View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by University of Strathclyde Institutional Repository

Angluin Learning via Logic*

Simone Barlocco and Clemens Kupke

Computer & Information Sciences,
University of Strathclyde,
Glasgow, Scotland
simone.barlocco@strath.ac.uk,
clemens.kupke@strath.ac.uk

Abstract. In this paper we will provide a fresh take on Dana Angluin's algorithm for learning using ideas from coalgebraic modal logic. Our work opens up possibilities for applications of tools & techniques from modal logic to automata learning and vice versa. As main technical result we obtain a generalisation of Angluin's original algorithm from DFAs to coalgebras for an arbitrary finitary set functor T in the following sense: given a (possibly infinite) pointed T-coalgebra that we assume to be regular (i.e. having an equivalent finite representation) we can learn its finite representation by asking (i) "logical queries" (corresponding to membership queries) and (ii) making conjectures to which the teacher has to reply with a counterexample. This covers (a known variant) of the original L* algorithm and the learning of Mealy/Moore machines. Other examples are bisimulation quotients of (probabilistic) transition systems.

Keywords: automata learning, coalgebra, modal logic

1 Introduction

Coalgebra studies "generated behaviour" that can be observed when interacting with a system. A lot of progress has been made thus far to formalise behaviour and to create languages that allow to specify and reason about it (cf. e.g. [1]). Intriguingly little, however, has been done to formalise the process of making observations and of using these observations to learn how a system works.

This has changed thanks to a series of recent work [2–5] but the connections between coalgebra & learning are still far from being completely understood. In this paper we will describe one such connection between the above mentioned coalgebraic specification languages (aka coalgebraic modal logics) and the well-known L* algorithm [6] for learning deterministic finite automata (DFA).

This algorithm constructs a minimal DFA accepting a (at the beginning unknown) regular language by asking a teacher membership queries of the form "is word w in the language?" and by making conjectures for what the language/DFA is, to which the teacher replies with a counterexample in case the conjecture is false. A central role in the algorithm is played by so-called tables that essentially

* Supported by EPSRC grant EP/N015843/1.

consist of two sets of finite words S and E of which the first corresponds to the states of the constructed DFA and the second set corresponds to observations or tests that we are performing on states.

The use of tests shows that the connection to logic is not at all surprising. A bit more surprising was for us the observation that closed & consistent tables can be best understood using the notion of a filtration from modal logic. We will not discuss this observation in detail - instead we will describe a generalisation of the L^* algorithm to coalgebras that was made possible by it. Our "L^{co} algorithm" allows - in principle - the learning of regular coalgebras for an arbitrary finitary set functor. This generalisation of L^* is the main contribution of this paper.

1.1 What the algorithm learns

The classical L^* algorithm looks very much like a bottom up procedure. The starting point of the algorithm are two singleton sets that grow step-by-step until the desired DFA has been learned. It is instructive, however, to think of the algorithm as follows: The regular language $\mathcal{L} \subseteq \Sigma^*$ that we intend to learn can be thought as represented by the infinite DFA

$$\langle o, \delta \rangle : \Sigma^* \to 2 \times (\Sigma^*)^{\Sigma}$$

where $o(w) = 1$ iff $w \in \mathcal{L}$ and $\delta(w)(a) = w \cdot a$ for all $w \in \Sigma^*$, $a \in \Sigma$. The assumption that $\mathcal{L}$ is a regular language can be rephrased by stating that the pointed coalgebra $(\Sigma^*, \langle o, \delta \rangle, \lambda)$ is behaviourally equivalent to a finite well-pointed coalgebra [7]. The aim of the algorithm is to learn this finite well-pointed coalgebra using queries that can be asked concerning the given infinite coalgebra. Our generalisation of Angluin's algorithm looks thus as follows: We are given a (possibly infinite) pointed T-coalgebra (X, γ, x) (corresponding to the language $\mathcal{L}$) and we assume that (X, γ, x) is *regular*, i.e. behaviourally equivalent to a finite well-pointed T-coalgebra (Y, δ, y). The goal of the algorithm is to learn (Y, δ, y).

1.2 Means to learn

The central device for learning in our setting is logic: we assume that we are provided with an expressive modal language that allows to characterise coalgebras up-to behavioural equivalence. The learner is able to ask two types of queries:

- Logical queries of the form *"is formula φ true at some state $x' \in X$?"*
- Conjectures of the form *"is this the correct well-pointed T-coalgebra?"*

Logical queries are answered truthfully with *Yes* or *No* by the teacher, conjectures are either confirmed or the teacher provides a counterexample in the shape of a formula $\psi \in \mathcal{F}(\Lambda)$ that can be used to distinguish the point of the conjectured coalgebra from the point x of the coalgebra that we are trying to learn.

1.3 Related Work

Within the large body of literature on Angluin learning [6], the line of research closely related to our paper is [2–5] which applies categorical techniques to automata learning. There are, however, important differences to our work. On the one hand, our logic-based methodology allows us to make the L^* algorithm *parametric in the system type*, which is less clear in the cited articles. On the other hand, our results are limited to the category of sets - results such as learning linear weighted or nominal automata [2, 5] are currently out of reach for our approach. Connections between modal logic and automata theory are of course well-known [8], but the link between modal logic and automata learning that we are describing is, to the best of our knowledge, new.

Acknowledgements. The authors would like to thank Nick Bezhanishvili and Alexandra Silva for helpful discussions and pointers to the literature.

2 Preliminaries

We assume that the reader is familiar with basic category theory [9] and coalgebra [10]. We briefly recall some definitions from coalgebra & modal logic that will play a central role in the paper. Throughout the paper we will be working with an arbitrary finitary set functor T and we will assume - without loss of generality [7, 11] - that T preserves intersections and for any sets X, Y with $X \subseteq Y$ the inclusion map $\iota_{X,Y} : X \to Y$ gets mapped to the inclusion $\iota_{TX,TY}$ (in particular, we have $TX \subseteq TY$). Under this assumption finitariness of the functor T means that for all sets X and all elements $t \in TX$ there is a finite subset $X' \subseteq X$ such that $t \in TX'$. Another functor that will be playing an important role in our paper is the contravariant power set functor $P : \mathsf{Set}^{op} \to \mathsf{Set}$ that maps a set X to the collection PX of subsets of X and a function $f : X \to Y$ to the inverse image function $Pf = f^{-1}$ where for $V \subseteq Y$ we have $f^{-1}(V) = \{x \in X \mid f(x) \in V\}$. Finally, for a set X we denote by $\#X$ the cardinality of X and we write $X' \subseteq_\omega X$ if X' is a *finite* subset of X.

2.1 Coalgebra

A T-coalgebra is a pair (X, γ) where X is a set and $\gamma : X \to TX$ is a function. A T-coalgebra morphism from a T-coalgebra (X, γ) to another T-coalgebra morphism (Y, δ) is a function $f : X \to Y$ such that the following diagram commutes.

