \to \psi \mid \Box_1 \rho \qquad (\rho \in \mathcal{L}')$$
$$\mathcal{L}' \ni \rho, \sigma ::= \mathsf{ff} \mid \sigma \to \rho \mid \Box_2 \varphi \qquad (\varphi \in \mathcal{L})$$

Languages of this kind can be used to specify properties of systems, whose signature functor T is the composition of two functors $T = T_1 \circ T_2$. In order to capture all possible behaviour described by T, we first have to describe the T_2 behaviour, and then use these descriptions to specify the observations which can be made according to T_1. Since propositional connectives will be in general necessary to capture all possible T_2

behaviour, the definition of the syntax constructor $\mathsf{S}_1 \circledcirc \mathsf{S}_2$ involves the closure under propositional connectives before applying S_1.

Similarly, languages of form $\mathcal{L}(\mathsf{S}_1 \otimes \mathsf{S}_2)$ and $\mathcal{L}(\mathsf{S}_1 \oplus \mathsf{S}_2)$ will be used to formalise properties of systems whose signature functors are of form $T_1 \times T_2$ and $T_1 + T_2$, respectively. The next proposition shows that the constructions in Definition 4 indeed give rise to syntax constructors:

Proposition 1. *$\mathsf{S}_1 \otimes \mathsf{S}_2$, $\mathsf{S}_1 \oplus \mathsf{S}_2$, $\mathsf{S}_1 \circledcirc \mathsf{S}_2$ are syntax constructors.*

In ordinary modal logic, the modal language $\mathcal{L}$ can be viewed as stratification $\mathcal{L} = \bigcup_{n\in\omega} \mathcal{L}^n$, where $\mathcal{L}^n$ contains all modal formulas of rank $\leq n$. This in particular allows us to use induction on the rank of formulas as a proof principle.

Definition 5. *Suppose S is a syntax constructor. Let $\mathcal{L}^0(\mathsf{S}) = \overline{\emptyset}$ and $\mathcal{L}^{n+1}(\mathsf{S}) = \overline{\mathsf{S}(\mathcal{L}^n)}$. If $\varphi \in \mathcal{L}^n(\mathsf{S})$, we say that φ* has rank at most n.

If $\mathsf{S} = \mathsf{S}_{\mathsf{M}}$ for a set M of modal operators, then $\mathcal{L}^n(\mathsf{S})$ contains the modal formulas, whose depth of modal operators is at most n. The fact that $\mathcal{L}(\mathsf{S})$ can be viewed as a stratification of $\mathcal{L}^n(\mathsf{S})$, for $n \in \omega$, is the content of the next lemma.

Lemma 1. *$\mathcal{L}(\mathsf{S}) = \bigcup_{n\in\omega} \mathcal{L}^n(\mathsf{S})$ and $\overline{\mathsf{S}(\mathcal{L}(\mathsf{S}))} = \mathcal{L}(\mathsf{S})$.*

4 Modular Construction of Coalgebraic Semantics

In the previous section, we have argued that a syntax constructor with associated language $\mathcal{L}$ specifies those features which have to be added to propositional logic in order to obtain $\mathcal{L}$. In standard modal logic, this boils down to adding the operator $\Box$, which can be used to describe the observable behaviour after one transition step. Abstracting from this example, we now introduce the *one-step semantics* of a syntax constructor, which relates the additional modal structure (specified by a syntax constructor) to the observations (specified by a signature functor) which can be made of a system in one transition step.

Throughout the section, S denotes a syntax constructor and T is an endofunctor; recall that $\overline{L}$ is the closure of the set L under propositional connectives. We write $\overline{\mathsf{S}} : \mathsf{Alg}(\Sigma_{\mathsf{BA}}) \to \mathsf{Alg}(\Sigma_{\mathsf{BA}})$ for the functor taking a Σ_{BA}-algebra L to the Σ_{BA}-algebra $\overline{\mathsf{S}(L)}$, and a Σ_{BA}-morphism $t : L \to L'$ to the obvious extension of $\mathsf{S}(t) : \mathsf{S}L \to \mathsf{S}L'$ to a Σ_{BA}-morphism. The following definition provides a semantics to syntax constructors. As we are dealing with extensions of propositional logic, we use algebras for the boolean signature as a notational vehicle.

Definition 6 (One-step semantics). *If L is a Σ_{BA}-algebra and X is a set, then an* interpretation of L *over X is a Σ_{BA}-morphism $d : L \to \mathcal{P}X$. A* morphism between interpretations $d : L \to \mathcal{P}X$ *and $d' : L' \to \mathcal{P}X'$ is a pair (t, f) with $t : L \to L'$ a Σ_{BA}-morphism and $f : X' \to X$ a function, such that $d' \circ t = f^{-1} \circ d$:*

$$\begin{array}{ccc} L & \xrightarrow{t} & L' \\ {\scriptstyle d}\downarrow & & \downarrow{\scriptstyle d'} \\ \mathcal{P}X & \xrightarrow{f^{-1}} & \mathcal{P}X' \end{array}$$

A one-step semantics $[\![\mathsf{S}]\!]^T$ *of a syntax constructor* S *w.r.t. an endofunctor* T *maps interpretations of* L *over* X *to interpretations of* $\overline{\mathsf{S}}(L)$ *over* TX*, in such a way that whenever* $(t, f) : d \to d'$ *is a morphism of interpretations, so is* $(\overline{\mathsf{S}}t, Tf) : [\![\mathsf{S}]\!]^T(d) \to [\![\mathsf{S}]\!]^T(d')$*. We omit the superscript on the one-step semantics if the associated endofunctor is clear from the context.*

A one-step semantics provides the glue between a language constructor and an endofunctor. The requirement that $[\![\mathsf{S}]\!]^T$ preserves morphisms of interpretations ensures that $[\![\mathsf{S}]\!]^T$ is defined uniformly on interpretations. This will subsequently guarantee that the (yet to be defined) coalgebraic semantics of the induced language $\mathcal{L}(\mathsf{S})$ is adequate w.r.t. behavioural equivalence; that is, behaviourally-equivalent states of coalgebras cannot be distinguished using formulas of the language.

