Explaining Behavioural Inequivalence Generically in Quasilinear Time

Thorsten Wißmann
Radboud University, Nijmegen, The Netherlands

Stefan Milius
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Lutz Schröder
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 $\mathcal{O}(mn)$.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Program analysis

Keywords and phrases bisimulation, partition refinement, modal logic, distinguishing formulae, coalgebra

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.32

Related Version *Full Version*: https://arxiv.org/abs/2105.00669

Funding *Thorsten Wißmann*: Work forms part of the NWO TOP project 612.001.852 and the DFG-funded project COAX (MI 717/5-2).
Stefan Milius: Work forms part of the DFG-funded project CoMoC (MI 717/7-1).
Lutz Schröder: Work forms part of the DFG-funded project CoMoC (SCHR 1118/15-1).

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. This implies 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 explains the difference of the two states' behaviour and is thus usually called a *distinguishing formula* [13]. For example, in the transition system in Figure 1, the formula $\Box\Diamond\top$ distinguishes the states x and y because x satisfies $\Box\Diamond\top$ whereas y does not. Given two states in a finite transition system with n states and m transitions, the algorithm by Cleaveland [13] computes a distinguishing formula in time $\mathcal{O}(mn)$. The algorithm builds on the Kanellakis-Smolka partition refinement algorithm [28, 29], which computes the bisimilarity relation on a transition system within the same time bound.

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 32; pp. 32:1–32:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

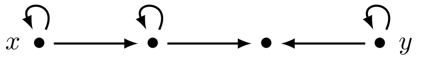


Figure 1 Example of a transition system.

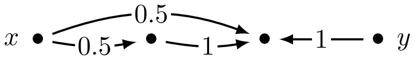


Figure 2 Example of a Markov chain.

Similar logical characterizations of bisimulation exist for other system types. For instance, Desharnais et al. [16, 17] characterize probabilistic bisimulation on (labelled) Markov chains, in the sense of Larsen and Skou [33] (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 [40]. 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 (transtion 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 [39, 42]. 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 in *quasilinear time*, and in fact *certificates* that uniquely describe behavioural equivalence classes in a system, in coalgebraic generality. We build on an existing efficient coalgebraic partition refinement algorithm [46], 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 [22]); even for labelled transition systems, we thus improve the previous best bound $\mathcal{O}(mn)$ [13] 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 (mentioned above) suffices for the construction of distinguishing formulae (but not certificates) [17] (see also [19]).

Related Work. Cleaveland's algorithm [13] for labelled transition systems is is based on Kanellakis and Smolka's partition refinement algorithm [29]. The coalgebraic partition refinement algorithm we employ [46] is instead related to the more efficient Paige-Tarjan algorithm [36]. König et al. [32] extract formulae from winning strategies in a bisimulation game in coalgebraic generality; their algorithm runs in $\mathcal{O}(n^4)$ and does not support negative transition weights. Characteristic formulae for behavioural equivalence classes taken across *all* models require the use of fixpoint logics [21]. The mentioned algorithm by Desharnais et al. for distinguishing formulae on labelled Markov processes [17, 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 [10] provide a similar algorithm for a logic with only disjunction. There are further generalizations along other axes, e.g. to behavioural preorders [12]. The TwoTowers tool set for the analysis of stochastic process algebras [8,9] 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 [14] or methods employed on business process models, based on model differences and event structures [5,6,18].

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

$$\ker(f) := \{(x, x') \in X \times X \mid f(x) = f(x')\}. \tag{1}$$

▶ **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 intersection $R \cap S$ of equivalence relations is again an equivalence relation. 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 [40] 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 $F(g \cdot f) = Fg \cdot Ff$. 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 $\mathcal{P}f = f[-]\colon \mathcal{P}X \to \mathcal{P}Y$ taking 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, 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 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

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

   whose elements we may understand as flat Σ-terms $\sigma(x_1, \dots, x_n)$ with variables from X. The action of T_Σ on maps $f\colon X \to Y$ is then given by $(T_\Sigma f)(\sigma(x_1, \dots, x_n)) = \sigma(f(x_1), \dots, f(x_n))$. For simplicity, we write σ (instead of in_σ) for the coproduct injections, and Σ in lieu of T_Σ for the signature functor. States in Σ-coalgebras describe possibly infinite Σ-trees.
4. For a commutative monoid $(M, +, 0)$, the *monoid-valued functor* $M^{(-)}$ [25] is given by

   $$M^{(X)} := \{\mu\colon X \to M \mid \mu(x) = 0 \text{ for all but finitely many } x \in X\} \tag{2}$$