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ {\scriptstyle\gamma}\downarrow & & \downarrow{\scriptstyle\delta} \\ TX & \xrightarrow[Tf]{} & TY \end{array}$$

We call two states $x_1 \in X_1$ and $x_2 \in X_2$ of two T-coalgebras (X_1, γ_1) and (X_2, γ_2) *behaviourally equivalent*[1] if there exists a T-coalgebra (Y, δ) and coalgebra morphisms $f_i : (X_i, \gamma_i) \to (Y, \delta)$ for $i = 1, 2$ such that $f_1(x_1) = f_2(x_2)$. In this case we write $x_1 \leftrightarrows_T x_2$ or simply $x_1 \leftrightarrows x_2$. A *pointed* T-coalgebra is a triple (X, γ, x) where (X, γ) is a T-coalgebra and $x \in X$ is a distinguished point (to be thought of as the "initial state"). A morphism $f : (X, \gamma, x) \to (Y, \delta, y)$ is a T-coalgebra morphism $f : (X, \gamma) \to (Y, \delta)$ such that $f(x) = y$. Two pointed coalgebras (X, γ, x) and (Y, δ, y) are behaviourally equivalent if $x \leftrightarrows y$. When discussing concrete constructions on transition systems it is important to be able to express the notion of a "successor state", i.e. a state that can be reached after performing one step in the transition structure. Coalgebra does not come with such a notion from the outset, but it is well-known that successors can be formalised via the notion of base.

Definition 1. *Given a finitary set functor T and an element $t \in TX$, we define*

$$\mathrm{Base}_X^T(t) := \bigcap \{Y \subseteq_\omega X \mid t \in TY\}.$$

Whenever T and X are clear from the context we will drop the super- and subscript, respectively, and simply write $\mathrm{Base}(t)$.

Fact 1 *Let T be a finitary set functor, let X be a set and let $t \in TX$. Then $t \in T\mathrm{Base}_X^T(t)$ and for any proper subset $X' \subset \mathrm{Base}_X^T(t)$ we have $t \notin TX'$.*

The notion of base can be used to associate with any T-coalgebra a graph that can be used to give a concrete representation of reachability.

Definition 2 ([7]). *Let (X, γ) be a T-coalgebra. The canonical graph $(X, \mathrm{can}_\gamma : X \to \mathcal{P}_f X)$ is defined by putting $\mathrm{can}_\gamma(x) := \mathrm{Base}_X^T(\gamma(x))$ for all $x \in X$.*

Note that we can confine ourselves to finitely branching canonical graphs as we are considering only *finitary* set functors.

Example 1.
1. $T = A \times \mathrm{Id}$ where A is a set and Id is the identity functor. In this case $\mathrm{Base}_X^T((a, x)) = \{x\}$ for $(a, x) \in A \times X$.
2. $T = 2 \times \mathrm{Id}^A$ where A is a finite set (the "alphabet"). Then $\mathrm{Base}_X^T((i, f)) = \{fa \mid a \in A\}$ for $(i, f) \in TX$.
3. Let $T = \mathcal{D}_\omega$ where $\mathcal{D}_\omega X$ denotes the collection of discrete probability distributions over X with finite support, i.e. $\mathcal{D}_\omega X = \{\mu : X \to [0, 1] \mid \sum_{x \in X} \mu(x) = 1, \mathrm{supp}(\mu) \text{ finite}\}$ and for a function $f : X \to Y$ and $\mu : X \to [0, 1]$ we have $(\mathcal{D}_\omega f)(\mu)(y) = \sum_{f(x)=y} \mu(x)$, where $\mathrm{supp}(\mu) = \{x \in X \mid \mu(x) \neq 0\}$ denotes the support of μ. In this case $\mathrm{Base}_X^{\mathcal{D}_\omega}(\mu) = \mathrm{supp}(\mu)$. Note that while $\mathcal{D}_\omega$ does not satisfy our condition that $X \subseteq Y$ implies $\mathcal{D}_\omega X \subseteq \mathcal{D}_\omega Y$, an isomorphic modification of $\mathcal{D}_\omega$ does, e.g. the functor $\mathcal{D}'_\omega$ that maps a set X to the set of *partial* funtions $\mu : X \rightharpoonup (0, 1]$ with $\mathrm{supp}(\mu)$ finite and $\sum_{x \in X} \mu(x) = 1$. We stick with the functor $\mathcal{D}_\omega$ as it is commonly used and as this technical subtlety is irrelevant for this specific example.

[1] Readers should think of "behavioural equivalence" as a general notion of bisimilarity. In all concrete examples in this paper both notions of equivalence coincide.

We are going to describe a learning algorithm that learns a *minimal* finite representation of the behaviour of a given pointed coalgebra. Minimal here means that the learned pointed coalgebra should not have any proper subcoalgebras or proper quotients. Let us first recall those notions.

Definition 3. *Let (X, γ, x) be a pointed T-coalgebra. We say (Y, δ, y) is a quotient of (X, γ, x) if there is a surjective morphism $q : (X, \gamma, x) \to (Y, \delta, y)$. We say (Y, δ, y) is a subobject of (X, γ, x) if there is an injection $m : (Y, \delta, y) \to (X, \gamma, x)$. A quotient or subobject (Y, δ, y) of (X, γ, x) is said to be proper if (Y, δ, y) is not isomorphic to (X, x, γ).*

Minimality of a pointed coalgebra is captured by the following notion from [7].

Definition 4. *A pointed T-coalgebra (X, γ, x) is called* well-pointed *if it is* simple *and* reachable*, i.e. if it does not have proper quotients or proper subcoalgebras.*

Example 2.
1. For $T = 2 \times (\_)^A$, the (finite) well-pointed T-coalgebras are DFAs where each state is reachable from the initial state and where no two states represent the same (regular) language.
2. For $T = \mathcal{P}_\omega$ (where $\mathcal{P}_\omega X = \{X' \mid X' \subseteq_\omega X\}$) well-pointed T-coalgebras are finitely-branching Kripke frames with a designated state such that each state is reachable from the designated state and such that no two distinct states are bisimilar.

2.2 Coalgebraic Modal Logic

Coalgebraic modal logics [12, 13] are a family of logics that allow to express properties of pointed coalgebras that are invariant under behavioural equivalence. We first recall the basic definitions and state an important fact on expressivity of these logics. Later in the paper formulas will serve as tests that allow to learn a given (pointed) coalgebra. The language of coalgebraic modal logic is determined by choosing a collection of modal operators - the so-called similarity type:

Definition 5. *A* (modal) similarity type *is a set of modal operators with arities. If Λ is a similarity type, a* Λ-structure *consists of an endofunctor $T : \mathsf{Set} \to \mathsf{Set}$, together with an assignment of an n-ary* predicate lifting*, that is, a natural transformation of type $[\![\heartsuit]\!] : (P)^n \to P \circ T$ where $P : \mathsf{Set} \to \mathsf{Set}^{op}$ is the contravariant powerset functor, to every n-ary operator $\heartsuit \in \Lambda$.*

The syntax is now given as propositional modal logic, where the modal operators are the elements of Λ. To keep things simple we treat propositional variables as nullary predicate liftings. The definition of the semantics is also standard.