A variant of the notion of one-step semantics, which treats syntax and the associated interpretation in the same framework, was studied in [8]. For languages with unary modalities, a one-step semantics corresponds to a choice of predicate liftings [23, 24].

The key feature of a one-step semantics of a syntax constructor is that it gives rise to a semantics of $\mathcal{L}(\mathsf{S})$ w.r.t. T-coalgebras, that is, it defines a satisfaction relation between T-coalgebras and formulas of $\mathcal{L}(\mathsf{S})$. Furthermore, we can define a one-step semantics of a combination of syntax constructors in terms of the one-step semantics of the ingredients. Before describing these constructions, we provide one-step semantics for some simple syntax constructors.

Example 5. We define one-step semantics for the syntax constructors introduced in Example 3.

(i) Suppose A is a set. Then the function which maps an arbitrary interpretation to the unique interpretation extending the identity function on A is a one-step semantics of S_A w.r.t. the constant functor $TX = A$.

(ii) A one-step semantics for $\mathsf{S}_{\mathcal{P}}$ w.r.t. $\mathcal{P}$ is given by

$$[\![\mathsf{S}_{\mathcal{P}}]\!](d) : \overline{\mathsf{S}_{\mathcal{P}}(L)} \to \mathcal{P}\mathcal{P}X \qquad [\![\mathsf{S}_{\mathcal{P}}]\!](d)(\Box\varphi) = \{x \subseteq X \mid x \subseteq d(\varphi)\}.$$

(iii) For the syntax constructor $\mathsf{S}_{\mathcal{D}}$ associated with the probability distribution functor, we define a one-step semantics by

$$[\![\mathsf{S}_{\mathcal{D}}]\!](d) : \overline{\mathsf{S}_{\mathcal{D}}(L)} \to \mathcal{P}\mathcal{D}X \qquad [\![\mathsf{S}_{\mathcal{D}}]\!](d)(L_p\varphi) = \{\mu : \sum_{x \in d(\varphi)} \mu(x) \geq p\}$$

where $d : L \to \mathcal{P}X$ in both cases.

We now return to the claim made at the beginning of this section and show, that a one-step semantics gives rise to an interpretation of the associated language $\mathcal{L}(\mathsf{S})$ over T-coalgebras.

Definition 7 (Coalgebraic semantics). *Suppose* S *is a syntax constructor with one-step semantics* $[\![\mathsf{S}]\!]^T$*, and* $(C, \gamma) \in \mathsf{CoAlg}(T)$.

The coalgebraic semantics $[\![\varphi]\!] = [\![\varphi]\!]_C \subseteq C$ *of a formula* $\varphi \in \mathcal{L}(\mathsf{S})$ *w.r.t. a* T*-coalgebra* (C, γ) *is defined inductively on the structure of formulas by*

$$[\![\mathsf{ff}]\!] = \emptyset \qquad [\![\varphi \to \psi]\!] = (C \setminus [\![\varphi]\!]) \cup [\![\psi]\!]$$
$$[\![\sigma]\!] = \gamma^{-1}([\![\mathsf{S}]\!]^T(d_\Phi)(\sigma)) \quad \textit{(for } \sigma \in \mathsf{S}\Phi\textit{)}$$

where we inductively assume that $[\![\varphi]\!]$ is already defined for $\varphi \in \Phi$, giving rise to the map $d_\Phi : \Phi \to \mathcal{P}(C)$, $\varphi \mapsto [\![\varphi]\!]$. Given $c \in C$, we write $c \models_C \varphi$ for $c \in [\![\varphi]\!]_C$, and $\mathrm{Th}(c) = \{\varphi \in \mathcal{L}(\mathsf{S}) \mid c \models_C \varphi\}$.

Before showing that this definition captures the standard interpretation of some known modal logics, we need to show that the coalgebraic semantics is well defined, as we can have $\sigma \in \mathsf{S}\Phi$ and $\sigma \in \mathsf{S}\Psi$ for two different Φ, Ψ.

Lemma 2. *The coalgebraic semantics of $\mathcal{L}(\mathsf{S})$ is well defined, that is, for $(C, \gamma) \in \mathsf{CoAlg}(T)$ and $\Phi, \Psi \subseteq \mathcal{L}(\mathsf{S})$, we have $[\![\mathsf{S}]\!]^T(d_\Phi)(\sigma) = [\![\mathsf{S}]\!]^T(d_\Psi)(\sigma)$ for all $\sigma \in \mathsf{S}\Phi \cap \mathsf{S}\Psi$.*

Note that the definition of the coalgebraic semantics generalises the semantics of modal formulas, as well as the semantics of the formulas considered in [12]:

Example 6. (i) Consider the syntax constructor $\mathsf{S}_\mathcal{P}$ defined in Example 3, and the associated semantics $[\![\mathsf{S}_\mathcal{P}]\!]$ as in Example 5. The induced coalgebraic semantics w.r.t. (C, γ) is defined inductively by

$$c \models \Box\varphi \;\text{ iff }\; e \models \varphi \text{ for all } e \in \gamma(c)$$

This is the standard textbook semantics of modal logic [5].

