

Angluin Learning: a Modal Logic Approach

PhD Thesis

Simone Barlocco

Mathematically Structured Programming Group

Computer and Information Sciences

University of Strathclyde, Glasgow

September 21, 2026

*To my mum and dad,
Gisella and Carlo*

Declaration

This thesis is the result of the author's original research. It has been composed by the author and has not been previously submitted for examination which has led to the award of a degree.

The copyright of this thesis belongs to the author under the terms of the United Kingdom Copyright Acts as qualified by University of Strathclyde Regulation 3.50. Due acknowledgement must always be made of the use of any material contained in, or derived from, this thesis.

Abstract

This thesis develops a framework for coalgebra learning by investigating the intersections between automata theory, the theory of coalgebras, and modal logic.

We begin by presenting the L^* algorithm together with foundational notions of automata theory, coalgebra and modal logic. We then reformulate this algorithm within a coalgebraic setting, showing an interesting link between the classical Angluin algorithm and the concept of filtration in modal logic. By reinterpreting the tests used in L^* as coalgebraic modal formulas, we provide a version of the algorithm specifically for coalgebras that is parametric in an arbitrary finitary set functor B . This algorithm enables the learning of a finite representation of a given, potentially infinite, regular pointed B -coalgebra.

Furthermore, by using a duality framework, we generalise this approach to provide an abstract learning algorithm. This framework uses logical formulas as tests based on a dual adjunction between states and logical theories. The resulting algorithm is stated in categorical terms together with proofs of correctness and termination. Also, we extend this framework to handle computational effects, such as non-determinism and alternation. This is achieved through the introduction of BT -coalgebras and forgetful logics, which allow to abstract away from branching behaviours while maintaining a consistent set of logical formulas whose syntax can describe the behaviour of BT -coalgebras and their B -coalgebra counterparts in the same way.

Finally, we study how to combine minimisation and reachability rephrasing what has been presented by Lee and Yannakakis using modal logic notions.

Acknowledgements

My deepest and first thanks go to my supervisor, Clemens Kupke. His support has been absolutely invaluable, both during the time I spent in Scotland and after I left. Even as I kept working on the thesis at a slower pace over the years, he showed immense patience, always finding the time to meet with me remotely and guide me through. His dedication to explaining complex concepts and his continuous encouragement are the primary reasons I am able to complete this work today.

The wider Mathematically Structured Programming (MSP) group at the University of Strathclyde also taught me a great deal. During my years in Scotland, the group provided a remarkably friendly and welcoming environment. I am deeply grateful to the people there who were always open to explaining the new topics they were passionately studying. Our discussions were consistently rich, helpful, and went well beyond what appears in this thesis.

Finally, I want to thank my family for always being there for me, especially my dad, Carlo, and my mum, Gisella. I am so grateful for your patience and support throughout this long journey. My thanks also go to my friends outside of academia. I especially want to thank those I met in Glasgow: Zinovia, who made me feel at home, and William, who introduced me to Glaswegian volleyball. I also want to thank those friends who have always been by my side, whether we were miles apart or living next door. Thank you all for your patience and for being there throughout this long journey and more.

Beyond the academic work, Glasgow left me with an important lesson: life happens. You can put all your effort into one goal, only to realise that your direction has changed. People leave our lives, leaving behind an empty space and a new understanding, while

Chapter 0. Acknowledgements

others arrive to join us, bringing something unexpected and meaningful. We should strive for our objectives, but be prepared to find that the path life chooses for us is not always the one we expected.

Contents

Abstract	iii
Acknowledgements	iv
List of Figures	viii
List of Tables	ix
List of Algorithms	x
1 Introduction	2
1.1 Background	2
1.2 Coalgebras and modal logics	5
1.3 Outline of the thesis and original contribution	6
1.4 Related work	9
2 Background	12
2.1 Automata and L^*	12
2.1.1 Automata	12
2.1.2 Learning regular sets	15
2.2 Coalgebras	21
2.2.1 Introducing coalgebras via an example	21
2.2.2 Coalgebras and their general framework	24
2.2.3 Example of coalgebraic structures	26
2.3 Coalgebraic modal logic and dual adjunction	28

Contents

2.3.1	Coalgebraic modal logic	28
2.3.2	Dual adjunction	32
3	Learning via Logic	36
3.1	The L^{co} algorithm at a glance	37
3.2	Preliminaries	38
3.2.1	Notion of successor state in coalgebras	39
3.2.2	Minimal representation	40
3.2.3	Logical quotient	41
3.3	The L^{co} algorithm	42
3.3.1	Filtrations and logical tables	43
3.3.2	Description of the algorithm	46
3.3.3	The L^{co} algorithm with two examples	47
3.3.4	Termination and correctness	52
3.4	Conclusion	57
4	Learning via Duality	58
4.1	Preliminaries	60
4.1.1	Subobjects and intersection-preserving functors	62
4.1.2	Subformula-closedness	63
4.1.3	The base	65
4.2	Learning via duality	73
4.2.1	Tables and counterexamples	75
4.2.2	The algorithm	83
4.2.3	Termination and correctness	84
4.3	Learning BT -structures	93
4.3.1	Forgetful logic	93
4.3.2	BT learning	96
4.3.3	Termination and correctness	100
4.4	Examples	110
4.5	Conclusion	115

Contents

5	Learning via Defects	118
5.1	Minimisation	119
5.2	Combining minimisation and reachability	126
5.3	Conclusion	133
6	Conclusion	135
	Bibliography	140

List of Figures

2.1	Minimal DFA accepting the language of Example 3.	15
2.2	Conjecture in the form of a DFA.	19
3.1	From a not closed table to a closed one.	49
3.2	Changing of the table according to the Base-step of the algorithm. . . .	50
4.1	Exemplification of the closing table process for NFA example.	112
4.2	Exemplification of the closing table process for AFA example.	114

List of Tables

1.1	Origin of the presented material.	9
2.1	Example of an observation table.	16
2.2	Observation table at step 0.	19
2.3	Observation table at step 1.	19
2.4	Observation table at step 2.	20
2.5	Observation table at step 3.	20

List of Algorithms

1	Sketch of the L^* algorithm	19
2	The L^{co} algorithm	47
3	Abstract learning algorithm	84
4	Logical minimisation	123
5	Logical minimisation with reachability	129

List of Algorithms

Chapter 1

Introduction

1.1 Background

In computer science, the ability to define models that abstractly represent the behaviour of a system is of particular interest. Such models enable formal techniques like model checking and verification, allowing both humans and computers to efficiently analyse the properties of a system [BK08; CGP99]. Among these models, automata are a fundamental type used to represent state-based systems. An automaton is typically defined as a state transition diagram, where states capture the internal configuration of the system (its behaviour) and transitions represent the actions allowed in a specific state.

In many real-world scenarios, a formal model of a specific system is unavailable and in this context, a family of techniques called automata learning has emerged to infer state-based models from the observations of the behaviour of a system. Over the past decade, automata learning has found diverse practical applications, including the analysis of smartcard readers, reverse engineering implementations of network protocols, refactoring legacy software and methodology to describe the errors in some programs [Cha+14; SHV16; Cha+15; FJV16; RP15].

Automata learning is fundamentally characterised as an interaction between a target system and a learning algorithm, frequently referred to as the *Learner*. The primary goal for the learner is to infer the structure of the target system and to represent it as

a formal model, such as an automaton. This process can be approached through two distinct paradigms. In *passive learning*, often referred to as *learning from examples*, the algorithm receives a fixed set of data. By analysing these recorded behaviours, the algorithm attempts to construct a model that generalises the observed data and approximates the underlying logic of the system [AMP22; CAL94]. In contrast, *active learning* involves a more dynamic interaction where the learner is permitted to actively query the target system to refine its internal hypothesis. Within this framework, the system is viewed as a *Teacher* or *Oracle* capable of providing precise information to the learner. This eventually allows a construction of the exact representation of the target system in the form of an automaton [Ang87; CAL94; Set12; SC00; WS16; Ton01].

There exist several classes of automata, categorized primarily by the nature of their transition structures. One of the most fundamental classes is the Deterministic Finite Automaton (DFA). A DFA consists of a finite set of states, an initial state, and a transition function that, given a state and an input, maps to exactly one successor state. Each state is further classified as either accepting or rejecting. The semantics of a DFA is defined by word acceptance: given a finite sequence of inputs (a word), the word is accepted if the unique path through the automaton ends in an accepting state, and rejected otherwise [HMU00].

In her seminal work, Angluin [Ang87] introduced the L^* algorithm, which is capable of inferring a regular language in the form of a DFA through two types of queries that can be asked to an *Oracle*: membership queries and equivalence queries. The former are used to check whether a specific word is accepted by the target system. The latter are used to verify whether a current hypothesis, called *conjecture* and represented as a DFA, is correct. If the conjecture is incorrect, the algorithm receives a counterexample: a word that is incorrectly classified by the hypothesis. The L^* algorithm is particularly significant because it introduces the idea of equivalence queries allowing to learn the target language in polynomial time relative to the size of the alphabet, the number of states in the minimal DFA, and the length of the longest counterexample. Prior to Angluin's contribution, foundational works in active learning were conducted by researchers such as Gold [Gol78], who defined the idea of learning *in the limit*, i.e., the

algorithm provides a stream of DFAs based on accumulating knowledge of the target language and guarantees only that the stream converges at some point to a correct DFA.

In [Moo56], Moore highlighted the limitations of only using membership queries, showing that such queries are insufficient for the exact identification of a DFA within finite time. The underlying intuition is that any finite set of observations could remain consistent with an infinite variety of languages. Without additional information, the learner cannot uniquely distinguish the target system from other candidates. Subsequently, Angluin [Ang81] established that even if the learner possesses prior knowledge of the number of states in the minimal DFA, any algorithm restricted to membership queries would still face an exponential query complexity in the worst case scenario. Recognising that a polynomial time solution required a more powerful interaction between the algorithm and the target system, Angluin in [Ang87] investigated the efficacy of adding a new type of query and it is in this scenario, where the L^* algorithm arose. Several variations of L^* have been proposed over the years. These include both optimisations to improve efficiency [MP95; RS93; KV94] as well as extensions to more complex models. For example, Vilar [Vil96] adapted the L^* algorithm to learn subsequential transducers. Based on it, several other algorithms have been defined to learn Mealy machines [PO98]. Furthermore, the learning paradigm has been extended to weighted automata [BV96; Bor09], alternating automata [AEF15; Ber+17] and non-deterministic ones [Bol+09] that for the nature of their transitions allow to learn a more compact version of an automaton compared to its deterministic counterpart. Another class of automata, which serves as the primary tool for reasoning about reactive systems and their behaviours, is that of ω -automata [LST23]. These automata recognise ω -regular languages. One of the first models introduced to accept this type of language was the Büchi automaton, which is designed to process infinite sequences of symbols [Büc62]. Since then, various other models have been proposed, classified by their specific acceptance conditions, such as Muller, Rabin, Streett, and parity automata [WS21]. More recently, Angluin et al. introduced families of DFAs, known as FDFAs, as an alternative formalism for recognising ω -regular languages. FDFAs pro-

vide a structured representation that correlates states with right congruence relations (analogous to the Myhill-Nerode theorem for finite words) and have proven effective in enabling the development of learning algorithms for this class of languages [AF18; ABF18; AF16]. Further variations of FDFAs are also presented in [LST23].

1.2 Coalgebras and modal logics

While extensive research has focused on generalising Angluin’s algorithm to various specific automata, there is a parallel line of research that aims to study transition systems more abstractly. In this regard, coalgebras play an important role. In fact, coalgebra is the study of states and their operations and properties [Jac16]. We can think of a coalgebra as a description of a system modelled as a collection of states and a map that defines the transitions between them. It acts as a black box and what we can observe is the behaviour of the system through the transitions of it. Coalgebras focus on the *generated behaviour* that is observable from a system [Rut00; Kur06; Rut98].

On the other hand, modal logic represents another significant field of research. This area defines and studies modal languages that are simple yet expressive and that allow talking about relational structures. Such structures are typically defined as a set together with a collection of relations on that set [BRV01]. Modal logic specifies information about these structures from an internal and local perspective. As described by Blackburn et al. [BRV01], a modal formula can be viewed as a *little automaton* standing at some state in a relational structure, and only permitted to explore the structure by making journeys to neighboring states. So, given that coalgebras act as black boxes where we observe the behaviour of a system through transitions between states, modal logic stands out as a natural way for describing these models. In this direction, significant work has been undertaken both to formalise these behaviours and to develop specific languages, known as coalgebraic modal logics, which are capable of specifying and reasoning about coalgebras [Kli07]. A foundational approach to extending modal logic to the coalgebraic setting was first presented by Moss [Mos99]. Subsequently, several works have been proposed to further refine the reasoning about coalgebras and their logical properties [Jac00; Kur01; Sch05; Pat01; Röß00; Cîr+11; KP11].

1.3 Outline of the thesis and original contribution

It is at the intersection of automata learning, coalgebras, and coalgebraic modal logics that this thesis has been developed. The objective of this work is to apply the coalgebraic setting to the learning techniques, so to provide a generalisation of the learning algorithm applicable to a wide range of state-based systems. This approach also permits a deeper investigation into the connections between learning and the theory of coalgebras. This thesis puts together modal logic, automata learning, and coalgebra theory. By doing so, we aim to uncover or to better understand the structural relationships between these fields and extend the scope of existing learning paradigms. In fact, despite the well-established link between modal logic and coalgebras, relatively little work has been done to formally capture the process of using observations to learn the behaviour of a system in a categorical sense. The connections between coalgebra and learning remain far from being fully understood, a gap that this thesis tries to partially address.

More specifically, our research began with a specific aim: to achieve a deeper understanding of the L^* algorithm by investigating its structure to understand it more abstractly. Our first observation was that the idea behind L^* is closely related to the notion of filtration in modal logic, i.e., a technique used to construct finite models by collapsing a system into a finite number of equivalence classes. Since the L^* algorithm describes how to learn a regular language in the form of a deterministic finite automaton, the use of coalgebras provided a natural framework for our intent.

By combining these two observations, we rephrased the L^* algorithm. More specifically, we viewed the *tests* used in the Angluin algorithm (i.e., words used to distinguish states) as Boolean-valued modal logic formulas. This allowed us to develop a version of the algorithm for coalgebras. While this provided a successful abstraction, it remained somewhat ad-hoc, as it was confined to Boolean-valued logics. To move beyond this, we studied the algorithm from a more categorical perspective using duality theory, which elegantly describes the relationship between coalgebras and their corresponding logics.

This categorical approach generalised our initial work by allowing many-valued tests

that extend the scope of the algorithm to a broader range of transition systems. Also, it clarified the fundamental categorical requirements of the learning process, enabling us to provide comprehensive proofs of correctness and termination. We then took this a step further by showing how effects such as non-determinism and alternation could be incorporated into our framework. The ability to learn non-deterministic finite automata, for instance, allows learning regular languages with more succinct transition structures, potentially reducing the computational effort of the learning process.

Finally, we turned our attention to a minimisation approach that combines reachability closely related to the L^* algorithm, but where the teacher is either absent or operates in the background. We approached it through our logical lens, and we have provided a preliminary investigation into how structural observations can be used to learn system behaviours even in the absence of an external teacher providing a termination proof.

As stated above, during the years, our work and studies followed a natural path. We first consolidated our knowledge on the foundational topics, then we proceeded to rephrase the Angluin L^* algorithm through the lens of coalgebras and modal logics. Later, we examined these concepts via a categorical duality perspective. Lastly, we provided a review of the work by Lee and Yannakakis [LY92], focusing on scenarios where a teacher is absent or operates implicitly. Within this context, we reformulated their algorithm using a modal logic framework and provided formal proofs of its termination.

This thesis reflects the same path and in what follows, we briefly describe the content of the subsequent chapters highlighting the original contribution of them, if present. Part of the original contribution has been already published in two conference papers, some other has been presented for the first time in this thesis.

Background

This chapter provides the reader with the necessary background introducing the basic notions from automata theory (in particular, DFAs and the L^* algorithm), coalgebra and coalgebraic modal logic. By its own nature, this first chapter does not present anything new. The goal is to introduce the main definitions and concepts that will be

used during the subsequent chapters and to fix their notations in order to make the thesis more understandable even for those readers not totally familiar with the topics.

Learning via logic

This chapter is mainly based on our first paper [BK18]. Here, we rephrase the L^* learning algorithm using notions from coalgebras and modal logics. This approach enables a generalization of the L^* algorithm allowing to learn not only DFAs but also any automata that can be expressed as a coalgebra over an arbitrary finitary set functor B . In other words, using logical queries (i.e., membership queries) and conjectures, we can learn regular pointed B -coalgebras.

The generalisation of L^* to a broader class of coalgebras introducing modal logics as the basis for describing the observation constitutes an original contribution.

Learning via duality

This chapter is built on ideas from our second paper [BKR19]. Here, we introduced a dual adjunction framework and the concept of base in categorical terms to explain the L^* algorithm. This viewpoint not only gives us a cohesive way to look at automata theory, but it also opens the door to the discussion of types of automata models, including Mealy machines and labelled transition systems.

This chapter also includes fresh content that has been written over the past few years that has not been published elsewhere. B -coalgebras that can be expressed as BT -coalgebras are the main focus here. The two functors represent different aspects of a system: one tracks state transitions while the other describes its branching behaviour. This dual approach allows us to explore additional state-based systems, such as non-deterministic finite automata (NFAs), under the same framework.

The introduction of the dual adjunction into the framework for the L^* algorithm and its application to BT -coalgebras for learning more complex automata like NFAs within a categorical setting are an original contribution.

Learning via defects

The underlying idea of this chapter is based on [LY92], where an algorithm to compute the minimal and reachable version of a transition system is provided. The idea is closely related to the L^* algorithm, in fact they are both iterative refinement processes but with a key difference: the teacher is missing and it is unclear which aspect of the algorithm makes up for it. In line with the previous chapters, we have established the foundations to restate this algorithm in terms of logic and coalgebras introducing the concept of defect, i.e., a modal formula that identifies and distinguishes some states of the transition system that were previously considered equivalent. This formalisation not only helps to clarify the underlying logical structures of the original approach but also potentially extends the algorithm’s applicability to a wider range of transition systems representable as finite coalgebras.

Exactly this idea of restating the algorithm with a logical formulation that could potentially extend its applicability to a broader class of systems constitutes an original contribution.

Lists of publications

To sum up, the following table outlines the relationship between the chapters of this thesis and the related publications. It also includes original drafts that appear here for the first time.

Chapter	Based on
2	Review of foundational work
3	[BK18]
4	[BKR19] and unpublished draft
5	unpublished draft

Table 1.1: Origin of the presented material.

1.4 Related work

In recent years, several frameworks have been proposed to unify the diverse landscape of automata learning, ranging from common algorithmic interfaces to deep category-

theoretic abstractions. Early attempts at unification, such as the work by Balcázar et al. [Bal+97], investigated learning through a cohesive lens. However, their primary focus remained on the efficiency of algorithms and the structural properties of language classes, rather than transitioning from DFAs to the variety of transition structures that can be expressed via coalgebras.

A significant and fundamental area of research aimed to abstracting the learning algorithm into a categorical framework has been conducted within the Categorical Automata Learning Framework (CALF) project [Hee16; HSS17; HSS20; JS14]. CALF employs category theory to generalise the learning process by varying the underlying base category, e.g., $\mathbf{Set}$, $\mathbf{Vect}$, or $\mathbf{Nom}$, where $\mathbf{Nom}$ represents the category of nominal sets, i.e., sets equipped with a group action in a way that allows them to be finitely representable despite being infinite, via the symmetries exposed by the action [Hee20]. By abstracting the L^* algorithm, the framework demonstrates that learning can be extended to diverse automata types, such as tree automata, weighted automata over semirings, and automata with side-effects determined by a monad. Our approach is orthogonal to CALF: it differs in its primary dimension of generalisation. While CALF focuses on varying the underlying category to encompass different classes of automata, our framework focuses on varying the transition structure while remaining within the category of $\mathbf{Set}$.

Urbat and Schröder [US20] have provided an algebraic approach to automata learning. Their framework focuses on learning automata over a variety of algebraic structures, using monads to handle distinct transition types and generalise learning for formal languages of various types (e.g., infinite words, weighted, and nominal languages). By viewing the process through the duality of algebras and coalgebras, they move beyond the category of $\mathbf{Set}$ to handle quantitative systems. This is different to our work, where the focus is on transition structures and their generalisation via duality within the category of $\mathbf{Set}$. However, the successful application of monads in their work suggests a compelling future direction for our framework: incorporating full monad structures could provide a more standardised method for modelling branching components within the theory of coalgebras [JSS15] as we will see in Chapter 4.

While Wissmann and Milius [Wiß+19] do not provide a specific learning algorithm, their work presents a novel iterative construction of the reachable part of a given pointed coalgebra. This construction is inspired by, and closely resembles, a standard breadth-first search in graphs, and it applies to coalgebras over any well-powered category $\mathbf{C}$ equipped with coproducts and a factorisation system. This method for defining the reachable part of a pointed coalgebra is closely related to the construction we present in Chapter 4; notably, both results were obtained independently and concurrently.

Finally, Colcombet, Petrisan, and Stabile [CPS21] propose a different category-theoretic approach to learning, proposing an algorithm called FunL^* , which does not rely on algebras or coalgebras. Instead, they present a unifying framework that exploits the symmetry of the word automata model, which is reflected in the self-duality of the category of input automata $\mathcal{I}_{A^*}$. This duality underpins the well-known correspondence between observability and reachability. FunL^* provides a generic L^* -like algorithm that can be instantiated for learning deterministic automata, automata weighted over fields, and subsequential transducers by varying the input category. Unlike our approach, which emphasizes the transition structure via coalgebras, FunL^* achieves generality through the categorical properties of the automata themselves, highlighting how reachability and observability can be treated as symmetric concepts in a self-dual setting.

Chapter 2

Background

The aim of this chapter is to introduce some concepts such as automata, coalgebras and coalgebraic modal logics that are essential to understand our work. Readers already familiar with these topics may find the chapter useful for fixing the notation we will use throughout the thesis. Also, we would like to sketch the original algorithm of Dana Angluin to give a comprehensive understanding of the starting point of this work.

The structure of this chapter is as follows: we present the definition of deterministic finite automata and the L^* algorithm (cf. Section 2.1), then we introduce the main coalgebraic notions that are shared across all chapters (cf. Section 2.2) beginning with an example to provide the reader with a more intuitive idea of the topic. Lastly, we introduce the language to express coalgebras, i.e., coalgebraic modal logics (cf. Section 2.3).

2.1 Automata and L^*

2.1.1 Automata

Automata theory is a well-established area of research aiming to study abstract computing machines and their computational capabilities and limitations [HMU00]. This field includes various kinds of automata, such as deterministic finite automata, push-down automata, non-deterministic automata, and Turing machines, to name a few examples [Sip13; Koz97]. In what follows, our primary focus will be on deterministic

finite automata. However, if other types of automata are used in the next chapters, their definitions will be introduced as necessary.

Automata theory is useful to model several types of hardware and software such as lexical analysers, parsers, vending machines and communication protocols [Aho+06; AU72]. These systems are typically represented as states, instantaneous descriptions of their configuration, and transitions that allow to move from a state to another based on some inputs.

Among the various types of automata mentioned, this chapter focuses on the finite automata, i.e., those ones that have only finitely many states and transitions. Moreover, among them, we call deterministic finite automata (DFAs) those ones where all the transitions lead to a single, specific state.

Before providing a technical definition of DFAs, following [HMu00], let's introduce a couple of additional concepts.

Definition 1. *An alphabet A is any finite, non-empty set of symbols. Given an alphabet A , a string (or word) u over A is finite-length sequence of elements of A .*

For example, given $A = \{a, b\}$, a is a string, bb is a string, and the empty sequence is a string itself. This string has zero symbol occurrences, it is called the empty word and denoted by λ . The set of all finite sequences of elements over A is indicated as A^* that is the Kleene closure of the alphabet A .

Strings can be concatenated by putting them together and obtaining a new string. For example, given the alphabet $A = \{a, b\}$ and two strings $u_1 = aa$ and $u_2 = b$ over it, then their concatenation aab is a new string (over A). Where needed, we will use the symbol $\cdot$ to make the concatenation operation more explicit and hence, we will write $aa \cdot b$. In what follows, given two sets of strings U and V , we also use the notation $U \cdot V$ to denote the concatenation of two strings, taking the first from U and the second from V . Moreover, given an alphabet A and the set A^* of all finite sequences of elements over A , given a string $u \in A^*$, we call prefixes all those strings w such that $u = w \cdot z$ for some $z \in A^*$. On the other hand, given a string $u \in A^*$, we call suffixes all those strings w such that $u = z \cdot w$ for some $z \in A^*$. For example, given the string aab , we say that λ , a and aa are the prefixes of the string and λ , ab and b are the suffixes. Also,

Chapter 2. Background

given a set of strings $S \subseteq A^*$, this set is *prefix-closed*, if for every string in the set, all of its prefixes are also in the set. The set is *suffix-closed*, if for every string in the set, all of its suffixes are also in the set.

Definition 2 (Language). *Given an alphabet A and a subset L of A^* , $L \subseteq A^*$, we call L a language over A .*

Intuitively, a language L is a set of specific strings obtained from an alphabet.

Example 3. Given $A = \{a, b\}$ as alphabet, the set of those strings whose number of a 's can be divided by 3 is a language L and it is a subset of A^* . More formally, we can express it as follows: $L := \{u \in \{a, b\}^* \mid \# \text{ a's mod } 3 = 0\} \subseteq \{a, b\}^*$.

Now, we introduce the definition of deterministic finite automata.

Definition 4 (Deterministic finite automata). *A deterministic finite automaton (DFA) D is a 5-tuple, $(Q, A, \delta, q_0, F \subseteq Q)$, where Q is a finite set of states, A is a finite, non-empty set called the input alphabet, $\delta: Q \times A \rightarrow Q$ represents the transition function, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of the accepting or final states.*

The transition function δ can be also extended to accept not only one single input but a sequence of them. The extended transition function, denoted as $\hat{\delta}: Q \times A^* \rightarrow Q$, is defined by induction from δ on the length of the input strings [Koz97]: given $u = \lambda$ then $\hat{\delta}(q, \lambda) = q$, i.e., if we are in a state q and we read no inputs, we stay in q . Given $u = w \cdot a$, then $\hat{\delta}(q, u) = \delta(\hat{\delta}(q, w), a)$, i.e., the state we achieve after reading a sequence of symbols is the state that we achieve from a previous state.

DFAs are used to describe and recognise languages. A particular language can be seen as the set of all words accepted by a specific DFA, meaning those words that when given as input to the DFA, it reaches an accepting state starting from its initial state.

More formally, we have the following.

Definition 5 (Language accepted by a DFA). *Given a DFA $D = (Q, A, \delta, q_0, F)$, a string $u \in A^*$ is accepted by D , if $\hat{\delta}(q_0, u) \in F$. The language accepted by D , denoted by $L(D)$, is the set of all strings $u \in A^*$ that are accepted by D .*

For instance, given the language of the Example 3, a DFA that describes it can be represented as per Figure 2.1: starting from q_0 , all those transitions that terminate in q_0 after reading a specific string u establish whether u is in the language or not. Moreover, this DFA is also the minimal one that accepts that language. That is, we cannot represent the same language with another DFA with fewer states than this one.

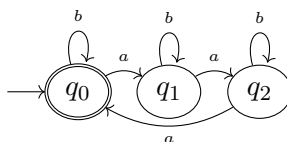


Figure 2.1: Minimal DFA accepting the language of Example 3.

The languages that can be accepted by a DFA are called regular languages [HMU00] and the language in the example above is a regular one. Regular languages are also referred to as regular sets.

2.1.2 Learning regular sets

In [Ang87], Angluin presents an algorithm called L^* . This algorithm efficiently learns an initially unknown regular set L by learning a minimal deterministic finite automaton by asking two types of questions to an oracle called a *Teacher*:

- *Membership queries*: given a specific string, they return *yes*, if the string is part of the language the algorithm is trying to learn, *no* otherwise.
- *Conjecture*: based on the answers to the above-mentioned questions, the L^* algorithm builds up a conjecture, i.e., a description of L . The *Teacher* will answer *yes* whether the conjecture is the one describing the language. Otherwise, it will provide a new string that is accepted by L but not by the conjecture or vice versa.

Learning a regular set from examples was not a new concept and this is not the main contribution of the Angluin paper. Indeed, previous works had been done in this direction [Gol78; AS83; Ang81]. What is important here is the introduction of the notion of a *Teacher*, sometimes called *Oracle*, that can answer specific questions. As highlighted

earlier in the introduction, this query-based approach opened the field of active learning where the computational complexity of automata learning is decreased [Ang87].

To learn a regular set, the L^* algorithm organises the information it gets from the two queries mentioned before under an observation table (S, E, T) . This observation table is defined by two sets of strings S and E where S is a prefix-closed set and E is suffix-closed one (as defined in the previous section at page 14) and a function mapping T (for table) defined as $T: (S \cup S \cdot A) \cdot E \rightarrow \{0, 1\}$ that tracks the membership query outputs. That is, it maps a string from the set S , possibly concatenated with a symbol from the alphabet A and a string from E to 1, if the resulting string is contained in the regular set, 0 otherwise. That is, $T(s \cdot e) = 1$ if and only if $s \cdot e \in L$. Also, given a string $s \in (S \cup S \cdot A)$, we use T to define the function $row(s): E \rightarrow \{0, 1\}$ as $row(s)(e) := T(s \cdot e)$.

The observation table can be thought of as a matrix as presented in Table 2.1 where the rows in the upper part of the matrix are indexed by strings from the set S , the rows in the lower part are indexed by strings from S concatenated with a letter of the alphabet A , i.e., $S \cdot A$, and the columns are strings from the set E . The entries of the matrix are 1's or 0's according to the function map T .

	λ	a
λ	1	0
a	1	1
b	0	1
aa	1	1
ab	1	0
ba	1	0
bb	1	0

Table 2.1: Example of an observation table.

From the observation table, the algorithm builds up a conjecture in the form of a DFA $D = (Q, A, \delta, q_0, F)$, where its components are defined as follows:

- **States (Q):** The set of states includes all the unique rows in the observation table that correspond to strings in S , i.e., $Q = \{row(s) \mid s \in S\}$.
- **Alphabet (A):** The alphabet A is identical to that of the target language.

- **Transition function (δ):** For any state $row(s)$ and input symbol a , the transition function is defined as $\delta(row(s), a) = row(s \cdot a)$.
- **Initial state (q_0):** The initial state is the row associated with the empty string, $q_0 = row(\lambda)$.
- **Final states (F):** Since strings and states overlap in this framework, if a string s when concatenated with the empty string is accepted by the target language (represented by $T(s \cdot \lambda) = 1$), then its corresponding state $row(s)$ is an accepting state, i.e., $F = \{row(s) \mid s \in S \text{ and } T(s) = 1\}$.

This DFA is well-defined when the observation table satisfies the closedness and consistency properties, which we detail next [Ang87]. If the given conjecture is correct, we are done and we have obtained the minimal DFA describing L , otherwise the *Teacher* will provide a counterexample, i.e., a string u that will be added to E together with all its suffixes [Bol+09]. Please note that in presenting the L^* algorithm, we introduce a slight variation from its standard presentation that we will present after the definition of consistency.

The algorithm provides a conjecture only when the observation table satisfies two properties: *closedness* and *consistency*, and during all its runs, the aim of the algorithm is indeed to keep the table *closed* and *consistent*. Intuitively, for a table being closed means that the set S already contains all and the only necessary states that we need to build up the correct DFA. Being consistent means that all those elements that we consider as equal actually behave the same, i.e., their successors are the same and these are both accepting or not. More formally, we have the following.

Definition 6 (Closedness). *An observation table is closed when $\forall u \in (S \cdot A), \exists s \in S$ such that $row(u) = row(s)$.*

Definition 7 (Consistency). *An observation table is consistent when given $s_1, s_2 \in S$ such that $row(s_1) = row(s_2)$ then $\forall a \in A, row(s_1 \cdot a) = row(s_2 \cdot a)$.*

Whenever a table is not closed, we add the string $u \in S \cdot A$ (for which $row(u)$ has no match in S) to the set S , i.e., to the upper part of the observation table. In

contrast, consistency is handled differently: when a counterexample is provided, it and all its suffixes are added to the set E . In fact, as presented in [MP95], instead of adding the counterexample and its prefixes to the set S , it can be added together with its suffixes to E , keeping the table always consistent. We decided to present L^* with this optimisation because in the next chapter, where we present our L^{co} algorithm, the consistency property, which we will call sharpness, is always preserved throughout the execution of the algorithm, making the comparison much closer.

as mentioned before.

Importantly, one of the fundamental theorems of regular languages on finite words is the Myhill-Nerode theorem, stating a one-to-one correspondence between the states of the minimal DFA for a language L and the equivalence classes of its *right congruence*, i.e., an equivalence relation $\sim$ on strings in A^* that is compatible with right-concatenation, meaning that $u \sim v$ implies $u \cdot a \sim v \cdot a$ for any symbol $a \in A$. The relationship between the equivalence classes of the right congruence and the states of a minimal DFA is at the heart of minimisation and learning algorithms for regular languages of finite words and in the context of L^* , the equivalence relation defined by matching table rows ($\text{row}(s_1) = \text{row}(s_2)$) serves as a finite approximation of this right congruence. As the algorithm refines the observation table through closedness checks and counterexample additions, these row equivalence classes progressively refine until they coincide with the Myhill-Nerode classes of the target language yielding the unique minimal DFA recognising the target regular set [AF18].

Before giving an example to make explicit all the definitions and the process described above, we sketch the L^* algorithm via Algorithm 1.

Example 8. Recalling the language presented in the Example 3, we run L^* and try to learn it.

Given an alphabet $A = \{a, b\}$, the initially unknown language we want to learn is $L = \{u \in \{a, b\}^* \mid \text{number of } a\text{'s mod } 3 = 0\}$. The initial observation table is given in Table 2.2.

The table is not closed as we have $\text{row}(\lambda) \neq \text{row}(a)$. So, we need to add a to the upper part of the table obtaining what exemplified in Table 2.3.

Algorithm 1 Sketch of the L^* algorithm

- 1: Initialize S and E to $\{\lambda\}$
 - 2: Construct the initial observation table (S, E, T) asking membership queries for λ and all the $a \in A$
 - 3: **repeat**
 - 4: **while** (S, E, T) is not closed **do**
 - 5: Find $s_1 \in S$ and $a \in A$ such that $\forall s \in S$ we have $row(s) \neq row(s_1 \cdot a)$,
 - 6: Add $s_1 \cdot a$ to S ,
 - 7: Extend T to $(S \cup S \cdot A) \cdot E$ using membership queries.
 - 8: **end while**
 - 9: Transform the observation table (S, E, T) into the DFA $D(S, E, T)$,
 - 10: Propose D as the conjecture.
 - 11: **if** The Teacher replies with a counterexample t **then**
 - 12: Add t and all its suffixes to E ,
 - 13: Extend T to $(S \cup S \cdot A) \cdot E$ using membership queries.
 - 14: **end if**
 - 15: **until** The Teacher replies *yes* to the conjecture.
 - 16: **return** D .
-

	λ
λ	1
a	0
b	1

Table 2.2: Observation table at step 0.

	λ
λ	1
a	0
b	1
aa	0
ab	0

Table 2.3: Observation table at step 1.

The table is both closed and consistent and we can provide a conjecture consisting of an automaton with two states as per Figure 2.2.

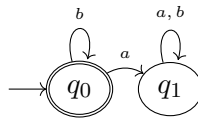


Figure 2.2: Conjecture in the form of a DFA.

	λ	a	aa	aaa
λ	1	0	0	1
a	0	0	1	0
b	1	0	0	1
aa	0	1	0	0
ab	0	0	1	0

Table 2.4: Observation table at step 2.

This is not the right conjecture, so we receive a counterexample in the form of the string aaa . Please note that aaa is just one of the possible counterexamples the teacher could provide since a counterexample is by definition any string that is accepted by the language but not by the conjecture or vice versa. We add the string aaa and its suffixes to the set E and we obtain a new observation table as per Table 2.4.

Once again, the table is not closed because, following the definition of closedness, we can find a string $u \in S \cdot A$ such that neither $row(\lambda) = row(u)$ nor $row(a) = row(u)$. This u is the string aa in the lower part of the table. So, we add aa to S and we obtain a closed table as per Table 2.5. In fact, all the elements in $S \cdot A$ have a representative in S .

	λ	a	aa	aaa
λ	1	0	0	1
a	0	0	1	0
aa	0	1	0	0
b	1	0	0	1
ab	0	0	1	0
aaa	1	0	0	1
aab	0	1	0	0

Table 2.5: Observation table at step 3.

Now, the observation table is closed and consistent and hence, we can provide the teacher with another conjecture described by the DFA we presented in Figure 2.1 at page 15. It is the right conjecture and we are done.

Please be aware that the Teacher that is mentioned in the Angluin paper is not really a person that is able to answer the two types of questions we mentioned, but it is a way to emphasize the following intuition: there is a minimum amount of information

and feedback required for the algorithm to guarantee correct and efficient learning.

2.2 Coalgebras

In this section, we introduce the notion of coalgebra and its related key concepts. We will first provide an intuitive understanding of them restating the definition of DFAs we discussed in the previous section according to a different perspective as in [Rut98]. Once this intuition is given, we will move to a more general notion of coalgebra, extending the DFA specific case to the abstract definition. In depth discussions about coalgebras can be found in [Rut00; JR97; Jac16].

Also, we will take some basic notions of category theory for granted when we talk about coalgebras. However, the reader can refer to [AHS90; Awo10; Lei14; Sim11; BW95] for a detailed introduction to the topic or a deep explanation of it.

