CALF
Categorical Automata Learning Framework

Gerrit Kornelis van Heerdt

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
at
University College London.

27th August 2020

I, Gerrit Kornelis van Heerdt, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

Automata learning is a popular technique used to automatically construct an automaton model from queries, and much research has gone into devising specific adaptations of such algorithms for different types of automata. This thesis presents a unifying approach to many existing algorithms using category theory, which eases correctness proofs and guides the design of new automata learning algorithms. We provide a categorical automata learning framework—CALF—that at its core includes an abstract version of the popular $L^\star$ algorithm. Using this abstract algorithm we derive several concrete ones.

We instantiate the framework to a large class of **Set** functors, by which we recover for the first time a tree automata learning algorithm from an abstract framework, which moreover is the first to cover also algebras of quotiented polynomial functors. We further develop a general algorithm to learn weighted automata over a semiring. On the one hand, we identify a class of semirings, principal ideal domains, for which this algorithm terminates and for which no learning algorithm previously existed; on the other hand, we show that it does not terminate over the natural numbers. Finally, we develop an algorithm to learn automata with side-effects determined by a monad and provide several optimisations, as well as an implementation with experimental evaluation. This allows us to improve existing algorithms and opens the door to learning a wide range of automata.

Impact

The framework developed in this thesis impacts future work in both academia and the industry. In academia, CALF provides deep insight into the fundamental theory that enables automata learning algorithms, which advances the general study of automata seen from an abstract perspective. In one concrete direction, CALF will supply the theoretical infrastructure for the CLeVer EPSRC Standard Grant,[1] which plans to investigate learning techniques for concurrent models and develop a novel verification framework for concurrency in hardware systems.

As is the case with many fields in Computer Science, there is a clear interplay between academia and the industry when it comes to automata learning. Industrial demands to learn models describing the behaviour of systems with ever increasing complexity stimulates research into developing such algorithms. Using CALF, new algorithms that are correct by construction can be derived from the abstract template. These algorithms are then applied for purposes such as verification, and as a result we contribute to ensuring that systems work as intended. The CLeVer project is an example of this: the developed verification methods for hardware systems will be of use to companies such as ARM. CALF also helps to transfer optimisations between algorithms for different types of automata, which will have an impact on tackling issues of scalability.

[1] `https://gow.epsrc.ukri.org/NGBOViewGrant.aspx?GrantRef=EP/S028641/1`

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Models to describe the behaviour of a system are ubiquitous in computer science. The abstractions made by such models enable efficient analysis by both humans and computers, via techniques such as model checking [CGP99; BK08]. One important type of model is the *automaton*, usually represented as a state diagram consisting of the behavioural states of the system, with transitions representing the actions one can take within a particular state. The simplicity of automata allows for efficient procedures to perform operations such as minimisation and equivalence checking.

Automata learning is the process of inferring an automaton model of a system from information about its behaviour. Automata learning algorithms have found diverse applications over the past decade, ranging from reverse-engineering implementations of network protocols [Cho+10; RP15; FJV16] and smartcard readers [Cha+14] to describing the errors in a program [Cha+15] and refactoring legacy software [SHV16] (see also [Vaa17] for an overview).

In this introduction we discuss automata learning and efforts to generalise the popular $\mathtt{L}^\star$ algorithm. We start by introducing the concept of learning from queries in Section 1.1, which ends with an example of manually learning an automaton (Section 1.1.1). We then discuss adaptations of $\mathtt{L}^\star$ for different types of automata in Section 1.2. This ends with a problem statement about unifying learning algorithms for these different automata, for which we introduce a method involving category theory in Section 1.3. Based on this, we set out the main aims of the thesis in Section 1.4. We then discuss related work in Section 1.5 and give an overview of the rest of the thesis and its contributions in Section 1.6.

1.1 Learning from Queries

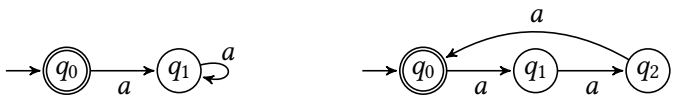
In this thesis we focus on *active automata learning* algorithms, which interact with (*query*) the target system. This allows such an algorithm to choose what information it wants to have based on previous interactions. Contrasted with active automata learning is *passive automata learning*, where one takes predefined data and tries to find an automaton that generalises the data as well as possible. The case studies mentioned earlier all use active learning algorithms, as the learner has access to the system under consideration.

Arguably the most fundamental type of automaton is the *deterministic finite automaton* (DFA). Parametric on an *alphabet* set of possible inputs, it consists of a set of states, each of which is either accepting or rejecting, together with a designated initial state and for each state and input a next state. Examples over a single input a are depicted in Figure 1.1, which will later be used to demonstrate the learning process. We indicate the initial state with an arrow without origin and mark accepting states with a double circle.

Semantics of DFAs is defined in terms of acceptance of *words*, finite sequences of inputs. Given a word, one can start from the initial state of a DFA, follow the transitions corresponding to the inputs of the sequence in order, and check if the resulting state is accepting. The word is accepted if and only if that state is. A classification of words into accepted and rejected ones is called a *language*, and for each language there is a unique minimal DFA accepting it. Systems exhibiting these languages as their behaviour are essentially simple *parsers* that read an input sequence and determine whether it is valid.

In active automata learning, the most straightforward interaction the learner could make with the target system is referred to as a *membership query:* the learner submits a word and observes whether the system accepts the word. Gold defined the concept of learning *in the limit:* the learner 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 [Gol67]. Later algorithms made additional assumptions on the size of the state space of the minimal automaton accepting the language, enabling a terminating learning algorithm that however requires an exponential number of membership queries [AZ69].

Moore showed that just membership queries are insufficient to correctly identify a correct DFA in finite time [Moo56]. The intuition behind this is simple: if only for a finite set of words it is known whether these are accepted, then there are multiple languages consistent with the information, and the algorithm has no means to distinguish between them. It was shown by Angluin [Ang81] that any algorithm using membership queries in combination with knowledge of the number of states of the minimal DFA must use a number of queries



(a) (b)

Figure 1.1: Example of manually learning a DFA.

exponential in that number in the worst case. To obtain a polynomial time algorithm, it was clear that a stronger assumption would be necessary.

Later, Angluin [Ang88] evaluated the use of various types of additional queries in automata learning. Apart from membership queries, the query types considered consisted in the learner submitting a *hypothesis* DFA and being told in return how this DFA relates to the target language. One such query that has gained much attention is the *equivalence query*, where the learner is told whether the hypothesis accepts the correct language and, if not, receives a *counterexample:* a word incorrectly classified by the hypothesis.

The $\mathrm{L}^\star$ algorithm [Ang87] works with membership and equivalence queries, using which it runs in time polynomial in the size of the alphabet, the number of states of the minimal DFA accepting the target language, and the length of the longest counterexample. Since equivalence queries are not as straightforward to implement as membership queries, which can often be realised just by interacting with the system, one speaks of an *oracle* or *teacher* providing an interface of membership and equivalence queries to the learner. The assumption of being able to pose equivalence queries is an elegant abstraction from an in practice often impossible problem, which can be approximated or otherwise dealt with according to the application. The abstraction enables a clearer view on the core of the learning algorithm that may be used as a template for practical solutions. Equivalence queries are usually implemented via a number of membership queries, either by assuming limits on the size of the target DFA in order to derive an exhaustive test set, or by determining according to the probably approximately correct framework [Val84] a number of random queries to test based on confidence and accuracy parameters.

1.1.1 Example of Learning from Membership and Equivalence Queries

We now perform an example of learning a DFA from membership and equivalence queries. Instead of getting lost in the details of the $\mathrm{L}^\star$ algorithm, we provide an intuitive account of the accumulating knowledge that drives it during a run. To this end, let us fix a singleton alphabet $\{a\}$ and consider an unknown target language $\mathcal{L} \subseteq \{a\}^\star$.

We know that a DFA accepting this language will need at least one state, the initial one.

To determine whether this state should be accepting, we may pose a membership query for the empty word. Suppose the teacher replies that the empty word is accepted. This means that the initial state has to be an accepting one. If the DFA is to have only a single state, there is only one transition on a possible for that state: a self-loop. One immediate check we can perform for this is posing a membership query for the word a. Supposing the teacher replies that a is rejected, we need to add a rejecting state and route the transition from the initial state to that state. It remains to determine where the transition coming from the second state leads. To do so, we pose a membership query for the word aa. The teacher replies it is not accepted, so we equip the second state with a self-loop, as shown in Figure 1.1a, and submit this hypothesis in an equivalence query.

Let us assume that the teacher replies that the above hypothesis is incorrect and provides the counterexample aaa: this word is rejected by the hypothesis but apparently occurs in the target language. Thus, we need to make sure that the next hypothesis accepts the word aaa. Consider again our original hypothesis from Figure 1.1a. Reading either the word a or the word aa, we end up in q_1, but reading another a from there, the acceptance is supposed to differ: aa is rejected while aaa is accepted. Thus, the state reached after reading a cannot be the same as the state reached after reading aa. In order to resolve this, we split q_1 and add a third state, q_2, to which the transition from q_1 leads. Since aa is rejected, q_2 becomes a rejecting state. We know that aaa is accepted, so the transition from q_2 leads to the only accepting state q_0. This gives us the hypothesis shown in Figure 1.1b. Supposing that on an equivalence query the teacher informs us that this hypothesis is correct, we conclude that the target language is given by the set of words over $\{a\}$ of which the length is a multiple of 3. Note that Figure 1.1b contains the minimal DFA accepting this language.

The $L^\star$ algorithm generalises the above process in a systematic way. Attempting to construct a hypothesis DFA consistent with the current knowledge of the language leads to the discovery of additional states. Once a hypothesis is ready and an equivalence query results in a counterexample, the counterexample further drives the state discovery process. This continues until no new counterexample is provided, which proves that the hypothesis is correct. We will explain the algorithmic details of $L^\star$ in Section 2.3.

1.2 Learning Different Types of Automata

Since $L^\star$ appeared in 1987 [Ang87], many variations on it have been proposed. These include adjustments to the way counterexamples are handled [MP95; RS93] and more efficient data structures [KV94; IHS14], but also many adaptations to other types of automata than the DFAs

that were originally learned.

Such adaptations were motivated by the fact that although the learning algorithm for DFAs provided an elegant theoretical result, applicability remained limited. In many applications, one wants to learn a model describing the behaviour of a system that produces one or more outputs after each individual input—a symbol from the alphabet—is submitted. Vilar generalised the $L^\star$ algorithm to learn *subsequential transducers*, which produce words over an output set on both states and transitions [Vil96]. Since then, many applications have focused on learning the less complex *Mealy machines*, which allow just a basic input/output pair on each transition. First appearing in [PO98], algorithms to learn Mealy machines have been applied in many case studies, including, with one exception, the ones listed near the beginning of this chapter.

Apart from deterministic automata, adaptations of $L^\star$ have been introduced to learn for instance weighted automata [BV96], which are used in image processing [CK93], text and speech recognition [MPR05], and bioinformatics [AMT08]. Other adaptations of $L^\star$ focus on nondeterministic automata [Bol+09] or universal and alternating automata [AEF15; Ber+17], all of which accept the same class of languages as DFAs but provide a more succinct state space via a more complex transition type. Learning such automata can thus be seen as an optimisation.

Another type of automaton is given by *Büchi automata*, which accept words of infinite length and are widely used for verification purposes [CGP99; BK08]. An algorithm to learn languages accepted by both Büchi and co-Büchi automata was developed in [MP95]. Much later, an algorithm was introduced to learn the full class of languages accepted by Büchi automata [AF16], via a type of automaton consisting of a family of DFAs. These accept languages of pairs of words that are similar in principle to the so-called *lasso languages*, which capture words made up by an infinite repetition following a finite prefix [CNP93].