(ii) Consider the syntax constructor $\mathsf{S}_\mathcal{D}$ defined in Example 3, and the associated semantics $[\![\mathsf{S}_\mathcal{D}]\!]$ as in Example 5. The induced coalgebraic semantics w.r.t. (C, γ) is defined inductively by

$$c \models L_p\varphi \;\text{ iff }\; \gamma(c)([\![\varphi]\!]) \geq p$$

The above example shows that the coalgebraic semantics specialises to known semantics in concrete cases. We now turn to the issue of combining semantics, and show that we can derive a one-step semantics for a combination of syntax constructors (see Definition 4) by combining one-step semantics for the ingredients.

Definition 8 (Combinations of one-step semantics). *Let $d_1 : L_1 \to \mathcal{P}X_1$ and $d_2 : L_2 \to \mathcal{P}X_2$ be interpretations of L_1 (respectively L_2) over X_1 (respectively X_2) and consider the functions*

$$d_1 \otimes d_2 : L_1 \otimes L_2 \to \mathcal{P}(X_1 \times X_2),\ \pi(\varphi_1, \varphi_2) \mapsto \{(x_1, x_2) \mid x_i \in d_i(\varphi_i)\}$$
$$d_1 \oplus d_2 : L_1 \oplus L_2 \to \mathcal{P}(X_1 + X_2),\ \langle\kappa_i\rangle\varphi_i \mapsto \{\iota_i(x_i) \mid x_i \in d_i(\varphi_i)\}.$$

If $[\![\mathsf{S}_i]\!]^{T_i}$ is a one-step semantics of a syntax constructor S_i w.r.t. an endofunctor T_i, for $i = 1, 2$, the one-step semantics of various combinations of S_1 and S_2 is given as follows:

$$[\![\mathsf{S}_1 \otimes \mathsf{S}_2]\!](d) = \overline{[\![\mathsf{S}_1]\!](d) \otimes [\![\mathsf{S}_2]\!](d)} \qquad [\![\mathsf{S}_1 \oplus \mathsf{S}_2]\!](d) = \overline{[\![\mathsf{S}_1]\!](d) \oplus [\![\mathsf{S}_2]\!](d)}$$
$$[\![\mathsf{S}_1 \circledcirc \mathsf{S}_2]\!](d) = [\![\mathsf{S}_1]\!]([\![\mathsf{S}_2]\!](d))$$

where we have notationally suppressed that $[\![S_1]\!] \,\mathrm{op}\, [\![S_2]\!]$ is a one-step semantics of $S_1 \,\mathrm{op}\, S_2$ w.r.t. $T_1 \,\mathrm{op}'\, T_2$, for $\mathrm{op} = \otimes, \oplus, \circledcirc$ and $\mathrm{op}' = \times, +, \circ$.

Note the absence of the closure operator $\overline{\cdot}$ in the last clause; this is already taken care of by the definition of $\mathsf{S}_1 \circledcirc \mathsf{S}_2$. The intuitions behind the definitions of $d_1 \oplus d_2$, $d_1 \otimes d_2$ are as follows. Assuming that L_1 and L_2 are interpreted over X_1 and X_2, respectively, we can interpret the language $L_1 \otimes L_2$ (respectively $L_1 \oplus L_2$) over $X_1 \times X_2$ (respectively $X_1 + X_2$). In the first case, a formula $\pi(\varphi_1, \varphi_2)$ holds at a state $x = (x_1, x_2)$ iff φ_i holds in x_i, $i = 1, 2$. Also, $\langle \kappa_i \rangle \varphi_i$ holds in $x \in X_1 + X_2$ iff $x = \iota_i(x_i)$ and φ_i holds in x_i.

We now show that the combination of one-step semantics is well defined. To make notation bearable we disregard the dependency on the endofunctor.

Proposition 2. *Suppose $[\![\mathsf{S}_i]\!]$ is a one-step semantics for S_i w.r.t. T_i for $i = 1, 2$. Then $[\![\mathsf{S}_1]\!] \otimes [\![\mathsf{S}_2]\!]$, $[\![\mathsf{S}_1]\!] \oplus [\![\mathsf{S}_2]\!]$ and $[\![\mathsf{S}_1]\!] \circledcirc [\![\mathsf{S}_2]\!]$ are one-step semantics for $T_1 \times T_2$, $T_1 + T_2$ and $T_1 \circ T_2$, respectively.*

We have therefore seen how we can combine syntax constructors and their associated one-step semantics. This gives rise to a modular way of constructing languages for coalgebras. The following two sections present applications of the modular approach. In the next section we show that a combination of logics has the Hennessy-Milner property if all the ingredients satisfy an expressiveness property. In the subsequent section, we show how to obtain sound and complete proof systems for a combination of logics by suitably combining sound and complete proof systems for the building blocks.