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

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

   A coalgebra $c\colon C \to M^{(C)}$ is a finitely branching weighted transition system, where $c(x)(x') \in M$ 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

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

   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* [33] or *labelled Markov chains* [17], and coalgebras for $(\mathcal{D}((-) + 1) + 1)^A$ are *partial labelled Markov chains* [17]. 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 $d \cdot h = Fh \cdot c$. States x, y in an F-coalgebra (C,c) are *behaviourally equivalent* ($x \sim y$) if there exists a coalgebra morphism h such that $h(x) = h(y)$.

$$\begin{tikzcd} C \arrow[r, "c"] \arrow[d, "h"'] & FC \arrow[d, "Fh"] \\ D \arrow[r, "d"] & FD \end{tikzcd}$$

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 [1].
5. For weighted and probabilistic systems, coalgebraic behavioural equivalence instantiates to weighted and probabilistic bisimilarity, respectively [41, Cor. 4.7], [7, 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.
2. We may assume without loss of generality that a set functor F preserves injective maps [43] (see also [2, 8.1.12–17]), that is, Ff is injective whenever f is.

2.2 Coalgebraic Logics

We briefly review basic concepts of coalgebraic modal logic [38, 42]. 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 signature Λ of *modal operators* (with assigned arities). Then, *formulae* ϕ are generated by the grammar

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

▶ **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* $[\![\heartsuit]\!]$ [38, 42], i.e. a family of maps $[\![\heartsuit]\!]_X\colon (2^X)^n \to 2^{FX}$, one for every set X, such that the *naturality* condition

$$Ff^{-1}\big[[\![\heartsuit]\!]_Y(P_1, \dots, P_n)\big] = [\![\heartsuit]\!]_X(f^{-1}[P_1], \dots, f^{-1}[P_n]) \tag{3}$$

for all $f\colon X \to Y$ and all $P_1, \dots, P_n \in 2^X$ (for categorically-minded readers, $[\![\heartsuit]\!]$ is a natural transformation $(2^{(-)})^n \to 2^{F^{\mathsf{op}}}$); the idea being to lift given predicates on states to predicates on structured collections of states. Given these data, the *extension* of a formula ϕ in an F-coalgebra (C, c) is a predicate $[\![\phi]\!]_{(C,c)}$, or just $[\![\phi]\!]$, on C, recursively defined by

$$[\![\top]\!]_{(C,c)} = C \qquad [\![\phi \wedge \psi]\!]_{(C,c)} = [\![\phi]\!]_{(C,c)} \cap [\![\psi]\!]_{(C,c)} \qquad [\![\neg\phi]\!]_{(C,c)} = C \setminus [\![\phi]\!]_{(C,c)}$$
$$[\![\heartsuit(\phi_1, \dots, \phi_n)]\!]_{(C,c)} = c^{-1}\big[[\![\heartsuit]\!]_C\big([\![\phi_1]\!]_{(C,c)}, \dots, [\![\phi_n]\!]_{(C,c)}\big)\big] \qquad (\heartsuit/n \in \Lambda)$$

(where we apply set operations to predicates with the evident meaning). We say that a state $x \in C$ *satisfies* ϕ if $[\![\phi]\!](x) = 1$. Notice how the clause for modalities says that x satisfies $\heartsuit(\phi_1, \dots, \phi_n)$ iff $c(x)$ satisfies the predicate obtained by lifting the predicates $[\![\phi_1]\!], \dots, [\![\phi_n]\!]$ on C to a predicate on FC according to $[\![\heartsuit]\!]$.

▶ **Example 2.8.** Over $F = \mathcal{P}$, we interpret $\Diamond$ by the predicate lifting

$$[\![\Diamond]\!]_X \colon 2^X \to 2^{\mathcal{P}X}, \quad P \mapsto \{K \subseteq X \mid \exists x \in K \colon x \in P\} = \{K \subseteq X \mid K \cap P \neq \emptyset\},$$

The arising notion of satisfaction over $\mathcal{P}$-coalgebras (C, c) is precisely the standard one: $x \in [\![\Diamond\phi]\!]_{(C,c)}$ iff $y \in [\![\phi]\!]_{(C,c)}$ for some transition $x \to y$.