One may also consider data languages, which can contain parameters from an infinite domain. Such languages are typically processed using *register automata*, of which there are many variations and for which several learning algorithms have been proposed [How+12; Bol+13; Cas+16]. One alternative to these automata is given by *nominal automata*, a redefinition of deterministic (and nondeterministic) automata based on *nominal sets* rather than plain sets. Nominal sets are 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 [Gab01; Pit13]. An adaptation of $L^\star$ for nominal automata was developed a few years ago [Moe+17].

Whenever a new type of automaton is identified for which an adaptation of $L^\star$ is desired, the details of the algorithm and its correctness proof need to be devised from scratch. The

mathematical intricacies associated with the type of automaton may distract from the connection with algorithms for other types of automata and thus cloud the fact that optimisations and other variations can often be transferred. In this thesis we develop a framework in which the core of the algorithm and its correctness proof can be studied on an abstract level. To do so, we will use category theory and the perspective it offers on automata, which are introduced in the next section.

1.3 Categorical Perspective on Automata and Learning

In computer science, category theory is often seen as a formalism using which various mathematical structures can be studied uniformly. *Algebras* and *coalgebras*, dependent on a *functor*, are prime examples covering many classes of structures. They play an important role in studying models of systems, where the functor determines the type of system [Man76; Rut00; Rut19]. Algebras intuitively describe how states of a system are *constructed* from previous states and additional input data, while coalgebras can be seen to *decompose* states of the system into successor states and output data, generating behaviour [Jac17].

In this context, automata are often modelled as either algebras or coalgebras together with initial states and outputs. For instance, automata that process binary trees are based on algebras defined using a transition type (functor) given by pairing a set with itself. Similarly, DFAs are recovered using algebras for the functor that pairs a set with the input alphabet, and nominal automata using the functor that pairs a nominal set with a nominal input alphabet. In these last cases the automata can also be modelled using coalgebras: DFAs are recovered using coalgebras for the functor that takes a set and returns the functions from the input alphabet into that set.

By varying the category and functor involved, many different types of automata can be recovered. These include *weighted finite automata* (WFAs), probabilistic automata, nondeterministic automata, and the families of DFAs mentioned earlier [CV12].

Given this diverse collection of automata covered within the categorical view, it would seem like a promising setting for generalising automata learning algorithms. This was first noticed by Jacobs and Silva [JS14], who redefined certain constructions in the $\mathtt{L}^\star$ algorithm using categorical tools and showed that it unified algorithms for DFAs and WFAs. Van Heerdt [Hee16] then described explicitly under which conditions these constructions work, captured the central data structure abstractly, and developed theorems to characterise correctness in terms of this data structure.

However, this work did not define an abstract version of $\mathtt{L}^\star$ but only provided very high

level guidelines for designing new algorithms. Moreover, the conditions on the categorical setting were too strict to enable for instance the setting of tree automata to be captured.

1.4 Main Aims

In this thesis we aim to build a categorical automata learning framework that features a provably correct abstract version of $\mathtt{L}^\star$, in which each step of the original algorithm receives an explicit abstract analogue. From this framework different algorithms for various types of automata can be studied and developed uniformly. Apart from resulting in new algorithms, we expect that this provides new insights into existing ones, allowing us to devise variations and optimisations by transferring those from different settings.

Our actual contributions will be discussed in detail in Section 1.6. We first review related work.

1.5 Related Work

Several categorical automata learning frameworks have been proposed over the last few years. This thesis itself is a further development of the work that started from [JS14] and was initially developed in the author's master thesis [Hee16]. The developments set out in Section 1.4 are additions and improvements to this work.

Barlocco, Kupke, and Rot [BKR19] proposed an abstract algorithm to learn coalgebras by using coalgebraic modal logic to characterise tests. Urbat and Schröder [US19] developed another abstract automata learning algorithm that is relatively close to our approach. However, their work focuses on automata that can be seen both as algebras and as coalgebras. They study a reduction of certain algebraic automata, but in the case of for example tree automata this reduction leads to automata over an infinite alphabet, leaving it unclear how to work with them in practice.

Apart from categorical approaches, there have been other proposals of automata learning frameworks. Balcázar et al. [Bal+97] provided a common interface for studying different learning algorithms for DFAs. In Isberner's thesis [Isb15], the focus is put on developing efficient algorithms, and although several types of automata are studied and terminology is shared between those, the approach does not scale to more complex automata than deterministic ones that read words.

Our abstract notion of automaton is based on the work of Arbib and Manes from the early 70s [AM74; AM75a; AM75b]. Those developments started with a unified formalism

that captured both deterministic and weighted automata due to Arbib and Zeiger [AZ69], which was first picked up by Goguen [Gog72a; Gog72b]. The abstract treatment of Goguen, however, did not allow for certain important examples, such as tree automata. These were studied by Arbib and Manes [AM74] and feature in Chapters 3 and 4 of the present thesis. An extensive overview of the theory of abstract automata and minimal realisations has been given by Adámek and Trnková [AT89]. Arbib and Manes also studied automata with side-effects [AM75b], for which we develop a (coalgebraic) learning algorithm in Chapter 6.

The aim in the original work of Arbib and Zeiger [AZ69] was not just to unify descriptions of automata, but also to provide a generic construction of minimal realisations. These constructions did not just include theoretical ones, in the way the Myhill–Nerode congruence for a regular language gives rise to a minimal DFA, but also a practical one. They investigated identification procedures to construct the minimal realisation by recording only for finitely many words whether they are in the language. For this purpose, however, they need to assume an upper bound on the number of states of the minimal realisation, in which the number of words evaluated in the language is exponential. This line of research continued with Gold [Gol72], who dropped the condition of knowing such an upper bound and instead provided an algorithm that learns in the limit: it continually collects additional data and guarantees only to converge to the minimal realisation after an unknown amount of time.

The abstract view on automata as (co)algebras has enabled generalisations of various algorithms other than automata learning over recent years. These include algorithms for minimisation [Adá+12; KK14; Dor+17], determinisation [Sil+13], and equivalence checking [Rot15; BP15]. Both minimisation and equivalence checking are deeply related to automata learning, as we showed in [HSS17a]: automata learning is essentially a generalised form of minimisation (see also Section 4.5), and (an extension of) the data structure produced in the process of learning an automaton can be used to decide whether it is equivalent to other automata. In Chapter 6 we will initially introduce an algorithm that learns the determinisation of an automaton with side-effects.

A detailed overview of related work will be provided at the end of each chapter.

1.6 Overview and Contributions

In Chapter 3 we explore minimisation of automata on an abstract level. Our main result is an iterative construction for minimising these automata (Theorem 3.3.6), which resembles partition refinement. Furthermore, we study a different characterisation of minimality via the Nerode equivalence and provide a generalisation using monads that allows to treat auto-

mata with equations. Crucially, we give conditions under which the existence of this abstract Nerode equivalence corresponds to the existence of a minimal automaton (Theorems 3.4.11 and 3.4.14). The chapter is based on the following paper, which is a joint effort between its authors:

- [Hee+19b] Gerco van Heerdt, Tobias Kappé, Jurriaan Rot, Matteo Sammartino and Alexandra Silva. "Tree automata as algebras: Minimisation and determinisation". In: *CALCO*. vol. 139. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 6:1–6:22. DOI: 10.4230/LIPIcs.CALCO.2019.6.

Chapter 4 contains the main results. We develop a categorical automata learning framework (CALF)[1] with abstract versions of each step in the $\mathtt{L}^\star$ algorithm, including an abstract treatment of counterexamples. Most importantly, in Theorem 4.4.7 we prove that the algorithm terminates with a correct automaton, and we give conditions under which this automaton is minimal. We then instantiate our abstract $\mathtt{L}^\star$ algorithm to a concrete setting, providing the first learning algorithm for tree automata derived abstractly (Section 4.6). The chapter is based on the following papers, of which the author of this thesis is the main author:

- [HSS17a] Gerco van Heerdt, Matteo Sammartino and Alexandra Silva. "CALF: Categorical Automata Learning Framework". In: *CSL*. vol. 82. LIPIcs. 2017, 29:1–29:24. DOI: 10.4230/LIPIcs.CSL.2017.29.

- [Hee+20a] Gerco van Heerdt, Tobias Kappé, Jurriaan Rot, Matteo Sammartino and Alexandra Silva. "A Categorical Framework for Learning Generalised Tree Automata". In: *CoRR* (2020). arXiv: 2001.05786. URL: https://arxiv.org/abs/2001.05786.

In Chapter 5 we develop an algorithm to learn weighted automata over *principal ideal domains*. We first introduce a general weighted adaptation of $\mathtt{L}^\star$ parametric on an arbitrary semiring, together with conditions for termination that we prove sufficient (Theorem 5.2.10). We then prove that not all semirings satisfy these conditions, and in particular that the algorithm does not terminate when instantiated to the natural numbers (Theorem 5.3.1). We then provide our main result in which we prove that the algorithm terminates if the semiring is a principal ideal domain (Theorem 5.4.10). This yields the first active learning algorithm for WFAs over the integers. The chapter is based on the following paper, which is a joint effort between its authors:

[1] http://www.calf-project.org

- [Hee+20b] Gerco van Heerdt, Clemens Kupke, Jurriaan Rot and Alexandra Silva. "Learning Weighted Automata over Principal Ideal Domains". In: *FoSSaCS*. LNCS. Springer. 2020, pp. 602–621. DOI: 10.1007/978-3-030-45231-5_31.

In Chapter 6 we generalise the algorithm from Chapter 5 to learn automata with side-effects given by a monad (with a finiteness restriction). Our main result is a general algorithm to learn automata in the category of algebras for a monad, which we prove correct (Section 6.3.1). We then optmise by replacing the hypothesis by a succinct one that exploits the side-effects enabled by the monad, and we also optimise the counterexample handling method. By doing so we transfer this last optimisation that was originally developed for learning DFAs to various settings where it had not been considered before, including NFAs and WFAs. We provide a Haskell library to apply the algorithm and explain in detail how it can be instantiated to NFAs and WFAs over a finite semiring. Finally, we describe experimental results for the NFA and WFA cases, comparing the optimisations enabled by our library. For NFAs we show that the counterexample handling optimisation leads to an improvement in the number of membership queries. The chapter is based on the following paper, of which the author of this thesis is the main author:

- [HSS20] Gerco van Heerdt, Matteo Sammartino and Alexandra Silva. "Learning Automata with Side-Effects". In: *CMCS*. 2020, to appear.

An extended but unpublished version of this paper is given in [HSS17b]. This includes the implementation and experiment sections missing in the above publication.

The algorithm presented in Chapter 5 would be an instance of the one in Chapter 6, with the succinctness optimisation applied, if the latter did not have the requirement of the monad preserving finite sets. That requirement is satisfied whenever the semiring considered in Chapter 5 is finite (Example 5.2.12). The initial algorithm developed in Chapter 6 can be seen as an instance of the main algorithm in Chapter 4 when instantiating to the opposite of the category of algebras for the monad.

We provide final thoughts on the future of automata learning from our categorical perspective in Chapter 7. First, in Chapter 2, we start with preliminary notions that will be useful throughout the thesis.

We note that the articles [HSS17a; HSS20] are originally based on the author's master thesis:

- [Hee16] Gerco van Heerdt. "An Abstract Automata Learning Framework". MA thesis. Radboud University Nijmegen, 2016. URL: https://www.ru.nl/publish/pages/

769526/gerco_van_heerdt.pdf.

1.6.1 Additional Publications

Apart from the publications mentioned above, the author produced the following articles over the course of the studies that culminated in the present thesis.

- [Hee+18b] Gerco van Heerdt, Bart Jacobs, Tobias Kappé and Alexandra Silva. "Learning to Coordinate". In: *It's All About Coordination*. LNCS. Springer, 2018, pp. 139–159. DOI: 10.1007/978-3-319-90089-6_10.

  In this work we extended the original foundations of CALF laid by Jacobs and Silva [JS14], adding a new corrected proof of minimality for hypotheses and an example application to Reo automata, which provide a semantics for Reo circuits. The present author's contributions consisted in providing a new proof of minimality for hypotheses, making use of recursive coalgebras and corecursive algebras; bringing the presentation closer to the implicit categorical generalisation; and correcting an example.