5 Behavioural versus Logical Equivalence

In this section, we show that any two behaviourally equivalent points necessarily have the same logical theory. In order to prove the Hennessy-Milner property for a logic which arises from a combination of syntax constructors, we introduce the notion of expressiveness for an interpretation $L \to \mathcal{P}X$, and show that the language associated with a one-step semantics which preserves expressiveness has the Hennessy-Milner property. To treat languages which arise from a combination of syntax constructors, we show that the combination of one-step semantics preserves expressiveness if all of the ingredients do. This in particular allows us to establish the Hennessy-Milner property for combined languages in a modular fashion. We begin with the easy part and show that behaviourally equivalent states cannot be distinguished by formulas of a logic which is induced by a syntax constructor.

Proposition 3. *Suppose S is a syntax constructor with one-step semantics $[\![\mathsf{S}]\!]$, and $(C, \gamma), (D, \delta) \in \mathsf{CoAlg}(T)$. Then, $\mathrm{Th}(c) = \mathrm{Th}(d)$ whenever $c \simeq d$.*

The remainder of the section is concerned with the converse of Proposition 3. For that, we introduce the notion of one-step expressiveness, which allows to derive a Hennessy-Milner property for the language associated with a syntax constructor. Moreover, we show that this condition automatically holds for a combination of syntax constructors, if it is valid for the ingredients of the construction.

Definition 9 (One-step expressiveness).

(i) An interpretation $d : L \to \mathcal{P}X$ is expressive *if the associated language map $d^\sharp : X \to \mathcal{P}L$, given by $x \mapsto \{\varphi \in \mathcal{L} : x \in d(\varphi)\}$ is injective.*

(ii) A one-step semantics $[\![\mathsf{S}]\!]^T$ is one-step expressive *if $[\![\mathsf{S}]\!]^T(d)$ is expressive whenever d is.*

Using this terminology, our first main result can be stated as follows:

Theorem 1. *If $[\![\mathsf{S}]\!]^T$ is one-step expressive, then $\mathcal{L}(\mathsf{S})$ is expressive w.r.t. $\simeq_\omega$, i.e. $\mathrm{Th}(c) = \mathrm{Th}(d)$ iff $c \simeq_\omega d$ for all (C,γ), $(D,\delta) \in \mathsf{CoAlg}(T)$ and $(c,d) \in C \times D$.*

In other words, the logic is strong enough to distinguish all states, which exhibit different behaviour, which can be witnessed by observing finitely many steps only. The proof of this theorem uses induction on the rank of formulas (see Definition 5), and a semantical representation of a formula of rank n as a subset of $T^n 1$. Using the fact that ω-behavioural equivalence coincides with behavioural equivalence for coalgebras of an ω-accessible endofunctor (see [31]), we have the following corollary:

Corollary 1. *If T is ω-accessible, then $\mathcal{L}(\mathsf{S})$ is expressive, that is, $\mathrm{Th}(c) = \mathrm{Th}(d)$ iff $c \simeq d$ for all (C,γ), $(D,\delta) \in \mathsf{CoAlg}(T)$ and all $(c,d) \in C \times D$.*

Note that the accessibility degree of the underlying endofunctor T basically limits the branching degree of T-coalgebras [24], so the above corollary is a coalgebraic Hennessy-Milner result.

It is easy to see that the one-step semantics of all basic syntax constructors are one-step expressive:

Example 7. The one-step semantics of the syntax constructors from Example 5 are one-step expressive, if we consider the *finite* powerset functor $\mathcal{P}_f$ in clause (ii).

Our next goal is to show that one-step expressiveness is preserved by all the combinations of syntax constructors. Again suppressing the dependency on the endofunctor T we obtain:

Proposition 4. *Suppose $[\![\mathsf{S}_i]\!]$ are one-step expressive, for $i = 1, 2$. Then so are $[\![\mathsf{S}_1]\!] \otimes [\![\mathsf{S}_2]\!]$, $[\![\mathsf{S}_1]\!] \oplus [\![\mathsf{S}_2]\!]$ and $[\![\mathsf{S}_1]\!] \circledcirc [\![\mathsf{S}_2]\!]$.*

Thus, Theorem 1 applies to any combination of one-step semantics which are one-step expressive. Note that this in particular implies that the language associated with the combination of two syntax constructors distinguishes any two states up to ω-behavioural equivalence, or in case T is ω-accessible, even up to behavioural equivalence. As an immediate application, we obtain expressive languages for all system types discussed in Example 1.

6 Modular Construction of Proof Systems

This section extends the methods presented so far to also include the compositional construction of proof systems. Our main result shows that this can be done in such a way that the combined proof system inherits soundness and completeness from its building blocks. The key notion needed to formulate the modularisation of proof systems is that of a proof system constructor.

