QUASILINEAR-TIME COMPUTATION OF GENERIC MODAL WITNESSES FOR BEHAVIOURAL INEQUIVALENCE

THORSTEN WISSMANN [a], STEFAN MILIUS [b], AND LUTZ SCHRÖDER [b]

[a] Radboud University, Nijmegen, The Netherlands

[b] Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

ABSTRACT. We provide a generic algorithm for constructing formulae that distinguish behaviourally inequivalent states in systems of various transition types such as nondeterministic, probabilistic or weighted; genericity over the transition type is achieved by working with coalgebras for a set functor in the paradigm of universal coalgebra. For every behavioural equivalence class in a given system, we construct a formula which holds precisely at the states in that class. The algorithm instantiates to deterministic finite automata, transition systems, labelled Markov chains, and systems of many other types. The ambient logic is a modal logic featuring modalities that are generically extracted from the functor; these modalities can be systematically translated into custom sets of modalities in a postprocessing step. The new algorithm builds on an existing coalgebraic partition refinement algorithm. It runs in time $\mathcal{O}((m + n) \log n)$ on systems with n states and m transitions, and the same asymptotic bound applies to the dag size of the formulae it constructs. This improves the bounds on run time and formula size compared to previous algorithms even for previously known specific instances, viz. transition systems and Markov chains; in particular, the best previous bound for transition systems was $\mathcal{O}(mn)$.

1. INTRODUCTION

For finite transition systems, the Hennessy-Milner theorem guarantees that two states are bisimilar if and only if they satisfy the same modal formulae. Equivalently, this means that whenever two states are not bisimilar, then one can find a modal formula that holds at one of the states but not at the other. Such a formula, usually called a *distinguishing formula* [Cle91], explains the difference in the behaviour of the two states. For example, in the transition system in Figure 1, the formula $\Box\Diamond\top$ distinguishes the states x and y; specifically it is satisfied at x but not at y. This gives rise to the verification task of actually computing distinguishing formulae. Cleaveland [Cle91] presents an algorithm that computes distinguishing formulae for states in a finite transition system with n states and m transitions in time $\mathcal{O}(mn)$. The algorithm builds on the Kanellakis-Smolka partition

FIGURE 1. Example of a
transition system

FIGURE 2. Example of a
Markov chain

refinement algorithm [KS83, KS90], which computes the bisimilarity relation on a transition system within the same time bound.

Logical characterizations of bisimulation analogous to the Hennessy-Milner theorem exist for other system types. For instance, Desharnais et al. [DEP98, DEP02] characterize probabilistic bisimulation on (labelled) Markov chains, in the sense of Larsen and Skou [LAS91] (for each label, every state has either no successors or a probability distribution on successors). In their logic, a formula $\Diamond_{\geq p}\phi$ holds at states that have a transition probability of at least p to states satisfying ϕ. For example, the state x in Figure 2 satisfies $\Diamond_{\geq 0.5}\Diamond_{\geq 1}\top$ but y does not. Desharnais et al. provide an algorithm that computes distinguishing formulae for labelled Markov chains in run time (roughly) $\mathcal{O}(n^4)$.

In the present work, we construct such counterexamples generically for a variety of system types. We achieve genericity over the system type by modelling state-based systems as coalgebras for a set functor in the framework of universal coalgebra [Rut00]. Examples of coalgebras for a set functor include transition systems, deterministic automata, or weighted systems (e.g. Markov chains). Universal coalgebra provides a generic notion of behavioural equivalence that instantiates to standard notions for concrete system types, e.g. bisimilarity (transition systems), language equivalence (deterministic automata), or probabilistic bisimilarity (Markov chains). Moreover, coalgebras come equipped with a generic notion of modal logic that is parametric in a choice of modalities whose semantics is constructed so as to guarantee invariance w.r.t. behavioural equivalence; under easily checked conditions, such a *coalgebraic modal logic* in fact characterizes behavioural equivalence in the same sense as Hennessy-Milner logic characterizes bisimilarity [Pat04, Sch08]. Hence, as soon as suitable modal operators are found, coalgebraic modal formulae serve as distinguishing formulae.

In a nutshell, the contribution of the present paper is an algorithm that computes distinguishing formulae for behaviourally inequivalent states, and in fact *certificates* that uniquely describe behavioural equivalence classes in a system, in *quasilinear time* and in coalgebraic generality. We build on an existing efficient coalgebraic partition refinement algorithm [WDMS20], thus achieving run time $\mathcal{O}(m \log n)$ on coalgebras with n states and m transitions (in a suitable encoding). The dag size of formulae is also $\mathcal{O}(m \log n)$ (for tree size, exponential lower bounds are known [FG10]); even for the basic case of transition systems, we thus improve the previous best bound $\mathcal{O}(mn)$ [Cle91] for both run time and formula size. We systematically extract the requisite modalities from the functor at hand, requiring binary and nullary modalities in the general case, and then give a systematic method to translate these generic modal operators into more customary ones (such as the standard operators of Hennessy-Milner logic).

We subsequently identify a notion of *cancellative* functor that allows for additional optimization. E.g. functors modelling weighted systems are cancellative if and only if the weights come from a cancellative monoid, such as $(\mathbb{Z}, +)$, or $(\mathbb{R}, +)$ as used in probabilistic systems. For cancellative functors, much simpler distinguishing formulae can be constructed: the binary modalities can be replaced by unary ones, and only conjunction is needed in the propositional base. On labelled Markov chains, this complements the result that a logic

with only conjunction and different unary modalities (the modalities $\Diamond_{\geq p}$ mentioned above) suffices for the construction of distinguishing formulae (but not certificates) [DEP02] (see also [Dob09]).

Related Work. As mentioned above, Cleaveland's algorithm for labelled transition systems [Cle91] is based on Kanellakis and Smolka's partition refinement algorithm [KS90], while the coalgebraic partition refinement algorithm we employ [WDMS20] is instead related to the more efficient Paige-Tarjan algorithm [PT87]. We do note that in the current paper we formally cover only unlabelled transition systems; the labelled case requires an elaboration of compositionality mechanisms in coalgebraic logic, which is not in the technical focus of the present work. Details are discussed in Remark 3.7. Hopcroft's automata minimization algorithm [Hop71] and its generalization to variable input alphabets [Gri73, Knu01] have quasi-linear run time; in these algorithms, a word distinguishing two inequivalent states of interest can be derived directly from a run of the algorithm. König et al. [KMMS20] extract formulae from winning strategies in a bisimulation game in coalgebraic generality, under more stringent restrictions on the functor than we employ here (specifically, they assume that the functor is *separable by singletons*, which is stronger than our requirement that the functor is *zippable* [KMMS20, Lemma 14]). Their algorithm runs in $\mathcal{O}(n^4)$; it does not support negative transition weights. Characteristic formulae for behavioural equivalence classes taken across *all* models require the use of fixpoint logics [DMSW18]. The mentioned algorithm by Desharnais et al. for distinguishing formulae on labelled Markov processes [DEP02, Fig. 4] is based on Cleaveland's. No complexity analysis is made but the algorithm has four nested loops, so its run time is roughly $\mathcal{O}(n^4)$. Bernardo and Miculan [BM19] provide a similar algorithm for a logic with only disjunction. There are further generalizations along other axes, e.g. to behavioural preorders [CC95]. The TwoTowers tool set for the analysis of stochastic process algebras [BCSS98, Ber04] computes distinguishing formulae for inequivalent processes, using variants of Cleaveland's algorithm. Some approaches construct alternative forms of certificates for inequivalence, such as Cranen et al.'s notion of evidence [CLW15] or methods employed on business process models, based on model differences and event structures [Dij08, AGD13, ABDG14].

In constructive mathematics, *apartness relations* capture provable difference of elements, and recently, Geuvers and Jacobs [GJ21] introduced apartness relations as an inductive notion for the inequality of states in a coalgebra, or in general, in a state-based system. In active automata learning, Vaandrager et al. [VGRW22] base their learning algorithm $L^{\#}$ on an apartness notion for automata. Whenever two states turn out to be apart, this is *witnessed* by an input word for which the two states behave differently, and these witnesses are used in the subsequent learning process. Instead of words, we construct modal formulae as universal witnesses for systems of different type. In this sense, our results may eventually relate to variants of coalgebraic active automata learning in which words are similarly replaced with coalgebraic modal formulae [BKR19].

This paper is an extended and revised version of a conference publication [WMS21]. It contains full proofs as well as additional material on simplifications that apply in case the coalgebra functor is cancellative (Section 4). Moreover, we include a new, elementary proof of the known fact that the tree size of certificates can be exponential [FG10] in Appendix A.

2. Preliminaries

We first recall some basic notation. We denote by $0 = \emptyset$, $1 = \{0\}$, $2 = \{0, 1\}$, and $3 = \{0, 1, 2\}$ the sets representing the natural numbers 0, 1, 2 and 3. For every set X, there is a unique map $!\colon X \to 1$. We write Y^X for the set of functions $X \to Y$, so e.g. $X^2 \cong X \times X$. In particular, 2^X is the set of 2-valued *predicates* on X, which is in bijection with the *powerset* $\mathcal{P}X$ of X, i.e. the set of all subsets of X; in this bijection, a subset $A \in \mathcal{P}X$ corresponds to its *characteristic function* $\chi_A \in 2^X$, given by $\chi_A(x) = 1$ if $x \in A$, and $\chi(x) = 0$ otherwise. We freely convert between predicates and subsets; in particular we apply set operations as well as the subset and elementhood relations to predicates, with the evident meaning. We generally indicate injective maps by $\rightarrowtail$. Given maps $f\colon Z \to X$, $g\colon Z \to Y$, we write $\langle f, g \rangle$ for the map $Z \to X \times Y$ given by $\langle f, g \rangle(z) = (f(z), g(z))$. We denote the disjoint union of sets X, Y by $X + Y$, with canonical inclusion maps

$$\mathsf{in}_1 \colon X \rightarrowtail X + Y \qquad \text{and} \qquad \mathsf{in}_2 \colon Y \rightarrowtail X + Y.$$

More generally, we write $\coprod_{i \in I} X_i$ for the disjoint union of an I-indexed family of sets $(X_i)_{i \in I}$, and $\mathsf{in}_i \colon X_i \rightarrowtail \coprod_{i \in I} X_i$ for the i-th inclusion map. For a map $f\colon X \to Y$ (not necessarily surjective), we denote by $\ker(f) \subseteq X \times X$ the *kernel* of f, i.e. the equivalence relation

$$\ker(f) := \{(x, x') \in X \times X \mid f(x) = f(x')\}. \tag{2.1}$$

Notation 2.1 (Partitions)**.** Given an equivalence relation R on X, we write $[x]_R$ for the equivalence class $\{x' \in X \mid (x, x') \in R\}$ of $x \in X$. If R is the kernel of a map f, we simply write $[x]_f$ in lieu of $[x]_{\ker(f)}$. The partition corresponding to R is denoted by

$$X/R = \{[x]_R \mid x \in X\}.$$

Note that $[-]_R \colon X \to X/R$ is a surjective map and that $R = \ker([-]_R)$.

A *signature* is a set Σ, whose elements are called *operation symbols*, equipped with a function $a\colon \Sigma \to \mathbb{N}$ assigning to each operation symbol its *arity*. We write $\sigma/n \in \Sigma$ for $\sigma \in \Sigma$ with $a(\sigma) = n$. We will apply the same terminology and notation to collections of modal operators.

2.1. **Coalgebra.** *Universal coalgebra* [Rut00] provides a generic framework for the modelling and analysis of state-based systems. Its key abstraction is to parametrize notions and results over the transition type of systems, encapsulated as an endofunctor on a given base category. Instances cover, for example, deterministic automata, labelled (weighted) transition systems, and Markov chains.

Definition 2.2. A *set functor* $F\colon \mathsf{Set} \to \mathsf{Set}$ assigns to every set X a set FX and to every map $f\colon X \to Y$ a map $Ff\colon FX \to FY$ such that identity maps and composition are preserved: $F\mathsf{id}_X = \mathsf{id}_{FX}$ and $F(g \cdot f) = Fg \cdot Ff$ whenever the composite $g \cdot f$ is defined. An *F-coalgebra* is a pair (C, c) consisting of a set C (the *carrier*) and a map $c\colon C \to FC$ (the *structure*). When F is clear from the context, we simply speak of a *coalgebra*.

In a coalgebra $c\colon C \to FC$, we understand the carrier set C as consisting of *states*, and the structure c as assigning to each state $x \in C$ a structured collection of successor states, with the structure of collections determined by F. In this way, the notion of coalgebra subsumes numerous types of state-based systems, as illustrated next.

Example 2.3. (1) The *powerset functor* $\mathcal{P}$ sends a set X to its powerset $\mathcal{P}X$ and a map $f\colon X \to Y$ to the map $\mathcal{P}f = f[-]\colon \mathcal{P}X \to \mathcal{P}Y$ that takes direct images. A $\mathcal{P}$-coalgebra $c\colon C \to \mathcal{P}C$ is precisely a transition system: It assigns to every state $x \in C$ a set $c(x) \in \mathcal{P}C$ of *successor* states, inducing a transition relation $\to$ given by $x \to y$ iff $y \in c(x)$. Similarly, the coalgebras for the finite powerset functor $\mathcal{P}_{\mathsf{f}}$ (with $\mathcal{P}_{\mathsf{f}}X$ being the set of finite subsets of X) are precisely the finitely branching transition systems.

(2) Coalgebras for the functor $FX = 2 \times X^A$, where A is a fixed input alphabet, are deterministic automata (without an explicit initial state). Indeed, a coalgebra structure $c = \langle f, t \rangle\colon C \to 2 \times C^A$ consists of a finality predicate $f\colon C \to 2$ and a transition map $C \times A \to C$ in curried form $t\colon C \to C^A$.

(3) Every signature Σ defines a *signature functor* that maps a set X to the set

$$F_\Sigma X = \coprod_{\sigma/n \in \Sigma} X^n,$$

whose elements we may understand as flat Σ-terms $\sigma(x_1, \ldots, x_n)$ with variables from X. The action of F_Σ on maps $f\colon X \to Y$ is then given by

$$F_\Sigma f\colon F_\Sigma X \to F_\Sigma Y \qquad (F_\Sigma f)(\sigma(x_1, \ldots, x_n)) = \sigma(f(x_1), \ldots, f(x_n)).$$

For simplicity, we write σ (instead of in_σ) for the coproduct injections, and Σ in lieu of F_Σ for the signature functor. A Σ-coalgebra is a kind of tree automaton: it consists of a set C of states and a transition map $c\colon C \to \coprod_{\sigma/n \in \Sigma} C^n$, which essentially assigns to each state an operation symbol $\sigma/n \in \Sigma$ and n-successor states. Hence, every state in a Σ-coalgebra describes a (possibly infinite) Σ-tree, that is, a rooted and ordered tree whose nodes are labelled by operation symbols from Σ such that a node with n successor nodes is labelled by an n-ary operation symbol.

(4) For a commutative monoid $(M, +, 0)$, the *monoid-valued functor* $M^{(-)}$ [GS01] is defined on a set X by

$$M^{(X)} := \{\mu\colon X \to M \mid \mu(x) = 0 \text{ for all but finitely many } x \in X\}. \qquad (2.2)$$

For a map $f\colon X \to Y$, the map $M^{(f)}\colon M^{(X)} \to M^{(Y)}$ is defined by

$$(M^{(f)})(\mu)(y) = \sum_{x \in X, f(x)=y} \mu(x).$$

A coalgebra $c\colon C \to M^{(C)}$ is a finitely branching weighted transition system: for $x, x' \in C$, $c(x)(x') \in M$ is the transition weight from x to x'. For the Boolean monoid $\mathbb{B} = (2, \vee, 0)$, we recover $\mathcal{P}_{\mathsf{f}} = \mathbb{B}^{(-)}$. Coalgebras for $\mathbb{R}^{(-)}$, with $\mathbb{R}$ understood as the additive monoid of the reals, are $\mathbb{R}$-weighted transition systems. The functor

$$\mathcal{D}X = \{\mu \in \mathbb{R}_{\geq 0}^{(X)} \mid \sum_{x \in X} \mu(x) = 1\},$$

which assigns to a set X the set of all finite probability distributions on X (represented as finitely supported probability mass functions), is a subfunctor of $\mathbb{R}^{(-)}$.

(5) Functors can be composed; for instance, given a set A of labels, the composite of $\mathcal{P}$ and the functor $A \times (-)$ (whose action on sets maps a set X to the set $A \times X$) is the functor $FX = \mathcal{P}(A \times X)$, whose coalgebras are A-labelled transition systems. Coalgebras for $(\mathcal{D}(-) + 1)^A$ have been termed *probabilistic transition systems* [LAS91] or *labelled Markov chains* [DEP02], and coalgebras for $(\mathcal{D}((-) + 1) + 1)^A$ are *partial labelled Markov chains* [DEP02]. Coalgebras for $SX = \mathcal{P}_{\mathsf{f}}(A \times \mathcal{D}X)$ are variously known as *simple Segala systems* or *Markov decision processes*.

We have a canonical notion of *behaviour* on F-coalgebras:

Definition 2.4. An F-coalgebra *morphism* $h\colon (C,c) \to (D,d)$ is a map $h\colon C \to D$ such that the square below commutes:

$$
\begin{array}{ccc}
C & \xrightarrow{\;c\;} & FC \\
{\scriptstyle h}\downarrow & & \downarrow{\scriptstyle Fh} \\
D & \xrightarrow{\;d\;} & FD
\end{array}
$$

States x, y in an F-coalgebra (C,c) are *behaviourally equivalent* (notation: $x \sim y$) if there exists a coalgebra morphism h such that $h(x) = h(y)$.

Thus, we effectively define the behaviour of a state as those of its properties that are preserved by coalgebra morphisms. The notion of behavioural equivalence subsumes standard branching-time equivalences:

Example 2.5. (1) For $F \in \{\mathcal{P}, \mathcal{P}_\mathsf{f}\}$, behavioural equivalence on F-coalgebras, i.e. on transition systems, is *bisimilarity* in the usual sense.

(2) For deterministic automata as coalgebras for $FX = 2 \times X^A$, two states are behaviourally equivalent iff they accept the same formal language.

(3) For a signature functor Σ, two states of a Σ-coalgebra are behaviourally equivalent iff they describe the same Σ-tree.

(4) For labelled transition systems as coalgebras for $FX = \mathcal{P}(A \times X)$, coalgebraic behavioural equivalence precisely captures Milner's strong bisimilarity; this was shown by Aczel and Mendler [AM89].

(5) For weighted and probabilistic systems, coalgebraic behavioural equivalence instantiates to weighted and probabilistic bisimilarity, respectively [RdV99, Cor. 4.7], [BSdV04, Thm. 4.2].

Remark 2.6. (1) The notion of behavioural equivalence extends straightforwardly to states in different coalgebras, as one can canonically define the disjoint union of coalgebras:

Given a pair of F-coalgebra (C,c) and (D,d), we have a canonical F-coalgebra structure on the disjoint union $C + D$ of their carriers:

$$
C + D \xrightarrow{\;c+d\;} FC + FD \xrightarrow{\;[F\mathsf{in}_1, F\mathsf{in}_2]\;} F(C + D),
$$

where $[-, -]$ denotes case distinction on the disjoint components FC and FD. It is easy to see that the canonical inclusion maps $\mathsf{in}_1\colon C \to C + D$ and $\mathsf{in}_2\colon D \to C + D$ are F-coalgebra morphisms. We say that states $x \in C$ and $y \in D$ are *behaviourally equivalent* if $\mathsf{in}_1(x) \sim \mathsf{in}_2(y)$ holds in $C + D$. This definition coincides with the standard one, according to which x, y are behaviourally equivalent if there exist coalgebra morphisms $f\colon (C,c) \to (E,e)$ and $g\colon (D,d) \to (E,e)$ such that $f(x) = g(y)$. Moreover, the extended definition is consistent with Definition 2.4 in the sense that states x, y in the coalgebra (C,c) are behaviourally equivalent according to Definition 2.4 iff $\mathsf{in}_1(x) \sim \mathsf{in}_2(y)$ in the canonical coalgebra on $C + C$.

(2) As shown by Trnková [Trn71], we may assume without loss of generality that a set functor F preserves injective maps (see also Barr [Bar93, Proof of Thm. 3.2]) that is, Ff is injective whenever f is. In fact, for every set functor F there exists a set functor $\bar{F}$ (called the *Trnková hull* of F [ABLM12]) that coincides with F on nonempty sets and functions and preserves injections.

Moreover, $\bar{F}$ preserves all finite intersections (pullbacks of pairs of injective maps); it is even the reflection of F in the category of set functors preserving finite intersections [ABLM12, Cor. VII.2].

Since $\bar{F}$ only differs from F on the empty set, both functors have the same coalgebras and coalgebra morphisms. All functors in Example 2.3 already preserve injective maps.

2.2. Coalgebraic Logics.

We continue with a brief review of basic concepts of coalgebraic modal logic [Pat03, Sch08]. Coalgebraic modal logics are parametric in a functor F determining the type of systems underlying the semantics, and additionally in a choice of modalities interpreted in terms of *predicate liftings*. For now, we use $F = \mathcal{P}$ as a basic example, deferring further examples to Section 5.

Syntax. The syntax of coalgebraic modal logic is parametrized over the choice of a signature Λ of *modal operators* (with assigned arities). Then, *formulae* ϕ are generated by the following grammar

$$\phi_1, \ldots, \phi_n ::= \top \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid \heartsuit(\phi_1, \ldots, \phi_n) \qquad (\heartsuit/n \in \Lambda).$$

Example 2.7. For $F = \mathcal{P}$, one often takes $\Lambda = \{\Diamond/1\}$; the induced syntax is that of (single-action) Hennessy-Milner logic. As usual, we write $\Box\phi :\equiv \neg\Diamond\neg\phi$.

Semantics. We interpret formulae as sets of states in F-coalgebras. This interpretation arises by assigning to each modal operator $\heartsuit/n \in \Lambda$ an n-ary *predicate lifting* $\llbracket\heartsuit\rrbracket$ [Pat03, Sch08], i.e. a family of maps $\llbracket\heartsuit\rrbracket_X\colon (2^X)^n \to 2^{FX}$, one for every set X, such that the *naturality* condition

$$Ff^{-1}\big[\llbracket\heartsuit\rrbracket_Y(P_1, \ldots, P_n)\big] = \llbracket\heartsuit\rrbracket_X(f^{-1}[P_1], \ldots, f^{-1}[P_n]) \qquad (2.3)$$

holds for all $f\colon X \to Y$ and all $P_1, \ldots, P_n \in 2^X$. Thus, $\llbracket\heartsuit\rrbracket_X$ lifts n given predicates on states to a predicate on structured collections of states. Categorically speaking, $\llbracket\heartsuit\rrbracket$ is a natural transformation $(2^{(-)})^n \to 2^{F^{\mathrm{op}}}$; that is, the naturality square

$$
\begin{array}{ccc}
(2^Y)^n & \xrightarrow{\;\llbracket\heartsuit\rrbracket_Y\;} & 2^{FY} \\
{\scriptstyle(2^f)^n}\big\downarrow & & \big\downarrow{\scriptstyle 2^{Ff}} \\
(2^X)^n & \xrightarrow{\;\llbracket\heartsuit\rrbracket_Y\;} & 2^{FX}
\end{array}
$$

commutes for $f\colon X \to Y$. Explicitly, $2^{(-)}$ denotes the contravariant powerset functor, which sends a set X to the set 2^X of 2-valued predicates on X (equivalently to the powerset of X), and a map $f\colon X \to Y$ to the inverse image map $2^f = f^{-1}[-]\colon 2^Y \to 2^X$; writing down the commutativity of the above square element-wise then yields precisely (2.3). By the Yoneda lemma, one equivalently can define predicate liftings as follows:

Lemma 2.8 [Sch08, Proposition 43]. *Predicate liftings* $(2^X)^n \to 2^{FX}$ *of arity n are in one-to-one correspondence with subsets of $F(2^n)$. The correspondence sends a subset $S \subseteq F(2^n)$ to the predicate lifting*

$$\lambda_X(P_1, \ldots, P_n) = \{t \in FX \mid F\underbrace{\langle P_1, \ldots, P_n\rangle}_{X \to 2^n}(t) \in S\} \qquad (P_i\colon X \to 2).$$

Given the above data, the *extension* of a formula ϕ in an F-coalgebra (C, c) is a predicate $\llbracket\phi\rrbracket_{(C,c)}$, or just $\llbracket\phi\rrbracket$, on C, recursively defined by

$$\llbracket\top\rrbracket_{(C,c)} = C, \qquad \llbracket\phi \wedge \psi\rrbracket_{(C,c)} = \llbracket\phi\rrbracket_{(C,c)} \cap \llbracket\psi\rrbracket_{(C,c)}, \qquad \llbracket\neg\phi\rrbracket_{(C,c)} = C \setminus \llbracket\phi\rrbracket_{(C,c)},$$

$$\llbracket\heartsuit(\phi_1, \ldots, \phi_n)\rrbracket_{(C,c)} = c^{-1}\big[\llbracket\heartsuit\rrbracket_C\big(\llbracket\phi_1\rrbracket_{(C,c)}, \ldots, \llbracket\phi_n\rrbracket_{(C,c)}\big)\big] \qquad \text{for } (\heartsuit/n \in \Lambda).$$

(Recall that we implicitly convert between predicates and subsets.) We say that a state $x \in C$ *satisfies* ϕ if $x \in \llbracket\phi\rrbracket$. Notice how the clause for modalities says that x satisfies $\heartsuit(\phi_1, \ldots, \phi_n)$ iff $c(x)$ satisfies the predicate obtained by lifting the predicates $\llbracket\phi_1\rrbracket, \ldots, \llbracket\phi_n\rrbracket$ on C to a predicate on FC according to $\llbracket\heartsuit\rrbracket$.