2.2.1 Introducing coalgebras via an example

Instead of being defined as a 5-tuple as we stated in the previous section, a DFA can be alternatively defined as a triple, as follows.

Definition 9. *Given an alphabet A , a deterministic finite automaton (DFA) is a 3-tuple, (S, o, t) , consisting of a finite set S of states, an output function $o: S \rightarrow 2$ and a transition function $t: S \rightarrow S^A$.*

In this definition, 2 represents the set $\{0, 1\}$. The output function o takes a state $s \in S$ and returns whether it is an accepting state or not, $o(s) = 1$ or $o(s) = 0$, respectively. The transition function t assigns to each state $s \in S$ a function $t(s): A \rightarrow S$. This function receives an alphabet symbol $a \in A$ and returns the successor state of s after reading the input a . Thus, t precisely captures the transitions within the automaton.

If we do not consider the initial state, the definition of DFA given in Section 2.1 is equivalent to the latter because of the existence of the following two bijections [Rut98] where $P(S)$ denotes the powerset of S , i.e., the collection of all subsets of S :

$$P(S) \cong (S \rightarrow 2) \text{ and } (S \times A \rightarrow S) \cong (S \rightarrow S^A)$$

Chapter 2. Background

The first bijection establishes a one-to-one correspondence between the set of accepting states and the output function. The set of accepting states is a subset of all states, whereas the output function maps each state to 1 if it is accepting, or to 0 if not. The bijection ensures that for every possible set of accepting states, there is one unique output function o that represents it, and vice versa. The second bijection is a concept well-known as currying [CF58]. The two forms used to define transitions are just different syntactic ways to express the same underlying information.

If both the alphabet A and the set of states S are finite, we refer to (S, o, t) as a finite automaton.

Given this alternative formulation of the definition of DFAs, we define a bisimulation as a binary relation $R \subseteq S \times S'$ between two DFAs $S = (S, o, t)$ and $S' = (S', o', t')$ when $\forall (s, s') \in R$, we have:

$$sRs' \Rightarrow \begin{cases} o(s) = o'(s'), \text{ and} \\ \forall a \in A, t(s)(a)Rt'(s')(a) \end{cases}$$

Intuitively, two states are related by a bisimulation whether they have the same observable output (i.e., accept or reject) and their successor states remain in the relation.

As stated in [Rut98], a bisimulation between S and itself is called a bisimulation *on* S , union and composition of bisimulations are also bisimulations. The greatest bisimulation on S , obtained by taking the union of all bisimulations on S , is called the bisimilarity relation and it is denoted by $s \sim s'$.

Intuitively, this new definition of DFAs and the bisimulation relation together say that any automaton is like a black box where it is only possible to observe whether a state is accepting or not, and what is the state transition based on a specific input. The bisimulation captures the notion of observational indistinguishability: two states are bisimilar if they have the same observational behaviour (in coalgebraic terms, we do not speak about identity, but more about behavioural equivalence).

Furthermore, given two sets of states, S and S' , from two DFAs and a function f between them, $f: S \rightarrow S'$, we call such function *morphism* when $\forall s \in S, o(s) = o'(f(s))$ and $\forall a \in A, f(t(s)(a)) = t'(f(s))(a)$. Morphisms are structure-preserving maps, i.e.,

they map states in a way that preserves both the output behaviour and the transition structure.

Moreover, given two automata $S = (S, o, t)$ and $S' = (S', o, t)$, if $S' \subseteq S$ and the inclusion function $i: S' \rightarrow S$ is a morphism, then S' is said to be a *subautomaton* of S . We indicate as $\langle s \rangle$ the subautomaton generated by any state $s \in S$: it is the smallest subautomaton of S containing s , and can be obtained by including all states from S that are reachable via a finite number of transitions from s .

Also with this formulation of DFAs, it is possible to define those languages accepted by a DFA. Given a DFA $S = (S, o, t)$ and a state $s \in S$, the language accepted by s , denoted by $L_S(s)$ is defined as the set of all those $u \in A^*$ that read from the state s lead to an accepting state. More formally, $L_S(s) = \{u \in A^* \mid o(t^*(s)(u)) = 1\}$, where $t^*(s)(u)$ is the function obtained from $t(s): A \rightarrow S$ extending it to strings inductively on the length of the string.

Moreover, since a language is a set (of words), it is quite natural to consider the set of all languages. We can define it as $\mathcal{L} = \{L \mid L \subseteq A^*\}$. This set can be equipped with an automaton-like structure. To do so, we introduce the concept of derivative, originally proposed by [Brz64; BS86; Con71] and elaborated in a coalgebraic context afterwards in [Jac16; Rut00]. Intuitively, a derivative describes how a system evolves looking at the possible successor states. In our case, given an input symbol $a \in A$, we call a -derivative the set of all those words u such that concatenated with a , i.e., $a \cdot u$, form the language L . More formally, $L_a = \{u \in A^* \mid a \cdot u \in L\}$. The concept of a -derivative can be extended to words such that given $u \in A^*$, $L_u = \{v \in A^* \mid u \cdot v \in L\}$. It is possible to give an automaton-like structure to the set of all possible languages $\mathcal{L}$ which is not finite by definition as $\mathcal{L} = (\mathcal{L}, o_{\mathcal{L}}, t_{\mathcal{L}})$ where the set of states is the set of all languages as above, the output function and the transition function are defined as follows.

$$o_{\mathcal{L}}(L) = \begin{cases} 1 & \text{if } \lambda \in L \\ 0 & \text{if } \lambda \notin L \end{cases}, \text{ and}$$

$$t_{\mathcal{L}}(L)(a) = L_a$$

That is, the output function provides an immediate observation about the language L checking if the empty word is in the language. This tells us whether L itself represents an accepting state in $\mathcal{L}$. The transition function is obtained using the concept of derivative to define the successor state.

An interesting property is that minimal DFAs can arise as a finite subautomaton within $\mathcal{L}$ and given two or more DFAs describing the same language, they will always be identified with the same finite subautomaton in $\mathcal{L}$. The finite subautomaton exactly describes the behaviour of the DFAs.

2.2.2 Coalgebras and their general framework

All those concepts we saw in the example above have been generalised into the more abstract framework of coalgebras by [Rut00; JR97; Jac16].

Definition 10 (Coalgebra). *Given the category $\mathbf{Set}$ of sets and an endofunctor $B: \mathbf{Set} \rightarrow \mathbf{Set}$, a coalgebra is a pair (X, γ) where $X \in \mathbf{Set}$ is a set of states and $\gamma: X \rightarrow BX$ is a coalgebra map (or successor function).*

It is easy to see how this general definition applies to our earlier automata definition. In fact, a DFA $S = (S, o, t)$ can be translated into coalgebraic terms defining the functor B as $B(S) = 2 \times S^A$ (where B takes a state $s \in S$ and returns a pair whose first component specifies whether the state s is accepting or not and the second component returns the successor state in the form of a function) and combining the two functions $o: S \rightarrow 2$ and $t: S \rightarrow S^A$ into a single coalgebra map $\langle o, t \rangle: 2 \times S^A$ that maps an element $s \in S$ to the pair $\langle o(s), t(s) \rangle$. Indeed, the previous definition of DFA is a specific instance of a B -coalgebra.

Having established the general notion of coalgebra, we now restate both the definition of structure-preserving map and the one of behavioural equivalence.

Definition 11 (Morphism). *Given two B -coalgebras (X, γ) and (Y, δ) , a B -coalgebra morphism $f: (X, \gamma) \rightarrow (Y, \delta)$ is a function $f: X \rightarrow Y$ such that the following diagram*

Chapter 2. Background

commutes.

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ \gamma \downarrow & & \downarrow \delta \\ BX & \xrightarrow{Bf} & BY \end{array}$$

Intuitively, this diagram means that if we apply to a given state $s \in X$ the coalgebra map γ and then the function Bf , it returns the same result as applying f first and then the coalgebra map δ to $f(x)$.

Now, we generalise another important concept already sketched in the DFA example when the set of all languages has been mentioned, namely the notion of final coalgebra. The underlying idea of a final coalgebra is that it provides a formalisation of a behaviour of a system. It stands for the most abstract and minimal representation of such behaviour and for a specific functor, it captures all the possible behaviours for it. This minimality implies that no two distinct states within the final coalgebra are behaviourally equivalent.

Definition 12 (Final coalgebra). *Given an endofunctor $B: \mathbf{Set} \rightarrow \mathbf{Set}$, a final B -coalgebra is a final object in the category of $\mathbf{Coalg}(B)$. That is, it is a coalgebra (Z, ζ) such that for any coalgebra (X, γ) , there is a unique coalgebra morphism $f: (X, \gamma) \rightarrow (Z, \zeta)$ as depicted in the following diagram.*

$$\begin{array}{ccc} X & \xrightarrow{\quad f \quad} & Z \\ \gamma \downarrow & & \downarrow \zeta \\ BX & \xrightarrow{\quad Bf \quad} & BZ \end{array}$$

This unique morphism is often referred to as the behaviour map because it maps each state from (X, γ) to its corresponding behaviour in the final coalgebra. That is, the final B -coalgebra, if exists, contains for each state $x \in X$ of an arbitrary coalgebra exactly one state that is behaviourally equivalent to x . If f goes from the final coalgebra to itself, due to the uniqueness, f is the identity morphism.

Having established the notions of a coalgebra, coalgebra morphism, and final coalgebra, we now introduce the coalgebraic notion of behavioural equivalence as follows.

Definition 13 (Behavioural Equivalence). *Let $B: \mathbf{Set} \rightarrow \mathbf{Set}$ be an endofunctor and*

Chapter 2. Background

$(X_1, \gamma_1), (X_2, \gamma_2)$ be B -coalgebras. We say that two states $x_1 \in X_1$ and $x_2 \in X_2$ are behaviourally equivalent if there are a B -coalgebra (Y, δ) and B -coalgebra morphisms $f_1: (X_1, \gamma_1) \rightarrow (Y, \delta)$ and $f_2: (X_2, \gamma_2) \rightarrow (Y, \delta)$ such that $f_1(x_1) = f_2(x_2)$. That is, the following diagram commutes.

$$\begin{array}{ccccc} X_1 & \xrightarrow{f_1} & Y & \xleftarrow{f_2} & X_2 \\ \gamma_1 \downarrow & & \downarrow \delta & & \downarrow \gamma_2 \\ BX_1 & \xrightarrow{Bf_1} & BY & \xleftarrow{Bf_2} & BX_2 \end{array}$$

Intuitively, two states are behaviourally equivalent if a coalgebra morphism can identify them. When two states x_1 and x_2 are behavioural equivalent, we write $x_1 \Leftrightarrow_B x_2$ or simply $x_1 \Leftrightarrow x_2$. Also, if two systems are behaviourally equivalent they are mapped to the same behaviour in the final coalgebra.

Moreover, we can extend the definition of coalgebra specifying a state to be thought of as the initial state. In such cases, we talk about *pointed* coalgebra. In particular, a *pointed* B -coalgebra is a triple (X, γ, x) where (X, γ) is a B -coalgebra as defined before and $x \in X$ is a distinguished point. A morphism $f: (X, \gamma, x) \rightarrow (Y, \delta, y)$ is a B -coalgebra morphism $f: (X, \gamma) \rightarrow (Y, \delta)$ such that $f(x) = y$ and two pointed coalgebras are behaviourally equivalent if $x \Leftrightarrow y$. DFAs where each state is reachable from the initial state can be expressed as pointed B -coalgebras for the functor $B = 2 \times S^A$. Indeed, their initial state can naturally be expressed by the distinguished point x of the coalgebra. Also, the requirement that every state in the automaton is reachable ensures that the carrier set coincides precisely with the subcoalgebra generated by that initial state.

2.2.3 Example of coalgebraic structures

Before concluding this section, to have a better and a more general understanding of the abstract notions we discussed so far, we now present two additional concrete examples: streams, which represent infinite sequences of observations and Kripke frames, which are fundamental in modal logics.

Streams as Coalgebras

Streams can be used to model the behaviour of reactive and concurrent systems, representing infinite sequences of elements to be thought of as outputs or observations [Rut00]. Formally, given an alphabet A , an infinite stream over A is defined as a function $\sigma: \mathbb{N} \rightarrow A$ which maps natural numbers (as time steps) to elements of the set A . A stream σ can be thought of as an infinite sequence $(\sigma(0), \sigma(1), \sigma(2), \sigma(3), \dots)$.

We can decompose a stream into its head and its tail. Given the sequence representation above, the head of the stream, i.e., its first element, is $\sigma(0)$ and the tail, the remainder of the stream is $(\sigma(1), \sigma(2), \sigma(3), \dots)$. This decomposition illustrates the behaviour of the stream where at each step it is possible to both observe one element of the system and perform a transition to a new state.

Coalgebra can be used to describe this type of systems. Indeed, given an arbitrary set of symbols A and an endofunctor $B: \mathbf{Set} \rightarrow \mathbf{Set}$ defined as $BX = A \times X$, a stream coalgebra is a pair (X, γ) where X is a set of states and $\gamma: X \rightarrow BX$ is the coalgebra map defined for any states $x \in X$ as $\gamma(x) = (h(x), t(x))$, i.e., a pair whose first component represents the element we are observing and the second component is a stream itself.

A morphism between two stream coalgebras (X, γ) and (Y, δ) is a function $f: X \rightarrow Y$ that preserves the coalgebraic structure. That is, a morphism such that the following diagram commutes.

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ \gamma \downarrow & & \downarrow \delta \\ BX & \xrightarrow{id \times f} & BY \end{array}$$

This diagram means that for every $x \in X$, we have $(id \times f)(\gamma(x)) = \delta(f(x))$.

As already mentioned, intuitively a morphism between two streams ensures that the behaviour is preserved and that they are observationally indistinguishable.

Moreover, for the functor $B = A \times X$, the final coalgebra exists and is given by (A^ω, ζ) , where A^ω is the set of all infinite streams over A . The coalgebra map $\zeta: A^\omega \rightarrow A \times A^\omega$ is defined as $\zeta(\sigma(0), \sigma(1), \sigma(2), \sigma(3), \dots) = (\sigma(0), (\sigma(1), \sigma(2), \sigma(3), \dots))$, i.e., given an infinite stream, the coalgebra map maps it into a pair whose first component

is the head of the stream and the second component is the tail. This map is usually called the unfold map and it captures the unfolding behaviour of any stream. Given a stream, if we apply the unfolding map step by step and we subsequently read the outputs, we obtain the stream itself.

Kripke frames as Coalgebras

Kripke frames are commonly used in modal logic to model possible worlds and accessibility relations between them. A standard Kripke frame is a pair (W, R) where W is a set of states (or worlds) and $R \subseteq W \times W$ is a binary relation that establishes which states are accessible from any given state within the frame [BRV01].

This can be modelled as coalgebras as follows. Consider the power set functor $P: \mathbf{Set} \rightarrow \mathbf{Set}$ that maps a set X to its power set $P(X)$ and a function $f: X \rightarrow Y$ to $P(f): P(X) \rightarrow P(Y)$ defined as $P(f)(U) = \{f(z) \mid z \in U\}$ for any $U \subseteq X$. A Kripke frame (W, R) can be represented as a P -coalgebra (W, γ_R) where, again, W is the set of states and γ_R maps an element of $w \in W$ to the set of all states that can be seen from w , i.e., $\gamma_R(w) = \{w' \in W \mid (w, w') \in R\}$.

A morphism between two coalgebraic Kripke frames (W_1, γ_{R_1}) and (W_2, γ_{R_2}) is a function $f: W_1 \rightarrow W_2$ such that $Pf \circ \gamma_{R_1} = \gamma_{R_2} \circ f$. This means that for every $w \in W_1$, if w' is accessible from w , $(w, w') \in R_1$ then $f(w')$ must be accessible from $f(w)$, $(f(w), f(w')) \in R_2$. This is the standard notion of bounded morphism in modal logic [Kur06].

The final coalgebra for the power set functor P does not exist in $\mathbf{Set}$.

2.3 Coalgebraic modal logic and dual adjunction

2.3.1 Coalgebraic modal logic

Modal logics provide simple yet expressive languages for describing and reasoning about relational structures. They offer a good balance between expressive power and low computational complexity for their decidability [BRV01; Var97].

As we have seen in the previous sections, coalgebras of the form $(X, \gamma: X \rightarrow BX)$

(over the category $\mathbf{Set}$) serve as generic mathematical framework for describing state-based transition systems whose concrete structure is defined by the functor $B: \mathbf{Set} \rightarrow \mathbf{Set}$. We can think of the functor as the type of transition or behaviour that the system performs. In this setting, coalgebraic modal logics establish as a natural way to describe these transition systems. These logics are parametric in the type of transition. That is, they are generic enough to be applied to any system that is representable by a B -coalgebra. So, we can build a modal language that allows us to reason about and describe the properties and the characteristics of our system [KP11].

To define the right language for a given transition type, on the syntactic side, we need to extend the standard propositional modal logic with a collection of modal operators, according to the specific transition type we would like to describe. On the semantic side, we must specify a way to interpret the modal operators we defined over arbitrary B -coalgebras.

There are different formal approaches to do so. One is the relation lifting approach described by Moss in [Mos99], where the relations are lifted to logical equivalences. This part is not under the scope of the thesis. Another way to deal with it, the approach we are embracing throughout the thesis, is the predicate lifting approach [Pat03] where the focus is on how properties (predicates) are transformed across states and their successors.

To define and extend classical modal logic to describe transition systems as coalgebras, we need to choose a collection of modal operators according to our needs. This collection is known as (modal) similarity type. The definition of similarity type requires also the use of the contravariant powerset functor $Q: \mathbf{Set} \rightarrow \mathbf{Set}^{op}$ that maps a set X to the collection PX of its subsets of X , and a function $f: X \rightarrow Y$ to the inverse image function $Qf = f^{-1}: QY \rightarrow QX$ where given $V \subseteq Y$, we have $f^{-1} = \{x \in X \mid f(x) \in V\}$.

Definition 14 (Similarity Type). *A (modal) similarity type is a set Λ of modal operators with arities. If Λ is a similarity type, a Λ -structure consists of an endofunctor $B: \mathbf{Set} \rightarrow \mathbf{Set}$, together with an assignment of an n -ary predicate lifting, that is, a natural transformation of type $\llbracket \heartsuit \rrbracket : (Q)^n \rightarrow Q \circ B$ where $Q: \mathbf{Set} \rightarrow \mathbf{Set}^{op}$ is the*

Chapter 2. Background

contravariant powerset functor, to every n -ary operator $\heartsuit \in \Lambda$.

Given a modal similarity type Λ and a corresponding Λ -structure (in other words, a functor B and a suitable predicate lifting for each operator in Λ), we can define both the syntax and the semantics of our logic. The syntax of the formulas is that of the standard propositional modal logic, extended with elements of Λ . The semantics is defined by providing the set of states $U \subseteq X$ of our coalgebra in which a formula φ is true. We indicate that set as $\llbracket \varphi \rrbracket_{(X, \gamma)}$. The semantics for the modal operators is given by the assigned predicate lifting. Intuitively, it is a way to lift a logical property from the successor states of a system back to the current state.

The formal definition is as follows.

Definition 15. *The language induced by a modal similarity type Λ is the set $\mathcal{F}(\Lambda)$ of formulas given by*

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

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

$$\begin{aligned} \llbracket \top \rrbracket_{(X, \gamma)} &= X \\ \llbracket \varphi \wedge \psi \rrbracket_{(X, \gamma)} &:= \llbracket \varphi \rrbracket_{(X, \gamma)} \cap \llbracket \psi \rrbracket_{(X, \gamma)} & \llbracket \neg \varphi \rrbracket_{(X, \gamma)} &:= X \setminus \llbracket \varphi \rrbracket_{(X, \gamma)}, \\ \llbracket \heartsuit(\varphi_1, \dots, \varphi_n) \rrbracket_{(X, \gamma)} &:= (Q\gamma \circ \llbracket \heartsuit \rrbracket_X)(\llbracket \varphi_1 \rrbracket_{(X, \gamma)}, \dots, \llbracket \varphi_n \rrbracket_{(X, \gamma)}). \end{aligned}$$

Instead of $x \in \llbracket \varphi \rrbracket_{(X, \gamma)}$ we will often write $(X, \gamma) \models \varphi$. In words, “the coalgebra (X, γ) satisfies φ ”.

Example 16. Recalling our definition of DFAs modelled as coalgebra $(X, \gamma: X \rightarrow BX)$ where $B = 2 \times X^A$ with A the set of input. A modal similarity type for this coalgebra includes two modal operators: a nullary propositional constant Acc , i.e., a nullary operator that holds only in the accepting states and a unary modal operator $\langle a \rangle$ for each input symbol $a \in A$, where $\langle a \rangle \varphi$ holds if the state reached by the input a satisfies φ .

Chapter 2. Background

The predicate lifting related to those modal operators is defined as follows: $\lambda_{\text{Acc}} = \{(i, f) \in 2 \times X^A \mid i = 1\}$ and $\lambda_{\langle a \rangle} U = \{(i, f) \in 2 \times X^A \mid f(a) \in U\}$ given $U \subseteq X$. Intuitively, predicate lifting defines what conditions the successor structures must meet for a specific formula to hold regardless a specific state or set of states.

The language induced by this modal similarity type is:

$$\mathcal{F}(\Lambda) \ni \varphi, \psi ::= \text{Acc} \mid \varphi \wedge \psi \mid \neg \varphi \mid \langle a \rangle \varphi. \forall a \in A.$$

Given the coalgebra map γ of the DFA defined as $\gamma(x) = (o, t(x))$ where $t(x): A \rightarrow X$, and o defined as usual, the semantics for the language is defined as follows:

$$\begin{aligned} \llbracket \text{Acc} \rrbracket &= \gamma^{-1}(\lambda_{\text{Acc}}()) \\ &= \{s \in S \mid \gamma(s) \in \lambda_{\text{Acc}}()\} \\ &= \{s \in S \mid (o(s), t(s)) \in \{(i, f) \mid i = 1\}\} \\ &= \{s \in S \mid o(s) = 1\} \end{aligned}$$

and

$$\begin{aligned} \llbracket \langle a \rangle \varphi \rrbracket &= \gamma^{-1}(\lambda_{\langle a \rangle}(\llbracket \varphi \rrbracket)) \\ &= \{s \in S \mid \gamma(s) \in \lambda_{\langle a \rangle}(\llbracket \varphi \rrbracket)\} \\ &= \{s \in S \mid (o(s), t(s)) \in \{(i, f) \mid f(a) \in \llbracket \varphi \rrbracket\}\} \\ &= \{s \in S \mid t(s)(a) \in \llbracket \varphi \rrbracket\} \end{aligned}$$

The semantics for the other operators ($\wedge, \neg$) remain as usual, since they are standard boolean operations on the set of states.

The logics give rise to the notion of logical equivalence.

Definition 17. Let (X, γ) and (Y, δ) be B -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 with respect to Σ if for all formulas $\varphi \in \Sigma$ we have $x \models \varphi$ if and only if $y \models \varphi$. 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 $|x|$.

Chapter 2. Background

The semantics of formulas of coalgebraic modal logics 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 [Sch05].

Definition 18. *The language $\mathcal{F}(\Lambda)$ is called expressive if for B -coalgebras (X, γ) and (Y, δ) and all $x \in X, y \in Y$ we have $x \equiv y$ if and only if $x \xleftrightarrow{\quad} y$.*

For a finitary set functor B we are always able to find an expressive language [Sch05].

Moreover, we introduce the definition of separating, i.e., a condition that ensures expressivity [Pat04; Sch05].

Definition 19 (Separatingness). *Let $B: \mathbf{Set} \rightarrow \mathbf{Set}$ and let Λ be a set of predicate liftings for B . We say Λ is separating if for all $X \in \mathbf{Set}$ and all $t_1, t_2 \in BX$ with $t_1 \neq t_2$, there exists $\lambda \in \Lambda$, there exists subset $U_1 \subseteq X, \dots, U_n \subseteq X$ s.t. either $t_1 \in \lambda_X(U_1, \dots, U_n)$ & $t_2 \notin \lambda_X(U_1, \dots, U_n)$ or $t_1 \notin \lambda_X(U_1, \dots, U_n)$ & $t_2 \in \lambda_X(U_1, \dots, U_n)$.*

2.3.2 Dual adjunction

So far, we have seen how the semantics of modal formulas can be defined by elements of a given set of states. Given X and Φ , arbitrary sets of states and formulas, respectively, the definition of semantics can explicitly be seen as a function $\llbracket _ \rrbracket: \Phi \rightarrow 2^X$, which maps a formula to the subset of states that satisfy it. On the other hand, we can define the semantics of the system whose states are represented by the set X by mapping a state to the set of formulas that hold in that state. This is represented by the function $th: X \rightarrow 2^\Phi$, called the theory map. This map associates a state with all its logical properties. As we can see, there is a close relationship between coalgebras and their logics and this relationship can be elegantly captured via the notion of dual adjunction [PMW06; Kli07; KP11; KKP04].

Intuitively, this notion borrowed from category theory embodies the idea that two different mathematical structures are the *best fit* for each other and that they share a symmetrical relationship between two opposite processes. In our case, on one side, we have coalgebras and on the other side, we have modal logics.

Chapter 2. Background

A dual adjunction can be sketched as per the following diagram.

$$\begin{array}{ccc}
 B \circlearrowleft C & \begin{array}{c} \xrightarrow{P} \\ \perp \\ \xleftarrow{Q} \end{array} & D^{op} \circlearrowright L
 \end{array}$$

The basic setting is given by two categories $\mathbf{C}$, $\mathbf{D}$ connected by two functors $P: \mathbf{C}^{op} \rightarrow \mathbf{D}$ and $Q: \mathbf{D}^{op} \rightarrow \mathbf{C}$ such that $P \dashv Q: \mathbf{C} \rightleftarrows \mathbf{D}$, i.e., P is the left adjoint functor, Q is the right adjoint functor and together with the two categories $\mathbf{C}$, $\mathbf{D}$, they form a dual adjunction. The adjunction is contravariant. The key property of dual adjunction is that there exists a natural bijection (a one-to-one correspondence) between the the two categories and their morphisms such that $\mathbf{C}(X, Q\Phi) = \mathbf{D}(\Phi, PX)$ for $X \in \mathbf{C}$, $\Phi \in \mathbf{D}$.

In our framework, the objects in $\mathbf{C}$ can be thought of as collections of states on which coalgebra structures are defined whereas $\mathbf{D}$ is the logical universe for propositional logic on which we define a functor for modelling modal operators. The functor P takes an object $X \in \mathbf{C}$ to the collection PX of predicates over X whereas the functor Q maps a formula into the set of theories.

We also define two endofunctors $B: \mathbf{C} \rightarrow \mathbf{C}$ and $L: \mathbf{D} \rightarrow \mathbf{D}$. The former describes the behaviour type of coalgebras, the latter is used to construct the syntax of the logic for those coalgebras. When L has an initial algebra $a: L\Phi \rightarrow \Phi$, Φ can be thought of as the collection of the formulas of the logic.

Remark 20 (Initial Algebra). In the opposite way as final coalgebra, an initial algebra for an endofunctor $L: \mathbf{D} \rightarrow \mathbf{D}$ is an object Φ in $\mathbf{D}$ equipped with a structure map $a: L\Phi \rightarrow \Phi$ such that for every object Y with a structure map $g: LY \rightarrow Y$, there exists a unique morphism $h: \Phi \rightarrow Y$ that makes the following diagram commute.

$$\begin{array}{ccc}
 L\Phi & \xrightarrow{Lh} & LY \\
 a \downarrow & & \downarrow g \\
 \Phi & \xrightarrow{h} & Y
 \end{array}$$

The connection between coalgebras and their logics and so, the relationship between the functors B and L , is established via a natural transformation $\alpha: LP \Rightarrow PB$. This transformation is sometimes referred to as the one-step semantics of the logic, as it

defines how the logical operators in L are interpreted over the single-step transitions of B . Given this natural transformation, the full semantics of the logic are induced naturally: for any B -coalgebra (X, γ) , the semantic map $\llbracket - \rrbracket : \Phi \rightarrow PX$ is the unique L -algebra morphism obtained via the universal property of the initial L -algebra (Φ, a) that makes the following diagram commute.

$$\begin{array}{ccccc} L\Phi & \xrightarrow{L\llbracket - \rrbracket} & LPX & \xrightarrow{\alpha_X} & PBX \\ a \downarrow & & & & \downarrow P_\gamma \\ \Phi & \xrightarrow{\exists! \llbracket - \rrbracket} & & & PX \end{array}$$

The bijective correspondence between P and Q allows us to define the dual counterpart of the one-step semantics α , the mate of α , denoted as $\alpha^\flat : BQ \Rightarrow QL$ [Kli07; KKP04]. Intuitively, while α describes how to interpret logical operators over system transitions, the mate $\alpha^\flat$ describes how transitions define the logical properties (or theories) of the states. Formally, $\alpha^\flat$ is obtained using the unit $\eta : \text{Id}_{\mathcal{C}} \Rightarrow QP$ and the counit $\epsilon : \text{Id}_{\mathcal{D}} \Rightarrow PQ$ of the dual adjunction as follows:

$$\alpha^\flat = QL\epsilon \circ Q\alpha Q \circ \eta BQ$$

Moreover, the counterpart of the semantic map $\llbracket - \rrbracket$ is the theory map $th : X \rightarrow Q\Phi$, which associates each state with its theories, i.e., the collection of all logical formulas it satisfies. Because $\llbracket - \rrbracket$ and th are related by the bijective correspondence $\mathcal{D}(\Phi, PX) \cong \mathcal{C}(X, Q\Phi)$, the uniqueness of the semantic map (established by the initiality of Φ as above) implies a corresponding uniqueness for the theory map. One can then show that th is the unique morphism making the following diagram commute [PMW06].

$$\begin{array}{ccccc} BX & \xrightarrow{Bth} & BQ\Phi & \xrightarrow{\alpha_\Phi^\flat} & QL\Phi \\ \gamma \uparrow & & & & \uparrow Q_a \\ X & \xrightarrow{\exists! th} & & & Q\Phi \end{array}$$

Example 21. Given a DFA as a coalgebra $(X, \gamma : X \rightarrow BX)$, the initial L -algebra is $\Phi = A^*$ whose a structure is defined as $a = [\epsilon, cons] : 1 + A \times A^* \rightarrow A^*$ where ϵ represents

Chapter 2. Background

the empty word and *cons* takes a pair $(u \in A, w \in A^*)$ into the concatenated word $uw \in A^*$. In this setting, if we want to identify states in a coalgebra, we need to check which of them make the same formulas true and a test is true if, starting from a given state, there exists a unique sequence of transitions that ends up in an accepting state of the coalgebra, after reading the test itself. That is, the truth of tests stands for the word acceptance.

Chapter 3

Learning via Logic

Our primary goal is to rephrase and generalise the well-known Angluin L^* learning algorithm introduced in the Chapter 2 using those notions from coalgebras and modal logics we presented there. In this chapter, we explore the connections between coalgebras and learning to understand in a more comprehensive way the process of making observations and of using them to learn how a system works.

The main contribution of this chapter is the definition of L^{co} , an algorithm that generalizes the L^* algorithm, to learn regular B -coalgebras over any finitary set functor B [Jac16]. This generalization allows us to learn all automata that can be expressed as B -coalgebras representing a class wider than just DFAs such as Mealy/Moore machines and bisimulation quotients of probabilistic transition systems. Specifically, we show that by using logical queries (the logical counterpart of the membership queries) and conjectures, we can learn regular pointed B -coalgebras.

The results presented in this chapter are manifold. The first one is defining an algorithm that in terms of coalgebras and modal logics is able to learn a language in the form of a minimal DFA. We use the same notation of the L^* algorithm employing an observation table and the two sets S and E containing states and observations (or tests) respectively. The interpretation of tests as modal formulas (as we will see) shows an expected connection with modal logic. On the other hand, a critical output is observing that the closed and consistent tables can be best captured by the modal logic concept of filtration. Moreover, this algorithm allows, in principle, the learning of

regular coalgebras for an arbitrary finitary set functor B , i.e., for those functor we are always able to find an expressive language [Sch05].

This chapter is mainly based on our work presented in the paper [BK18] and it is structured as follows: we provide a high-level overview of the L^{co} algorithm, highlighting both how it works and the fundamental differences from the classical Angluin L^* algorithm (cf. Section 3.1). Then, we introduce some assumptions and those preliminary notions that have not been covered during the previous chapter, but that are important to understand this specific algorithm (cf. Section 3.2). Finally, we present the complete L^{co} algorithm. Here, we introduce the formal definitions of filtration and logical tables, provide a detailed description of the algorithm, and offer a comprehensive proof of termination and correctness. We also include concrete examples to clarify the construction and the execution of the L^{co} algorithm (cf. Section 3.3).

3.1 The L^{co} algorithm at a glance

The classical L^* algorithm aims to learn a minimal DFA representing an initially unknown regular language L where $L \subseteq A^*$. In our coalgebraic setting, we think of L as being represented by an infinite pointed coalgebra (A^*, γ, λ) where $\gamma: A^* \rightarrow 2 \times (A^*)^A$ is given by $\gamma(u) = \langle o(u), t(u)(a) \rangle$ for any $u \in A^*$ with $o(u) = 1$ iff $u \in L$ and $t(u)(a) = u \cdot a$ for all $u \in A^*$ and $a \in A$ and λ is the empty word. The automaton represented as (A^*, γ, λ) is, by definition, always infinite with each state corresponding to a single word and in this setting, the assumption that L is regular is equivalent to the statement that the pointed coalgebra (A^*, γ, x) is behavioural equivalent to some finite, minimal, well-pointed B -coalgebra (Y, δ, y) where $BX = 2 \times X^A$ [Adá+13] and this last one is what we are trying to learn. Moreover, the L^{co} algorithm aims to generalise this goal. It learns a finite, well-pointed B -coalgebra (Y, δ, y) that is behaviourally equivalent to a possible infinite pointed B -coalgebra (X, γ, x) that is regular where B is an arbitrary finitary set functor and (X, γ, x) is our target system whose behaviour we want to learn. Therefore, we won't only learn DFAs, but also those systems that can be represented by these types of coalgebras such as Mealy Machine represented by $B = (- \times O)^I$ [HR10] where I and O are finite sets that constitute the input and

output alphabet of the Mealy machine, respectively and discrete Markov chain represented as $P\mathbf{At} \times \mathcal{D}_\omega$ -coalgebras where $\mathbf{At}$ is a set of propositional variables and $\mathcal{D}_\omega$ is the probability distribution functor [Sok11].

As in L^* , also in our setting, we assume an oracle called a *Teacher* and this *Teacher* can answer different types of queries:

- *Logical queries*: given a formula $\varphi \in \mathcal{F}(\Lambda)$ of the logic defined for the specific B -coalgebra, the *Teacher* returns *yes*, if that formula is true at the distinguished state $x \in X$ (these are the counterpart of the membership queries in L^*).
- *Conjecture*: given a well-pointed B -coalgebra, that is behavioural equivalent to the original one, the *Teacher* returns *yes*, if the conjecture is correct. Otherwise, they return a counterexample in the form of a formula $\varphi \in \mathcal{F}(\Lambda)$ that distinguishes the point of the conjectured coalgebra from the point x of the coalgebra we are trying to learn (same as in L^*).
- *Successor queries*: these queries are used to determine which are the successors of a specific state after performing one step in the transition structure. We require them to obtain the successors of a specific state and then to decide whether the Observation Table satisfies the closed condition.

3.2 Preliminaries

Even though Chapter 2 introduced general coalgebraic and modal logical notions that will be used throughout the thesis, the L^{co} algorithm requires specific assumptions on the functor and additional concepts related to the transition structure. In what follows, we introduce these necessary ideas.

Remark 22. 1. Throughout this chapter, we work with an arbitrary finitary set functor $B: \mathbf{Set} \rightarrow \mathbf{Set}$ and we will assume that B preserves finite intersections. Moreover, for any sets X, Y with $X \subseteq Y$ the inclusion map $\iota_{X,Y}: X \rightarrow Y$ is mapped to the inclusion $\iota_{BX, BY}: BX \rightarrow BY$ and in particular, we have $BX \subseteq BY$. Under this assumption, the finitariness of the functor B means that for all sets X and

all elements $b \in BX$, there is a finite subset $X' \subseteq X$ such that $b \in BX'$ [AT90; Adá+13]. Also, we write $X' \subseteq_\omega X$ if X' is a finite subset of X .

2. Another functor that will play an important role in this chapter is the contravariant power set functor (cf. page 29).
3. For a given set X , we denote by $\#X$ the cardinality of X .

3.2.1 Notion of successor state in coalgebras

Coalgebras do not come naturally equipped with a notion of *successor state*, i.e., a state that can be reached after performing one step in the transition structure. This notion is important when we deal with concrete constructions of transition systems. It is possible to formalise it via the notion of base, though.

Definition 23 (Base). *Given a finitary set functor $B: \mathbf{Set} \rightarrow \mathbf{Set}$ and an element $b \in BX$, we define*

$$\text{Base}_X^B(b) := \bigcap \{Y \subseteq_\omega X \mid b \in BY\}.$$

Whenever B and X are clear from the context we will drop the superscript and subscript, respectively, and simply write $\text{Base}(b)$.

The base identifies the smallest possible set of successor states of a specific state x . The following fact states that the base is the minimal set containing the successors and if we apply the functor B to it, the minimality is preserved.