Definition 10 (Proof system constructor). *Suppose* S *is a syntax constructor. A* proof system constructor *for* S *is a functor* $\mathsf{P} : \mathsf{Preord}_{\mathsf{BA}} \to \mathsf{Preord}_{\mathsf{BA}}$ *such that*

(i) $\overline{\mathsf{S}} \circ U = U \circ \mathsf{P}$*, where* $U : \mathsf{Preord}_{\mathsf{BA}} \to \mathsf{Alg}(\Sigma_{\mathsf{BA}})$ *is the forgetful functor;*
(ii) P *preserves order-reflecting morphisms.*

The intuition is as follows. The syntax constructor S specifies a set of modalities to be added to propositional logic, while the induced functor $\overline{\mathsf{S}}$ produces the language which arises by applying the given modal operators exactly once, and subsequently closing under propositional connectives. Now a corresponding proof system constructor takes a boolean preorder $\vdash_L \subseteq\ L \times L$, which represents all facts that can be proved about formulas in L, and produces a boolean preorder $\vdash_{\overline{\mathsf{S}}(L)} \subseteq\ \overline{\mathsf{S}}(L) \times \overline{\mathsf{S}}(L)$, which defines all provable sequents over the next transition step, that can be derived from sequents over L. In other words, a proof system constructor specifies how we can lift sequents to formulas containing an extra degree of nesting of the modal operators. The second requirement in Definition 10 formalises a well-behavedness property of proof system constructors, which will ensure that the proof systems induced by proof system constructors can be constructed inductively.

Since the axioms of modal logic involve formulas of rank one only, we can give a straightforward encoding of modal logic in a proof system constructor.

Example 8. Consider the syntax constructor $\mathsf{S}_{\mathcal{P}}$ defined in Example 3. For a boolean preorder $(L, \vdash_L)$, define $\mathsf{P}_{\mathcal{P}}(L, \vdash_L) = (\overline{\mathsf{S}_{\mathcal{P}}}(L), \vdash_{\overline{\mathsf{S}_{\mathcal{P}}}(L)})$ where $\vdash_{\overline{\mathsf{S}_{\mathcal{P}}}(L)}$ is the relation generated by the following axioms and rules:

$$\frac{\varphi \vdash_L \psi}{\Box\varphi \vdash_{\overline{\mathsf{S}_{\mathcal{P}}}(L)} \Box\psi} \qquad \mathsf{tt} \vdash_{\overline{\mathsf{S}_{\mathcal{P}}}(L)} \Box\mathsf{tt} \qquad \Box\varphi \wedge \Box\psi \vdash_{\overline{\mathsf{S}_{\mathcal{P}}}(L)} \Box(\varphi \wedge \psi)$$

augmented with the axioms and rules of propositional logic. Then $\mathsf{P}_{\mathcal{P}}$ is a proof system constructor for $\mathsf{S}_{\mathcal{P}}$.

In the case of probabilistic transition systems, the logic in [12] can also be captured by a proof system constructor.

Example 9. Consider the syntax constructor $\mathsf{S}_{\mathcal{D}}$ defined in Example 3. For $p \in [0,1]$, let $M_p\varphi ::= L_{1-p}\neg\varphi$, and $E_p\varphi ::= L_p\varphi \wedge M_p\varphi$. Also, for a finite sequence of formulas $\varphi_1, \ldots, \varphi_m$, let $\varphi^{(k)}$ stand either for $\bigvee_{1 \le l_1 < \ldots < l_k \le m} (\varphi_{l_1} \wedge \ldots \wedge \varphi_{l_k})$, if $k \le m$, or for ff, if $k > m$. Thus, the formula $\varphi^{(k)}$ states that, from among the formulas $\varphi_1, \ldots, \varphi_m$, at least k are true at any point.

Now for each boolean preorder $(L, \vdash)$, define $\mathsf{P}_{\mathcal{D}}(L, \vdash) = (\overline{\mathsf{S}_{\mathcal{D}}}(L), \vdash')$, where the relation $\vdash'$ is generated by the axioms and rules in Figure 1, augmented with the axioms and rules of propositional logic. All but the last of these axioms and rules capture immediate properties of the one-step semantics $[\![\mathsf{S}_{\mathcal{D}}]\!]$ defined in Example 5. The last rule describes a more involved property of probability distributions, see [12] for details.

The functor $\mathsf{P}_{\mathcal{D}} : \mathsf{Preord}_{\mathsf{BA}} \to \mathsf{Preord}_{\mathsf{BA}}$ defined above qualifies as a proof system constructor for $\mathsf{S}_{\mathcal{D}}$.

A proof system constructor P for S induces a derivability relation $\vdash^g_{\mathsf{P}}$ on the language $\mathcal{L}(\mathsf{S})$, defined as the set of judgements, which one can infer by applying the proof system constructor.

$$\vdash' L_0\varphi \qquad \vdash' L_p\mathtt{tt} \qquad \vdash' \neg L_p\varphi \to M_p\varphi \qquad \vdash' L_p\varphi \to \neg L_q\neg\varphi$$

$$\frac{\varphi \vdash \psi}{L_p\varphi \vdash' L_p\psi} \qquad \frac{\vdash \bigwedge_{k=1}^{max(m,n)} \varphi^{(k)} \leftrightarrow \psi^{(k)}}{\vdash' (\bigwedge_{i=1}^{m} L_{p_i}\varphi_i) \wedge (\bigwedge_{j=2}^{n} M_{q_i}\psi_i) \to L_{p_1+\ldots+p_m-(q_2+\ldots+q_n)}\psi_1}$$

Fig. 1. Axioms and Rules for $\mathcal{D}$, where $p + q > 1$

Definition 11 (Global proof system induced by P). *The* global proof system induced by P *is the least boolean preorder* $(\mathcal{L}(\mathsf{S}), \vdash^g_{\mathsf{P}})$ *such that* $\mathsf{P}(\mathcal{L}(\mathsf{S}), \vdash^g_{\mathsf{P}}) \subseteq (\mathcal{L}(\mathsf{S}), \vdash^g_{\mathsf{P}})$.