The naturality condition (3) of predicate liftings guarantees invariance of the logic under coalgebra morphisms, and hence under behavioural equivalence:

▶ **Proposition 2.9** (Adequacy [38, 42]). *Behaviourally equivalent states satisfy the same formulae: $x \sim y$ implies that for all formulae ϕ, we have $x \in [\![\phi]\!]$ iff $y \in [\![\phi]\!]$.*

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. [1, 24, 34, 40]), generalizing the Park-Milner bisimulation principle [35, 37]. It is less obvious how to certify behavioural *inequivalence* $x \not\sim y$, showing that such a morphism h does *not* exist. By Proposition 2.9, 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 [39, 42]. More generally, we consider separation of *sets* of states by formulae, following Cleaveland [13, 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 $X \subseteq [\![\phi]\!]$ and $Y \cap [\![\phi]\!] = \emptyset$. 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 $[\![\phi]\!] = X$.

Note that ϕ distinguishes X from Y iff $\neg\phi$ distinguishes Y from X. Certificates have also been referred to as *descriptions* [22]. 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 thus suffices to construct certificates for all behavioural equivalence classes, and our algorithm does just that. 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 $C/R_0, C/R_1, \dots$ of consecutively finer partitions (i.e. $R_{i+1} \subseteq R_i$) on the state space, where every *block* $B \in C/R_i$ is a union of behavioural equivalence classes, and the final partition is precisely $C/{\sim}$. Indeed, Cleaveland's algorithm for computing certificates on labelled transition systems [13] correspondingly extends Kanellakis and Smolka's partition refinement algorithm [28, 29], 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 [36], 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 all i), 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 $P_i \subsetneqq Q_i$, 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. If no such T is properly split, then $P_{i+1} = Q_{i+1}$, and the algorithm terminates. 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.
- In the refinement step, suppose that δ, β are certificates of $S \in C/P_i$ and $B \in C/Q_i$, respectively, where $S \subsetneqq B$. For every predecessor block T of B, the three blocks obtained by splitting T are distinguished (see Definition 3.1) as follows:

$$\text{(C1)}\quad \neg\Diamond(\beta\wedge\neg\delta), \qquad \text{(C2)}\quad \Diamond(\delta)\wedge\Diamond(\beta\wedge\neg\delta), \qquad \text{(C3)}\quad \neg\Diamond\delta. \tag{4}$$

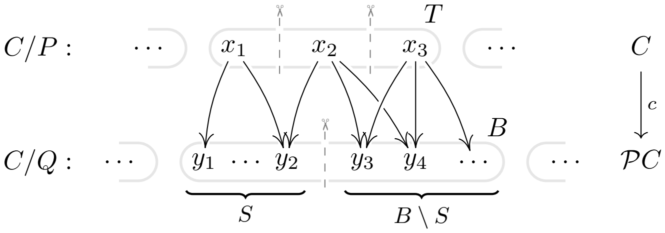


Figure 3 The refinement step as illustrated in [46, Figure 6].

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: $\phi \wedge \neg\Diamond(\beta \wedge \neg\delta)$, etc.

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 [44], and it has recently been generalized to coalgebras [20,46]. 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 $S \subsetneq B \in C/Q_i$, the three cases distinguish between the equivalence classes $[x]_{\mathcal{P}\chi_S^B \cdot c}$ for $x \in T$, where the map $\chi_S^B\colon C \to 3$ in the composite $\mathcal{P}\chi_S^B \cdot c\colon C \to \mathcal{P}3$ is defined by

$$\chi_S^B\colon C \to 3 \qquad \chi_S^B(x) = \begin{cases} 2 & \text{if } x \in S, \\ 1 & \text{if } x \in B \setminus S, \\ 0 & \text{if } x \in C \setminus B, \end{cases} \qquad \text{for sets } S \subseteq B \subseteq C. \tag{5}$$