Fact 24. *Let B be a finitary set functor, let X be a set and let $b \in BX$. Then $b \in B\text{Base}_X^B(b)$ and for any proper subset $X' \subset \text{Base}_X^B(b)$, we have $b \notin BX'$.*

The notion of base can be used to associate a graph with any B -coalgebra. This graph can be used to give a concrete representation of reachability [Adá+13]. In the following definition, we use the finite powerset functor $P_\omega: \mathbf{Set} \rightarrow \mathbf{Set}$ that maps any set X to the collection of its finite subsets, i.e., $P_\omega(X) = \{S \subseteq X \mid S \text{ is finite}\}$ and for any function $f: X \rightarrow Y$, the functor takes the direct image, mapping each finite subset $S \in P_\omega(X)$ to its forward image under f , i.e., $P_\omega(f)(S) = \{f(s) \mid s \in S\}$.

Definition 25 (Canonical graph). *Let (X, γ) be a B -coalgebra. The canonical graph $(X, \text{can}_\gamma : X \rightarrow P_\omega X)$ is defined by putting $\text{can}_\gamma(x) := \text{Base}_X^B(\gamma(x))$ 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 26. 1. $B = A \times \text{Id}$ where A is a set and Id is the identity functor. In this case $\text{Base}_X^B((a, x)) = \{x\}$ for $(a, x) \in A \times X$.

2. $B = 2 \times \text{Id}^A$ where A is a finite set (the *alphabet*). Then $\text{Base}_X^B((i, f)) = \{fa \mid a \in A\}$ for $(i, f) \in BX$.

3. Let $B = \mathcal{D}_\omega$ where $\mathcal{D}_\omega X$ denotes the collection of discrete probability distributions over X with finite support, i.e. , $\mathcal{D}_\omega X = \{\mu : X \rightarrow [0, 1] \mid \sum_{x \in X} \mu(x) = 1, \text{supp}(\mu) \text{ finite}\}$ and for a function $f : X \rightarrow Y$ and $\mu : X \rightarrow [0, 1]$ we have $(\mathcal{D}_\omega f)(\mu)(y) = \sum_{f(x)=y} \mu(x)$, where $\text{supp}(\mu) = \{x \in X \mid \mu(x) \neq 0\}$ denotes the support of μ . In this case $\text{Base}_X^{\mathcal{D}_\omega}(\mu) = \text{supp}(\mu)$. 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 $\mu : X \rightarrow (0, 1]$ with $\text{supp}(\mu)$ 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.

3.2.2 Minimal representation

As said, the L^{co} algorithm learns a minimal finite representation of the behaviour of a given pointed coalgebra. In this setting, minimality means that the learned pointed coalgebra should not have any proper subcoalgebras or proper quotients. These definitions are as follows:

Definition 27. *Let (X, γ, x) be a pointed B -coalgebra. We say (Y, δ, y) is a quotient of (X, γ, x) if there is a surjective morphism $q : (X, \gamma, x) \rightarrow (Y, \delta, y)$. We say (Y, δ, y) is a subobject of (X, γ, x) if there is an injection $m : (Y, \delta, y) \rightarrow (X, \gamma, x)$. 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, γ, x) .*

The following notion from [Adá+13] captures the minimality of a pointed coalgebra.

Definition 28. *A pointed B -coalgebra (X, γ, x) is called well-pointed if it is simple and reachable, i.e., if it does not have proper quotients and proper subcoalgebras.*

Example 29. 1. For $B = 2 \times (-)^A$, the finite well-pointed B -coalgebras are DFAs where each state is reachable from the initial state and where no two states represent the same (regular) language.

2. For $B = P_\omega$ (where $P_\omega X = \{X' \mid X' \subseteq_\omega X\}$), the well-pointed B -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.

3.2.3 Logical quotient

We introduce the logical quotient because it allows to identify states in the initial infinite coalgebra that share the same observable behaviour. Since the behaviour of each state is characterized by logical formulas, the quotient precisely collapses states that satisfy the exact same set of formulas. When this logical quotient is maximal, it yields the desired minimal and finite well-pointed B -coalgebra. This state identification procedure is the coalgebraic and logical counterpart to the mechanism of the Angluin L^* algorithm, where maintaining a closed and consistent table introduces the minimal number of states required to define the DFA representing the language we want to learn.

More formally, given an expressive modal language $\mathcal{F}(\Lambda)$ for B (cf. Subsection 2.3.1), we can compute the maximal quotient of a coalgebra by identifying states that satisfy the same modal formulas in $\mathcal{F}(\Lambda)$. Recalling Definition 17 in Subsection 2.3.1, we use the notation $|x|_\Sigma$ to denote the equivalence class of $x \in X$ together with all those other states that satisfy the same logical formulas in $\Sigma \subseteq \mathcal{F}(\Lambda)$. The set of all equivalent classes forms the state space of the quotient and it is denoted as $|X|_\Sigma$, when Σ is clear from the context, we omit it. Also, we view the operation of identifying equivalent states as a function $|-|_\Sigma : X \rightarrow |X|_\Sigma$. We can apply the functor B to this

function as usual. The following fact confirms that this construction is a valid and maximal coalgebraic quotient.

Fact 30. *Let (X, γ) be a B -coalgebra and let $\mathcal{F}(\Lambda)$ be an expressive language, let $|X| = \{|x| \mid x \in X\}$ and define the logical quotient $\gamma_{\mathcal{F}(\Lambda)}: |X| \rightarrow B|X|$ by putting $\gamma_{\mathcal{F}(\Lambda)}(|x|) := (B|-|)(\gamma(x))$. Then the logical quotient $\gamma_{\mathcal{F}(\Lambda)}$ is well-defined and $|-|: X \rightarrow |X|$ given by $x \mapsto |x|$ is a B -coalgebra map that computes the maximal quotient of (X, γ) .*

The proof of well-definedness can be found in e.g., [KL09]. The maximality of the quotient 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 B -coalgebra.

Lemma 31. *Let (X, γ, x) be a pointed B -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 B preserves finite intersections [Rut00, Proposition 6.5] and [Adá+13, Remark 2.7]. 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 [Adá+13, Lemma 3.16] reachability of its canonical graph implies reachability of a pointed B -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 28). $\square$

3.3 The L^{co} algorithm

In what follows, we first describe the configurations of the algorithm. Then, we provide the algorithm parametric in a finitary set functor B together with a couple of concrete examples and lastly, we prove termination and correctness for it.

3.3.1 Filtrations and logical tables

Following Angluin’s original terminology, we will refer to configurations of our algorithm as logical tables or simply as tables. Before giving an intuition of how our idea of table is related to the one of Angluin and why this is achieved via the notion of filtration, we need to introduce some terminology concerning sets of formulas from $\mathcal{F}(\Lambda)$.

Definition 32 (Subformula-closedness). *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 $\heartsuit(\varphi_1, \dots, \varphi_n) \in \Sigma$ implies $\varphi_i \in \Sigma$ for all $i \in \{1, \dots, n\}$.

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, where filtration is a method that produces finite models by taking a large, possibly infinite model and identifying as many states as possible and the truth lemma ensures that the truth value of any formula $\varphi \in \Sigma$ is preserved when moving from the original structure to the filtered structure [BRV01, Theorem 2.39]. 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 require a subformula-closed set of formulas because this condition intuitively replicates the role of suffix-closedness in the classical L^* algorithm. In L^* , the set of tests E is suffix-closed (i.e., if $uv \in E$, then $v \in E$), which ensures that the immediate precursor of any observable test is also an observable test. Analogously, in our setting, the formulas used as tests must be structured such that if a formula $\langle a \rangle\varphi$ is in our observation set, the relevant one-step predecessor formulas (such as φ) are also included, which is essential for ensuring the table can achieve the closed condition.

We now give a coalgebraic account of the filtrations that play a role in our algorithm.

Definition 33 (Filtration). *Let (X, γ) be a B -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*

Chapter 3. Learning via Logic

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 $x' \in \text{Base}(\gamma(x))$ 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 following map:

$$\gamma_{S, \Sigma}(|x|_\Sigma) := (B|_{|\Sigma})(\gamma(x)).$$

Please note that usually we require Σ to be finite but that is not necessary for defining a filtration.

Lemma 34. *Under the conditions on (S, Σ) from the previous definition, the (S, Σ) -filtration of a B -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 $(B|_{|\Sigma})(\gamma(x)) \in B|S|_\Sigma$ as S satisfies condition 2. $\square$

Definition 33 is reminiscent of the definition of a logical quotient in Fact 30. 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 $|_{|\Sigma}$ preserves truth of formulas in Σ .

Lemma 35. *Let (X, γ) be a B -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*

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

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 $\varphi = \neg\psi$ follow easily from the induction hypothesis since we are working with subformula-closed set of formulas. Suppose now that $\varphi = \heartsuit(\psi_1, \dots, \psi_n)$.

Chapter 3. Learning via Logic

We have $x \models \heartsuit(\psi_1, \dots, \psi_n)$ iff $\gamma(x) \in \llbracket \heartsuit \rrbracket_X(\llbracket \psi_1 \rrbracket_{(X, \gamma)}, \dots, \llbracket \psi_n \rrbracket_{(X, \gamma)})$ iff $x \in P\gamma(\llbracket \heartsuit \rrbracket_X(\llbracket \psi_1 \rrbracket_{(X, \gamma)}, \dots, \llbracket \psi_n \rrbracket_{(X, \gamma)}))$. The last statement is by I.H. on the ψ_i 's equivalent to

$$\sigma(x) \in P\gamma\left(\llbracket \heartsuit \rrbracket_X(P|_{-\Sigma}(\llbracket \psi_1 \rrbracket_{(|S|_{\Sigma}, \gamma_{S, \Sigma})}), \dots, P|_{-\Sigma}(\llbracket \psi_n \rrbracket_{(|S|_{\Sigma}, \gamma_{S, \Sigma})}))\right)$$

which is by naturality of $\llbracket \heartsuit \rrbracket$ equivalent to

$$x \in (P\gamma \circ PB|_{-\Sigma})\left(\llbracket \heartsuit \rrbracket_{|S|_{\Sigma}}(\llbracket \psi_1 \rrbracket_{(|S|_{\Sigma}, \gamma_{S, \Sigma})}, \dots, \llbracket \psi_n \rrbracket_{(|S|_{\Sigma}, \gamma_{S, \Sigma})})\right)$$

which is equivalent to $B|_{-\Sigma}(\gamma(x)) \in \llbracket \heartsuit \rrbracket_{|S|_{\Sigma}}(\llbracket \psi_1 \rrbracket_{(|S|_{\Sigma}, \gamma_{S, \Sigma})}, \dots, \llbracket \psi_n \rrbracket_{(|S|_{\Sigma}, \gamma_{S, \Sigma})})$ which finally is equivalent to $|x|_{\Sigma} \models \heartsuit(\psi_1, \dots, \psi_n)$ as required. $\square$

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

Definition 36 (Logical table). *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 $| \subseteq |_{\text{Base}(\gamma(x))}|S|_{\Sigma}$. 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 37. Readers familiar with the L^* algorithm will recognise the closedness condition. In fact, the closedness condition in L^* ensures that the current set of states S is sufficient to explain all one-step transitions of the DFA (the table is closed if every possible successor row, generated by taking a one-step transition from a representative state in S , matches a row already present in S). Analogously, in the coalgebraic setting, the condition states that the set of immediate successor states of any $x \in S$, which is derived by the computation of the $\text{Base}(\gamma(x))$, must belong to the equivalence classes already defined by S based on the set of formula Σ . If this inclusion holds, it means that the behaviour of every potential successor is already captured by one of the states in S , guaranteeing that the current conjecture contains the necessary states to model the system transitions. 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 using the same approach detailed at page 18 of Chapter 2 [MP95].

3.3.2 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 now describe the 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) such that $x \in S$ and such that the pointed B -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 Algorithm 2, makes the following steps:

1. The start table is $(\{x\}, \{\top\})$ where the first component ensures that the distinguished point of (X, γ, x) is represented and 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 so, it yields the conjecture as described in item 4. If not, it proceeds as described in item 3.
3. Given a configuration (S, Σ) that is not closed, pick an element $x' \in S$ such that $\text{Base}(\gamma(x')) \not\subseteq |S|_\Sigma$. For all $x'' \in \text{Base}(\gamma(x'))$ 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 $\text{Base}(\gamma(x'))$. Note that we are adding elements of $\text{Base}(\gamma(x'))$ one-by-one to maintain sharpness of the table and that to keep track of the successor states and exclude those already checked, we introduce in the concrete algorithm a working set $U = \text{Base}(\gamma(x'))$.
4. Given the closed configuration (S, Σ) , present the (S, Σ) -filtration $\gamma_{S, \Sigma}$ to the Teacher. If the Teacher accepts, the algorithm terminates. If the Teacher rejects

it, they have to provide a counterexample, i.e., a formula $\varphi \in \mathcal{F}(\Lambda)$ such that $x \models \varphi$ and $|x|_\Sigma \not\models \varphi$ or vice versa. In this case we update Σ with $\Sigma \cup \text{Sub}(\varphi)$ and the algorithm continues as described in the item 2.

Algorithm 2 The L^{co} algorithm

Input: Teacher (X, γ, x) , expressive language $\mathcal{F}(\Lambda)$

Output: Pointed B -coalgebra $(|S|_\Sigma, \gamma_{S,\Sigma}, |x|_\Sigma)$

```

1:                                     ▷ Initialization
2: Initialize table  $(S, \Sigma)$  with  $S = \{x\}$  and  $\Sigma = \{\top\}$ 
3:                                     ▷ Main
4: while true do
5:   if  $(S, \Sigma)$  is closed then
6:     Conjecture:  $\gamma_{S,\Sigma}: |S|_\Sigma \rightarrow B|S|_\Sigma$ 
7:      $|x|_\Sigma \mapsto (B|-|_\Sigma)(\gamma(x))$ 
8:     if Conjecture is false then
9:       Provide  $\varphi \in \mathcal{F}(\Lambda)$  s.t.  $x \models \varphi$  and  $|x| \not\models \varphi$ 
10:       $\Sigma \leftarrow \Sigma \cup \text{Sub}(\varphi)$ 
11:     else
12:       return  $\gamma_{S,\Sigma}: |S|_\Sigma \rightarrow B|S|_\Sigma$ 
13:     end if
14:   else
15:     Pick  $x' \in S$  s.t.  $\text{Base}(\gamma(x')) \not\subseteq |S|_\Sigma$ 
16:      $U \leftarrow \text{Base}(\gamma(x'))$ 
17:     while  $U \neq \emptyset$  do
18:       Take  $x'' \in U$ 
19:       if  $|x''| \in |S|_\Sigma$  then
20:          $U \leftarrow U \setminus \{x''\}$ 
21:       else
22:          $S \leftarrow S \cup \{x''\}$ 
23:          $U \leftarrow U \setminus \{x''\}$ 
24:       end if
25:     end while
26:   end if
27: end while

```

3.3.3 The L^{co} algorithm with two examples

We illustrate the L^{co} algorithm via two examples: one example is on learning Mealy machines and here, there is a difference with the standard Angluin algorithm adapted for Mealy machines [Aar+14; SG09] where the entries of the table are the output values

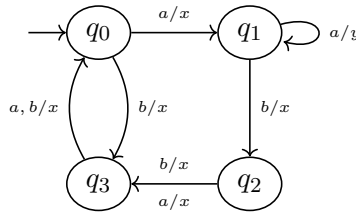
that the Mealy automata produce. In our setting, the entries are always the values 0 or 1 according to the truth values of the formulas. The output values are given by the coalgebraic map γ . The other example is to the best of our knowledge new, an Angluin learning algorithm for discrete Markov chains. While learning probabilistic automata is an active area [BCG13; Mao+11; Tze92], existing work on the matter focuses on constructing minimal automata for a probabilistic language rather than, as our algorithm does, on *bisimulation* quotients of transition systems.

Example 38 (Mealy Machines). Here, we show how the L^{co} algorithm learns functions $L: I^+ \rightarrow O$, 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 $B = (- \times O)^I$ [HR10]. 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 $\gamma(w)(a) = \langle wa, L(wa) \rangle$. The expressive language for the Mealy functor $B = (- \times O)^I$ is defined as:

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

Given a B -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 $\pi_1(\gamma(x)(i)) \models \varphi$.

As a concrete example, we want to learn 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 2, the first thing to do is checking if the starting table

	$\top$	$p_{a/y}$	$[a]p_{a/y}$
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
$\gamma(a)(a)$	1	1	1
$\gamma(a)(b)$	1	0	0
$\gamma(b)(a)$	1	0	1
$\gamma(b)(b)$	1	0	1

Figure 3.1: From a not closed table to a closed one.

$\mathcal{T} = (\{\lambda\}, \{\top\})$ is closed. The table is trivially closed, as $\top$ is always true and the first conjecture of the algorithm is as follows:

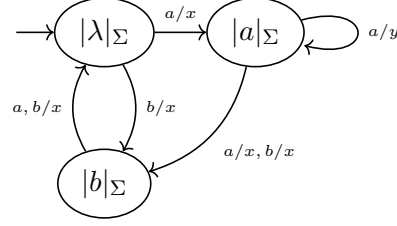


Since the conjecture is incorrect, the teacher returns a counterexample, e.g., the formula $[a]p_{a/y}$. We update the set Σ of formulas with it together with all its subformulas.

The table is now $\mathcal{T} = (\{\lambda\}, \{\top, p_{a/y}, [a]p_{a/y}\})$ and it is not closed because if we pick up the only element we have in S , its base is not included in the set of equivalent classes generated by the elements in S . Therefore, we add another state to the set of states according to the **else** part of the **if**-statement 5. 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 Figure 3.1.

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:



The conjecture is incorrect and we 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** 8 and we compute our conjecture. Figure 3.2 represents this step.

	$\top$	$p_{a/y}$	$[a]p_{a/y}$	$p_{a/x}$	$[a]p_{a/x}$	$[b][a]p_{a/x}$	$[b][b][a]p_{a/x}$	$[a][b][b][a]p_{a/x}$
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
$\gamma(a)(a)$	1	1	1	0	0	1	1	1
$\gamma(a)(b)$	1	0	0	1	1	1	0	1
$\gamma(b)(a)$	1	0	1	1	0	1	0	1
$\gamma(b)(b)$	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
$\gamma(a)(b)$	1	0	0	1	1	1	0	1
$\gamma(b)(a)$	1	0	1	1	0	1	0	1
$\gamma(b)(b)$	1	0	1	1	0	1	0	1
$\gamma(aa)(a)$	1	1	1	0	0	1	1	1
$\gamma(aa)(b)$	1	1	1	0	0	1	1	1

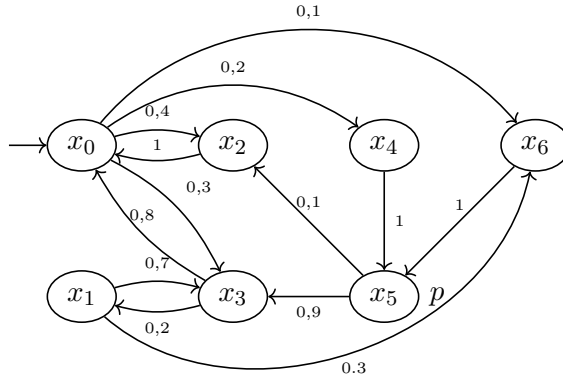
Figure 3.2: Changing of the table according to the Base-step of the algorithm.

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.

Example 39 (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 $P\mathbf{At} \times \mathcal{D}_{\omega-}$ -coalgebras where $\mathbf{At}$ is a set of propositional variables. In our example we will assume $\mathbf{At} = \{p\}$ 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 [Sok11]. An expressive language for $B = P\mathbf{At} \times \mathcal{D}_{\omega-}$ is given by

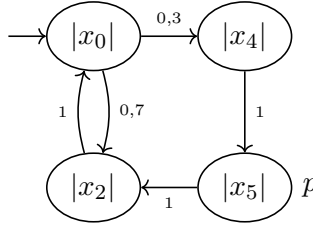
$$\mathcal{F}(\Lambda) \ni \varphi ::= \top \mid p, p \in \mathbf{At} \mid \varphi_1 \wedge \varphi_2 \mid L_q \varphi, q \in [0, 1],$$

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 . Consider now the following pointed $P\mathbf{At} \times \mathcal{D}_{\omega-}$ -coalgebra based on $X = \{x_0, x_1, \dots, x_6\}$ 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 $\varphi = L_{0.2}L_1p$ as $x_0 \models \varphi$ and $|x_0| \not\models \varphi$. This leads to the new table $(\{x_0\}, \Sigma_1)$ with $\Sigma_1 = \{L_{0.2}L_1p, L_1p, p, \top\}$. 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 $\text{Base}(\{\gamma(x_2)\}) = \{x_0, x_5\}$ and obviously $|x_5|_{\Sigma_1} \not\subseteq |\{x_0, x_2, x_4\}|_{\Sigma_1}$ 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:



3.3.4 Termination and 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 40. *Let (X, γ, x) be a pointed B -coalgebra and suppose it is behaviourally equivalent to a finite well-pointed B -coalgebra (Y, δ, y) . Let $\mathcal{F}(\Lambda)$ be an expressive language for B -coalgebras. The L^{co} algorithm with teacher (X, γ, x) and test language $\mathcal{F}(\Lambda)$ terminates and correctly returns the finite, well-pointed B -coalgebra (Y, δ, y) that is behaviourally equivalent to (X, γ, x) .*

Proof. The proof of this theorem can be split into the three step below. We provide for each of them specific Lemma that all together establish the final proof.

- The algorithm builds tables (S, Σ) where S is a subset of the carrier of the smallest pointed subcoalgebra (X', γ', x) of (X, γ, x) that contains x . The existence of the finite well-pointed coalgebra (Y, δ, y) implies that Y is the maximal quotient of

X' . Therefore the size of S is bound by the number of elements of Y (Lemma 41 and Lemma 42).

- 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 43).
- Whenever the teacher provides a counterexample the resulting table will not be closed (Lemma 44).

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 35, 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

$$\begin{array}{ccc} BS & \xrightarrow{\text{Base}_S^B} & PS \\ B|-|_\Sigma \downarrow & & \downarrow P|-|_\Sigma \\ B|S|_\Sigma & \xrightarrow{\text{Base}_{|S|_\Sigma}^B} & P|S|_\Sigma \end{array}$$

where $|-|_\Sigma$ is the mapping to equivalence classes restricted to $S \subseteq X$. Clearly $|-|_\Sigma$ is mono (due to sharpness of (S, Σ)) and thus we know from [Adá+13, Rem. 2.49] that the square commutes (and even forms a pullback). Whenever the algorithm adds a state x' to S we know that $x' \in \text{Base}(\gamma(x''))$ for some $x'' \in S$ that has been already added at an earlier stage. Commutation of the diagram implies that $|x'|_\Sigma \in \text{Base}(B|-|_\Sigma(\gamma(x''))) = \text{Base}(\gamma_{S, \Sigma}(|x''|_\Sigma))$ where the equality is a consequence of the definition of the filtration. A routine induction argument together with [Adá+13, 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. $\square$

Now, we prove the claims from the proof of Theorem 40. In the following let (X, γ, x) be a pointed B -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 41. *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 $(X', \gamma|_{X'}, x)$ be the smallest pointed subcoalgebra of (X, γ, x) . Clearly we have $\text{Base}(\gamma(x')) \subseteq X'$ for all $x' \in X$ as the coalgebra map γ restricts to a map $X' \rightarrow BX'$. 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 $S \subseteq S' \cup \bigcup_{x' \in S'} \text{Base}(\gamma(x'))$. 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. $\square$

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

Lemma 42. *Let (S, Σ) be a table computed by the L^{co} algorithm with teacher (X, γ, x) . Then $\#S \leq \#Y$.*

Proof. By Lemma 31, 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 $f: S \rightarrow Y$ 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 $f: S \rightarrow Y$ has to be injective which shows that $\#S \leq \#Y$ as required. $\square$

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 43. *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 42 this can only happen finitely often which implies that after finitely many additions to S the table will be closed as required. $\square$

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

Lemma 44. *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 that (S, Σ') is closed (and hence, according to our setting, the table will obviously be sharp because this condition is automatically preserved). For every set $Z \subseteq X$ let $f_Z: |Z|_{\Sigma'} \rightarrow |Z|_{\Sigma}$ 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 B -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:

$$\begin{aligned} \gamma_{S, \Sigma}(f_S(|x'|_{\Sigma'})) &= \gamma_{S, \Sigma}(|x'|_{\Sigma}) = (B|-|_{\Sigma})(\gamma(x')) = B(f_X \circ |-|_{\Sigma'})(\gamma(x')) \\ &= Bf_X(B|-|_{\Sigma'}(\gamma(x'))) \\ &\stackrel{B|-|_{\Sigma'}(\gamma(x')) \in B|S|_{\Sigma'}}{=} Bf_S(B|-|_{\Sigma'}(\gamma(x'))) \end{aligned}$$

which shows that f_S is indeed a pointed B -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 without loss of generality $|x|_{\Sigma'} \models \varphi$ and $|x|_{\Sigma} \not\models \varphi$. $\square$

This finishes the proofs of the claims that are necessary to prove Theorem 40. 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 B -coalgebra (X, γ) is *k-branching*, if there exists some $k \in \omega$ such that for all $x \in X$ we have $\# \text{Base}(\gamma(x)) \leq k$.

Proposition 45. *Let (X, γ, x) be a k -branching pointed B -coalgebra, let n be the size of the behavioural equivalent well-pointed B -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 $\mathcal{O}(k \cdot m \cdot n^3)$ logical queries and at most n equivalence queries.*

Proof. Let (S, Σ) be a table occurring during the run of the algorithm. By Lemma 42 $\#S \leq n$. 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 44 and 43. 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 and we need to perform these queries at most $(n - 1)$ times. This shows that we need $\mathcal{O}(k \cdot m \cdot n^3)$ logical queries. The upper bound on equivalence queries is obvious. $\square$

Our complexity discussion demonstrates that the move from deterministic finite automata to B -coalgebras does not essentially alter the number of queries.

3.4 Conclusion

In this chapter, we introduced L^{co} , a generalisation of the Angluin L^* algorithm that allows us to learn regular B -coalgebras for an arbitrary finitary set functor B . A key feature of this work was the shift from using words as tests to using modal formulas, providing a logical interpretation for the process of state identification.

However, this framework remains confined to those systems whose observable behaviour is modelled using Boolean-valued modal logics. Also, it does not explicitly use duality. This is one of the main motivations for what we will present in the next chapter: the move to a fully generalized framework based on dual adjunctions. This represents a significant improvement as it enables us to state the learning algorithm, including its proofs of correctness and termination, in purely categorical terms.

Beyond the categorical framework, a future extension of the L^{co} algorithm should approach the challenge of approximate learning and the consideration of many-valued logics. These non-Boolean logics are essential for learning an additional variety of systems, such as weighted automata or fuzzy transition systems, where observations are real numbers or transitions are partially true or partially applicable rather than simple truth values.

The concrete implementation of the L^* algorithm has been presented in several papers such as [Ber+05; RSB05] and additional future work related to the content of this chapter would be to implement the L^{co} algorithm. We anticipate two interesting challenges. On the one hand, to implement a teacher that provides *good* counterexamples and on the other hand, to move from words checking to formula checking, that is, implementing the logical generalisation we provided in this chapter.

Chapter 4

Learning via Duality

In the previous chapter, we introduced the idea that the observation component of the Angluin framework, i.e., the tests, could be characterised by modal formulas of a specific logical language. This allowed us to obtain a generalisation of Angluin’s original algorithm from DFAs to coalgebras for an arbitrary finitary set functor B in the sense that given a (possibly infinite) pointed B -coalgebra that we assume to be regular (i.e., having an equivalent finite representation), we can learn its finite representation through *logical queries* and making conjectures. That approach is confined to Boolean-valued modal logics and does not explicitly use duality. Also, the definition of base is not given in purely categorical terms.

This chapter introduces a significant generalization and improvement of the coalgebraic learning framework mainly using the key insight that dual adjunctions connect coalgebras and tailor-made logical languages [KP11; Bor94; KR16]. This structural connection allows us to move beyond the category of sets to a more general class of categories and to redefine some core concepts such as the notion of base and subformula-closedness, presenting them in purely categorical terms. Also, the result of this chapter is still an algorithm parametric in the type of system B that can be applied to a broader spectrum of state-based systems and together with the algorithm, we also provide a proof of correctness and termination of it.

Within this new categorical framework, standard variants of Angluin’s algorithm for learning Mealy machines [Aar+14; SG09] can fit more naturally than in what we

presented in the previous chapter (cf. Subsection 3.3.3). There, formulas had to be Boolean-valued, whereas, as we will see, this framework allows to use an intuitive language where a formula is simply a input word w whose truth value at a state x is the observed output after entering w at x . Later on, in the Example 46, we will provide a concrete intuition of it and the technical details.

The core contribution of this chapter is twofold. On the one hand, we present the results that have been published in our second paper [BKR19] which establishes, as said, a generic duality-based learning algorithm parametric to the type of system under consideration where there is a fundamental distinction between states of the learned system and tests, modelled as logical formulas. In this context, we are able to learn not only DFAs and Mealy Machines but also bisimilarity quotients of labelled transition systems. On the other hand, we include additional content that extends this learning framework. The underlying idea is the use of a composite functor BT to learn a representation of B -coalgebras in the form of equivalent BT -coalgebras, where B represents the transitions of the system and T represents the branching behavioural part. Within this framework, we use forgetful logics, which are constructed by combining an expressive modal logic for B with special logics for T whose purpose is to resolve branching [KR16]. Intuitively, these logics allow us to maintain a consistent syntax across different system types; the *forgetful* aspect resides in the interpretation, where the branching expressed by T is collapsed into observable values. The effect is that we can use the same set of test formulas whether we are modelling B -coalgebras or their branching BT counterparts. This shift expands the scope of our learning algorithm, allowing it to handle more complex state-based systems such as non-deterministic finite automata (NFAs) and alternating automata by treating branching as an internal behaviour to be resolved during observation.

The structure of this chapter is as follows: we introduce all those preliminary notions that underpin the new learning algorithm (cf. Section 4.1). These concepts are in common to both the duality paper [BKR19] and the BT enhanced version. Also, we rephrase some ideas already presented in Chapter 3 but in categorical terms. Then, we present the algorithm for learning via duality with its proof of termination (cf.

Section 4.2). Then, we present the *BT* learning algorithm together with the idea of forgetful logics highlighting the modifications to the proofs and definitions required to handle this enhanced learning framework (cf. Section 4.3). Lastly, we present a couple of practical examples to give the intuition of how both algorithms work and to show how it is possible to learn specific structures (cf. Section 4.4).

4.1 Preliminaries

This section introduces the necessary formal notions that allow us to define a general learning algorithm for *B* and *BT*-coalgebras. Here, we introduce some categorical notions (such as well-powered and complete categories, subobjects and intersection preserving functors) and restate some learning concepts presented in the previous chapter (such as subformula-closedness and the base) using a more specific categorical terminology. Before these formal definitions, we first recall the idea of dual adjunctions: a categorical framework (cf. Chapter 2, Subsection 2.3.2) which links the category of systems to their associated logical structures:

$$\begin{array}{c}
 B \circlearrowleft C \begin{array}{c} \xrightarrow{P} \\ \perp \\ \xleftarrow{Q} \end{array} D^{op} \circlearrowright L
 \end{array}$$

The basic setting is given by two categories $\mathbf{C}$, $\mathbf{D}$ connected by two functors $P: \mathbf{C}^{op} \rightarrow \mathbf{D}$ and $Q: \mathbf{D}^{op} \rightarrow \mathbf{C}$ such that $P \dashv Q: \mathbf{C} \rightleftarrows \mathbf{D}$. Here, the objects in $\mathbf{C}$ can be thought of as collections of states on which coalgebra structures are defined whereas $\mathbf{D}$ is the logical universe for propositional logic on which we define a functor for modelling modal operators. The functor P takes an object $X \in \mathbf{C}$ to the collection PX of predicates over X whereas the functor Q maps a formula into the set of theories. The two endofunctors $B: \mathbf{C} \rightarrow \mathbf{C}$ and $L: \mathbf{D} \rightarrow \mathbf{D}$ describe the behaviour type of coalgebras and constructs the syntax of the logic for those coalgebras, respectively. We use this framework as foundation of our central idea: dual adjunction connects coalgebras and tailor-made logical languages allowing us to determine a generic learning algorithm for coalgebras that is parametric in the type of system under consideration. In this setting, we can

distinguish between states of the learned system and tests, modelled as logical formulas. Below, we provide two examples of this duality to concretely refresh what we presented in Chapter 2.

Example 46. Let $\mathbf{C} = \mathbf{D} = \mathbf{Set}$ and consider the functors $P = Q = O^-$, $B = (O \times -)^A$ and $L = A \times (1 + -)$, where A and O are fixed sets, thought of as input and output alphabet, respectively. Then B -coalgebras correspond to Mealy machines and the initial L -algebra is given by the set A^+ of finite non-empty words over A . For $X \in \mathbf{C}$, the one-step semantics $\delta_X: A \times (1 + O^X) \rightarrow O^{BX}$ is defined by $\delta_X(a, \text{inl}(*)) = \lambda f. \pi_1(f(a))$ and $\delta_X(a, \text{inr}(g)) = \lambda f. g(\pi_2(f(a)))$. Concretely, formulas are words in A^+ , the (O -valued) semantics of $w \in A^+$ at state x is the output $o \in O$ that is produced after processing the input w from state x . Please note that in the previous chapter in the example 38, the formulas describing Mealy machines are of the form of $p_{i/o}$ and such formula is true at a state x whether the output at x after an input i is o . With O -valued formulas, we can test the behaviour of a Mealy machine more directly by testing what a modal formula that corresponds to an input i evaluates to. In the boolean case, we have several formulas that represent the concept that on a specific input i the system returns o , then we can check whether the statement is true or false. Also, if the set O would be infinite, the boolean approach would quickly become impossible.

Example 47. Let $\mathbf{C} = \mathbf{Set}$ and $\mathbf{D} = \mathbf{BA}$, where the latter denotes the category of Boolean algebras. Again $P = Q^-$, is the contravariant powerset functor, but this time QX is interpreted as a Boolean algebra. The functor Q maps a Boolean algebra to the collection of ultrafilters over it where an ultrafilter on a Boolean algebra is a collection of subsets closed under finite intersections and containing either an element or its complement, serving in the duality context as a complete theory, or a *state of affairs*, that decides for every proposition whether it holds or not [BRV01]. Furthermore $B = (P-)^A$ where P denotes covariant power set that takes an object $X \in \mathbf{C}$ to the set of all subsets of X and given a function $f: X \rightarrow Y$, the map $P(f): P(X) \rightarrow P(Y)$ maps a subset $U \subseteq X$ to its image $f(U) = \{f(x) | x \in U\} \subseteq Y$ and A a set of actions. Coalgebras for this functor correspond to labelled transition systems, where a state has a set of successors that depends on the action/input from A . The dual functor $L: \mathbf{BA} \rightarrow$

$\mathbf{BA}$ is defined as $LY := F_{\mathbf{BA}}(\{\langle a \rangle y \mid a \in A, y \in Y\})/\equiv$ where $F_{\mathbf{BA}}: \mathbf{Set} \rightarrow \mathbf{BA}$ denotes the free Boolean algebra functor and where, roughly speaking, $\equiv$ is the congruence generated from the axioms $\langle a \rangle \perp \equiv \perp$ and $\langle a \rangle (y_1 \vee y_2) \equiv \langle a \rangle (y_1) \vee \langle a \rangle (y_2)$ for each $a \in A$. This is explained in more detail in [KP11]. The initial algebra for this functor is the so called Lindenbaum-Tarski algebra [BRV01] of modal formulas quotiented by logical equivalence. The expressive language for it is defined as:

$$\mathcal{F}(\Lambda) \ni \varphi ::= \perp \mid \phi \vee \phi \mid \neg \phi \mid \langle a \rangle \phi$$

The definition of an appropriate δ can be found in, e.g., [KP11]—the semantics $\llbracket _ \rrbracket$ of a formula then amounts to the standard one [BRV01].

4.1.1 Subobjects and intersection-preserving functors

In our framework, the notion of subobject is important because it provides a way to define the reachable and observable part of the system we aim to learn. Subobjects allow us to identify the minimal structure required to support a specific transition or a logical property, preventing the learner from working with unreachable states. As we will see, the definition of subobject is strictly related to the notion of base. Together, these two concepts serve as the primary ingredients for our algorithm. They ensure that the learner consistently constructs a conjecture based on the minimal and reachable part of the target coalgebra.

We denote by $\mathbf{Sub}(X)$ the collection of subobjects of an object $X \in \mathbf{C}$ [Jac16]. Let $\leq$ be the order on subobjects $s: S \rightarrowtail X, s': S' \rightarrowtail X$ given by $s \leq s'$ if and only if there is $m: S \rightarrow S'$ such that $s = s' \circ m$. The *intersection* $\bigwedge J \rightarrowtail X$ of a family $J = \{s_i: S_i \rightarrowtail X\}_{i \in I}$ is defined as the greatest lower bound with respect to the order $\leq$. In a complete category, it can be computed by (wide) pullback. We denote the maps in the limiting cone by $x_i: \bigwedge J \rightarrowtail S_i$.

For a functor $B: \mathbf{C} \rightarrow \mathbf{D}$, we say B *preserves (wide) intersections* if it preserves these wide pullbacks, i.e., if $(B(\bigwedge J), \{Bx_i\}_{i \in I})$ is the pullback of $\{Bs_i: BS_i \rightarrowtail BX\}_{i \in I}$. By [Adá+13, Lemma 3.53] (building on [Trn71]), *finitary* functors on $\mathbf{Set}$ ‘almost’ preserve

wide intersections: for every such functor B there is a functor B' which preserves wide intersections and agrees with B on all non-empty sets. Finally, if B preserves intersections, then it preserves monos.