Definition 6. *The* language *induced by a modal similarity type Λ is the set $\mathcal{F}(\Lambda)$ of formulas given by*

$$\mathcal{F}(\Lambda) \ni \varphi, \psi ::= \top \mid \varphi \wedge \psi \mid \neg\varphi \mid \heartsuit(\varphi_1, \ldots, \varphi_n) \qquad (\heartsuit \in \Lambda \text{ } n\text{-ary})$$

Given a Λ-structure T and a T-coalgebra (X,γ), the semantics *of $\varphi \in \mathcal{F}(\Lambda)$ is inductively given by*

$$\begin{array}{l}[\![\top]\!]_{(X,\gamma)} = X \\ [\![\varphi \wedge \psi]\!]_{(X,\gamma)} := [\![\varphi]\!]_{(X,\gamma)} \cap [\![\psi]\!]_{(X,\gamma)} \qquad [\![\neg\varphi]\!]_{(X,\gamma)} := X \setminus [\![\varphi]\!]_{(X,\gamma)}, \\ [\![\heartsuit(\varphi_1,\ldots,\varphi_n)]\!]_{(X,\gamma)} := P\gamma \circ [\![\heartsuit]\!]_X([\![\varphi_1]\!]_{(X,\gamma)},\ldots,[\![\varphi_n]\!]_{(X,\gamma)}),\end{array}$$

Instead of $x \in [\![\varphi]\!]_{(X,\gamma)}$ we will often write $(X,\gamma,x) \models \varphi$ or simply $x \models \varphi$. In words "the pointed coalgebra (X,γ,x) satisfies φ", or simply "x satisfies φ".

The logics give rise to the notion of logical equivalence.

Definition 7. *Let (X,γ) and (Y,δ) be T-coalgebras and let $\Sigma \subseteq \mathcal{F}(\Lambda)$ be a set of formulas. Two states $x \in X$ and $y \in Y$ are logically equivalent wrt Σ if for all formulas $\varphi \in \Sigma$ we have $x \models \varphi$ iff $y \models \varphi$. In this case we write $x \equiv_\Sigma y$. The equivalence classes wrt $\equiv_\Sigma$ on a coalgebra (X,γ) will be denoted by $|x|_\Sigma$, i.e. we put $|x|_\Sigma = \{x' \in X \mid x' \equiv_\Sigma x\}$. If $\Sigma = \mathcal{F}(\Lambda)$ is the set of all formulas we simply write $\equiv$ for the equivalence and denote the equivalence class of some x by $|x|$.*

The semantics of formulas of coalgebraic modal logic is invariant under behavioural equivalence. An important stronger property that coalgebraic modal logics often possess is expressivity, i.e. the property that logical equivalence implies behavioural equivalence.

Definition 8. *The language $\mathcal{F}(\Lambda)$ is called* expressive *if for all T-coalgebras (X,γ) and (Y,δ) and all $x \in X$, $y \in Y$ we have $x \equiv y$ iff $x \leftrightarroweq y$.*

For a finitary set functor T we are always able to find an expressive language, cf. e.g. [14]. Many expressive coalgebraic modal logics have been considered in the literature, we mention two examples - note that in both examples we do not need the full Boolean structure of the language to obtain expressivity. Also note that we are simply sketching the semantics of the logics without explicitly explaining how its definition can be seen as special case of Definition 6.

Example 3. 1. For $T = (\_ \times O)^I$ coalgebras correspond to so-called Mealy machines [15] and we define:

$$\mathcal{F}(\Lambda) \ni \varphi ::= \top \mid p_{i/o}, i \in I, o \in O \mid [i]\varphi, i \in I.$$

Given a T-coalgebra (X,γ), a state $x \in X$ and $i \in I$, $o \in O$, we put $x \models p_{i/o}$ if $\pi_2(\gamma(x)(i)) = o$. Intuitively, $p_{i/o}$ is true in x if the output of the coalgebra at x on input i is equal to o. Furthermore we put $x \models [i]\varphi$ if $\pi_1(\gamma(x)(i)) \models \varphi$.

2. For $T = \mathcal{P}\mathsf{At} \times \mathcal{D}_{\omega^-}$ with At a set of propositional variables, coalgebras correspond to probabilistic transition systems and an expressive language [16] is given by

$$\mathcal{F}(\Lambda) \ni \varphi ::= \top \mid p, p \in \mathsf{At} \mid \varphi_1 \wedge \varphi_2 \mid L_q\varphi, q \in [0,1],$$

where for a coalgebra (X,γ) we have $x \models p$ if $p \in \pi_1(\gamma(x))$ and $x \models L_q\varphi$ if $\sum_{x' \models \varphi} \pi_2(\gamma(x))(x') \geq q$, in words, if the probability of reaching a successor of x that makes φ true is at least q.

2.3 Logical quotients

Given an expressive modal language $\mathcal{F}(\Lambda)$ for T, it is well known that we can compute the maximal quotient of a coalgebra simply by identifying states that satisfy the same modal formulas in $\mathcal{F}(\Lambda)$.

Fact 2 *Let (X, γ) be a T-coalgebra and let $\mathcal{F}(\Lambda)$ be an expressive language, let $|X| = \{|x| \mid x \in X\}$ and define $\gamma_{\mathcal{F}\Lambda} : |X| \to T|X|$ by putting $\gamma_{\mathcal{F}(\Lambda)}(|x|) := (T|\_|)(\gamma(x))$. Then the* logical quotient *$\gamma_{\mathcal{F}\Lambda}$ is well-defined and $q : X \to |X|$ given by $x \mapsto |x|$ is a T-coalgebra map that computes the maximal quotient of (X, γ).*

The proof of well-definedness can e.g. be found in [17]. That the quotient is maximal is an immediate consequence of the fact that truth of modal formulas is preserved by coalgebra morphisms. The method also allows us to obtain the well-pointed coalgebra that is equivalent to a given pointed T-coalgebra.

Lemma 1. *Let (X, γ, x) be a pointed T-coalgebra and let (Y, δ, x) be its smallest pointed subcoalgebra. The logical quotient $(|Y|, \delta_{\mathcal{F}(\Lambda)}, |x|)$ is well-pointed.*

Proof. First note that the smallest pointed subcoalgebra (Y, δ, x) exists as T preserves intersections. It is reachable by definition. Therefore also its quotient will be reachable because (i) it is easy to see that the canonical graph of the quotient is a quotient of the canonical graph of (Y, δ, x) and (ii) by [7, Lemma 3.16] reachability of its canonical graph implies reachability of a pointed T-coalgebra. Furthermore the logical quotient of (Y, δ, x) is simple as any two distinct states can be distinguished by a formula in $\mathcal{F}(\Lambda)$ and can thus not be behaviourally equivalent. This implies that $(|Y|, \delta_{\mathcal{F}(\Lambda)}, |x|)$ is well-pointed (cf. Definition 4).

3 The $\mathbf{L^{co}}$ algorithm

We first describe the possible configurations of the algorithm. After that we describe the algorithm parametric in a finitary set functor T. Finally we prove termination and correctness of the algorithm. Throughout this section we assume that we are given a pointed T-coalgebra (X, γ, x) whose finite representation we are trying to learn. Furthermore we are given an expressive language $\mathcal{F}(\Lambda)$ for T.

3.1 Filtrations & Logical Tables

Following Angluin's original terminology we will refer to configurations of our algorithm as logical tables or simply as tables. First we need some terminology concerning sets of formulas from $\mathcal{F}(\Lambda)$.

Definition 9. *A set $\Sigma \subseteq \mathcal{F}(\Lambda)$ is* closed under taking subformulas *or* subformula closed *if we have*

- *$\varphi_1 \wedge \varphi_2 \in \Sigma$ implies $\varphi_i \in \Sigma$ for $i \in \{1, 2\}$*
- *$\neg\varphi \in \Sigma$ implies $\varphi \in \Sigma$, and*

– *for all* $\heartsuit \in \Lambda$ *we have* $\heartsuit(\varphi_1, \ldots, \varphi_n) \in \Sigma$ *implies* $\varphi_i \in \Sigma$ *for all* $i \in \{1, \ldots, n\}$.