Every case is a possible value of $t := \mathcal{P}\chi_S^B(c(x)) \in \mathcal{P}3$: (C1) $2 \in t \not\ni 1$, (C2) $2 \in t \ni 1$, and (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 of B iff $0 \in t$. However, because of the previous refinement steps performed by the algorithm, either all or no states states of T have an edge to $C \setminus B$ (a property called *stability* [36]), 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** [46, Alg. 4.9, (5.1)]. Given a coalgebra $c\colon C \to FC$, put

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

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 $S \subsetneq B$ and $2 \cdot |S| \leq |B|$

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

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

This algorithm formalizes the intuitive steps from Section 3.1. Again, two sequences of partitions P_1, 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; e.g. 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, and for $FX = 2\times X^A$, the value $F1(c(x)) \in F1 = 2 \times 1^A \cong 2$ 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 by $\{[x]_{F\chi_S^B(c(x))} \mid x \in T\}$. Again, this corresponds to the distinction of the three cases (C1)–(C3). For example, 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.

The following property of F is needed for correctness [46, Ex. 5.11].

▶ **Definition 3.3** [46]. A functor F is *zippable* if map

$$\langle F(A{+}!), F(! + B)\rangle\colon\ F(A + B) \longrightarrow F(A + 1) \times F(1 + B)$$

is injective for all sets A, B.

Intuitively, $t \in F(A + B)$ is a term in variables from A and B. If F is zippable, then t is uniquely determined by the two elements in $F(A + 1)$ and $F(1 + B)$ obtained by identifying all B- and all A-variables with $0 \in 1$, respectively. E.g. $FX = X^2$ 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 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 [46].

▶ **Remark 3.4.** To apply the algorithm to coalgebras for composites FG of zippable functors, e.g. $\mathcal{P}(A \times (-))$, there is a reduction [46, 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.

▶ **Theorem 3.5** [46, Thm 4.20, 5.20]. *On a finite coalgebra (C, c) for a zippable functor, Algorithm 3.2 terminates after $i \le |C|$ loop iterations, and the resulting partition identifies precisely the behaviourally equivalent states ($P_i = {\sim}$).*

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 known equivalence between n-ary predicate liftings and (in this case, singleton) subsets of $F(2^n)$ [42] (termed *tests* by Klin [30]):

▶ **Definition 3.6.** 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 the predicate lifting

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

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, respectively: $[\![\delta]\!] = S$ 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 3.7.** For $F = \mathcal{P}$, $\ulcorner\{0, 2\}\urcorner(\delta, \beta)$ 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 3.8.** *Given an F-coalgebra (C, c), $x \in C$, and formulae δ and β such that $[\![\delta]\!] \subseteq [\![\beta]\!] \subseteq C$, we have $x \in [\![\ulcorner t \urcorner(\delta, \beta)]\!]$ if and only if $F\chi^{[\![\beta]\!]}_{[\![\delta]\!]}(c(x)) = t$.*

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 3.9.** 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 3.10.** *For $x \in C$, $c \colon C \to FC$, and $t \in F1$, we have $x \in [\![\ulcorner t \urcorner]\!]$ if and only if $F!(c(x)) = t$.*

3.4 Algorithmic Construction of Certificates

The $F3$-modalities introduced above (Definition 3.6) 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, $\beta_i(B)$, $\delta_i(S)$ denote corresponding certificates: we will have

$$\forall B \in X/Q_i \colon [\![\beta_i(B)]\!] = B \qquad \text{and} \qquad \forall S \in X/P_i \colon [\![\delta_i(S)]\!] = S. \tag{6}$$

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

▶ **Algorithm 3.11.** We extend Algorithm 3.2 by the following. Initially, put

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

In the i-th iteration, extend steps (A2) and (A3) by the following assignments:

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

(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. Since the partition C/P_0 distinguishes by the "output" (e.g. final vs. non-final states), the certificate of $[x]_{P_0}$ specifies what $F!(c(x)) \in F1$ is (Lemma 3.10).

In the i-th iteration of the main loop, we have certificates $\delta_i(S)$ and $\beta_i(B)$ for $S \subsetneqq B$ in step (A1) satisfying (6) available from the previous iterations. In (A'2), the Boolean connectives describe how B is split into S and $B \setminus S$. 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 3.8.