Moreover, we will assume that $\mathbf{C}$ is a complete and well-powered category. Intuitively, completeness is required to ensure that we can always compute the intersection of subobjects. This is fundamental to our framework because our algorithm identifies the reachable part of a system by finding the smallest subcoalgebra containing the initial state. Completeness guarantees that such intersection exists and it is itself a valid object. Furthermore, well-poweredness ensures that the collection of all possible subobjects is still a set. As we will see, this allows the collection of subobjects to form a complete lattice, and ultimately allows us to compute the least fixpoints required to formalize the notion of reachability and ensures the algorithm can successfully terminate. Our assumptions imply [Bor94, Proposition 4.4.3] that every morphism f in $\mathbf{C}$ factors uniquely (up to isomorphism) as $f = m \circ e$ with m a mono and e a strong epi where an epimorphism $e: X \rightarrow Y$ is strong if for every commutative square where the bottom arrow is a monomorphism, there exists a unique diagonal morphism d such that the diagram below commutes.

$$\begin{array}{ccc} X & \xrightarrow{e} & Y \\ h \downarrow & \swarrow d & \downarrow g \\ U & \xrightarrow{m'} & Z \end{array}$$

4.1.2 Subformula-closedness

As seen in the previous chapter, a central component of the L^{co} algorithm is the construction of conjectures that are correct with respect to a specific set of formulas. In this chapter, we review this idea through the dual adjunction framework, presenting the notion of subformula-closedness in a more categorical manner. Specifically, the collection of all tests is now formalised as the initial L -algebra $(\Phi, \alpha: L\Phi \rightarrow \Phi)$ providing a formal way to define the increasing of the set of tests at each run of the algorithm. The core idea will not change though: the algorithm will construct conjectures that are *partially* correct, i.e., correct with respect to a subobject of the collection of all formulas/tests. To define this notion of partial correctness we need to consider subob-

jects of Φ to which we can restrict the theory map. This is formalised via the notion of *subformula-closed* subobject of Φ and the definition of such subobjects is based on the notion of *recursive coalgebra*: for an endofunctor $L: \mathbf{D} \rightarrow \mathbf{D}$, a coalgebra $f: X \rightarrow LX$ is called *recursive* if for every L -algebra $g: LY \rightarrow Y$ there is a unique *coalgebra-to-algebra* map $g^\dagger$ making the following diagram commute.

$$\begin{array}{ccc} LX & \xrightarrow{Lg^\dagger} & LY \\ f \uparrow & & g \downarrow \\ X & \xrightarrow{g^\dagger} & Y \end{array}$$

The definition of subformula-closed is now as follows.

Definition 48. A subobject $j: \Psi \rightarrow \Phi$ is called a *subformula-closed collection (of formulas)* if there is a unique L -coalgebra structure $\sigma: \Psi \rightarrow L\Psi$ such that (Ψ, σ) is a recursive L -coalgebra and j is the (necessarily unique) coalgebra-to-algebra map from (Ψ, σ) to the initial algebra (Φ, α) .

Remark 49. The uniqueness of σ in Definition 48 is implied if L preserves monomorphisms. This is the case in our examples. The notion of recursive coalgebra goes back to [Tay99; Osi74]. Also, if L preserves preimages, then the first item of our definition of subformula-closed collection is implied by the second one [ALM07]. In our examples both properties of (Ψ, σ) are verified directly, rather than by relying on general categorical results.

Remark 50. This categorical definition generalizes Definition 32 presented in the previous chapter at page 43 into a single structural property. In the previous setting, we explicitly ensured that if a formula was present in the set Σ , its components were also included in Σ . Categorically, this idea of unfolding is now handled by the L -coalgebra structure σ which maps a formula in Ψ to its immediate subformulas. By requiring that j is a coalgebra-to-algebra map and that the L -coalgebra is recursive, we ensure that this decomposition is consistent with the logic defined in Φ and that every formula can be decomposed into its subformulas. This new definition characterizes a subformula-closed collection as a subobject that is structurally closed under the syntactic operations of L .

The key property of subformula-closed objects Ψ is that we can restrict our attention to the so called Ψ -theory map. Intuitively, the theory map is the map that assigns to each state $x \in X$ the collection of formulas (tests) that x satisfies and the subformula-closedness is what allows us to define this theory map inductively.

Lemma 51. *Let $\Psi \xrightarrow{j} \Phi$ be a subformula-closed collection, with coalgebra structure $\sigma: \Psi \rightarrow L\Psi$. Then $th_\Psi^\gamma = Qj \circ th_\Phi^\gamma$ is the unique map making the following diagram commute.*

$$\begin{array}{ccc} X & \xrightarrow{\quad th_\Psi^\gamma \quad} & Q\Psi \\ \downarrow \gamma & & \uparrow Q\sigma \\ BX & \xrightarrow{Bth_\Psi^\gamma} BQ\Psi \xrightarrow{\delta_\Psi^\flat} & QL\Psi \end{array}$$

We call th_Ψ^γ the Ψ -theory map, and omit the Ψ if it is clear from the context.

Proof.

$$\begin{aligned} th_\Psi^\gamma &= Q\alpha \circ \delta_\Psi^\flat \circ Bth_\Psi^\gamma \circ \gamma \\ &= Q\alpha \circ \delta_\Psi^\flat \circ B(Qj \circ th_\Phi^\gamma) \circ \gamma && \text{By definition of } th_\Psi^\gamma \\ &= Q\alpha \circ \delta_\Psi^\flat \circ BQj \circ Bth_\Phi^\gamma \circ \gamma && \text{By composition of } B \\ &= Q\alpha \circ QLj \circ \delta_\Psi^\flat \circ Bth_\Phi^\gamma \circ \gamma && \text{By naturality of the mate} \\ &= Q\alpha \circ QLj \circ Q\alpha \circ th_\Phi^\gamma && \text{By subformula-closedness} \\ &= Qj \circ th_\Phi^\gamma \end{aligned}$$

Moreover, because the theory map is unique and also j is the unique coalgebra-to-algebra map, then th_Ψ^γ is unique. $\square$

4.1.3 The base

Now, we introduce the categorical notion of base of an endofunctor, following the work presented in [Blo12]. Continuing the transition from the set-theoretic approach of the previous chapter to our current framework, we rephrase the core components of the L^{co} algorithm through a categorical point of view. The base allows us to formally define the direct successors of states within a coalgebra. By identifying these successors, we can categorically characterize reachability, providing another important ingredient required

for the learning algorithm.

Definition 52. Let $B: \mathbf{C} \rightarrow \mathbf{C}$ be an endofunctor. We say B has a base if for every arrow $f: X \rightarrow BY$ there exist $g: X \rightarrow BZ$ and $m: Z \rightarrowtail Y$ with m a monomorphism such that $f = Bm \circ g$, and for any pair $g': X \rightarrow BZ', m': Z' \rightarrowtail Y$ with $Bm' \circ g' = f$ and m' a monomorphism there is a unique arrow $h: Z \rightarrow Z'$ such that $Bh \circ g = g'$ and $m' \circ h = m$, as per the following diagram.

$$\begin{array}{ccccc}
 & & f & & \\
 & \nearrow & & \searrow & \\
 X & \xrightarrow{g} & BZ & \xrightarrow{Bm} & BY \\
 & \searrow & \downarrow Bh & \nearrow & \\
 & & BZ' & \xrightarrow{Bm'} &
 \end{array}$$

g' $\nearrow$ BZ' $\xrightarrow{Bm'}$

We call (Z, g, m) the B -base of the morphism f .

We sometimes refer to $m: Z \rightarrowtail Y$ as the base of f , omitting the g when it is irrelevant, or clear from the context. Note that the terminology *the* base is justified, as it is easily seen to be unique up to isomorphism [Blo12].

For example, let $B: \mathbf{Set} \rightarrow \mathbf{Set}$, $BX = 2 \times X^A$. The base of a map $f: X \rightarrow BY$ is given by $m: Z \rightarrowtail Y$, where $Z = \{(\pi_2 \circ f)(x)(a) \mid x \in X, a \in A\}$, and m is the inclusion. The associated $g: X \rightarrow BZ$ is the corestriction of f to BZ .

For $B = (P-)^A: \mathbf{Set} \rightarrow \mathbf{Set}$, the B -base of $f: X \rightarrow BY$ is given by the inclusion $m: Z \rightarrowtail Y$, where $Z = \{y \in Y \mid \exists x \in X, \exists a \in A \text{ such that } y \in f(x)(a)\}$ and also in this example, the function $g: X \rightarrow BZ$ is the corestriction of f to BZ .

Proposition 53. Suppose $\mathbf{C}$ is complete and well-powered, and $B: \mathbf{C} \rightarrow \mathbf{C}$ preserves (wide) intersections. Then B has a base.

Proof. Let $f: X \rightarrow B(Y)$. Consider the collection of all pairs of maps $g_k: X \rightarrow B(U_k)$, $m_k: U_k \rightarrow Y$ such that $B(m_k) \circ g_k = f$ and m_k is a subobject, indexed by $k \in K$. Let $m: \bigwedge \{m_k\}_{k \in K} \rightarrow Y$ be the intersection of all the m_k , this is a (small) set since $\mathbf{C}$ is well-powered. We abbreviate $\bigwedge \{m_k\}_{k \in K}$ by I .

Since B preserves intersections, $B(m): B(I) \rightarrow B(Y)$ is the intersection of all the subobjects $B(m_k)$. Now the g_k 's form a cone over the $B(m_k)$'s, so we get a unique $g: X \rightarrow B(I)$ from the universal property of the pullback $B(I)$.

We claim that (I, g, m) is the base of f . To see this, first of all, note that m is mono, and $B(m) \circ g = f$ by definition of m and g . Further, if there is any $g': X \rightarrow B(U), m': U \rightarrow Y$ with $B(m') \circ g' = f$ and m' monic then it is (up to isomorphism) one of the g_k, m_k pairs. Hence, there is the map $x_k: I \rightarrow U_k$ in the limiting cone, i.e., $m_k \circ x_k = m$, and we have $B(x_k) \circ g = g_k$. Finally x_k is unique among such maps, since B preserves monos (as it preserves intersections). $\square$

If $\mathbf{C}$ is a locally presentable category, then it is complete and well-powered [AR94, Remark 1.56]. Hence, in that case, any functor $B: \mathbf{C} \rightarrow \mathbf{C}$ which preserves intersections has a base as per following corollary.

Corollary 54. *If $B: \mathbf{C} \rightarrow \mathbf{C}$ preserves intersections and $\mathbf{C}$ is locally presentable, then B has a base.*

Proof. Any locally presentable category is well-powered and complete, see Remark 1.56 in Adamek/Rosicky. $\square$

The following lemma establishes a naturality property for the base, ensuring it behaves consistently when we map one object into another. Intuitively, the following result states that the categorical notion of reachability is stable: the set of reachable successors in the system we are learning corresponds exactly to the successors in the origin system. It is crucial for the proof of Theorem 68, where we show that the existence of a conjecture implies the closedness property of the observation table. Specifically, this lemma guarantees that the successors identified by the algorithm via the theory map correspond to the actual successors in the original system.

Lemma 55. *Let $B: \mathbf{C} \rightarrow \mathbf{C}$ be a functor that has a base and that preserves preimages. Let $f: S \rightarrow BX$ and $h: X \rightarrow Y$ be morphisms, let (Z, g, m) be the base of f and let $e: Z \rightarrow W, m': W \rightarrow Y$ be the (strong epi, mono)-factorisation of $h \circ m$. Then $(W, Be \circ g, m')$ is the base of $Bh \circ f$.*

Proof. By our assumption on Z there exists a morphism $g: S \rightarrow BZ$ such that $Bm \circ g = f$. Therefore we have $Bm' \circ Be \circ g = Bh \circ f$ which shows that m' is a candidate for

the base of $Bh \circ f$. We still need to check the universal property of the base. To this aim let $g' : S \rightarrow BU$ and $n : U \rightarrow Y$ be the base of $Bh \circ f$:

$$\begin{array}{ccccc}
 S & \xrightarrow{f} & BX & \xrightarrow{Bh} & BY \\
 & \searrow g & \nearrow Bm & & \nearrow Bm' \\
 & & BZ & \xrightarrow{Be} & BW \\
 & \searrow g' & & \nearrow Bj & \nearrow Bn \\
 & & BU & &
 \end{array}$$

By the universal property of the base there is a morphism $j : U \rightarrow W$ making the lower right diagram commute. Now, consider the following pullback:

$$\begin{array}{ccc}
 P & \xrightarrow{p_n} & X \\
 p_h \downarrow & \lrcorner & \downarrow h \\
 U & \xrightarrow{n} & Y
 \end{array}$$

This is a preimage because n is mono and by assumption on B we have that this pullback is preserved under application of B . S forms a cone over the diagram with $S \rightarrow BX, S \rightarrow BU$. So there exists a map from S to the pullback.

$$\begin{array}{ccccc}
 S & \xrightarrow{f} & BX & \xrightarrow{Be} & BY \\
 & \searrow g & \nearrow Bm & & \nearrow Bm' \\
 & & BZ & \xrightarrow{Be} & BW \\
 & \searrow w & \downarrow Bk & \nearrow Bd & \nearrow Bn \\
 & & BP & \xrightarrow{Bj} & BU \\
 & \searrow g' & & &
 \end{array}$$

p_n is mono, so $Bp_n \circ w$ is a base factorization. So, we get an arrow from BZ to BP , i.e., $k : Z \rightarrow P$ such that:

(i) $Bk \circ g = w$

(ii) $p_n \circ k = m$

Consider the following diagram. According to the diagonal filling property, we have

$\exists! d: W \rightarrow U$ such that (i) $n \circ d = m'$ and (ii) $d \circ e = p_h \circ k$.

$$\begin{array}{ccc}
 Z & \xrightarrow{e} & W \\
 k \downarrow & \nearrow \exists! d & \downarrow m' \\
 P & & Y \\
 p_h \downarrow & \nearrow n & \\
 U & \xrightarrow{n} & Y
 \end{array}$$

By the universal property of the base we have $d \circ j = id_U$. Moreover, $m' \circ j \circ d = n \circ d = m'$, and because m' is monic we have $j \circ d = id_W$. $\square$

The B -base provides an elegant way to relate reachability within a coalgebra to a monotone operator on the (complete) lattice of subobjects of the carrier of the coalgebra. Moreover, we will see that the least subcoalgebra that contains a given subobject of the carrier can be obtained via a standard least fixpoint construction. Finally, we will introduce the notion of prefix-closed subobject of a coalgebra, generalising the prefix-closedness condition from Angluin's algorithm.

By our assumption on $\mathbf{C}$, the collection of subobjects $(\text{Sub}(X), \leq)$ ordered as usual forms a complete lattice. This can be stated in the following lemma.

Lemma 56. *If $\mathbf{C}$ is complete and well-powered, then for each $X \in \mathbf{C}$, we have $\text{Sub}(X)$ has arbitrary meets. Consequently, $\text{Sub}(X)$ is a complete lattice.*

Proof. Consider some $X \in \mathbf{C}$ and an arbitrary family of subobjects $\{m_i: S_i \rightarrow X\}_{i \in I}$. Let P be the pullback with pullback maps $\{p_i: P \rightarrow S_i\}_{i \in I}$. As the m_i are mono, the p_i are mono as well, so let define $p := m_i \circ p_i: P \rightarrow X \in \text{Sub}(X)$. Obviously, we have $p \leq m_i$ for all $i \in I$. So, P is a lower bound. To see that P is the greatest lower bound, consider an arbitrary P' with $p': P' \rightarrow X$ that is a lower bound of the same family of subobjects. By definition of lower bound we have for each $i \in I$ a map $p'_i: P' \rightarrow S_i$ such that $m_i \circ p'_i = p'$. By universal property of the (wide) pullback, there exists a unique map $c: P' \rightarrow P$ such that $p \circ c = p'$, i.e., $p' \leq p$. As P' was an arbitrary lower bound, we showed that P is the greatest lower bound. This finishes the proof of the fact that $\text{Sub}(X)$ has arbitrary meets. Completeness of the lattice can now be proven

in a standard way by defining the join of an arbitrary collection of subobjects as the meet of all upper bounds of this collection. $\square$

Recall that the meet on $\mathbf{Sub}(X)$ (intersection) is defined via pullbacks. In categories with coproducts, the join $s_1 \vee s_2$ of subobjects $s_1, s_2 \in \mathbf{Sub}(X)$ is defined as the mono part of the factorisation of the map $[s_1, s_2]: S_1 + S_2 \rightarrow X$, i.e., $[s_1, s_2] = (s_1 \vee s_2) \circ e$ for a strong epi e . In $\mathbf{Set}$, this amounts to taking the union of subsets.

For a binary join $s_1 \vee s_2$ we denote by $inl_\vee: S_1 \rightarrow (S_1 \vee S_2)$ and $inr_\vee: S_2 \rightarrow (S_1 \vee S_2)$ the embeddings that exist by $s_i \leq s_1 \vee s_2$ for $i = \{1, 2\}$. Let us now define a key operator that we need in our further discussion.

Definition 57. *Let B be a functor that has a base, $s: S \rightarrowtail X$ a subobject of some $X \in \mathbf{C}$ and let (X, γ) be a B -coalgebra. Let $(\Gamma(S), g, \Gamma_\gamma^B(s))$ be the B -base of $\gamma \circ s$, as per the following diagram.*

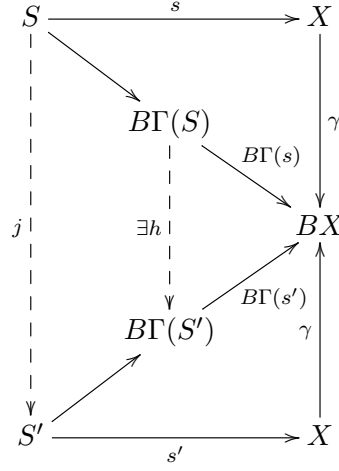
$$\begin{array}{ccc} S & \xrightarrow{s} & X \\ \downarrow g & & \downarrow \gamma \\ B\Gamma(S) & \xrightarrow{B\Gamma_\gamma^B(s)} & BX \end{array}$$

Whenever B and γ are clear from the context, we write $\Gamma(s)$ instead of $\Gamma_\gamma^B(s)$.

Lemma 58. *Let $B: \mathbf{C} \rightarrow \mathbf{C}$ be a functor with a base and let (X, γ) be a B -coalgebra. The operator $\Gamma: \mathbf{Sub}(X) \rightarrow \mathbf{Sub}(X)$ defined by $s \mapsto \Gamma(s)$ is monotone.*

Proof. It is obvious that Γ is well-defined. To check monotonicity, we consider the

following diagram for subobjects $s : S \rightarrow X$ and $s' : S' \rightarrow X$ such that $s \leq s'$.

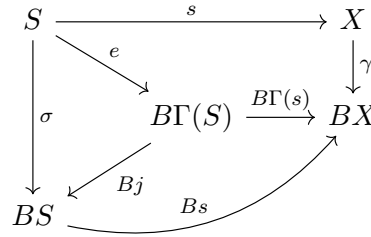


Here j exists by the definition of $\leq$ and h exists by the universal property of the base of $\gamma \circ s$. Therefore we have $\Gamma(s) \leq \Gamma(s')$ as required. $\square$

Intuitively, for a given set of states S , Γ computes the set of *immediate successors*, i.e., the set of states that can be reached by applying γ to an element of S . We will see that pre-fixpoints of Γ correspond to subcoalgebras. Furthermore, Γ is the key to formulate our notion of closed table in the learning algorithm.

Proposition 59. *Let $s : S \rightarrow X$ be a subobject and $(X, \gamma) \in \mathbf{Coalg}(B)$ for $X \in \mathbf{C}$ and $B : \mathbf{C} \rightarrow \mathbf{C}$ a functor that has a base. Then s is a subcoalgebra of (X, γ) if and only if $\Gamma(s) \leq s$. Consequently, the collection of subcoalgebras of a given B -coalgebra forms a complete lattice.*

Proof.



$\Rightarrow$ Consider the diagram above. As s is a subcoalgebra there exists $\sigma : S \rightarrow BS$ such that the outer square commutes. By the universal property of the base there

exists $j: \Gamma(S) \rightarrow S$ such that $s \circ j = \Gamma(s)$. In other words, $\Gamma(s) \leq s$ as required.

$\Leftarrow$ By assumption there exists a $j: \Gamma(S) \rightarrow S$ such that $s \circ j = \Gamma(s)$. We define a B -coalgebra structure on S by putting $\sigma := Bj \circ e$. We have to show that the outer square in the above diagram commutes, but this is easy to show because the inner square commutes by definition of the base, the left triangle commutes by definition of σ and the right one by assumption on j .

Finally, that the collection of subcoalgebras of (X, γ) forms a complete lattice is now a direct consequence of the fact that the collection of pre-fixpoints of the monotone operator Γ forms a complete lattice (Knaster-Tarski theorem). $\square$

Using this connection, reachability of a pointed coalgebra can be expressed in terms of the least fixpoint lfp of an operator defined in terms of Γ .

Theorem 60. *Let $B: \mathcal{C} \rightarrow \mathcal{C}$ be a functor that has a base. A pointed B -coalgebra (X, γ, x_0) is reachable iff $X \cong \text{lfp}(\Gamma \vee x_0)$ (isomorphic as subobjects of X , i.e., equal).*

Proof. Suppose $X \cong \text{lfp}(\Gamma \vee x_0)$, and let $s: S \rightarrow X$ be a subcoalgebra together with an arrow $s_0: 1 \rightarrow S$ with $x_0 = s \circ s_0$. The latter implies that $x_0 \leq s$. Further, since S is a subcoalgebra, by Proposition 59 we get $\Gamma(s) \leq s$. Hence $\Gamma(s) \vee x_0 \leq s$, i.e., s is a pre-fixed point of $\Gamma \vee x_0$. By the Knaster-Tarski theorem (using that $\text{Sub}(X)$ is a complete lattice), $\text{lfp}(\Gamma \vee x_0)$ is the least pre-fixed point, so it now suffices to prove that $s \leq \text{lfp}(\Gamma \vee x_0)$. But this follows easily, since s is a subobject of X and $X \cong \text{lfp}(\Gamma \vee x_0)$.

Conversely, suppose (X, γ, x_0) is reachable. We have that $\Gamma(\text{lfp}(\Gamma \vee x_0)) \leq \Gamma(\text{lfp}(\Gamma \vee x_0)) \vee x_0 = \text{lfp}(\Gamma \vee x_0)$, so by Proposition 59, $\text{lfp}(\Gamma \vee x_0)$ is a subcoalgebra of (X, γ) . Moreover, we have $x_0 \leq \Gamma(\text{lfp}(\Gamma \vee x_0)) \vee x_0 = \text{lfp}(\Gamma \vee x_0)$, so there exists a map $s_0: 1 \rightarrow \text{lfp}(\Gamma \vee x_0)$ such that $x_0 = s \circ s_0$, where $s: \text{lfp}(\Gamma \vee x_0) \rightarrow X$ is the inclusion. Hence, by definition of reachability, we get that s is an isomorphism. $\square$

This justifies defining the reachable part from an initial state $x_0: 1 \rightarrow X$ as the least fixpoint of the monotone operator $\Gamma \vee x_0$. Standard means of computing the least fixpoint by iterating this operator then give us a way to compute this subcoalgebra.

Further, Γ provides a way to generalise the notion of *prefix-closedness* from Angluin's L^* algorithm to our categorical setting.

Definition 61. Let $s_0, s \in \mathbf{Sub}(X)$ for some $X \in \mathbf{C}$ and let (X, γ) be a B -coalgebra. We call s s_0 -prefix-closed with respect to γ if $s = \bigvee_{i=0}^n s_i$ for some $n \geq 0$ and a collection $\{s_i \mid i = 1, \dots, n\}$ with $s_{j+1} \leq \Gamma(\bigvee_{i=0}^j s_i)$ for all j with $0 \leq j < n$.

This categorical definition offers a generalization of the prefix-closedness used in the L^* and L^{co} algorithm. In our previous setting, a set of strings $S \subseteq \Sigma^*$ is prefix-closed if, for every $w \in S$, all prefixes of w are also contained in S . Translated to the set of states of an automaton, this means that every state in S must be reachable from the initial state via a path that stays entirely within S . To see how the categorical definition captures this, consider the category $\mathbf{Set}$ and a DFA given as a coalgebra (X, γ, x_0) with $\gamma: S \rightarrow 2 \times X^\Sigma$ where s_0 is the subobject representing the initial state $\{x_0\} \subseteq X$. Here, the Γ operator characterizes the set of direct successors under all possible inputs from Σ , defined as $\Gamma(S) = \{\delta(x, a) \mid x \in X, a \in \Sigma\}$, while the join $\bigvee$ corresponds in $\mathbf{Set}$ to union. The definition ensures prefix-closedness through an inductive construction: each new subobject s_{j+1} must be a subset of the successors of the states already defined in the previous steps. This ensures that no state can be added to the collection unless its parent state is already present, effectively reconstructing the path-dependency of prefixes in a general way.

4.2 Learning via duality

We will now define a general learning algorithm for B -coalgebras. First, we describe the setting, in general and slightly informal terms. The teacher has a pointed B -coalgebra (X, γ, s_0) .

Our task is to *learn* a pointed B -coalgebra $(S, \hat{\gamma}, \hat{s}_0)$ such that:

- $(S, \hat{\gamma}, \hat{s}_0)$ is *correct* with respect to the collection Φ of all tests, i.e., the theory of (X, γ) and $(S, \hat{\gamma})$ coincide on the initial states s_0 and $\hat{s}_0$, (Definition 69);
- $(S, \hat{\gamma}, \hat{s}_0)$ is minimal with respect to logical equivalence;

- $(S, \hat{\gamma}, \hat{s}_0)$ is reachable.

The first point means that the learned coalgebra is *correct*, that is, it agrees with the coalgebra of the teacher on all possible tests from the initial state. The other two points state that the learned coalgebra is the minimal one and it contains all and only the states that are reachable from the initial state.

Concretely, we are only provided limited access to the coalgebra $\gamma: X \rightarrow BX$. Indeed, the teacher gives us:

- for any subobject $S \rightarrowtail X$ and subformula-closed subobject Ψ of Φ , the composite theory map $S \rightarrowtail X \xrightarrow{th_\Psi^\gamma} Q\Psi$;
- for $(S, \hat{\gamma}, \hat{s}_0)$ a pointed coalgebra, whether or not it is correct with respect to the collection Φ of all tests;
- in case of a negative answer to the previous question, a *counterexample*, which essentially is a subobject Ψ' of Φ representing some tests on which the learned coalgebra is wrong (defined more precisely below);
- for a given subobject S of X , the ‘next states’; formally, the computation of the B -base of the composite arrow $S \rightarrowtail X \xrightarrow{\gamma} BX$.

The first three points correspond respectively to the standard notions of membership query (*filling in* the table with rows S and columns Ψ), equivalence query and counterexample generation. The last point, about the base, is more unusual: it does not occur in the L^* algorithm, since there a canonical choice of (X, γ) is used, which allows to represent next states in a fixed manner. It is required in our setting of an arbitrary coalgebra (X, γ) and it has been required also in the L^{co} algorithm since we need to compute the base to perform what we called *successor queries*.

Now, we describe the abstract learning algorithm and its correctness. First, we describe the basic ingredients needed for the algorithm: tables, closedness, counterexamples and a procedure to close a given table. Based on these notions, the actual algorithm is presented, followed by proofs of correctness and termination.

Assumption 62. *Throughout this section, we assume*

- that we deal with coalgebras over the base category $\mathbf{C} = \mathbf{Set}$;
- a functor $B: \mathbf{C} \rightarrow \mathbf{C}$ that preserves preimages and wide intersections;
- a category $\mathbf{D}$ with an initial object 0 such that arrows with domain 0 are monic;
- a functor $L: \mathbf{D} \rightarrow \mathbf{D}$ with an initial algebra $L\Phi \xrightarrow{\cong} \Phi$;
- an adjunction $P \dashv Q: \mathbf{C} \rightleftarrows \mathbf{D}^{op}$, and a logic $\delta: LP \Rightarrow PB$.

Moreover, we assume a pointed B -coalgebra (X, γ, s_0) .

Remark 63. We restrict $\mathbf{C}$ to be the category of $\mathbf{Set}$, but a key contribution is to state the algorithm in categorical terms: the assumptions cover a wide class of functors on $\mathbf{Set}$, which is the main direction of generalisation. Further, the categorical approach will enable future generalisations. As said, we restrict what follows to the category of sets, but this framework can be extended to a general category $\mathbf{C}$ under the following assumptions: it is complete, well-powered and satisfies that for all (strong) epis $q: S \rightarrow \bar{S} \in \mathbf{C}$ and all monos $i: S' \rightarrow S$ such that $q \circ i$ is mono there is a morphism $q^{-1}: \bar{S} \rightarrow S$ such that (i) $q \circ q^{-1} = \text{id}$ and $q^{-1} \circ q \circ i = i$. In $\mathbf{Set}$, the existence of such a q^{-1} , i.e., the fact that (strong) epimorphisms split, is a direct consequence of the Axiom of Choice: since q is surjective, q^{-1} is obtained by choosing a preimage in each fibre of q . Determining whether other categories satisfy these assumptions, or how the framework might be modified to accommodate categories such as $\mathbf{Nom}$ or $\mathbf{Vect}$, remains an open problem. In particular, as it stands, our algorithm cannot be directly extended to $\mathbf{Nom}$ [US20].

4.2.1 Tables and counterexamples

Definition 64. A table is a pair $(S \xrightarrow{s} X, \Psi \xrightarrow{i} \Phi)$ consisting of a subobject s of X and a subformula-closed subobject i of Φ .

To make the notation a bit lighter, we sometimes refer to a table by (S, Ψ) , using s and i respectively to refer to the actual subobjects.

The pair (S, Ψ) represents *rows* and *columns* respectively, in the table; the *elements* of the table are given abstractly by the map $th_\Psi^\gamma \circ s$.

$$S \xrightarrow{s} X \xrightarrow{th_\Psi^\gamma} Q\Psi.$$

In particular, if $\mathbf{C} = \mathbf{D} = \mathbf{Set}$ and $Q = 2^-$, then this is a map $S \rightarrow 2^\Psi$, assigning a Boolean-value to every pair of a row (state) and a column (formula).

In the original L^* algorithm, an observation table is defined by two sets of strings, S and E . The set S contains strings that represent the states currently selected by the algorithm. Similarly, in the L^{co} framework, S is a subset of X , representing a selection of some states of the state space of the (potentially infinite) target coalgebra. The current categorical definition generalizes these concepts through the subobject s , which abstractly represents the collection of elements that have been already reached and explored. Furthermore, while E in the L^* algorithm consists of experiments, i.e., strings, used to distinguish states, the L^{co} algorithm moves from strings to formulas. In that setting, the experiments are represented by a subformula-closed set of formulas. In our current categorical definition, we abstract this further: the experiments are no longer just a set of formulas, but a subformula-closed subobject $i: \Psi \rightarrow \Phi$.

For the definition of closedness, we use the operator $\Gamma(S)$ from Definition 57, which characterises the successors of a subobject $S \rightarrow X$.

Definition 65. A table (S, Ψ) is closed if there exists a map $k: \Gamma(S) \rightarrow S$ such that the following diagram commutes.

$$\begin{array}{ccccc} S & \xrightarrow{s} & X & \xrightarrow{th^\gamma} & Q\Psi \\ \uparrow k & & & \nearrow th^\gamma & \\ \Gamma(S) & \xrightarrow{\Gamma(s)} & X & & \end{array}$$

A table (S, Ψ) is sharp if the composite map $S \xrightarrow{s} X \xrightarrow{th^\gamma} Q\Psi$ is monic.

Thus, a table (S, Ψ) is closed if all the successors of states (elements of $\Gamma(S)$) are already represented in S , up to equivalence with respect to the tests in Ψ . In

other terms, the rows corresponding to successors of existing rows are already in the table. Again, this definition provides a categorical abstraction of the logical closedness previously introduced with Definition 36 at page 45. In the setting of the L^{co} algorithm, a table was considered closed if the behaviour of every successor of all the states in S was already represented within the filtration of S , i.e., for every successor, there existed an equivalent state in S modulo the set of logical formulas under consideration. Now, this requirement is captured by the existence of the morphism k . The diagram asserts that the observations of the successors (factoring through $\Gamma(s)$) are already contained within our subobject. In other words, every logical behaviour exhibited by a successor state is already witnessed by some state in S under the theory map th^γ .

Sharpness amounts to minimality with respect to logical equivalence: every row has a unique value. The latter will be an invariant of the algorithm (Theorem 76).

A *conjecture* is a coalgebra on S , which is not quite a subcoalgebra of X : instead, it is a subcoalgebra *up to equivalence with respect to Ψ* . That is, the successors agree up to logical equivalence.

Definition 66. *Let (S, Ψ) be a table. A coalgebra structure $\hat{\gamma}: S \rightarrow BS$ is called a conjecture (for (S, Ψ)) if the following diagram commutes.*

$$\begin{array}{ccccc} S & \xrightarrow{s} & X & \xrightarrow{\gamma} & BX \\ \hat{\gamma} \downarrow & & & & \downarrow Bth^\gamma \\ BS & \xrightarrow{Bs} & BX & \xrightarrow{Bth^\gamma} & BQ\Psi \end{array}$$

Given the definition of conjecture, we can redraw the diagram as follows.

$$\begin{array}{ccccc} S & \xrightarrow{s} & X & & \\ \hat{\gamma} \downarrow & & \downarrow \gamma & & \\ BS & \xrightarrow{Bs} & BX & \xrightarrow{Bth^\gamma} & BQ\Psi \end{array}$$

It is easy to see from the latter diagram, that intuitively the conjecture is an approximation of the target system up to logical equivalence. In fact, given the conjecture $(S, \hat{\gamma})$ in the form of a coalgebra, $s: S \rightarrow X$ is a coalgebra morphism up to Ψ , i.e., the states in S and in X are not mapped to the same elements in BX but their observable

behaviours through the set of formulas Ψ is the same.

We can show that, if the table is sharp and the functor B preserves monos, then given a conjecture, this conjecture is unique.

Lemma 67. *If the table is sharp and B preserves monos, then the conjecture $\hat{\gamma}$ is unique.*

Proof. Suppose that there are two conjectures $\hat{\gamma}_1$ and $\hat{\gamma}_2$ such that that $Bth^\gamma \circ Bs \circ \hat{\gamma}_1 = Bth^\gamma \circ \gamma \circ s$ and $Bth^\gamma \circ Bs \circ \hat{\gamma}_2 = Bth^\gamma \circ \gamma \circ s$. Then, we have $Bth^\gamma \circ Bs \circ \hat{\gamma}_1 = Bth^\gamma \circ Bs \circ \hat{\gamma}_2$. As B is a functor, it preserves composition, so we can rewrite the equation as $B(th^\gamma \circ s) \circ \hat{\gamma}_1 = B(th^\gamma \circ s) \circ \hat{\gamma}_2$. Also, we know that the table is sharp, therefore $(th^\gamma \circ s)$ is mono and since B preserves mono also $B(th^\gamma \circ s)$ is mono. By definition, a morphism f is mono if $f \circ g_1 = f \circ g_2 \Rightarrow g_1 = g_2$, then $\hat{\gamma}_1 = \hat{\gamma}_2$. $\square$

Once we have a closed table, we have to be able to construct a conjecture. The following theorem ensures it.

Theorem 68. *A sharp table is closed if and only if there exists a conjecture for it.*

Proof. Given a table (S, Ψ) that is closed, it is straightforward to construct a conjecture $\hat{\gamma}$ as composite of $g : S \rightarrow B\Gamma(S)$ and the arrow $Bk : B\Gamma(S) \rightarrow S$, where g is part of the base $(\Gamma(S), g, \Gamma(s))$ of $\gamma \circ s$ and $k : \Gamma(S) \rightarrow S$ is the morphism that exists by closedness of (S, Ψ) .

For the converse, consider a conjecture $(S, \hat{\gamma})$ for a sharp table (S, Ψ) and let $(\Gamma(S), g, \Gamma(s))$ be the base of $\gamma \circ s$ and let $(h : \Gamma(S) \rightarrow Y, m : Y \rightarrow Q\Psi)$ be the factorisation of $th^\gamma \circ \Gamma(s)$. By Lemma 55, as h is epi, we have that $(Y, Bh \circ g, m)$ is the base of $Bth^\gamma \circ \gamma \circ s$. The situation is depicted in the (commuting) upper square of the diagram below.

$$\begin{array}{ccccccc}
 S & \xrightarrow{s} & X & \xrightarrow{\gamma} & BX & \xrightarrow{Bth^\gamma} & BQ(\Psi) \\
 & \searrow g & & & & \nearrow Bm & \\
 & & B(\Gamma(S)) & \xrightarrow{Bh} & B(Y) & & \\
 & \searrow \hat{\gamma} & & \nearrow Bj & & \nearrow Bth^\gamma & \\
 & & BS & \xrightarrow{Bs} & BX & &
 \end{array}$$

The bigger outer square commutes by the fact that $\hat{\gamma}$ is assumed to be a conjecture. As the table is sharp, we have $th^\gamma \circ s$ is mono. Therefore the universal property of the base yields existence of a morphism $j : Y \rightarrow S$ such that $th^\gamma \circ s \circ j = m$. We define $k := j \circ h$ and claim that this k is a witness for (S, Ψ) , i.e., that k makes the relevant diagram from Definition 65 commute. To see this, we calculate:

$$th^\gamma \circ \Gamma(s) = m \circ h = th^\gamma \circ s \circ j \circ h = th^\gamma \circ s \circ k.$$

This finishes the proof. □

Theorem 68 is a stronger variation of Proposition 59. While the latter requires that all immediate successors of S are contained within S , i.e., S is already a closed sub-system, the theorem relaxes this condition in favour of an observational one. It establishes that for a closed table, even if a transition from a state in S leads to a state outside of S , there exists an equivalent state in S that is logically indistinguishable with respect to the formulas in Ψ allowing us to treat S as if it is a subcoalgebra modulo the logical theory Ψ .