In particular, since $\vdash^g_{\mathsf{P}}$ is a boolean preorder, it contains all instances of propositional tautologies. We now apply our main programme also to this definition, and show, that the global proof system can be viewed as stratification of a sequence of relations $\vdash^n_{\mathsf{P}} \subseteq \mathcal{L}^n(\mathsf{S}) \times \mathcal{L}^n(\mathsf{S})$. This will open the road for the proof of soundness and completeness using induction on the rank of the formulas.

Definition 12 (Inductive proof system induced by P). *For $n \in \omega$, define $\vdash_n \subseteq \mathcal{L}^n(\mathsf{S}) \times \mathcal{L}^n(\mathsf{S})$ by:*

- *$\varphi \vdash_0 \psi$ whenever $\varphi \to \psi$ is a propositional tautology;*
- *$(\mathcal{L}^{n+1}(\mathsf{S}), \vdash_{n+1}) = \mathsf{P}(\mathcal{L}^n(\mathsf{S}), \vdash_n)$*

The inductive proof system induced by P *is given by* $(\mathcal{L}(\mathsf{S}), \vdash_{\mathsf{P}}) = \bigcup_{n\in\omega}(\iota_n \times \iota_n)(\mathcal{L}^n(\mathsf{S}), \vdash_n)$ *where $\iota_n : \mathcal{L}^n(\mathsf{S}) \to \mathcal{L}(\mathsf{S})$ denote the inclusions arising from Lemma 1.*

Lemma 3. *$(\mathcal{L}^n(\mathsf{S}), \vdash_n) \subseteq (\mathcal{L}^{n+1}(\mathsf{S}), \vdash_{n+1})$ and $\mathsf{P}(\mathcal{L}(\mathsf{S}), \vdash_{\mathsf{P}}) = (\mathcal{L}(\mathsf{S}), \vdash_{\mathsf{P}})$.*

The two requirements in the definition of proof system constructors are exactly what is needed to show that the two proof systems induced by P coincide.

Proposition 5. *The boolean preorders $(\mathcal{L}(\mathsf{S}), \vdash^g_{\mathsf{P}})$ and $(\mathcal{L}(\mathsf{S}), \vdash_{\mathsf{P}})$ coincide.*

We can therefore use induction on n to prove properties of $(\mathcal{L}(\mathsf{S}), \vdash^g_{\mathsf{P}})$. In the following, we consider soundness and completeness of $(\mathcal{L}(\mathsf{S}), \vdash^g_{\mathsf{P}})$ w.r.t. the coalgebraic semantics induced by some one-step semantics $[\![\mathsf{S}]\!]^T$, and show that these follow from soundness and completeness conditions involving $[\![\mathsf{S}]\!]^T$ and P.

Definition 13 (One-step soundness and completeness).

(i) A boolean preorder $\vdash \subseteq L \times L$ is sound *(*complete*) w.r.t. an interpretation $d : L \to \mathcal{P}X$ if $\varphi \vdash \psi$ implies $d(\varphi) \subseteq d(\psi)$ ($d(\varphi) \subseteq d(\psi)$ implies $\varphi \vdash \psi$) for any $\varphi, \psi \in L$.*

(ii) A proof system constructor P for S is one-step sound *(*complete*) w.r.t. a one-step semantics $[\![\mathsf{S}]\!]^T$ if $\mathsf{P}(L, \vdash)$ is sound (complete) w.r.t. $[\![\mathsf{S}]\!]^T(d) : \overline{\mathsf{S}}L \to \mathcal{P}TX$ whenever $\vdash$ is sound (complete) w.r.t. $d : L \to \mathcal{P}X$.*

Using induction, we can derive soundness and completeness in the standard way from their one-step counterparts:

Theorem 2 (Soundness and completeness). *Assume $T : \mathsf{Set} \to \mathsf{Set}$ is such that $T1 \neq \emptyset$. If the proof system constructor P for S is one-step sound (complete) w.r.t. $[\![\mathsf{S}]\!]^T$, then $(\mathcal{L}(\mathsf{S}), \vdash_{\mathsf{P}})$ is sound (complete) w.r.t. the coalgebraic semantics of $\mathcal{L}(\mathsf{S})$.*

In the case of probabilistic transition systems, the axioms and rules given in Example 9 form a sound and complete proof system. This was proved in [12] using the standard filtration method. For us, this result is of limited usefulness, as we must show that the proof system constructor defined in Example 9 is one-step sound and complete. This will later allow us to derive sound and complete proof systems for more complex types of probabilistic systems.

The following proposition, which deals with the base case of the probability distribution functor, puts us into the position to apply our techniques to a large class of probabilistic systems.

Proposition 6. *The proof system constructor $\mathsf{P}_{\mathcal{D}}$ of Example 9 is one-step sound and complete w.r.t. $[\![\mathsf{S}_{\mathcal{D}}]\!]$.*

The proof of this result makes use of Rockafellar's Theorem [25].