▶ **Remark 3.12.** In step (A'2), $\beta_{i+1}(D)$ can be simplified to be no larger than $\delta_i(S)$. To see this, note that for $S \subseteq B \subseteq C$, $S \in X/P_i$, and $B \in X/Q_i$, 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 $\beta_{i+1}(D)$.

▶ **Theorem 3.13.** *For zippable F, Algorithm 3.11 is correct, i.e.* (6) *holds for all i. Thus, upon termination δ_i assigns certificates to each block of $C/{\sim} = C/P_i$.*

▶ **Corollary 3.14** (Hennessy-Milner)**.** *For zippable F, states x, y in a finite F-coalgebra are behaviourally equivalent iff they agree on all $F3$-modal formulae.*

▶ **Remark 3.15.** A smaller formula distinguishing a state x from a state y can be extracted from the certificates in time $\mathcal{O}(|C|)$. It is the leftmost conjunct that is different in the respective certificates of x and y. 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 $F3$.

3.5 Complexity Analysis

The operations introduced by Algorithm 3.11 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 [45], 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 $S \subsetneqq B$ contains the subformulas $\delta_i(S)$ and $\beta_i(B)$. Therefore, it is advantageous to represent all certificates constructed as one directed acyclic graph (dag) with nodes labelled by modal operators and conjunction and having precisely two outgoing edges. Moreover, edges have a binary flag indicating whether they represent negation $\neg$. Initially, there is only one node representing $\top$, and the operations of Algorithm 3.11 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 [26].

▶ **Definition 3.16.** *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 3.17.** *For a coalgebra with n states and m transitions, the formula dag constructed by Algorithm 3.11 has size $\mathcal{O}(m \cdot \log n + n)$ and height at most $n + 1$.*

▶ **Theorem 3.18.** *Algorithm 3.11 adds only constant run time overhead, thus it has the same run time as Algorithm 3.2 (regardless of the optimization from Remark 3.12).*

For a tighter run time analysis of the underlying partition refinement algorithm, one additionally requires that F is equipped with a *refinement interface* [46, 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 3.16 for all functors considered. All zippable functors we consider here have refinement interfaces [15, 46]. 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 recovers the run time of the Paige-Tarjan algorithm.

▶ **Remark 3.19.** 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 [31, 45]; the other partition C/Q_i is only represented implicitly in terms of a queue of blocks $S \subsetneqq B$ witnessing $P_i \subsetneqq Q_i$, requiring additional care when splitting blocks in the queue [44, Fig. 3]. Moreover, grouping the elements of a block by $F3$ involves the consideration of a *possible majority candidate* [44].

▶ **Theorem 3.20.** *On a coalgebra with n states and m transitions for a zippable set functor with a refinement interface with factor $p(c)$, Algorithm 3.11 runs in time $\mathcal{O}((m+n) \cdot \log n \cdot p(c))$.*

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 [27] and Valmari and Franceschinis' algorithm for weighted systems (e.g. Markov chains) [44] 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 (5) and note that $\chi_{\{1,2\}} \cdot \chi_S^B = \chi_B$ and $\chi_{\{2\}} \cdot \chi_S^B = \chi_S$, 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 $s, b_1, b_2 \in M$.

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

Hence, $\mathbb{R}^{(-)}$ is cancellative, but $\mathcal{P}_\mathsf{f}$ is not. Moreover, all signature functors are cancellative:

▶ **Proposition 4.3.** *The class of cancellative functors contains the all constant functors as well as the identity functor, and it is closed under subfunctors, products, and coproducts.*

For example, $\mathcal{D}$ is cancellative, but $\mathcal{P}$ is not because of its subfunctor $\mathcal{P}_\mathsf{f}$.

▶ **Remark 4.4.** Cancellative functors are neither closed under quotients nor under composition. Zippability and cancellativity are independent properties. Zippability in conjunction with cancellativity implies m-zippability for all $m \in \mathbb{N}$, the m-ary variant [32] of zippability.