Our goal is to learn a pointed coalgebra which is correct with respect to all formulas. To this aim we ensure correctness with respect to an increasing sequence of subformula-closed collections Ψ .

Definition 69. *Let (S, Ψ) be a table, and let $(S, \hat{\gamma}, \hat{s}_0)$ be a pointed B -coalgebra on S . We say $(S, \hat{\gamma}, \hat{s}_0)$ is correct with respect to Ψ if the following diagram commutes.*

$$\begin{array}{ccccc} & & X & & \\ & s_0 \nearrow & & \searrow th^\gamma & \\ 1 & \xrightarrow{\hat{s}_0} & S & \xrightarrow{th^{\hat{\gamma}}} & Q\Psi \end{array}$$

As we will see in the Algorithm 3 at page 84, we are constructing conjectures that will be correct with respect to the subformula-closed collection Ψ of formulas under consideration. The aim of the following lemma is to guarantee that every time we have a closed table and hence, we provide a conjecture, that conjecture is correct with respect to the set of formulas we are considering.

Lemma 70. *Suppose (S, Ψ) is closed, and $\hat{\gamma}$ is a conjecture. Then $th_{\Psi}^{\gamma} \circ s = th_{\Psi}^{\hat{\gamma}}: S \rightarrow Q\Psi$. If $\hat{s}_0: 1 \rightarrow S$ satisfies $s \circ \hat{s}_0 = s_0$ then $(S, \hat{\gamma}, \hat{s}_0)$ is correct with respect to Ψ .*

Proof. The map $th^{\hat{\gamma}}$ is, by definition, the unique map making the following diagram commute.

$$\begin{array}{ccc} S & \xrightarrow{th^{\hat{\gamma}}} & Q\Psi \\ \hat{\gamma} \downarrow & & \uparrow d \\ BS & \xrightarrow{Bth^{\hat{\gamma}}} BQ\Psi \xrightarrow{\rho_{\Psi}^b} & QL\Psi \end{array}$$

Here, $d: \Psi \rightarrow L\Psi$ is the coalgebra structure from subformula-closedness of Ψ .

Consider the following diagram.

$$\begin{array}{ccccccc} S & \xrightarrow{s} & X & \xrightarrow{th_{\Psi}^{\gamma}} & Q\Psi \\ \hat{\gamma} \downarrow & & \downarrow \gamma & & \uparrow d \\ BS & \xrightarrow{Bs} & BX & \xrightarrow{Bth^{\hat{\gamma}}} BQ\Psi \xrightarrow{\rho_{\Psi}^b} & QL\Psi \end{array}$$

The rectangle on the right commutes by definition of th_{Ψ}^{γ} . Together with $\hat{\gamma}$ being a conjecture, it follows that the outside of the diagram commutes. Since $th_{\Psi}^{\hat{\gamma}}$ is the unique such map, we have $th_{\Psi}^{\gamma} \circ s = th_{\Psi}^{\hat{\gamma}}$. $\square$

We next define the crucial notion of *counterexample* to a pointed coalgebra: a subobject Ψ' of Ψ on which it is *incorrect*. The role of the counterexample is fundamental to the learning process. In fact, it allows to expand the set of formulas Ψ so that the algorithm can distinguish states that previously appeared equivalent, thereby forcing an increase in the size of the state space. This process ensures that the algorithm continues step by step to refine its conjectures eventually learning the correct coalgebra equivalent to the target one.

Definition 71. *Let (S, Ψ) be a table, and let $(S, \hat{\gamma}, \hat{s}_0)$ be a pointed B -coalgebra on S . Let Ψ' be a subformula-closed subobject of Ψ , such that Ψ is a subcoalgebra of Ψ' . We say Ψ' is a counterexample (for $(S, \hat{\gamma}, \hat{s}_0)$, extending Ψ) if $(S, \hat{\gamma}, \hat{s}_0)$ is not correct with respect to Ψ' .*

The following elementary lemma states that if there are no more counterexamples

for a coalgebra, then it is correct with respect to the object Φ of all formulas.

Lemma 72. *Let (S, Ψ) be a table, and let $(S, \hat{\gamma}, \hat{s}_0)$ be a pointed B -coalgebra on S . Suppose that there are no counterexamples for $(S, \hat{\gamma}, \hat{s}_0)$ extending Ψ . Then $(S, \hat{\gamma}, \hat{s}_0)$ is correct with respect to Φ .*

Proof. If there is no counterexample, then in particular Φ is not a counterexample. The object Φ is subformula-closed subobject of itself, and Ψ is a subcoalgebra of Φ . Hence, by definition of counterexamples, it must be the case that $(S, \hat{\gamma}, \hat{s}_0)$ is correct with respect to Φ . $\square$

The following describes, for a given table, how to extend it with the successors (in X) of all states in S . As we will see below, by repeatedly applying this construction, one eventually obtains a closed table.

Definition 73. *Let (S, Ψ) be a sharp table. Let $(\bar{S}, q, r)$ be the (strong epi, mono)-factorisation of the map $th^\gamma \circ (s \vee \Gamma(s))$, as in the following diagram.*

$$\begin{array}{ccccc} S \vee \Gamma(S) & \xrightarrow{s \vee \Gamma(s)} & X & \xrightarrow{th^\gamma} & Q\Psi \\ & \searrow q & & \nearrow r & \\ & & \bar{S} & & \end{array}$$

We define $\text{close}(S, \Psi) := \{\bar{s}: \bar{S} \rightarrow X \mid th^\gamma \circ \bar{s} = r, s \leq \bar{s} \leq s \vee \Gamma(s)\}$. For each $\bar{s} \in \text{close}(S, \Psi)$ we have $s \leq \bar{s}$ and thus $s = \bar{s} \circ \kappa$ for some $\kappa: S \rightarrow \bar{S}$.

This definition provides a categorical formalization of the process already described in the previous chapter in the Algorithm 2 at page 47, where the L^{co} algorithm closes a table by picking a state $x \in S$ whose successors are not yet represented in S modulo the logical formulas in Σ (i.e., $\downarrow_{\text{Base}(\gamma(x))} \subseteq |S|_\Sigma$). Intuitively, that approach focused on identifying specific states whose behaviours were not already captured by the current set of states according to the logical formulas in Σ . Now, we generalize this idea via the (strong epi, mono)-factorization applied to the map $th^\gamma \circ (s \vee \Gamma(s))$.

In this framework, the object $\bar{S}$ represents the collection of unique behaviours (i.e., rows) present in the expanded space of states, i.e., S together with the one-step successors $\Gamma(S)$. The set $\text{close}(S, \Psi)$ then identifies the subobjects $\bar{s}$ that can be used to

extend our current selection. By selecting such a subobject, we ensure that s is expanded just enough to include the missing successor behaviours, allowing us to define a conjecture that is behaviourally consistent with the formulas in Ψ .

Having defined how a table can be extended, we must ensure that such an extension is always possible. Specifically, we need to prove that the expanded state space is related to the current one and that the set of subobjects needed to close the table is non-empty and this is crucial for the algorithm's progress. In fact, if the set $\text{close}(S, \Psi)$ were empty, the algorithm would not be able to progress and close the table. To guarantee it, we introduce the following two lemmas: Lemma 74 ensures that the relationship between S and the expanded object is uniquely determined and consistent with the theory map, while Lemma 75 establishes that a valid expansion of S is guaranteed to exist.

Lemma 74. *In Definition 73, for each $\bar{s} \in \text{close}(S, \Psi)$, we have $\kappa = q \circ \text{inl}_V$.*

Proof. Let $\bar{s} \in \text{close}(S, \Psi)$. We calculate:

$$\begin{aligned} th^\gamma \circ \bar{s} \circ \kappa &= th^\gamma \circ s = th^\gamma \circ (s \vee \Gamma(s)) \circ \text{inl}_V = r \circ q \circ \text{inl}_V \\ &= th^\gamma \circ \bar{s} \circ q \circ \text{inl}_V \end{aligned}$$

which implies $\kappa = q \circ \text{inl}_V$ as $r = th^\gamma \circ \bar{s}$ is a mono. □

We will refer to $\kappa = q \circ \text{inl}_V$ as the connecting map from s to $\bar{s}$.

Lemma 75. *In Definition 73, if there exists $q^{-1}: \bar{S} \rightarrow S \vee \Gamma(S)$ such that $q \circ q^{-1} = \text{id}$ and $q^{-1} \circ q \circ \text{inl}_V = \text{inl}_V$, then $\text{close}(S, \Psi)$ is non-empty.*

Proof. Given the assumption of the lemma we are able to define a morphism $\bar{s}: \bar{S} \rightarrow X$ by putting $\bar{s} := (s \vee \Gamma(s)) \circ q^{-1}$. Obviously $\bar{s}$ is a mono as it is defined as composition of monos. Furthermore, by definition, we have $\bar{s} \leq s \vee \Gamma(s)$. To see that $s \leq \bar{s}$, we calculate

$$\bar{s} \circ q \circ \text{inl}_V = (s \vee \Gamma(s)) \circ q^{-1} \circ q \circ \text{inl}_V = (s \vee \Gamma(s)) \circ \text{inl}_V = s.$$

Finally, the condition concerning the theory map also follows easily:

$$th^\gamma \circ \bar{s} = th^\gamma \circ (s \vee \Gamma(s)) \circ q^{-1} = r \circ q \circ q^{-1} = r.$$

This finishes the proof of the lemma. □

By our assumptions, the hypothesis of Lemma 75 is satisfied (Remark 63), hence $\text{close}(S, \Psi)$ is non-empty. It is precisely (and only) at this point that we need the strong condition about existence of right inverses to epimorphisms.

4.2.2 The algorithm

Having defined closedness, counterexamples and a procedure for closing a table, we are ready to define the abstract algorithm. In the algorithm, the teacher has access to a function $\text{counter}((S, \hat{\gamma}, \hat{s}_0), \Psi)$, which returns the set of all counterexamples (extending Ψ) for the conjecture $(S, \hat{\gamma}, \hat{s}_0)$. If this set is empty, the coalgebra $(S, \hat{\gamma}, \hat{s}_0)$ is correct (see Lemma 72), otherwise the teacher picks one of its elements Ψ' .

We also make use of $\text{close}(S, \Psi)$, as given in Definition 73.

Algorithm 3 Abstract learning algorithm**Input:** (X, γ, s_0) **Output:** $(S, \hat{\gamma}, \hat{s}_0)$

```

1:                                     ▷ Initialization
2:  $(S \xrightarrow{s} X) \leftarrow (1 \xrightarrow{s_0} X)$ 
3:  $\hat{s}_0 \leftarrow \text{id}_1$ 
4:  $\Psi \leftarrow 0$ 
5:                                     ▷ Main
6: while true do
7:   while  $(S \xrightarrow{s} X, \Psi)$  is not closed do
8:     let  $(\bar{S} \xrightarrow{\bar{s}} X) \in \text{close}(S, \Psi)$ , with connecting map  $\kappa: S \rightarrow \bar{S}$ 
9:      $(S \xrightarrow{s} X) \leftarrow (\bar{S} \xrightarrow{\bar{s}} X)$ 
10:     $\hat{s}_0 \leftarrow \kappa \circ \hat{s}_0$ 
11:   end while
12:   let  $(S, \hat{\gamma})$  be a conjecture for  $(S, \Psi)$ 
13:   if  $\text{counter}((S, \hat{\gamma}, \hat{s}_0), \Psi) = \emptyset$  then
14:     return  $(S, \hat{\gamma}, \hat{s}_0)$ 
15:   else
16:      $\Psi \leftarrow \Psi'$  for some  $\Psi' \in \text{counter}((S, \hat{\gamma}, \hat{s}_0), \Psi)$ 
17:   end if
18: end while

```

The algorithm takes as input the coalgebra (X, γ, s_0) (which we fixed throughout this section). In every iteration of the outside loop, the table is first closed by repeatedly applying the procedure in Definition 73. Then, if the conjecture corresponding to the closed table is correct, the algorithm returns it (Line 14). Otherwise, a counterexample is chosen (Line 16), and the algorithm continues.

4.2.3 Termination and correctness

Correctness is stated in Theorem 80. It relies on establishing loop invariants as per the following theorem.

Theorem 76. *The following is an invariant of both loops in Algorithm 3:*

1. (S, Ψ) is sharp,
2. $s \circ \hat{s}_0 = s_0$, and
3. s is s_0 -prefix-closed with respect to γ .

Chapter 4. Learning via Duality

We need a few auxiliary lemmas to prove the above theorem.

Lemma 77. *If (S, Ψ) is sharp, then $(\bar{s}: \bar{S} \rightarrow X, \Psi)$ is sharp for any $\bar{s} \in \text{close}(S, \Psi)$.*

Proof. This follows immediately from Definition 73, since $th^\gamma \circ \bar{s} = r$, where r is monic. $\square$

Lemma 78. *Let Ψ and Ψ' be subformula-closed, with Ψ a subcoalgebra of Ψ' , witnessed by a mono $i: \Psi \rightarrow \Psi'$. Then we have $Qi \circ th_{\Psi'}^\gamma = th_\Psi^\gamma$.*

Proof. Let $\sigma': \Psi' \rightarrow L\Psi'$ and $\sigma: \Psi \rightarrow L\Psi$ be the coalgebra structures from subformula-closedness of Ψ' and Ψ respectively. Consider the diagram below.

$$\begin{array}{ccccccc}
 X & \xrightarrow{th_{\Psi'}^\gamma} & Q(\Psi') & \xrightarrow{Qi} & Q\Psi \\
 \gamma \downarrow & & \uparrow \sigma' & & \uparrow \sigma \\
 BX & \xrightarrow{Bth_{\Psi'}^\gamma} & BQ(\Psi') & \xrightarrow{\rho_{\Psi'}^b} & QL(\Psi') & \xrightarrow{QLi} & QL\Psi \\
 & & \searrow BQi & & \nearrow \rho_\Psi^b & & \\
 & & & BQ\Psi & & &
 \end{array}$$

By definition, $th_{\Psi'}^\gamma$ is the unique map making the left rectangle commute. The (right) square commutes by assumption that i is a coalgebra homomorphism, and the (lower) triangle by naturality. Since th_Ψ^γ is the unique map such that $th_\Psi^\gamma = Q\sigma \circ \rho_\Psi^b \circ Bth_{\Psi'}^\gamma \circ \gamma$, we have $th_\Psi^\gamma = Qi \circ th_{\Psi'}^\gamma$. $\square$

Lemma 79. *Suppose (S, Ψ) is sharp, and Ψ' is a counterexample. Then (S, Ψ') is again sharp.*

Proof. Let $i: \Psi \rightarrow \Psi'$ be the inclusion of the coalgebra (Ψ, σ) into (Ψ', σ') . By Lemma 78, we have $Qi \circ th_{\Psi'}^\gamma = th_\Psi^\gamma$. Hence $Qi \circ th_{\Psi'}^\gamma \circ s = th_\Psi^\gamma \circ s$, and since $th_\Psi^\gamma \circ s$ is monic, it follows that $th_{\Psi'}^\gamma \circ s$ is monic. $\square$

We are now ready to prove Theorem 76.

Proof. To show that each of these is an invariant of both loops, it suffices to prove that they hold once we enter the first iteration of the outer loop, and that both loops

preserve them (that it holds at the start of each first iteration of the inner loop then follows).

1. (Holds at entry of the outer loop.) At the beginning of the algorithm our table (S, Ψ) is initialised as $(s_0, 0)$. Since Q is a right adjoint, it maps 0 to the terminal object $Q0 = 1$ of $\mathbf{C}$. Hence, the map from $S_0 = 1$ to $Q\Psi$ is of the form $1 \xrightarrow{s_0} X \xrightarrow{th^\gamma} Q\Psi = 1$ which is an iso, so in particular monic.

(Preserved by the inner loop.) This follows from Lemma 77.

(Preserved by the loop body.) If (S, Ψ) is sharp on entry of the body of the outer loop, and the inner loop terminates, then (S, Ψ) is again sharp at Line 12. It only remains to show that if Ψ' is a counterexample (extending Ψ') for a conjecture for (S, Ψ) , then (S, Ψ') is sharp. This follows, in turn, from Lemma 79.

2. (Holds at entry of the outer loop.) Follows immediately from the first two lines of the algorithm.

(Preserved by the inner loop.) Suppose $\hat{s}_0 \circ s = s_0$, and let $\bar{s} \in \text{close}(S, \Psi)$. We need to prove that $\bar{s}_0 \circ \kappa \circ \hat{s}_0 = s_0$ where $\kappa : S \rightarrow \bar{S}$ is the connecting map. Indeed, we have $\bar{s}_0 \circ \kappa \circ \hat{s}_0 = s \circ \hat{s}_0 = s_0$, by definition of κ and assumption, respectively.

(Preserved by the outer loop.) This follows immediately from preservation by the inner loop.

3. Clearly the initial configuration $(s_0, 0)$ is s_0 -prefix-closed. Suppose now that (S, Ψ) is a table with s being s_0 -prefix-closed. We need to check that any $\bar{s} \in \text{close}(S, \Psi)$ is s_0 -prefix-closed as well. By assumption on (S, Ψ) we have $s = \bigvee_{i=0}^n s_i$ for a suitable family of subobjects $s_0, \dots, s_n$. Let $\bar{s} \in \text{close}(S, \Psi)$. Then by definition we have $\bar{s} \leq s \vee \Gamma(s)$, so we put $s_{n+1} := \Gamma(s) \wedge \bar{s}$. It is then easy to check that $\bar{s}$ is s_0 -prefix-closed:

$$\bigvee_{i=0}^{n+1} s_i = \bigvee_{i=0}^n s_i \vee s_{n+1} = s \vee (\Gamma(s) \wedge \bar{s}) = (s \vee \Gamma(s)) \wedge (s \vee \bar{s}) = \bar{s}$$

where the last equality follows from $s \leq \bar{s} \leq s \vee \Gamma(s)$. By definition we have $s_{n+1} \leq \Gamma(s) = \Gamma(\bigvee_{i=0}^n s_i)$ as required.

□

After proving the loop invariants of the algorithm, we are now ready to prove the correctness and termination of it.

Theorem 80. *If Algorithm 3 terminates, then it returns a pointed coalgebra $(S, \hat{\gamma}, \hat{s}_0)$ which is minimal with respect to logical equivalence, reachable and correct with respect to Φ .*

Proof. Minimality with respect to logical equivalence follows from the fact that sharpness of the table is maintained throughout. As the algorithm terminated there is no counterexample, which means by Lemma 72 that the coalgebra is correct with respect to Φ . For reachability we show that the pointed coalgebra that is returned by the algorithm is reachable by showing that *any* conjecture that is constructed during the run of the algorithm is reachable. While running the algorithm we will only encounter conjectures that are built from tables that are both sharp and closed. Therefore we consider an arbitrary sharp and closed table (S, Ψ) together with the conjecture $(S, \hat{\gamma})$ that exists according to Theorem 68. We are going to prove that $(S, \hat{\gamma}, \hat{s}_0)$ is reachable.

By Theorem 76 we know that (S, Ψ) is s_0 -prefix-closed. This means that $s = \bigvee_{i=0}^n s_i$ for suitable subobjects $s_1, \dots, s_n \in \text{Sub}(X)$. Suppose now that $(\bar{S}, \bar{\gamma}, \bar{s}_0)$ is a subcoalgebra of $(S, \hat{\gamma}, \hat{s}_0)$ with inclusion $j : \bar{S} \rightarrow S$ such that $j \circ \bar{s}_0 = \hat{s}_0$. We prove by induction on i that $s_i \leq \bar{s}$ for all $i \in \{0, \dots, n\}$ and thus $s \leq \bar{s}$, this will imply $s = \bar{s}$ and thus, as $\bar{s}$ was assumed to be an arbitrary (pointed) subcoalgebra, reachability of $(S, \hat{\gamma}, \hat{s}_0)$.

Case $i = 0$. Then $\bar{s} \circ \bar{s}_0 = s \circ j \circ \bar{s}_0 = s \circ \hat{s}_0 = s_0$ and thus $s_0 \leq \bar{s}$ as required.

Case $i = j + 1$. Then

$$th^\gamma \circ s_{j+1} \leq th^\gamma \circ \Gamma\left(\bigvee_{i=0}^j s_i\right) \stackrel{\text{I.H.}}{\leq} th^\gamma \circ \Gamma(\bar{s}) \stackrel{\text{Thm. 68}}{\leq} th^\gamma \circ \bar{s}$$

where we slightly abuse notation by writing $f \leq g$ for arbitrary morphisms $f : X_1 \rightarrow Y$ and $g : X_2 \rightarrow Y$ if there exists a morphism $m : X_1 \rightarrow X_2$ such that $g \circ m = f$. The inequality implies that there is a map $k_{j+1} : S_i \rightarrow \bar{S}$ such that $th^\gamma \circ \bar{s} \circ k_{j+1} = th^\gamma \circ s_{j+1}$.

This implies

$$th^\gamma \circ s \circ j \circ k_{j+1} = th^\gamma \circ s_{j+1} \quad (4.1)$$

On the other hand, we have $s \circ \text{in}_{j+1} = s_{j+1}$ where $\text{in}_{j+1} = s_{j+1}$ denotes the inclusion of s_{j+1} into s . Therefore we have $th^\gamma \circ s \circ \text{in}_{j+1} = th^\gamma \circ s_{j+1}$. Together with (4.1) this implies $th^\gamma \circ s \circ \text{in}_{j+1} = th^\gamma \circ s \circ j \circ k_{j+1}$. By sharpness of the table s we obtain $\text{in}_{j+1} = j \circ k_{j+1}$ and finally $s_{j+1} = s \circ \text{in}_{j+1} = s \circ j \circ k_{j+1} = s' \circ k_{j+1}$ which shows that $s_{j+1} \leq s'$. This finishes the induction proof. $\square$

In our termination arguments, we have to make an assumption about the coalgebra which is to be learned. It does not need to be finite itself, but it should be finite up to logical equivalence, for example, in the case of deterministic automata the teacher has a (possibly infinite) automaton representing a regular language, but the learned automaton will consist of a finite representation of it. To speak about this precisely, let Ψ be a subobject of Φ . We take a (strong epi, mono)-factorisation of the theory map, i.e., $th_\Psi^\gamma = \left(X \xrightarrow{e_\Psi} |X|_\Psi \xrightarrow{m_\Psi} Q\Psi \right)$ for some strong epi e and mono m . We call the object $|X|_\Psi$ in the middle the Ψ -logical quotient. For the termination result (Theorem 106), $|X|_\Phi$ is assumed to have finitely many quotients and subobjects, which just amounts to finiteness in **Set**. Please note that this logical quotient is the categorical version of what has been defined at page 41, but there is a key difference between the two approaches. In the previous chapter, we assume to have an expressive language so to compute the bisimulation quotient via the logical quotient. In this approach, we just do the logical quotient, i.e., if the language we use is not expressive we might be able to characterise a *weaker* quotient than bisimulation.

We start with termination of the inner while loop (Corollary 83). This relies on two results: first, that once the connecting map κ is an iso, the table is closed, and second, that, under a suitable assumption on the coalgebra (X, γ) , during execution of the inner while loop, the map κ will eventually be an isomorphism.

Theorem 81. *Let (S, Ψ) be a sharp table, let $\bar{S} \in \text{close}(S, \Psi)$ and let $\kappa: S \rightarrow \bar{S}$ be the connecting map. If κ is an isomorphism, then (S, Ψ) is closed.*

Proof. Suppose κ is an isomorphism. Consider the following diagram (using the nota-

tion from Definition 73), where $g: S \rightarrow BX$ is the map which forms the base of $\gamma \circ s$ together with $\Gamma(s): \Gamma(S) \rightarrow X$.

$$\begin{array}{ccccc}
 S & \xrightarrow{s} & X & \xrightarrow{\gamma} & BX \\
 g \downarrow (1) & & B(\Gamma(s)) & \nearrow & \downarrow Bth^\gamma \\
 B(\Gamma(S)) & & & & \\
 Binr_\vee \downarrow (2) & & B(s \vee \Gamma(s)) & \nearrow & \\
 B(S \vee \Gamma(S)) & & & & \\
 Bq \downarrow (3) & & & & \\
 B(\overline{S}) & & Br & \nearrow & \\
 B\kappa^{-1} \downarrow (4) & & & & \\
 BS & \xrightarrow{Bs} & BX & \xrightarrow{Bth^\gamma} & BQ(\Psi)
 \end{array}$$

The inner shapes commute, from top to bottom: (1) by definition the base, (2) by definition of $inr_\vee$, (3) by definition of (q, r) ; for the bottom triangle (4), we have

$$th^\gamma \circ s \stackrel{\text{Def. of } \kappa}{=} th^\gamma \circ \overline{s} \circ \kappa \stackrel{\overline{s} \in \text{close}(S, \Psi)}{=} r \circ \kappa$$

which suffices since κ is an iso. Since the entire diagram commutes, the coalgebra structure on S gives a conjecture for (S, Ψ) . Hence, by Theorem 68, the table is closed. $\square$

Lemma 82. *Consider a sequence of sharp tables $(S_i \xrightarrow{s_i} X, \Psi)_{i \in \mathbb{N}}$ such that $s_{i+1} \in \text{close}(S_i, \Psi)$ for all i . Moreover, let $(\kappa_i: S_i \rightarrow S_{i+1})_{i \in \mathbb{N}}$ be the connecting maps (Definition 73). If the logical quotient $|X|_\Phi$ of X has finitely many subobjects, then κ_i is an isomorphism for some $i \in \mathbb{N}$.*

Proof. First, observe that the S_i 's form an increasing chain of subobjects of X . Since all these tables (S_i, Ψ) are sharp, they give rise to an increasing chain of subobjects of $Q(\Psi)$, by composition with th_Ψ^γ , given by $th_\Psi^\gamma \circ s_i: S_i \rightarrow Q(\Psi)$. By Lemma 78, it follows that each $th_\Phi^\gamma \circ s_i: S_i \rightarrow Q\Phi$ is monic, and we obtain a sequence of subobjects

of Φ :

$$\begin{array}{ccccccc}
 S_0 & \xrightarrow{\kappa_0} & S_1 & \xrightarrow{\kappa_1} & S_2 & \xrightarrow{\kappa_2} & \dots \\
 \downarrow s_0 & & \downarrow s_1 & & \downarrow s_2 & & \\
 X & & X & & X & & \dots \\
 & \searrow th_\Phi^\gamma & \downarrow th_\Phi^\gamma & \swarrow th_\Phi^\gamma & & & \\
 & & Q\Phi & & & &
 \end{array}$$

It follows that this induces a chain of subobjects of $|X|_\Phi$:

$$\begin{array}{ccccccc}
 S_0 & \xrightarrow{\kappa_0} & S_1 & \xrightarrow{\kappa_1} & S_2 & \xrightarrow{\kappa_2} & \dots \\
 \downarrow s_0 & & \downarrow s_1 & & \downarrow s_2 & & \\
 X & & X & & X & & \dots \\
 & \searrow e_\Phi & \downarrow e_\Phi & \swarrow e_\Phi & & & \\
 & & |X|_\Phi & & & &
 \end{array}$$

By assumption, $|X|_\Phi$ has finitely many subobjects, so κ_i must be an isomorphism for some i . □

Corollary 83. *If the Φ -logical quotient $|X|_\Phi$ has finitely many subobjects, then the inner while loop of Algorithm 3 terminates.*

Proof. The while loop computes a chain of subobjects of X as in Lemma 82; in particular, each of these forms a sharp table (with Ψ), since sharpness is an invariant (Theorem 76). Hence, after a finite number of iterations, κ is an iso. By Theorem 81 this implies that (S, Ψ) is closed, which means the guard of the while loop is false. □

Please note that we refer to finitely many subobjects rather than simply finitely many states, as was the case in the previous chapter (cf. Section 3.2 and page 39). This distinction is necessary because the categorical proofs presented here are not limited to the category **Set**, but are developed for any general category $\mathcal{C}$ satisfying certain structural properties. In **Set**, these two notions coincide. However, in other categories a system may have infinitely many states while maintaining a finite-dimensional lattice of subobjects [HSS17].

For the outer loop, we assume that $|X|_\Phi$ has finitely many quotients, ensuring that every sequence of counterexamples proposed by the teacher is finite.

Theorem 84. *If the Φ -logical quotient $|X|_\Phi$ has finitely many quotients and finitely many subobjects, then Algorithm 3 terminates.*

For termination of the outer loop, we need several auxiliary lemmas.

Lemma 85. *Let (S, Ψ) be table, and let Ψ' be a subformula-closed subobject of Φ , such that Ψ is a subcoalgebra of Ψ' . Then there is a unique map q making the following diagram commute.*

$$\begin{array}{ccccc} X & \xrightarrow{e_{\Psi'}} & |X|_{\Psi'} & \xrightarrow{m_{\Psi'}} & Q\Psi' \\ & \searrow e_\Psi & \downarrow q & & \downarrow Qi \\ & & |X|_\Psi & \xrightarrow{m_\Psi} & Q\Psi \end{array}$$

Moreover, this map q is an epimorphism.

Proof. The outside of the diagram commutes by Lemma 78. The map q arises by the unique fill-in property. That q is an epi follows since e_Ψ is an epi, and $e_\Psi = q \circ e_{\Psi'}$. $\square$

Lemma 86. *Let $(S \xrightarrow{s} X, \Psi)$ be a closed table, and $(S, \hat{\gamma}, \hat{s}_0)$ a pointed coalgebra, such that $(S, \hat{\gamma})$ is a conjecture and $s \circ \hat{s}_0 = s_0$. If Ψ' is a counterexample for $(S, \hat{\gamma}, \hat{s}_0)$, then the map $q: |X|_{\Psi'} \rightarrow |X|_\Psi$ from Lemma 85 is not an isomorphism.*

Proof. Suppose that q is an iso; we prove that, in that case, $(S, \hat{\gamma}, \hat{s}_0)$ is correct with respect to Ψ' . Let q^{-1} be the inverse of q . Since $q \circ e_{\Psi'} = e_\Psi$ we also have $e_{\Psi'} = q^{-1} \circ e_\Psi$. Hence, the two shapes on the lower right in the following diagram commute.

$$\begin{array}{ccccccc} S & \xrightarrow{s} & X & \xrightarrow{\gamma} & BX & & \\ \hat{\gamma} \downarrow & & & & \downarrow Be_\Psi & \searrow Be_{\Psi'} & \\ BS & \xrightarrow{Bs} & BX & \xrightarrow{Be_\Psi} & B(|X|_\Psi) & & \\ & & & \searrow Be_{\Psi'} & \downarrow Bq^{-1} & \swarrow & \\ & & & & B(|X|_{\Psi'}) & & \end{array}$$

The rectangle commutes since $\hat{\gamma}$ is a conjecture for the closed table (S, Ψ) . Since the entire diagram commutes, it shows that $(S, \hat{\gamma})$ is a conjecture for the closed table (S, Ψ') as well. Together with $s \circ \hat{s}_0 = s_0$, by Lemma 70, we obtain that $(S, \hat{\gamma}, \hat{s}_0)$ is correct with respect to Ψ' . $\square$

We are now ready to prove Theorem 84.

Proof. The inner while loop terminates in each iteration of the outer loop by Corollary 83. The outer loop generates a sequence $\Psi_0, \Psi_1, \Psi_2, \dots$ of subobjects, such that for each i , there is a pointed coalgebra $(S_i, \hat{\gamma}, \hat{s}_0)$ such that

- $(S_i \xrightarrow{s_i} X, \Psi_i)$ is a closed table,
- $(S_i, \hat{\gamma})$ is a conjecture for this table,
- $s_i \circ \hat{s}_0 = s_0$, and
- Ψ_{i+1} is a counterexample for $(S_i, \hat{\gamma}, \hat{s}_0)$.

We will show that such a sequence is necessarily finite.

By the last point and Lemma 85, for each i , there exists a map $q_{i+1,i}$ making the diagram on the left-hand side commute:

$$\begin{array}{ccc} & X & \\ e_{\Psi_i} \swarrow & & \searrow e_{\Psi_{i+1}} \\ |X|_{\Psi_i} & \xleftarrow{q_{i+1,i}} & |X|_{\Psi_{i+1}} \end{array} \qquad \begin{array}{ccc} & X & \\ e_{\Psi_i} \swarrow & & \searrow e_{\Phi} \\ |X|_{\Psi_i} & \xleftarrow{q_i} & |X|_{\Phi} \end{array}$$

Moreover, again by Lemma 85, for each i , there is a map q_i making the diagram on the right-hand side above commute. For each i , we have

$$q_{i+1,i} \circ q_{i+1} \circ e_\Phi = q_{i+1,i} \circ e_{i+1} = e_{\Psi_i} = q_i \circ e_\Phi$$

and since e_{Φ} is epic, we obtain $q_{i+1,i} \circ q_{i+1} = q_i$. Hence, we get the following sequence of quotients:

$$\begin{array}{ccccc}
& & & & |X|_\Phi \\
& & & \swarrow q_2 & \searrow \dots \\
& & & \Downarrow & \\
& & & |X|_\Psi_2 & \leftarrow \dots \\
& & \swarrow q_1 & \leftarrow q_{2.1} & \\
& & \Downarrow & \leftarrow & \\
& & |X|_\Psi_1 & \leftarrow q_{3.2} & \dots \\
& \swarrow q_0 & \leftarrow q_{1.0} & & \\
& |X|_\Psi_0 & & &
\end{array}$$

It follows from Lemma 86 and the previous assumptions that none of the quotients $q_{i+1,i}$ can be an iso. But since for each i , $|X|_{\Psi_i}$ is a quotient of $|X|_{\Phi}$, and the latter has only finitely many quotients, the sequence of counterexamples must be finite. $\square$

4.3 Learning BT -structures

In this section, we aim to learn a succinct representation of transition structures traditionally identified as B -coalgebras. We achieve this by introducing BT -coalgebras and adapting our learning algorithm to this framework. For instance, it is well-known that non-deterministic finite automata can be much smaller than deterministic automata for a regular language in terms of number of states and transitions [HSS20]. To do so, we first explain the concept of forgetful logics [KR16] and subsequently provide an extension of the algorithm presented in the previous section to cover this objective. We will show how this framework can be used to learn complex structures such as non-deterministic and alternating finite automata. The core of this approach lies in using a composite functor BT to learn B -coalgebras via their equivalent BT -representations, where B defines the transition structure and T characterises the branching behaviour of the system. Forgetful logics will be used as tests. In what follows, we first present forgetful logics, then we extend our categorical definitions to adapt the Algorithm 3 to this new framework and we rephrase the proofs for termination and correctness.

4.3.1 Forgetful logic

Intuitively, a forgetful logic is a logic that forgets its syntax component. In our case, this allows us to maintain the same set of formulas as tests whenever we are modelling coalgebras using B or BT functors. That is, we will have the same theories but different interpretations of them according to the two functors. Indeed, this forgetful logic allows to abstract away from branching. On the other hand, the coalgebraic semantics of the functors B and BT is different. The behavioural functor in the latter case is a composition of B and T where T describes the branching aspect of the transitions and B gives information about the observations.

More technically, as we view logics as natural transformations (cf. Section 2.3), we can compose logics easily [CP04]. That is, given B, L defined as we did previously and two more functors, $T: \mathbf{C} \rightarrow \mathbf{C}$ describing a signature for a coalgebra and $N: \mathbf{D} \rightarrow \mathbf{D}$ being a syntax for T , we can compose their logics and obtain the logic for the functor

composition in the following way: given a logic for B , $\alpha : LP \Rightarrow PB$ and a logic for T , $\beta : NP \Rightarrow PT$, we have a logic for BT as $(\alpha \odot \beta) = \alpha T \circ L \beta = LNP \Rightarrow PBT$. Similarly, we can compose the mates that we defined in Section 2.3 at page 28 of the two logics such that, given $\alpha^b : BQ \Rightarrow QL$ and $\beta^b : TQ \Rightarrow QN$, we obtain the composition of the mate as $(\alpha \odot \beta)^b = \alpha^b N \odot B \beta^b = BTQ \Rightarrow QLN$ by $\alpha^b \odot \beta^b = (\alpha \odot \beta)^b$ [CP04; Jac16].

Following [KR16], we require a logic that is interpretable over both B -coalgebras and BT -coalgebras to be able to have a notion of equivalence between them. In fact, given a logic that can be interpreted over both B and BT -coalgebras, we can determine that those two representations are equivalent if they satisfy exactly the same set of formulas. To achieve this, the functor N is defined as the identity functor ($N = \text{Id}$). The composition of the mates is given then by $(\alpha \odot \beta)^b : BTQ \Rightarrow QL$. In this setting, the branching behaviour of the transition system is summarised within the semantic interpretation. The syntax is unchanged. Indeed, the syntax for BT remains identical to that of B , since $L \circ N = L \circ \text{Id} = L$.

This forgetful logic approach is what enables us to learn a succinct BT -representation (such as an NFA) that remains functionally equivalent to the target B -system (such as a DFA).

Example 87. The following example is paradigmatic for this section. A deterministic finite automaton can be represented as a coalgebra $(X, \gamma : X \rightarrow BX)$ where $X \in \mathbf{Set}$ and $B = 2 \times -^A$, with A the input alphabet. The initial L -algebra is $\Phi = A^*$ whose a structure is defined as $a = [\lambda, \text{cons}] : 1 + A \times A^* \rightarrow A^*$ where λ represents the empty word and cons takes a pair $(u \in A, w \in A^*)$ into the concatenated word $uw \in A^*$. In this setting, if we want to identify states in a coalgebra, we need to check which of them make the same formulas true where a test is true if, starting from a given state, there exists a unique sequence of transitions that ends up in an accepting state of the coalgebra, after reading the test itself. That is, the truth of tests stands for the word acceptance. On the other hand, a non-deterministic automaton (NFA) is viewed as a BT -coalgebra, where $T = P$ is the powerset functor that characterizes the branching. The transition structure is given by $\gamma : X \rightarrow 2 \times P(X)^A$. Although a state $x \in X$ after an input $a \in A$ ends up in a set of next states, we use the same syntax $\Phi = A^*$ to

describe the transition system.