- [Hee+18a] Gerco van Heerdt, Justin Hsu, Joël Ouaknine and Alexandra Silva. "Convex language semantics for nondeterministic probabilistic automata". In: *ICTAC*. vol. 11187. LNCS. Springer. 2018, pp. 472–492. DOI: 10.1007/978-3-030-02508-3_25.

  We explored language semantics for nondeterministic probabilistic automata, proving that the categorical view on automata leads to exactly two natural options. We also showed that in both of these cases the nondeterminism allows to express more than what could be expressed with deterministic probabilistic automata, and we proved that language equivalence is undecidable (or at least hard, in the case of a unary alphabet). We finally provided a discounted metric to approximate language equivalence to arbitrary precision. The author of this thesis is the main author of the paper, apart from the proofs of extended expressivity and hardness of equivalence, both in the case of a unary alphabet.

- [Hee+19a] Gerco van Heerdt, Joshua Moerman, Matteo Sammartino and Alexandra Silva. "A (co)algebraic theory of succinct automata". In: *JLAMP* 105 (2019), pp. 112–125. DOI: 10.1016/j.jlamp.2019.02.008.

  We studied the reverse question to determinisation—given a deterministic automaton and a type of side-effect, can we find an automaton with such side-effects that is as small as possible? We answered this question positively with an algorithm—although the

automata produced are minimal with respect to a certain property and not necessarily minimal in size—and explored various examples, including alternating automata and weighted automata. The author of this thesis is the main author of the paper.

Chapter 2

Preliminaries

In this chapter we recall concepts and set notation that will be used throughout this thesis. We start by introducing semirings and their semimodules in Section 2.1. These will be useful in Chapters 5 and 6. We then move on to category theory in Section 2.2, which ends with an introduction to categorical automata. Knowledge of category theory will not be required for Chapter 5. Finally, we explain the $\mathtt{L}^\star$ algorithm in Section 2.3, which is relevant to every chapter apart from Chapter 3.

2.1 Semirings and Semimodules

A *semiring* is a ring that does not necessarily admit an additive inverse. For instance, the natural numbers with the usual addition and multiplication operations form a semiring, but not a ring. We give a formal definition below.

Definition 2.1.1 (Semiring)**.** A *semiring* $\mathbb{S}$ is a set with two monoid structures denoted by $+$ and $\cdot$, where the *addition* $+$ is commutative and has an identity element 0, the *multiplication* $\cdot$ has an identity element 1, and the following distributivity equations hold (for $a, b, c \in \mathbb{S}$):

$$a \cdot (b + c) = a \cdot b + a \cdot c \qquad (a + b) \cdot c = a \cdot c + b \cdot c \qquad 0 \cdot a = a \cdot 0 = 0$$

Apart from the natural numbers, examples include the non-negative rational numbers and the *boolean semiring* $\{\mathbf{true}, \mathbf{false}\}$ with addition given by "or" and multiplication by "and".

The generalisation of the notion of vector space from fields to semirings is called *semimodule*.

Definition 2.1.2 (Semimodule)**.** A *(left) semimodule* M over a semiring $\mathbb{S}$ consists of a monoid structure on M, written using $+$ as the operation and 0 as the unit, together with a scalar

multiplication map $\cdot : \mathbb{S} \times M \longrightarrow M$ such that:

$$s \cdot 0_M = 0_M \qquad 0_{\mathbb{S}} \cdot m = 0_M \qquad 1 \cdot m = m$$
$$s \cdot (m + n) = s \cdot m + s \cdot n \qquad (s + r) \cdot m = s \cdot m + r \cdot m \qquad (sr) \cdot m = s \cdot (r \cdot m).$$

When the semiring is in fact a ring, we speak of a *module* rather than a semimodule. In the case of a field, the concept instantiates to a *vector space*.

As an example, commutative monoids are the semimodules over the semiring of natural numbers. Any semiring forms a semimodule over itself by instantiating the scalar multiplication map to the internal multiplication. If X is any set and M is a semimodule, then M^X with pointwise operations also forms a semimodule. A similar semimodule is the *free semimodule* over X, which differs from M^X in that it fixes M to be the relevant semiring and requires its elements to have *finite support*. This enables an important operation called *linearisation*. Below we first define free semimodules.

Definition 2.1.3 (Free semimodule)**.** Given a semiring $\mathbb{S}$, the *free semimodule* over a set X is given by the set

$$V(X) = \{f : X \longrightarrow \mathbb{S} \mid \mathsf{supp}(f) \text{ is finite}\}$$

with pointwise operations. Here $\mathsf{supp}(f) = \{x \in X \mid f(x) \neq 0\}$. We sometimes identify the elements of $V(X)$ with formal sums over X. Any semimodule isomorphic to $V(X)$ for some set X is called free.

If X is a finite set, then $V(X) = \mathbb{S}^X$. We now define the linearisation of a function into a semimodule, which uniquely extends it to a semimodule homomorphism, witnessing the fact that $V(X)$ is free.

Definition 2.1.4 (Linearisation)**.** Given a set X, a semimodule M, and a function $f : X \longrightarrow M$, we define the *linearisation* of f as the semimodule homomorphism $f^\sharp : V(X) \longrightarrow M$ given by

$$f^\sharp(\alpha) = \sum_{x \in X} \alpha(x) \cdot f(x).$$

The operation $(-)^\sharp$ has an inverse that maps a semimodule homomorphism $g : V(X) \longrightarrow M$ to the function $g^\dagger : X \longrightarrow M$ given by

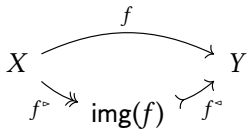
$$g^\dagger(x) = g(\partial_x), \qquad \partial_x(y) = \begin{cases} 1 & \text{if } y = x \\ 0 & \text{if } y \neq x. \end{cases}$$

2.2 Category Theory

We assume basic knowledge of categories, functors, natural transformations, limits, and duality (see for instance [Awo10]). Below we discuss several further topics: factorisation systems (Section 2.2.1), algebras and coalgebras (Section 2.2.2), monads and algebras for a monad (Section 2.2.3), and finally automata (Section 2.2.4).

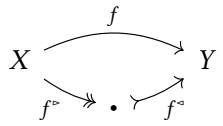
2.2.1 Factorisation Systems

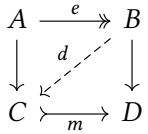
The notion of *factorisation system* generalises the ability to take images of functions to the categorical level. The idea is that a function $f : X \longrightarrow Y$ can be factorised as



where $f^{\triangleright}$ is the surjective version of f—restricted to having its image as the codomain—and $f^{\triangleleft}$ is the inclusion of the image of f into Y. This factorisation also has a uniqueness property, which altogether generalises as follows.

Definition 2.2.1 (Factorisation system)**.** An $(\mathcal{E}, \mathcal{M})$-*factorisation system* on a category $\mathbf{C}$ consists of classes of morphisms $\mathcal{E}$ and $\mathcal{M}$, closed under composition with isos, such that $\mathcal{E}$ consists of epis, $\mathcal{M}$ consists of monos, and for every morphism f in $\mathbf{C}$ there exist $f^{\triangleright} \in \mathcal{E}$ and $f^{\triangleleft} \in \mathcal{M}$ with $f = f^{\triangleleft} \circ f^{\triangleright}$, as indicated below on the left, and we have a unique diagonal fill-in property: for every commutative square as below on the right, with $e \in \mathcal{E}$ and $m \in \mathcal{M}$, there exists a unique diagonal $d : B \longrightarrow C$ making both triangles commute.





We often denote morphisms in $\mathcal{E}$ using double-headeded arrows and morphisms in $\mathcal{M}$ using tailed arrows when these properties are relevant.

As suggested above, (surjective functions, injective functions) forms a factorisation system in **Set**. Similarly, in the category **Vect** of vector spaces and linear maps, a factorisation system is given by (surjective linear maps, injective linear maps).

Define by $\leq$ the order on morphisms with common domain given by $f \leq g$ if and only if there exists a morphism h such that $h \circ f = g$. This induces an equivalence relation on such

morphisms. In the context of a factorisation system $(\mathcal{E}, \mathcal{M})$, a *quotient* of an object X is a morphism $q : X \twoheadrightarrow X'$ in $\mathcal{E}$ identified up to the equivalence, i.e., an equivalence class. This works because $\mathcal{E}$ is closed under composition with isomorphisms. For quotients of an object, the order defined above intuitively says a quotient is bigger if it identifies more.

Similarly, we also denote by $\leq$ the order on morphisms with common codomain given by $f \leq g$ if and only if there exists a morphism h such that $g \circ h = f$, which again induces an equivalence relation on such morphisms. A *subobject* of an object X is a morphism $m : X' \rightarrowtail X$ in $\mathcal{M}$ identified up to this equivalence. For subobjects of an object, the order defined above intuitively says a subobject is bigger if it contains more.

We sometimes refer to quotients or subobjects by representatives of the equivalence class. Note that two representatives are isomorphic (ordered in both directions via an isomorphism) if and only if they are in the same equivalence class. We thus sometimes speak of isomorphic quotients or subobjects when considering two representatives that are in the same equivalence class.

2.2.2 Algebras and Coalgebras

An algebra for a functor is the categorical generalisation of an algebraic operation, where the functor determines the type of the operation. Such algebras will form the core of the automata defined in Section 2.2.4, generalising the transition functions of DFAs.

Definition 2.2.2 (Algebra for a functor)**.** Given a category $\mathbf{C}$, an *algebra* for a functor $F : \mathbf{C} \to \mathbf{C}$, also called an F-algebra, is a tuple (X, χ), where $\chi : FX \to X$ is any morphism. Given two F-algebras (X, χ) and (Y, ψ), an *F-algebra homomorphism* $f : (X, \chi) \to (Y, \psi)$ is a morphism $X \to Y$ making the diagram below commute.

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

For example, an algebra for the functor $FX = X \times X$ on **Set** is a binary operation on the set X. An F-algebra homomorphism is a function preserving the operation. For instance, if (X, χ) and (Y, ψ) are F-algebras for the above definition of F, then an F-algebra homomorphism $f : (X, \chi) \to (Y, \psi)$ is a function that satisfies $f(\chi(x_1, x_2)) = \psi(f(x_1), f(x_2))$ for all $x_1, x_2 \in X$.

The dual of an algebra is called a coalgebra. Both algebras and coalgebras are used to abstractly describe the inner workings of systems, but whereas algebras describe how the

system is *constructed* from the operations expressed by the functor F, coalgebras describe how to *deconstruct* it and extract observations from states.

Definition 2.2.3 (Coalgebra for a functor)**.** Given a category $\mathbf{C}$, a *coalgebra* for a functor $F : \mathbf{C} \to \mathbf{C}$, also called an F-coalgebra, is a tuple (X, χ), where $\chi : X \to FX$ is any morphism. Given two F-coalgebras (X, χ) and (Y, ψ), an *F-coalgebra homomorphism* $f : (X, \chi) \to (Y, \psi)$ is a morphism $X \to Y$ making the diagram below commute.

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

For example, a coalgebra for the functor $FX = X \times X$ on **Set** is an operation that takes one element from the set X and produces two in return. An F-coalgebra homomorphism is a function preserving this operation. For instance, if (X, χ) and (Y, ψ) are F-coalgebras for the above definition of F, then an F-algebra homomorphism $f : (X, \chi) \to (Y, \psi)$ is a function that satisfies $(f(x_1), f(x_2)) = \psi(f(x))$ for all $x \in X$ and letting $(x_1, x_2) = \chi(x)$.

One can describe various types of state-based systems using coalgebras, including labelled transition systems and automata such as deterministic, nondeterministic, and weighted automata [Rut00]. We note that, as coalgebras, these models do not include initial states. Similarly, deterministic automata can be modelled as algebras, in which case their states do not carry the accept/reject distinction. In Section 2.2.4 we will introduce an extended model that fully captures the components of an automaton.