Sets that are closed under taking subformulas are used in modal logic to compute filtrations of Kripke models and to prove a truth lemma for such filtrations [18]. It turns out that a similar idea is at work in the L* algorithm, although the relevant construction is a modification of the standard filtration. We first give a coalgebraic account of the filtrations that play a role in our algorithm.

Definition 10. *Let* (X, γ) *be a* T*-coalgebra, let* $\Sigma \subseteq \mathcal{F}(\Lambda)$ *be a subformula closed set of formulas and let* $S \subseteq X$ *be a selection of points in* X *such that*

1. *for all* $x_1, x_2 \in S$ *we have* $x_1 \not\equiv_\Sigma x_2$
2. *for all* $x \in S$ *and all* $x' \in \mathrm{Base}(\gamma(x))$ *we have* $|x'|_\Sigma \in |S|_\Sigma$

where $|S|_\Sigma = \{|x|_\Sigma \mid x \in S\}$. *The* (S, Σ)-filtration *of* (X, γ) *has as carrier the set* $|S|_\Sigma$ *and as coalgebra structure* $\gamma_{S,\Sigma}$ *we define the map*

$$\gamma_{S,\Sigma}(|x|_\Sigma) := (T|\text{-}|_\Sigma)(\gamma(x))$$

where we view the operation of identifying equivalent points as a function $|\text{-}|_\Sigma : X \to |X|_\Sigma$ *to which the functor* T *is applied.*

Lemma 2. *Under the conditions on* (S, Σ) *from the previous definition, the* (S, Σ)*-filtration of a* T*-coalgebra* (X, γ) *is well-defined.*

Proof. This follows as $x \equiv_\Sigma x'$ implies $x = x'$ and thus $\gamma(x) = \gamma(x')$ for all $x, x' \in S$ by condition 1 and as $(T|\text{-}|_\Sigma)(\gamma(x)) \in T|S|_\Sigma$ as S satisfies condition 2.

Definition 10 is reminiscent of the definition of a logical quotient in Fact 2 - the differences are that we select only one representant for each $\equiv_\Sigma$-equivalence class and that a (S, Σ)-filtration of a coalgebra is in general not a quotient. Instead a weaker property holds: restricted to elements in S, the map $|\text{-}|_\Sigma$ preserves truth of formulas in Σ.

Lemma 3. *Let* (X, γ) *be a* T*-coalgebra, let* $\Sigma \subseteq \mathcal{F}(\Lambda)$ *be a subformula closed set of formulas and let* $(|S|_\Sigma, \gamma_{S,\Sigma})$ *be the* (S, Σ)*-filtration of* (X, γ) *for some suitable* $S \subseteq X$. *Then for all* $x \in S$ *and all* $\varphi \in \Sigma$ *we have*

$$(X, \gamma, x) \models \varphi \qquad \textit{iff} \qquad (|S|_\Sigma, \gamma_{S,\Sigma}, |x|_\Sigma) \models \varphi.$$

In particular, $(|S|_\Sigma, \gamma_{S,\Sigma})$ *is simple.*

Proof. We prove the statement by induction on the structure of the formula φ. In case $\varphi = \top$ is obvious. Similarly, the Boolean cases $\varphi = \psi_1 \wedge \psi_2$ and $\varphi = \neg\psi$ easily follow from the induction hypothesis. Suppose now that $\varphi = \heartsuit(\psi_1, \ldots, \psi_n)$. We have $x \models \heartsuit(\psi_1, \ldots, \psi_n)$ iff $\gamma(x) \in [\![\heartsuit]\!]_X([\![\psi_1]\!]_{(X,\gamma)}, \ldots, [\![\psi_n]\!]_{(X,\gamma)})$ iff $x \in P\gamma\left([\![\heartsuit]\!]_X([\![\psi_1]\!]_{(X,\gamma)}, \ldots, [\![\psi_n]\!]_{(X,\gamma)})\right)$. The last statement is by I.H. on the ψ_i's equivalent to

$$\sigma(x) \in P\gamma\left([\![\heartsuit]\!]_X(P|\text{-}|_\Sigma([\![\psi_1]\!]_{(|S|_\Sigma,\gamma_{S,\Sigma})}), \ldots, P|\text{-}|_\Sigma([\![\psi_n]\!]_{(|S|_\Sigma,\gamma_{S,\Sigma})}))\right)$$

which is by naturality of $[\![\heartsuit]\!]$ equivalent to

$$x \in (P\gamma \circ PT|_-|_\Sigma)\left([\![\heartsuit]\!]_{|S|_\Sigma}([\![\psi_1]\!]_{(|S|_\Sigma,\gamma_{S,\Sigma})},\ldots,[\![\psi_n]\!]_{(|S|_\Sigma,\gamma_{S,\Sigma})})\right)$$

which is equivalent to $T|_-|_\Sigma(\gamma(x)) \in [\![\heartsuit]\!]_{|S|_\Sigma}([\![\psi_1]\!]_{(|S|_\Sigma,\gamma_{S,\Sigma})}),\ldots,[\![\psi_n]\!]_{(|S|_\Sigma,\gamma_{S,\Sigma})})$ which finally is equivalent to $|x|_\Sigma \models \heartsuit(\psi_1,\ldots,\psi_n)$ as required.

Given our observations on filtrations we are now ready to introduce the tables that will form the configurations of our algorithm.

Definition 11. *A* (logical) table *is a pair (S,Σ) where $S \subseteq X$ and $\Sigma \subseteq \mathcal{F}(\Lambda)$ is a set of formulas that is closed under taking subformulas. A table (S,Σ) is* closed *if for all $x \in S$ we have $|\mathrm{Base}(\gamma(x)|_\Sigma \subseteq |S|_\Sigma$. Finally we call (S,Σ)* sharp *if for all $x_1,x_2 \in S$ we have $x_1 \neq x_2$ implies $x_1 \not\equiv_\Sigma x_2$.*

Remark 1. Readers familiar with the L* algorithm will recognise the closedness condition. The condition that a table is sharp implies the consistency requirement in Angluin's work. Sharpness will be maintained by adding counterexamples to the tests and not to the states, similar to what is done e.g. in [19].

3.2 Description of the algorithm

Let (X,γ,x) be a pointed coalgebra that is behaviourally equivalent to a finite well-pointed coalgebra and let $\mathcal{F}(\Lambda)$ be an expressive language. We will now describe a procedure that allows to learn the well-pointed coalgebra by asking queries to a teacher that knows (X,γ,x). Our algorithm will compute a closed and sharp table (S,Σ)[2] such that $x \in S$ and such that the pointed T-coalgebra $(|S|_\Sigma,\gamma_{S,\Sigma},|x|_\Sigma)$ that is based on the (S,Σ)-filtration of (X,γ,x) is the finite quotient of a subcoalgebra of (X,γ,x) that contains x. The algorithm - depicted in Figure 1 - makes the following steps:

1. The start table is $(\{x\},\{\top\})$ - the first component ensures that the distinguished point of (X,γ,x) is represented, the second component equals $\{\top\}$ to keep the formulation uniform as $\top \in \mathcal{F}(\Lambda)$ independently of the choice of language.
2. At configuration (S,Σ), check whether (S,Σ) is closed. If yes, jump to Step 4. If not, proceed with the next step.
3. Given a configuration (S,Σ) that is not closed, pick an element $x' \in S$ such that $|\mathrm{Base}(\gamma(x'))|_\Sigma \not\subseteq |S|_\Sigma$. For all $x'' \in \mathrm{Base}(\gamma(x'))$ check whether $|x''|_\Sigma \in |S|_\Sigma$, i.e. whether the equivalence class of x'' is already represented by some other element of S. If not, then add x'' to S. Otherwise check the next element of $\mathrm{Base}(\gamma(x'))$. Note that we are adding elements of $\mathrm{Base}(\gamma(x'))$ one-by-one to maintain sharpness of the table.