▶ **Theorem 4.5.** *If F is a cancellative functor, $\ulcorner F\chi_S^B(c(x))\urcorner(\delta_i(S), \beta_i(B))$ in Algorithm 3.11 can be replaced with $\ulcorner F\chi_S^C(c(x))\urcorner(\delta_i(S), \top)$. Then, the algorithm still correctly computes certificates in the given F-coalgebra (C, c).*

Note that in this optimized algorithm, the computation of β can be omitted because it is not used anymore. Hence, the resulting formulae only involve $\wedge$, $\top$, and modalities from the set $F3$ (with the second parameter fixed to $\top$). These modalities are equivalently unary modalities induced by elements of $F2$, which we term *$F2$-modalities*; hence, the corresponding Hennessy-Milner Theorem (Corollary 3.14) adapts to $F2$ for cancellative functors, as follows:

▶ **Corollary 4.6.** *For zippable and cancellative F, states in an F-coalgebra are behaviourally equivalent iff they agree on modal formulae built using $\top$, $\wedge$, and unary $F2$-modalities.*

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. $\Box$ 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 in general 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 $\Box = \neg\Diamond\neg$.

▶ **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

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

(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 axiom 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 3.7):

▶ **Example 5.3.**

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 $\tau_{\{0\}} = \Diamond\top$ and $\tau_\emptyset = \neg\Diamond\top$. For $t \in \mathcal{P}3$, we put

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

   The certificates obtained via this translation are precisely the ones generated in the example using the Paige-Tarjan algorithm, cf. (4), with ρ in lieu of $\beta \wedge \neg\delta$.

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

$$\begin{aligned} [\![\sigma]\!]_X &= \{\sigma(x_1,\dots,x_n) \mid x_1,\dots,x_n \in X\} \subseteq \Sigma X, \\ [\![\langle{=}I\rangle]\!](S) &= \{\sigma(x_1,\dots,x_n) \in \Sigma X \mid \forall i \in \mathbb{N}\colon i \in I \leftrightarrow (1 \le i \le n \;\wedge\; x_i \in S)\}. \end{aligned}$$

   Intuitively, $\langle{=}I\rangle\,\phi$ states that the ith successor satisfies ϕ iff $i \in I$. We then have a domain-specific interpretation (τ,λ) given by $\tau_o := \sigma$ for $o = \sigma(0,\dots,0) \in \Sigma1$ and $\lambda_t(\delta,\rho) := \langle{=}I\rangle\delta$ for $t = \sigma(x_1,\dots,x_n) \in \Sigma3$ and $I = \{i \in \{1,\dots,n\} \mid x_i = 2\}$.

3. For a monoid-valued functor $M^{(-)}$, take $\Lambda = \{\langle{=}m\rangle/1 \mid m \in M\}$, interpreted by the predicate liftings $[\![\langle{=}m\rangle]\!]_X\colon 2^X \to 2^{M^{(X)}}$ 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 $\tau_o = \langle{=}o(0)\rangle\,\top$ for $o \in M^{(1)}$ and $\lambda_t(\delta,\rho) = \langle{=}t(2)\rangle\,\delta \wedge \langle{=}t(1)\rangle\,\rho$ for $t \in M^{(3)}$. 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. [17]. This gives rise to the interpretation:

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

Given a domain-specific interpretation (τ,λ) for a modal signature Λ for the functor F, we can postprocess certificates ϕ produced by Algorithm 3.11 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 Remark 3.12; the latter conjunction has essentially the same size as $T(\delta)$.

▶ **Proposition 5.4.** *For every certificate ϕ of a behavioural equivalence class of a given coalgebra produced by either Algorithm 3.11 or its optimization (Theorem 4.5), $T(\phi)$ is also a certificate for that class.*

Thus, the domain-specific modal signatures also inherit a Hennessy-Milner Theorem.

▶ **Example 5.5.** For labelled Markov chains ($FX = (\mathcal{D}X + 1)^A$) and the interpretation via the modalities $\langle a \rangle_p$ (Example 5.3.4), this yields certificates (thus in particular distinguishing formulae) in run time $\mathcal{O}(|A| \cdot m \cdot \log n)$, with the same bound on formula size. Desharnais et al. describe an algorithm [17, Fig. 4] that computes distinguishing formulae in the negation-free fragment of the same logic (they note also that this fragment does not suffice for certificates). They do not provide a run-time analysis, but the nested loop structure indicates that the asymptotic complexity is roughly $|A| \cdot n^4$.