2.2.3 Monads and their Algebras

In computer science, *monads* are often seen as modelling a type of computation. They comprise a functor T together with natural transformations and compatibility laws between them that allow for composing morphisms $X \to TY$ with morphisms $Y \to TZ$ to form functions $X \to TZ$. The additional expressive power enabled by the functor T in these computations is sometimes referred to as a *side-effect* (see for instance Chapter 6). Below we give the formal definition.

Definition 2.2.4 (Monad)**.** A *monad* in a category $\mathbf{C}$ is a triple (T, η, μ), where $T : \mathbf{C} \to \mathbf{C}$ is a functor and $\eta : \mathsf{Id} \Rightarrow T$ and $\mu : TT \Rightarrow T$ are natural transformations making the diagrams

below commute.

We often identify the monad (T, η, μ) with the functor T.

One obvious example is the *identity monad* on any category: it consists of the identity functor with identity natural transformations. Below we give a few examples on the category of sets and functions.

Example 2.2.5 (Monads in **Set**)**.** An example of a monad in **Set** is the triple $(\mathcal{P}, \{-\}, \bigcup)$, where $\mathcal{P}$ denotes the powerset functor assigning to each set its set of subsets, $\{-\}$ is the singleton operation, and $\bigcup$ is union of sets. Another example is the triple $(V(-), e, m)$, where $V(X)$ for a semiring $\mathbb{S}$ is the free semimodule over X (see Definition 2.1.3), $e : X \longrightarrow V(X)$ assigns the characteristic function of x to each $x \in X$, which has weight 1 assigned to x and weight 0 to every other element, and $m : V(V(X)) \longrightarrow V(X)$ is defined for $\varphi \in V(V(X))$ and $x \in X$ as

$$m(\varphi)(x) = \sum_{\psi \in V(X)} \varphi(\psi) \cdot \psi(x).$$

An important concept is an algebra for a monad, which generalises for instance the semimodules over a semiring introduced in Section 2.1. It differs from an algebra over a functor in that there are additional laws involving the unit and multiplication of the monad that need to be obeyed.

Definition 2.2.6 (Algebras for a monad)**.** An *Eilenberg–Moore algebra algebra for a monad* (T, η, μ), also called a T-algebra[1], in a category $\mathbf{C}$ is a T-algebra (X, x) making the diagrams below commute.

Given two T-algebras (X, x) and (Y, y), a T-algebra homomorphism $f : (X, x) \longrightarrow (Y, y)$ is just a T-algebra homomorphism. Algebras for T and their homomorphisms form a category $\mathsf{EM}(T)$ called the *Eilenberg–Moore category of* T.

[1]Note that this is the same terminology as for an algebra for the functor T. The present notion will be intentended whenever there is a monad structure on T in the context, unless explicitly stated otherwise.

Example 2.2.7 (Monad algebras in **Set**)**.** Algebras for the powerset monad are complete join semilattices, with the operation $\mathcal{P}X \longrightarrow X$ representing the join. Algebras for the free semimodule monad V for a semiring $\mathbb{S}$ are precisely the semimodules over $\mathbb{S}$.

One important type of algebra over a monad is the *free T-algebra*.

Definition 2.2.8 (Free T-algebra)**.** Given a monad (T, η, μ) and an object X, the *free T-algebra over X* is given by (TX, μ_X). It is free in that for every T-algebra (Y, y) and morphism $f : X \longrightarrow Y$, there exists a unique T-algebra homomorphism $f^\sharp : (TX, \mu_X) \longrightarrow (Y, y)$ satisfying $f^\sharp \circ \eta_X = f$. We call this T-algebra homomorphism $f^\sharp$ the *extension* of f. The extension operation has an inverse $(-)^\dagger$ that maps a T-algebra homomorphism $g : (TX, \mu_X) \longrightarrow (Y, y)$ to a function $g^\dagger = g \circ \eta_X : X \longrightarrow Y$.

Apart from free algebras for a monad, we will also need free monads generated by a functor. These are known as *algebraically free* monads, which we introduce below.

Definition 2.2.9 (Algebraically free monad)**.** An *algebraically free monad* $(F^\star, \eta, \mu)$ *over a functor* $F : \mathbf{C} \longrightarrow \mathbf{C}$ is a monad in $\mathbf{C}$ such that the category of $F^\star$-algebras is isomorphic to the category of F-algebras, with the isomorphism commuting with the forgetful functors into $\mathbf{C}$. The F-algebra corresponding to a free $F^\star$-algebra $(F^\star X, \mu_X)$ is a component of a natural transformation $\theta : FF^\star \Longrightarrow F^\star$. Given an F-algebra (Y, y), we denote by $y^\star = \mathsf{id}_Y^\sharp : F^\star Y \longrightarrow Y$ the corresponding algebra for the monad, which is also an $F^\star$-algebra homomorphism $(F^\star Y, \mu_Y) \longrightarrow (Y, y)$.

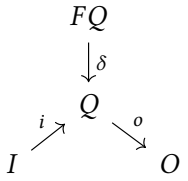
Algebraically free monads are also free in the sense of [Bar70], and conversely a free monad is algebraically free if the category it is defined on is locally small and complete [Kel80].

2.2.4 Automata and Languages

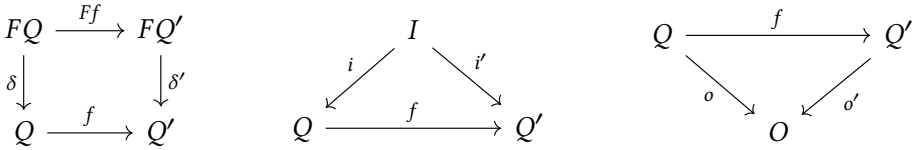
Below we introduce the categorical notion of automaton due to Arbib and Manes [AM74]. To this end, we fix an arbitrary category $\mathbf{C}$ with a factorisation system $(\mathcal{E}, \mathcal{M})$ and a functor $F : \mathbf{C} \longrightarrow \mathbf{C}$, as well as objects I and O. We assume F admits a free monad $(F^\star, \eta, \mu)$. An automaton is an (algebraic) transition structure together with initial states and outputs. The objects I and O serve to select initial states and provide output options, respectively.

Definition 2.2.10 (Automaton)**.** An *automaton* is a quadruple (Q, δ, i, o), where Q is the *state space*, $\delta : FQ \longrightarrow Q$ is the *dynamics*, $i : I \longrightarrow Q$ is the *initial state map*, and $o : Q \longrightarrow O$ is the

output map.



Given automata $\mathcal{A} = (Q, \delta, i, o)$ and $\mathcal{A}' = (Q', \delta', i', o')$ An *automaton homomorphism* $f : \mathcal{A} \to \mathcal{A}'$ is a morphism $Q \to Q'$ making the diagrams below commute.



Example 2.2.11. If $\mathbf{C} = \mathbf{Set}$ with $(\mathcal{E}, \mathcal{M}) = (\text{surjective}, \text{injective})$, $F = (-) \times A$ for a finite set A, $I = 1 = \{*\}$, and $O = 2 = \{0, 1\}$, we recover deterministic automata (DAs) as automata: the state space is a set Q, the transition function is the dynamics, the initial state is represented as a morphism $1 \to Q$, and the classification of states into accepting and rejecting ones is represented by a morphism $Q \to 2$. In this case we obtain the free monad $((-) \times A)^\star = (-) \times A^\star$, with its unit pairing an element with the empty word ε and the multiplication concatenating words. The extension of $\delta : Q \times A \to Q$ to $\delta^\star : Q \times A^\star \to Q$ is the usual one that lets the automaton read a word starting from a given state.

The *behaviour* of an automaton is referred to as its *language*, defined below. In the DA setting above this instantiates to the actual language—the classification of words—accepted by a DA. In general, it is a classification of $F^\star I$ into the outputs O, which are $1 \times A^\star$ and 2 respectively in the DA case.

Definition 2.2.12 (Language)**.** A *language* is a morphism $F^\star I \to O$.

The object $F^\star I$ serves as a generalisation of the set of words, which in the DFA case can be used to characterise from which inputs to reach states of the DFA. Before we can define the language accepted by an automaton, we need to generalise this concept of reachability.

Definition 2.2.13 (Reachability map)**.** Given an automaton $\mathcal{A} = (Q, \delta, i, o)$, its *reachability map* is given by

$$\mathsf{reach}_{\mathcal{A}} = i^\sharp : F^\star I \to Q.$$

Given an automaton $\mathcal{A} = (Q, \delta, i, o)$, the reachability map $\mathsf{reach}_{\mathcal{A}}$ is the unique F-algebra homomorphism $(F^\star I, \theta_I) \to (Q, \delta)$ satisfying $\mathsf{reach}_{\mathcal{A}} \circ \eta_I = i$. In fact, this makes it the unique automaton homomorphism $(F^\star I, \theta_I, \eta_I, \mathcal{L}_{\mathcal{A}}) \to \mathcal{A}$.

Using the reachability map, we obtain a language for an automaton by composing its output map. This instantiates to familiar notions of behaviour, such as the language accepted by a DFA. One can compute that language by taking for each word the acceptance decision associated with the state reached after reading that word starting from the initial state.

Definition 2.2.14 (Language accepted by an automaton)**.** The *language accepted by an automaton* $\mathcal{A} = (Q, \delta, i, o)$ is given by

$$\mathcal{L}_{\mathcal{A}} = F^\star I \xrightarrow{\mathsf{reach}_{\mathcal{A}}} Q \xrightarrow{o} O.$$

The existence of an automaton homomorphism between two automata witnesses their languages being equivalent. That is, if $\mathcal{A}$ and $\mathcal{A}'$ are automata and $h : \mathcal{A} \to \mathcal{A}'$ is any automaton homomorphism, then $\mathcal{L}_{\mathcal{A}} = \mathcal{L}_{\mathcal{A}'}$.

Since the reachability map generalises the function that assigns to a word the state reached in a DFA, we can also generalise the property of a DFA to be *reachable:* for each state there is a word reaching it. This is the case if the reachability map is surjective, which we generalise using the $\mathcal{E}$ part of the factorisation system.

Definition 2.2.15 (Reachability)**.** We say that an automaton $\mathcal{A}$ is *reachable* if $\mathsf{reach}_{\mathcal{A}} \in \mathcal{E}$.

Reachability is one step towards minimality, which in the DFA case also requires distinct states to accept different languages. This turns out to be equivalent to being final among all reachable automata accepting the same language.

Definition 2.2.16 (Minimality)**.** An automaton $\mathcal{A}$ is *minimal* if it is reachable and for each reachable automaton $\mathcal{A}'$ with $\mathcal{L}_{\mathcal{A}'} = \mathcal{L}_{\mathcal{A}}$ there exists a unique automaton homomorphism $\mathcal{A}' \to \mathcal{A}$.

Example 2.2.17. Consider the DA setting from Example 2.2.11. A DA $\mathcal{A} = (Q, \delta, i, o)$ is reachable if and only if $\mathsf{reach}_{\mathcal{A}} : A^\star \to Q$ is surjective. That is, for each $q \in Q$ there needs to exist $u \in A^\star$ with $\mathsf{reach}_{\mathcal{A}}(u) = q$. Suppose $\mathcal{A}$ is reachable and consider the state-minimal DA $\mathcal{A}_m$ satisfying $\mathcal{L}_{\mathcal{A}_m} = \mathcal{L}_{\mathcal{A}}$. We know that $\mathcal{A}_m$ is reachable and that each of its states accepts a different language. This DA is in fact minimal, with the unique homomorphism $h : \mathcal{A} \to \mathcal{A}_m$ defined by $h(\mathsf{reach}_{\mathcal{A}}(u)) = \mathsf{reach}_{\mathcal{A}_m}(u)$. It takes a state $\mathsf{reach}_{\mathcal{A}}(u)$ and maps it to the one state in $\mathcal{A}_m$ accepting the same language. One can show that this state is $\mathsf{reach}_{\mathcal{A}_m}(u)$ using that the reachability maps are automaton homomorphisms.