The critical point that relates the DFA and NFA is the forgetful logic that, in the case of the NFA, can intuitively be seen as the union of sets of theories. In fact, while the semantics of the DFA is handled by the mate $\alpha^b: BQ \rightarrow QL$, the NFA semantics is defined by the composite mate $(\alpha \odot \beta)^b$. Here, the branching part represented by the mate $\beta^b: \mathcal{P}Q \rightarrow Q$ takes the set of theories reachable from a state and collapses them into a single theory by checking if at least one transition leads to an accepting state. Intuitively, β^b *summarises away* the branching before α^b processes the observation. The NFA behaves as a DFA from the perspective of the logic. This ensures that the set of possible theories for the NFA can be handled exactly like the DFA case.

The general idea of this section is therefore learning a representation of a pointed B -coalgebra in the form of an equivalent BT -coalgebra using a forgetful logic [KR16] as tests.

The above composition is usually made of a functor $B: \mathbf{Set} \rightarrow \mathbf{Set}$ and in addition a monad T . We will be slightly more general in only requiring T to be a pointed endofunctor, i.e, a functor $T: \mathbf{Set} \rightarrow \mathbf{Set}$ together with a natural transformation $\eta: \text{Id} \Rightarrow T$. Note that since we require that the logic for T consists of a single unary operator that can be eliminated through a syntax representation of logical formulas, the semantics of it is $\beta: P \Rightarrow PT$ and equivalently its mate is $\beta^b: TQ \Rightarrow Q$.

To ensure that our forgetful logic provides an interpretation across both B and BT structures, we require the following.

Assumption 88. β^b is a forgetful logic for the pointed endofunctor T , i.e., a forgetful logic β^b such that the following diagram commutes.

$$\begin{array}{ccc} TQ & \xleftarrow{\eta} & Q \\ \beta^b \downarrow & \swarrow Id & \\ Q & & \end{array}$$

The following proposition rephrases the intuition we previously gave via the DFA and NFA example: evaluating the theory of a state directly yields the same result as embedding that state into a branching structure and then applying the forgetful logic.

Proposition 89. *In our setting, we have that for all $X \in \mathbf{C}$, $\Psi \in \mathbf{D}$, and $f: X \rightarrow Q\Psi$, the following diagram commutes.*

$$\begin{array}{ccc} X & \xrightarrow{f} & Q\Psi \\ \eta_X \downarrow & & \uparrow \beta_\Psi^b \\ TX & \xrightarrow{Tf} & TQ\Psi \end{array}$$

Proof.

$$\begin{aligned} \eta_{Q\Psi} \circ f &= Tf \circ \eta_X && \text{By naturality of } \eta \\ \Rightarrow & \beta_\Psi^b \circ \eta_{Q\Psi} \circ f = \beta_\Psi^b \circ Tf \circ \eta_X && \text{Composition } \beta_\Psi^b \text{ both sides} \\ \Rightarrow & f = \beta_\Psi^b \circ Tf \circ \eta_X && \text{By Assumption 88} \end{aligned}$$

□

4.3.2 BT learning

In what follows, we adapt the definitions and lemmas previously established for B -coalgebras to the composite setting of B and T . Our objective is to provide a version of Algorithm 3 that is capable of learning BT -coalgebras. To ensure that this adapted version of the algorithm is sound, we reformulate the core definitions using these two functors and prove the necessary lemmas to guarantee the algorithm's correctness and termination. Throughout this section, we will highlight the key differences between the previous setting and the one currently under consideration.

We begin by adapting the definition of closedness. Intuitively, a table is closed when all successors of the elements currently in the table can be represented by states already present in that table. In our framework, this requirement is weakened: a table is considered closed if all successor rows are constructible from the rows already selected. Given an observation table, this means that a successor row is not required to be an exact match of an existing row. Instead, it can be viewed as a combination of those rows currently in the table. The specific type of combinations is defined by the functor T . For instance, in our NFA example, T is the powerset functor and here, a successor row is considered constructible if it can be represented by the union of rows already in the

table. This weaker requirement allows the algorithm to represent complex transitions using the existing elements of the observation table, leading to more succinct models.

Definition 90 (*T-closedness*). A table (S, Ψ) is closed if there exists a map $k: \Gamma(S) \rightarrow S$ such that the diagram below commutes.

$$\begin{array}{ccccc} TS & \xrightarrow{Ts} & TX & \xrightarrow{Tth_\gamma} & TQ\Psi \\ \exists k \uparrow & & & & \downarrow \alpha_\Psi^b \\ \Gamma(S) & \xrightarrow{\Gamma(s)} & X & \xrightarrow{th_\gamma^\Psi} & Q\Psi \end{array}$$

As in the previous section, the Γ operator identifies the successors of the elements in the subobject $S \mapsto X$ according to the definition of the base. Since our goal is to learn a B -coalgebra in the form of a BT -coalgebra in order to have a more succinct representation of it, we can maintain the same notion of *next state* as before. Specifically, we define the successors by computing the B -base of the composite arrow $S \mapsto X \xrightarrow{\gamma} BX$.

The underlying idea of the notion of closedness is always the same. A table is closed whenever all successors of all the states we have in the set S can already be represented by elements in S themselves up to equivalence wrt the set of test Ψ . Indeed, we can prove that the definition of T -closedness is implied by the notion of closedness (that is, there exists an arrow $k: \Gamma(S) \rightarrow S$ such that $th_\gamma \circ s \circ k = th_\gamma \circ \Gamma(s)$). Moreover, the definition of sharpness is the same as presented at page 76.

Lemma 91. *Closedness implies T-closedness as shown in the diagram below.*

$$\begin{array}{ccccc} TS & \xrightarrow{Ts} & TX & \xrightarrow{Tth_\gamma^\Psi} & TQ\Psi \\ \eta_S \uparrow & & \eta_X \uparrow & & \downarrow \alpha_\Psi^b \\ S & \xrightarrow{s} & X & \searrow th_\gamma^\Psi & \\ k \uparrow & & & & \\ \Gamma(S) & \xrightarrow{\Gamma(s)} & X & \xrightarrow{th_\gamma^\Psi} & Q\Psi \end{array}$$

Proof. We assume that the table is closed, i.e., there exists a $k: \Gamma(S) \rightarrow S$ such that the bottom diagram commutes. Under this assumption, the composition of k with η gives rise to an arrow which satisfies the definition of T -closedness and we prove it by showing

that the above outer diagram commutes. It commutes because the upper left square commutes by naturality, the bottom diagram commutes, as we said, by closedness and the diagram on the left by Proposition 89. $\square$

Once the observation table is closed, we construct a conjecture in the form of a BT -coalgebra on S . The transition map is defined as $\hat{\gamma}: S \rightarrow BTS$. This is an important difference compared to our previous version. Now, the conjecture incorporates the branching behaviour of T , allowing each state in S to move to a combination of states (such as a set as per NFA example) rather than to a single successor.

Definition 92 (Conjecture). *Let (S, Ψ) be a table, a BT -conjecture for $(X, \gamma: X \rightarrow BX)$ is a BT -coalgebra $(S, \hat{\gamma}: S \rightarrow BTS)$ such that the following diagram commutes.*

$$\begin{array}{ccccccc} S & \xrightarrow{s} & X & \xrightarrow{\gamma} & BX & & \\ \downarrow \hat{\gamma} & & & & \searrow Bth_{\gamma}^{\Psi} & & \\ BTS & \xrightarrow{BTs} & BTX & \xrightarrow{BTth_{\gamma}^{\Psi}} & BTQ\Psi & \xrightarrow{B\alpha^b} & BQ\Psi \end{array}$$

It is important to ensure that once the table is closed, we can effectively construct a conjecture from it. The following theorem proves that.

Theorem 93. *If a sharp (S, Ψ) table is T -closed, then there exists a BT -conjecture $\hat{\gamma}: S \rightarrow BTS$ such that the following diagram commutes.*

$$\begin{array}{ccccccc} S & \xrightarrow{s} & X & \xrightarrow{\gamma} & BX & & \\ & \searrow g & & \nearrow B\Gamma(s) & \downarrow Bth_{\gamma}^{\Psi} & & \\ & & B\Gamma(S) & & BQ\Psi & & \\ \downarrow \hat{\gamma} & & \nwarrow Bk & & \uparrow B\alpha^b & & \\ BTS & \xrightarrow{BTs} & BTX & \xrightarrow{BTth_{\gamma}^{\Psi}} & BTQ\Psi & & \end{array}$$

Proof. The upper triangle commutes by the definition of the B -base of $s \circ \gamma$. The map $g: S \rightarrow B\Gamma(S)$ is the factorisation of the transition map through its base. The lower diagram commutes by the assumption that the table is T -closed. The left-hand side triangle commutes because of the base definition. Hence, the full diagram commutes from the combination of the parts above. $\square$

Chapter 4. Learning via Duality

In what follows, we prove that every conjecture we build up during the algorithm is correct with respect to the set of tests Ψ we are considering. Here, correctness means that given the initial object for X , $x_0: 1 \rightarrow X$ and for S , $s_0: 1 \rightarrow S$, we have $th_\gamma \circ x_0 = th_{\hat{\gamma}} \circ s_0$.

Theorem 94 (Correctness wrt Ψ). *Given a closed table (S, Ψ) and a conjecture $\hat{\gamma}$ for it, if $\hat{s}_0: 1 \rightarrow S$ satisfies $s \circ s_0 = x_0$ then $(S, \hat{\gamma}, \hat{s}_0)$ is correct wrt Ψ , i.e., $th_\gamma^\Psi \circ s = th_{\hat{\gamma}}^\Psi: S \rightarrow Q\Psi$.*

Proof. In order to prove $th_\gamma^\Psi \circ s = th_{\hat{\gamma}}^\Psi$, we have to show that the map $th_\gamma^\Psi \circ s$ makes the following diagram commute.

$$\begin{array}{ccccc}
 S & \xrightarrow{s} & X & \xrightarrow{th_\gamma^\Psi} & Q\Psi \\
 \downarrow \hat{\gamma} & & \downarrow \gamma & & \uparrow Qc \\
 & & BX & \xrightarrow{Bth_\gamma^\Psi} & BQ\Psi \xrightarrow{\rho_\Psi^b} QL\Psi \\
 & & & & \uparrow \rho_\Psi^b \circ B\alpha^b \\
 BTS & \xrightarrow{BTs} & BTX & \xrightarrow{BTth_\gamma^\Psi} & BTQ\Psi
 \end{array}$$

Because $\hat{\gamma}$ is a conjecture for S we have, precomposing with $Qc \circ \rho_\Psi^b$:

$$\begin{aligned}
 Qc \circ \rho_\Psi^b \circ B\alpha^b \circ BTth_\gamma^\Psi \circ BTs \circ \hat{\gamma} &= \text{by Definition 92} \\
 Qc \circ \rho_\Psi^b \circ Bth_\gamma^\Psi \circ \gamma \circ s &= \text{by Lemma 51} \\
 th_\gamma^\Psi \circ s &
 \end{aligned} \tag{4.2}$$

Now, because of Lemma 51, we know that there exists a unique map making the previous diagram commute, i.e., $th_{\hat{\gamma}}^\Psi$, but as we have seen, also $th_\gamma^\Psi \circ s$ does, then $th_\gamma^\Psi \circ s = th_{\hat{\gamma}}^\Psi$. $\square$

The mechanisms for identifying counterexamples and extending the observation table remain identical to those presented in Section 4.2.1. In particular, the definition of a counterexample (Definition 71) and the proof that the absence of counterexamples implies correctness w.r.t the set of formulas under consideration (Lemma 72) are also valid for the BT setting.

Similarly, the process of closing a table by expanding the subobject $S \mapsto X$ via the Γ operator remains the same. The definition of the set $\text{close}(S, \Psi)$ (Definition 73) and the associated lemmas regarding the connecting map κ and the existence of $\bar{s}$ (Lemmas 74 and 75) are unchanged. These results are independent of the branching functor T because they concern the discovery of states in the target system rather than the branching structures expressed by the T functor.

4.3.3 Termination and correctness

We established the key notions of the *BT*-learning framework. Now, we can turn to the algorithm itself and prove its termination and correctness.

The structure of the learning process closely follows Algorithm 3 presented in Section 4.2.2, meaning that the code itself requires no changes and need not be restated here, but some specific lines require reinterpretation within the current setting.

At Line 7, *closed* must now be understood as *T*-closed in the sense we detailed above rather than closed in the sense of Definition 65. At Line 12, *a conjecture for* (S, Ψ) must be understood as a *BT*-conjecture as states in Definition 92, whose transition map has type $\hat{\gamma}: S \rightarrow BTS$ rather than $\hat{\gamma}: S \rightarrow BS$. Correspondingly, the algorithm now returns a *BT*-coalgebra. On the other hand, the initialisation, the procedure $\text{close}(S, \Psi)$ used to grow S , and the handling of counterexamples are unaffected, since they concern the discovery of new *B*-successor states in X via Γ , independently of T .

Furthermore, it is important to notice that $\text{close}(S, \Psi)$ is designed to restore closedness in the original, stronger sense of Definition 65. It grows S whenever a successor is not represented, up to logical equivalence, by a *single* existing state. It does not attempt to first check whether a successor can instead be represented as a *combination* of existing states via T , which is all that *T*-closedness requires. As a consequence, using $\text{close}(S, \Psi)$ unchanged as the growth procedure may add states to S that are not strictly necessary for *T*-closedness, even though it always terminates in a *T*-closed table as we will prove in Proposition 97. Whether close can be refined to first and primarily exploit *T*-combinations before growing S is left open, but as we can see in the worked examples of Section 4.4 this check is possible and can be performed explicitly against all relevant

combinations. As a consequence, the conjecture produced in this setting may not be the canonical one. The algorithm, and the correctness result of Theorem 96, remain valid regardless of which such conjecture is chosen at Line 12, but the resulting output is not canonical in the way it is for plain B -coalgebras. As also noted in the conclusion of this chapter, in [MS22] and [HSS20], the authors explored learning automata with side-effects and proved that this class possesses canonical representatives that can be constructed via a finite number of observations. It is an interesting direction for our future research to investigate whether our construction can be adapted and extended within their framework, so to obtain a canonical conjecture.

Given the above remark, generally speaking, the algorithm continues to operate by closing the observation table in an iterative manner and refining the conjecture based on counterexamples provided by the teacher. However, while the structure of the algorithm remains unchanged, the proofs of correctness and termination must be adapted to the BT framework:

- **Correctness:** in the BT setting, we need to ensure that the summarised theory of the branching transition in the conjecture matches the theory of the target transition. This is mediated by the composite mate $(\alpha \odot \beta)^b$.
- **Termination:** it still relies on the growth of the subobject $S \rightarrowtail X$. However, the use of T -closedness means that the algorithm can terminate with a significantly smaller state space S , as many branching behaviours are now summarised by the logic rather than requiring new states to be added to the table.

As in the previous setting, we represent the teacher as a function $\text{teacher}((S, \hat{\gamma}, \hat{s}_0), \Psi)$, which either returns a set of counterexamples Ψ' extending Ψ or returns $\emptyset$ if the conjecture is correct. In the latter case, the proposed BT -coalgebra $(S, \hat{\gamma}, \hat{s}_0)$ is the final result.

Correctness is stated in Theorem 96 below and it relies on the following establishing loop invariants.

Theorem 95. *The following are invariants of both loops in Algorithm 3:*

1. (S, Ψ) is sharp,

2. $s \circ \hat{s}_0 = s_0$, and

3. s is s_0 -prefix-closed with respect to γ .

Proof. Cf. proof of Theorem 76 at page 84. □

Theorem 96. *If Algorithm 3 terminates, then it returns a pointed coalgebra $(S, \hat{\gamma}, \hat{s}_0)$ which is minimal with respect to logical equivalence and correct with respect to Φ .*

Proof. Minimality with respect to logical equivalence follows from the fact that sharpness of the table is maintained throughout. As the algorithm terminated there is no counterexample, which means by Lemma 72 that the coalgebra is correct with respect to Φ . □

Also in this framework, in order to prove the termination of the algorithm, we still have to assume that the coalgebra we are going to learn is finite up to logical equivalence. That is, the object $|X|_\Psi$, the Ψ -logical quotient has finitely many quotients and subobjects (which just amounts to finitariness, in **Set**), where $|X|_\Psi$ is given as the object in the middle of a (strong epi, mono)-factorisation of the theory map, i.e., $th_\Psi^\gamma: X \twoheadrightarrow |X|_\Psi \rightarrowtail Q\Psi$ for some strong epi e and mono m , and Ψ subobject of Φ .

We start with termination of the inner while loop (Corollary 83). This relies on two results: first, that once the connecting map κ is an isomorphism, the table is closed, and second, that, under a suitable assumption on the coalgebra (X, γ) , during execution of the inner while loop, the map κ will eventually be an isomorphism.

Proposition 97. *If κ is an isomorphism, then (S, Ψ) is T -closed.*

Proof. We know that if κ is an isomorphism, then the table (S, Ψ) is closed as proved in Theorem 81. Moreover, by Lemma 91, we have that closedness implies T -closedness. Hence, the statement of this theorem easily follows. □

The following lemma and corollary correspond to Lemma 82 and Corollary 83, respectively. These results remain valid in the BT setting as the definition of the set $\text{close}(S, \Psi)$ remains unchanged. While the proofs follow the same logic as in the previous section, we restate the results here for clarity and ease of reference in subsequent proofs.

Lemma 98. *Consider a sequence of sharp tables $(S_i \xrightarrow{s_i} X, \Psi)_{i \in \mathbb{N}}$ such that $s_{i+1} \in \text{close}(S_i, \Psi)$ for all i . Moreover, let $(\kappa_i: S_i \rightarrow S_{i+1})_{i \in \mathbb{N}}$ be the connecting maps (Definition 73). If the logical quotient $|X|_\Phi$ of X has finitely many subobjects, then κ_i is an isomorphism for some $i \in \mathbb{N}$.*

Corollary 99. *If the Φ -logical quotient $|X|_\Phi$ has finitely many subobjects, then the inner while loop of Algorithm 3 terminates.*

Because the definition of the set $\text{close}(S, \Psi)$ did not change with respect to the one in the previous section and so the requirements on $|X|_\Psi$, the proofs of the Lemma 98 and of Corollary 99 are the same as before.

Now, the last thing left to prove in the termination of the outer loop. For this, as we said, we assume that $|X|_\Phi$ has finitely many quotients in order to ensure that every sequence of counterexamples proposed by the teacher is finite, but before providing it, we need some other lemmas, required in the termination proof.

Intuitively, the following proposition shows that whenever we extend the set of tests, the truth of already performed tests do not change, even when we take into account combinations of states.

Proposition 100. *Let Ψ and Ψ' be subformula-closed, with Ψ a subcoalgebra of Ψ' , witnessed by a mono $i: \Psi \rightarrow \Psi'$. Then we have $Qi \circ \alpha_{\Psi'}^b \circ Th_{\Psi'}^\gamma = \alpha_\Psi^b \circ Th_\Psi^\gamma$.*

$$\begin{array}{ccccc}
 TX & \xrightarrow{Th_{\Psi'}^\gamma} & TQ\Psi' & \xrightarrow{\alpha_{\Psi'}^b} & Q\Psi' \\
 & \searrow Th_\Psi^\gamma & \downarrow TQi & & \downarrow Qi \\
 & & TQ\Psi & \xrightarrow{\alpha_\Psi^b} & Q\Psi
 \end{array}$$

Proof. The triangle commutes because if we let $\sigma': \Psi' \rightarrow L\Psi'$ and $\sigma: \Psi \rightarrow L\Psi$ be the coalgebra structures from subformula-closedness of Ψ' and Ψ respectively and we

consider the diagram below.

$$\begin{array}{ccccc}
 X & \xrightarrow{th_{\Psi'}^\gamma} & Q(\Psi') & \xrightarrow{Qi} & Q\Psi \\
 \gamma \downarrow & & \uparrow \sigma' & & \uparrow \sigma \\
 BX & \xrightarrow{Bth_{\Psi'}^\gamma} & BQ(\Psi') & \xrightarrow{\rho_{\Psi'}^b} & QL(\Psi') & \xrightarrow{QLi} & QL\Psi \\
 & & \searrow BQi & & \nearrow \rho_\Psi^b & & \\
 & & & BQ\Psi & & &
 \end{array}$$

We have by definition, $th_{\Psi'}^\gamma$ is the unique map making the left rectangle commute. The (right) square commutes by assumption that i is a coalgebra homomorphism, and the (lower) triangle by naturality. Since $th_{\Psi'}^\gamma$ is the unique map such that $th_{\Psi'}^\gamma = Q\sigma \circ \rho_\Psi^b \circ Bth_{\Psi'}^\gamma \circ \gamma$, we have $th_{\Psi'}^\gamma = Qi \circ th_{\Psi'}^\gamma$, i.e., $Qi \circ th_{\Psi'}^\gamma = th_{\Psi'}^\gamma$. Also, the square commutes by naturality of α^b . Hence, the full diagram commutes. $\square$

Now, we restate the definition of the logical quotient with respect to the BT settings. In the previous section, the quotient identified individual states that are logically equivalent. Here, the quotient identifies branching structures that are logically equivalent under the forgetful logic.

Definition 101. *Given the theory map for TX , $\alpha_\Psi^b \circ Th_\gamma^\Psi: TX \rightarrow Q\Psi$, we define $|TX|_\Psi$ as the intermediate object obtained from the strong epi mono factorization of that map as follows:*

$$\alpha_\Psi^b \circ Th_\gamma^\Psi: TX \xrightarrow{e} |TX|_\Psi \xrightarrow{m} Q\Psi$$

If we consider the NFA example, an element of TX is a set of states. The theory map here represents the union of all the languages (theories) arising from that set. $|TX|_\Psi$ is the quotient that identifies different sets of states as equivalent if they happen to accept the same language.

Intuitively, if, for example, we are dealing with NFA, the theory map represents the union of all the theories arose from the states and the combinations of states in the NFA and $|TX|_\Psi$ is the quotient of those states with respect to logical equivalence.

The following lemma states that after receiving a counterexample, we will be able

to distinguish more states. That is, the algorithm will add to the table S at least a state or a combination of states that were not in S before.

Lemma 102. *Let (S, Ψ) be a table, and let Ψ' be a subformula-closed subobject of Φ , such that Ψ is a subcoalgebra of Ψ' . Then there is a unique map q making the following diagram commute.*

$$\begin{array}{ccccc}
 & & \alpha_{\Psi'}^b \circ Th_{\gamma}^{\Psi'} & & \\
 & \nearrow & & \searrow & \\
 TX & \xrightarrow{e_{\Psi'}} & |TX|_{\Psi'} & \xrightarrow{m_{\Psi'}} & Q\Psi' \\
 \parallel & & \downarrow \exists q & & \downarrow Qi \\
 TX & \xrightarrow{e_{\Psi}} & |TX|_{\Psi} & \xrightarrow{m_{\Psi}} & Q\Psi \\
 & \searrow & & \nearrow & \\
 & & \alpha_{\Psi}^b \circ Th_{\gamma}^{\Psi} & &
 \end{array}$$

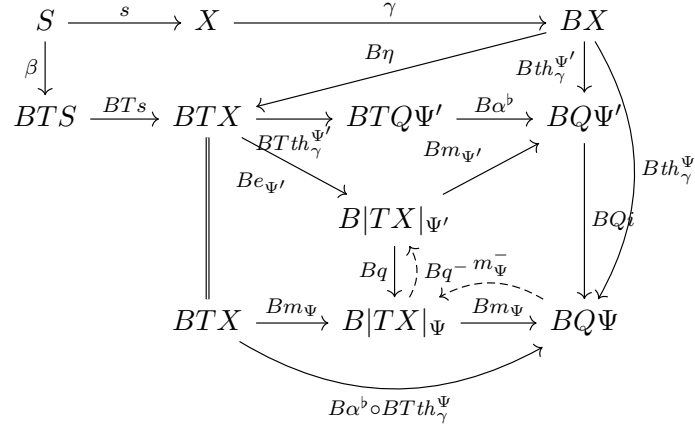
Moreover, this map q is an epimorphism.

Proof. The outer diagram commutes by Lemma 100. According to Definition 101, we can factorize the two theory maps and the unique map $q: |TX|_{\Psi'} \rightarrow |TX|_{\Psi}$ arises from the diagonal fill-in property. Moreover, we know that e_{Ψ} is an epimorphism and that $e_{\Psi} = q \circ e_{\Psi'}$, hence q is an epimorphism. In fact, $f \circ q = g \circ q \Rightarrow f \circ q \circ e_{\Psi'} = g \circ q \circ e_{\Psi'} \Rightarrow f = g$ because $q \circ e_{\Psi'}$ is epi, then q is epi. $\square$

With the following lemma, we demand that the map q of Lemma 102 is not an isomorphism. Intuitively, this requirement aims to ensure that at each step of the algorithm, after receiving a counterexample, the number of quotients will increase. That is, if the conjecture is wrong, we need to have a counterexample that can actually allow to distinguish states.

Lemma 103. *Let (S, Ψ) be a closed table and $(S, \hat{\gamma}, \hat{s}_0)$ a conjecture. If Ψ' is a counterexample then the map $q: |TX|_{\Psi'} \rightarrow |TX|_{\Psi}$ is not an isomorphism.*

Proof. Suppose q is an iso. We show that this implies that $(S, \hat{\gamma}, \hat{s}_0)$ is again a conjecture for (S, Ψ') in order to derive a contradiction, i.e., the counterexample was not able to distinguish more states and therefore, it cannot be a counterexample.



The full outer diagram commutes because we have a conjecture for Ψ . Moreover, m_Ψ is a mono with non-empty domain (in fact, we can assume that $X \neq \emptyset$, since X represents a system and it contains at least one state. Consequently, since T is assumed to be pointed, also $TX \neq \emptyset$), hence we get an inverse such that $Bm_\Psi^- \circ Bm_\Psi = id_{|X|_\Psi}$. Furthermore, because of Assumption 88, we have:

$$B\alpha^b \circ BTth_\gamma^{\Psi'} \circ B\eta = Bth_\gamma^{\Psi'} \quad (4.3)$$

Proving $Bth_\gamma^{\Psi'} \circ \gamma \circ s = B\alpha^b \circ BTth_\gamma^{\Psi'} \circ BTs \circ \beta$ is equivalent to show that $B\alpha^b \circ BTth_\gamma^{\Psi'} \circ B\eta \circ \gamma \circ s = B\alpha^b \circ BTth_\gamma^{\Psi'} \circ BTs \circ \beta$ because of Equation 4.3. Furthermore because $B\alpha_\Psi^b \circ BTth^{\Psi'} = Bm_\Psi' \circ Be_{\Psi'}$, it suffices to prove that

$$Be_{\Psi'} \circ B\eta \circ \gamma \circ s = Be_{\Psi'} \circ BTs \circ \beta \quad (4.4)$$

and Equation 4.4 can be obtained by precomposing both sides with $Bm_{\Psi'}$.

Now,

$$\begin{aligned} BQi \circ Bth_\gamma^{\Psi'} \circ \gamma \circ s &= BQi \circ B\alpha^b \circ BTth_\gamma^{\Psi'} \circ BTs \circ \beta && \text{by Lemma 102} \\ \Rightarrow BQi \circ B\alpha^b \circ BTth_\gamma^{\Psi'} \circ B\eta \circ \gamma \circ s &= BQi \circ B\alpha^b \circ BTth_\gamma^{\Psi'} \circ BTs \circ \beta && \text{by Equation 4.3} \\ \Rightarrow BQi \circ Bm_{\Psi'} \circ Be_{\Psi'} \circ B\eta \circ \gamma \circ s &= BQi \circ Bm_{\Psi'} \circ Be_{\Psi'} \circ BTs \circ \beta && \text{by factorization} \end{aligned}$$

Chapter 4. Learning via Duality

Applying both sides $Bq^- \circ Bm_{\Psi}^-$ and we get:

$$Bq^- \circ Bm_{\Psi}^- \circ BQi \circ Bm_{\Psi'} \circ Be_{\Psi'} \circ B\eta \circ \gamma \circ s = Bq^- \circ Bm_{\Psi}^- \circ BQi \circ Bm_{\Psi'} \circ Be_{\Psi'} \circ BTs \circ \beta$$

And it is what we wanted in Equation 4.4, because

$$Bq^- \circ Bm_{\Psi}^- \circ BQi \circ Bm_{\Psi'} \circ Be_{\Psi'} = Bq^- \circ Bm_{\Psi}^- \circ Bq \circ Be_{\Psi'} = Be_{\Psi'}$$

□

Lemma 104. *If $|X|_{\Phi}$ has only finitely many quotients and T preserves finite sets, then $|TX|_{\Phi}$ is finite and thus has finitely many quotients.*

Proof. By assumption, if $|X|_{\Phi}$ has finitely many quotients, then $|X|_{\Phi}$ is finite. Now, we show that if $|TX|_{\Phi}$ is finite, then also $|TX|_{\Phi}$ is finite and we do that by showing that exists a unique arrow $d: |TX|_{\Phi} \rightarrow |TX|_{\Phi}$ and this arrow is an epimorphism.

$$\begin{array}{ccccc}
 TX & \xrightarrow{Th_{\Phi}} & TQ\Phi & \xrightarrow{\alpha_{\Phi}^b} & Q\Phi \\
 & \searrow Te & \uparrow Tm & & \uparrow \\
 & & T|X|_{\Phi} & & \\
 & \searrow q & \downarrow d & \nearrow m' & \\
 & & |TX|_{\Phi} & &
 \end{array}$$

As we can see from the diagram above, the arrow d exists because of the diagonal fill-in property.

Moreover, given $f, g: |TX|_{\Phi} \rightarrow Q\Phi$, we compose them with d : $f \circ d = g \circ d$. We know that Te is an epimorphism by definition, so we can precompose the previous arrows with it and because $d \circ Te$ is epi, then $f = g$, hence also d is epi. In fact, $f \circ d = g \circ d \Rightarrow f \circ d \circ Te = g \circ d \circ Te \Rightarrow f = g$ □

The following example describes the quotient from the previous Lemma in the case that T is the power set functor.

Example 105. Consider the case $T = P$. In this case we can represent $|PX|_{\Phi}$ and $|X|_{\Phi}$

concretely as follows:

$$\begin{aligned} |X|_{\Phi} &= \{th^{\Phi}(x) \mid x \in X\} \\ |PX|_{\Phi} &= \left\{ \Gamma \mid \exists V \subseteq X. \bigcup \{th^{\Phi}(x) \mid x \in V\} = \Gamma \right\} \end{aligned}$$

It is easy to see that for $Y \subseteq |X|_{\Phi}$ there exists $V_Y \subseteq X$ such that

$$Y = \{th^{\Phi}(x) \mid x \in V_Y\}$$

Therefore the map $d : P|X|_{\Phi} \rightarrow |PX|_{\Phi}$ given by $d(Y) = \bigcup Y$ for all $Y \subseteq |X|_{\Phi}$ is well-defined. Furthermore, given an arbitrary $\Gamma \in |PX|_{\Phi}$, there is some $V \subseteq X$ with $\bigcup \{th^{\Phi}(x) \mid x \in V\} = \Gamma$, i.e., $\Gamma = d(\{th^{\Phi}(x) \mid x \in V\})$. As Γ was arbitrary this shows that d is surjective.

We are now able to prove termination of our algorithm.

Theorem 106. *If the Φ -logical quotient $|TX|_{\Phi}$ has finitely many quotients and finitely many subobjects, then Algorithm 3 terminates.*

Proof. The closing table loop terminates in each iteration of the conjecture loop by Corollary 99. This last loop generates a sequence $\Psi_0, \Psi_1, \Psi_2, \dots$ of subobjects, such that for each i , there is a pointed coalgebra $(S_i, \hat{\gamma}, s_0)$ such that

1. $(S_i \xrightarrow{s_i} X, \Psi_i)$ is a T -closed table,
2. $(S_i, \hat{\gamma}: S_i \rightarrow BTS_i)$ is a conjecture for this table,
3. $s_i \circ s_0 = x_0$, and
4. Ψ_{i+1} is a counterexample for $(S_i, \hat{\gamma}, s_0)$.

We will show that such a sequence is necessarily finite.

Given the following diagram:

$$\begin{array}{ccccc}
 & & TX & & \\
 & e_{\Psi_i} \swarrow & \downarrow e_{\Psi_{i+1}} & \searrow e_{\Phi} & \\
 |TX|_{\Psi_i} & \xleftarrow{q_{i+1,i}} & |TX|_{\Psi_{i+1}} & \xleftarrow{q_{i+1}} & |TX|_{\Phi}
 \end{array}$$

$\xleftarrow{q_i}$

We know that exists a map $q_{i+1,i}$ making the left-hand side triangle commute because of Lemma 102 and the previous point 4. Also, the map $q_{i+1,i}$ is the only candidate that makes the diagram commute. In fact, given two quotient maps $e_{\Psi_i}, e_{\Psi_{i+1}}$, there is at most one morphism from $e_{\Psi_{i+1}}$ to e_{Ψ_i} . Suppose $f, g: |TX|_{\Psi_{i+1}} \rightarrow |TX|_{\Psi_i}$, then $f \circ e_{\Psi_{i+1}} = e_{\Psi_i} = g \circ e_{\Psi_{i+1}}$, but because $e_{\Psi_{i+1}}$ is an epimorphism, we have $f = g$.

Moreover, again by Lemma 102, for each i , there is a map q_{i+1} making the right-hand side triangle commute and for the same reason as above this map is the only candidate.

Lastly, for each i , we have $q_{i+1,i} \circ q_{i+1} \circ e_{\Phi} = q_{i+1,i} \circ e_{\Psi_{i+1}} = e_{\Psi_i} = q_i \circ e_{\Phi}$ and since e_{Φ} is epic, we obtain $q_{i+1,i} \circ q_{i+1} = q_i$. Hence, we get the following sequence of quotients:

$$\begin{array}{ccccccc}
 & & & & |TX|_{\Phi} & & \\
 & & q_0 \swarrow & & \downarrow q_2 & \searrow \dots & \\
 |TX|_{\Psi_0} & \xleftarrow{q_{1,0}} & |TX|_{\Psi_1} & \xleftarrow{q_{2,1}} & |TX|_{\Psi_2} & \xleftarrow{q_{3,2}} & \dots
 \end{array}$$

It follows from Lemma 103 and the previous assumptions that none of the quotients $q_{i+1,i}$ can be an iso. But since for each i , $|TX|_{\Psi_i}$ is a quotient of $|TX|_{\Phi}$, and the latter has only finitely many quotients by assumption, the sequence of counterexamples must be finite. $\square$

Remark 107. It can happen that even if we add a counterexample to Ψ as a result of a wrong conjecture, the number of states in S remains the same during the next closing table process, i.e., the algorithm recognises that the table is still closed even without adding states to S , that is why we need the assumption on TX and termination argument has to work on the number of equivalent classes of TX and not on the number of states.

In summary, we have the following.

Corollary 108. *If $|X|_\Phi$ has only finitely many quotients and T preserves finite sets (T restricts to finite sets), then the Theorem 106 terminates.*

Proof. The two conditions are those ones required by Lemma 104. Hence, $|TX|_\Phi$ has finitely many quotients. Then the algorithm terminates. $\square$

This corollary shows that Theorem 106 is an extension of Theorem 84.

4.4 Examples

In order to show how the algorithm works in practice, and to give an intuitive and concrete idea of the definitions introduced above, we present two examples: the learning of an NFA and of an alternating automaton, for two regular languages.

In both examples, given the specific language $\mathcal{L}$ we want to learn, we assume we have a DFA associated to it. As usual in our framework, the DFA is expressed as a pointed B -coalgebra (X, γ, x_0) where $B = 2 \times X^A$. Also, $X = A^*$, $x_0 = \lambda$ and $\gamma = \langle o, t \rangle$ with $o(w) = 1$ iff $w \in \mathcal{L}$ and $t(w)(a) = wa$ for $a \in A$. Moreover, throughout these examples the set of tests Φ is given as $\Phi = A^*$, and a set $\Psi \subseteq \Phi$ of tests is subformula-closed if it is a suffix-closed subset of A^* . Here, we assume that tests are words and the state space is composed of states represented as words themselves.

Example 109 (Non-deterministic finite automaton). We first consider the case of learning an NFA. The target language $\mathcal{L}$ is the one that contains all those words with an a at the third-last position, that is, $\mathcal{L} = \Sigma^* a \Sigma^2$ with $\Sigma = \{a, b\}$ [Bol+09].

As per Example 87, we fix the functor $T = P$, so that (finite) BT -coalgebras correspond to NFAs.

Given the initial configuration, i.e., a starting table $(S = \{\lambda\}, \Psi = 0)$, the first step of the algorithm is to check whether the table is P -closed, that is, whether there exists a $k: \Gamma(S) \rightarrow PS$ such that

$$\alpha_\Psi^b \circ Pth_\gamma^\Psi \circ Ps \circ k = th_\gamma^\Psi \circ \Gamma(s), \quad (4.5)$$

where $\alpha_\Psi^b: PQ\Psi \rightarrow Q\Psi$ is the mate of the forgetful logic for P , which intuitively takes the union of the elements of $PQ\Psi$ to obtain an element of $Q\Psi$. The subobject s can be thought of as the inclusion of S into X , so that Ps is the inclusion of subsets of S into subsets of X . Given a subformula-closed set Ψ , the theory map $th_\gamma^\Psi: X \rightarrow Q\Psi$ takes a state $x \in X$ to the set of all tests in Ψ that are true at x .