In what follows, we will show how one can combine proof system constructors for simple languages in order to derive proof systems for more complex languages. Moreover, we will show that whenever the building blocks of such constructions are one-step sound and complete w.r.t. some given one-step semantics, the resulting proof system is sound and complete w.r.t. the induced coalgebraic semantics. To describe the combinations of proof systems, we introduce the following notation:

$$[\pi_1]\varphi_1 ::= \pi(\varphi_1, \mathsf{tt}) \in (\mathsf{S}_1 \otimes \mathsf{S}_2)X \qquad [\pi_2]\varphi_2 ::= \pi(\mathsf{tt}, \varphi_2) \in (\mathsf{S}_1 \otimes \mathsf{S}_2)X$$
$$[\kappa_i]\varphi_i ::= \neg\langle\kappa_i\rangle(\neg\varphi_i) \in \overline{(\mathsf{S}_1 \oplus \mathsf{S}_2)X} \qquad \text{if } \varphi_i \in \overline{\mathsf{S}_i X} \text{ for } i = 1, 2.$$

Definition 14 (Combinations of proof system constructors). *Let $(L_1, \vdash_1)$ and $(L_2, \vdash_2)$ be boolean preorders.*

(i) We let $(L_1, \vdash_1) \otimes (L_2, \vdash_2) = (\overline{L_1 \otimes L_2}, \vdash_\otimes)$, where $L_1 \otimes L_2$ is as in Definition 4, and the relation $\vdash_\otimes$ is generated by the following axioms and rules:

$$\vdash_\otimes [\pi_i]\mathsf{tt} \qquad [\pi_i]\varphi \wedge [\pi_i]\psi \vdash_\otimes [\pi_i](\varphi \wedge \psi) \qquad \frac{\varphi \vdash_i \psi}{[\pi_i]\varphi \vdash_\otimes [\pi_i]\psi}$$

augmented with the axioms and rules of propositional logic.

(ii) We let $(L_1, \vdash_1) \oplus (L_2, \vdash_2) = (\overline{L_1 \oplus L_2}, \vdash_\oplus)$, where $L_1 \oplus L_2$ is as in Definition 4, and the relation $\vdash_\oplus$ is generated by the following axioms and rules:

$$\vdash_\oplus [\kappa_i]\mathsf{tt} \qquad [\kappa_i]\varphi \wedge [\kappa_i]\psi \vdash_\oplus [\kappa_i](\varphi \wedge \psi)$$
$$[\kappa_1]\mathsf{ff} \wedge [\kappa_2]\mathsf{ff} \vdash_\oplus \mathsf{ff} \qquad \frac{\varphi \vdash_i \psi}{[\kappa_i]\varphi \vdash_\oplus [\kappa_i]\psi}$$

augmented with the axioms and rules of propositional logic.

If P_1 and P_2 are proof system constructors for S_1 and S_2, respectively, define:

$$(\mathsf{P}_1 \otimes \mathsf{P}_2)(L, \vdash) = \mathsf{P}_1(L, \vdash) \otimes \mathsf{P}_2(L, \vdash) \quad (\mathsf{P}_1 \oplus \mathsf{P}_2)(L, \vdash) = \mathsf{P}_1(L, \vdash) \oplus \mathsf{P}_2(L, \vdash)$$
$$(\mathsf{P}_1 \circledcirc \mathsf{P}_2)(L, \vdash) = \mathsf{P}_1(\mathsf{P}_2(L, \vdash))$$

With these definitions we obtain that soundness and completeness is preserved by combinations of proof system constructors; for readability we have suppressed the dependency of the one-step semantics on the endofunctor.

Proposition 7. *Suppose* P_i *is a proof system constructor for* S_i*, for* $i = 1, 2$*. Then,* $\mathsf{P}_1 \otimes \mathsf{P}_2$*,* $\mathsf{P}_1 \oplus \mathsf{P}_2$ *and* $\mathsf{P}_1 \circledcirc \mathsf{P}_2$ *are proof system constructors for* $\mathsf{S}_1 \otimes \mathsf{S}_2$*,* $\mathsf{S}_1 \oplus \mathsf{S}_2$ *and* $\mathsf{S}_1 \circledcirc \mathsf{S}_2$*, respectively. Moreover, if* P_1 *and* P_2 *are one-step sound (complete) w.r.t.* $[\![\mathsf{S}_1]\!]$ *and* $[\![\mathsf{S}_2]\!]$*, respectively, then* $\mathsf{P}_1 \otimes \mathsf{P}_2$*,* $\mathsf{P}_1 \oplus \mathsf{P}_2$ *and* $\mathsf{P}_1 \circledcirc \mathsf{P}_2$ *are one-step sound (complete) w.r.t.* $[\![\mathsf{S}_1]\!] \otimes [\![\mathsf{S}_2]\!]$*,* $[\![\mathsf{S}_1]\!] \oplus [\![\mathsf{S}_2]\!]$ *and* $[\![\mathsf{S}_1]\!] \circledcirc [\![\mathsf{S}_2]\!]$*, respectively.*

Note that, if P_1 and P_2 are defined in terms of axioms and rules, all their combinations can be described in the same way.

As we have already argued in the beginning, a large class of probabilistic systems can be modelled as coalgebras of signature functors of the following form:

$$T ::= A \mid \mathcal{I}d \mid \mathcal{P}_f \mid \mathcal{D} \mid T_1 \times T_2 \mid T_1 + T_2 \mid T_1 \circ T_2.$$

We can therefore use Propositions 1, 2 and 7 to derive, for any probabilistic system type of the above form, a logic which is sound, complete and expressive.