2.3 The $\mathtt{L}^\star$ Algorithm

The $\mathtt{L}^\star$ algorithm [Ang87] learns the minimal DFA accepting a language $\mathcal{L} \subseteq A^\star$ over a finite alphabet A. It assumes the existence of a *teacher*, which is an oracle that can answer two types of queries:

- *Membership queries:* given a word $w \in A^\star$, the teacher replies with 0 or 1 according to whether w belongs to $\mathcal{L}$.
- *Equivalence queries:* given a *hypothesis* DFA $\mathcal{H}$, the teacher replies **yes** if $\mathcal{L}_\mathcal{H}$ equals $\mathcal{L}$. If not, the teacher returns a *counterexample*: a word $w \in A^\star$ incorrectly classified by $\mathcal{H}$ (i.e., $w \in \mathcal{L} \iff w \notin \mathcal{L}_\mathcal{H}$).

In practice, membership queries are often easily implemented by interacting with the system one wants to model the behaviour of. Equivalence queries are more complicated—as the target automaton is not known they are commonly approximated by testing. Such testing can however be done exhaustively if a bound on the number of states of the target automaton is known [Cho78; Vas73]. Equivalence queries can also be implemented exactly when learning algorithms are being compared experimentally on a generated automaton whose language forms the target. In this case, standard methods for language equivalence, such as building a bisimulation, can be used.

The learning algorithm incrementally builds an *observation table* made up of two parts: a top part, with rows ranging over a finite set $S \subseteq A^\star$; and a bottom part, with rows ranging over $S \cdot A$ (where $\cdot$ is pointwise concatenation). Columns range over a finite set $E \subseteq A^\star$. For each $u \in S \cup S \cdot A$ and $v \in E$, the corresponding cell in the table contains 1 if and only if $uv \in \mathcal{L}$ and this can be determined using a membership query. Intuitively, each row u approximates the Myhill–Nerode equivalence class of u with respect to the target language—rows with the same content are considered members of the same equivalence class. Recall that the Myhill–Nerode right congruence $\equiv_\mathcal{L}$ of $\mathcal{L}$ is defined for all words $u_1, u_2 \in A^\star$ by

$$u_1 \equiv_\mathcal{L} u_2 \iff \forall v \in A^\star.\, \mathcal{L}(u_1 v) = \mathcal{L}(u_2 v).$$

For $S = E = A^\star$, the relation that identifies words of S having the same rows is precisely the Myhill–Nerode right congruence.

As an example, and to set notation, consider the table below over $A = \{a, b\}$. It shows that the language $\mathcal{L}$ it is based on contains the word aa and does not contain the words ε (the empty word), a, b, ba, aaa, and baa.

		E: ε	a	aa
S	ε	0	0	1
$S \cdot A$	a	0	1	0
	b	0	0	0

$$\mathsf{row} : S \longrightarrow 2^E$$
$$\mathsf{row}(u)(v) = 1 \iff uv \in \mathcal{L}$$
$$\mathsf{srow} : S \cdot A \longrightarrow 2^E$$
$$\mathsf{srow}(ua)(v) = 1 \iff uav \in \mathcal{L}$$

We use functions row and srow to describe the top and bottom (successor) parts of the table, respectively. Notice that S and $S \cdot A$ may intersect. For conciseness, when tables are depicted, elements in the intersection are only shown in the top part.

A key idea of the algorithm is to construct a hypothesis DFA from the rows in the table, with rows having equal content being identified. That is, the state space of the hypothesis is given by the set $H = \{\mathsf{row}(s) \mid s \in S\}$. The construction is analogous to that of the minimal DFA from the Myhill-Nerode equivalence. The initial state is $\mathsf{row}(\varepsilon)$, and we use the ε column to determine whether a state is accepting: $\mathsf{row}(s)$ is accepting whenever $\mathsf{row}(s)(\varepsilon) = 1$. A row $\mathsf{row}(s)$ advances on a transition with $a \in A$ to the state given by $\mathsf{srow}(sa)$. (Notice that the continuation is drawn from the bottom part of the table). For this hypothesis automaton to be well-defined, ε must be in S and E, and the table must satisfy two properties:

- **Closedness** states that each transition actually leads to a state of the hypothesis. That is, the table is closed if for all $t \in S$ and $a \in A$ there is $s \in S$ such that $\mathsf{row}(s) = \mathsf{srow}(ta)$.

- **Consistency** states that there is no ambiguity in determining the transitions. That is, the table is consistent if for all $s_1, s_2 \in S$ such that $\mathsf{row}(s_1) = \mathsf{row}(s_2)$ we have $\mathsf{srow}(s_1 a) = \mathsf{srow}(s_2 a)$ for all $a \in A$.

The algorithm updates the sets S and E to satisfy these properties, constructs a hypothesis, submits it in an equivalence query, and, when given a counterexample, refines the hypothesis. This process continues until the hypothesis is correct.

More concretely, we show L* in Figure 2.1. It is organised into two procedures: Algorithm 2.1 makes a table closed and consistent, and Algorithm 2.2 performs the learning iterations. The latter works as follows: Initially the table corresponding to $S = E = \{\varepsilon\}$ is made closed and consistent (line 1). Then as long as the corresponding hypothesis, denoted by $\mathcal{H}_{(S,E)}$, is shown to be incorrect via an equivalence query, denoted by EQ, resulting in a counterexample $c \in A^\star$ (line 2), the prefixes of c are added to S (line 3) and the table is made

Algorithm 2.1. Make table closed and consistent

1: **function** Fix(S, E)
2: **while** (S, E) is not closed or not consistent **do**
3: **if** (S, E) is not closed **then**
4: find $s \in S$, $a \in A$ such that $\forall t \in S.\ \mathsf{srow}(sa) \neq \mathsf{row}(t)$
5: $S \longleftarrow S \cup \{sa\}$
6: **else if** (S, E) is not consistent **then**
7: find $s_1, s_2 \in S$, $a \in A$ and $e \in E$ such that
$\mathsf{row}(s_1) = \mathsf{row}(s_2)$ and $\mathsf{srow}(s_1 a)(e) \neq \mathsf{srow}(s_2 a)(e)$
8: $E \longleftarrow E \cup \{ae\}$
9: **return** S, E

Algorithm 2.2. L* algorithm

1: $S, E \longleftarrow$ Fix$(\{\varepsilon\}, \{\varepsilon\})$
2: **while** $\mathsf{EQ}(\mathcal{H}_{(S,E)}) = c \in A^\star$ **do**
3: $S \longleftarrow S \cup \mathsf{prefixes}(c)$
4: $S, E \longleftarrow$ Fix(S, E)
5: **return** $\mathcal{H}_{(S,E)}$

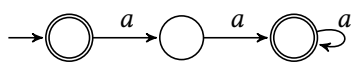
Figure 2.1: Angluin's L* algorithm

closed and consistent again (line 4). Once an equivalence query results in a positive answer, the hypothesis is returned (line 5).

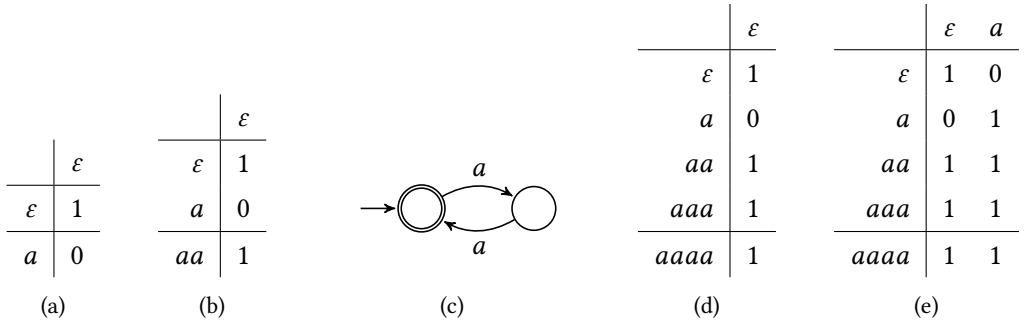
Example Run. We run the algorithm with the target language

$$\mathcal{L} = \{w \in \{a\}^\star \mid |w| \neq 1\},$$

of which the minimal DFA accepting it is shown below.

$\mathcal{A} =$ 

Initially, $S = E = \{\varepsilon\}$. We build the observation table given in Figure 2.2a. This table is not closed, because the row with label a, having 0 in the only column, does not appear in the top



	ε
ε	1
a	0

(a)

	ε
ε	1
a	0
aa	1

(b)

(c)

	ε
ε	1
a	0
aa	1
aaa	1
$aaaa$	1

(d)

	ε	a
ε	1	0
a	0	1
aa	1	1
aaa	1	1
$aaaa$	1	1

(e)

Figure 2.2: Example run of $\mathrm{L}^\star$ on $\mathcal{L} = \{w \in \{a\}^\star \mid |w| \neq 1\}$.

part of the table: the only row ε has 1. To fix this, we add the word a to the set S. Now the table (Figure 2.2b) is closed and consistent, so we construct the hypothesis that is shown in Figure 2.2c and pose an equivalence query. The teacher replies *no* and informs us that the word aaa should have been accepted. To process this counterexample, we add all its prefixes to the set S. We only have to add aa and aaa in this case. The next table (Figure 2.2d) is closed, but not consistent: the rows ε and aa both have value 1, but their extensions a and aaa differ. To fix this, we prepend the continuation a to the column ε on which they differ and add $a \cdot \varepsilon = a$ to E. This distinguishes $\mathsf{row}(\varepsilon)$ from $\mathsf{row}(aa)$, as seen in the next table in Figure 2.2e. The table is now closed and consistent, and the new hypothesis automaton is precisely $\mathcal{A}$.

Variations. Several variations on the original algorithm exist. For instance, Maler and Pnueli [MP95] proposed to add all suffixes of a counterexample to E instead of its prefixes to S (line 3 in Algorithm 2.2). Since this variation maintains the invariant that for any two $s_1, s_2 \in S$ with $s_1 \neq s_2$ we have $\mathsf{row}(s_1) \neq \mathsf{row}(s_2)$, consistency is always trivially satisfied and thus does not have to be checked. We give the resulting modified algorithm, referred to as $\mathrm{L}^{\star\,\mathrm{MP}}$, in Figure 2.3. An example is given below. Another variation on handling counterexamples is given by Rivest and Schapire [RS93], who add only a single suffix of a counterexample to E. The purpose of adding these columns is to distinguish two rows that were previously equal, so finding a single new column that does this avoids a large amount of unnecessary data in the table, which would potentially require expensive membership queries to obtain. We will discuss this variation in detail in Section 6.5.

Example Run of $\mathrm{L}^{\star\,\mathrm{MP}}$. We revisit the earlier example with target language

$$\mathcal{L} = \{w \in \{a\}^\star \mid |w| \neq 1\},$$

Algorithm 2.3. Make table closed

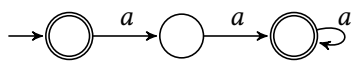
```
1: function Fix(S, E)
2:     while (S, E) is not closed do
3:         find s ∈ S, a ∈ A such that ∀t ∈ S. srow(sa) ≠ row(t)
4:         S ← S ∪ {sa}
5:     return S, E
```

Algorithm 2.4. $\mathrm{L}^{\star\,\mathrm{MP}}$ algorithm

```
1: S, E ← Fix({ε}, {ε})
2: while EQ(H_(S,E)) = c ∈ A* do
3:     S ← S ∪ suffixes(c)
4:     S, E ← Fix(S, E)
5: return H_(S,E)
```

Figure 2.3: Maler and Pnueli's variation on $\mathrm{L}^\star$

of which the minimal DFA is given below.