Example 2.9. Over $F = \mathcal{P}$, we interpret $\Diamond$ by the predicate lifting

$$\llbracket\Diamond\rrbracket_X \colon 2^X \to 2^{\mathcal{P}X}, \quad P \mapsto \{K \subseteq X \mid \exists x \in K \colon x \in P\} = \{K \subseteq X \mid K \cap P \neq \emptyset\}.$$

The arising notion of satisfaction over $\mathcal{P}$-coalgebras (C, c) is precisely the standard one:

$$x \in \llbracket\Diamond\phi\rrbracket_{(C,c)} \qquad \text{iff} \qquad y \in \llbracket\phi\rrbracket_{(C,c)} \text{ for some transition } x \to y.$$

The naturality condition (2.3) of predicate liftings guarantees invariance of the logic under coalgebra morphisms, and hence under behavioural equivalence:

Proposition 2.10 (Adequacy [Pat03, Sch08])**.** *Behaviourally equivalent states satisfy the same formulae:* $x \sim y$ *implies that for all formulae* ϕ*, we have* $x \in \llbracket\phi\rrbracket$ *iff* $y \in \llbracket\phi\rrbracket$*.*

In our running example $F = \mathcal{P}$, this instantiates to the well-known fact that modal formulae are bisimulation-invariant, that is, bisimilar states in transition systems satisfy the same formulae of Hennessy-Milner logic.

3. Constructing Distinguishing Formulae

A proof method certifying behavioural equivalence of states x, y in a coalgebra is immediate by definition: One simply needs to exhibit a coalgebra morphism h such that $h(x) = h(y)$. In fact, for many system types, it suffices to relate x and y by a coalgebraic *bisimulation* in a suitable sense (e.g. [AM89, Rut00, GS13, MV15]), generalizing the Park-Milner bisimulation principle [Mil89, Par81]. It is less obvious how to certify behavioural *inequivalence* $x \not\sim y$, showing that such a morphism h does *not* exist. By Proposition 2.10, one option is to exhibit a (coalgebraic) modal formula ϕ that is satisfied by x but not by y. In the case of (image-finite) transition systems, such a formula is guaranteed to exist by the Hennessy-Milner theorem, which moreover is known to generalize to coalgebras [Pat04, Sch08]. More generally, we consider separation of *sets* of states by formulae, following Cleaveland [Cle91, Def. 2.4]:

Definition 3.1. Let (C, c) be an F-coalgebra. A formula ϕ *distinguishes* a set $X \subseteq C$ from a set $Y \subseteq C$ if $X \subseteq \llbracket\phi\rrbracket$ and $Y \cap \llbracket\phi\rrbracket = \emptyset$. In case $X = \{x\}$ and $Y = \{y\}$, we just say that ϕ *distinguishes* x *from* y. We say that ϕ is a *certificate* of X if ϕ distinguishes X from $C \setminus X$, that is if $\llbracket\phi\rrbracket = X$.

Note that ϕ distinguishes X from Y iff $\neg\phi$ distinguishes Y from X. Certificates have also been referred to as *descriptions* [FG10]. If ϕ is a certificate of a behavioural equivalence class $[x]_\sim$, then by definition, ϕ distinguishes x from y whenever $x \not\sim y$. To obtain distinguishing formulae for behaviourally inequivalent states in a coalgebra, it therefore suffices to construct certificates for all behavioural equivalence classes, which indeed is what our algorithm does.

Of course, every certificate must be at least as large as a smallest distinguishing formula. However, already on transition systems, distinguishing formulae and certificates have the same asymptotic worst-case size (cf. Section 6).

A natural approach to computing certificates for behavioural equivalence classes is to extend algorithms that compute these equivalence classes. In particular, *partition refinement* algorithms compute a sequence $C/R_0, C/R_1, \ldots$ of consecutively finer partitions (i.e. $R_{i+1} \subseteq R_i$ for every $i \geq 0$) on the state space, where every *block* $B \in C/R_i$ is a union of behavioural equivalence classes. Since C is finite, this sequence stabilizes, and the final partition is precisely $C/\sim$. Indeed, Cleaveland's algorithm for computing certificates on (labelled) transition systems [Cle91] correspondingly extends Kanellakis and Smolka's partition refinement algorithm [KS83, KS90], which runs in $\mathcal{O}(mn)$ on systems with $n = |C|$ states and m transitions. Our generic algorithm will be based on a more efficient partition refinement algorithm.

3.1. **Paige-Tarjan with Certificates.** Before we turn to constructing certificates in coalgebraic generality, we informally recall and extend the Paige-Tarjan algorithm [PT87], which computes the partition modulo bisimilarity of a given transition system with n states and m transitions in time $\mathcal{O}((m + n) \log n)$. We fix a given finite transition system, viewed as a $\mathcal{P}$-coalgebra $c \colon C \to \mathcal{P}C$.

The algorithm computes two sequences $(C/P_i)_{i \in \mathbb{N}}$ and $(C/Q_i)_{i \in \mathbb{N}}$ of partitions of C (with Q_i, P_i equivalence relations), where only the most recent partition is held in memory and i indexes the iterations of the main loop. Throughout the execution, C/P_i is finer than C/Q_i (that is, $P_i \subseteq Q_i$ for every $i \geq 0$), and the algorithm terminates when $P_i = Q_i$. Intuitively, P_i is 'one transition ahead' of Q_i: if Q_i distinguishes states x and y, then P_i is based on distinguishing transitions to x from transitions to y.

Initially, $C/Q_0 := \{C\}$ consists of only one block and C/P_0 of two blocks: the live states and the deadlocks (i.e. states with no outgoing transitions). If $P_i \subsetneq Q_i$, then there is a block $B \in C/Q_i$ that is the union of at least two blocks in C/P_i. In such a situation, the algorithm chooses $S \subseteq B$ in C/P_i to have at most half the size of B and then splits the block B into S and $B \setminus S$ in the partition C/Q_i:

$$C/Q_{i+1} = (C/Q_i \setminus \{B\}) \ \cup \ \{S, B \setminus S\}.$$

This is correct because every state in S is already known to be behaviourally inequivalent to every state in $B \setminus S$. By the definition of bisimilarity, this implies that every block $T \in C/P_i$ with some transition to B may contain behaviourally inequivalent states as illustrated in Figure 3; that is, T may need to be split into smaller blocks, as follows:

(C1) states in T with successors in S but not in $B \setminus S$ (e.g. x_1 in Figure 3),

(C2) states in T with successors in S and $B \setminus S$ (e.g. x_2), and

(C3) states in T with successors $B \setminus S$ but not in S (e.g. x_3).

The partition C/P_{i+1} arises from C/P_i by splitting all such predecessor blocks T of B accordingly. The algorithm terminates as soon as $P_{i+1} = Q_{i+1}$ holds. It is straightforward to construct certificates for the blocks arising during the execution:

- The certificate for the only block $C \in C/Q_0$ is $\top$, and the blocks for live states and deadlocks in C/P_0 have certificates $\Diamond \top$ and $\neg \Diamond \top$, respectively.
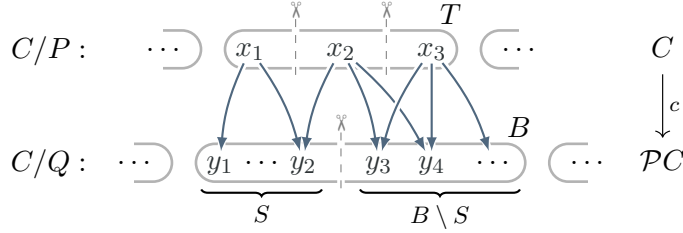
FIGURE 3. The refinement step as illustrated in [WDMS20, Figure 6].

- In the refinement step, suppose that δ, β are certificates of $S \in C/P_i$ and $B \in C/Q_i$, respectively, where $S \subsetneq B$. For every predecessor block T of B, the three blocks obtained by splitting T are distinguished (in the sense of Definition 3.1) as follows:

$$\textbf{(C1)} \quad \neg\Diamond(\beta \wedge \neg\delta), \qquad \textbf{(C2)} \quad \Diamond(\delta) \wedge \Diamond(\beta \wedge \neg\delta), \qquad \textbf{(C3)} \quad \neg\Diamond\delta. \qquad (3.1)$$

Of course these formulae only distinguish the states in T from each other (e.g. there may be states in other blocks with transitions to both S and B). Hence, given a certificate ϕ of T, one obtains certificates of the three resulting blocks in C/P_{i+1} via conjunction:

$$\textbf{(C1)} \quad \phi \wedge \neg\Diamond(\beta \wedge \neg\delta), \qquad \textbf{(C2)} \quad \phi \wedge \Diamond(\delta) \wedge \Diamond(\beta \wedge \neg\delta), \qquad \textbf{(C3)} \quad \phi \wedge \neg\Diamond\delta.$$

Upon termination, every bisimilarity class $[x]_\sim$ in the transition system is annotated with a certificate. A key step in the generic development will be to come up with a coalgebraic generalization of the formulae for **(C1)**–**(C3)**.

3.2. **Generic Partition Refinement.** The Paige-Tarjan algorithm has been adapted to other system types, e.g. weighted systems [VF10], and it has recently been generalized to coalgebras [DMSW17, WDMS20]. A crucial step in this generalization is to rephrase the case distinction **(C1)**–**(C3)** in terms of the functor $\mathcal{P}$: Given a predecessor block T in C/P_i for $S \subsetneq B \in C/Q_i$, we define the map $\chi_S^B \colon C \to 3$ by

$$\chi_S^B(x) = \begin{cases} 2 & \text{if } x \in S, \\ 1 & \text{if } x \in B \setminus S, \\ 0 & \text{if } x \in C \setminus B, \end{cases} \qquad \text{for sets } S \subseteq B \subseteq C. \qquad (3.2)$$

We then consider the composite

$$C \xrightarrow{\ c\ } \mathcal{P}C \xrightarrow{\ \mathcal{P}\chi_S^B\ } \mathcal{P}3.$$

The three cases **(C1)**–**(C3)** distinguish between the equivalence classes $[x]_{\mathcal{P}\chi_S^B \cdot c}$ for $x \in T$; that is, every case is a possible value of $t := \mathcal{P}\chi_S^B(c(x)) \in \mathcal{P}3$:

$$\textbf{(C1)} \ 2 \in t \not\ni 1, \qquad \textbf{(C2)} \ 2 \in t \ni 1, \text{ and} \qquad \textbf{(C3)} \ 2 \notin t \ni 1.$$

Since T is a predecessor block of B, the 'fourth case' $2 \notin t \not\ni 1$ is not possible. There is a transition from x to some state outside B iff $0 \in t$. However, because of the previous refinement steps performed by the algorithm, either every or no state of T has an edge to $C \setminus B$ (a property called *stability* [PT87]), hence no distinction on $0 \in t$ is necessary.

It is now easy to generalize from transition systems to coalgebras by simply replacing the functor $\mathcal{P}$ with F in the refinement step. We recall the algorithm:

Algorithm 3.2 [WDMS20, Alg. 4.9, (5.1)]**.** Given a coalgebra $c\colon C \to FC$, put

$$C/Q_0 := \{C\} \qquad \text{and} \qquad P_0 := \ker\bigl(C \xrightarrow{\ c\ } FC \xrightarrow{\ F!\ } F1\bigr).$$

Starting at iteration $i = 0$, repeat the following while $P_i \neq Q_i$:

(A1) Pick $S \in C/P_i$ and $B \in C/Q_i$ such that $S \subsetneqq B$ and $2 \cdot |S| \leq |B|$

(A2) $C/Q_{i+1} := (C/Q_i \setminus \{B\}) \cup \{S, B \setminus S\}$

(A3) $P_{i+1} := P_i \cap \ker\bigl(C \xrightarrow{\ c\ } FX \xrightarrow{\ F\chi_S^B\ } F3\bigr)$

This algorithm formalizes the intuitive steps from Section 3.1. Again, two sequences of partitions P_i, Q_i are constructed, and $P_i = Q_i$ upon termination. Initially, Q_0 identifies all states, and P_0 distinguishes states by only their output behaviour.

Example 3.3. (1) For $F = \mathcal{P}$ and $x \in C$, the value $\mathcal{P}!(c(x)) \in \mathcal{P}1$ is $\emptyset$ if x is a deadlock, and $\{1\}$ if x is a live state.

(2) For $FX = 2 \times X^A$, the value $F!(c(x)) \in F1 = 2 \times 1^A \cong 2$ indicates whether x is a final or non-final state.

In the main loop, blocks $S \in C/P_i$ and $B \in C/Q_i$ witnessing $P_i \subsetneqq Q_i$ are picked, and B is split into S and $B \setminus S$, like in the Paige-Tarjan algorithm. Note that step **(A2)** is equivalent to directly defining the equivalence relation Q_{i+1} as

$$Q_{i+1} := Q_i \cap \ker \chi_S^B.$$

A similar intersection of equivalence relations is performed in step **(A3)**. The intersection splits every block $T \in C/P_i$ into smaller blocks such that $x, x' \in T$ end up in the same block iff $F\chi_S^B(c(x)) = F\chi_S^B(c(x'))$, i.e. T is replaced with $\{[x]_{F\chi_S^B(c(x))} \mid x \in T\}$. Again, this corresponds to the distinction of the three cases **(C1)**–**(C3)**.

Example 3.4. For $FX = 2 \times X^A$, there are $|F3| = 2 \cdot 3^{|A|}$ cases to be distinguished, and so every $T \in C/P_i$ is split into at most that many blocks in C/P_{i+1}.

The following property of F is needed for correctness [WDMS20, Ex. 5.11].

Definition 3.5 [WDMS20]**.** A functor F is *zippable* if the following maps are injective:

$$\langle F(A+!), F(!+B) \rangle \colon F(A+B) \longrightarrow F(A+1) \times F(1+B) \qquad \text{for all sets } A, B.$$

Intuitively, $t \in F(A+B)$ is a structured collection of elements from A and B. If F is zippable, then t is uniquely determined by the two structured collections in $F(A+1)$ and $F(1+B)$ obtained by identifying all B- and all A-elements, respectively, with $0 \in 1$.

Example 3.6. The functor $FX = X \times X$ is zippable: $t = (\mathsf{in}_1(a), \mathsf{in}_2(b)) \in (A+B)^2$ is uniquely determined by $(\mathsf{in}_1(a), \mathsf{in}_2(0)) \in (A+1)^2$ and $(\mathsf{in}_1(0), \mathsf{in}_2(b)) \in (1+B)^2$, and similarly for the three other cases of t.

In fact, all signature functors as well as $\mathcal{P}$ and all monoid-valued functors (see Example 2.3(4)) are zippable. Moreover, the class of zippable functors is closed under products, coproducts, and subfunctors but not under composition, e.g. $\mathcal{P}\mathcal{P}$ is not zippable [WDMS20].

The intuitive reason why $\mathcal{P}\mathcal{P}$ is not zippable is that $\mathcal{P}\mathcal{P}$ has two 'unordered levels', and so we can not uniquely reconstruct $t \in \mathcal{P}\mathcal{P}(A+B)$ given only the restrictions to $\mathcal{P}\mathcal{P}(A+1)$

and $\mathcal{PP}(1 + B)$. In the following example, we take distinct $a_1, a_2 \in A$ and $b_1, b_2 \in B$, and omit some of the coproduct injections for the sake of brevity:

$$\mathcal{PP}(A + B) \xrightarrow{\langle \mathcal{PP}(A+!), \mathcal{PP}(!+B) \rangle} \mathcal{PP}(A + 1) \times \mathcal{PP}(1 + B) \xleftarrow{\langle \mathcal{PP}(A+!), \mathcal{PP}(!+B) \rangle} \mathcal{PP}(A + B)$$

$$\{\{a_1, b_1\}, \{a_2, b_2\}\} \longmapsto \begin{pmatrix} \{\{a_1, \mathsf{in}_2(0)\}, \{a_2, \mathsf{in}_2(0)\}\}, \\ \{\{\mathsf{in}_1(0), b_1\}, \{\mathsf{in}_1(0), b_2\}\} \end{pmatrix} \longleftmapsto \{\{a_1, b_2\}, \{a_2, b_1\}\}$$

Both the left-hand and the right-hand set of sets yield the same terms when restricting to $A + 1$ and $1 + B$ separately, showing that the map in Definition 3.5 is not injective for $\mathcal{PP}$. This example extends to a coalgebra for which partition refinement based on characteristic maps χ_S^B would compute wrong results [WDMS20, Ex. 5.11].

Remark 3.7. To apply the algorithm to coalgebras for composites FG of zippable functors, e.g. $\mathcal{P}(A \times (-))$, there is a reduction [WDMS20, Section 8] that embeds every FG-coalgebra into a coalgebra for the zippable functor $(F + G)(X) := FX + GX$. This reduction preserves and reflects behavioural equivalence, but introduces an intermediate state for every transition. The reduction factors through an encoding of composite functors via multisorted coalgebra [SP11], in which, e.g., a composite functor FG would be represented in a setting with two sorts $1, 2$ as a pair of functors $\hat{F}, \hat{G}$, one going from sort 1 to sort 2 and one going the other way around. The multisorted framework comes with a corresponding multisorted coalgebraic modal logic. There are conversion functors between multisorted coalgebras (e.g. for a multisorted functor made up of $\hat{F}$ and $\hat{G}$) and single-sorted coalgebras (e.g. for FG) which, in the end, guarantee compositionality (w.r.t. functor composition, including composition with multi-argument functors such as binary sum) of most semantic and algorithmic properties of coalgebraic modal logics, including the Hennessy-Milner property (see [SP11] for details).

In principle, these results imply in particular that the algorithms and complexity results developed in the present paper are compositional w.r.t. functor composition. Establishing this formally will require transferring the framework of multisorted coalgebraic modal logic along the above-mentioned translation from multisorted coalgebras to single-sorted coalgebras for sums of functors. To keep the paper focused, we refrain from carrying this out in the present paper. We do note that this implies that we do not, at the moment, cover labelled transition systems, i.e. coalgebras for the composite functor $\mathcal{P} \circ (A \times (-))$, in full formality.

Theorem 3.8 [WDMS20, Thm. 4.20, 5.20]**.** *On a finite coalgebra (C, c) for a zippable functor, Algorithm 3.2 terminates after $i \leq |C|$ loop iterations, and the resulting partition identifies precisely the behaviourally equivalent states ($P_i = \sim$).*

In the correctness proof, the zippability is used to show that it is sufficient to incrementally refine the partition using the characteristic map χ_S^B under the functor F in step **(A3)**. When constructing certificates in the following, this refinement turns into a logical conjunction and the characteristic map under the functor turns into a modal operator.

3.3. **Generic Modal Operators.** The extended Paige-Tarjan algorithm (Section 3.1) constructs a distinguishing formula according to the three cases **(C1)**–**(C3)**. In the coalgebraic Algorithm 3.2, these cases correspond to elements of $F3$, which determine in which block an element of a predecessor block T ends up. Indeed, the elements of $F3$ will also serve as

generic modalities in characteristic formulae for blocks of states, essentially by the equivalence between n-ary predicate liftings and (in this case, singleton) subsets of $F(2^n)$ (Lemma 2.8); such singletons are also known as *tests* [Kli05].

Definition 3.9. The signature of $F3$-*modalities* for a functor F is

$$\Lambda = \{\ulcorner t \urcorner/2 \mid t \in F3\};$$

that is, we write $\ulcorner t \urcorner$ for the syntactic representation of a binary modality for every $t \in F3$. The interpretation of $\ulcorner t \urcorner$ for $F3$ is given by

$$[\![\ulcorner t \urcorner]\!] : (2^X)^2 \to 2^{FX}, \qquad [\![\ulcorner t \urcorner]\!](S, B) = \{t' \in FX \mid F\chi^B_{S \cap B}(t') = t\}.$$

Lemma 3.10. *The above maps $[\![\ulcorner t \urcorner]\!] : (2^X)^2 \to 2^{FX}$ form a binary predicate lifting.*

Proof. There is a canonical quotient $q \colon 2^2 \to 3$ given by

$$q(1,1) = 2 \qquad q(0,1) = 1 \qquad q(0,0) = 0 \qquad q(1,0) = 0.$$

The map q satisfies

$$\chi^B_{S \cap B} = \big(X \xrightarrow{\langle \chi_S, \chi_B \rangle} 2 \times 2 \cong 2^2 \xrightarrow{q} 3\big).$$

For a fixed $t \in F3$, define a predicate lifting via the subset

$$(Fq)^{-1}[\{t\}] \quad \subseteq \quad F(2^2)$$

By Lemma 2.8, the corresponding predicate lifting is given by:

$$\begin{aligned}
[\![\ulcorner t \urcorner]\!](S, B) &= \{t' \in FX \mid F\langle \chi_S, \chi_B \rangle(t') \in (Fq)^{-1}[\{t\}]\} \\
&= \{t' \in FX \mid Fq(F\langle \chi_S, \chi_B \rangle(t')) \in \{t\}\} \\
&= \{t' \in FX \mid F(q \cdot \langle \chi_S, \chi_B \rangle)(t') = t\} \\
&= \{t' \in FX \mid F\chi^B_{S \cap B}(t') = t\} \qquad\qquad \square
\end{aligned}$$

The intended use of $\ulcorner t \urcorner$ is as follows: Suppose a block B is split into subblocks $S \subseteq B$ and $B \setminus S$, with certificates δ and β for S and B, respectively; that is, $[\![\delta]\!] = S$ and $[\![\beta]\!] = B$. As in Figure 3, we then split every predecessor block T of B into smaller parts, each of which is uniquely characterized by the formula $\ulcorner t \urcorner(\delta, \beta)$ for some $t \in F3$.

Example 3.11. For $F = \mathcal{P}$, the formula $\ulcorner\{0,2\}\urcorner(\delta, \beta)$ is equivalent to

$$\overbrace{\Diamond \neg \beta}^{\text{`0'}} \wedge \neg \overbrace{\Diamond(\beta \wedge \neg \delta)}^{\text{`1'}} \wedge \overbrace{\Diamond(\delta \wedge \beta)}^{\text{`2'}}.$$

Lemma 3.12. *Given an F-coalgebra (C, c), a state $x \in C$, and formulae δ and β such that $[\![\delta]\!] \subseteq [\![\beta]\!] \subseteq C$, we have*

$$x \in [\![\ulcorner t \urcorner(\delta, \beta)]\!] \quad \Longleftrightarrow \quad F\chi^{[\![\beta]\!]}_{[\![\delta]\!]}(c(x)) = t.$$

Proof. This follows directly from Definition 3.9 applied to $S := [\![\phi_S]\!]$ and $B := [\![\phi_B]\!]$, using that $S \cap B = S$:

$$\begin{aligned}
[\![\ulcorner t \urcorner(\phi_S, \phi_B)]\!] &= c^{-1}[[\![\ulcorner t \urcorner]\!]_C([\![\phi_S]\!], [\![\phi_B]\!])] \\
&= c^{-1}[[\![\ulcorner t \urcorner]\!]_C(S, B)] \\
&= c^{-1}[\{t' \in FC \mid F\chi^B_{S \cap B}(t') = t\}] \\
&= \{x \in C \mid F\chi^B_S(c(x)) = t\}. \qquad\qquad \square
\end{aligned}$$

In the initial partition C/P_0 on a transition system (C, c), we used the formulae $\Diamond\top$ and $\neg\Diamond\top$ to distinguish live states and deadlocks. In general, we can similarly describe the initial partition using modalities induced by elements of $F1$:

Notation 3.13. Define the injective map $j_1\colon 1 \rightarrowtail 3$ by $j_1(0) = 2$. Then the injection $Fj_1\colon F1 \rightarrowtail F3$ provides a way to interpret elements $t \in F1$ as nullary modalities $\ulcorner t\urcorner$:

$$\ulcorner t\urcorner := \ulcorner Fj_1(t)\urcorner(\top, \top) \qquad \text{for } t \in F1.$$

(Alternatively, we could introduce $\ulcorner t\urcorner$ directly as a nullary modality.)