[2] Instead, we could use triples $(S,\Sigma,\models_S)$ to be in line with [6] but we decided to leave the third "bookkeeping" component implicit.

4. Given the closed configuration (S, Σ), present the (S, Σ)-filtration $\gamma_{S,\Sigma}$ to the teacher. If teacher accepts, the algorithm terminates. If teacher rejects, she has to provide a counterexample, i.e. a formula $\varphi \in \mathcal{F}(\Lambda)$ s.t. $x \models \varphi$ and $|x|_\Sigma \not\models \varphi$ or vice versa. In this case we put $\Sigma' = \Sigma \cup \mathrm{Sub}(\varphi)$ and the algorithm continues at Step 2 with (S, Σ').

Algorithm 1: The L^{co} algorithm with teacher (X, γ, x) and language $\mathcal{F}(\Lambda)$

```
   Initialize table 𝒯 = (S, Σ), with S = {x} and Σ = {⊤}
⋆  Check if 𝒯 = (S, Σ) is closed
1  if Closed(𝒯) then
       Given 𝒯 closed
       Conjecture: γ_{S,Σ} : |S|_Σ → T|S|_Σ
       |x|_Σ ↦ (T|_|_Σ)(γ(x))
2      if Conjecture ==⊥ then
           Provide φ ∈ ℱ(Λ) s.t. x ⊨ φ and |x| ⊭ φ
           Update Σ ← Σ ∪ Sub(φ)
           return 𝒯
           Go to ⋆
       else
           return Aut(𝒯)
   else
       Given 𝒯 not closed
       Pick x ∈ S s.t. |Base(γ(x))|_Σ ⊈ |S|_Σ
3      while Base(γ(x)) ≠ ∅ do
           Take y ∈ Base(γ(x))
4          if |y| ∈ |S|_Σ then
               Base(γ(x)) ← Base(γ(x)) \ {y}
           else
               Update S ← S ∪ {y}
               Base(γ(x)) ← Base(γ(x)) \ {y}
       return 𝒯
       Go to ⋆
```

3.3 Termination & correctness

In this section we are going to prove termination and correctness of our L^{co} algorithm. We first state the theorem and sketch its proof. After that we will provide the proofs for the necessary technical lemmas.

Theorem 3. *Let (X, γ, x) be a pointed T-coalgebra and suppose (X, γ, x) is behaviourally equivalent to a finite well-pointed T-coalgebra (Y, δ, y). Let $\mathcal{F}(\Lambda)$ be*

an expressive language for T-coalgebras. The L^{co} algorithm with teacher (X, γ, x) and test language $\mathcal{F}(\Lambda)$ terminates and returns the correct well-pointed coalgebra.

Proof. That the algorithm terminates can be seen as follows:

- The algorithm builds tables (S, Σ) where S is a subset of the carrier of the smallest subcoalgebra of (X, γ, x) that contains x. Therefore the size of S is bound by the number of elements of Y (Lemma 4 and Lemma 5).
- Whenever a table is not closed, the algorithm will be able to close it. Whenever the algorithm turns a table into a closed one, the size of S strictly increases (Lemma 6).
- Whenever the teacher provides a counterexample the resulting table will not be closed (Lemma 7).

Collectively these claims show that the teacher eventually will accept the conjecture that the algorithm produces. Correctness of the algorithm then follows: Let (S, Σ) be a closed & sharp table. By Lemma 3 we have that the (S, Σ)-filtration $(|S|_\Sigma, \gamma_{S,\Sigma}, |x|_\Sigma)$ of (X, γ, x) is simple. To see that it is reachable consider the following diagram

$$\begin{array}{ccc} TS & \xrightarrow{\mathrm{Base}^T_S} & \mathcal{P}_\omega S \\ {\scriptstyle T|\text{-}|_\Sigma}\downarrow & & \downarrow{\scriptstyle \mathcal{P}_\omega|\text{-}|_\Sigma} \\ T|S|_\Sigma & \xrightarrow[\mathrm{Base}^T_{|S|_\Sigma}]{} & \mathcal{P}_\omega|S|_\Sigma \end{array}$$

where $|\text{-}|_\Sigma$ is the mapping to equivalence classes restricted to $S \subseteq X$. Clearly $|\text{-}|_\Sigma$ is mono (due to sharpness of (S, Σ)) and thus we know from [7, Rem. 2.49] that the square commutes (and even forms a pullback). Whenever the algorithm adds a state x' to S we know that $x' \in \mathrm{Base}(\gamma(x''))$ for some $x'' \in S$ that has been already added at an earlier stage. Commutation of the diagram implies that $|x'|_\Sigma \in \mathrm{Base}(T|\text{-}|_\Sigma(\gamma(x''))) = \mathrm{Base}(\gamma_{S,\Sigma}(|x''|_\Sigma))$ where the equality is a consequence of the definition of the filtration. A routine induction argument together with [7, Lemma 3.16] reducing reachability of the coalgebra to reachability of its canonical graph can now be used to show that the (S, Σ)-filtration is reachable and hence well-pointed. If, in addition, the teacher accepts the (S, Σ)-filtration of (X, γ, x) we have that $(X, \gamma, x) \models \varphi$ iff $(|S|_\Sigma, \gamma_{S,\Sigma}, |x|_\Sigma) \models \varphi$ for all formulas $\varphi \in \mathcal{F}(\Lambda)$. By expressivity of $\mathcal{F}(\Lambda)$ this means (X, γ, x) is behaviourally equivalent to $(|S|_\Sigma, \gamma_{S,\Sigma}, |x|_\Sigma)$, i.e. we have learned the correct well-pointed coalgebra.

3.4 Termination Lemmas

In this section we prove the claims from the proof of Theorem 3. In the following let (X, γ, x) be a pointed T-coalgebra that is behaviourally equivalent to a finite well-pointed coalgebra (Y, δ, y). We start by proving that states occurring in tables are always taken from the reachable part of (X, γ, x).

Lemma 4. *Let (S, Σ) be a table that is obtained in a run of the L^{co} algorithm with teacher (X, γ, x). Then S is contained in the subcoalgebra of (X, γ) that is generated by x.*

Proof. Let $(X', \gamma_{\restriction X'}, x)$ be the smallest pointed subcoalgebra of (X, γ, x). Clearly we have $\mathrm{Base}(\gamma(x')) \subseteq X'$ for all $x' \in X$ as the coalgebra map γ restricts to a map $X' \to TX'$. Now we can easily prove the claim on the number n of steps the L^{co} algorithm was closing a table to reach the table (S, Σ). For $n = 0$ we have $S = \{x\}$ and obviously $x \in X'$. For $n = m + 1$ there is a table (S', Σ) such that S is obtained from S by closing the table (S', Σ). By I.H. we know that $S' \subseteq X'$ and by the definition of the algorithm we have $S \subseteq S' \cup \bigcup_{x' \in S'} \mathrm{Base}(\gamma(x'))$. But as we saw at the beginning of our proof the latter is a subset of X' which shows that $S \subseteq X'$ as required.