$\mathcal{A} =$ 

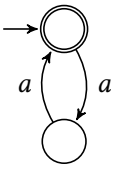
This time we apply $\mathrm{L}^{\star\,\mathrm{MP}}$ instead of $\mathrm{L}^\star$. Again, initially $S = E = \{\varepsilon\}$. As in the original example, the table in Figure 2.4a is made closed in Figure 2.4b and results in the hypothesis Figure 2.4c, an equivalence query for which leads to the teacher giving the counterexample *aaa*. Now we add all the suffixes of the counterexample to the set E, which means we add *a*, *aa*, and *aaa*. The next table (Figure 2.4d) is not closed: the lower row *aa* does not appear in the upper part. Thus, we add *aa* to S to obtain the closed (and consistent) table in Figure 2.4e. The new hypothesis automaton is precisely $\mathcal{A}$.

	ε
ε	1
a	0

(a)

	ε
ε	1
a	0
aa	1

(b)



(c)

	ε	a	aa	aaa
ε	1	0	1	1
a	0	1	1	1
aa	1	1	1	1

(d)

	ε	a	aa	aaa
ε	1	0	1	1
a	0	1	1	1
aa	1	1	1	1
aaa	1	1	1	1

(e)

Figure 2.4: Example run of $\mathrm{L}^{\star\,\mathtt{MP}}$ on $\mathcal{L} = \{w \in \{a\}^\star \mid |w| \neq 1\}$.

Chapter 3

Minimisation of Automata

Automata have been extensively studied using category theory, both from an algebraic and a coalgebraic perspective [AM74; Hol82; AT89; Rut98]. Categorical insights have enabled the development of generic algorithms for minimisation [Adá+12], determinisation [Sil+13], and equivalence checking [BP15].

A fruitful line of work has focused on characterising the semantics of different types of automata as final coalgebras. The final coalgebra contains unique representatives of behaviour, and the existence of a minimal automaton can be formalised by a suitable factorisation of the map from a given automaton into the final coalgebra. Algorithms to compute the minimal automaton can be devised based on the final sequence, which yields procedures resembling classical partition refinement [KK14; Dor+17]. Unfortunately, *bottom-up tree automata*, which generalise DFAs by processing trees instead of words, do not fit the abstract framework of final coalgebras.[1] This impeded the application of abstract algorithms for minimisation, determinisation, and equivalence. We embrace the categorical *algebraic* view on automata due to Arbib and Manes [AM74], introduced in Section 2.2.4, to study bottom-up tree automata (Section 3.1). This algebraic approach is also treated in detail by Adámek and Trnková [AT89], who, among other results, give conditions under which minimal realisations exist (see also [Adá77]). However, generic *algorithms* for minimisation have not been studied in this context.

The contributions of this chapter are as follows.

1. First, we explore the notion of *cobase* to devise an iterative construction for minimising tree automata, at the abstract level of algebras, resembling partition refinement (Sec-

[1] The language semantics of top-down tree automata represented as coalgebras is given in [KR16], based on a transformation to bottom-up tree automata. In this chapter, we focus on bottom-up automata only.

tion 3.3). The notion of cobase is dual to that of base [Blo12], which plays a key role in reachability of coalgebras [Wiß+19; BKR19] and therefore in minimisation of automata.

2. Second, we study a different characterisation of minimality via the Nerode equivalence, again based on work of Arbib and Manes [AM74], and provide a generalisation using monads that allows to treat automata with equations (Section 3.4).

Before presenting our contributions, we recall bottom-up tree automata in Section 3.1, after which we review and relate different notions of abstract automaton minimality in Section 3.2.

Throughout this chapter we work in an arbitrary category $\mathbf{C}$ with a factorisation system $(\mathcal{E}, \mathcal{M})$ and fix a functor $F : \mathbf{C} \to \mathbf{C}$ and objects I and O in $\mathbf{C}$. We assume that F maps morphisms in $\mathcal{E}$ to epimorphisms and admits an algebraically free monad $(F^\star, \eta, \mu)$. We also assume that $\mathbf{C}$ is cocomplete (has all small colimits) and *cowellpowered*, which will be explained in Section 3.3.

3.1 Bottom-Up Tree Automata

In this section we show how automata as defined in Section 2.2.4 can capture (deterministic) bottom-up tree automata. We first recall some basic concepts.

A *ranked alphabet* is a finite set of symbols Γ, where each $\gamma \in \Gamma$ is equipped with an *arity* $\mathsf{arity}(\gamma) \in \mathbb{N}$. The set of Γ-*trees* over a set of symbols I, denoted $\mathcal{T}_\Gamma(I)$, is the smallest set such that $I \subseteq \mathcal{T}_\Gamma(I)$, and for all $\gamma \in \Gamma$ we have that $t_1, \dots, t_{\mathsf{arity}(\gamma)} \in \mathcal{T}_\Gamma(I)$ implies $(\gamma, t_1, \dots, t_{\mathsf{arity}(\gamma)}) \in \mathcal{T}_\Gamma(I)$. In other words, $\mathcal{T}_\Gamma(I)$ consists of finite trees with leaves labelled by symbols from I and internal nodes labelled by symbols from Γ; the number of children of each internal node matches the arity of its label.

A ranked alphabet Γ gives rise to a polynomial *signature endofunctor* $\Sigma_\Gamma : \mathbf{Set} \to \mathbf{Set}$ given by $\Sigma_\Gamma X = \coprod_{\gamma \in \Gamma} X^{\mathsf{arity}(\gamma)}$. A *bottom-up tree automaton* is an automaton $\mathcal{A} = (Q, \delta, i, o)$ where Q is finite, the dynamics functor F is a signature endofunctor Σ_Γ, and the output set $O = 2$. Here Q is the set of *states*, $i : I \to Q$ is the *initial assignment*, $o : Q \to 2$ is the characteristic function of *final* states, and for each $\gamma \in \Gamma$ we have a transition function $\delta_\gamma = \delta \circ \kappa_\gamma : Q^{\mathsf{arity}(\gamma)} \to Q$, where $\kappa_\gamma : Q^{\mathsf{arity}(\gamma)} \to \Sigma_\Gamma Q$ is the coproduct injection.

The *language* $\mathcal{L}_\mathcal{A}$ of a bottom-up tree automaton is the set of all Γ-trees t such that $(o \circ \hat{\delta})(t) = 1$, where $\hat{\delta} : \mathcal{T}_\Gamma(I) \to Q$ extends δ to trees by structural recursion:

$$\hat{\delta}(\ell) = i(\ell) \quad (\ell \in I) \qquad\qquad \hat{\delta}(\gamma, t_1, \dots, t_k) = \delta_\gamma(\hat{\delta}(t_1), \dots, \hat{\delta}(t_k))$$

In other words, $\mathcal{L}_{\mathcal{A}}$ contains the trees that evaluate to a final state. The map $\hat{\delta}$ above is the transpose $i^\sharp$ in the adjunction between **Set** and the category of Σ_Γ-algebras, where the left adjoint sends a set I to the Σ_Γ-algebra with carrier $\mathcal{T}_\Gamma(I)$ and the obvious structure map.

Example 3.1.1. As an example, let the ranked alphabet be given by $\Gamma = \{\bullet\}$ with $\mathsf{arity}(\bullet) = 2$ and take $I = \{a, b\}$. We consider the the bottom-up tree automaton (Q, δ, i, o), where $Q = \{q_a, q_{a+}, q_b, q_{b+}, q_\bot, q_\top\}$, $i : I \longrightarrow Q$ is given by $i(a) = q_a$ and $i(b) = q_b$, and $o : Q \longrightarrow 2$ is given by

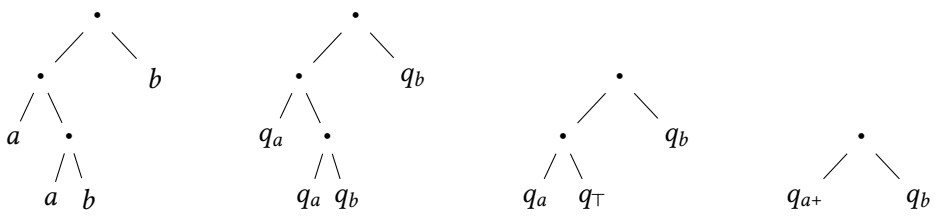
$$o(q_a) = o(q_{a+}) = o(q_b) = o(q_{b+}) = o(q_\bot) = 0 \qquad\qquad o(q_\top) = 1.$$

That is, $q_\top$ is the only accepting state. Furthermore, we define $\delta_\bullet : Q^2 \longrightarrow Q$ via the following assignments.

$$(q_a, q_b) \mapsto q_\top \quad (q_a, q_{b+}) \mapsto q_\top \quad (q_a, q_\top) \mapsto q_{a+} \quad (q_{a+}, q_b) \mapsto q_\top \quad (q_\top, q_b) \mapsto q_{b+}$$

All remaining pairs are assigned $q_\bot$. The language accepted by this automaton consists of binary trees over $\{a, b\}$ where the yield of the tree—its sequence of leaf symbols from left to right—is a word of the form $a^n b^n$ for some $0 < n \in \mathbb{N}$ and every binary node in the tree has one leaf child.

For instance, consider the tree below on the left.



In order to apply $i^\sharp = \hat{\delta} : \mathcal{T}_\Gamma(I) \longrightarrow Q$, we first apply $i : I \longrightarrow Q$ to the leaves of the tree to obtain the middle left tree. We now apply binary transitions from bottom to top. First, this involves replacing the bottom pair (q_a, q_b) with $q_\top$, resulting in the middle right tree. Now we replace $(q_a, q_\top)$ with q_{a+}, as shown in the tree on the right. Finally, the pair (q_{a+}, q_b) becomes $q_\top$. Since $q_\top$ is the accepting state, the tree is in the language.

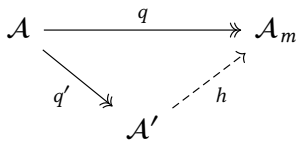
In the next section we explore different notions of minimality on the abstract level of automata as defined in Section 2.2.4. This therefore applies also to the bottom-up tree automata introduced above.

3.2 Notions of Minimality

In Section 2.2.4 we defined an automaton to be minimal if it is reachable and if each reachable automaton accepting the same language has a unique automaton homomorphism into it. In this section we define two related notions that do not refer to reachability: *minimisation* and *simplicity*. Minimisation is a notion relative to an automaton that characterises its "smallest quotient"; simplicity states that an automaton has only trivial quotients.

Below we define the minimisation of an automaton as the limit among its *quotient automata*. Given an automaton $\mathcal{A}$, we refer to an automaton $\mathcal{A}'$ as a quotient automaton of $\mathcal{A}$ if it comes with a quotient $q : \mathcal{A} \twoheadrightarrow \mathcal{A}'$. That is, if Q and Q' are the respective state spaces of $\mathcal{A}$ and $\mathcal{A}'$, then q is a morphism $Q \twoheadrightarrow Q' \in \mathcal{E}$ (see also Section 2.2.1) that forms an automaton homomorphism.

Definition 3.2.1 (Minimisation)**.** The *minimisation* of an automaton $\mathcal{A}$ is a quotient automaton $\mathcal{A}_m$, $q : \mathcal{A} \twoheadrightarrow \mathcal{A}_m$, such that for any quotient automaton $\mathcal{A}'$, $q' : \mathcal{A} \twoheadrightarrow \mathcal{A}'$ of $\mathcal{A}$ there exists a (necessarily unique) automaton homomorphism $h : \mathcal{A}' \twoheadrightarrow \mathcal{A}_m$ such that $h \circ q' = q$.



Note that the morphism h in the definition of minimisation is in $\mathcal{E}$ because q' and q are. Minimisation is called *minimal reduction* in [AT89], but note that there initial state maps are not taken into account.

We relate the existence of minimisations to the existence of minimal automata with Proposition 3.2.3 below, for which we need a simple lemma.