At the first run, the table is trivially P -closed: if we define k to map both possible successors of λ , i.e. $a, b \in \Gamma(S)$, to $\{\lambda\}$ itself, Equation (4.5) is satisfied. The resulting conjecture is an automaton with a single non-accepting state and a loop on it reading both a and b .



This conjecture is not correct, so we receive a counterexample in the form of a subformula-closed set of formulas Ψ' containing at least one formula that is accepted by our conjecture but not by the original automaton, or vice versa. In this case the distinguishing formula is aaa , so $\Psi' = \{\lambda, a, aa, aaa\}$.

Given $(S = \{\lambda\}, \Psi')$ as the new table, we check closedness again. This is not the case: $th_\gamma^{\Psi'}$ maps $a \mapsto \{a, aa\}$ and $b \mapsto \{aaa\}$, whereas the elements of PS , namely $\{\lambda\}$ and $\emptyset$, are mapped to $\{aaa\}$ and $\emptyset$ respectively. There is thus no value we can assign to $k(a)$ satisfying (4.5), so the table is not P -closed.

We therefore add a to S and check P -closedness again. The table is still not closed: the successor of a , i.e., aa , is mapped by $th_\gamma^{\Psi'}$ to $\{a, aa, aaa\}$, and again no k satisfying (4.5) exists. We add aa to S as well. Once more the table fails to be closed, since $ab \mapsto \{a, aaa\}$ and none of the subsets of $S = \{\lambda, a, aa\}$ give rise to the same set of formulas, as showed by Figure 4.1 with an exemplification of all possible subsets.

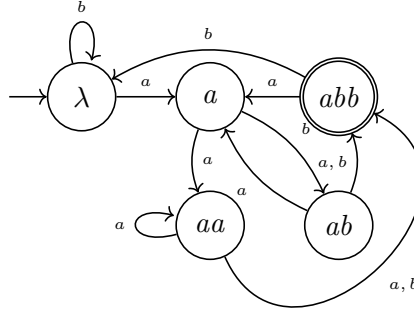
None of the eight subsets of S produces the target set $\{a, aaa\}$, so ab must be added to S as well. Checking again, we find that $abb \mapsto \{\lambda, aaa\}$ has no correspondence in PS either, so we add abb too. With $S = \{\lambda, a, aa, ab, abb\}$, the table is finally P -closed: every successor of an element of S can now be expressed as a combination of elements already in S . For example, $aba \mapsto \{\lambda, aa, aaa\}$ is obtained as $\{a, abb\} \mapsto \{\{aa, aaa\}, \{\lambda, aaa\}\} \mapsto \{\lambda, aa, aaa\}$. Note that the same table would *not*

$\{\lambda\}$	$\xrightarrow{Pi}$	$\{\lambda\}$	$\xrightarrow{Pth}$	$\{\{aaa\}\}$	$\xrightarrow{\cup}$	$\{aaa\}$
$\{a\}$	$\xrightarrow{Pi}$	$\{a\}$	$\xrightarrow{Pth}$	$\{\{aa, aaa\}\}$	$\xrightarrow{\cup}$	$\{aa, aaa\}$
$\{aa\}$	$\xrightarrow{Pi}$	$\{aa\}$	$\xrightarrow{Pth}$	$\{\{a, aa, aaa\}\}$	$\xrightarrow{\cup}$	$\{a, aa, aaa\}$
$\{\lambda, a\}$	$\xrightarrow{Pi}$	$\{\lambda, a\}$	$\xrightarrow{Pth}$	$\{\{\lambda\}, \{aa, aaa\}\}$	$\xrightarrow{\cup}$	$\{aa, aaa\}$
$\{\lambda, aa\}$	$\xrightarrow{Pi}$	$\{\lambda, aa\}$	$\xrightarrow{Pth}$	$\{\{aaa\}, \{a, aa, aaa\}\}$	$\xrightarrow{\cup}$	$\{a, aa, aaa\}$
$\{a, aa\}$	$\xrightarrow{Pi}$	$\{a, aa\}$	$\xrightarrow{Pth}$	$\{\{aa, aaa\}, \{a, aa, aaa\}\}$	$\xrightarrow{\cup}$	$\{a, aa, aaa\}$
$\{\lambda, a, aa\}$	$\xrightarrow{Pi}$	$\{\lambda, a, aa\}$	$\xrightarrow{Pth}$	$\{\{aaa\}, \{aa, aaa\}, \{a, aa, aaa\}\}$	$\xrightarrow{\cup}$	$\{a, aa, aaa\}$
$\emptyset$	$\xrightarrow{Pi}$	$\emptyset$	$\xrightarrow{Pth}$	$\{\emptyset\}$	$\xrightarrow{\cup}$	$\emptyset$
ab	$\xrightarrow{\Gamma i}$			ab	$\xrightarrow{th_{\gamma}^{\Psi}}$	$\{a, aaa\}$

Figure 4.1: Exemplification of the closing table process for NFA example.

be closed in the sense of Definition 65, since no single element of S is by itself a suitable representative for every such successor (e.g., aba).

The table is closed and the conjecture is the following automaton.



Recalling Definition 92 and Theorem 93, we can determine whether a state is accepting because this information is already encoded in the functor B : via $g: S \rightarrow B\Gamma(S)$, the algorithm reads off the acceptance condition directly, and in the case above λ is not a word of the language. The second component of the BT -conjecture, i.e., $k: B\Gamma(S) \rightarrow BTS$, is instead what determines which states of $\Gamma(S)$ are equivalent to states already present in the table.

The final NFA consists of five states, which is more succinct than the DFA for the same language, which has at least 8 states.

It is worth noting that a counterexample does not always force S to grow. Take $\Sigma = \{a\}$ and $\mathcal{L} = \Sigma^*$: the first run of the algorithm yields a conjecture consisting of a single accepting state with no transitions, since the successor of λ is mapped into $\emptyset$. The teacher then returns the counterexample $\Psi' = \{\lambda, a\}$. Checking P -closedness again shows that the table is still closed: this time a can be mapped to the state $\lambda \in S$ itself instead of to $\emptyset$, without any need to enlarge S . The resulting conjecture,



is correct, and the algorithm terminates without ever adding a state to S . This illustrates concretely the observation made after Theorem 106: termination is guaranteed by the growth of the number of equivalence classes of PS with respect to the tests, not by the growth of S itself.

Example 110 (Alternating automaton). Alternating finite automata [CKS81], AFA in short, generalise NFAs by adding the notion of universal branching. That is, besides the existential branching of an NFA, where a word is accepted if there exists at least one transition leading to an accepting state, an AFA also incorporates universal branching through the notion of *forks*, i.e., sets of states. Reading the first letter a of a word aw , the automaton moves to a collection of forks, and aw is accepted if there exists at least one fork (existential branching) all of whose states accept w (universal branching). Dually, aw is rejected if every fork contains at least one state that rejects w .

Coalgebraically, this type of automaton is captured by the functor BPP , where B is the usual DFA functor and both occurrences of P are the covariant powerset functor, so that $\gamma: X \rightarrow 2 \times P(P(X))^A$: a successor is a set of forks, and each fork is a set of states.

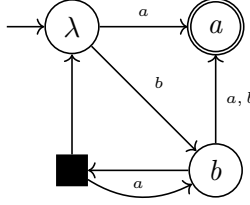
Our algorithm can also be used to learn this type of automaton. We use a simple but effective example already presented in [Hee+19], learning the small AFA representation of the DFA for the language $\mathcal{L} = \{a, ba, bb, baa\}$.

We first note that a BPP -coalgebra fits naturally into our framework: although

Neither a nor b can be mapped to λ , so we add a to the upper part of the table and, since it is still not closed, b as well (Checks 1–2). At the third check, two rows in the lower part still lack a matching combination: aa and ba . The row aa is matched by $\emptyset$, the empty join of upper rows, intuitively, a deadlock with no transitions. The row ba , instead, is matched by the meet of the rows labelled λ and b , joined with the row labelled a . That is, we set $ba \mapsto \{\{\lambda, b\}, \{a\}\}$, and indeed

$$\{\{\lambda, b\}, \{a\}\} \xrightarrow{PPi} \{\{\lambda, b\}, \{a\}\} \xrightarrow{PPth} \{\{\{a, baa\}, \{a, aa\}\}, \{\{\lambda\}\}\} \xrightarrow{\cup(\cap)} \{\lambda, a\},$$

which agrees with $th_\gamma(ba) = \{\lambda, a\}$, meeting the PP -closedness condition for ba . The table is now PP -closed, and the following conjecture is constructed.



Here the black square denotes the fork $\{\lambda, b\}$ (the meet of rows λ and b above). This AFA is the correct conjecture and the algorithm terminates. The DFA accepting the same language has five states [Hee+19].

4.5 Conclusion

In this chapter, we showed the central and natural role of duality in generalising automata learning by formally linking system states and logical tests via the dual adjunction. Our main contribution was twofold: first, we provided a learning framework and we extended it to handle, e.g., non-deterministic and alternating automata introducing in the framework the idea of forgetful logics. In this setting, we had been able to handle the branching part of coalgebras using BT functors. On the other hand, even if we defined all the notions in categorical terms, our work is based on the category of set and the main focus had been learning different types of coalgebras based on their functor B . Hence, we should consider this approach more as coalgebra learning than automata

learning.

Regarding automata learning, a closely related work is the one that has been carried out during the CALF project [Hee16; HSS17; HSS20; JS14]. The CALF framework focuses on generalizing the learning process across different underlying categories (e.g., **Set**, **Vect**), thereby studying a large variety of automata types defined on various domains. In contrast, our approach is more focused on the variation in the type of the transition structures while remaining based in the category of **Set**. These two different works suggest that the integration of our duality approach with the CALF framework could be an area of exploration for future work, as it could potentially combine our capacity for handling branching with their ability to generalise across domains. It is important to note that in [MS22] and [HSS20], the authors have already explored the possibility of learning automata with side-effects, specifically focusing on non-determinism within the context of residual nominal automata. One of their primary achievements was developing the theory of residual nominal automata and proving that this class possesses canonical representatives that can be constructed via a finite number of observations. This is a crucial property for active learning, where the goal is to identify the minimal automaton representing a specific language. Consequently, the framework presented in Section 4.3 is not intended to propose a new method for learning non-deterministic automata. Rather, it demonstrates how our algorithm can be extended in that direction. An interesting area for future research would be to investigate whether our construction can be extended within their framework, so to address also the open points we mentioned earlier in this chapter.

Furthermore, another approach to the automata learning field has been proposed in [US20] where a generic categorical framework for learning formal languages of various types (e.g., finite or infinite words, weighted and nominal languages) is described. This work is related to what we presented but it proceeds in a different direction. In fact, our focus is on transition structures and their generalisation via duality within the category of **Set**. On the other hand, they use monads and abstract automata to generalise learning over different types of algebraic structures, moving beyond the category of **Set**. The successful use of monads in their work suggests a future direction for us: incorporating

monads into our framework to provide a more accurate and standard way to model and handle branching components within the existing theory of coalgebras [JSS15].

Finally, moving from theory to application, another open problem is the concrete implementation of a learning algorithm parametric in the type of coalgebra B . The main challenge here is finding an efficient and concrete way to represent the tests we presented as logical formulas and executing queries against them (generating an explicit logical formula for a long word can be computationally expensive). However, works such as [Dei+19] and [Dor+17] offer a promising starting point to find an efficient way to define such an algorithm that is parametric in each B -coalgebra. It is important to note that within this abstract framework, the teacher could potentially return the entire set of formulas Φ as a counterexample. This is not the case for the framework presented in the previous chapters, where a counterexample consists of a single additional formula, or the traditional L^* algorithm, where a counterexample is a specific string that distinguishes previously equivalent states. While an *Oracle* returning the complete set Φ would allow the learner to close the table and produce a correct conjecture immediately, such a response is trivial in the learning process. Furthermore, in cases where the logic Φ is infinite, providing the entire set would be computationally impossible, highlighting the necessity of incremental counterexamples in practical applications.

Chapter 5

Learning via Defects

The starting point of this chapter is the work of Lee and Yannakakis [LY92] where we can find two algorithms to perform transition system minimisation. The first algorithm minimises a given transition system, while the second computes the minimal reachable part of a specific system. The underlying concept is closely related to the L^* algorithm discussed in the previous chapters, in fact they are both iterative refinement processes, but with a significant difference: the Teacher seems missing and it is unclear which aspect of the algorithm makes up for it. Because the process of handling counterexamples is refined and reduced, a concrete implementation of this approach may be easier compared to the traditional ones, even if the complexity of the algorithm itself won't be reduced. In line with our work, we would like to apply logic and coalgebras concepts to abstract the algorithm behaviour in order to better understand how it works, extend it to a variety of transition systems and to prove termination. Indeed, the aim is to extend the algorithm applicability to transition systems that can be represented as coalgebras with a finite quotient.

This chapter provides a foundational study of the Lee and Yannakakis algorithms, aiming to review the mechanics of joint minimality and reachability through a logical lens. Rather than a final characterisation, our logical representation is just the starting point for a deeper study of this way of learning transition systems. For example, some open questions are to determine whether the assumption we will maintain throughout this chapter, i.e., the closure under duals, can be relaxed or eliminated while still guar-

anteeing termination and to understand and establish the connection of this work with other coalgebraic partition refinement approaches [Rot+17; KK14; Adá+12; Dei+19].

Following the structure of the original work by Lee and Yannakakis, we first restate their minimisation algorithm within our logical framework, providing formal proofs of its correctness and termination (cf. Section 5.1). This minimisation algorithm serves as the foundation for incorporating reachability. In fact, in the next section, we investigate a version that computes both reachability and minimisation simultaneously, viewed through a logical lens. On the one hand, the minimisation approach aligns with standard partition refinement algorithms that determine behavioural equivalence for coalgebras by factoring out redundant information [KK14]. On the other hand, Lee and Yannakakis also describe an algorithm specifically designed for symbolic transition systems, a feature that our logical approach fully supports. So, we then move beyond simple minimisation to consider the more complex problem of computing the minimal reachable part of a system relative to a given initial state. We explore the extension of our algorithm to incorporate reachability, discussing the current limits of our approach while outlining the core idea for future development (cf. Section 5.2).

5.1 Minimisation

The objective of this section is to restate the minimisation algorithm proposed by Lee and Yannakakis within a logical framework. The algorithm aims to compute the reduced version of a transition system by identifying the partition induced by bisimulation equivalence [LY92]. In this context, the set of states is grouped into blocks, i.e., sets that contain only those states that are behaviourally equivalent. The set of blocks forms a partition. The partition is iteratively refined checking for stability, a condition that ensures that for any two blocks B and C in the partition, every state in B has a transition to some states in C . This way, the algorithm refines the partition until all the blocks contain only those states that are behaviourally equivalent. The resulting system is a minimal version of the target one.

In our setting, we rephrase these concepts using ideas from modal logic. We define the idea of logical partition where each block is represented by the modal formula that

represents it. Also, rather than speaking explicitly about stability, we introduce the notion of defect, i.e., a modal formula that allows us to distinguish states. Defects allow to refine the partition.

As in previous sections, we work with a functor $B : \mathbf{Set} \rightarrow \mathbf{Set}$ and a set of predicate liftings Λ for B . The aim of this section is to describe an algorithm for minimising a B -coalgebra (X, γ) . We assume that B preserves inclusions, i.e., $X \subseteq Y$ implies $BX \subseteq BY$.

We begin by recalling how given a modal similarity type Λ and its corresponding Λ -structure, we can define a logic (cf. Definition 15 at page 30). Based on this, we define a logical partition as follows.

Definition 111 (Logical partition). *A logical partition Π of a B -coalgebra (X, γ) is a (finite) set of formulas $\{\psi_i\}_{i \in I}$ such that for all $x \in X$ we have $x \models \psi_i$ for exactly one $i \in I$ and for all $i \in I$ there exists a $x \in X$ such that $x \models \psi_i$.*

In this context, with abuse of notation, given a logical partition $\{\psi_i\}_{i \in I}$, we will use ψ_i to denote both the formula ψ_i and the subset $U \subseteq X$ such that $U = \{x \in X \mid x \models \psi_i\}$, i.e., ψ_i represents both a formula and a block of (X, γ) , i.e., the block contains all the states in which ψ_i is true. Please note that in case we want to explicitly distinguish formulas from the blocks they represent, we will indicate the blocks as $\underline{\psi}$. Also, because the logical partition can be seen as both the formula and the block, we can define a function that takes the formula and returns an element of the block that makes that formula true. This function is called a *selection of points* and it is defined as follows.

Definition 112 (Selection of points). *Given a logical partition $\Pi = \{\psi_i\}_{i \in I}$, we call $s : \Pi \rightarrow X$ a selection of points if $s(\psi_i) \models \psi_i$.*

Please note that by definition of logical partition there exists at least one (but usually more than one) selection of points for Π .

As we mentioned before, a key distinction between the original work by Lee and Yannakakis and the framework presented here lies in the mechanism used for partition refinement. Rather than explicitly checking for stability as they do, we introduce the concept of defect. Informally, in our setting, a defect is a modal formula that acts

as a witness of behavioural inconsistency, revealing that our current partition is too coarse because it groups behaviourally distinct states together. Specifically, suppose we examine a block ψ_i in our partition. By looking at the one-step transitions of its states via a modal operator (below in the definition, it will be called $\langle\lambda\rangle$) targeting other blocks ψ_j of the partition, we might find that some states inside ψ_i can reach those target blocks while others cannot. A defect at ψ_i captures this exact disagreement using formulas formed from other blocks ψ_j . When we discover such a formula φ , it allows us to split the block ψ_i into two refined blocks, i.e., $\psi_i \wedge \varphi$ and $\psi_i \wedge \neg\varphi$. The partition is then more accurate and it can distinguish states previously considered equivalent.

We can define it formally as follows.

Definition 113 (Defect). *Given a logical partition $\Pi = \{\psi_i\}_{i \in I}$ of (X, γ) , a defect at $\psi_i \in \Pi$ is a modal formula $\varphi = \langle\lambda\rangle \bigvee_{j \in J} \psi_j$ for some $\lambda \in \Lambda$ and $J \subseteq I$ such that*

$$\{\psi_j\}_{j \in I, j \neq i} \cup \{\psi_i \wedge \varphi\} \cup \{\psi_i \wedge \neg\varphi\}$$

is a logical partition.

Please note that here we adopt the notation $\langle\lambda\rangle$ for our modal operators, which marks a difference from what we previously introduced in Definition 14 of Chapter 2 where we used a generic symbol $\heartsuit$. Indexing the operator with λ explicitly connects the syntactic diamond to its semantic counterpart, that is the specific natural transformation λ (the predicate lifting) defining its interpretation. This notation aims to clarify the interaction with predicate liftings, simplifying our upcoming proofs and lemmas, while also allowing the framework to naturally accommodate multiple distinct operators belonging to the similarity type Λ .

If, for example, we were dealing with a DFA, the transition type corresponds to reading a specific letter a of the alphabet, yielding the concrete modal operator $\langle a \rangle$. Our defects are then defined using these operators to check whether a state can transition via specific actions into designated target blocks, as illustrated in Example 16 on page 30.

For sake of simplicity, given a set of predicate lifting Λ and a set of formulas Ψ , we refer to the $\langle\lambda\rangle \bigvee_{j \in J} \psi_j$ formula as the $\Lambda\Psi$ - disjunction formula, if $\lambda \in \Lambda$ and $\psi \in \Psi$.

Intuitively, a defect is a certain $\Lambda\Psi$ - disjunction formula φ satisfied by the state x and not by another state z or vice versa in a specific block. We call x the witness state.

We now introduce the following assumption on the $\Lambda\Psi$ - disjunction formula φ . This assumption is required to ensure the termination of the algorithm. In fact, the algorithm must be capable of evaluating a potential split triggered by a defect φ . It has to verify that both $\psi_i \wedge \varphi$ and $\psi_i \wedge \neg\varphi$ are satisfiable. This prevents the algorithm from branching into empty, unreachable blocks, which would otherwise compromise the termination of it.

Assumption 114. *For each formula ψ in the model, we can decide whether it is satisfiable in the model.*

The aim of the algorithm is to obtain the minimal quotient that is equivalent to the target coalgebra. So, given a partition and a selection of points, we define the Π -quotient as follows.

Definition 115. *Given a logical partition $\Pi = \{\psi_i\}_{i \in I}$ of (X, γ) and a selection of points s , the Π -quotient relative to s , denoted by (Π, γ_s) is given as*

$$\Pi \xrightarrow{s} X \xrightarrow{\gamma} BX \xrightarrow{Bb} B\Pi$$

where $b : X \rightarrow \Pi$ is the map that maps each $x \in X$ to its block ψ_i in the partition, i.e., $x \models b(x)$.

Given the formal definitions above, we can now outline the minimisation procedure detailed in the Algorithm 4. Intuitively, it begins with the coarsest possible assumption: all the states in the system are behaviourally equivalent and we represent this initial assumption setting up a partition Π containing only the top formula $\top$, which is satisfied by every state in X , the set of states of our coalgebra representing the target system. We then set a variable, **change**, which tracks whether the partition is the finest one. In each iteration, the algorithm scans the current logical partition and checks for defects. As defined previously, a defect is a modal formula that identifies states that were assumed to be equivalent within the previous settings. When a defect is found, we remove the

previous formula from the partition and we add the new modal formula. This allows to refine the partition adding a new block to it. This refinement continues until no further defects can be found. At this point, the algorithm uses the map s to pick a representative state for each logical block and constructs the minimal quotient (Π, γ_s) . This resulting system is the most succinct representation of the original B -coalgebra.

Algorithm 4 Logical minimisation

Input: (X, γ)

Output: (Π, γ_s)

```

1:                                     ▷ Initialisation
2:  $\Pi \leftarrow \{\top\}$ 
3:  $\text{change} \leftarrow \top$ 
4:                                     ▷ Split
5: while  $\text{change} = \top$  do
6:    $\text{change} \leftarrow \perp$ 
7:   for each  $\psi_i \in \Pi$  do
8:      $S \leftarrow \text{CHECKFORDEFECT}(\psi_i, \Pi)$ 
9:     if  $S \neq \emptyset$  then
10:      let  $\varphi \in S$ 
11:      remove  $\psi_i$  from  $\Pi$ 
12:      add  $\psi_i \wedge \varphi$  and  $\psi_i \wedge \neg\varphi$  to  $\Pi$ 
13:       $\text{change} \leftarrow \top$ 
14:     end if
15:   end for
16: end while
17: Choose a selection of points  $s : \Pi \rightarrow X$  such that  $s(\psi_i) \models \psi_i$  for all  $\psi_i \in \Pi$ 
18: return  $(\Pi, \gamma_s)$ 

```

In particular, the function `CHECKFORDEFECT` below allows us to find a defect, if present. Intuitively, a defect occurs when a state satisfies a formula describing a specific transition, but its successors are not contained within the block induced by that formula or vice versa.

```

1: function CHECKFORDEFECT( $\psi_i, \Pi$ )
2:   for each  $\Lambda\Psi$ -disjunction formula  $\varphi = \langle \lambda \rangle \bigvee_{j \in J} \psi_j$  do
3:     if  $\exists x \in X$  s.t.  $x \models \psi_i \wedge \varphi$  and  $\gamma_s(\psi_i) \notin \lambda_\Pi(\{\psi_j\}_{j \in J})$  then
4:       return  $\{\varphi\}$ 
5:     else if  $\exists x \in X$  s.t.  $x \models \psi_i \wedge \neg\varphi$  and  $\gamma_s(\psi_i) \in \lambda_\Pi(\{\psi_j\}_{j \in J})$  then
6:       return  $\{\varphi\}$ 

```

```

7:      end if
8:  end for
9:  return  $\emptyset$ 
10: end function

```

After sketching the algorithm, we will show that it terminates and it is correct. To prove it, we need some lemmas.

The first one ensures that the output of the algorithm is the minimal quotient of the original system.

Lemma 116. *Given a B -coalgebra (X, γ) and a logical partition $\Pi = \{\psi_i\}_{i \in I}$, we have*

$$\forall x \in X, \forall J \subseteq I, \forall \lambda \in \Lambda, x \models \langle \lambda \rangle \bigvee_{j \in J} \psi_j \Leftrightarrow Bb(\gamma(x)) \in \lambda_\Pi(\{\psi_j\}_{j \in J}).$$

Proof.

$$\begin{aligned}
& x \models \langle \lambda \rangle \bigvee_{j \in J} \psi_j \\
\Leftrightarrow & \gamma(x) \in \lambda_X(\bigcup_{j \in J} \underline{\psi_j}) && \text{by predicate lifting} \\
\Leftrightarrow & \gamma(x) \in \lambda_X(2^b(\{\psi_j\}_{j \in J})) && \text{by } 2^b(\{\psi_j\}_{j \in J}) := \bigcup_{j \in J} \underline{\psi_j} \\
\Leftrightarrow & \gamma(x) \in 2^{Bb}(\lambda_\Pi(\{\psi_j\}_{j \in J})) && \text{by naturality} \\
\Leftrightarrow & Bb(\gamma(x)) \in \lambda_\Pi(\{\psi_j\}_{j \in J}) && \text{by definition of preimage function}
\end{aligned}$$

□

The following lemma ensures that given a logical partition, we are able to find a defect, i.e., the logical partition can be refined, only when the current partition over-approximates or under-approximates the target system. That is, there exists an $x \in X$ that behaves differently compared to the set of formulas that defines the partition Π .

Lemma 117. *Let (X, γ) be a B -coalgebra and $\Pi = \{\psi_i\}_{i \in I}$ be a logical partition and $s : \Pi \rightarrow X$ be a selection of points. Then Π has a defect $\{\psi_j\}_{j \in J}$ at ψ_i if and only if one of the following conditions is true:*

Chapter 5. Learning via Defects

1. $\exists x \in X, \exists J \subseteq I$ and $\exists \lambda \in \Lambda$ s.t.

$$x \models \langle \lambda \rangle \bigvee_{j \in J} \psi_j \text{ and } \gamma_s(b(x)) \notin \lambda_\Pi(\{\psi_j\}_{j \in J})$$

2. $\exists x \in X, \exists J \subseteq I$ and $\exists \lambda \in \Lambda$ s.t.

$$x \not\models \langle \lambda \rangle \bigvee_{j \in J} \psi_j \text{ and } \gamma_s(b(x)) \in \lambda_\Pi(\{\psi_j\}_{j \in J})$$

Proof. The claim follows easily from the observation that $\gamma_s(b(x)) = Bb(\gamma(s(b(x))))$ by definition of γ_s and that the latter is equivalent to

$$s(b(x)) \models \langle \lambda \rangle \left(\bigvee_{j \in J} \psi_j \right)$$

by the previous lemma. □

We now prove that if there are no defects, the map b we defined in Definition 115 is a coalgebra morphism.

Lemma 118. *If there are no defects, $b: X \rightarrow \Pi$ is a coalgebra morphism, i.e., $Bb \circ \gamma = \gamma_s \circ b$.*

Proof. In order to prove this lemma, by separatingness it suffices to prove that $\gamma_s(b(x)) \in \lambda_\Pi(\{\psi_j\}_{j \in J}) \leftrightarrow Bb \circ \gamma(x) \in \lambda_\Pi(\{\psi_j\}_{j \in J})$.

$$\begin{aligned} & \gamma_s(b(x)) \in \lambda_\Pi(\{\psi_j\}_{j \in J}) \\ \Leftrightarrow & x \models \langle \lambda \rangle \left(\bigvee_{j \in J} \psi_j \right) && \text{by definition of defect} \\ \Leftrightarrow & Bb(\gamma(x)) \in \lambda_\Pi(\{\psi_j\}_{j \in J}) && \text{by Lemma 117} \end{aligned}$$

□

Lemma 119. *If there are no defects, then (Π, γ_s) does not have a proper quotient.*

Proof. Follows from the fact that for each $\psi_i, \psi_j \in \Pi$ we have $(\Pi, \gamma_s), \psi_i \models \psi_j$ iff $i = j$. □

In order to prove that the algorithm terminates, we define an upper bound and a lower bound to the number of states the algorithm can generate. The upper bound is given by the number of states in the minimal system, whereas the lower bound is given by the number of formulas in the logical partition. By definition of Π it is clear that $|\Pi| \leq n$ where n is the maximum number of pairwise distinct non-bisimilar points in (X, γ) (because elements of different blocks are non-bisimilar as they are distinguishable by a modal formula). The size of Π increases whenever a defect is removed, so the algorithm part of removing defects will terminate in at most n steps. The algorithm will return a correct quotient (Lemma 118) which is *maximal*, i.e., no further quotienting is possible (Lemma 119).

5.2 Combining minimisation and reachability

The aim of what follows is extending the minimization algorithm to also embrace reachability. That is, given a specific state of a transition system computing only the part of the system that is made of those states that are reachable starting from that given initial state. There are mainly two ways for building up a reachable minimal graph. On the one hand, it is possible to compute first all the reachable states starting from the initial state and then minimise the transition system induced on them or, on the other hand, it is possible to minimise the transition system first and then compute the reachable part. However, combining these two approaches could be better in performance. In their work, Lee and Yannakakis introduce the definition of marked block, i.e., a block together with a state. These kinds of blocks are those reachable in the system, the associated state is a representative for that block. Also, they introduce the notion of semi-stability to compute the reachable minimal transition system [LY92].

In what follows, we would like to extend the minimisation algorithm to compute only the reachable part of the transition system in a way where the reachability part of the algorithm is combined with the minimality one. In order to do that, we need first to introduce an assumption to ensure that our algorithm can correctly terminate using the logical framework and the notion of defects we have defined. Intuitively, the new algorithm refines blocks by considering only those ones that are currently marked as

reachable. However, a situation may arise where two states ought to be distinguished by a transition into a block that is not yet known to be reachable, preventing us from constructing a valid defect directly. Ensuring that the set of predicate liftings is closed under duals resolves this issue by allowing us to equivalently characterise the behavioural discrepancy using only the reachable blocks we have already discovered.

Assumption 120. *The set Λ of predicate liftings is closed under duals, i.e., $\lambda \in \Lambda$ implies $\lambda^d \in \Lambda$ where $\lambda_X^d(U) = BX \setminus \lambda(X \setminus U)$.*

Also, we need to extend the previous definition of defect introducing the notion of reachability. Given a partition Π and a subset Ψ containing only the formulas of those blocks that are reachable, the role of a reachable defect is to allow the algorithm to refine the partition using exclusively known reachable information. More formally, we define it as follows.

Definition 121. *Let $\Pi = \{\psi_i\}_{i \in I}$ be a logical partition of (X, γ) and let $\Psi \subseteq \Pi$. We call a defect $\varphi = \langle \lambda \rangle \bigvee_{j \in J} \psi_j$ at ψ_i Ψ -reachable if for all $j \in J$, we have $\psi_j \in \Psi$. In this case, we refer to this defect as the $\Lambda\Psi_R$ -disjunction formula.*

We will now modify the *logical* minimisation algorithm presented in Algorithm 4 by accounting for an initial state: the algorithm that we will describe takes a pointed B -coalgebra (X, γ, x_0) as input and return the maximal quotient $(R_X, \hat{\gamma}, x_0)$ of the reachable part, assuming that that quotient is finite.

Please note that in the following discussion, $R \subseteq \Psi \times X$ denotes a set of pairs of the form $\langle \psi, x \rangle$ and each pair represents a marked block. We define R_Ψ as the set of reachable formulas and R_X as the set of reachable states. Formally, letting $p_0: \Psi \times X \rightarrow \Psi$ and $p_1: \Psi \times X \rightarrow X$ be the standard canonical projections, these sets are given by $R_\Psi = p_0(R)$ and $R_X = p_1(R)$, respectively. Throughout the rest of this section, we will leave these projections implicit for the sake of readability.

Intuitively, the core idea behind the algorithm is simple. After initialising the set of formulas and the set of pairs, the algorithm first identifies all reachable blocks starting from the initial state. Following this reachability search, it splits any blocks where a defect can be found. The set U of successors is computed according to the definition

of the base previously established in Section 23 on page 39. The way for identifying and resolving these inconsistencies is detailed in the `CHECKANDSPLITDEFECT` function defined below. Please note that the algorithm does not simply compute the entire reachable part of the system first and then perform all the possible and necessary splits. Rather, the two processes are closely combined. The algorithm constructs the reachable part until no more new states can be found. This is achieved by verifying if any successor of a marked state (a state in a pair) satisfies a formula in the partition that has not yet been identified as reachable. If so, the corresponding formula and state pair is added to the set of reachable pairs R . This mechanism is closely related to the process of closing the observation table in the L^* algorithm. Just as L^* ensures the table is closed by adding successors whose behaviour is not the same of the predecessor state, this algorithm ensures reachability by expanding the set R until no new logical behaviours can be reached from the current settings.

To compute the reachable part, the algorithm checks if there exists a successor state of a specific marked state that makes true a formula in the partition that is not considered reachable yet, we add this pair to the set of reachable pairs. As said, this process is closely related to the process of closing a table in L^* where the algorithm checks whether the successors of a state already in the table behave as this state, if no, the successors are added to the table.

The algorithm then triggers a split to resolve any defects. Because a split changes the partition, the algorithm must check reachability again. This loop continues until both reachability and split parts cannot be performed any further. Once neither search nor split loops can be performed, the algorithm terminates and returns the $(R_X, \hat{\gamma}, x_0)$ coalgebra.

The algorithm looks as follows.

Algorithm 5 Logical minimisation with reachability**Input:** (X, γ, x_0) , set of formulas Ψ , set of pairs R **Output:** $(R_X, \hat{\gamma}, x_0)$

```

1:                                     ▷ Initialization
2:  $\Psi \leftarrow \{\top\}, R \leftarrow \{\langle \top, x_0 \rangle\}$ 
3: change  $\leftarrow \top$ 
4:                                     ▷ Search
5: while change =  $\top$  do
6:   change  $\leftarrow \perp$ 
7:   while  $\exists \langle \psi, x \rangle \in R$  with unexplored transitions do
8:      $U \leftarrow \text{Base}(\gamma(x))$ 
9:     if  $\exists \phi \in \Psi$  s.t  $\phi$  does not occur in  $R_\Psi, \exists y \in U. y \models \phi$  then
10:       add  $\langle \phi, y \rangle$  to  $R$ 
11:       change  $\leftarrow \top$ 
12:     end if
13:   end while
14:                                     ▷ Split
15:   for each  $\langle \psi, x \rangle \in R$  do
16:     oldR  $\leftarrow R$ 
17:      $\text{CHECKANDSPLITDEFECT}(R, \Psi, \langle \psi, x \rangle)$ 
18:     if oldR  $\neq R$  then
19:       change  $\leftarrow \top$ 
20:     end if
21:   end for
22: end while
23: return  $(R_X, \hat{\gamma}, x_0)$ 

```

Intuitively, the following function checks for defects within a block. If a defect is found, the algorithm splits the block into two more precise formulas and updates the corresponding reachable pair in R . Finally, we set the change flag to true. This forces the algorithm to execute another reachability check, as splitting a block may have exposed new transitions or isolated others.

```

1: function CHECKANDSPLITDEFECT( $R, \Psi, \langle \psi, x \rangle$ )
2:   for each  $\Lambda\Psi$ -disjunction formula  $\varphi = \langle \lambda \rangle \bigvee_{j \in J} \psi_j$  do
3:     if  $x \models \varphi$  and  $\psi \wedge \neg\varphi$  is satisfiable in the model then
4:       replace  $\langle \psi, x \rangle$  with  $\langle \psi \wedge \varphi, x \rangle$  in  $R$ 
5:       add  $\psi \wedge \neg\varphi$  to  $\Psi$ 
6:     end if
7:     if  $x \not\models \varphi$  and  $\psi \wedge \varphi$  is satisfiable in the model then
8:       replace  $\langle \psi, x \rangle$  with  $\langle \psi \wedge \neg\varphi, x \rangle$  in  $R$ 
9:       add  $\psi \wedge \varphi$  to  $\Psi$ 
10:    end if
11:  end for
12: end function

```

As we did for the minimisation algorithm, we now introduce some lemmas that allow us to prove termination. The first lemma shows that if the search part of the algorithm terminates, then also the split part does.

Lemma 122. *If the search part of the algorithm terminates, then split part also eventually terminates.*

Proof. Splits are triggered only by R_Ψ -reachable defects (cf. Line 15 of the Algorithm 5). If the search part stops extending R , then there is a fixed finite number of R_Ψ -reachable defects, which gives an upper bound to the number of possible splits. $\square$

Intuitively, the following lemma says that the search loop terminates if and only if all the characteristic formulas that represent the reachable states are already present in the set R_Ψ . That is, the if condition of the Line 9 of the algorithm is false and therefore, we cannot explore the transition system any further.

Lemma 123. $\forall x \in R_X. Bb(Base(\gamma(x))) \subseteq R_\Psi \Leftrightarrow \text{search loop terminates.}$

Proof. Suppose that the algorithm does not terminate, but we have $\forall x \in R_X. Bb(Base(\gamma(x))) \subseteq R_\Psi$. Then we have, $\forall x \in R_X. \{b(x) | x \in Base(\gamma(x))\} \subseteq R_\Psi$. Because we define $R_X := \pi_1[R]$, then we have $\forall (\langle \psi, x \rangle) \in R, \{\psi | \psi \in \pi_0[R]\} \subseteq R_\Psi$. If this is the case, the if condition at Line 9 of the algorithm would be false for all the possible pair in R and the search loop should terminate, but this is a contradiction with the assumption. Let now suppose that the algorithm terminates, but we have

$\forall x \in R_X. Bb(Base(\gamma(x))) \not\subseteq R_\Psi$. This means that $\exists \phi \in Bb(Base(\gamma(x)))$ and $\phi \notin R_\Psi$. Hence, $\exists y \in Base(\gamma(x)). y \models \phi$ and $\phi \notin R_\Psi$. Indeed, the if condition of the algorithm at Line 9 should be true, but as we assumed the search loop should terminate, hence such ϕ should not exist. We have a contradiction also in this second case. $\square$

The following lemma allows us to define a coalgebra map $\hat{\gamma}: R_X \rightarrow BR_\Psi$ by restricting $Bb \circ \gamma$ to BR_Ψ .