Lemma 3.14. *Given a coalgebra $c\colon C \to FC$, a state $x \in C$, and $t \in F1$, we have*

$$x \in [\![\ulcorner t\urcorner]\!] \iff F!(c(x)) = t$$

Proof. Note that for $\chi_C^C\colon C \to 3$, we have $\chi_C^C = (C \xrightarrow{!} 1 \xrightarrow{j_1} 3)$ where $j_1(0) = 2$.

$$
\begin{aligned}
[\![\ulcorner t\urcorner]\!] &= [\![\ulcorner Fj_1(t)\urcorner(\top, \top)]\!] && \text{(Notation 3.13)} \\
&= \{x \in C \mid F\chi_C^C(c(x)) = Fj_1(t)\} && \text{(Lemma 3.12, } [\![\top]\!] = C) \\
&= \{x \in C \mid Fj_1(F!(c(x))) = Fj_1(t)\} && (\chi_C^C = j_1\cdot!) \\
&= \{x \in C \mid F!(c(x)) = t\} && (Fj_1 \text{ injective})
\end{aligned}
$$

In the last step we use our running assumption that, w.l.o.g., F preserves injective maps (Remark 2.6(2)). $\qquad\square$

3.4. **Algorithmic Construction of Certificates.** The $F3$-modalities introduced above (Definition 3.9) induce an instance of coalgebraic modal logic (Section 2.2). We refer to coalgebraic modal formulae employing the $F3$-modalities as $F3$-*modal formulae*, and write $\mathcal{M}$ for the set of $F3$-modal formulae. As in the extended Paige-Tarjan algorithm (Section 3.1), we annotate every block arising during the execution of Algorithm 3.2 with a certificate in the shape of an $F3$-modal formula. Annotating blocks with formulae means that we construct maps

$$\beta_i\colon C/Q_i \to \mathcal{M} \qquad \text{and} \qquad \delta_i\colon C/P_i \to \mathcal{M} \qquad \text{for } i \in \mathbb{N}.$$

As in Algorithm 3.2, i indexes the loop iterations. For blocks B, S in the respective partition, we denote by $\beta_i(B)$ and $\delta_i(S)$ the corresponding certificates. We shall prove in Theorem 3.16 further below that the following invariants hold, which immediately imply correctness:

$$\forall B \in X/Q_i\colon [\![\beta_i(B)]\!] = B \qquad \text{and} \qquad \forall S \in X/P_i\colon [\![\delta_i(S)]\!] = S, \qquad \text{for every } i. \quad (3.3)$$

We construct $\beta_i(B)$ and $\delta_i(S)$ iteratively, using certificates for the blocks $S \subsetneq B$ at every iteration:

Algorithm 3.15. We extend Algorithm 3.2 as follows. We add initializations

$$\beta_0(\{C\}) := \top \qquad \text{and} \qquad \delta_0([x]_{P_0}) := \ulcorner F!(c(x))\urcorner \quad \text{for every } [x]_{P_0} \in C/P_0.$$

In the i-th iteration, we add the following assignments to steps **(A2)** and **(A3)**, respectively:

$$
\textbf{(A'2)} \quad \beta_{i+1}(D) \quad =
\begin{cases}
\delta_i(S) & \text{if } D = S \\
\beta_i(B) \wedge \neg\delta_i(S) & \text{if } D = B \setminus S \\
\beta_i(D) & \text{if } D \in C/Q_i
\end{cases}
$$

(A'3) $\delta_{i+1}([x]_{P_{i+1}}) = \begin{cases} \delta_i([x]_{P_i}) & \text{if } [x]_{P_{i+1}} = [x]_{P_i} \\ \delta_i([x]_{P_i}) \wedge \ulcorner F\chi_S^B(c(x))\urcorner(\delta_i(S), \beta_i(B)) & \text{otherwise.} \end{cases}$

Upon termination, return δ_i.

Like in Section 3.1, the only block of C/Q_0 has $\beta_0(\{C\}) = \top$ as a certificate. The partition C/P_0 distinguishes by the 'output' $F!(c(x)) \in F1$ (e.g. final vs. non-final states of an automaton), and by Lemma 3.14, the certificate of $[x]_{P_0}$ specifies precisely this output; in particular, $\delta_0([x]_{P_0})$ is well-defined.

In the i-th iteration of the main loop, we have certificates $\delta_i(S)$ and $\beta_i(B)$ for $S \subsetneq B$ in step **(A1)** satisfying (3.3) available from the previous iterations. In **(A'2)**, the Boolean connectives describe how B is split into S and $B\backslash S$. In **(A'3)**, new certificates are constructed for every predecessor block $T \in C/P_i$ that is refined. If T does not change, then neither does its certificate. Otherwise, the block $T = [x]_{P_i}$ is split into the blocks $[x]_{F\chi_S^B(c(x))}$ for $x \in T$ in step **(A3)**, which is reflected by the $F3$ modality $\ulcorner F\chi_S^B(c(x))\urcorner$ as per Lemma 3.12.

Theorem 3.16. *For every zippable functor F, Algorithm 3.15 is correct: the invariants in (3.3) hold. Thus, upon termination δ_i assigns certificates to each block of $C/\sim = C/P_i$.*

Proof. (1) We first observe that given $x \in C$, $S \subseteq B \subseteq C$, and certificates ϕ_S and ϕ_B of S and B, respectively, we have:

$$\begin{aligned} [\![\ulcorner F\chi_S^B(c(x))\urcorner(\phi_S, \phi_B)]\!] &= \{x' \in C \mid F\chi_{[\![\phi_S]\!]}^{[\![\phi_B]\!]}(c(x')) = F\chi_S^B(c(x))\} \\ &= [x]_{F\chi_S^B(c(x))}, \end{aligned} \tag{3.4}$$

where the first equation uses Lemma 3.12 and the second one holds because $[\![\phi_B]\!] = B$ and $[\![\phi_S]\!] = S$.

(2) We verify (3.3) by induction on i.

- In the base case $i = 0$, we have $[\![\beta_0(\{C\})]\!] = [\![\top]\!] = C$ for the only block in X/Q_0. Since $P_0 = \ker(F! \cdot c)$, δ_0 is well-defined, and by Lemma 3.14 we have

$$[\![\delta_0([x]_{P_0})]\!] = [\![\ulcorner F!(c(x))\urcorner]\!] = \{y \in C \mid F!(c(x)) = F!(c(y))\} = [x]_{P_0}.$$

- The inductive hypothesis states that

$$[\![\delta_i(S)]\!] = S \qquad \text{and} \qquad [\![\beta_i(B)]\!] = B. \tag{IH}$$

We prove that β_{i+1} is correct:

$$[\![\beta_{i+1}([x]_{Q_{i+1}})]\!]$$

$$= \begin{cases} [\![\delta_i(S)]\!] & \text{if } [x]_{Q_{i+1}} = S, \text{ hence } S = [x]_{P_i} \\ [\![\beta_i(B)]\!] \cap C \setminus [\![\delta_i(S)]\!] & \text{if } [x]_{Q_{i+1}} = B \setminus S, \text{ hence } B = [x]_{Q_i} \\ [\![\beta_i([x]_{Q_i})]\!] & \text{if } [x]_{Q_{i+1}} \in C/Q_i \end{cases}$$

$$\overset{\text{(IH)}}{=} \begin{cases} S & \text{if } [x]_{Q_{i+1}} = S \\ B \cap C \setminus S & \text{if } [x]_{Q_{i+1}} = B \setminus S \\ [x]_{Q_i} & \text{if } [x]_{Q_{i+1}} \in C/Q_i \end{cases}$$

$$= \begin{cases} [x]_{Q_{i+1}} & \text{if } [x]_{Q_{i+1}} = S = [x]_{P_i} \\ [x]_{Q_{i+1}} & \text{if } [x]_{Q_{i+1}} = B \setminus S \qquad \text{(since } B \cap C \setminus S = B \setminus S) \\ [x]_{Q_{i+1}} & \text{if } [x]_{Q_{i+1}} \in C/Q_i \qquad \text{(since } [x]_{Q_i} \text{ is not split)} \end{cases}$$

$$= [x]_{Q_{i+1}}.$$

For δ_{i+1}, we compute as follows:

$$\llbracket \delta_{i+1}([x]_{P_{i+1}}) \rrbracket$$

$$= \begin{cases} \llbracket \delta_i([x]_{P_i}) \rrbracket & \text{if } [x]_{P_{i+1}} = [x]_{P_i} \\ \llbracket \delta_i([x]_{P_i}) \rrbracket \cap \llbracket \ulcorner F\chi_S^B(c(x)) \urcorner (\delta_i(S), \beta_i(B)) \rrbracket & \text{otherwise} \end{cases}$$

$$\overset{\text{(IH) \& (3.4)}}{=} \begin{cases} [x]_{P_i} & \text{if } [x]_{P_{i+1}} = [x]_{P_i} \\ [x]_{P_i} \cap [x]_{F\chi_S^B(c(x))} & \text{otherwise} \end{cases}$$

$$\overset{\text{def. } P_{i+1}}{=} \begin{cases} [x]_{P_{i+1}} & \text{if } [x]_{P_{i+1}} = [x]_{P_i} \\ [x]_{P_{i+1}} & \text{otherwise} \end{cases}$$

$$= [x]_{P_{i+1}} \qquad\qquad \square$$

The assumption of zippability is used in the correctness of the underlying partition refinement algorithm. But for the construction of certificates, this assumption translates into the ability to describe certificates as only a conjunction of $F3$-modalities in **(A'3)**. As a consequence, we obtain a Hennessy-Milner-type property of our $F3$-modal formulae:

Corollary 3.17. *For zippable F, states x, y in a finite F-coalgebra are behaviourally equivalent iff they agree on all $F3$-modal formulae.*

Construction 3.18. Given a coalgebra $c: C \to FC$ and states $x, y \in C$, a smaller formula distinguishing a state x from a state y can be extracted from the certificates of the behavioural equivalence classes of x and y in time $\mathcal{O}(|C|)$: It is the leftmost conjunct that is different in the respective certificates of x and y. In other words, this is the subformula starting at the modal operator introduced in δ_i for the least i with $(x, y) \notin P_i$; hence, x satisfies $\ulcorner t \urcorner (\delta, \beta)$ but y satisfies $\ulcorner t' \urcorner (\delta, \beta)$ for some $t' \neq t$ in $F3$.

Hence, when only distinguishing formula for states x, y is of interest, it is sufficient to run the algorithm until x is split from y, and the distinguishing formula is conjunct added in step **(A'3)**. This leads to an earlier termination in practice but does not change the run time complexity in $\mathcal{O}$-notation.

Proposition 3.19. *Construction 3.18 correctly extracts a formula distinguishing x from y.*

Proof. In order to verify that the first differing conjunct is a distinguishing formula, we distinguish cases on the least i such that $(x, y) \notin P_i$:

If x and y are already split by P_0, then the conjunct at index 0 in the respective certificates of $[x]_\sim$ and $[y]_\sim$ differs, and we have $t = F!(c(x))$ and $t' = F!(c(y))$. By Lemma 3.14, $\ulcorner t \urcorner$ distinguishes x from y (and $\ulcorner t' \urcorner$ distinguishes y from x).

If x and y are split by P_{i+1} (but $(x, y) \in P_i$), then

$$\underbrace{F\chi_S^B(c(x))}_{t \, :=} \neq \underbrace{F\chi_S^B(c(y))}_{t' \, :=}.$$

Thus, the conjuncts that differ in the respective certificates for $[x]_\sim$ and $[y]_\sim$ are the following ones at index $i + 1$:

$$\ulcorner t \urcorner (\delta_i(S), \beta_i(B)) \qquad \text{and} \qquad \ulcorner t' \urcorner (\delta_i(S), \beta_i(B)).$$

By Lemma 3.12, $\ulcorner t \urcorner (\delta_i(S), \beta_i(B))$ distinguishes x from y (and $\ulcorner t \urcorner (\delta_i(S), \beta_i(B))$ distinguishes y from x). $\qquad \square$

3.5. **Complexity Analysis.** The operations introduced by Algorithm 3.15 can be implemented with only constant run time overhead. To this end, one implements β and δ as arrays of formulae of length $|C|$ (note that at any point, there are at most $|C|$-many blocks). In the refinable-partition data structure [VL08], every block has an index (a natural number) and there is an array of length $|C|$ mapping every state $x \in C$ to the block it is contained in. Hence, for both partitions C/P and C/Q, one can look up a state's block and a block's certificate in constant time.

It is very likely that the certificates contain a particular subformula multiple times and that certificates of different blocks share common subformulae. For example, every certificate of a block refined in the i-th iteration using $S \subsetneq B$ contains the subformulae $\delta_i(S)$ and $\beta_i(B)$. Therefore, it is advantageous to represent all certificates constructed as one directed acyclic graph (dag) with inner nodes labelled by either a modal operator or conjunction and having precisely two outgoing edges, and leaf nodes labelled by either $\top$ or a nullary modal operator. Moreover, edges have a binary flag indicating whether they represent negation $\neg$. Initially, there is only one (leaf) node representing $\top$, and the operations of Algorithm 3.15 allocate new nodes and update the arrays for β and δ to point to the right nodes. For example, if the predecessor block $T \in C/P_i$ is refined in step (**A'3**), yielding a new block $[x]_{P_{i+1}}$, then a new node labelled $\wedge$ is allocated with edges to the nodes $\delta_i(T)$ and to another new node labelled $F\chi_S^B(c(x))$ with edges to the nodes $\delta_i(S)$ and $\delta_i(B)$.

For purposes of estimating the size of formulae generated by the algorithm, we use a notion of *transition* in coalgebras, inspired by the notion of canonical graph [Gum05].

Definition 3.20. For states x, y in an F-coalgebra (C, c), we say that there is a *transition* $x \to y$ if $c(x) \in FC$ is not in the image $Fi[F(C \setminus \{y\})]$ $(\subseteq FC)$, where $i \colon C \setminus \{y\} \rightarrowtail C$ is the inclusion map.

Theorem 3.21. *For a coalgebra with n states and m transitions, the formula dag constructed by Algorithm 3.15 has at most*

$$2 \cdot m \cdot (\log_2 n + 1) + 2 \cdot n$$

nodes, each with outdegree ≤ 2, and a height of at most $n + 1$. Hence, the dag size is in $\mathcal{O}(m \cdot \log_2 n + n)$.

Before proving Theorem 3.21, we need to establish a sequence of lemmas on the underlying partition refinement algorithm. Recall from Remark 2.6(2) that we may assume that F preserves finite intersections by working with its Trnková hull instead.

Let (C, c) be a coalgebra for F. We define a binary relation $\to$ on $\mathcal{P}(C)$ by

$$T \to S \qquad \Longleftrightarrow \qquad \exists x \in T, y \in S \colon x \to y$$

for $T, S \subseteq C$. In other words, we write $T \to S$ if there is a transition from some state of T to some state of S. Also we define the set $\mathsf{pred}(S)$ of predecessor states of a set S as

$$\mathsf{pred}(S) = \big\{x \in C \mid \{x\} \to S\big\}.$$

Lemma 3.22. *For every F-coalgebra (C, c), $x \in C$, and $S \subseteq B \subseteq C$ with S finite, we have*

$$\{x\} \not\to S \qquad \Longrightarrow \qquad F\chi_S^B(c(x)) = F\chi_\emptyset^B(c(x)).$$

Proof. For every $y \in S$, we have that $x \not\rightarrow y$. Hence, for every $y \in S$, there exists $t_y \in F(C \setminus \{y\})$ such that

$$c(x) = Fi(t_y) \qquad \text{for } i \colon C \setminus \{y\} \rightarrowtail C.$$

The set $C \setminus S$ is the intersection of all sets $C \setminus \{y\}$ with $y \in S$:

$$C \setminus S = \bigcap_{y \in S} (C \setminus \{y\}).$$

Since F preserves finite intersections and S is finite, we have that

$$F(C \setminus S) = \bigcap_{y \in S} F(C \setminus \{y\}).$$

Since $c(x) \in FC$ is contained in every $F(C \setminus \{y\})$ (as witnessed by t_y) it is also contained in their intersection. That is, for $m \colon C \setminus S \rightarrowtail C$ being the inclusion map, there is $t' \in F(C \setminus S)$ such that $Fm(t') = c(x)$. Now consider the following diagrams:

$$
\begin{array}{ccc}
c(x) & \in & FC \xrightarrow{\quad F\chi_S^B \quad} F3 \\
\uparrow & & \uparrow{\scriptstyle Fm} \quad \nearrow{\scriptstyle F\chi_\emptyset^B} \\
t' & \in & F(C \setminus S)
\end{array}
\qquad \text{and} \qquad
\begin{array}{c}
FC \xrightarrow{\quad F\chi_\emptyset^B \quad} F3 \\
\uparrow{\scriptstyle Fm} \quad \nearrow{\scriptstyle F\chi_\emptyset^B} \\
F(C \setminus S)
\end{array}
$$

Both triangles commute because $\chi_\emptyset^B = \chi_S^B \cdot m$ and $\chi_\emptyset^B = \chi_\emptyset^B \cdot m$. Thus, we conclude

$$F\chi_S^B(c(x)) = F\chi_S^B(Fm(t')) = F\chi_\emptyset^B(t') = F\chi_\emptyset^B(Fm(t')) = F\chi_\emptyset^B(c(x)). \qquad \square$$

The classical Paige-Tarjan algorithm maintains the invariant that the partition P_i is *stable* [PT87], meaning that for all states $(x, x') \in P_i$ in the same block, either both x, x' or none of x, x' have a transition to a given block in C/Q_i. In terms of characteristic functions $\chi_\emptyset^B$ and for general F, this can be rephrased as follows.

Lemma 3.23. *For all $(x, x') \in P_i$ and $B \in C/Q_i$ in Algorithm 3.2, we have*

$$F\chi_\emptyset^B(c(x)) = F\chi_\emptyset^B(c(x')).$$

Proof. One can show [WDMS20, Prop. 4.12] that in every iteration i we have a map $c_i \colon C/P_i \to F(C/Q_i)$ that satisfies $F[-]_{Q_i} \cdot c = c_i \cdot [-]_{P_i}$:

$$
\begin{array}{ccc}
C & \xrightarrow{\quad c \quad} & FC \\
{\scriptstyle [-]_{P_i}} \downarrow & & \downarrow {\scriptstyle F[-]_{Q_i}} \\
C/P_i & \xrightarrow{\quad c_i \quad} & F(C/Q_i)
\end{array}
$$

where the maps $[-]_{P_i}, [-]_{Q_i}$ send elements of C to their equivalence class (Notation 2.1). The map $\chi_\emptyset^B \colon C \to 3$ for $B \in C/Q_i$ can be decomposed as follows:

$$
\begin{array}{c}
C \xrightarrow{\quad \chi_\emptyset^B \quad} 3 \\
{\scriptstyle [-]_{Q_i}} \downarrow \quad \nearrow {\scriptstyle \chi_\emptyset^{\{B\}}} \\
C/Q_i
\end{array}
$$

Combining these two diagrams, we obtain

$$F\chi_\emptyset^B \cdot c = F\chi_\emptyset^{\{B\}} \cdot F[-]_{Q_i} \cdot c = F\chi_\emptyset^{\{B\}} \cdot c_i \cdot [-]_{P_i}. \tag{3.5}$$

Since for all $(x, x') \in P_i$, we have $[x]_{P_i} = [x']_{P_i}$, we conclude that

$$F\chi_\emptyset^B(c(x)) \overset{(3.5)}{=} F\chi_\emptyset^{\{B\}}(c_i([x]_{P_i})) = F\chi_\emptyset^{\{B\}}(c_i([x']_{P_i})) \overset{(3.5)}{=} F\chi_\emptyset^B(c(x')). \qquad \square$$

Combining the previous two lemmas, we obtain that in the refinement step for $S \subsetneq B$, only predecessor blocks of S need to be adjusted in **(A3)**, so that only the formulae for these blocks need to be updated:

Lemma 3.24. *For $S \subsetneq B \in C/Q_i$ in the ith iteration of Algorithm 3.2, a block $T \in C/P_i$ with no edge to S is not modified; in symbols:*

$$T \not\to S \quad \implies \quad T \in C/P_{i+1} \qquad\qquad \text{for all } T \in C/P_i.$$

Proof. Since $T \not\to S$, we have $\{x\} \not\to S$ and $\{x'\} \not\to S$ for all $x, x' \in T$. Thus,

$$
\begin{aligned}
F\chi_S^B(c(x)) &= F\chi_\emptyset^B(c(x)) & &(\text{Lemma 3.22}, \{x\} \not\to S) \\
&= F\chi_\emptyset^B(c(x')) & &(\text{Lemma 3.23}, (x, x') \in P_i) \\
&= F\chi_S^B(c(x')) & &(\text{Lemma 3.22}, \{x'\} \not\to S). \qquad \square
\end{aligned}
$$

The above lemma shows that the number of blocks T that are split is bounded by the number of predecessor blocks of the state set S. Since most of the resulting smaller blocks T' in the new partition C/P_{i+1} must have an edge to S, the number of these new blocks must also be bounded essentially in the predecessors of S, as we show next:

Lemma 3.25. *For $S \subseteq C$ and finite C in the ith iteration of Algorithm 3.2,*

$$|\{T' \in C/P_{i+1} \mid T' \notin C/P_i\}| \ \leq \ 2 \cdot |\mathsf{pred}(S)|.$$

Proof. Let $S \subsetneq B \in C/Q_i$ be used for splitting in iteration i. If $T' \in C/P_{i+1}$ and $T' \notin C/P_i$, then the block $T \in C/P_i$ with $T' \subseteq T$ satisfies $T \notin C/P_{i+1}$ and therefore, by Lemma 3.24, has a transition to S. By finiteness of C, T is split into finitely many blocks $T_1, \ldots, T_k \in C/P_{i+1}$, representing the equivalence classes of the kernel of $F\chi_S^B \cdot c \colon C \to F3$. By Lemma 3.22 we know that if $x \in T$ has no transition to S, then $F\chi_S^B(c(x)) = F\chi_\emptyset^B(c(x))$. Moreover, all elements of $T \in C/P_i$ are sent to the same value by $F\chi_\emptyset^B \cdot c$ (Lemma 3.23). Hence, there is at most one block T_j with no transition to S, and all other blocks $T_{j'}$, $j' \neq j$, have transitions to S. Therefore, the number k of blocks T_j is at most $|T \cap \mathsf{pred}(S)| + 1$. Summing over all predecessor blocks T of S, we obtain

$$
\begin{aligned}
&|\{T' \in C/P_{i+1} \mid T' \notin C/P_i\}| \\
&\leq |\{T' \in C/P_{i+1} \mid T' \subseteq T \in C/P_i \text{ and } T \to S\}| & &(\text{Lemma 3.24}) \\
&= \sum_{\substack{T \in C/P_i \\ T \to S}} |\{T' \in C/P_{i+1} \mid T' \subseteq T\}| \\
&\leq \sum_{\substack{T \in C/P_i \\ T \to S}} (|T \cap \mathsf{pred}(S)| + 1) & &(\text{bound on } k \text{ above}) \\
&\leq 2 \cdot \sum_{\substack{T \in C/P_i \\ T \to S}} |T \cap \mathsf{pred}(S)| & &(|T \cap \mathsf{pred}(S)| \geq 1) \\
&\leq 2 \cdot |\mathsf{pred}(S)| & &(T \in C/P_i \text{ are disjoint}) \qquad \square
\end{aligned}
$$

We can now show a bound for the total number of blocks that exist at some point during the execution of the partition refinement algorithm:

Lemma 3.26. *Given an input coalgebra (C, c) with $n = |C|$ states and m transitions, the following holds throughout the execution of Algorithm 3.2:*

$$|\{T \subseteq C \mid T \in C/P_i \text{ for some } i\}| \leq 2 \cdot m \cdot \log_2 n + 2 \cdot m + n.$$

Note that the proof is similar to arguments given in the complexity analysis of the Paige-Tarjan algorithm (cf. [PT87, p. 980]).