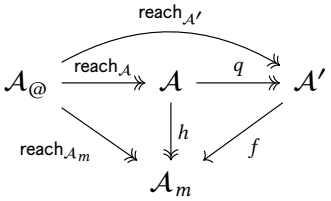
Lemma 3.2.2. *An automaton $\mathcal{A}$ is minimal if and only if it is the minimisation of $(F^\star I, \theta_I, \eta_I, \mathcal{L}_\mathcal{A})$.*

Proof. Consider an automaton $\mathcal{A}$, and let $\mathcal{A}_@ = (F^\star I, \theta_I, \eta_I, \mathcal{L}_\mathcal{A})$. Since any automaton homomorphism $\mathcal{A}_@ \to \mathcal{A}$ must be the reachability map of $\mathcal{A}$, $\mathcal{A}$ is a quotient automaton of $\mathcal{A}_@$ if and only if it is reachable. The automaton $\mathcal{A}$ is the minimisation of $\mathcal{A}_@$ (necessarily via $\mathsf{reach}_\mathcal{A}$) if and only if for any quotient automaton $\mathcal{A}'$, $q : \mathcal{A}_@ \twoheadrightarrow \mathcal{A}'$, with $\mathcal{L}_{\mathcal{A}'} = \mathcal{L}_{\mathcal{A}_@}$ there exists an automaton homomorphism $h : \mathcal{A}' \to \mathcal{A}$ satisfying $h \circ q = \mathsf{reach}_\mathcal{A}$. By the uniqueness of reachability maps we have $q = \mathsf{reach}_{\mathcal{A}'}$ and the above equality is automatically satisfied. Thus, $\mathcal{A}$ is the minimisation of $\mathcal{A}_@$ if and only if for every reachable automaton $\mathcal{A}'$

with $\mathcal{L}_{\mathcal{A}'} = \mathcal{L}_{\mathcal{A}_@}$ there exists an automaton homomorphism $h : \mathcal{A}' \longrightarrow \mathcal{A}$, which is exactly what it means for $\mathcal{A}$ to be minimal. □

Proposition 3.2.3. *There exists a minimisation for every reachable automaton if and only if a minimal automaton exists for every language. In that case, if an automaton $\mathcal{A}$ is reachable, then the minimisation of $\mathcal{A}$ is minimal.*

Proof. For the equivalence, the implication from left to right follows from Lemma 3.2.2. For the converse, consider a reachable automaton $\mathcal{A}$ and let $\mathcal{A}_m$ be the minimal automaton accepting $\mathcal{L}_{\mathcal{A}}$. Since $\mathcal{A}$ and $\mathcal{A}_m$ are reachable and accept the same language, there is by minimality of $\mathcal{A}_m$ a unique homomorphism $h : \mathcal{A} \longrightarrow \mathcal{A}_m$ satisfying $h \circ \mathsf{reach}_{\mathcal{A}} = \mathsf{reach}_{\mathcal{A}_m}$, and we know $h \in \mathcal{E}$. Suppose $\mathcal{A}'$, $q : \mathcal{A} \longrightarrow \mathcal{A}'$ is any quotient automaton of $\mathcal{A}$. Then $q \circ \mathsf{reach}_{\mathcal{A}} = \mathsf{reach}_{\mathcal{A}'}$ by the uniqueness of reachability maps, and $\mathcal{L}_{\mathcal{A}'} = \mathcal{L}_{\mathcal{A}}$ by the existence of the automaton homomorphism q. By minimality of $\mathcal{A}_m$ there then exists a unique automaton homomorphism $f : \mathcal{A}' \longrightarrow \mathcal{A}_m$. We thus have the following situation, where we let $\mathcal{A}_@ = (F^\star I, \theta_I, \eta_I, \mathcal{L}_{\mathcal{A}})$.



Since $f \circ q \circ \mathsf{reach}_{\mathcal{A}} = \mathsf{reach}_{\mathcal{A}_m}$ by the uniqueness of reachability maps, we have by the uniqueness property of h that $f \circ q = h$. We conclude that $\mathcal{A}_m$ is the minimisation of $\mathcal{A}$ via h.

For the second statement, consider a reachable automaton $\mathcal{A}$ and let $\mathcal{A}'$, $q : \mathcal{A} \twoheadrightarrow \mathcal{A}'$ be its minimisation. Then $\mathsf{reach}_{\mathcal{A}'} = q \circ \mathsf{reach}_{\mathcal{A}} \in \mathcal{E}$ by the uniqueness of reachability maps, so $\mathcal{A}'$ is reachable. We also have $\mathcal{L}_{\mathcal{A}_m} = \mathcal{L}_{\mathcal{A}}$ by the existence of the automaton homomorphism q. Let $\mathcal{A}_m$ be the minimal automaton accepting $\mathcal{L}_{\mathcal{A}}$. Then there exists a unique homomorphism $f : \mathcal{A}' \twoheadrightarrow \mathcal{A}_m$. This makes $\mathcal{A}_m$ a quotient automaton of $\mathcal{A}$ via $f \circ q : \mathcal{A} \longrightarrow \mathcal{A}_m$. By $\mathcal{A}'$ being the minimisation of $\mathcal{A}$ there exists a unique homomorphism $g : \mathcal{A}_m \longrightarrow \mathcal{A}'$ satisfying $g \circ f \circ q = q$. Since q is an epi we have $g \circ f = \mathsf{id}$ and using that $f \circ g \circ f = f$ and f is an epi we have $f \circ g = \mathsf{id}$. We conclude that $\mathcal{A}'$ is minimal, being isomorphic to $\mathcal{A}_m$. □

The definition of minimality relies on reachability. It is also interesting to ask whether there is another property that, together with reachability, implies minimality, but is not itself dependent on reachability [AM75a]. Here we propose precisely such a condition.

Definition 3.2.4 (Simplicity)**.** An automaton $\mathcal{A}$ is called *simple* if for every quotient automaton $\mathcal{A}'$ the associated quotient $q : \mathcal{A} \twoheadrightarrow \mathcal{A}'$ is an isomorphism.

We note that the notion of minimisation is also independent of reachability. Indeed, we can alternatively characterise simplicity via minimisation, as we show next.

Proposition 3.2.5. *An automaton $\mathcal{A}$ is simple if and only if it is its own minimisation.*

Proof. First suppose $\mathcal{A}$ is simple. We will show that it is its own minimisation via the identity morphism. Thus, suppose $\mathcal{A}'$, $q : \mathcal{A} \twoheadrightarrow \mathcal{A}'$ is any quotient automaton of $\mathcal{A}$. By simplicity we have that q is an isomorphism. Then $q^{-1} : \mathcal{A}' \longrightarrow \mathcal{A}$ satisfies $q^{-1} \circ q = \mathsf{id}$. For uniqueness, if $f : \mathcal{A}' \longrightarrow \mathcal{A}$ is any automaton homomorphism satisfying $f \circ q = \mathsf{id}$, then $f = q^{-1}$. We conclude that $\mathcal{A}$ is its own minimisation.

Now suppose $\mathcal{A}$ is its own minimisation via the identity morphism. To see that $\mathcal{A}$ is simple, consider any quotient automaton $\mathcal{A}'$, $q : \mathcal{A} \twoheadrightarrow \mathcal{A}'$. We need to show that q is an isomorphism. By $\mathcal{A}$ being its own minimisation, there exists a unique homomorphism $f : \mathcal{A}' \longrightarrow \mathcal{A}$ satisfying $f \circ q = \mathsf{id}$. Then $q \circ f \circ q = q$, so by q being an epi we have $q \circ f = \mathsf{id}$ and conclude that q is an isomorphism. □

In Corollary 3.2.7 we will show an equivalence between minimality and the combination of reachability and simplicity. To this end we first establish a further connection between simplicity and minimisation. The result below asserts that any quotient of an automaton is simple if and only if the quotient is the minimisation of the automaton. It can be seen as a refinement (and dual) of [BKR19, Theorem 17], computing the reachable part of a coalgebra.

Proposition 3.2.6. *Let $\mathcal{A} = (Q, \delta, i, o)$ be an automaton that has a minimisation, and let $\mathcal{A}'$, $q : \mathcal{A} \twoheadrightarrow \mathcal{A}'$ be a quotient automaton. Then $\mathcal{A}'$ is simple if and only if it is the minimisation of $\mathcal{A}$.*

Proof. We denote the minimisation of $\mathcal{A}$ by $\mathcal{A}_m$, $q_m : \mathcal{A} \twoheadrightarrow \mathcal{A}_m$. Suppose $\mathcal{A}'$ is simple. Since $\mathcal{A}_m$ is the minimisation of $\mathcal{A}$ and $\mathcal{A}'$ is a quotient automaton of $\mathcal{A}$, there exists a homomorphism of automata $h : \mathcal{A}' \twoheadrightarrow \mathcal{A}_m$. Since $\mathcal{A}'$ is simple, this homomorphism is an iso.

Conversely, suppose $\mathcal{A}'$ is the minimisation of $\mathcal{A}$ and consider any quotient automaton $\mathcal{A}''$ of $\mathcal{A}'$, witnessed by some $q' : \mathcal{A}' \twoheadrightarrow \mathcal{A}''$. Then $\mathcal{A}''$ is also a quotient automaton of $\mathcal{A}$, via $q' \circ q$. Because $\mathcal{A}'$ is the minimisation of $\mathcal{A}$, there exists $k : \mathcal{A}'' \longrightarrow \mathcal{A}'$ such that $k \circ q' \circ q = q$. Thus $k \circ q' = \mathsf{id}$, using that q is an epi. Since $q' \circ k \circ q' = q'$ and q' is an epi as well, we also have $q' \circ k = \mathsf{id}$. Hence q' is an iso, as needed. □

The corollary below follows from the above together with Lemma 3.2.2, which identifies the minimal automaton as a particular minimisation.

Corollary 3.2.7. *Consider an automaton $\mathcal{A}$ such that the minimal automaton accepting $\mathcal{L}_{\mathcal{A}}$ exists. Then $\mathcal{A}$ is minimal if and only if it is reachable and simple.*

A crucial ingredient for the application of the above results is that the minimisation of a given automaton exists. In the next section we will explore conditions under which this is guaranteed. In fact, we will develop a procedure to find minimisations.

3.3 Minimisation via the Cobase

We will show how to compute the minimisation of a given automaton using the so-called *cobase* [Blo12]. This is the dual of the base, which is used in [BKR19; Wiß+19] for reachability of coalgebras. We will need some additional notions before we can define the cobase.

Given an object X, we denote by $\mathsf{Quot}(X)$ the class of all quotients of X (see Section 2.2.1). Recall that we assume $\mathbf{C}$ to be *cowellpowered*, which implies[2] that $\mathsf{Quot}(X)$ is a set. Since $\mathbf{C}$ is also assumed to be cocomplete (all small colimits exist), $\mathsf{Quot}(X)$ forms a complete lattice, with the order given by $\leq$ (see Section 2.2.1). Least upper bounds (joins) are given by *cointersections*, i.e., wide pushouts of quotients, under which $\mathcal{E}$ is closed [AHS09]. The cobase will allow us to characterise the minimisation of an automaton (Q, δ, i, o) as the greatest fixed point of a certain monotone operator on $\mathsf{Quot}(Q)$.

Definition 3.3.1 (Cobase)**.** Let $f : FX \to Y$ be a morphism. The *($\mathcal{E}$)-cobase* of f (if it exists) is the greatest quotient $q \in \mathsf{Quot}(X)$ such that there exists a morphism g with $g \circ Fq = f$.

The map g in the above definition is unique because we assume that F maps morphisms in $\mathcal{E}$ to epis. A concrete instance of the cobase will be given below in Example 3.3.5. The cobase can be computed as the join of all quotients satisfying the relevant condition, provided that the functor preserves cointersections. A functor $F : \mathbf{C} \to \mathbf{C}$ is said to *preserve $\mathcal{E}$-cointersections* if it preserves wide pushouts of epimorphisms in $\mathcal{E}$. In that case, for an epimorphism e, if $e \in \mathcal{E}$, then Fe is an epimorphism.