Consequently, we obtain an upper bound for the number of elements of S for each table S.

Lemma 5. *Let (S, Σ) be a table computed by the L^{co} algorithm with teacher (X, γ, x). Then $\#S \leq \#Y$.*

Proof. By Lemma 1 we know that (Y, δ, y) is the quotient of the smallest pointed subcoalgebra (X', γ', x) of (X, γ, x) and by the previous lemma we have $S \subseteq X'$. The quotient map from (X', γ') to (Y, δ) is a coalgebra morphism and preserves the truth of modal formulas. Therefore, for any element of $x' \in S \subseteq X'$ there exists a $y \in Y$ such that $x' \equiv y$, in other words there is a function $f : S \to Y$ such that $x \equiv f(x)$. We also know that (S, Σ) is sharp which means that $x_1 \equiv y$ and $x_2 \equiv y$ implies $x_1 = x_2$. Therefore the function $f : S \to Y$ has to be injective which shows that $\#S \leq \#Y$ as required.

This finishes the argument for why there is a finite upper bound on the number of states stored in a table. Let us now turn to what happens in case the current table is not closed.

Lemma 6. *Let (S, Σ) be a table computed by the L^{co} algorithm with teacher (X, γ, x). If (S, Σ) is not closed then L^{co} computes a closed table (S', Σ) such that $\#S < \#S'$.*

Proof. Suppose (S, Σ) is a table that is not closed. While (S, Σ) is not closed, the L^{co} algorithm extends the collection of states S by elements of X as described in the previous section. By Lemma 5 this can only happen finitely often which implies that after finitely many additions to S the table will be closed as required.

Finally we need to investigate what happens in case the teacher is providing a counterexample.

Lemma 7. *Let (S, Σ) be a closed and sharp table computed by the L^{co} algorithm with teacher (X, γ, x). If the teacher provides a counterexample φ, then (S, Σ') is not closed where Σ' is the smallest set of formulas that contains $\Sigma \cup \{\varphi\}$ and that is closed under subformulas.*

Proof. Suppose for a contradiction that (S, Σ') is closed (obviously it will be sharp). For every set $Z \subseteq X$ let $f_Z : |Z|_{\Sigma'} \to |Z|_\Sigma$ given by $f_Z(|x'|_{\Sigma'}) := |x'|_\Sigma$ for all $x' \in Z$. Note this is well-defined as $\Sigma' \supseteq \Sigma$ and thus $x_1 \equiv_{\Sigma'} x_2$ implies $x_1 \equiv_\Sigma x_2$. We now claim that f_S is a pointed T-coalgebra morphism from the (S, Σ')-filtration of (X, γ, x) to its (S, Σ) filtration. To check this we first note that $f_X \circ |\_|_{\Sigma'} = |\_|_\Sigma$. Let $x' \in S$ be arbitrary. We calculate:

$$\begin{aligned}\gamma_{S,\Sigma}(f_S(|x'|_{\Sigma'})) = \gamma_{S,\Sigma}(|x'|_\Sigma) = (T|\_|_\Sigma)(\gamma(x')) = T(f_X \circ |\_|_{\Sigma'})(\gamma(x')) \\ = Tf_X(T|\_|_{\Sigma'}(\gamma(x'))) \overset{T|\_|_{\Sigma'}(\gamma(x')) \in T|S|_{\Sigma'}}{=} Tf_S(T|\_|_{\Sigma'}(\gamma(x')))\end{aligned}$$

which shows that f_S is indeed a pointed T-coalgebra morphism between the filtrations. This is, however, a contradiction to Σ' containing a counterexample, i.e. a formula $\varphi \in \mathcal{F}(\Lambda)$ such that w.l.o.g. $|x|_{\Sigma'} \models \varphi$ and $|x|_\Sigma \not\models \varphi$.

This finishes the proofs of the claims that are necessary to prove Theorem 3. In summary, we have proved termination and correctness of our algorithm. Remarkably, if we measure the complexity of the algorithm in the number of logical and equivalence queries, our algorithm meets similar bounds as Angluin's L^*-algorithm. The main difference is that the run-time of our algorithm also depends on the maximal number of successors a state of the original coalgebra has - in the case of finite automata the branching is bounded by a constant, namely the size of the input alphabet. We need the following definitions for our complexity considerations:

- The *size of a formula* is the number of its distinct subformulas.
- A T-coalgebra (X, γ) is *k-branching* if there exists some $k \in \omega$ such that for all $x \in X$ we have $\#\mathrm{Base}(\gamma(x)) \leq k$.

Proposition 1. *Let (X, γ, x) be a k-branching pointed T-coalgebra, let n be the size of the behavioural equivalent well-pointed T-coalgebra that the algorithm learns and let m be the maximal size of the counterexample provided by the Teacher. Then the algorithm terminates after asking $\mathcal{O}(k \cdot m \cdot n^2)$ logical queries and at most n equivalence queries.*

Proof. Let (S, Σ) be a table occurring during the run of the algorithm. By Lemma 5 we have $\#S \leq n$. Furthermore, the size of Σ is bound by $m \cdot (n-1)$ as each counterexample is of size at most m and at most $n-1$ counterexamples can be added to Σ as the addition of a counterexample always results in an increase of the size of S by Lemmas 7 and 6. To check closedness of a table (S, Σ) we need to ask logical queries about the elements of S and their successors. Therefore, we need at most $(k+1) \cdot n \cdot m \cdot (n-1)$ such queries. This shows that we need $\mathcal{O}(k \cdot m \cdot n^2)$ logical queries. The upper bound on equivalence queries is obvious.

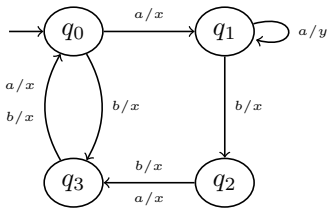
Our discussion demonstrates that the move from deterministic finite automata to T-coalgebras does not essentially alter the number of queries. At this stage we cannot predict how fast queries can be answered in practice. To answer this question we plan to implement our algorithm in the near future.

4 Examples

We will illustrate our algorithm with two examples: One known example on learning Mealy machines and one example that is to the best of our knowledge new, an Angluin learning algorithm for discrete Markov chains. While learning probabilistic automata is an active area [20–22], existing work uses other learning paradigms and focuses on constructing minimal automata for a probabilistic language rather than - as our algorithm does - on *bisimulation* quotients of transition systems.

Example 4 (Mealy machines). In the first example we show how the algorithm learns functions $L\colon I^+ \to O$, i.e. maps from finite sequences over I to elements of O where I and O are finite sets that constitute the input and output alphabet, respectively. The "regularity" assumption on L is that L can be represented by a finite Mealy machine. The latter are a generalization of deterministic automata where each transition has an associated input and output letter. Such Mealy machines correspond to coalgebras for the Mealy functor $T = (\_ \times O)^I$ [15]. Learning L is equivalent to learning a finite representation of the coalgebra $I^* \xrightarrow{\gamma} (I^* \times O)^I$ with designated point $\lambda \in I^*$, where $\gamma(w)(a) = \langle wa, L(wa)\rangle$. Recall the expressive language from Example 3 for the Mealy functor.