Lemma 124. *If the search loop terminates then $Base(Bb \circ \gamma(x)) \subseteq R_\Psi$.*

Proof. Consider the following diagram.

$$\begin{array}{ccc} BX & \xrightarrow{Base} & PX \\ Bb \downarrow & & \downarrow P_b \\ B\Psi & \xrightarrow{Base} & P\Psi \end{array}$$

It commutes by naturality of $Base$. Hence, $Base(Bb\gamma(x)) = Pb(Base(\gamma(x)))$ by naturality, where $\gamma(x)$ is to be considered as an element of BX . By Lemma 123, we know that $Pb(Base(\gamma(x))) \subseteq P\Psi$, therefore $Base(Bb\gamma(x)) \subseteq P\Psi$. $\square$

We need the following new lemma about predicate liftings.

Lemma 125. *Let $t \in BX$, $X \subseteq Y$ and suppose $t \in \lambda(U)$ for some $U \subseteq Y$. Then $t \in \lambda(U \cap X)$.*

Proof. This is a direct consequence of λ being a natural transformation, which can be seen by looking at the naturality diagram for λ applied to the inclusion $i: X \rightarrow Y$.

$$\begin{array}{ccc} 2^Y & \xrightarrow{\lambda_Y} & 2^{BY} \\ 2^i \downarrow & & \downarrow 2^{Bi} \\ 2^X & \xrightarrow{\lambda_X} & 2^{BX} \end{array}$$

$\square$

Now, we need to prove that the algorithm is correct and it terminates. In order to prove correctness, we need the following lemma.

Lemma 126. *Assuming that both the search and the split loops terminate, every reachable state should be contained in a marked block.*

Proof. We prove this lemma by induction on the depth of a path, i.e., the minimal number of the steps in the system we need to follow to reach a state from the initial one.

Step = 0: The initial state x_0 is by construction the selected point representing the first block. Once a point marks a block, it will stay in the final output.

Step = $n + 1$: We assume by induction that the claim holds for those states reachable in n steps. So, given any reachable $x \in X$ and a pair $\langle \psi, y \rangle$ where the formula ψ defined the block $\underline{\psi}$ and x belongs to $\underline{\psi}$, we have that $\underline{\psi}$ is reachable. Moreover, given an $x' \in \text{Base}(\gamma(x))$, reachable by construction of Base, we need to prove that given a formula ψ' where $x' \models \psi'$, the block $\underline{\psi'}$ identified by ψ' is reachable. To do that, a case distinction is needed.

1. x is the selected point of the block, i.e., x is the second component of the pair $\langle \psi, x \rangle$. Hence, by construction of the search loop, $x' \models \psi'$ and $\underline{\psi'}$ is reachable. Indeed, according to the Line 9 of the algorithm if there exists a formula that is not yet considered as reachable, but exists a state in the successors of the selected point that makes that formula true, then we add the formula to the set of reachable elements. Hence, we know that also $\underline{\psi'}$ represented by the formula would be considered reachable.
2. x is not the selected point of the pair $\langle \psi, y \rangle$. Then, given $y \models \psi$ and y the selected point, we have two additional cases.
 - $\underline{\psi'} \cap \text{Base}(\gamma(y)) \neq \emptyset$ and we are done; otherwise
 - $\underline{\psi'} \cap \text{Base}(\gamma(y)) = \emptyset$, but $\underline{\psi'} \cap \text{Base}(\gamma(x)) \neq \emptyset$ (by assumption that $x' \in \underline{\psi'}$ and x' is a successor of x).

We use separatingness that we provided in Definition 19 at page 32 to prove that there exists a formula φ s.t. $y \models \varphi$ and $x \not\models \varphi$ or vice versa. Since we assumed that the algorithm does not split nor search anymore, this is a contradiction and we are done.

In fact, we have what follows.

$$\begin{aligned}
 & \underline{\psi'} \cap \text{Base}(\gamma(y)) = \emptyset \ \& \ \underline{\psi'} \cap \text{Base}(\gamma(x)) \neq \emptyset \\
 \Leftrightarrow & \ \underline{\psi'} \notin \text{Pb}(\text{Base}(\gamma(y))) \ \& \ \underline{\psi'} \in \text{Pb}(\text{Base}(\gamma(x))) \quad \text{applying } b \\
 \Leftrightarrow & \ \underline{\psi'} \notin \text{Base}(\text{Bb}(\gamma(y))) \ \& \ \underline{\psi'} \in \text{Base}(\text{Bb}(\gamma(x))) \quad \text{by naturality of base} \\
 \Leftrightarrow & \ \text{Base}(\text{Bb}(\gamma(y))) \neq \text{Base}(\text{Bb}(\gamma(x))) \quad \text{by def. on inclusivity} \\
 \Leftrightarrow & \ \text{Bb}(\gamma(y)) \neq \text{Bb}(\gamma(x)) \quad \text{applying def. of base}
 \end{aligned}$$

Now, because we have two elements that are not equal, by definition of separatingness, there exists w.l.o.g. a subset $\Gamma \subseteq \Psi$ and a λ s.t. $x \not\models \langle \lambda \rangle \vee \Gamma$ and $y \models \langle \lambda \rangle \vee \Gamma$. This is w.l.o.g. because if $x \models \langle \lambda \rangle \vee \Gamma$ and $y \not\models \langle \lambda \rangle \vee \Gamma$, then we can take the dual modal operator λ^d and $\Gamma' = \Psi \setminus \Gamma$ to obtain $x \not\models \langle \lambda^d \rangle \vee \Gamma'$ and $y \models \langle \lambda^d \rangle \vee \Gamma'$.

We have $y \models \langle \lambda \rangle \vee \Gamma$ is equivalent to $\text{Bb}(\gamma(y)) \in \lambda(\Gamma)$ and the latter is an element of BR_Ψ . By Lemma 125 this implies $\text{Bb}(\gamma(y)) \in \lambda(\Gamma \cap R_\Psi)$. At the same time we have $x \not\models \langle \lambda \rangle \vee \Gamma$ which implies $\text{Bb}(\gamma(x)) \notin \lambda(\Gamma)$ and thus $\text{Bb}(\gamma(x)) \notin \lambda(\Gamma \cap R_\Psi)$.

Together the above implies that $\vee \Gamma \cap R_\Psi$ is a R_Ψ -reachable defect at the block Ψ of y which is a contradiction to our assumption that both search and split terminated.

□

In other words, this lemma says that the final coalgebra is partitioned by the marked blocks or, also, that all the reachable states are covered.

5.3 Conclusion

As we have seen, this chapter provides the necessary components to reformulate the algorithms presented by Lee and Yannakakis in logical terms, specifically for computing transition systems that can be represented as coalgebras with a finite quotient.

We also mentioned that there is a relationship between the L^* algorithm and the

simultaneous computation of the reachable and minimal versions of a transition system. We have highlighted parallels between these two approaches. For instance, the process of closing a table in L^* where the algorithm ensures that all transitions lead to states already represented in the model, can be viewed as the reachability search described in the present algorithm. Furthermore, the partition refinement process uses modal formulas to split blocks and test behavioural equivalence in a manner really close to our versions of the L^* algorithm, where modal formulas serve as tests. It would be interesting to further explore these connections and unify our coalgebraic framework to fully encompass this algorithm as well. Additionally, as noted at the outset, the minimisation algorithm proposed by Lee and Yannakakis aligns with established coalgebraic partition refinement processes [Rot+17; KK14; Adá+12; Dei+19]. A compelling direction for future work would be to determine whether these refinement techniques could be used to translate not only the minimisation algorithm into a coalgebraic setting but also the version that incorporates reachability. This would potentially allow the algorithm to be extended to cover the learning of transition systems described on categories other than $\mathbf{Set}$.

Finally, a significant open question concerns our assumption regarding closure under duals (or the presence of negation) in the modal logic. Currently, our proof of termination for the minimisation algorithm relies on the logic being sufficiently expressive to distinguish any two non-bisimilar states. It remains to be investigated whether this assumption is a mechanical necessity or if termination can be guaranteed without this assumption.

Chapter 6

Conclusion

This thesis has investigated the field of automata learning through the lens of coalgebra and coalgebraic modal logics. We approached the classical Angluin L^* algorithm and we studied it in a more abstract way. We started with the observation that the concrete process of learning a regular language, represented as a minimal deterministic finite automaton, is closely related to the notion of filtration in modal logic. By viewing the tests of the L^* algorithm as logical formulas used to collapse a model into a finite number of equivalence classes, we developed a version of the Angluin algorithm for coalgebras using Boolean-valued modal formulas as tests. This approach allowed us to adapt the L^* framework into a new algorithm, L^{co} , which is capable of learning regular B -coalgebras for an arbitrary finitary set functor B . The key intuition was to shift from using words as tests to modal formulas, providing a logical interpretation of the process that allows to identify states that are behaviourally equivalent. In this context, we rephrased some classical definitions from the L^* framework in a coalgebraic and modal logic setting, replacing the notion of suffix-closedness with subformula closedness and the concept of a successor state with the idea of the base. These notions allowed us to define a logical observation table with its own definitions for closedness and consistency, which we termed sharpness. With these ingredients, we defined an algorithm that learns not only DFAs but also other transition systems, such as Mealy machines (whose formulas are Boolean-valued formulas) and Discrete Markov chains. Finally, we provided the proofs for termination and correctness together with a complexity analysis. While this

initial version provided a successful abstraction, it remained somewhat ad-hoc due to its reliance on Boolean-valued formulas and definitions that, while abstract, were not fully expressed in categorical terms. To address this, we worked within a framework based on the categorical concept of duality.

This shift to a dual adjunction framework allowed us to move beyond the category of sets to a more general class of categories. We redefined core concepts such as the notion of the base and subformula closedness. We began by introducing the notion of a subobject, which is the ground to introduce other main concepts such as the conjecture or presenting the definitions of subformula closedness and the base in categorical terms. We also restated the definition of closedness categorically. While intuitively a table is closed when all successors of a state are already represented within the elements of the conjecture, we stated that a table is closed when there exists a morphism such that all successor states behave identically under a specific set of formulas. That is, we moved from a definition focused on the specific elements of a set to a broader one focused on the maps. Following this, we proved the correctness and termination of this algorithm in a proper categorical manner. Moreover, we extended these results by adapting the definitions, lemmas, and proofs to learn not only B -coalgebras but also BT -coalgebras. This extension enables the learning of more complex transition structures, such as non-deterministic finite automata or alternating automata, allowing to learn more succinct versions of transition systems. The key observation was the identification of a specific class of forgetful logics, which maintain the same set of formulas to describe both B and BT -coalgebras by abstracting away from the branching represented by the functor T , while the semantics ensure the correct behaviour is interpreted.

Furthermore, we investigated the work of Lee and Yannakakis, who proposed an algorithm for the simultaneous computation of reachability and minimisation in transition systems. This approach seems closely related to the L^* algorithm, in fact they are both iterative refinement processes but with a key difference: in this context the teacher is missing and it is unclear which aspect of the algorithm makes up for it. This approach is particularly compelling for developing concrete implementations, as the removal of the requirement for an active Oracle can simplify the algorithmic architecture.

In line with our research objectives, we have applied logical and coalgebraic concepts to formalise the behaviour of this algorithm, aiming to extend its applicability to a broader class of transition systems representable as coalgebras with a finite quotient. Our logical representation serves as a foundational framework for deeper investigation, providing a lens through which the structural properties of these teacherless learning methods can be further explored.

Future work

This thesis aimed to approach the learning framework through the lens of coalgebra and modal logic. However, studies in this area are far from being exhausted. There remain several research topics open to future investigation, ranging from categorical generalisations to the concrete implementation of a coalgebraic learning algorithm. In what follows, we try to sum up the main central topics.

Generalisations

In this work, we show how duality plays a central role in generalising automata learning. We proposed an algorithm parametric in the type of category, but we remain mainly focused on the category of sets, with a primary focus on learning different types of coalgebras based on their functor B . This approach can be considered more as *coalgebra learning* than automata learning. A closely related line of research is the CALF project which focuses on generalising the learning process across different underlying categories (such as vector spaces or nominal sets) to study a variety of automata types across diverse domains. In contrast, as said, our approach focuses on the variation of transition structures within the category of sets. Integrating our approach with the CALF framework represents a promising area for future work, as it could combine our capacity for handling branching with their ability to learn automata across different domains.

Also, we provided a learning framework able to handle non-deterministic and alternating automata by introducing the concept of forgetful logics. This setting allowed

us to successfully manage the branching part of coalgebras using *BT*-functors. Extensive work [JSS15] has been conducted to study the correlation between branching and monads moving beyond the category of sets. The success of monads in those frameworks suggests a future direction for our research: incorporating monads into our dual adjunction framework to provide a more standard way to model and handle branching components within existing coalgebraic theory.

Moreover, as already reported in [MS22] and [HSS20], the authors have already explored the possibility of learning automata with side-effects, specifically focusing on non-determinism within the context of residual nominal automata developing the theory of residual nominal automata and proving that this class possesses canonical representatives that can be constructed via a finite number of observations. This is a crucial property for active learning, where the goal is to identify the minimal automaton representing a specific language. An interesting area for future research would be to investigate whether our construction can be extended and empowered within their framework.

Many-Valued Logics and Approximate Learning

As established in the previous chapters, modal logic is a framework for describing transition systems and distinguishing non-equivalent states. However, for some systems such as probabilistic or weighted automata, behavioural distance provides a more nuanced framework to distinguish states compared with the Boolean-valued logics [Fah+23].

It would be of considerable interest to investigate many-valued logics [KP21] and fuzzy logics to determine whether our categorical learning algorithm can be extended to systems where observations are real numbers or where transitions represent degrees of applicability. This is particularly relevant for the learning of weighted automata or fuzzy transition systems, where behaviours are represented by degrees of truth rather than true or false.

Building on this, future research should address the challenge of approximate learning, where the objective is to identify a sufficiently close model rather than an exact representation. This shift would require a deep review of our framework, specifically

regarding the construction of the observation table. Also, the properties of closedness and consistency must be redefined. Such an extension would bridge the gap between exact learning and the approximate nature of real-world quantitative observations.

Lee and Yannakakis Framework

In the work of Lee and Yannakakis [LY92], two algorithms are proposed for transition system minimisation: the first minimises a given system, while the second computes its minimal reachable part. The underlying concept is closely related to the L^* algorithm, with the significant distinction that here the teacher appears to be absent. Nevertheless, we observe a close relationship with L^* , where the process of closing a table and the use of tests to distinguish states can be mapped to the search and split phases of the Lee and Yannakakis algorithm, respectively. It would be of great interest to investigate this connection further to deeply understand the underlying learning mechanism. Such an understanding would provide an opportunity to extend our coalgebraic framework to fully encompass this algorithm as well. Also, while the minimisation component appears relatively straightforward topic, largely due to its proximity to established areas in coalgebraic partition refinement [KK14], the reachability aspect presents a more significant challenge.

Moreover, a substantial open question remains regarding our assumption of closure under duals in the modal logic. At present, our proof of termination for the algorithm relies on the logic being sufficiently expressive to distinguish any two states that do not behave the same. It remains to be investigated whether this assumption is a mechanical necessity or if termination can be guaranteed in its absence.

Finally, a concrete implementation of this methodology may prove more feasible than traditional active learning methods, as the reliance on a teacher is reduced or removed entirely, potentially simplifying the algorithmic concrete implementation.

Concrete implementation

From a practical perspective, implementing a learning algorithm that is truly parametric in the functor B presents significant challenges. Future work must find efficient ways

to represent modal formulas and execute queries against them to avoid computational inconsistencies [Dei+19] and [Dor+17] . Additionally, the role of the teacher requires further attention, in an abstract setting, an oracle could potentially return an infinite set of formulas as a counterexample, or trivially the full set of formulas that allow to characterise the target model, but developing a concrete oracle that provides minimal and incremental counterexamples is crucial to have an algorithm that is applicable to real-world scenarios. Also, the algorithm proposed by Lee and Yannakakis could be of interest to investigate a possible way of defining a concrete implementation of an active learning algorithm in a more simple way.

Bibliography

- [Aar+14] Fides Aarts et al. “Improving active Mealy machine learning for protocol conformance testing”. In: *Machine Learning* 96.1-2 (2014), pp. 189–224.
- [ABF18] Dana Angluin, Udi Boker, and Dana Fisman. “Families of DFAs as Acceptors of ω -Regular Languages”. In: *Logical Methods in Computer Science* (2018).
- [Adá+12] Jiří Adámek et al. “A Coalgebraic Perspective on Minimization and Determinization”. In: *FoSSaCS 2012*. Vol. 7213. Springer, 2012, pp. 58–73.
- [Adá+13] Jiří Adámek et al. “Well-Pointed Coalgebras”. In: *Logical Methods in Computer Science* Volume 9, Issue 3 (2013).
- [AEF15] Dana Angluin, Sarah Eisenstat, and Dana Fisman. “Learning Regular Languages via Alternating Automata”. In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*. IJCAI Organization, 2015, pp. 3308–3314.
- [AF16] Dana Angluin and Dana Fisman. “Learning regular omega languages”. In: *Theoretical Computer Science* 650 (2016). Algorithmic Learning Theory, pp. 57–72.
- [AF18] Dana Angluin and Dana Fisman. “Regular omega-Languages with an Informative Right Congruence”. In: *Electronic Proceedings in Theoretical Computer Science* 277 (2018), pp. 265–279.
- [Aho+06] Alfred V. Aho et al. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley, 2006.

Bibliography

- [AHS90] Jiří Adámek, Horst Herrlich, and George Strecker. *Abstract and Concrete Categories*. Wiley-Interscience, 1990.
- [ALM07] Jiří Adámek, Dominik Lücke, and Stefan Milius. “Recursive coalgebras of finitary functors”. In: *ITA* 41.4 (2007), pp. 447–462.
- [AMP22] Bernhard K. Aichernig, Edi Muškardin, and Andrea Pferscher. “Active vs. Passive: A Comparison of Automata Learning Paradigms for Network Protocols”. In: *Electronic Proceedings in Theoretical Computer Science* 371 (2022), pp. 1–19.
- [Ang81] Dana Angluin. “A note on the number of queries needed to identify regular languages”. In: *Information and Control* 51.1 (1981), pp. 76–87.
- [Ang87] Dana Angluin. “Learning Regular Sets from Queries and Counterexamples”. In: *Information and Computation* 75.2 (1987), pp. 87–106.
- [AR94] Jiří Adámek and Jiří Rosický. *Locally Presentable and Accessible Categories*. Cambridge Tracts in Mathematics. Cambridge University Press, 1994.
- [AS83] Dana Angluin and Carl H. Smith. “Inductive Inference: Theory and Methods”. In: *ACM Comput. Surv.* 15 (1983), pp. 237–269.
- [AT90] Jiří Adámek and Věra Trnková. *Automata and Algebras in Categories*. 1st. Kluwer Academic Publishers, 1990.
- [AU72] Alfred V. Aho and Jeffrey D. Ullman. *The theory of parsing, translation, and compiling*. USA: Prentice-Hall, Inc., 1972.
- [Awo10] Steve Awodey. *Category Theory*. 2nd. Oxford University Press, Inc., 2010.
- [Bal+97] José L. Balcázar et al. “Algorithms for Learning Finite Automata from Queries: A Unified View”. In: *Advances in Algorithms, Languages, and Complexity*. Ed. by Ding-Zhu Du and Ker-I Ko. Boston, MA: Springer US, 1997, pp. 53–72.

Bibliography

- [BCG13] Borja Balle, Jorge Castro, and Ricard Gavaldà. “Learning probabilistic automata: A study in state distinguishability”. In: *Theoretical Computer Science* 473 (2013), pp. 46–60.
- [Ber+05] Therese Berg et al. “Insights to Angluin’s Learning”. In: *Electronic Notes in Theoretical Computer Science* 118 (2005). Proceedings of the International Workshop on Software Verification and Validation (SVV 2003), pp. 3–18.
- [Ber+17] Sebastian Berndt et al. “Learning Residual Alternating Automata”. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*. AAAI Press, 2017, pp. 1749–1755.
- [BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. 2008.
- [BK18] Simone Barlocco and Clemens Kupke. “Angluin Learning via Logic”. In: *Logical Foundations of Computer Science*. Springer International Publishing, 2018, pp. 72–90.
- [BKR19] Simone Barlocco, Clemens Kupke, and Jurriaan Rot. “Coalgebra Learning via Duality”. In: *Foundations of Software Science and Computation Structures*. Springer, 2019, pp. 55–72.
- [Blo12] Alwin Blok. “Interaction, observation and denotation”. MA thesis. ILLC Amsterdam, 2012.
- [Bol+09] Benedikt Bollig et al. “Angluin-Style Learning of NFA.” In: *IJCAI*. Morgan Kaufmann Publishers Inc., 2009, pp. 1004–1009.
- [Bor09] Michele Boreale. “Weighted Bisimulation in Linear Algebraic Form”. In: *CONCUR 2009 – Concurrency Theory: 20th International Conference, Bologna, Italy, September 1-4, 2009. Proceedings*. Ed. by Mario Bravetti and Gianluigi Zavattaro. Vol. 5710. Lecture Notes in Computer Science. Springer, 2009, pp. 163–177.
- [Bor94] Francis Borceux. *Handbook of Categorical Algebra*. Vol. 1. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1994.

Bibliography

- [BRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Cambridge University Press, 2001.
- [Brz64] Janusz A. Brzozowski. “Derivatives of Regular Expressions”. In: *J. ACM* 11.4 (1964), pp. 481–494. ISSN: 0004-5411.
- [BS86] Gerard Berry and Ravi Sethi. “From regular expressions to deterministic automata”. In: *Theoretical Computer Science* 48 (1986), pp. 117–126.
- [Büc62] J. Richard Büchi. “On a decision method in restricted second order arithmetic”. In: *Proceedings of the 1960 International Congress on Logic, Methodology and Philosophy of Science*. Stanford, Calif.: Stanford University Press, 1962, pp. 1–11.
- [BV96] Francesco Bergadano and Stefano Varricchio. “Learning Behaviors of Automata from Multiplicity and Equivalence Queries”. In: *SIAM Journal on Computing* 25.2 (1996), pp. 418–432.
- [BW95] Michael Barr and Charles Wells. *Category Theory for Computing Science*. 2nd. Prentice Hall International (UK) Ltd., 1995.
- [CAL94] David Cohn, Les Atlas, and Richard Ladner. “Improving generalization with active learning”. In: *Machine Learning* 15.2 (1994), pp. 201–221.
- [CF58] Haskell Brooks Curry and Robert M. Feys. *Combinatory Logic Vol. 1*. Amsterdam, Netherlands, 1958.
- [CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. London, Cambridge: MIT Press, 1999.
- [Cha+14] Georg Chalupar et al. “Automated reverse engineering using LEGO®”. In: USA: USENIX Association, 2014.
- [Cha+15] Martin Chapman et al. “Learning the Language of Error”. In: *Automated Technology for Verification and Analysis (ATVA 2015)*. Vol. 9364. Lecture Notes in Computer Science. Springer, 2015, pp. 114–130.
- [Cîr+11] Corina Cîrstea et al. “Modal Logics are Coalgebraic”. In: *The Computer Journal* 54.1 (2011), pp. 31–41.

Bibliography

- [CKS81] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. “Alternation”. In: *J. ACM* 28.1 (1981), pp. 114–133.
- [Con71] John Horton Conway. *Regular algebra and finite machines*. Chapman and Hall mathematics series. London: Chapman and Hall, 1971.
- [CP04] Corina Cîrstea and Dirk Pattinson. “Modular Construction of Modal Logics”. In: *CONCUR 2004 - Concurrency Theory*. Ed. by Philippa Gardner and Nobuko Yoshida. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 258–275.
- [CPS21] Thomas Colcombet, Daniela Petrişan, and Riccardo Stabile. “Learning Automata and Transducers: A Categorical Approach”. In: *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*. Ed. by Christel Baier and Jean Goubault-Larrecq. Vol. 183. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 15:1–15:17.
- [Dei+19] Hans-Peter Deifel et al. *Generic Partition Refinement and Weighted Tree Automata*. 2019.
- [Dor+17] Ulrich Dorsch et al. *Efficient Coalgebraic Partition Refinement*. 2017.
- [Fah+23] Uli Fahrenberg et al. “Quantitative Hennessy-Milner Theorems via Notions of Density”. In: *31st EACSL Annual Conference on Computer Science Logic (CSL 2023)*. Ed. by Bartek Klin and Elaine Pimentel. Vol. 252. Leibniz International Proceedings in Informatics (LIPIcs). 2023, 19:1–19:18.
- [FJV16] Paul Fiterău-Broştean, Ramon Janssen, and Frits Vaandrager. “Combining Model Learning and Model Checking to Analyze TCP Implementations”. In: *Computer Aided Verification (CAV 2016)*. Vol. 9780. Lecture Notes in Computer Science. Springer, 2016, pp. 454–471.
- [Gol78] Mark E. Gold. “Complexity of automaton identification from given data”. In: *Information and Control* 37.3 (1978), pp. 302–320.

Bibliography

- [GP20] Alexandre Goy and Daniela Petrisan. “Combining probabilistic and non-deterministic choice via weak distributive laws”. In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science* (2020).
- [Hee+19] Gerco van Heerdt et al. “A (co)algebraic theory of succinct automata”. In: *Journal of Logical and Algebraic Methods in Programming* 105 (2019), pp. 112–125.
- [Hee16] Gerco van Heerdt. “An Abstract Automata Learning Framework”. MA thesis. Radboud Universiteit Nijmegen, 2016.
- [Hee20] Gerco van Heerdt. “Automata Learning: A Categorical Perspective”. PhD thesis. University College London, 2020.
- [HMU00] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computability*. 2nd. Addison-Wesley, 2000.
- [HR10] H.H. Hansen and J.J.M.M. Rutten. “Symbolic synthesis of Mealy machines from arithmetic bitstream functions”. In: *Scientific Annals of Computer Science* 20 (2010), pp. 97–130.
- [HSS17] Gerco van Heerdt, Matteo Sammartino, and Alexandra Silva. “CALF: Categorical Automata Learning Framework”. In: *26th EACSL Annual Conference on Computer Science Logic, CSL 2017*. 2017, 29:1–29:24.
- [HSS20] Gerco van Heerdt, Matteo Sammartino, and Alexandra Silva. “Learning Automata with Side-Effects”. In: *Coalgebraic Methods in Computer Science (CMCS 2020)*. Vol. 12290. Lecture Notes in Computer Science. Springer, 2020, pp. 68–89.
- [Jac00] Bart Jacobs. “Towards a duality result in the modal logic for coalgebras”. In: *Coalgebraic Methods in Computer Science (CMCS 2000)*. Vol. 33. Electronic Notes in Theoretical Computer Science. Elsevier, 2000, pp. 160–195.
- [Jac16] Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*. Cambridge University Press, 2016.

Bibliography

- [JR97] Bart Jacobs and Jan J. M. M. Rutten. “A tutorial on (co)algebras and (co)induction”. In: 1997.
- [JS14] Bart Jacobs and Alexandra Silva. “Automata Learning: A Categorical Perspective”. In: *Horizons of the Mind. A Tribute to Prakash Panangaden - Essays Dedicated to Prakash Panangaden on the Occasion of His 60th Birthday*. 2014, pp. 384–406.
- [JSS15] Bart Jacobs, Alexandra Silva, and Ana Sokolova. “Trace semantics via determinization”. In: *J. Comput. Syst. Sci.* 81.5 (2015), pp. 859–879.
- [KK14] Barbara König and Sebastian Küpper. “Generic Partition Refinement Algorithms for Coalgebras and an Instantiation to Weighted Automata”. In: *Theoretical Computer Science*. Ed. by Josep Diaz, Ivan Lanese, and Davide Sangiorgi. 2014, pp. 311–325.
- [KKP04] Clemens Kupke, Alexander Kurz, and Dirk Pattinson. “Algebraic Semantics for Coalgebraic Logics”. In: *Electr. Notes Theor. Comput. Sci.* 106 (2004), pp. 219–241.
- [KL09] Clemens Kupke and Raul Andres Leal. “Characterising Behavioural Equivalence: Three Sides of One Coin”. In: *Algebra and Coalgebra in Computer Science*. Ed. by Alexander Kurz, Marina Lenisa, and Andrzej Tarlecki. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 97–112.
- [Kli07] Bartek Klin. “Coalgebraic Modal Logic Beyond Sets”. In: *Electr. Notes Theor. Comput. Sci.* 173 (2007), pp. 177–201.
- [Koz97] Dexter C. Kozen. *Introduction to Automata Theory, Languages and Computability*. Springer-Verlag, 1997.
- [KP11] Clemens Kupke and Dirk Pattinson. “Coalgebraic semantics of modal logics: An overview”. In: *Theoretical Computer Science* 412.38 (2011), pp. 5070–5094.

Bibliography

- [KP21] Alexander Kurz and Wolfgang Poiger. “Many-Valued Coalgebraic Logic: From Boolean Algebras to Primal Varieties”. In: *9th Conference on Algebra and Coalgebra in Computer Science (CALCO 2021)*. Ed. by Fabio Gadducci and Alexandra Silva. Vol. 211. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 20:1–20:18.
- [KR16] Bartek Klin and Jurriaan Rot. “Coalgebraic trace semantics via forgetful logics”. In: *Logical Methods in Computer Science* 12.4 (2016).
- [KS18] Bartek Klin and Julian Salamanca. “Iterated Covariant Powerset is not a Monad”. In: *Electronic Notes in Theoretical Computer Science* 341 (2018), pp. 261–276.
- [Kur01] Alexander Kurz. “Specifying coalgebras with modal logic”. In: *Theoretical Computer Science* 260.1-2 (2001), pp. 119–138.
- [Kur06] Alexander Kurz. “Coalgebras and their logics”. In: *SIGACT News* 37.2 (2006).
- [KV94] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. Cambridge, MA, USA: MIT Press, 1994.
- [Lei14] Tom Leinster. *Basic Category Theory*. Cambridge, 2014.
- [LST23] Yong Li, Sven Schewe, and Qiyi Tang. “A Novel Family of Finite Automata for Recognizing and Learning ω -Regular Languages”. In: *Automated Technology for Verification and Analysis (ATVA)*. Vol. 14214. Lecture Notes in Computer Science. Springer, 2023, pp. 37–53.
- [LY92] David Lee and Mihalis Yannakakis. “Online minimization of transition systems (extended abstract)”. In: *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*. 1992, pp. 264–274.
- [Mao+11] Hua Mao et al. “Learning Probabilistic Automata for Model Checking”. In: *2011 Eighth International Conference on Quantitative Evaluation of Systems*. 2011, pp. 111–120.

Bibliography

- [Moo56] Edward F. Moore. “Gedanken-experiments on Sequential Machines”. In: *Automata Studies*. Vol. 34. Annals of Mathematics Studies. Princeton University Press, 1956, pp. 129–153.
- [Mos99] Lawrence S. Moss. “Coalgebraic logic”. In: *Annals of Pure and Applied Logic* 96.1 (1999), pp. 277–317.
- [MP95] Oded Maler and Amir Pnueli. “On the Learnability of Infinitary Regular Sets”. In: *Information and Computation* 118.2 (1995), pp. 316–326.
- [MS22] Joshua Moerman and Matteo Sammartino. “Residuality and Learning for Nondeterministic Nominal Automata”. In: *Log. Methods Comput. Sci.* 18.1 (2022).
- [Osi74] Gerhard Osius. “Categorical set theory: A characterization of the category of sets”. In: *Journal of Pure and Applied Algebra* 4.1 (1974), pp. 79–119.
- [Pat01] Dirk Pattinson. “Semantical Principles in the Modal Logic of Coalgebras”. In: *STACS 2001: 18th Annual Symposium on Theoretical Aspects of Computer Science*. Vol. 2010. Lecture Notes in Computer Science. Springer, 2001, pp. 514–526.
- [Pat03] Dirk Pattinson. “Coalgebraic modal logic: soundness, completeness and decidability of local consequence”. In: *Theoretical Computer Science* 309.1 (2003), pp. 177–193.
- [Pat04] Dirk Pattinson. “Expressive Logics for Coalgebras via Terminal Sequence Induction”. In: *Notre Dame Journal of Formal Logic* 45.1 (2004), pp. 19–33.
- [PMW06] Dusko Pavlovic, Michael W. Mislove, and James Worrell. “Testing Semantics: Connecting Processes and Process Logics”. In: *Algebraic Methodology and Software Technology, 11th International Conference, AMAST 2006, Proceedings*. 2006, pp. 308–322.

Bibliography

- [PO98] Jorge M. Peña and Arlindo L. Oliveira. “A New Algorithm For The Reduction Of Incompletely Specified Finite State Machines”. In: *Proceedings of the 1998 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. ACM / IEEE Computer Society, 1998, pp. 482–489.
- [Röß00] Martin Rößiger. “Coalgebras and modal logic”. In: *Coalgebraic Methods in Computer Science (CMCS 2000)*. Vol. 33. Electronic Notes in Theoretical Computer Science. Elsevier, 2000, pp. 299–320.
- [Rot+17] Jurriaan Rot et al. “Enhanced coalgebraic bisimulation”. In: *Mathematical Structures in Computer Science* 27.7 (2017), pp. 1236–1264.
- [RP15] Joeri de Ruiter and Erik Poll. “Protocol State Fuzzing of TLS Implementations”. In: *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, 2015, pp. 193–206.
- [RS93] Ronald L. Rivest and Robert E. Schapire. “Inference of Finite Automata Using Homing Sequences”. In: *Information and Computation* 103.2 (1993), pp. 299–347.
- [RSB05] Harald Raffelt, Bernhard Steffen, and Therese Berg. “LearnLib: a library for automata learning and experimentation”. In: *Proceedings of the 10th International Workshop on Formal Methods for Industrial Critical Systems*. New York, NY, USA: Association for Computing Machinery, 2005, pp. 62–71.
- [Rut00] Jan J.M.M Rutten. “Universal coalgebra: a theory of systems”. In: *Theoretical Computer Science* 249.1 (2000), pp. 3–80.
- [Rut98] Jan J. M. M. Rutten. “Automata and Coinduction (An Exercise in Coalgebra)”. In: *CONCUR*. Springer, 1998, pp. 194–218.
- [SC00] Greg Schohn and David Cohn. “Less is More: Active Learning with Support Vector Machines”. In: *International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., 2000, pp. 839–846.

Bibliography

- [Sch05] Lutz Schröder. “Expressivity of Coalgebraic Modal Logic: The Limits and Beyond”. In: *Foundations of Software Science and Computational Structures*. Springer Berlin Heidelberg, 2005, pp. 440–454.
- [Set12] Burr Settles. “Active Learning”. In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* (2012).
- [SG09] Muzammil Shahbaz and Roland Groz. “Inferring Mealy Machines”. In: *Formal Methods*. Springer Berlin Heidelberg, 2009, pp. 207–222.
- [SHV16] Mathijs Schuts, Jozef Hooman, and Frits Vaandrager. “Refactoring of Legacy Software Using Model Learning and Equivalence Checking: An Industrial Experience Report”. In: *Proceedings of the 12th International Conference on Integrated Formal Methods - Volume 9681*. Berlin, Heidelberg: Springer-Verlag, 2016, pp. 311–325.
- [Sim11] Harold Simmons. *An Introduction to Category Theory*. Cambridge University Press, 2011.
- [Sip13] Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, 2013.
- [Sok11] Ana Sokolova. “Probabilistic systems coalgebraically: A survey”. In: *Theoretical Computer Science* 412.38 (2011), pp. 5095–5110.
- [Tay99] Paul Taylor. *Practical Foundations of Mathematics*. Cambridge University Press, 1999.
- [Ton01] Simon Tong. “Active learning: theory and applications”. PhD thesis. Stanford University, 2001.
- [Trn71] Věra Trnková. “On descriptive classification of set-functors. I.” In: *Commentationes Mathematicae Universitatis Carolinae* 12.1 (1971), pp. 143–174.
- [Tze92] Wen-Guey Tzeng. “Learning Probabilistic Automata and Markov Chains via Queries”. In: *Machine Learning* 8.2 (1992), pp. 151–166.

Bibliography

- [US20] Henning Urbat and Lutz Schröder. “Automata Learning: An Algebraic Approach”. In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 900–914.
- [Var97] Y. Moshe Vardi. *Why Is Modal Logic So Robustly Decidable?* Tech. rep. 1997.
- [Vil96] Juan Miguel Vilar. “Query Learning of Subsequential Transducers”. In: *Grammatical Inference: Learning Syntax from Sentences (ICGI)*. Vol. 1147. Lecture Notes in Computer Science (LNCS). Springer, 1996, pp. 72–83.
- [Wiß+19] Thorsten Wißmann et al. “A coalgebraic view on reachability”. In: *Commentationes Mathematicae Universitatis Carolinae* 60.4 (2019), pp. 605–638.
- [WS16] Yining Wang and Aarti Singh. “Noise-Adaptive Margin-Based Active Learning and Lower Bounds under Tsybakov Noise Condition”. In: *Conference on Artificial Intelligence*. 2016, pp. 2180–2186.
- [WS21] Thomas Wilke and Sven Schewe. “ ω -Automata”. In: *Handbook of Automata Theory*. Ed. by Jean-Éric Pin. Zürich, Switzerland: European Mathematical Society Publishing House, 2021, pp. 189–234.

Bibliography