Proof. Since $|S| \leq \frac{1}{2} \cdot |B|$ holds in step **(A1)** of Algorithm 3.2, one can show that every state $x \in C$ is contained in the set S picked in step **(A1)** in at most $\log_2(n) + 1$ iterations [WDMS20, Lem. 7.15]. More formally, let $S_i \subsetneq B_i \in C/Q_i$ be the blocks picked in the ith iteration of Algorithm 3.2. Then we have

$$|\{S_i \mid x \in S_i\}| \leq \log_2 n + 1 \qquad \text{for all } x \in C. \tag{3.6}$$

Let the algorithm terminate after ℓ iterations, returning C/P_ℓ. Then, the number of new blocks introduced by step **(A3)** is bounded as follows:

$$\sum_{0 \leq i < \ell} |\{T' \in C/P_{i+1} \mid T' \notin C/P_i\}|$$

$$\leq \sum_{0 \leq i < \ell} 2 \cdot |\mathsf{pred}(S_i)| \qquad\qquad \text{(Lemma 3.25)}$$

$$\leq 2 \cdot \sum_{0 \leq i < \ell} \sum_{x \in S_i} |\mathsf{pred}(\{x\})|$$

$$= 2 \cdot \sum_{x \in C} \sum_{0 \leq i < \ell \mid x \in S_i} |\mathsf{pred}(\{x\})|$$

$$= 2 \cdot \sum_{x \in C} |\mathsf{pred}(\{x\})| \cdot \sum_{0 \leq i < \ell \mid x \in S_i} 1$$

$$\leq 2 \cdot \sum_{x \in C} |\mathsf{pred}(\{x\})| \cdot (\log_2 n + 1) \qquad\qquad \text{(by (3.6))}$$

$$= 2 \cdot m \cdot (\log_2 n + 1) = 2 \cdot m \cdot \log_2 n + 2 \cdot m$$

The only blocks we have not counted so far are the blocks of C/P_0. Since $|C/P_0| \leq n$, we have at most $2 \cdot m \cdot \log_2 n + 2 \cdot m + n$ different blocks in $(C/P_i)_{0 \leq i < \ell}$. $\quad\square$

Every block in the final or intermediate partitions C/P_i corresponds to a certificate that is constructed at some point. Hence, the bound on the blocks is also a bound for the dag size of formulae created by Algorithm 3.15.

Proof of Theorem 3.21. Regarding the height of the dag, it is immediate that δ_i and β_i have height at most $i + 1$. Since $|C/Q_i| < |C/Q_{i+1}| \leq |C| = n$ for all i, there are at most n iterations, with the final partition being $C/P_{n+1} = C/Q_{n+1}$.

In Algorithm 3.15 we create a new modal operator in the dag whenever Algorithm 3.2 creates a new block in C/P_i (either by initialization or in step **(A'3)**). By Lemma 3.26, the number of modalities in the dag is thus bounded by

$$2 \cdot m \cdot \log_2 n + 2 \cdot m + n.$$

In every iteration of the main loop, β is extended by two new formulae, one for S and one for $B \setminus S$. The formula $\beta_{i+1}(S)$ does not increase the size of the dag, because no new node needs to be allocated. For $\beta_{i+1}(B \setminus S)$, we need to allocate one new node for the conjunction, so there are at most n new such nodes allocated throughout the execution of the whole algorithm. Thus, the total number of nodes in the dag is bounded by

$$2 \cdot m \cdot \log_2 n + 2 \cdot m + 2 \cdot n$$

and each such node has an outdegree of at most 2 by construction. □

Theorem 3.27. *Algorithm 3.15 adds only constant run time overhead per step of Algorithm 3.2, and thus has the same asymptotic run time as Algorithm 3.2.*

Like for the run time analysis of the underlying Algorithm 3.2, we assume that the memory model supports random access, i.e. array access runs in constant time.

Proof. The arrays for β and δ are re-used in every iteration. Hence, the index i can be neglected; it is only used to refer to a value before or after the loop iteration. The analysis then proceeds as follows:

(1) Initialization step:

- The only block $\{C\}$ in C/Q_0 has index 0, and so we make $\beta(0)$ point to the node $\top$, which takes constant time.
- For every block T in C/P_0, Algorithm 3.2 has computed $F!(c(x)) \in F1$ for some (in fact every) $x \in T$. Since $F1$ canonically embeds into $F3$ (Notation 3.13), we create a new node labelled $\ulcorner Fj_1(F!(c(x)))\urcorner$ with two edges to $\top$. For every $T \in C/P_0$, this runs in constant time, which we allocate to the step where Algorithm 3.2 creates T.

(2) In the refinement step, we can look up the certificates $\delta_i(S)$ and $\beta_i(B)$ in constant time using the indices of the blocks S and B. Whenever the original algorithm creates a new block, we also immediately construct the certificate of this new block by creating at most two new nodes in the dag (with at most four outgoing edges). However, if a block does not change (that is, $[x]_{Q_i} = [x]_{Q_{i+1}}$ or $[x]_{P_i} = [x]_{P_{i+1}}$, resp.), then the corresponding certificate is not changed in step **(A'2)** or step **(A'3)**, respectively.

In the loop body we update the certificates as follows:

(A'2) The new block $S \in C/Q_{i+1}$ just points to the certificate $\delta_i(S)$ constructed earlier. For the new block $(B \setminus S) \in C/Q_{i+1}$, we allocate a new node $\wedge$, with one edge to $\beta_i(B)$ and one negated edge to $\delta_i(S)$.

(A'3) Not all resulting blocks have a transition to S. There may be (at most) one new block $T' \in C/P_{i+1}$, $T' \subseteq T$ with no transition to S (see the proof of Lemma 3.25). In the refinable partition structure, such a block will inherit the index from T (i.e. the index of T in C/P_i equals the index of T' in C/P_{i+1}). Moreover, every $x \in T'$ satisfies $F\chi_S^B(c(x)) = F\chi_\emptyset^B(c(x))$ (by Lemma 3.22), and $F\chi_\emptyset^B(c(x)) = F\chi_\emptyset^B(c(y))$ for every $y \in T$ (by Lemma 3.23).

Now, one first saves the node of the certificate $\delta_i(T)$ in some variable δ', say. Then the array δ is updated at index T by the formula

$$\ulcorner F\chi_\emptyset^B(c(y))\urcorner(\delta_i(S), \beta_i(B)) \qquad \text{for an arbitrary } y \in T.$$

Consequently, a block T' inheriting the index of T automatically has the correct certificate.

The allocation of nodes for this formula is completely analogous to the one for an ordinary block $[x]_{P_{i+1}} \subsetneqq T$ having edges to S: One allocates a new node labelled $\wedge$ with

edges to the saved node δ' (the original value of $\delta_i(T)$) and to another newly allocated node labelled $\ulcorner F\chi_S^B(c(x))\urcorner$ with edges to the nodes $\delta_i(S)$ and $\delta_i(B)$. $\qquad\square$

In order to keep the formula size smaller, one can implement the following optimization. Intuitively, note that for $S \in X/P_i$ and $B \in X/Q_i$ such that $S \subseteq B \subseteq C$, every conjunct of $\beta_i(B)$ is also a conjunct of $\delta_i(S)$. In $\beta_i(B) \wedge \neg\delta_i(S)$, one can hence remove all conjuncts of $\beta_i(B)$ from $\delta_i(S)$, obtaining a formula δ', and then equivalently use $\beta_i(B) \wedge \neg\delta'$ in the definition of $\beta_{i+1}(D)$.

Proposition 3.28. *In step **(A'2)**, $\beta_{i+1}(D)$ can be simplified to be no larger than $\delta_i(S)$ without increasing the overall run time.*

Proof. Mark every modal operator node $\ulcorner t\urcorner(\delta, \beta)$ in the formula dag with a boolean flag expressing whether

$$\ulcorner t\urcorner(\delta, \beta) \text{ is a conjunct of some } \beta_i\text{-formula.}$$

Thus, every new modal operator in **(A'3)** is marked 'false' initially. When the block B in C/Q_i is split into S and $B \setminus S$ in step **(A'2)**, the formula for block $B \setminus S$ is a conjunction of $\beta_i(B)$ and the negation of all conjuncts of $\delta_i(S)$ marked 'false'. Afterwards these conjuncts are all marked 'true', because they are inherited by $\beta_i(S)$. The conjuncts marked 'false' always form a prefix of all conjuncts of a formula in δ_i. It therefore suffices to greedily take conjuncts from the root of a formula dag while they are marked 'false'.

As a consequence, step **(A'3)** no longer runs in constant time but instead takes as many steps as there are conjuncts marked 'false' in $\delta_i(S)$. However, over the whole execution of the algorithm this eventually amortizes because every newly allocated modal operator is initially marked 'false' and later marked 'true' precisely once. Thus, in the analysis conducted in the proof of Theorem 3.27, the additional run time can be neglected asymptotically. $\qquad\square$

For a tighter run time analysis of the underlying partition refinement algorithm, one additionally requires that F is equipped with a *refinement interface* [WDMS20, Def. 6.4], which is based on a given encoding of F-coalgebras in terms of *edges* between states (encodings serve only as data structures and have no direct semantic meaning, in particular do not entail a semantic reduction to relational structures). This notion of edge yields the same numbers (in $\mathcal{O}$-notation) as Definition 3.20 for all functors considered. All zippable functors we consider here have refinement interfaces [WDMS20, WDMS21]. In presence of a refinement interface, step **(A3)** can be implemented efficiently, with resulting overall run time $\mathcal{O}((m + n) \cdot \log n \cdot p(c))$ where $n = |C|$, m is the number of edges in the encoding of the input coalgebra (C, c), and the *run-time factor* $p(c)$ is associated with the refinement interface. In most instances, e.g. for $\mathcal{P}$, $\mathbb{R}^{(-)}$, one has $p(c) = 1$; in particular, the generic algorithm has the same run time as the Paige-Tarjan algorithm. Usually, it is less of a challenge to find some refinement interface for a functor F but more to find one with low run time, i.e. low $p(c)$. For example, for general monoid-valued functors $FX = M^{(X)}$ the refinement interface uses binary search trees as additional data structures resulting in an additional logarithmic factor $p(c) = \log m$ [WDMS21].

Remark 3.29. The claimed run time relies on close attention to a number of implementation details. This includes use of an efficient data structure for the partition C/P_i [Knu01, VL08]; the other partition C/Q_i is only represented implicitly in terms of a queue of blocks $S \subsetneq B$ witnessing $P_i \subsetneq Q_i$, requiring additional care when splitting blocks in the queue [VF10,

Fig. 3]. Moreover, grouping the elements of a block by $F3$ involves the consideration of a *possible majority candidate* [VF10].

Theorem 3.30. *For a zippable set functor with a refinement interface with factor $p(c)$ and an input coalgebra with n states and m transitions, Algorithm 3.15 runs in time*

$$\mathcal{O}((m + n) \cdot \log n \cdot p(c)).$$

Indeed, the time bound holds for the underlying Algorithm 3.2, and is inherited by Algorithm 3.15 due to Theorem 3.27.

If the functor F satisfies additional assumptions, we can simplify the certificates even further, as discussed next.

4. Cancellative Functors

Our use of binary modalities relates to the fact that, as observed already by Paige and Tarjan, when splitting a block according to an existing partition of a block B into $S \subseteq B$ and $B \setminus S$, it is not in general sufficient to look only at the successors in S. However, this does suffice for some transition types. E.g. Hopcroft's algorithm for deterministic automata [Hop71] and Valmari and Franceschinis' algorithm for weighted systems (e.g. Markov chains) [VF10] both split only with respect to S. In the following, we exhibit a criterion on the level of functors that captures that splitting w.r.t. only S is sufficient:

Definition 4.1. A functor F is *cancellative* if the map

$$\langle F\chi_{\{1,2\}}, F\chi_{\{2\}}\rangle \colon F3 \to F2 \times F2$$

is injective.

To understand the role of the above map, recall the function $\chi_S^B \colon C \to 3$ from (3.2) and note that

$$\chi_{\{1,2\}} \cdot \chi_S^B = \chi_B \qquad \text{and} \qquad \chi_{\{2\}} \cdot \chi_S^B = \chi_S, \tag{4.1}$$

so the composite $\langle F\chi_{\{1,2\}}, F\chi_{\{2\}}\rangle \cdot F\chi_S^B$ yields information about the accumulated transition weights into B and S but not about the one into $B \setminus S$. The injectivity condition means that for cancellative functors, this information suffices in the splitting step for $S \subseteq B \subseteq C$. The term *cancellative* stems from the respective property on monoids; recall that a monoid M is *cancellative* if $s + b_1 = s + b_2$ implies $b_1 = b_2$ for all $s, b_1, b_2 \in M$.

4.1. **Properties of Cancellative Functors.** Before presenting the optimized algorithm, we gather properties of cancellative functors and compare them to zippability and related notions, starting with the property that gave cancellative functors their name.

Proposition 4.2. *The monoid-valued functor $M^{(-)}$ for a commutative monoid M is cancellative if and only if M is a cancellative monoid.*

Proof. First note that for $FX = M^{(X)}$, the maps $F\chi_{\{1,2\}}$, $F\chi_{\{2\}}$ used in Definition 4.1 are given by

$$M^{(\chi_{\{1,2\}})} \colon M^{(3)} \to M^{(2)}, \qquad t \mapsto (t(0), t(1) + t(2)),$$

$$M^{(\chi_{\{2\}})} \colon M^{(3)} \to M^{(2)}, \qquad t \mapsto (t(0) + t(1), t(2)),$$

where we write $s \in M^{(2)}$ as the pair $(s(0), s(1))$.

The functor $M^{(-)}\colon \mathsf{Set} \to \mathsf{Set} \ldots$	$\Leftrightarrow$	The monoid $M \ldots$	
is cancellative	$\Leftrightarrow$	is cancellative	(Proposition 4.2)
preserves inverse images	$\Leftrightarrow$	is positive	[Sch01, 4.74] & [GS01]
preserves weak kernel pairs	$\Leftrightarrow$	is refinable	[Sch01, 4.74] & [GS01]
preserves weak pullbacks	$\Leftrightarrow$	is positive and refinable	[Sch01, 4.35]

TABLE 1. Correspondence between properties of the functor and the monoid

For "$\Leftarrow$", let $s, t \in M^{(3)}$ such that

$$\langle M^{(\chi_{\{1,2\}})}, M^{(\chi_{\{2\}})}\rangle(s) = \langle M^{(\chi_{\{1,2\}})}, M^{(\chi_{\{2\}})}\rangle(t),$$

which is written point-wise as follows:

$$(s(0), s(1) + s(2)) = (t(0), t(1) + t(2))$$
$$(s(0) + s(1), s(2)) = (t(0) + t(1), t(2)).$$

We thus have $s(0) = t(0)$, $s(2) = t(2)$, and

$$s(1) + s(2) = t(1) + t(2) = t(1) + s(2).$$

Since M is cancellative, it follows that $s(1) = t(1)$, so $s = t$. Thus, the map $\langle M^{(\chi_{\{1,2\}})},$ $M^{(\chi_{\{2\}})}\rangle$ is injective.

For "$\Rightarrow$", let $a, b, c \in M$ such that $c + a = c + b$. Define $s, t \in M^{(3)}$ by

$$s(0) = s(2) = c, \quad s(1) = a \qquad \text{and} \qquad t(0) = t(2) = c, \quad t(1) = b.$$

Thus,

$$
\begin{aligned}
M^{(\chi_{\{1,2\}})}(s) &= (s(0), s(1) + s(2)) & M^{(\chi_{\{2\}})}(s) &= (s(0) + s(1), s(2)) \\
&= (c, a + c) & &= (c + a, c) \\
&= (c, b + c) & &= (c + b, c) \\
&= (t(0), t(1) + t(2)) & &= (t(0) + t(1), t(2)) \\
&= M^{(\chi_{\{1,2\}})}(t), & &= M^{(\chi_{\{2\}})}(t).
\end{aligned}
$$

Since $\langle M^{(\chi_{\{1,2\}})}, M^{(\chi_{\{2\}})}\rangle$ is injective, it follows that $s = t$. Thus, we have $a = s(1) = t(1) = b$, so M is cancellative. $\square$

The property of cancellativity nicely extends a list of correspondences between properties of the monoid-valued functor $M^{(-)}\colon \mathsf{Set} \to \mathsf{Set}$ on the one hand and the underlying commutative monoid M on the other hand, see Table 1 and work by Gumm and Schröder [GS01, Sch01] for more details.

Example 4.3. The functor $\mathbb{R}^{(-)}$ is cancellative, but $\mathcal{P}_f$, being naturally isomorphic to $M^{(-)}$ for the (non-cancellative) Boolean monoid $M = 2$, is not.

All signature functors are cancellative:

Proposition 4.4. *The class of cancellative functors contains the identity functor and all constant functors, and is closed under subfunctors, products, and coproducts.*

Proof. (1) The identity functor is cancellative because the map $\langle \chi_{\{1,2\}}, \chi_{\{2\}} \rangle$ is clearly injective.

(2) For the constant functor C_X with value X, $C_X(\chi_S)$ is the identity map on X for every set S. Therefore C_X is cancellative.

(3) Let $\alpha \colon F \rightarrowtail G$ be a natural transformation with injective components and let G be cancellative. Combining the naturality squares of α for $\chi_{\{1,2\}}$ and $\chi_{\{2\}}$, we obtain the commutative square

$$
\begin{array}{ccc}
F3 & \xrightarrow{\langle F\chi_{\{1,2\}}, F\chi_{\{2\}} \rangle} & F2 \times F2 \\
\alpha_3 \Big\downarrow & & \Big\downarrow \alpha_2 \times \alpha_2 \\
G3 & \xrightarrowtail[\langle G\chi_{\{1,2\}}, G\chi_{\{2\}} \rangle]{} & G2 \times G2,
\end{array}
$$

in which the composite $F3 \to G2 \times G2$ is injective by hypothesis. Hence, $\langle F\chi_{\{1,2\}}, F\chi_{\{2\}} \rangle$ is injective as well, showing that the subfunctor F is cancellative.

(4) Let $(F_i)_{i \in I}$ be a family of cancellative functors, and suppose that we have elements $s, t \in (\prod_{i \in I} F_i)(3) = \prod_{i \in I} F_i 3$ with

$$
\Big(\prod_{i \in I} F_i \chi_{\{1,2\}}\Big)(s) = \Big(\prod_{i \in I} F_i \chi_{\{1,2\}}\Big)(t) \quad \text{and} \quad \Big(\prod_{i \in I} F_i \chi_{\{2\}}\Big)(s) = \Big(\prod_{i \in I} F_i \chi_{\{2\}}\Big)(t).
$$

Write pr_i for the ith projection function from the product. For every $i \in I$ we have:

$$
F_i \chi_{\{1,2\}}(\mathsf{pr}_i(s)) = F_i \chi_{\{1,2\}}(\mathsf{pr}_i(t)) \qquad \text{and} \qquad F_i \chi_{\{2\}}(\mathsf{pr}_i(s)) = F_i \chi_{\{2\}}(\mathsf{pr}_i(t)).
$$

Since every F_i is cancellative, we have $\mathsf{pr}_i(s) = \mathsf{pr}_i(t)$ for every $i \in I$. This implies $s = t$ since the product projections $(\mathsf{pr}_i)_{i \in I}$ are jointly injective.

(5) Again, let $(F_i)_{i \in I}$ be a family of cancellative functors. Suppose that we have elements $s, t \in (\coprod_{i \in I} F_i)(3) = \coprod_{i \in I} F_i 3$ satisfying

$$
\Big(\coprod_{i \in I} F_i \chi_{\{1,2\}}\Big)(s) = \Big(\coprod_{i \in I} F_i \chi_{\{1,2\}}\Big)(t) \quad \text{and} \quad \Big(\coprod_{i \in I} F_i \chi_{\{2\}}\Big)(s) = \Big(\coprod_{i \in I} F_i \chi_{\{2\}}\Big)(t).
$$

This implies that there exists an $i \in I$ and $s', t' \in F_i 3$ with $s = \mathsf{in}_i(s')$, $t = \mathsf{in}_i(t')$, and

$$
F_i \chi_{\{1,2\}}(s') = F_i \chi_{\{1,2\}}(t') \qquad \text{and} \qquad F_i \chi_{\{2\}}(s') = F_i \chi_{\{2\}}(t').
$$

Since F_i is cancellative, we have $s' = t'$, which implies $s = t$. $\qquad\qquad\square$

A consequence of closure under subfunctors is that, for example, $\mathcal{D}$ is cancellative, being a subfunctor of $\mathbb{R}^{(-)}$, but $\mathcal{P}$ is not, as we have already seen that its subfunctor $\mathcal{P}_\mathsf{f}$ fails to be cancellative.

Proposition 4.5. *Cancellative functors are neither closed under quotients nor under composition. Zippability and cancellativity are independent properties.*

Proof. Table 2 shows an overview of all counterexamples used in the present proof.

(1) Cancellative functors are not closed under quotients: e.g. the non-cancellative functor $\mathcal{P}_\mathsf{f}$ is a quotient of the signature functor $X \mapsto \coprod_{n \in \mathbb{N}} X^n$ (which is cancellative by Proposition 4.4).

Operation	cancellative	non-cancellative		cancellative	non-cancel.
Quotient	$X \mapsto \coprod_{n\in\mathbb{N}} X^n$	$\mathcal{P}_{\mathsf{f}}$	zippable	$X \mapsto X$	$\mathcal{P}_{\mathsf{f}}$
Composition	$\mathcal{B} = \mathbb{N}^{(-)}$	$\mathcal{B}\mathcal{B}$	non-zippable	see (4.2)	$\mathcal{P}_{\mathsf{f}}\mathcal{P}_{\mathsf{f}}$

(A) Operations   (B) Independence of zippability

Table 2. Counter examples regarding cancellative functors.

(2) Cancellative functors are not closed under composition. For the additive monoid $(\mathbb{N}, +, 0)$ of natural numbers, the monoid-valued functor $\mathcal{B} = \mathbb{N}^{(-)}$ sends X to the set of finite multisets on X ('bags'). Since $\mathbb{N}$ is cancellative, $\mathcal{B}$ is a cancellative functor. However, $\mathcal{B}\mathcal{B}$ is not (below we write $\{\!\!\{\cdots\}\!\!\}$ to denote multisets, so $\{\!\!\{0,1\}\!\!\} = \{\!\!\{1,0\}\!\!\}$ but $\{\!\!\{1\}\!\!\} \neq \{\!\!\{1,1\}\!\!\}$):

$$\langle \mathcal{B}\mathcal{B}\chi_{\{1,2\}}, \mathcal{B}\mathcal{B}\chi_{\{2\}} \rangle \big(\{\!\!\{ \{\!\!\{0,1\}\!\!\}, \{\!\!\{1,2\}\!\!\} \}\!\!\} \big)$$
$$= \big(\{\!\!\{ \{\!\!\{0,1\}\!\!\}, \{\!\!\{1,1\}\!\!\} \}\!\!\}, \{\!\!\{ \{\!\!\{0,0\}\!\!\}, \{\!\!\{0,1\}\!\!\} \}\!\!\} \big)$$
$$= \big(\{\!\!\{ \{\!\!\{0,1\}\!\!\}, \{\!\!\{1,1\}\!\!\} \}\!\!\}, \{\!\!\{ \{\!\!\{0,1\}\!\!\}, \{\!\!\{0,0\}\!\!\} \}\!\!\} \big)$$
$$= \langle \mathcal{B}\mathcal{B}\chi_{\{1,2\}}, \mathcal{B}\mathcal{B}\chi_{\{2\}} \rangle \big(\{\!\!\{ \{\!\!\{0,2\}\!\!\}, \{\!\!\{1,1\}\!\!\} \}\!\!\} \big).$$

Thus, the map $\langle \mathcal{B}\mathcal{B}\chi_{\{1,2\}}, \mathcal{B}\mathcal{B}\chi_{\{2\}} \rangle$ is not injective.

(3) The identity functor $X \mapsto X$ is both zippable [WDMS20] and cancellative (Proposition 4.4).

(4) The monoid-valued functor $\mathcal{P}_{\mathsf{f}} = \mathbb{B}^{(-)}$ is zippable [WDMS20], but not cancellative (Proposition 4.2), because $\mathbb{B}$ is a non-cancellative monoid.