As a concrete example, we want to learn the function L represented by the Mealy machine in the following diagram, where the input alphabet is $I = \{a, b\}$ and the output alphabet is $O = \{x, y\}$:



According to the algorithm 1, the first thing to do is checking if the starting table $\mathcal{T} = (\{\lambda\}, \{\top\})$ is closed. The table is trivially closed, as $\top$ is always true and the first conjecture of the algorithm is as follows:

As the conjecture is incorrect the teacher returns a counterexample, e.g. the formula $[a]p_{a/y}$. We update the set Σ of formulas with this new formula together with all its subformulas. Note that this process is a well-known variant of the one described in [6] where the counterexample is usually added to the set S of states and not to Σ. We use this variant that allows to automatically maintain the table consistent (cf. [19]), in order to be sure to have a sharp table at every step.

The table is now $\mathcal{T} = (\{\lambda\}, \{\top, p_{a/y}, [a]p_{a/y}\})$ and it is not closed. Therefore, we add another state to the set of states according to the **else** part of the **if**-statement 1. We pick a state $x' \in |S|_\Sigma$ such that its successors' equivalence classes

are not represented in S, in symbols: $|\text{Base}(\gamma(x'))|_\Sigma \not\subseteq |S|_\Sigma$. In our particular case, we can pick only λ; the two successors of λ are $\langle a, x\rangle$ and $\langle b, x\rangle$ and we have $\text{Base}(\gamma(\lambda)) = \{a, b\}$. We take one element of this set, say a, the equivalence class of a is not equal to $|\lambda|_\Sigma$, therefore we add a to S and we remove it from $\text{Base}(\gamma(\lambda))$. We do the same for b. Because $|b|_\Sigma \notin |\{\lambda, a\}|_\Sigma$, b is also added to S.

This procedure can be seen as the standard table filling process in the L*-algorithm: in the lower part of the table, we identify those states that do not have a corresponding representative in the set of states S and we add them in the upper part of the table; in our case both a and b have to be added. The entries of the table are 1 and 0, according to the truth values which the formula assumes in a particular state. We can schematically see this process in Table 1, where the values in the first column are shorthands for $\pi_1(\gamma(w)(a))$, i.e., we write wa for the state reached from w after reading input a.

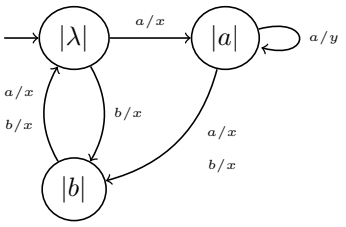
Table 1. From a not closed table to a closed one

	$\top$	$p_{a/y}$	$[a]p_{a/y}$
λ	1	0	1
a	1	1	1
b	1	0	0

$\Rightarrow$

	$\top$	$p_{a/y}$	$[a]p_{a/y}$
λ	1	0	1
a	1	1	1
b	1	0	0
aa	1	1	1
ab	1	0	0
ba	1	0	1
bb	1	0	1

Having added a and b to S, we have a new table: $\mathcal{T} = (\{\lambda, a, b\}, \{\top, p_{a/y}, [a]p_{a/y}\})$, that is closed. Now, we can make our conjecture:



But the conjecture is incorrect and we again receive a counterexample, e.g. $[a][b][b][a]p_{a/x}$. The resulting table is not closed, as $|\text{Base}(\gamma(a))|_\Sigma \not\subseteq |S|_\Sigma$. Indeed, the successor of a, aa, should satisfy $[a][b][b][a]p_{a/x}$, whereas the formula is false in a. We only pick the element aa from $\text{Base}(\gamma(a)) = \{aa, ab\}$ as the equivalence class of ab is already represented. The new table is: $\mathcal{T} = (\{\lambda, a, b, aa\}, \Sigma)$.

Closing the table with aa, we go to **else** 2 and we compute our conjecture. Table 2 on page 16 represents this step. No counterexamples are given back, therefore the conjecture is the right one and we have the same automaton $Aut(\mathcal{T})$ describing the Mealy machine we wanted to learn.

Note that there is a difference with the standard Angluin algorithm adapted for Mealy machines. Usually the entries of the tables are the output values that

Table 2. Changing of the table according to the Base-step of the algorithm

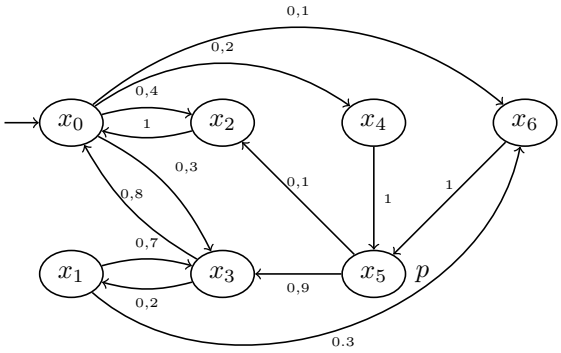
	$\top$	$p_{a/y}$	$[a]p_{a/y}$	$p_{a/x}$	$[a]p_{a/x}$	$[b][a]p_{a/x}$	$[b][b][a]p_{a/x}$	$[a][b][b][a]p_{a/x}$
λ	1	0	1	1	0	1	0	1
a	1	1	1	0	0	1	1	1
b	1	0	0	1	1	0	1	0
aa	1	1	1	0	0	1	1	1
ab	1	0	0	1	1	1	0	1
ba	1	0	1	1	0	1	0	1
bb	1	0	1	1	0	1	0	1

$\Downarrow$

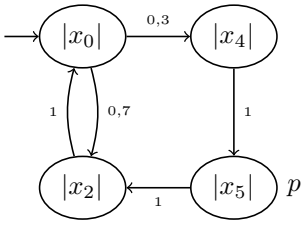
	$\top$	$p_{a/y}$	$[a]p_{a/y}$	$p_{a/x}$	$[a]p_{a/x}$	$[b][a]p_{a/x}$	$[b][b][a]p_{a/x}$	$[a][b][b][a]p_{a/x}$
λ	1	0	1	1	0	1	0	1
a	1	1	1	0	0	1	1	1
b	1	0	0	1	1	0	1	0
aa	1	1	1	0	0	1	1	1
ab	1	0	0	1	1	1	0	1
ba	1	0	1	1	0	1	0	1
bb	1	0	1	1	0	1	0	1
aaa	1	1	1	0	0	1	1	1
aab	1	1	1	0	0	1	1	1

the Mealy automata produce. This is not the case in our algorithm. The entries are always the values 1 or 0 according to the truth values of the formulas, whereas the output values are given by the function γ.