6 Worst Case Tree Size of Certificates

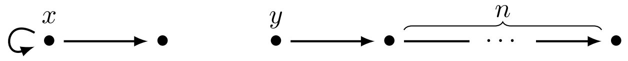
In the complexity analysis (Section 3.5), we have seen that certificates – and thus also distinguishing formulae – have dag size $\mathcal{O}(m \cdot \log n + n)$ on input coalgebras with n states and m transitions. However, when formulae are written in the usual linear way, multiple occurrences of the 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*.

Figueira and Gorín [22] show that exponential tree size is inevitable even for distinguishing formulae. The proof is based on winning strategies in bisimulation games, a technique that is also applied in other results on lower bounds on formula size [3, 4, 23].

▶ **Open Problem 6.1.** *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.

▶ **Remark 6.2.** Cleaveland [13, 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 *minimal* 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



for $n \in \mathbb{N}$,

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 $\Diamond\neg\Diamond\top$.

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 coalgebraic partition refinement [46], 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 [39, 42], with binary modalities extracted from the type functor in a systematic way. We have shown that for *cancellative* functors, the construction of formulae and, more importantly, the logic 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 an open source implementation of the underlying partition refinement algorithm [15], which may serve as a basis for a future implementation.

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 [11]. 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

1 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.

2 Jiří Adámek, Stefan Milius, and Lawrence S. Moss. Initial algebras, terminal coalgebras, and the theory of fixed points of functors. draft book, available online at `https://www8.cs.fau.de/ext/milius/publications/files/CoalgebraBook.pdf`, 2021.

3 Micah Adler and Neil Immerman. An *n!* lower bound on formula size. In *LICS 2001*, pages 197–206. IEEE Computer Society, 2001. `doi:10.1109/LICS.2001.932497`.

4 Micah Adler and Neil Immerman. An *n!* lower bound on formula size. *ACM Trans. Comput. Log.*, 4(3):296–314, 2003. `doi:10.1145/772062.772064`.

5 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.

6 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.

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

8 Marco Bernardo. TwoTowers 5.1 user manual, 2004.

9 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.

10 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.

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

12 Ufuk Celikkan and Rance Cleaveland. Generating diagnostic information for behavioral preorders. *Distributed Computing*, 9(2):61–75, 1995.

13 Rance Cleaveland. On automatically explaining bisimulation inequivalence. In *Computer-Aided Verification*, pages 364–372. Springer, 1991.

14 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`.

15 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, October 2019.

16 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.

17 Josée Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for labelled markov processes. *Information and Computation*, 179(2):163–193, 2002.

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

19 Ernst-Erich Doberkat. *Stochastic Coalgebraic Logic.* Springer, 2009. `doi:10.1007/978-3-642-02995-0`.

20 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`.

21 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. `doi:10.1007/978-3-030-00389-0_5`.

22 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. URL: `http://www.aiml.net/volumes/volume8/Figueira-Gorin.pdf`.

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

24 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. `doi:10.1007/978-3-642-40206-7_19`.

25 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. `doi:10.1016/S1571-0661(04)80908-3`.

26 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.

27 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.

28 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.

29 Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Inf. Comput.*, 86(1):43–68, 1990. `doi:10.1016/0890-5401(90)90025-D`.

30 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. `doi:10.1007/11548133_16`.

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

32 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.

33 Kim Guldstrand Larsen and Arne Arne Skou. Bisimulation through probabilistic testing. *Inform. Comput.*, 94(1):1–28, 1991. doi:10.1016/0890-5401(91)90030-6.

34 Johannes Marti and Yde Venema. Lax extensions of coalgebra functors and their logic. *J. Comput. Syst. Sci.*, 81(5):880–900, 2015. doi:10.1016/j.jcss.2014.12.006.

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

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

37 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.

38 Dirk Pattinson. Coalgebraic modal logic: soundness, completeness and decidability of local consequence. *Theoretical Computer Science*, 309(1):177–193, 2003.

39 Dirk Pattinson. Expressive logics for coalgebras via terminal sequence induction. *Notre Dame J. Formal Log.*, 45(1):19–33, 2004. doi:10.1305/ndjfl/1094155277.

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

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

42 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.

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

44 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.

45 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. doi:10.4230/LIPIcs.STACS.2008.1328.

46 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.