(5) The functor $\mathcal{P}\mathcal{P}$ is neither zippable [WDMS20, Ex. 5.10] nor cancellative because

$$\langle \mathcal{P}\mathcal{P}\chi_{\{1,2\}}, \mathcal{P}\mathcal{P}\chi_{\{2\}} \rangle(\{\{0\}, \{2\}\}) = (\{\{0\}, \{1\}\}, \{\{0\}, \{1\}\})$$
$$= \langle \mathcal{P}\mathcal{P}\chi_{\{1,2\}}, \mathcal{P}\mathcal{P}\chi_{\{2\}} \rangle(\{\{0\}, \{1\}, \{2\}\}).$$

(6) Every functor F satisfying $|F(2+2)| > 1$ and $|F3| = 1$ is cancellative but not zippable:

- Indeed, every map with domain 1 is injective, in particular the map

$$\langle F\chi_{\{1,2\}}, F\chi_{\{2\}} \rangle \colon 1 \cong F3 \longrightarrow F2 \times F2,$$

  whence F is cancellative.

- If $|F(2+2)| > 1$ and $|F3| = 1$ we have that the map

$$\langle 2+!, !+2 \rangle \colon \underbrace{F(2+2)}_{|-|>1} \to \underbrace{F(2+1)}_{\cong F3 \cong 1} \times \underbrace{F(1+2)}_{\cong F3 \cong 1} \cong 1$$

  is not injective, whence F is not zippable.

A concrete example of such a functor is given by

$$FX = \{S \subseteq X : |S| = 0 \text{ or } |S| = 4\} \tag{4.2}$$

which sends a map $f \colon X \to Y$ to the map $Ff \colon FX \to FY$ defined by

$$Ff(S) = \begin{cases} f[S] & \text{if } |f[S]| = 4 \\ \emptyset & \text{otherwise.} \end{cases} \qquad\qquad \square$$

In related work, König et al. [KMMS20] construct distinguishing formulae in coalgebraic generality. Their assumption is a generalized version of zippability, where the binary coproduct is replaced with an m-ary coproduct for $m \in \mathbb{N}$:

Definition 4.6 [KMMS20]**.** A functor F is m-zippable if the canonical map

$$\mathsf{unzip}_m \colon F(A_1 + A_2 + \ldots + A_m) \longrightarrow F(A_1 + 1) \times F(A_2 + 1) \times \cdots \times F(A_m + 1)$$

is injective. Explicitly, unzip_m is given by

$$\langle F[\Delta_{i,j}]_{j\in\bar{m}}\rangle_{i\in\bar{m}} \colon F\big(\textstyle\coprod_{j=1}^m A_j\big) \longrightarrow \textstyle\prod_{i=1}^m F(A_i + 1)$$

where $\bar{m}$ is the set $\bar{m} = \{1, \ldots, m\}$ and the map $\Delta_{i,j}$ is defined by

$$\Delta_{i,j} \colon A_j \to A_i + 1 \qquad \Delta_{i,j} := \begin{cases} A_j \xrightarrow{\mathsf{in_1}} A_i + 1 & \text{if } i = j \\ A_j \xrightarrow{!} 1 \xrightarrow{\mathsf{in_2}} A_i + 1 & \text{if } i \neq j. \end{cases}$$

Ordinary zippability is then equivalent to 2-zippability.

Proposition 4.7. *Every zippable and cancellative set functor is m-zippable for every m.*

Proof. First, we show that for a zippable and cancellative set functor F, the map

$$g_{A,B} := F(A + 1 + B) \xrightarrow{\langle F(A+!), F(!+B)\rangle} F(A + 1) \times F(1 + B)$$

is injective for all sets A, B. Indeed, we have the following chain of injective maps, where the index at the 1 is only notation to distinguish coproduct components more easily:

$$F(A + (1_M + B))$$
$$\quad \big\downarrow \langle F(A+!), F(! + (1_M + B))\rangle \qquad\qquad\qquad\qquad (F \text{ is zippable})$$
$$F(A + 1) \times F(1_A + 1_M + B)$$
$$\quad \big\downarrow \mathsf{id} \times \langle F(! + B), F(1_A + 1_M +!)\rangle \qquad\qquad\qquad (F \text{ is zippable})$$
$$F(A + 1) \times F(1 + B) \times F(1_A + 1_M + 1_B)$$
$$\quad \big\downarrow \mathsf{id} \times \mathsf{id} \times \langle F\chi_{1_M+1_B}, F\chi_{1_B}\rangle \qquad (F \text{ is cancellative, } 1_A + 1_M + 1_B \cong \{0, 1, 2\})$$
$$F(A + 1) \times F(1 + B) \times F2 \times F2$$

Call this composite f. It factorizes through $g_{A,B}$, because it matches with $g_{A,B}$ on the components $F(A + 1)$ and $F(1 + B)$, and for the other components, one has the map

$$h := F(A + 1) \times F(1 + B) \xrightarrow{F\chi_1 \times F\chi_B} F2 \times F2$$

with $f = \langle \mathsf{id}_{F(A+1)\times F(1+B)}, h\rangle \cdot g_{A,B}$. Since f is injective, $g_{A,B}$ must be injective, too.

Also note that a rewriting Definition 4.1 along the isomorphisms $1 + 1 + 1 \cong 3$ and $1 + 1 \cong 2$, we obtain that a functor F is cancellative iff the map

$$\langle F(1+!), F(! + 1)\rangle \colon F(1 + 1 + 1) \longrightarrow F(1 + 1) \times F(1 + 1)$$

(where $! \colon 1 + 1 \to 1$) is injective.

We now proceed with the proof of the desired implication by induction on m. In the base cases $m = 0$ and $m = 1$, there is nothing to show because every functor is 0- and 1-zippable, and for $m = 2$, the implication is trivial (zippability coincides with 2-zippability by definition). In the inductive step, given that F is 2-zippable, m-zippable ($m \geq 2$), and cancellative, we show that F is $(m + 1)$-zippable.

We have the following chain of injective maps, where we again annotate some of the singleton sets 1 with indices to indicate from which coproduct components they come:

$$F(A_1 + \ldots + A_{m-1} + (A_m + A_{m+1}))$$

$$\downarrow \mathsf{unzip}_m \qquad\qquad\qquad\qquad (F \text{ is } m\text{-zippable})$$

$$\big(\textstyle\prod_{i=1}^{m-1} F(A_i + 1) \big) \times F(A_m + A_{m+1} + 1_{1..(m-1)})$$

$$\cong \big(\textstyle\prod_{i=1}^{m-1} F(A_i + 1) \big) \times F(A_m + 1_{1..(m-1)} + A_{m+1})$$

$$\downarrow \mathsf{id} \times g_{A_m, A_{m+1}} \qquad\qquad (\text{the above injective helper map } g)$$

$$\big(\textstyle\prod_{i=1}^{m-1} F(A_i + 1) \big) \times F(A_m + 1) \times F(1 + A_{m+1})$$

$$\cong \big(\textstyle\prod_{i=1}^{m-1} F(A_i + 1) \big) \times F(A_m + 1) \times F(A_{m+1} + 1)$$

This composite thus is injective as well, and coincides with unzip_{m+1}, showing that F is $(m+1)$-zippable. $\qquad\square$

The converse, however, does not hold because the finite powerset functor $\mathcal{P}_\mathsf{f}$ is m-zippable for all m [KMMS20, Ex. 10, Lem. 14], but not cancellative as we have seen.

4.2. **Optimized Partition Refinement and Certificates.** The optimization present in the algorithms for Markov chains [VF10] and automata [Hop71] can now be adapted to coalgebras for cancellative functors, where it suffices to split only according to transitions into S, ignoring transitions into $B \setminus S$. More formally, this means that we replace the three-valued $\chi_S^B \colon C \to 3$ with $\chi_S \colon C \to 2$ in the refinement step **(A3)**:

Proposition 4.8. *Let F be a cancellative set functor. For $S \in C/P_i$ in the i-th iteration of Algorithm 3.2, we have $P_{i+1} = P_i \cap \ker(C \xrightarrow{c} FC \xrightarrow{F\chi_S} F2)$.*

Proof. From the definition (2.1) of the kernel, we immediately obtain the following properties for all maps $f, g \colon Y \to Z$, $h \colon X \to Y$:

$$f \text{ injective} \implies \ker(f \cdot h) = \ker(h) \tag{4.3}$$

$$\ker(f) = \ker(g) \implies \ker(f \cdot h) = \ker(g \cdot h) \tag{4.4}$$

$$\ker(\langle f, g \rangle) = \ker(f) \cap \ker(g). \tag{4.5}$$

For every coalgebra $c \colon C \to FC$ and $S \subseteq B \subseteq C$ we have by (4.1) that

$$\langle F\chi_B, F\chi_S \rangle = \langle F\chi_{\{1,2\}}, F\chi_{\{2\}} \rangle \cdot F\chi_S^B.$$

Since F is cancellative, $\langle F\chi_{\{1,2\}}, F\chi_{\{2\}} \rangle$ is injective, and we thus obtain

$$\ker(\langle F\chi_B, F\chi_S \rangle) = \ker(\langle F\chi_{\{1,2\}}, F\chi_{\{2\}} \rangle \cdot F\chi_S^B) \overset{(4.3)}{=} \ker(F\chi_S^B). \tag{4.6}$$

By (4.4), this implies that

$$\ker(\langle F\chi_B, F\chi_S \rangle \cdot c) = \ker(F\chi_S^B \cdot c). \tag{4.7}$$

Let $B \in C/Q_i$ be the block that is split into S and $B \setminus S$ in iteration i. Since P_i is finer than Q_i and $B \in C/Q_i$, we have $P_i \subseteq Q_i \subseteq \ker(F\chi_B \cdot c)$; thus:

$$P_i = P_i \cap \ker(C \xrightarrow{c} FC \xrightarrow{F\chi_B} F2). \tag{4.8}$$

Now we verify the desired property:

$$
\begin{aligned}
P_{i+1} &= P_i \cap \ker(C \xrightarrow{c} FC \xrightarrow{F\chi_S^B} F2) & \text{(by \textbf{(A3)})} \\
&= P_i \cap \ker(\langle F\chi_B, F\chi_S \rangle \cdot c) & \text{(by (4.7))} \\
&= P_i \cap \ker(\langle F\chi_B \cdot c, F\chi_S \cdot c \rangle) & \\
&= P_i \cap \ker(F\chi_B \cdot c) \cap \ker(F\chi_S \cdot c) & \text{(by (4.5))} \\
&= P_i \cap \ker(F\chi_S \cdot c) & \text{(by (4.8))} \qquad \Box
\end{aligned}
$$

Note that this result is independent of certificate construction and already improves the underlying partition refinement algorithm.

Example 4.9. Suppose that F is a a signature functor Σ or a monoid-valued functor $M^{(-)}$ for a cancellative monoid M. Given an input coalgebra for F, the refinement step **(A3)** of Algorithm 3.2 can be optimized to compute P_{i+1} according to Proposition 4.8.

Observe that, in the optimized step **(A3)**, B is no longer mentioned. It is therefore unsurprising that we do not need a certificate for it when constructing certificates for the blocks of P_{i+1}. Instead, we can reflect the map $F\chi_S \cdot c \colon C \to F2$ in the coalgebraic modal formula and take (unary) modal operators from $F2$. Just like $F1$ in Notation 3.13, the set $F2$ canonically embeds into $F3$.

Notation 4.10. Define the injective map $j_2 \colon 2 \rightarrowtail 3$ by $j_2(0) = 1$ and $j_2(1) = 2$. The injection $Fj_2 \colon F2 \rightarrowtail F3$ provides a way to interpret elements $t \in F2$ as unary modalities $\ulcorner t \urcorner$:

$$
\ulcorner t \urcorner(\delta) := \ulcorner Fj_2(t) \urcorner(\delta, \top).
$$

Remark 4.11. There are several different ways to define $\ulcorner t \urcorner(\delta)$ for $t \in F2$, depending on the definition of the inclusion j_2.

$j_2 \colon 2 \rightarrowtail 3$	$j_2 \cdot \chi_S$ for $S \subseteq C$	Definition for $t \in F2$
$0 \mapsto 0, 1 \mapsto 1$	$j_2 \cdot \chi_S = \chi_\emptyset^S$	$\ulcorner t \urcorner(\delta) := \ulcorner Fj_2(t) \urcorner(\bot, \delta)$
$0 \mapsto 0, 1 \mapsto 2$	$j_2 \cdot \chi_S = \chi_S^S$	$\ulcorner t \urcorner(\delta) := \ulcorner Fj_2(t) \urcorner(\delta, \delta)$
$0 \mapsto 1, 1 \mapsto 2$	$j_2 \cdot \chi_S = \chi_S^C$	$\ulcorner t \urcorner(\delta) := \ulcorner Fj_2(t) \urcorner(\delta, \top)$

All these variants make the following Lemma 4.12 true because in each case, the data j_2, ϕ, ψ used in the definition of $\ulcorner t \urcorner(\delta)$ as $\ulcorner Fj_2(t) \urcorner(\phi, \psi)$ satisfy

$$
j_2 \cdot \chi_{\llbracket \delta \rrbracket} = \chi_{\llbracket \phi \rrbracket}^{\llbracket \psi \rrbracket}.
$$

In analogy to Lemma 3.12, we can show:

Lemma 4.12. *Given a cancellative functor F, an F-coalgebra (C, c), $t \in F2$, a formula δ, and a state $x \in C$, we have*

$$
x \in \llbracket \ulcorner t \urcorner(\delta) \rrbracket \iff F\chi_{\llbracket \delta \rrbracket}(c(x)) = t.
$$

Proof. By the definition of j_2, we have $j_2 \cdot \chi_S = \chi_S^C$ for all $S \subseteq C$. Thus,

$$
\begin{aligned}
\llbracket \ulcorner t \urcorner(\delta) \rrbracket &= \llbracket \ulcorner Fj_2(t) \urcorner(\delta, \top) \rrbracket & \text{(Notation 4.10)} \\
&= \{x \in C \mid F\chi_{\llbracket \delta \rrbracket}^C(c(x)) = Fj_2(t)\} & \text{(Lemma 3.12, } \llbracket \top \rrbracket = C) \\
&= \{x \in C \mid Fj_2(F\chi_{\llbracket \delta \rrbracket}(c(x))) = Fj_2(t)\} & (\chi_{\llbracket \delta \rrbracket}^C = j_2 \cdot \chi_{\llbracket \delta \rrbracket})
\end{aligned}
$$

$$= \{x \in C \mid F\chi_{\llbracket \delta \rrbracket}(c(x)) = t\} \qquad\qquad (Fj_2 \text{ injective})$$

In the last step, we use that F preserves injective maps (Remark 2.6(2)). $\qquad\square$

In Algorithm 3.15, the family β is only used in the definition of δ_{i+1} to characterize the larger block B that has been split into the smaller blocks $S \subseteq B$ and $B \setminus S$. For a cancellative functor, we can replace

$$\ulcorner F\chi_S^B(c(x))\urcorner(\delta_i(S), \beta_i(B)) \qquad \text{with} \qquad \ulcorner F\chi_S(c(x))\urcorner(\delta_i(S))$$

in the definition of δ_{i+1}. Hence, we can omit β_i from Algorithm 3.15 altogether, obtaining the following algorithm, which is again based on coalgebraic partition refinement (Algorithm 3.2).

Algorithm 4.13. We extend Algorithm 3.2 as follows. Initially, define

$$\delta_0([x]_{P_0}) = \ulcorner F!(c(x))\urcorner.$$

In the i-th iteration, extend step **(A3)** by the additional assignment

(A'3) $\delta_{i+1}([x]_{P_{i+1}}) = \begin{cases} \delta_i([x]_{P_i}) & \text{if } [x]_{P_{i+1}} = [x]_{P_i} \\ \delta_i([x]_{P_i}) \wedge \ulcorner F\chi_S(c(x))\urcorner(\delta_i(S)) & \text{otherwise.} \end{cases}$

Theorem 4.14. *For cancellative functors, Algorithm 4.13 is correct; that is, we have:*

$$\forall S \in X/P_i \colon \llbracket \delta_i(S) \rrbracket = S \qquad \text{for all } i \in \mathbb{N}.$$

Remark 4.15. Note that the optimized Algorithm 4.13 can also be implemented treating the unary modal operators in $F2$ as first class citizens, in lieu of embedding them into $F3$ as we did in Notation 4.10. The only difference between the two implementation approaches w.r.t. the size of the formula dag is one edge per modality, namely the edge to the node $\top$ from the node $\ulcorner Fj_2(F\chi_S(c(x)))\urcorner(\delta_i(\delta_i), \top)$ that arises when step **(A'3)** is expanded according to Notation 4.10.

Proof of Theorem 4.14. Induction over i, the index of loop iterations.

The definition of δ_0 is identical to the definition in Algorithm 3.15, whence

$$\llbracket \delta_0(S) \rrbracket = S \qquad \text{for all } S \in C/P_0,$$

proved completely analogously as in the proof of Theorem 3.16.

In the i-th iteration with chosen block $S \in C/P_i$, we distinguish cases on whether a block $[x]_{P_{i+1}} \in C/P_{i+1}$ remains the same or is split into other blocks:

- If $[x]_{P_{i+1}} = [x]_{P_i}$, then we have

$$\llbracket \delta_{i+1}([x]_{P_i}) \rrbracket \overset{\textbf{(A'3)}}{=} \llbracket \delta_i([x]_{P_i}) \rrbracket \overset{\text{I.H.}}{=} [x]_{P_i} = [x]_{P_{i+1}}.$$

- If $[x]_{P_{i+1}} \neq [x]_{P_i}$, we compute as follows:

$$
\begin{aligned}
\llbracket \delta_{i+1}([x]_{P_{i+1}}) \rrbracket &= \llbracket \delta_i([x]_{P_i}) \wedge \ulcorner F\chi_S(c(x))\urcorner(\delta_i(S)) \rrbracket && \textbf{(A'3)} \\
&= \llbracket \delta_i([x]_{P_i}) \rrbracket \cap \llbracket \ulcorner F\chi_S(c(x))\urcorner(\delta_i(S)) \rrbracket \\
&= [x]_{P_i} \cap \llbracket \ulcorner F\chi_S(c(x))\urcorner(\delta_i(S)) \rrbracket && \text{(I.H.)} \\
&= [x]_{P_i} \cap \{x' \in C \mid F\chi_{\llbracket \delta_i(S) \rrbracket}(c(x')) = F\chi_S(c(x))\} && \text{(Lemma 4.12)} \\
&= [x]_{P_i} \cap \{x' \in C \mid F\chi_S(c(x')) = F\chi_S(c(x))\} && \text{(I.H.)} \\
&= [x]_{P_i} \cap \{x' \in C \mid (x, x') \in \ker(F\chi_S \cdot c)\} && \text{(def. ker)} \\
&= [x]_{P_i} \cap [x]_{F\chi_S \cdot c} && \text{(def. } [x]_R)
\end{aligned}
$$

$$= [x]_{P_{i+1}}.$$

The last step is follows from $P_{i+1} = P_i \cap \ker(F\chi_S \cdot c)$ (see Proposition 4.8). □

The formulae resulting from the optimized construction involve only $\wedge$, $\top$, and modalities from the set $F2$ (or $F3$ with the second parameter fixed to $\top$), which we term $F2$-*modalities*. Thus, Hennessy-Milner Theorem (Corollary 3.17) can be sharpened for cancellative functors as follows.

Corollary 4.16. *For a zippable and cancellative set functor F, states in a finite F-coalgebra are behaviourally equivalent iff they agree on modal formulae built using $\top$, $\wedge$, and unary $F2$-modalities.*

The certificates thus computed are reduced to roughly half the size compared to Algorithm 3.15; the asymptotic run time and formula size (Section 3.5) remain unchanged.

5. Domain-Specific Certificates

On a given specific system type, one is typically interested in certificates and distinguishing formulae expressed via modalities whose use is established in the respective domain, e.g. $\square$ and $\Diamond$ for transition systems. We next describe how the generic $F3$ modalities can be rewritten to domain-specific ones in a postprocessing step. The domain-specific modalities will not always be equivalent to $F3$-modalities, but still yield certificates.

Definition 5.1. The *Boolean closure* $\bar{\Lambda}$ of a modal signature Λ has as n-ary modalities propositional combinations of atoms of the form $\heartsuit(i_1, \ldots, i_k)$, for $\heartsuit/k \in \Lambda$, where $i_1, \ldots, i_k$ are propositional combinations of elements of $\{1, \ldots, n\}$. Such a modality λ/n is interpreted by predicate liftings $[\![\lambda]\!]_X \colon (2^X)^n \to FX$ defined inductively in the obvious way.

For example, the Boolean closure of $\Lambda = \{\Diamond/1\}$ contains the unary modality $\square = \neg\Diamond\neg1$.

Definition 5.2. Given a modal signature Λ for a functor F, a *domain-specific interpretation* consists of functions $\tau \colon F1 \to \bar{\Lambda}$ and $\lambda \colon F3 \to \bar{\Lambda}$ assigning to each $o \in F1$ a nullary modality τ_o and to each $t \in F3$ a binary modality λ_t such that the predicate liftings $[\![\tau_o]\!]_X \in 2^{FX}$ and $[\![\lambda_t]\!]_X \colon (2^X)^2 \to 2^{FX}$ satisfy

$$[\![\tau_o]\!]_1 = \{o\} \quad (\text{in } 2^{F1}) \quad \text{and} \quad [t]_{F\chi_{\{1,2\}}} \cap [\![\lambda_t]\!]_3(\{2\},\{1\}) = \{t\} \quad (\text{in } 2^{F3}).$$

(Recall that $\chi_{\{1,2\}} \colon 3 \to 2$ is the characteristic function of $\{1,2\} \subseteq 3$, and $[t]_{F\chi_{\{1,2\}}} \subseteq F3$ denotes the equivalence class of t w.r.t. $F\chi_{\{1,2\}} \colon F3 \to F2$.)

Thus, τ_o holds precisely at states with output behaviour $o \in F1$. Intuitively, $\lambda_t(\delta, \rho)$ describes the refinement step of a predecessor block T when splitting $B := [\![\delta]\!] \cup [\![\rho]\!]$ into $S := [\![\delta]\!]$ and $B \setminus S := [\![\rho]\!]$ (Figure 3), which translates into the arguments $\{2\}$ and $\{1\}$ of $[\![\lambda_t]\!]_3$. In the refinement step, we know from previous iterations that all elements have the same behaviour w.r.t. B. This is reflected in the intersection with $[t]_{F\chi_{\{1,2\}}}$. The given condition on λ_t thus guarantees that λ_t characterizes $t \in F3$ uniquely, but only within the equivalence class representing a predecessor block. Thus, λ_t can be much smaller than equivalents of $\ulcorner t \urcorner$ (cf. Example 3.11):

Example 5.3. We provide examples for set functors of interest; the verification that these are indeed domain-specific interpretations follows in Lemma 5.6 further below.

(1) For $F = \mathcal{P}$, we have a domain-specific interpretation over the modal signature $\Lambda = \{\Diamond/1\}$. For $\emptyset, \{0\} \in \mathcal{P}1$, take $\tau_\emptyset = \neg\Diamond\top$ and $\tau_{\{0\}} = \Diamond\top$. For $t \in \mathcal{P}3$, we put

$$\begin{array}{ll}
\lambda_t(\delta, \rho) = \neg\Diamond\rho & \text{if } 2 \in t \not\ni 1 \\
\lambda_t(\delta, \rho) = \neg\Diamond\delta & \text{if } 2 \notin t \ni 1
\end{array} \qquad \begin{array}{ll}
\lambda_t(\delta, \rho) = \Diamond\delta \wedge \Diamond\rho & \text{if } 2 \in t \ni 1 \\
\lambda_t(\delta, \rho) = \top & \text{if } 2 \notin t \not\ni 1.
\end{array}$$

The certificates obtained via this translation are precisely the ones generated in the example using the Paige-Tarjan algorithm, cf. (3.1), with ρ in lieu of $\beta \wedge \neg\delta$.