Example 10 (Probabilistic automata). Simple probabilistic automata [28] are modelled coalgebraically using the functor $TX = \mathcal{P}_f(A \times \mathcal{D}X)$. The language $\mathcal{L} = \mathcal{L}(T)$ obtained by applying the modular techniques presented earlier can be described by the following grammar:

$$\begin{array}{lll} \mathcal{L} \ni \varphi ::= \mathsf{ff} \mid \varphi \to \varphi' \mid \Box\psi & & (\psi \in \mathcal{L}') \\ \mathcal{L}' \ni \psi ::= \mathsf{ff} \mid \psi \to \psi' \mid (a, \xi) & & (\xi \in \mathcal{L}'') \\ \mathcal{L}'' \ni \xi ::= \mathsf{ff} \mid \xi \to \xi' \mid L_p\varphi & & (\varphi \in \mathcal{L}) \end{array}$$

The coalgebraic semantics and the associated proof system are similarly given by a three layer construction.

References

1. P. Aczel and N. Mendler. A final coalgebra theorem. In D. H. Pitt et al, editor, *Category Theory and Computer Science*, volume 389 of *LNCS*. Springer, 1989.
2. F. Bartels. *On generalised coinduction and probabilistic specification formats*. PhD thesis, CWI, Amsterdam, 2004.
3. F. Bartels, A. Sokolova, and E. de Vink. A hierarchy of probabilistic system types. In H.P. Gumm, editor, *Proc. CMCS 2003*, volume 82 of *ENTCS*. Elsevier, 2003.
4. M. Bidoit, M. V. Cengarle, and R. Hennicker. Proof systems for structured specifications and their refinements. In H. J. Kreowski, E. Astesiano, and B. Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, IFIP State-of-the-Art Reports, pages 385–434. Springer, 1999.
5. P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2000.
6. T. Borzyszkowski. Logical systems for structured specifications. *Theoret. Comput. Sci.*, 286(2):197–245, 2002.
7. C. Caleiro. *Combining Logics*. PhD thesis, 2002.

8. C. Cîrstea. On expressivity and compositionality in logics for coalgebras. In H.P. Gumm, editor, *Proc. CMCS 2003*, volume 82 of *ENTCS*. Elsevier, 2003.
9. E. Clarke and E. Emerson. Synthesis of synchronisation skeletons for branching temporal logics. In *Workshop on Logics of Programs*, volume 131 of LNCS. Springer, 1981.
10. E. de Vink and J. Rutten. Bisimulation for probabilistic transition systems: a coalgebraic approach. *Theoret. Comput. Sci.*, 221:271–293, 1999.
11. D. Gabbay. *Fibring Logics*. Oxford University Press, 1998.
12. A. Heifetz and P. Mongin. Probability logic for type spaces. *Games and Economic Behaviour*, 35:31–53, 2001.
13. B. Jacobs. Many-sorted coalgebraic modal logic: a model-theoretic study. *Theor. Inform. Appl.*, 35(1):31–59, 2001.
14. M. Kick. Bialgebraic modelling of timed processes. In P. Widmayer, F. Ruiz, R. Bueno, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *Proc. ICALP 2002*, volume 2380 of *LNCS*. Springer, 2002.
15. D. Kozen. Results on the propositional mu-calculus. *Theoret. Comput. Sci.*, 27:333–354, 1983.
16. A. Kurz. Specifying Coalgebras with Modal Logic. *Theoret. Comput. Sci.*, 260(1–2):119–138, 2001.
17. A. Kurz and D. Pattinson. Definability, canonical models, and compactness for finitary coalgebraic modal logic. In L.S. Moss, editor, *Proc. CMCS 2002*, volume 65 of *ENTCS*. Elsevier, 2002.
18. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inform. and Comput.*, 94:1–28, 1991.
19. R. Milner. *Communication and Concurrency*. International series in computer science. Prentice Hall, 1989.
20. L.S. Moss. Coalgebraic logic. *Ann. Pure Appl. Logic*, 96:277–317, 1999.
21. P. Naur and B. Randell, editors. *Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968*. Scientific Affairs Division, NATO, 1969.
22. D. Park. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI Conference*, volume 104 of *LNCS*. Springer, 1981.
23. D. Pattinson. Coalgebraic modal logic: Soundness, completeness and decidability of local consequence. *Theoret. Comput. Sci.*, 309(1–3):177–193, 2003.
24. D. Pattinson. Expressive logics for coalgebras via terminal sequence induction. *Notre Dame J. Formal Logic*, 2004. To appear.
25. R.T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.
26. M. Rößiger. From Modal Logic to Terminal Coalgebras. *Theoret. Comput. Sci.*, 260:209–228, 2001.
27. J.J.M.M. Rutten. Universal coalgebra: A theory of systems. *Theoret. Comput. Sci.*, 249:3–80, 2000.
28. R. Segala. *Modelling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995.
29. D. Turi and G. Plotkin. Towards a mathematical operational semantics. In *Proc. 12$^{\text{th}}$ LICS Conference*, pages 280–291. IEEE, Computer Society Press, 1999.
30. M. Wirsing. Algebraic specification. In J. van Leuween, editor, *Handbook of Theoretical Computer Science*. Elsevier, 1990.
31. J. Worrell. On the Final Sequence of an Accessible Set Functor. *Theoret. Comput. Sci.*, 2003. To appear.
32. A. Zanardo, A. Sernadas, and C. Sernadas. Fibring: Completeness preservation. *J. Symbolic Logic*, 66(1):414–439, 2001.