Example 5 (Discrete Markov Chain). In this example we show that our algorithm can learn bisimulation quotients of (discrete) probabilistic transition systems. We confine ourselves to a finite example and we model probabilistic transition systems as $\mathcal{P}\mathsf{At} \times \mathcal{D}_\omega$-coalgebras where At is a set of propositional variables - in our example we will assume $\mathsf{At} = \{p\}$ for a single proposition p. Note, however, that the input model of our learning algorithm could be infinite and that our coalgebraic framework is general enough to cover a large variety of probabilistic systems [23]. Recall an expressive language for this example from Example 3. Consider now the following pointed $\mathcal{P}\mathsf{At} \times \mathcal{D}_\omega$-coalgebra based on $X = \{x_0, x_1, \ldots, x_6\}$ with initial state x_0 and transition map γ, where transitions are labelled with their likelihoods and proposition p holds exclusively at state x_5:



The algorithm tries to learn a minimal pointed coalgebra that is behaviourally equivalent to (X, γ, x_0). As always the algorithm starts with table $(\{x_0\}, \{\top\})$. This table is trivially closed and our first conjecture will be a state at which p does not hold and with a loop of probability 1. The teacher has to reject the conjecture and provides a counter example, e.g. the formula $\varphi = L_{0.2}L_1 p$ as $x_0 \models \varphi$ and $|x_0| \not\models \varphi$. This leads to the new table $(\{x_0\}, \Sigma_1)$ with $\Sigma_1 = \{L_{0.2}L_1 p, L_1 p, p, \top\})$. We have $\text{Base}(\gamma(x_0)) = \{x_2, x_3, x_4, x_6\}$ and it is easy to see that $|\{x_2, x_3, x_4, x_6\}|_{\Sigma_1} \not\subseteq |\{x_0\}|_{\Sigma_1}$. Therefore the table is not closed. It can easily be checked that $x_2 \equiv_{\Sigma_1} x_3$ and $x_4 \equiv_{\Sigma_1} x_6$ and closing the table leads to $(\{x_0, x_2, x_4\}, \Sigma_1)$. We have $\text{Base}(\gamma(x_4)) = \{x_5\}$ and obviously $|x_5|_{\Sigma_1} \notin |\{x_0, x_2, x_4\}|_{\Sigma_1}$ as x_5 is the only state satisfying proposition p. Closing the table we arrive at $(\{x_0, x_2, x_4, x_5\}, \Sigma_1)$ and this table is closed as readers can easily convince themselves. The table leads to the correct conjecture:



5 Conclusions

While the last example was simple it demonstrates how our algorithm can be used to determine the quotient of a discrete Markov chain modulo behavioural equivalence, i.e. bisimilarity. Other transition systems can be quotiented in the same way. This is interesting as bisimulation quotients play an important role in verification [24] and as the complexity of standard algorithms is determined by the size of the system before taking the quotient whereas the complexity of the learning algorithm depends on the size of the potentially much smaller quotient. To explore the practical usefulness of our algorithm in these cases we are planning

to provide an implementation in the near future. The biggest challenge will in our view be to implement a teacher that provides "good" counterexamples.

There are many more questions concerning our work that we would like to clarify: Our approach is currently very much based on the category of sets. This excludes for example the important example of linear weighted automata [2]. We believe that our arguments can be extended by (i) building on a duality between algebras and coalgebras and (ii) understanding how filtrations translate via this duality. There are some partial answers to the latter question in the modal logic literature (cf. e.g. [25, 26]) but modal logicians usually focus on properties of filtrations that do not play a role in learning. Closely related to this question is a more diagrammatic understanding of termination and correctness of our algorithm: we believe that all essential ingredients for this are contained in our work but we need to understand filtrations on a more abstract, categorical level.

References

1. Jacobs, B.: Introduction to Coalgebra: Towards Mathematics of States and Observation. Cambridge Tracts in TCS. Cambridge University Press (2016)
2. Jacobs, B., Silva, A.: Automata learning: A categorical perspective. Volume 8464 of LNCS. Springer (2014) 384–406
3. van Heerdt, G.: An abstract automata learning framework. Master's thesis, Radboud Universiteit Nijmegen (2016)
4. Moerman, J., Sammartino, M., Silva, A., Klin, B., Szynwelski, M.: Learning nominal automata. POPL 2017 (2017)
5. van Heerdt, G., Sammartino, M., Silva, A.: Learning automata with side-effects. CoRR **abs/1704.08055** (2017)
6. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75**(2) (1987) 87 – 106
7. Adámek, J., Milius, S., Moss, L.S., Sousa, L.: Well-pointed coalgebras. Logical Methods in Computer Science **9**(3) (2013)
8. Grädel, E., Thomas, W., Wilke, T., eds.: Automata, Logic, and Infinite Games. Volume 2500 of LNCS. Springer (2002)
9. Mac Lane, S.: Categories for the Working Mathematician. Volume 5 of Graduate texts in Mathematics. Springer-Verlag, New York (1971)
10. Jacobs, B., Rutten., J.: An introduction to (co)algebras and (co)induction. In: Advanced topics in bisimulation and coinduction. Volume 52 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press (2011) 38–99
11. Adámek, J., Trnková, V.: Automata and Algebras in Categories. Kluwer Academic Publishers (1990)
12. Cirstea, C., Kurz, A., Pattinson, D., Schröder, L., Venema, Y.: Modal logics are coalgebraic. The Computer Journal **54**(1) (2009) 31–41
13. Kupke, C., Pattinson, D.: Coalgebraic semantics of modal logics: An overview. Theoretical Computer Science **412**(38) (2011) 5070 – 5094
14. Schröder, L.: Expressivity of coalgebraic modal logic: The limits and beyond. Theoretical Computer Science **390**(2) (2008) 230 – 247
15. Hansen, H.H., Rutten, J.J.M.M.: Symbolic synthesis of mealy machines from arithmetic bitstream functions. Sci. Ann. Comp. Sci. **20** (2010) 97–130

16. Desharnais, J., Edalat, A., Panangaden, P.: Bisimulation for labelled markov processes. Information and Computation **179**(2) (2002) 163 – 193
17. Kupke, C., Leal, R.A.: Characterising behavioural equivalence: Three sides of one coin. In Kurz, A., Lenisa, M., Tarlecki, A., eds.: Calco Proceedings, Berlin, Heidelberg, Springer Berlin Heidelberg (2009) 97–112
18. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Number 53 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press (2001)
19. Maler, O., Pnueli, A.: On the learnability of infinitary regular sets. Information and Computation **118**(2) (1995) 316 – 326
20. Balle, B., Castro, J., Gavald, R.: Learning probabilistic automata: A study in state distinguishability. TCS **473** (2013) 46 – 60
21. Mao, H., Chen, Y., Jaeger, M., Nielsen, T.D., Larsen, K.G., Nielsen, B.: Learning probabilistic automata for model checking. In: 2011 Eighth International Conference on Quantitative Evaluation of SysTems. (2011) 111–120
22. Tzeng, W.G.: Learning probabilistic automata and markov chains via queries. Machine Learning **8**(2) (1992) 151–166
23. Sokolova, A.: Probabilistic systems coalgebraically: A survey. TCS **412**(38) (2011) 5095–5110
24. Glück, R., Möller, B., Sintzoff, M.: Model refinement using bisimulation quotients. In Johnson, M., Pavlovic, D., eds.: AMAST 2010. Springer (2011)
25. Ghilardi, S.: Continuity, freeness, and filtrations. Journal of Applied Non-Classical Logics **20**(3) (2010) 193–217
26. Bezhanishvili, G., Bezhanishvili, N., Iemhoff, R.: Stable Canonical Rules. The Journal of Symbolic Logic **81**(1) (2016) 284–315