(2) For a signature (functor) Σ, take $\Lambda = \{\sigma/0 \mid \sigma/n \in \Sigma\} \cup \{\langle =I\rangle/1 \mid I \in \mathcal{P}_f(\mathbb{N})\}$. We interpret Λ by the predicate liftings

$$[\![\sigma]\!]_X = \{\sigma(x_1, \ldots, x_n) \mid x_1, \ldots, x_n \in X\} \subseteq \Sigma X,$$

$$[\![\langle =I\rangle]\!](S) = \{\sigma(x_1, \ldots, x_n) \in \Sigma X \mid \forall i \in \mathbb{N}: i \in I \leftrightarrow (1 \leq i \leq n \ \wedge \ x_i \in S)\}.$$

Intuitively, σ states that the next operation symbol is σ, and $\langle =I\rangle \phi$ states that the ith successor satisfies ϕ iff $i \in I$. We then have a domain-specific interpretation (τ, λ) given by

$$\begin{array}{ll}
\tau_o = \sigma & \text{for } o = \sigma(0, \ldots, 0) \in \Sigma 1, \text{ and} \\
\lambda_t(\delta, \rho) = \langle =I\rangle \delta & \text{for } t = \sigma(x_1, \ldots, x_n) \in \Sigma 3 \text{ and } I = \{i \in \{1, \ldots, n\} \mid x_i = 2\}.
\end{array}$$

(3) For a monoid-valued functor $M^{(-)}$, take $\Lambda = \{\langle =m\rangle/1 \mid m \in M\}$, interpreted by the predicate liftings $[\![\langle =m\rangle]\!]_X: 2^X \to 2^{M^{(X)}}$ given by

$$[\![\langle =m\rangle]\!]_X(S) = \{\mu \in M^{(X)} \mid m = \textstyle\sum_{x \in S} \mu(x)\}.$$

A formula $\langle =m\rangle \delta$ thus states that the accumulated weight of the successors satisfying δ is exactly m. A domain-specific interpretation (τ, λ) is then given by

$$\begin{array}{ll}
\tau_o = \langle =o(0)\rangle\top & \text{for } o \in M^{(1)}, \text{ and} \\
\lambda_t(\delta, \rho) = \langle =t(2)\rangle \delta \wedge \langle =t(1)\rangle \rho & \text{for } t \in M^{(3)}.
\end{array}$$

In case M is cancellative, we can also simply put $\lambda_t(\delta, \rho) = \langle =t(2)\rangle \delta$.

(4) For labelled Markov chains, i.e. $FX = (\mathcal{D}X + 1)^A$, let $\Lambda = \{\langle a\rangle_p/1 \mid a \in A, p \in [0, 1]\}$, where $\langle a\rangle_p\phi$ denotes that on input a, the next state will satisfy ϕ with probability at least p, as in cited work by Desharnais et al. [DEP02]. This gives rise to the interpretation:

$$\tau_o = \bigwedge_{\substack{a \in A \\ o(a) \in \mathcal{D}1}} \langle a\rangle_1 \top \wedge \bigwedge_{\substack{a \in A \\ o(a) \in 1}} \neg\langle a\rangle_1 \top, \qquad \lambda_t(\delta, \rho) = \bigwedge_{\substack{a \in A \\ t(a) \in \mathcal{D}3}} (\langle a\rangle_{t(a)(2)} \delta \wedge \langle a\rangle_{t(a)(1)} \rho).$$

To ease the verification of the requisite properties of interpretations, we will now show that for cancellative F, domain-specific interpretations can be derived from a simpler kind of interpretation, one where the set $F3$ in Example 5.3 is replaced with $F2$.

Definition 5.4. Given a modal signature Λ for a functor F, a *simple domain-specific interpretation* consists of functions $\tau: F1 \to \bar{\Lambda}$ and $\kappa: F2 \to \bar{\Lambda}$ assigning a nullary modality τ_o to each $o \in F1$ and a unary modality κ_s to each $s \in F2$ such that the predicate liftings $[\![\tau_o]\!]_X \in 2^{FX}$ and $[\![\kappa_s]\!]: 2^X \to 2^{FX}$ satisfy

$$[\![\tau_o]\!]_1 = \{o\} \quad (\text{in } 2^{F1}) \qquad \text{and} \qquad [s]_{F!} \cap [\![\kappa_s]\!]_2(\{1\}) = \{s\} \qquad (\text{in } 2^{F2}).$$

Proposition 5.5. *Let Λ be a modal signature for a cancellative functor F, and (τ, κ) a simple domain-specific interpretation. Define $\lambda: F3 \to \bar{\Lambda}$ by $\lambda_t(\delta, \rho) = \kappa_{F\chi_{\{2\}}(t)}(\delta)$. Then (τ, λ) is a domain-specific interpretation.*

Proof. Given $t \in F3$, we put $s = F\chi_{\{2\}}(t) \in F2$. We have to show that

$$[t]_s \cap [\![\lambda_t]\!]_3(\{2\}, \{1\}) = \{t\} \qquad \text{in } 2^{F3}.$$

By the naturality of the predicate lifting $[\![\kappa_s]\!]$, the following square commutes (recall that $2^{(-)}$ is contravariant):

$$
\begin{array}{ccc}
2^2 & \xrightarrow{\;[\![\kappa_s]\!]_2\;} & 2^{F2} \\
{\scriptstyle 2^{\chi_{\{2\}}}}\big\downarrow & & \big\downarrow{\scriptstyle 2^{F\chi_{\{2\}}}} \\
2^3 & \xrightarrow{\;[\![\kappa_s]\!]_3\;} & 2^{F3}
\end{array}
\tag{5.1}
$$

We thus have

$$
\begin{aligned}
[\![\lambda_t]\!]_3(\{2\}, \{1\}) &= [\![\kappa_s]\!]_3(\{2\}) && (\text{def. } \lambda_t) \\
&= [\![\kappa_s]\!]_3(\chi_{\{2\}}^{-1}[\{1\}]) && (\text{def. } \chi_{\{2\}}) \\
&= [\![\kappa_s]\!]_3(2^{\chi_{\{2\}}}(\{1\})) && (\text{def. } 2^{(-)}) \\
&= 2^{F\chi_{\{2\}}}([\![\kappa_s]\!]_2(\{1\})) && (\text{by } (5.1)) \\
&= \{t' \in F3 \mid F\chi_{\{2\}}(t') \in [\![\kappa_s]\!]_2(\{1\})\} && (\text{def. } 2^{(-)}).
\end{aligned}
$$

The square

$$
\begin{array}{ccc}
3 & \xrightarrow{\;\chi_{\{1,2\}}\;} & 2 \\
{\scriptstyle \chi_{\{2\}}}\big\downarrow & & \big\downarrow{\scriptstyle !} \\
2 & \xrightarrow{\;\;!\;\;} & 1
\end{array}
$$

trivially commutes. Hence, given $t' \in [t]_{F\chi_{\{1,2\}}}$, that is, $F\chi_{\{1,2\}}(t') = F\chi_{\{1,2\}}(t)$, postcomposing this equation with $F!$ and using the above commutative square under F we see that

$$F! \cdot F\chi_{\{2\}}(t') = F! \cdot F\chi_{\{2\}}(t),$$

which yields

$$F\chi_{\{2\}}(t') \in [F\chi_{\{2\}}(t)]_{F!}. \tag{5.2}$$

Using this, we have for every $t' \in F3$ the following chain of equivalences

$$
\begin{aligned}
&t' \in [t]_{F\chi_{\{1,2\}}} \cap [\![\lambda_t]\!]_3(\{2\}, \{1\}) \\
\Leftrightarrow\; &t' \in [t]_{F\chi_{\{1,2\}}} \text{ and } t' \in [\![\lambda_t]\!]_3(\{2\}, \{1\}) \\
\Leftrightarrow\; &t' \in [t]_{F\chi_{\{1,2\}}} \text{ and } F\chi_{\{2\}}(t') \in [\![\kappa_s]\!]_2(\{1\}) && (\text{by the above calculation}) \\
\Leftrightarrow\; &t' \in [t]_{F\chi_{\{1,2\}}} \text{ and } F\chi_{\{2\}}(t') \in [F\chi_{\{2\}}(t)]_{F!} \cap [\![\kappa_s]\!]_2(\{1\}) && (\text{by } (5.2) \text{ since } t' \in [t]_{F\chi_{\{1,2\}}}) \\
\Leftrightarrow\; &t' \in [t]_{F\chi_{\{1,2\}}} \text{ and } F\chi_{\{2\}}(t') \in [s]_{F!} \cap [\![\kappa_s]\!]_2(\{1\}) && (\text{def. } s) \\
\Leftrightarrow\; &t' \in [t]_{F\chi_{\{1,2\}}} \text{ and } F\chi_{\{2\}}(t') \in \{s\} && (\text{assumption on } \kappa_s) \\
\Leftrightarrow\; &t' \in [t]_{F\chi_{\{1,2\}}} \text{ and } F\chi_{\{2\}}(t') \in \{F\chi_{\{2\}}(t)\} && (\text{def. } s) \\
\Leftrightarrow\; &F\chi_{\{1,2\}}(t') = F\chi_{\{1,2\}}(t) \text{ and } F\chi_{\{2\}}(t') = F\chi_{\{2\}}(t) \\
\Leftrightarrow\; &\langle F\chi_{\{1,2\}}, F\chi_{\{2\}}\rangle(t') = \langle F\chi_{\{1,2\}}, F\chi_{\{2\}}\rangle(t) && (\text{def. } \langle -, -\rangle) \\
\Leftrightarrow\; &t' = t && (F \text{ cancellative})
\end{aligned}
$$

For the last step, recall that $\langle F\chi_{\{1,2\}}, F\chi_{\{2\}}\rangle$ is injective because F is cancellative. $\qquad\square$

Lemma 5.6. *Example 5.3 correctly defines domain-specific interpretations.*

Proof. We verify the items in Example 5.3 separately:

(1) Recall that for $t \in \mathcal{P}3$, we have defined

$$\lambda_t(\delta, \rho) \;=\; \begin{cases} \neg \Diamond \rho & \text{if } 2 \in t \not\ni 1 \\ \Diamond \delta \wedge \Diamond \rho & \text{if } 2 \in t \ni 1 \\ \neg \Diamond \delta & \text{if } 2 \notin t \ni 1 \\ \top & \text{if } 2 \notin t \not\ni 1 \end{cases}$$

Evaluating $[\![\lambda_t]\!]_3(\delta, \rho)$ on $(\{2\}, \{1\})$ for the above cases, we obtain

$$[\![\neg \Diamond \rho]\!]_3(\{2\}, \{1\}) = \{t' \in \mathcal{P}3 \mid 1 \notin t'\}, \qquad\qquad \text{if } 2 \in t \not\ni 1$$
$$[\![\Diamond \delta \wedge \Diamond \rho]\!]_3(\{2\}, \{1\}) = \{t' \in \mathcal{P}3 \mid 2 \in t' \text{ and } 1 \in t'\}, \qquad \text{if } 2 \in t \ni 1$$
$$[\![\neg \Diamond \delta]\!]_3(\{2\}, \{1\}) = \{t' \in \mathcal{P}3 \mid 2 \notin t'\} \qquad\qquad \text{if } 2 \notin t \ni 1$$
$$[\![\Diamond \delta \wedge \Diamond \rho]\!]_3(\{2\}, \{1\}) = \mathcal{P}3. \qquad\qquad\qquad \text{if } 2 \notin t \not\ni 1$$

Intersecting with $[t]_{\mathcal{P}\chi_{\{1,2\}}}$ yields $\{t\}$ as desired in all cases:

t	$\lambda_t(\delta, \rho)$	$[\![\lambda_t]\!]_3(\{2\}, \{1\})$	$\cap\,[t]_{\mathcal{P}\chi_{\{1,2\}}}$	$= \{t\}$
$\{2\}$	$\neg \Diamond \rho$	$\{t' \in \mathcal{P}3 \mid 1 \notin t'\}$	$\cap \big\{\{2\}, \{1\}, \{2,1\}\big\}$	$= \big\{\{2\}\big\}$
$\{2,0\}$	$\neg \Diamond \rho$	$\{t' \in \mathcal{P}3 \mid 1 \notin t'\}$	$\cap \big\{\{2,0\}, \{1,0\}, \{2,1,0\}\big\}$	$= \big\{\{2,0\}\big\}$
$\{2,1\}$	$\Diamond \delta \wedge \Diamond \rho$	$\{t' \in \mathcal{P}3 \mid 2 \in t' \ \& \ 1 \in t'\}$	$\cap \big\{\{2\}, \{1\}, \{2,1\}\big\}$	$= \big\{\{2,1\}\big\}$
$\{2,1,0\}$	$\Diamond \delta \wedge \Diamond \rho$	$\{t' \in \mathcal{P}3 \mid 2 \in t' \ \& \ 1 \in t'\}$	$\cap \big\{\{2,0\}, \{1,0\}, \{2,1,0\}\big\}$	$= \big\{\{2,1,0\}\big\}$
$\{1\}$	$\neg \Diamond \delta$	$\{t' \in \mathcal{P}3 \mid 2 \notin t'\}$	$\cap \big\{\{2\}, \{1\}, \{2,1\}\big\}$	$= \big\{\{1\}\big\}$
$\{1,0\}$	$\neg \Diamond \delta$	$\{t' \in \mathcal{P}3 \mid 2 \notin t'\}$	$\cap \big\{\{2,0\}, \{1,0\}, \{2,1,0\}\big\}$	$= \big\{\{1,0\}\big\}$
$\{0\}$	$\top$	$\mathcal{P}3$	$\cap \big\{\{0\}\big\}$	$= \big\{\{0\}\big\}$
$\emptyset$	$\top$	$\mathcal{P}3$	$\cap \big\{\emptyset\big\}$	$= \big\{\emptyset\big\}$

Hence, $[\![\lambda_t]\!]_3(\{2\}, \{1\}) \cap [t]_{\mathcal{P}\chi_{\{1,2\}}} = \{t\}$.

(2) For a signature functor Σ, we first define a helper map $v\colon \Sigma 2 \to \mathcal{P}_\mathsf{f}\mathbb{N}$ by

$$v(\sigma(x_1, \ldots, x_n)) = \{i \in \mathbb{N} \mid x_i = 1\}.$$

The predicate lifting for the (unary) modal operator $\langle = I \rangle$, for $I \subseteq \mathbb{N}$, is obtained from Lemma 2.8 by the predicate $f_I\colon \Sigma 2 \to 2$ corresponding to the subset

$$f_I = \{t \in \Sigma 2 \mid v(t) = I\}.$$

This gives rise to the predicate lifting

$$[\![\langle = I \rangle]\!]_X(P) = \{t \in \Sigma X \mid F\chi_P(t) \in f_I\} \qquad\qquad \text{(Lemma 2.8)}$$
$$= \{t \in \Sigma X \mid v(F\chi_P(t)) = I\} \qquad\qquad \text{(def. } f_I\text{)}.$$

Similarly, for the nullary modal operator σ given by $\sigma/n \in \Sigma$, take the predicate $\Sigma 1 \to 2$ corresponding to the subset

$$g_\sigma = \{\sigma(0, \ldots, 0)\} \subseteq \Sigma 1$$

Since $1 = 2^0$, this gives rise to the 0-ary predicate lifting

$$[\![\sigma]\!]_X = \{t \in \Sigma X \mid F!(t) \in g_\sigma\} \qquad\qquad \text{(Lemma 2.8)}$$
$$= \{t \in \Sigma X \mid F\chi_P(t) \in \{\sigma(0, \ldots, 0)\}\} \qquad\qquad \text{(def. } g_\sigma\text{)}$$

$$= \{\sigma(x_1, \ldots, x_n) \mid x_1, \ldots, x_n \in X\}.$$

We now put

$$\kappa_s(\delta) := \langle =v(s)\rangle\delta \qquad \text{for } s \in \Sigma 2,$$

and we proceed to show that this yields a simple domain-specific interpretation (Definition 5.4) and that it induces the desired λ_t via Proposition 5.5:

$$\lambda_{\sigma(x_1, \ldots, x_n)}(\delta, \rho) = \langle =\{i \in \mathbb{N}|x_i=2\}\rangle\delta \qquad \text{for } \sigma(x_1, \ldots, x_n) \in \Sigma 3.$$

There is nothing to show for $\tau_o := \sigma$ since it has the correct semantics by the definition of $[\![\sigma]\!]_1$. Next note that the map $\langle\Sigma!, v\rangle \colon \Sigma 2 \to \Sigma 1 \times \mathcal{P}_{\mathsf{f}}\mathbb{N}$ is injective because for every $s \in \Sigma 2$, the operation symbol and all its parameters (from 2) are uniquely determined by $\Sigma!(s)$ and $v(s)$. Recalling that $[s]_{\Sigma!} = \{s' \in \Sigma 2 \mid \Sigma!(s) = \Sigma!(s')\}$, we now compute

$$
\begin{aligned}
[s]_{\Sigma!} \cap [\![\kappa_s]\!]_2(\{1\}) &= \{s' \in \Sigma 2 \mid s' \in [s]_{\Sigma!} \text{ and } s' \in [\![\kappa_s]\!]_2(\{1\})\} \\
&= \{s' \in \Sigma 2 \mid \Sigma!(s) = \Sigma!(s') \text{ and } s' \in [\![\langle =v(s)\rangle]\!]_2(\{1\})\} \\
&= \{s' \in \Sigma 2 \mid \Sigma!(s) = \Sigma!(s') \text{ and } v(\Sigma\chi_{\{1\}}(s')) = v(s)\} \quad (\text{def. } [\![\langle =v(s)\rangle]\!]_2) \\
&= \{s' \in \Sigma 2 \mid \Sigma!(s) = \Sigma!(s') \text{ and } v(s') = v(s)\} \quad (\mathsf{id}_2 = \chi_{\{1\}} \colon 2 \to 2) \\
&= \{s' \in \Sigma 2 \mid \langle\Sigma!, v\rangle(s) = \langle\Sigma!, v\rangle(s')\} \quad (\text{def. } \langle -, -\rangle) \\
&= \{s\} \quad (\langle\Sigma!, v\rangle \text{ injective}).
\end{aligned}
$$

(3) For every $m \in M$, let $f_m \colon M^{(2)} \to 2$ be the predicate corresponding to the subset

$$\{\mu \in M^{(2)} \mid \mu(1) = m\}.$$

It induces the unary predicate lifting $[\![\langle =m\rangle]\!]$ by

$$
\begin{aligned}
[\![\langle =m\rangle]\!]_X(P) &= \{\mu \in M^{(X)} \mid M^{(P)}(\mu) \in f_m\} \qquad\qquad (\text{Lemma 2.8}) \\
&= \{\mu \in M^{(X)} \mid M^{(P)}(\mu)(1) = m\} \qquad\qquad (\text{def. } f_m).
\end{aligned}
$$

To see that we have a domain-specific interpretation (Definition 5.2), we note first (using $[\![\top]\!]_1 = 1 = \{0\}$) that τ satisfies

$$
\begin{aligned}
[\![\tau_o]\!]_1 = [\![\langle =o(0)\rangle\top]\!]_1 &= \{\mu \in M^{(1)} \mid \sum_{x \in [\![\top]\!]_1} \mu(x) = o(0)\} \\
&= \{\mu \in M^{(1)} \mid \mu(0) = o(0)\} \\
&= \{o\}.
\end{aligned}
$$

For the second component of the domain-specific interpretation, we proceed by case distinction:

- If M is non-cancellative, we have $\lambda_t(\delta, \rho) = \langle =t(2)\rangle\delta \wedge \langle =t(1)\rangle\rho$ for $t \in M^{(3)}$. Thus, we obtain the following chain of equivalences for every $t' \in M^{(3)}$:

$$
\begin{aligned}
&t' \in ([t]_{F\chi_{\{1,2\}}} \cap [\![\lambda_t]\!]_3(\{2\}, \{1\})) \\
\Leftrightarrow\ &t' \in [t]_{F\chi_{\{1,2\}}} \text{ and } t' \in [\![\lambda_t]\!]_3(\{2\}, \{1\}) \\
\Leftrightarrow\ &t' \in [t]_{F\chi_{\{1,2\}}} \text{ and } t' \in [\![(\delta, \rho) \mapsto \langle =t(2)\rangle\delta \wedge \langle =t(1)\rangle\rho]\!]_3(\{2\}, \{1\}) \quad (\text{def. } \lambda_t) \\
\Leftrightarrow\ &t' \in [t]_{F\chi_{\{1,2\}}} \text{ and } t' \in [\![\langle =t(2)\rangle]\!]_3(\{2\}) \cap [\![\langle =t(1)\rangle]\!]_3(\{1\}) \\
\Leftrightarrow\ &t' \in [t]_{F\chi_{\{1,2\}}} \text{ and } t' \in [\![\langle =t(2)\rangle]\!]_3(\{2\}) \text{ and } t' \in [\![\langle =t(1)\rangle]\!]_3(\{1\}) \\
\Leftrightarrow\ &t' \in [t]_{F\chi_{\{1,2\}}} \text{ and } t'(2) = t(2) \text{ and } t'(1) = t(1) \quad (\text{def. } [\![\langle =m\rangle]\!]) \\
\Leftrightarrow\ &t'(0) = t(0) \text{ and } t'(1) + t'(2) = t(1) + t(2) \text{ and}
\end{aligned}
$$

$$t'(2) = t(2) \text{ and } t'(1) = t(1)$$
$$\Leftrightarrow t'(0) = t(0) \text{ and } t'(2) = t(2) \text{ and } t'(1) = t(1)$$
$$\Leftrightarrow t' = t$$
$$\Leftrightarrow t' \in \{t\}.$$

- If M is cancellative, we put $\kappa_s(\delta) = \langle =s(1) \rangle\, \delta$ for $s \in M^{(2)}$, which then induces $\lambda_t(\delta, \rho) = \langle =s(2) \rangle\, \delta$ via Proposition 5.5. Hence, we verify that κ is part of a simple domain-specific interpretation (Definition 5.4). Indeed, for every $s' \in M^{(2)}$ we have the following chain of equivalences:

$$s' \in ([s]_{F!} \cap [\![\kappa_s]\!]_2(\{1\}))$$
$$\Leftrightarrow s' \in [s]_{F!} \text{ and } s' \in [\![\kappa_s]\!]_2(\{1\})$$
$$\Leftrightarrow F!(s') = F!(s) \text{ and } s' \in [\![\langle =s(1) \rangle]\!]_2(\{1\}) \qquad\qquad (\text{def. } \kappa_s)$$
$$\Leftrightarrow F!(s') = F!(s) \text{ and } \textstyle\sum_{x \in \{1\}} s'(x) = s(1) \qquad\qquad (\text{def. } \langle =s(1) \rangle)$$
$$\Leftrightarrow F!(s') = F!(s) \text{ and } s'(1) = s(1)$$
$$\Leftrightarrow s'(0) + s'(1) = s(0) + s(1) \text{ and } s'(1) = s(1)$$
$$\Leftrightarrow s'(0) = s(0) \text{ and } s'(1) = s(1) \qquad\qquad (M \text{ cancellative})$$
$$\Leftrightarrow s' = s$$
$$\Leftrightarrow s' \in \{s\}.$$

(4) For $FX = (\mathcal{D}X + 1)^A$, recall that the predicate lifting $[\![\langle a \rangle_p]\!]$, where $a \in A$, $p \in [0,1]$, is given by

$$[\![\langle a \rangle_p]\!]_X(S) = \{t \in FX \mid \text{if } p > 0, \text{ then } t(a) \in \mathcal{D}X \text{ and } \textstyle\sum_{x \in S} t(a)(x) \geq p\}.$$

First note that

$$[\![\langle a \rangle_1 \top]\!]_1 = \{o \in F1 \mid o(a) \in \mathcal{D}1\} \qquad \text{and} \qquad [\![\neg\langle a \rangle_1 \top]\!]_1 = \{o \in F1 \mid o(a) \in 1\}.$$

Thus, we have:

$$[\![\tau_o]\!]_1 = \Big[\!\!\Big[\bigwedge_{\substack{a \in A \\ o(a) \in \mathcal{D}1}} \langle a \rangle_1 \top \wedge \bigwedge_{\substack{a \in A \\ o(a) \in 1}} \neg\langle a \rangle_1 \top \Big]\!\!\Big]_1$$

$$= \bigcap_{\substack{a \in A \\ o(a) \in \mathcal{D}1}} \{o' \in F1 \mid o'(a) \in \mathcal{D}1\} \cap \bigcap_{\substack{a \in A \\ o(a) \in 1}} \{o' \in F1 \mid o'(a) \in 1\}$$

$$= \{o\}.$$

For λ_t, $t \in F3 = (\mathcal{D}3 + 1)^A$, we have the following chain of equivalences for every $t' \in F3$ (note that the crucial step is the arithmetic argument for replacing the inequalities with equalities):

$$t' \in ([t]_{F\chi_{\{1,2\}}} \cap [\![\lambda_t]\!]_3(\{2\}, \{1\}))$$
$$\Leftrightarrow t' \in [t]_{F\chi_{\{1,2\}}} \text{ and } t' [\![\lambda_t]\!]_3(\{2\}, \{1\})$$
$$\Leftrightarrow t' \in [t]_{F\chi_{\{1,2\}}} \text{ and } t' \in \Big[\!\!\Big[(\delta, \rho) \mapsto \bigwedge_{\substack{a \in A \\ t(a) \in \mathcal{D}3}} (\langle a \rangle_{t(a)(2)}\, \delta \wedge \langle a \rangle_{t(a)(1)}\, \rho) \Big]\!\!\Big]_3(\{2\}, \{1\})$$

$\Leftrightarrow t' \in [t]_{F\chi_{\{1,2\}}}$ and $t' \in \bigcap_{\substack{a \in A \\ t(a) \in \mathcal{D}3}} [\![(\delta, \rho) \mapsto \langle a \rangle_{t(a)(2)} \delta \wedge \langle a \rangle_{t(a)(1)} \rho]\!]_3(\{2\}, \{1\})$

$\Leftrightarrow t' \in [t]_{F\chi_{\{1,2\}}}$ and $t' \in \bigcap_{\substack{a \in A \\ t(a) \in \mathcal{D}3}} [\![\langle a \rangle_{t(a)(2)}]\!]_3(\{2\}) \cap [\![\langle a \rangle_{t(a)(1)}]\!]_3(\{1\})$

$\Leftrightarrow t' \in [t]_{F\chi_{\{1,2\}}}$ and $\forall a \in A, t(a) \in \mathcal{D}3 : t' \in [\![\langle a \rangle_{t(a)(2)}]\!]_3(\{2\}) \cap [\![\langle a \rangle_{t(a)(1)}]\!]_3(\{1\})$

$\Leftrightarrow t' \in [t]_{F\chi_{\{1,2\}}}$ and $\forall a \in A, t(a) \in \mathcal{D}3 : t'(a)(2) \geq t(a)(2) \wedge t'(a)(1) \geq t(a)(1)$
    (def. $[\![\langle a \rangle p]\!]$)

$\Leftrightarrow \forall a \in A : (t'(a) \in 1 \leftrightarrow t(a) \in 1)$ and if $t(a) \in \mathcal{D}3$ then:
    $t'(a)(0) = t(a)(0),\ t'(a)(1) + t'(a)(2) = t(a)(1) + t(a)(2),$
    $t'(a)(2) \geq t(a)(2),\ t'(a)(1) \geq t(a)(1)$

$\Leftrightarrow \forall a \in A : (t'(a) \in 1 \leftrightarrow t(a) \in 1)$ and if $t(a) \in \mathcal{D}3$ then:
    $t'(a)(0) = t(a)(0),\ t'(a)(1) = t(a)(1),\ t'(a)(2) = t(a)(2)$ (arithmetic)

$\Leftrightarrow \forall a \in A : (t'(a) \in 1 \leftrightarrow t(a) \in 1)$ and if $t(a) \in \mathcal{D}3$ then $t'(a) = t(a)$

$\Leftrightarrow t' \in \{t\}.$ $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

The intuitive meaning of a domain-specific interpretation (Definition 5.2) is formalized by the following technical result.

Lemma 5.7. *Let (τ, λ) be a domain-specific interpretation for F. For all $t \in FC$ and $S \subseteq B \subseteq C$, we have*

$$\left([t]_{F\chi_B} \ \cap\ [\![\lambda_{F\chi_S^B(t)}]\!]_C(S, B \setminus S)\right) = [t]_{F\chi_S^B} \qquad in\ 2^{FC}.$$

Proof. Put $d = F\chi_S^B(t)$; the naturality square of $[\![\lambda_d]\!]$ for $\chi_S^B : C \to 3$ is

$$
\begin{array}{ccc}
2^3 \times 2^3 & \xrightarrow{\ [\![\lambda_d]\!]_3\ } & 2^{F3} \\
{\scriptstyle 2^{\chi_S^B} \times 2^{\chi_S^B}} \downarrow & & \downarrow {\scriptstyle 2^{F\chi_S^B}} \\
2^C \times 2^C & \xrightarrow{\ [\![\lambda_d]\!]_C\ } & 2^{FC}
\end{array}
$$

Hence:

$$(F\chi_S^B)^{-1}\big[[\![\lambda_d]\!]_3(\{2\}, \{1\})\big] = [\![\lambda_d]\!]_C((\chi_S^B)^{-1}[\{2\}], (\chi_S^B)^{-1}[\{1\}])$$
$$= [\![\lambda_d]\!]_C(B, B \setminus S). \qquad\qquad\qquad (*)$$

Now we have the following chain of equivalences for every $t' \in FC$:

$t' \in \big([t]_{F\chi_B} \ \cap\ [\![\lambda_d]\!]_C(S, B \setminus S)\big)$

$\Leftrightarrow t' \in [t]_{F\chi_B}$ and $t' \in [\![\lambda_d]\!]_C(S, B \setminus S)$

$\Leftrightarrow t' \in [t]_{F\chi_B}$ and $t' \in (F\chi_S^B)^{-1}\big[[\![\lambda_d]\!]_3(\{2\}, \{1\})\big]$ $\qquad\qquad\qquad$ (by $(*)$)

$\Leftrightarrow F\chi_S^B(t') \in [F\chi_S^B(t)]_{F\chi_{\{1,2\}}}$ and $F\chi_S^B(t') \in [\![\lambda_d]\!]_3(\{2\}, \{1\})$ $\qquad$ $(\chi_{\{1,2\}} \cdot \chi_S^B = \chi_B)$

$\Leftrightarrow F\chi_S^B(t') \in [F\chi_S^B(t)]_{F\chi_{\{1,2\}}} \cap [\![\lambda_d]\!]_3(\{2\}, \{1\})$

$\Leftrightarrow F\chi_S^B(t') \in \{F\chi_S^B(t)\}$ $\qquad\qquad\qquad\qquad\qquad$ (Definition 5.2, $d = F\chi_S^B(t)$)

$\Leftrightarrow F\chi_S^B(t') = F\chi_S^B(t)$

$$\Leftrightarrow t' \in [t]_{F\chi_S^B}. \qquad\qquad\qquad \square$$

Given a domain-specific interpretation (τ, λ) for a modal signature Λ for the set functor F, we can postprocess certificates ϕ produced by Algorithm 3.15 by replacing the modalities $\ulcorner t \urcorner$ for $t \in F3$ according to the translation T recursively defined by the following clauses for modalities and by commutation with propositional operators:

$$T\big(\ulcorner t\urcorner(\top, \top)\big) = \tau_{F!(t)} \qquad T\big(\ulcorner t\urcorner(\delta, \beta)\big) = \lambda_t\big(T(\delta), T(\beta) \wedge \neg T(\delta)\big).$$

Note that one can replace $T(\beta) \wedge \neg T(\delta)$ with $T(\beta) \wedge \neg T(\delta')$ for the optimized δ' from Proposition 3.28; the latter conjunction has essentially the same size as $T(\delta)$.

The domain-specific modal signatures inherit a Hennessy-Milner Theorem.

Proposition 5.8. *For every certificate ϕ of a behavioural equivalence class of a given coalgebra produced by either Algorithm 3.15 or its optimization (Algorithm 4.13), $T(\phi)$ is also a certificate for that class.*

Proof. We prove by induction on the index i of main loop iterations that $T(\delta_i([x]_{P_i}))$ and $T(\beta_i([x]_{Q_i}))$ are certificates for $[x]_{P_i}$ and $[x]_{Q_i}$, respectively. (In the cancellative case, Q_i and β_i are not defined; so just put $C/Q_i = \{C\}$, $\beta_i(C) = \top$ for convenience.)

(1) For $i = 0$, we trivially have

$$\llbracket T(\beta_0([x]_{P_i})) \rrbracket = \llbracket T(\top) \rrbracket = \llbracket \top \rrbracket = C.$$

Furthermore, unravelling Notation 3.13, we have

$$\delta_0([x]_{P_0}) = \ulcorner F!(c(x)) \urcorner = \ulcorner Fj_1(F!(c(x))) \urcorner(\top, \top).$$

Consequently,

$$T(\delta_0([x]_{P_0})) = \tau_{F!(Fj_1(F!(c(x))))} = \tau_{F!(c(x))}$$

using $! \cdot j_1 \cdot ! = !\colon C \to 1$. Naturality of $\llbracket \tau_o \rrbracket$, $o \in F1$, implies that

$$\llbracket \tau_o \rrbracket_X = \{t \in FX \mid F!(t) = o\}.$$

Hence, we obtain the desired equality:

$$\llbracket T(\delta_0([x]_{P_0})) \rrbracket = c^{-1}[\llbracket \tau_{F!(c(x))} \rrbracket_C] = \{x' \in C \mid F!(c(x')) = F!(c(x))\} = [x]_{P_0}.$$

(2) In the inductive step, there is nothing to show for β_{i+1} because it is only a boolean combination of β_i and δ_i. For δ_{i+1}, we distinguish two cases: whether the class $[x]_{P_i}$ is refined or not. If $[x]_{P_{i+1}} = [x]_{P_i}$, then

$$\llbracket T(\delta_{i+1}([x]_{P_{i+1}})) \rrbracket = \llbracket T(\delta_i([x]_{P_i})) \rrbracket = [x]_{P_i},$$

and we are done. Now suppose that $[x]_{P_{i+1}} \neq [x]_{P_i}$ in the i-th iteration with chosen $S \subsetneq B \subseteq C$. By step **(A'3)** of Algorithm 3.15 (or Algorithm 4.13, respectively), we have

$$\delta_{i+1}([x]_{P_{i+1}}) = \delta_i([x]_{P_i}) \wedge \ulcorner t \urcorner(\delta_i(S), \beta')$$

where β' is $\beta_i(B)$ or $\top$; in any case $\llbracket \delta_i(S) \rrbracket = S \subseteq \llbracket \beta' \rrbracket$. Note that t here is either $F\chi_S^B(c(x))$ (Algorithm 3.15) or $Fj_2(F\chi_S(c(x)))$ (Algorithm 4.13). Put $B' = B$ in the first case and $B' = C$ else. Using $\chi_S^C = j_2 \cdot \chi_S$, we see that

$$t = F\chi_S^{B'}(c(x)) \qquad \llbracket \beta' \rrbracket = B', \qquad \text{and} \qquad \llbracket T(\beta') \rrbracket = B',$$

where the last equation follows from the inductive hypothesis. Thus, we have

$$\delta_{i+1}([x]_{P_{i+1}}) = \delta_i([x]_{P_i}) \wedge \ulcorner F\chi_S^{B'}(c(x)) \urcorner(\delta_i(S), \beta'),$$

and therefore

$$T(\delta_{i+1}([x]_{P_{i+1}})) = T(\delta_i([x]_{P_i})) \wedge \lambda_{F\chi_S^{B'}(c(x))}\big(T(\delta_i(S)), T(\beta') \wedge \neg T(\delta_i(S))\big).$$

Moreover, we have

$$P_{i+1} = P_i \cap \ker(F\chi_S^{B'} \cdot c),$$

in the first case by step **(A3)**, and in the second case by Proposition 4.8, recalling that $\chi_S = \chi_S^C$.

We are now prepared for our final computation:

$$\begin{aligned}
&\llbracket T(\delta_{i+1}([x]_{P_{i+1}})) \rrbracket \\
&= \llbracket T(\delta_i([x]_{P_i})) \wedge \lambda_{F\chi_S^{B'}(c(x))}(T(\delta_i(S)), T(\beta') \wedge \neg T(\delta_i(S))) \rrbracket \\
&= \llbracket T(\delta_i([x]_{P_i})) \rrbracket \cap \llbracket \lambda_{F\chi_S^{B'}(c(x))}(T(\delta_i(S)), T(\beta') \wedge \neg T(\delta_i(S))) \rrbracket \\
&= \llbracket T(\delta_i([x]_{P_i})) \rrbracket \cap c^{-1}\big[\llbracket \lambda_{F\chi_S^{B'}(c(x))} \rrbracket_C (\llbracket T(\delta_i(S)) \rrbracket, \llbracket T(\beta') \rrbracket \cap C \setminus \llbracket T(\delta_i(S)) \rrbracket) \big] \\
&\hspace{8cm} \text{(semantics of } \lambda) \\
&= [x]_{P_i} \cap c^{-1}\big[\llbracket \lambda_{F\chi_S^{B'}(c(x))} \rrbracket_C (S, B' \cap (C \setminus S)) \big] \hspace{1.5cm} \text{(induction hypothesis)} \\
&= [x]_{P_i} \cap c^{-1}\big[\llbracket \lambda_{F\chi_S^{B'}(c(x))} \rrbracket_C (S, B' \setminus S) \big] \hspace{2cm} (B' \cap (C \setminus S) = B' \setminus S) \\
&= [x]_{P_i} \cap [x]_{F\chi_{B'} \cdot c} \cap c^{-1}\big[\llbracket \lambda_{F\chi_S^{B'}(c(x))} \rrbracket_C (S, B' \setminus S) \big] \hspace{1cm} (P_i \subseteq \ker F\chi_{B'} \cdot c) \\
&= [x]_{P_i} \cap c^{-1}\big[[c(x)]_{F\chi_{B'}} \big] \cap c^{-1}\big[\llbracket \lambda_{F\chi_S^{B'}(c(x))} \rrbracket_C (S, B' \setminus S) \big] \\
&= [x]_{P_i} \cap c^{-1}\big[[c(x)]_{F\chi_{B'}} \cap \llbracket \lambda_{F\chi_S^{B'}(c(x))} \rrbracket_C (S, B' \setminus S) \big] \\
&= [x]_{P_i} \cap c^{-1}\big[[c(x)]_{F\chi_S^{B'}} \big] \hspace{2.5cm} \text{(domain-specific interpret. (Lemma 5.7))} \\
&= [x]_{P_i} \cap [x]_{F\chi_S^{B'} \cdot c} \\
&= [x]_{P_{i+1}} \hspace{5.5cm} (P_{i+1} = P_i \cap \ker(F\chi_S^{B'} \cdot c))
\end{aligned}$$

Thus, $\llbracket T(\delta_{i+1}([x]_{P_{i+1}})) \rrbracket$ is a certificate. $\qquad\square$

Example 5.9. For labelled Markov chains $(FX = (\mathcal{D}X + 1)^A)$ and the interpretation via the modalities $\langle a \rangle_p$ (Example 5.3(4)), we thus obtain certificates (in particular also distinguishing formulae) in run time $\mathcal{O}(|A| \cdot m \cdot \log n)$, with the same bound on formula size. Indeed, the Algorithm 3.15 runs in $\mathcal{O}(m \cdot \log n)$ producing certificates of total size $\mathcal{O}(m \cdot \log n)$. By Proposition 5.8, we can translate these certificates into ones using the modalities $\langle a \rangle_p$. The translation blows up the size of certificates by the additional factor $|A|$ appearing in the above size estimate because of the big conjunctions in the domain-specific interpretation (Example 5.3(4)).

By comparison, the algorithm for distinguishing formulae by Desharnais et al. [DEP02, Fig. 4] runs roughly in time $\mathcal{O}(|A| \cdot n^4)$, as it nests four loops over all blocks seen so far and one additional loop over A. The distinguishing formulae computed by the algorithm live in the negation-free fragment of the logic that we use for certificates; Desharnais et al. note that this fragment does not suffice for certificates.

6. Worst Case Tree Size of Certificates

In the complexity analysis (Section 3.5), we have seen that certificates – and thus also distinguishing formulae – have dag size $\mathcal{O}(m \cdot \log n + n)$ on input coalgebras with n states and m transitions. However, when formulae are written in the usual linear way, multiple occurrences of the same subformula lead to an exponential blow up of the formula size in this sense, which for emphasis we refer to as the *tree size*.

6.1. **Transition Systems.** The certificate of a state can be exponentially large and even the size of formulae separating two particular states of interest is in the worst case as big as the certificate of one the states. This has been shown previously by Figueira and Gorín [FG10] via winning strategies in bisimulation games, a technique that is also applied in other works giving lower bounds for formula size [FvIK13, AI01, AI03]. For the convenience of the reader, we recall the example and give a direct argument for the size estimate in the appendix.

Example 6.1. We define a $\mathcal{P}_{\mathsf{f}}$-coalgebra (C, c) with state set $C = \bigcup_{i \in \mathbb{N}} L_i$ made up of *layers* $L_i = \{x_i, y_i, z_i\}$. The successors of states in layer L_{i+1} are in layer L_i; specifically,

$$c(x_0) = \{y_0\} \quad c(x_{i+1}) = (0, \{x_i, y_i, z_i\})$$
$$c(y_0) = \emptyset \quad\quad c(y_{i+1}) = (0, \{y_i, z_i\})$$
$$c(z_0) = \{x_0\} \quad c(z_{i+1}) = (0, \{x_i, z_i\}).$$

No two distinct states of (C, c) are bisimilar. For a lower bound on tree size of certificates, one shows that a certificate of x_{n+1} necessarily contains (distinct) certificates of x_n and y_n, and a certificate of y_{n+1} contains one of x_n. Hence, every certificate of x_n has size at least $\mathsf{fib}(n)$, the n-th Fibonacci number. Moreover, by the above, every formula distinguishing x_{n+1} from y_{n+1} contains a certificate for x_n, and thus also has size at least $\mathsf{fib}(n)$.

Cleaveland [Cle91, p. 368] also mentions that minimal distinguishing formulae may be exponential in size, however for a slightly different notion of minimality: a formula ϕ distinguishing x from y is called *minimal* by Cleaveland if no ϕ obtained by replacing a non-trivial subformula of ϕ with the formula $\top$ distinguishes x from y. This is weaker than demanding that the formula size of ϕ is as small as possible. For example, in the transition system

the formula $\phi = \Diamond^{n+2}\top$ distinguishes x from y and is minimal in the above sense. However, x can in fact be distinguished from y in size $\mathcal{O}(1)$, by the formula $\Diamond\neg\Diamond\top$.

To verify the minimality of $\phi = \Diamond^{n+2}\top$, one considers all possible replacements of subformulae of ϕ by $\top$:

$$\Diamond\top \quad\quad \Diamond\Diamond\top \quad\quad \cdots \quad\quad \Diamond^n\top \quad\quad \Diamond^{n+1}\top$$

All of these hold at both x and y, because x can perform arbitrarily many transitions, and y can perform $n + 1$ transitions.

6.2. **Weighted Systems.** In contrast to transition systems, lower bounds on the size of distinguishing formulae have not been established.

As a negative result, even the optimized algorithm for cancellative functors presented above (Algorithm 4.13) constructs certificates of exponential worst-case tree size, even in cases where linear-sized certificates exist.

Example 6.2. Define the $\mathbb{R}^{(-)}$-coalgebra c on $C = \bigcup_{k \in \mathbb{N}} \{w_k, x_k, y_k, z_k\}$ by

$$
\begin{aligned}
c(w_{k+1}) &= \{w_k \mapsto 1, x_k \mapsto 2, y_k \mapsto 1, z_k \mapsto 2\}, & c(w_0) &= \{w_0 \mapsto 1\}, \\
c(x_{k+1}) &= \{w_k \mapsto 1, x_k \mapsto 2, y_k \mapsto 2, z_k \mapsto 1\}, & c(x_0) &= \{x_0 \mapsto 2\}, \\
c(y_{k+1}) &= \{w_k \mapsto 2, x_k \mapsto 1, y_k \mapsto 1, z_k \mapsto 2\}, & c(y_0) &= \{y_0 \mapsto 3\}, \\
c(z_{k+1}) &= \{w_k \mapsto 2, x_k \mapsto 1, y_k \mapsto 2, z_k \mapsto 1\}, & c(z_0) &= \{z_0 \mapsto 4\}.
\end{aligned}
$$

We say that $L_k = \{w_k, x_k, y_k, z_k\}$ is the k-th *layer* of this coalgebra. So the states in the 0-th layer each just have a loop with weight 1, 2, 3, and 4, respectively, and the states of the $k+1$-st layer and the k-th one are connected by a complete bipartite graph with weights as indicated above.

We now show that Algorithm 4.13 constructs a certificate of size 2^n in the n-th layer. Hence, for the finite subcoalgebra $L_0 \cup \cdots \cup L_n$, the states in L_n receive certificates of size exponential in the size of the input coalgebra, that is the number $4 + 4n$ of states plus the number $4 + 16n$ of edges.

To see this, first note that the initial partition

$$
P_0 = \{\{w_0\}, \{x_0\}, \{y_0\}, \{z_0\}, L_1 \cup \cdots \cup L_n\}
$$

distinguishes on the total out-degree (being 1, 2, 3, 4, or 6). The states in L_0 are assigned the following certificates:

$$
w_0 = \langle =1 \rangle \top, \qquad x_0 = \langle =2 \rangle \top, \qquad y_0 = \langle =3 \rangle \top, \qquad \text{and} \qquad z_0 = \langle =4 \rangle \top. \tag{6.1}
$$

Assume that after i iterations of the main loop of the algorithm, the states w_k, x_k, y_k, z_k have just been found to be behaviourally different and all states of $L_{k+1} \cup \cdots \cup L_n$ are still identified. Then the algorithm has to use some of the blocks $\{w_k\}, \{x_k\}, \{y_k\}, \{z_k\}$ as the splitter S for further refinement. Assume wlog that the first block used as the splitter is $S = \{w_k\}$. Then the block $L_{k+1} \cup \cdots \cup L_n$ will be refined into the blocks

$$
\{w_{k+1}, x_{k+1}\}, \qquad \{y_{k+1}, z_{k+1}\}, \qquad \text{and} \qquad L_{k+2} \cup \cdots \cup L_n.
$$

Denote by $\delta(\{w_k\})$ the formula that we have at this point for $\{w_k\}$. The definition of δ in the algorithm annotates the block $\{w_{k+1}, x_{k+1}\}$ with $\langle =1 \rangle \delta(\{w_k\})$ and the block $\{y_{k+1}, z_{k+1}\}$ with $\langle =2 \rangle \delta(\{w_k\})$.

Splitting by $\{x_k\}$ does not lead to further refinement. However, when splitting by $S = \{y_k\}$ (or equivalently $\{z_k\}$), we split $\{w_{k+1}, x_{k+1}\}$ into $\{w_{k+1}\}$ and $\{x_{k+1}\}$ and likewise $\{y_{k+1}, z_{k+1}\}$ into $\{y_{k+1}\}$ and $\{z_{k+1}\}$. Let $\delta(\{y_k\})$ be the certificate constructed for $\{y_k\}$. This implies that the formulae for $\{w_{k+1}\}$ and $\{y_{k+1}\}$ are both extended by the conjunct $\langle =1 \rangle \delta(\{y_k\})$; likewise, the formulae for $\{x_{k+1}\}$ and $\{z_{k+1}\}$ are extended by a new conjunct $\langle =2 \rangle \delta\{y_k\}$. Hence, for every $s \in L_{k+1}$ the tree-size of the constructed formula is at least

$$
|\delta(\{s\})| \geq |\delta(\{w_k\})| + |\delta(\{y_k\})|.
$$

By induction we thus see that the certificates for $s \in L_k$ is of size at least 2^k.

Remark 6.3. However, note that in Example 6.2, linear-sized certificates do exist for all states.

Indeed, for states in L_0 we have the certificates from (6.1). Using those, we can easily read off the following certificates for w_1 and z_1 from the coalgebra structure:

$$\langle=1\rangle\langle=1\rangle\top \wedge \langle=2\rangle\langle=2\rangle\top \wedge \langle=1\rangle\langle=3\rangle\top \wedge \langle=2\rangle\langle=4\rangle\top, \text{ and}$$

$$\langle=1\rangle\langle=1\rangle\top \wedge \langle=2\rangle\langle=2\rangle\top \wedge \langle=2\rangle\langle=3\rangle\top \wedge \langle=1\rangle\langle=4\rangle\top.$$

For all other states, we obtain certificates as follows. For every $k \geq 0$ we have

$$\phi_k := \langle=3\rangle^k(\langle=1\rangle\top \vee \langle=4\rangle\top) \qquad \text{satisfying} \qquad [\![\phi_k]\!] = \{w_k, z_k\}.$$

This lets us define certificates for $\{x_{k+1}\}$ and $\{y_{k+1}\}$:

$$[\![\langle=2\rangle\phi_k]\!] = \{x_{k+1}\} \quad \text{and} \quad [\![\langle=4\rangle\phi_k]\!] = \{y_{k+1}\}.$$

For the remaining states w_k and z_k, we note that

$$[\![\langle=1\rangle\langle=4\rangle\phi_k]\!] = \{w_{k+2}, y_{k+2}\}.$$

Thus, we have certificates

$$[\![\phi_{k+2} \wedge \langle=1\rangle\langle=4\rangle\phi_k]\!] = \{w_{k+2}\} \quad \text{and} \qquad [\![\phi_{k+2} \wedge \neg\langle=1\rangle\langle=4\rangle\phi_k]\!] = \{z_{k+2}\}.$$

Since ϕ_k involves $k + 2$ modal operators, every state in L_k has a certificate with at most $2 \cdot k + 8$ modal operators.

Hence, in each of the finite coalgebras $L_0 \cup \cdots \cup L_k$ from Example 6.2, every state has a certificate whose size is linearly bounded in the size of the coalgebra.

Open Problem 6.4. Do states in $\mathbb{R}^{(-)}$-coalgebras generally have certificates of subexponential tree size in the number of states? If yes, can small certificates be computed efficiently?

We note that for another cancellative functor, the answer is well-known: On deterministic automata, i.e. coalgebras for $FX = 2 \times X^A$, the standard minimization algorithm constructs distinguishing words of linear length.

7. Conclusions and Further Work

We have presented a generic algorithm that computes distinguishing formulae for behaviourally inequivalent states in state-based systems of various types, cast as coalgebras for a functor capturing the system type. Our algorithm is based on an efficient coalgebraic partition refinement algorithm [WDMS20], and like that algorithm runs in time $\mathcal{O}((m+n)\cdot\log n\cdot p(c))$, with a functor-specific factor $p(c)$ that is 1 in many cases of interest. Independently of this factor, the distinguishing formulae constructed by the algorithm have dag size $\mathcal{O}(m\cdot\log n+n)$; they live in a dedicated instance of coalgebraic modal logic [Pat04, Sch08], with binary modalities extracted from the type functor in a systematic way. We have also introduced the notion of a *cancellative* functor, and we have shown that for such functors, the construction of formulae and, more importantly, the logic employed can be simplified, requiring only unary modalities and conjunction. We have also discussed how distinguishing formulae can be translated into a more familiar domain-specific syntax (e.g. Hennessy-Milner logic for transition systems).

There is a proof-of-concept implementation of the certificate construction, based on an earlier open source implementation of the underlying partition refinement algorithm [DMSW19, WDMS21].

In partition refinement, blocks are successively refined in a top-down manner, and this is reflected by the use of conjunction in distinguishing formulae. Alternatively, bisimilarity may be computed bottom-up, as in a recent partition *aggregation* algorithm [BC20]. It is an interesting point for future investigation whether this algorithm can be extended to compute distinguishing formulae, which would likely be of a rather different shape than those computed via partition refinement.

References

[ABDG14] Abel Armas-Cervantes, Paolo Baldan, Marlon Dumas, and Luciano García-Bañuelos. Behavioral comparison of process models based on canonically reduced event structures. In *Business Process Management*, pages 267–282. Springer, 2014.

[ABLM12] Jiří Adámek, Nathan Bowler, Paul B. Levy, and Stefan Milius. Coproducts of monads on Set. In *Proc. 27th Annual Symposium on Logic in Computer Science (LICS'12)*, pages 45–54. IEEE Computer Society, 2012.

[AGD13]   Abel Armas-Cervantes, Luciano García-Bañuelos, and Marlon Dumas. Event structures as a foundation for process model differencing, part 1: Acyclic processes. In *Web Services and Formal Methods*, pages 69–86. Springer, 2013.

[AI01]   Micah Adler and Neil Immerman. An *n!* lower bound on formula size. In *LICS 2001*, pages 197–206. IEEE Computer Society, 2001.

[AI03]   Micah Adler and Neil Immerman. An *n!* lower bound on formula size. *ACM Trans. Comput. Log.*, 4(3):296–314, 2003.

[AM89]   Peter Aczel and Nax Mendler. A final coalgebra theorem. In *Proc. Category Theory and Computer Science (CTCS)*, volume 389 of *LNCS*, pages 357–365. Springer, 1989.

[Bar93]   Michael Barr. Terminal coalgebras in well-founded set theory. *Theoretical Computer Science*, 114(2):299–315, 1993.

[BC20]   Johanna Björklund and Loek Cleophas. Aggregation-based minimization of finite state automata. *Acta Informatica*, 2020.

[BCSS98] Marco Bernardo, Rance Cleaveland, Steve Sims, and W. Stewart. TwoTowers: A tool integrating functional and performance analysis of concurrent systems. In *Formal Description Techniques and Protocol Specification, Testing and Verification, FORTE / PSTV 1998*, volume 135 of *IFIP Conference Proceedings*, pages 457–467. Kluwer, 1998.

[Ber04]   Marco Bernardo. TwoTowers 5.1 user manual, 2004.

[BKR19]   Simone Barlocco, Clemens Kupke, and Jurriaan Rot. Coalgebra learning via duality. In Mikolaj Bojanczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures, FOSSACS 2019*, volume 11425 of *LNCS*, pages 62–79. Springer, 2019. `doi:10.1007/978-3-030-17127-8\_4`.

[BM19]   Marco Bernardo and Marino Miculan. Constructive logical characterizations of bisimilarity for reactive probabilistic systems. *Theoretical Computer Science*, 764:80 – 99, 2019. Selected papers of ICTCS 2016.

[BSdV04] Falk Bartels, Ana Sokolova, and Erik de Vink. A hierarchy of probabilistic system types. *Theoret. Comput. Sci.*, 327:3–22, 2004.

[CC95]   Ufuk Celikkan and Rance Cleaveland. Generating diagnostic information for behavioral preorders. *Distributed Computing*, 9(2):61–75, 1995. `doi:10.1007/s004460050010`.

[Cle91]   Rance Cleaveland. On automatically explaining bisimulation inequivalence. In *Computer-Aided Verification*, pages 364–372. Springer, 1991. `doi:10.1007/BFb0023750`.

[CLW15]   Sjoerd Cranen, Bas Luttik, and Tim A. C. Willemse. Evidence for Fixpoint Logic. In *24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*, volume 41 of *LIPIcs*, pages 78–93. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015. `doi:10.4230/LIPIcs.CSL.2015.78`.

[DEP98]   J. Desharnais, A. Edalat, and P. Panangaden. A logical characterization of bisimulation for labeled markov processes. In *Proceedings. Thirteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No.98CB36226)*, pages 478–487, 1998. `doi:10.1109/LICS.1998.705681`.

[DEP02]   Josée Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for labelled markov processes. *Information and Computation*, 179(2):163–193, 2002. `doi:10.1006/inco.2001.2962`.

[Dij08]   Remco Dijkman. Diagnosing differences between business process models. In *Business Process Management*, pages 261–277, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[DMSW17]   Ulrich Dorsch, Stefan Milius, Lutz Schröder, and Thorsten Wißmann. Efficient coalgebraic partition refinement. In *Proc. 28th International Conference on Concurrency Theory (CONCUR 2017)*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. `doi:10.4230/LIPIcs.CONCUR.2017.32`.

[DMSW18]   Ulrich Dorsch, Stefan Milius, Lutz Schröder, and Thorsten Wißmann. Predicate liftings and functor presentations in coalgebraic expression languages. In *Coalgebraic Methods in Computer Science, CMCS 2018*, volume 11202 of *LNCS*, pages 56–77. Springer, 2018.

[DMSW19]   Hans-Peter Deifel, Stefan Milius, Lutz Schröder, and Thorsten Wißmann. Generic partition refinement and weighted tree automata. In *Formal Methods – The Next 30 Years, Proc. 3rd World Congress on Formal Methods (FM 2019)*, volume 11800 of *LNCS*, pages 280–297. Springer, 10 2019.

[Dob09]   Ernst-Erich Doberkat. *Stochastic Coalgebraic Logic*. Springer, 2009.

[FG10]   Santiago Figueira and Daniel Gorín. On the size of shortest modal descriptions. In *Advances in Modal Logic 8, papers from the eighth conference on "Advances in Modal Logic," held in Moscow, Russia, 24-27 August 2010*, pages 120–139. College Publications, 2010.

[FvIK13]   Tim French, Wiebe van der Hoek, Petar Iliev, and Barteld Kooi. On the succinctness of some modal logics. *Artificial Intelligence*, 197:56 – 85, 2013.

[GJ21]   Herman Geuvers and Bart Jacobs. Relating apartness and bisimulation. *Logical Methods in Computer Science*, Volume 17, Issue 3, July 2021. `doi:10.46298/lmcs-17(3:15)2021`.

[Gri73]   David Gries. Describing an algorithm by Hopcroft. *Acta Informatica*, 2:97–109, 1973.

[GS01]   H. Peter Gumm and Tobias Schröder. Monoid-labeled transition systems. In *Coalgebraic Methods in Computer Science, CMCS 2001*, volume 44(1) of *ENTCS*, pages 185–204. Elsevier, 2001.

[GS13]   Daniel Gorín and Lutz Schröder. Simulations and bisimulations for coalgebraic modal logics. In *Algebra and Coalgebra in Computer Science - 5th International Conference, CALCO 2013*, volume 8089 of *LNCS*, pages 253–266. Springer, 2013.

[Gum05]   H.Peter Gumm. From T-coalgebras to filter structures and transition systems. In *Algebra and Coalgebra in Computer Science*, volume 3629 of *LNCS*, pages 194–212. Springer, 2005.

[Hop71]   John Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.

[Kli05]   Bartek Klin. The least fibred lifting and the expressivity of coalgebraic modal logic. In *Algebra and Coalgebra in Computer Science, CALCO 2005*, volume 3629 of *LNCS*, pages 247–262. Springer, 2005.

[KMMS20]   Barbara König, Christina Mika-Michalski, and Lutz Schröder. Explaining non-bisimilarity in a coalgebraic approach: Games and distinguishing formulas. In *Coalgebraic Methods in Computer Science*, pages 133–154. Springer, 2020.

[Knu01]   Timo Knuutila. Re-describing an algorithm by Hopcroft. *Theor. Comput. Sci.*, 250:333 – 363, 2001.

[KS83]   Paris C. Kanellakis and Scott A. Smolka. Ccs expressions, finite state processes, and three problems of equivalence. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, PODC '83, pages 228–240. ACM, 1983.

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

[LAS91]   Kim Guldstrand Larsen and Arne Arne Skou. Bisimulation through probabilistic testing. *Inform. Comput.*, 94(1):1–28, 1991.

[Mil89]   R. Milner. *Communication and Concurrency*. International series in computer science. Prentice-Hall, 1989.

[MV15]   Johannes Marti and Yde Venema. Lax extensions of coalgebra functors and their logic. *J. Comput. Syst. Sci.*, 81(5):880–900, 2015.

[Par81]   D. Park. Concurrency and automata on infinite sequences. In *Proceedings of 5th GI-Conference on Theoretical Computer Science*, volume 104 of *LNCS*, pages 167–183, 1981.

[Pat03]   Dirk Pattinson. Coalgebraic modal logic: soundness, completeness and decidability of local consequence. *Theoretical Computer Science*, 309(1):177 – 193, 2003. `doi:10.1016/S0304-3975(03)00201-9`.

[Pat04]   Dirk Pattinson. Expressive logics for coalgebras via terminal sequence induction. *Notre Dame J. Formal Log.*, 45(1):19–33, 2004.

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

[RdV99]   Jan Rutten and Erik de Vink. Bisimulation for probabilistic transition systems: a coalgebraic approach. *Theoret. Comput. Sci.*, 221:271–293, 1999.

[Rut00]   Jan Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249:3–80, 2000.

[Sch01]   Tobias Schröder. *Coalgebren und Funktoren*. PhD thesis, Philipps-Universität Marburg, 2001. `doi:10.17192/z2001.0205`.

[Sch08]   Lutz Schröder. Expressivity of coalgebraic modal logic: The limits and beyond. *Theor. Comput. Sci.*, 390(2-3):230–247, 2008. `doi:10.1016/j.tcs.2007.09.023`.

[SP11]   Lutz Schröder and Dirk Pattinson. Modular algorithms for heterogeneous modal logics via multi-sorted coalgebra. *Math. Struct. Comput. Sci.*, 21(2):235–266, 2011. `doi:10.1017/S0960129510000563`.

[Trn71]   Věra Trnková. On a descriptive classification of set functors I. *Commentationes Mathematicae Universitatis Carolinae*, 12(1):143–174, 1971.

[VF10]   Antti Valmari and Giuliana Franceschinis. Simple $\mathcal{O}(m \log n)$ time Markov chain lumping. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2010*, volume 6015 of *LNCS*, pages 38–52. Springer, 2010.

[VGRW22]   Frits Vaandrager, Bharat Garhewal, Jurriaan Rot, and Thorsten Wißmann. A new approach for active automata learning based on apartness. In *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022*, Lecture Notes in Computer Science. Springer, 04 2022.

[VL08]   Antti Valmari and Petri Lehtinen. Efficient minimization of dfas with partial transition. In *Theoretical Aspects of Computer Science, STACS 2008*, volume 1 of *LIPIcs*, pages 645–656. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2008.

[WDMS20]   Thorsten Wißmann, Ulrich Dorsch, Stefan Milius, and Lutz Schröder. Efficient and Modular Coalgebraic Partition Refinement. *Logical Methods in Computer Science*, Volume 16, Issue 1, January 2020. `doi:10.23638/LMCS-16(1:8)2020`.

[WDMS21]   Thorsten Wißmann, Hans-Peter Deifel, Stefan Milius, and Lutz Schröder. From generic partition refinement to weighted tree automata minimization. *Form. Asp. Comput.*, 33:695–727, 2021.

[WMS21]   Thorsten Wißmann, Stefan Milius, and Lutz Schröder. Explaining behavioural inequivalence generically in quasilinear time. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021*, volume 203 of *LIPIcs*, pages 32:1–32:18, 2021. `doi:10.4230/LIPIcs.CONCUR.2021.32`.

Appendix A. Worst Case Tree Size of Certificates for Transition Systems

In the following, we provide a direct proof that the states of Example 6.1 only admit distinguishing formulae of at least exponential size. As the worst case size of a certificate, we will derive the minimum size of a certificate of x_i. The crucial argument here is that there is no certificate of x_i with only one top-level modality. To this end, we look at all formulae with only one top-level modality:

$$\Diamond\varphi \qquad \text{and} \qquad \neg\Diamond\varphi \tag{A.1}$$

for some formula φ (note that these patterns also subsume $\Box$-formulae). Given the semantics of φ in the layer L_i, the semantics of the formulae from (A.1) is clear:

$[\![\varphi]\!] \cap L_i$	$[\![\Diamond\varphi]\!] \cap L_{i+1}$	$[\![\neg\Diamond\varphi]\!] \cap L_{i+1}$
$\emptyset$	$\emptyset$	L_{i+1}
$\{x_i\}$	$\{x_{i+1}, z_{i+1}\}$	$\{y_{i+1}\}$
$\{y_i\}$	$\{x_{i+1}, y_{i+1}\}$	$\{z_{i+1}\}$
$\{z_i\}$	L_{i+1}	$\emptyset$
$\{x_i, y_i\}$	L_{i+1}	$\emptyset$
$\{x_i, z_i\}$	L_{i+1}	$\emptyset$
$\{y_i, z_i\}$	L_{i+1}	$\emptyset$
$\{x_i, y_i, z_i\}$	L_{i+1}	$\emptyset$

Table 3. Semantics of modalities in Example 6.1

Of course, $[\![\neg\Diamond\varphi]\!]$ is just the complement of $[\![\Diamond\varphi]\!]$. Since every state of L_{i+1} has a transition to z_i we have that $z_i \in [\![\varphi]\!] \cap L_i$ implies $[\![\Diamond\varphi]\!] = L_{i+1}$.

Lemma A.1. *All states of (C, c) in Example 6.1 have different behaviour.*

Proof. We show by induction on i that all states of $L_0 \cup \cdots \cup L_i$ have different behaviour.

- All states of L_0 have different behaviour because
  - $x_0 \not\sim y_0$ because $x_0 \to y_0$ but $y_0 \not\to$.
  - $z_0 \not\sim y_0$ because $z_0 \to x_0$ but $y_0 \not\to$.
  - $x_0 \not\sim z_0$ because $z_0 \to x_0$ and $z_0 \to y_0$ but $x_0 \not\sim y_0$.

- Assume that all elements of $L_0 \cup \cdots \cup L_i$ have different behaviour. Every element of L_{i+1} is behaviourally different from every state in $L_0 \cup \cdots \cup L_i$ because every state in L_{i+1} has a transition to z_i, but no state in $L_0 \cup \cdots \cup L_i$ has and (by the induction hypothesis) there is no other state in $L_0 \cup \cdots \cup L_i$ that is bisimilar to z_i. Since all state of L_i have different behaviour, the same holds for L_{i+1}:
  - $x_{i+1} \not\sim y_{i+1}$ because $x_{i+1} \to x_i$ but $y_{i+1} \not\to x_i$.
  - $x_{i+1} \not\sim z_{i+1}$ because $x_{i+1} \to y_i$ but $z_{i+1} \not\to y_i$.
  - $y_{i+1} \not\sim z_{i+1}$ because $y_{i+1} \to y_i$ but $z_{i+1} \not\to y_i$.

  Hence, all states of $L_0 \cup \cdots \cup L_i \cup L_{i+1}$ have different behaviour. $\qquad\square$

Lemma A.2. *Let φ be a formula, and let $a, b \in C$ such that $a \in [\![\varphi]\!]$ and $b \notin [\![\varphi]\!]$. Then φ has a subformula α such that*

(1) *α has one of the forms $\Diamond\beta$ or $\neg\Diamond\beta$;*

(2) *α appears at the top level, i.e. outside the scope of any modality, in φ;*

(3) *$a \in [\![\alpha]\!]$ and $b \notin [\![\alpha]\!]$.*

Proof. We may assume without loss of generality that φ is in disjunctive normal form on the top level, i.e. φ is a disjunction of conjunctions (called *conjunctive clauses*) of possibly negated $\Diamond$-formulae, since transforming φ into such a disjunctive normal form does not create new $\Diamond$-subformulae.

Since $a \in [\![\varphi]\!]$, the disjunctive normal form φ must contain a conjunctive clause ψ such that $a \in [\![\psi]\!]$. Since $b \notin [\![\varphi]\!]$, we necessarily have $b \notin [\![\psi]\!]$. Since $b \notin [\![\psi]\!]$, the conjunctive clause ψ must contain a conjunct α, of shape either $\Diamond\beta$ or $\neg\Diamond\beta$, such that $b \notin [\![\alpha]\!]$. Since $a \in [\![\psi]\!]$, we moreover necessarily have $a \in [\![\alpha]\!]$; this proves the claim. $\qquad\square$

Proposition A.3. *Let φ be a formula such that $[\![\varphi]\!] \cap L_{i+1} = \{x_{i+1}\}$. Then φ contains top-level subformulae (i.e. subformulae not in scope of a modality) $\Diamond\varphi_{x_i}$ and $\Diamond\varphi_{y_i}$ such that $[\![\varphi_{x_i}]\!] \cap L_i = \{x_i\}$ and $[\![\varphi_{y_i}]\!] \cap L_i = \{y_i\}$.*

Proof. By Lemma A.2, applied to $a = x_{i+1}$ and $b = y_{i+1}$, there must be a subformula α of φ such that $y_{i+1} \notin [\![\alpha]\!] \ni x_{i+1}$ and α has the form of either $\Diamond\psi$ or $\neg\Diamond\psi$. Since $x_{i+1}, y_{i+1} \in L_{i+1}$, we also have

$$y_{i+1} \notin [\![\alpha]\!] \cap L_{i+1} \ni x_{i+1}.$$

Looking at Table 3, we find that the only choice for α is $\Diamond\psi$ for some formula ψ satisfying $[\![\psi]\!] \cap L_i = \{x_i\}$; so ψ may serve as the desired φ_{x_i}. Similarly, since $z_{i+1} \notin [\![\varphi]\!] \ni x_{i+1}$, the formula φ must have a subformula α' of the form either $\Diamond\chi$ or $\neg\Diamond\chi$ such that

$$z_{i+1} \notin [\![\alpha']\!] \cap L_{i+1} \ni x_{i+1},$$

which by Table 3 implies that $[\![\chi]\!] \cap L_i = \{y_i\}$, so that χ may serve as φ_{y_i}. $\qquad\square$

Proposition A.4. *Let φ be a formula such that $[\![\varphi]\!] \cap L_{i+1} = \{y_{i+1}\}$. Then φ has a top-level subformula of the form $\Diamond\varphi_{x_i}$ such that $[\![\varphi_{x_i}]\!] \cap L_i = \{x_i\}$.*

Proof. Similarly as in the proof of the previous proposition, since $x_{i+1} \notin [\![\varphi]\!] \ni y_{i+1}$, the formula φ must, by Lemma A.2, have a subformula α of the shape either $\Diamond\psi$ or $\neg\Diamond\psi$ such that

$$x_{i+1} \notin [\![\alpha']\!] \cap L_{i+1} \ni y_{i+1}.$$

Looking at Table 3 we find that this implies that α has the form $\neg\Diamond\psi$ for some formula ψ such that $[\![\psi]\!] \cap L_i = \{x_i\}$, so that ψ may serve as φ_{x_i}. $\qquad\square$

Proposition A.5. *Every certificate of x_n has at least the size $\mathsf{fib}(n)$.*

Proof. We show more generally that if φ is a formula such that $[\![\varphi]\!] \cap L_n = \{x_n\}$, then φ has size at least $\mathsf{fib}(n)$. We proceed by induction on n, with trivial base cases $n \in \{0, 1\}$.

In the inductive step, we assume the statement for n and $n+1$ and prove it for $n+2$. If φ is a formula such that $[\![\varphi]\!] \cap L_{n+2} = \{x_{n+2}\}$, then by Proposition A.3, φ has subformulae $\Diamond\varphi_{x_{n+1}}$ and $\Diamond\varphi_{y_{n+1}}$ such that

$$[\![\varphi_{x_{n+1}}]\!] \cap L_{n+1} = \{x_{n+1}\} \qquad \text{and} \qquad [\![\varphi_{y_{n+1}}]\!] \cap L_{n+1} = \{y_{n+1}\}.$$

By their semantics, $\varphi_{x_{n+1}}$ and $\varphi_{y_{n+1}}$ are necessarily different. Hence,

$$|\varphi_{x_{n+2}}| \geq |\varphi_{x_{n+1}}| + |\varphi_{y_{n+1}}|, \tag{A.2}$$

where $|-|$ denotes formula size. Now Proposition A.4 implies that $\varphi_{y_{n+1}}$ has a subformula of the form $\lozenge\varphi_{x_n}$ for some φ_{x_n} such that $[\![\varphi_{x_n}]\!] \cap L_n = \{x_n\}$. Hence,

$$|\varphi_{y_{n+1}}| \geq |\varphi_{x_n}|. \tag{A.3}$$

Thus, we derive

$$|\varphi_{x_{n+2}}| \overset{\text{(A.2)}}{\geq} |\varphi_{x_{n+1}}| + |\varphi_{y_{n+1}}| \overset{\text{(A.3)}}{\geq} |\varphi_{x_{n+1}}| + |\varphi_{x_n}| \qquad \text{for all } n \geq 0.$$

Thus, we have $|\varphi_{x_n}| \geq \mathsf{fib}(n)$ by induction. $\qquad\square$

Proposition A.6. *Every formula φ distinguishing x_{i+1} from y_{i+1} contains a subformula φ_{x_i} such that $[\![\varphi_{x_i}]\!] \cap L_i = \{x_i\}$. Hence, every such φ has at least size $\mathsf{fib}(i)$.*

Proof. Let φ distinguish x_{i+1} from y_{i+1}, which means that $x_{i+1} \in [\![\varphi]\!]$ and $y_{i+1} \notin [\![\varphi]\!]$. By Lemma A.2, φ has a subformula α such that $x_{i+1} \in [\![\alpha]\!]$, $y_{i+1} \notin [\![\alpha]\!]$, and α has one of the forms $\lozenge\psi$ or $\neg\lozenge\psi$ for some formula ψ. Looking at Table 3, we see that the only choice for α is $\lozenge\psi$ for some formula ψ satisfying $[\![\psi]\!] \cap L_i = \{x_i\}$, so that ψ may serve as φ_{x_i}. $\qquad\square$