Theorem 3.3.2 (Existence of cobases)**.** *Suppose $F : \mathbf{C} \to \mathbf{C}$ preserves $\mathcal{E}$-cointersections. Then every map $f : FX \to Y$ has a cobase, given by the cointersection*

$$\bigvee \{ q \in \mathsf{Quot}(X) \mid \exists g.\, g \circ Fq = f \}.$$

[2] If $\mathcal{E}$ is the class of all epimorphisms, then $\mathbf{C}$ being cowellpowered is defined as $\mathsf{Quot}(X)$ being a set for every object X. Since $\mathcal{E}$ is a subclass of epimorphisms, $\mathsf{Quot}(X)$ is a set.

Proof. For $\mathcal{E}$ the class of all epis, the dual is shown in [BKR19; Wiß+19]. The proof goes through in the current, more general setting, using that $\mathcal{E}$ is closed under cointersections. As a proof sketch, one uses the fact that F preserves $\mathcal{E}$-cointersections to identify the cointersection

$$F \bigvee \{ q \in \mathsf{Quot}(X) \mid \exists g.\, g \circ Fq = f \} = \bigvee \{ Fq \mid q \in \mathsf{Quot}(X), \exists g.\, g \circ Fq = f \},$$

after which the universal property of the pushout can be applied to construct an appropriate map g witnessing the desired cobase property. □

Remark 3.3.3. Using the (surjective, injective) factorisation system, a **Set** functor preserves cointersections if and only if it is finitary [AT89]. In particular, this is the case for polynomial signature endofunctors (see Section 3.1).

We now define an operator on quotients of the state space of an automaton that characterises its minimisation and gives a way of computing it. To this end, given a F-algebra (Q, δ) and a quotient $q : Q \twoheadrightarrow Q' \in \mathsf{Quot}(Q)$, define the quotient $\Theta_\delta(q) : Q \twoheadrightarrow \Theta_\delta(Q')$ as the cobase of $q \circ \delta$, assuming it exists (for instance via Theorem 3.3.2). This defines a monotone operator $\Theta_\delta : \mathsf{Quot}(Q) \longrightarrow \mathsf{Quot}(Q)$ that has the following important property (see [BKR19; Wiß+19]):

Lemma 3.3.4. *Suppose F preserves $\mathcal{E}$-cointersections. For any F-algebra (Q, δ), a quotient $q : Q \twoheadrightarrow Q'$ in $\mathsf{Quot}(Q)$ satisfies $q \leq \Theta_\delta(q)$ if and only if there is an algebra structure $\delta' : FQ' \longrightarrow Q'$ turning q into an algebra homomorphism $(Q, \delta) \longrightarrow (Q', \delta')$.*

The operator Θ_δ allows us to quotient the transition structure of the automaton. In order to obtain the minimal automaton, we incorporate the output map $o : Q \longrightarrow O$ into the construction of a monotone operator based on Θ_δ. For technical convenience, we assume that this map is an element of $\mathsf{Quot}(Q)$.[3] The relevant monotone operator for minimisation is $\Theta_\delta \wedge o$ (where the meet $\wedge$ is taken pointwise in $\mathsf{Quot}(Q)$).

Example 3.3.5. Let $F : \mathbf{Set} \longrightarrow \mathbf{Set}$ be a polynomial functor induced by signature Γ. We first spell out what the cobase means concretely in this case and then study the operator Θ_δ in more detail. Since F is an endofunctor on **Set**, the cobase of a map $f : FX \longrightarrow Y$ is the largest quotient $q \in \mathsf{Quot}(X)$ such that for all $t, t' \in FX$:

$$\text{if } Fq(t) = Fq(t'), \text{ then } f(t) = f(t').$$

[3] This is not a real restriction: one can just pre-process the automaton by factorising o, i.e., keeping only those outputs actually occurring in the automaton.

This means that for every $\gamma \in \Gamma$ with $k = \mathsf{arity}(\gamma)$, and any $x_1, \dots, x_k, y_1, \dots y_k$, we have that

$$\frac{q(x_1) = q(y_1) \qquad \dots \qquad q(x_k) = q(y_k)}{f(\kappa_\gamma(x_1, \dots, x_k)) = f(\kappa_\gamma(y_1, \dots, y_k))}$$

or equivalently that for all $x_1, \dots, x_k$ and x_i' with $1 \leq i \leq k$ we have

$$\frac{q(x_i) = q(x_i')}{f(\kappa_\gamma(x_1, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_k)) = f(\kappa_\gamma(x_1, \dots, x_{i-1}, x_i', x_{i+1}, \dots, x_k))}$$

Suppose (Q, δ, i, o) is an automaton. For $q \in \mathsf{Quot}(Q)$, we have $q \leq \Theta_\delta(q) \wedge o$ if and only if

- for all $x, x' \in Q$: if $q(x) = q(x')$, then $o(x) = o(x')$; and
- for all $\gamma \in \Gamma$ with $k = \mathsf{arity}(\gamma)$, and $x_1, \dots, x_k$ and x_i' with $1 \leq i \leq k$ we have

$$\frac{q(x_i) = q(x_i')}{q(\delta_\gamma(x_1, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_k)) = q(\delta_\gamma(x_1, \dots, x_{i-1}, x_i', x_{i+1}, \dots, x_k))}$$

A partition q with the above two properties is known as a *forward bisimulation* [HMM09].

Theorem 3.3.6. *Suppose F preserves $\mathcal{E}$-cointersections. Let $\mathcal{A} = (Q, \delta, i, o)$ be an automaton, where $o \in \mathsf{Quot}(Q)$. Then $\mathsf{gfp}(\Theta_\delta \wedge o) \in \mathsf{Quot}(Q)$ is a quotient witnessing the minimisation of $\mathcal{A}$.*

Proof. Denote the quotient $\mathsf{gfp}(\Theta_\delta \wedge o)$ by $q_m : Q \twoheadrightarrow Q_m$. Thus $q_m \leq \Theta_\delta(q_m)$ and $q_m \leq o$, so (using Lemma 3.3.4) there exist δ_m, o_m turning q_m into an automaton homomorphism $\mathcal{A} \to (Q_m, \delta_m, q_m \circ i, o_m)$. We show that this is the minimisation of (Q, δ, i, o).

To this end, let $\mathcal{A}' = (Q', \delta', i', o')$, $q' : \mathcal{A} \twoheadrightarrow \mathcal{A}'$ be a quotient automaton of $\mathcal{A}$. By Lemma 3.3.4 we get $q' \leq \Theta_\delta(q')$, and since $o' \circ q' = o$ we have $q' \leq o$ and therefore $q' \leq \Theta_\delta(q') \wedge o$. Thus, $q' \leq \mathsf{gfp}(\Theta_\delta \wedge o)$, i.e., there is a quotient $h : Q' \twoheadrightarrow Q_m$ such that $h \circ q' = q_m$. It only remains to show that h is a homomorphism of automata. First, since $q' \in \mathcal{E}$ and F preserves $\mathcal{E}$-cointersections, Fq' is an epimorphism. Combined with the fact that q' and q_m are algebra homomorphisms and that $h \circ q' = q_m$, it easily follows that h is an algebra homomorphism. To see that it preserves the output, we have $o_m \circ h \circ q' = o_m \circ q_m = o = o' \circ q'$; hence, since q' is epic, we get $o_m \circ h = o'$. For preservation of the input, we have $h \circ i' = h \circ q' \circ i = q_m \circ i$, where the first step holds because q' is a homomorphism of automata. □

The above characterisation of minimisation of an automaton (Q, δ, i, o) gives us two ways of constructing it by standard lattice-theoretic computations. First, via the Knaster–Tarski theorem, we obtain it as the join of all post-fixed points of $\Theta_\delta \wedge o$, which, by Lemma 3.3.4,

amounts to the join of all quotient algebras respecting the output map o. That corresponds to the construction in [AT89]. Second, and perhaps most interestingly, we obtain the minimisation of (Q, δ, i, o) by iterating $\Theta_\delta \wedge o$, starting from the top element $\top$ of the lattice $\mathsf{Quot}(Q)$. The latter construction is analogous to the classical partition refinement algorithm: Starting from $\top$ corresponds to identifying all states as equivalent (or in other words, starting from the coarsest equivalence class of states). Every iteration step of $\Theta_\delta \wedge o$ splits the states that can be distinguished successively by just outputs, trees of depth 1, trees of depth 2, etc. If the state space is finite, this construction terminates, yielding the minimisation of the original automaton by Theorem 3.3.6.

3.4 Nerode Equivalence

We now show a generalised Nerode equivalence from which the minimal automaton can be constructed. Most of this section is based upon the work by Arbib and Manes [AM74], whose construction was further studied and refined by Anderson et al. [AAM76] and Adámek and Trnková [AT89]. We make a significant improvement in generality by phrasing the central equivalence definition (Definition 3.4.5) in terms of an arbitrary monad, which unlike the previous cited work allows applications to algebras satisfying a fixed set of equations. A monad generalisation of the Myhill–Nerode theorem appears in [Boj15], which confines itself to categories of sorted sets and characterises the quotient of the equivalence rather than the equivalence itself.

In this section we do not work with automata based on a dynamics functor F as in Definition 2.2.10. Instead, we note that such a functor induces the monad $F^\star$ and generalise by fixing an arbitrary monad (T, η, μ) in $\mathbf{C}$. Let $G \dashv U : \mathbf{C} \leftrightarrows \mathsf{EM}(T)$ be the adjunction with its category of (Eilenberg-Moore) algebras. Given a $\mathbf{C}$-morphism $f : X \longrightarrow UY$ for X in $\mathbf{C}$ and Y in $\mathsf{EM}(T)$, we write $f^\sharp : GX \longrightarrow Y$ for its adjoint transpose. We can then use a generalised notion of automaton.

Definition 3.4.1 (T-automaton)**.** A *T-automaton* is a tuple (Q, δ, i, o), where (Q, δ) is a T-algebra and $i : I \longrightarrow Q$ and $o : Q \longrightarrow O$ are morphisms in $\mathbf{C}$.[4] A *homomorphism* from (Q, δ, i, o) to (Q', δ', i', o') is a T-algebra homomorphism $h : (Q, \delta) \longrightarrow (Q', \delta')$ such that $h \circ i = i'$ and $o' \circ h = o$.

[4]The T-automata defined here differ from the more standard T-automata in Chapter 6 in that the algebraic structure on Q is the transition structure rather than additional structure on the state space that is preserved by the automaton structure. Furthermore, here we do not need to assume a T-algebra structure on O.

The reachability map of a T-automaton $\mathcal{A} = (Q, \delta, i, o)$ is given by $\mathsf{reach}_{\mathcal{A}} = U(i^\sharp) : TI \longrightarrow Q$ and is therefore the unique T-algebra homomorphism $(TI, \mu) \longrightarrow (Q, q)$ preserving initial states, taking $\eta_I : I \longrightarrow TI$ to be the initial state selector of TI. The language of $\mathcal{A}$ is given by $\mathcal{L}_{\mathcal{A}} = o \circ \mathsf{reach}_{\mathcal{A}} : TI \longrightarrow O$.

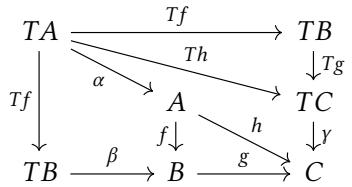
The automata defined in Section 2.2.4 are recovered using the following fact: the category of F-algebras is isomorphic to $\mathsf{EM}(T)$ for T the free F-algebra monad $F^\star$. In **Set** we may add equations to the signature, as any algebraic theory corresponds to a monad [Lan13, Chapter VI.8, Theorem 1]. For instance, if the signature contains just a binary operation that we require to be associative, commutative, and idempotent, then the algebras correspond to algebras in $\mathsf{EM}(T)$ for T the non-empty finite powerset monad.

Assumption 3.4.2. In this section we will need the class $\mathcal{E}$ to be the reflexive regular epis.[5]

The second lemma below, which needs the first, will be used in proving our main theorems.

Lemma 3.4.3. *If $f : A \longrightarrow B$ and $h : A \longrightarrow C$ in $\mathsf{EM}(T)$ are such that there exists $g : UB \longrightarrow UC$ in $\mathbf{C}$ with $g \circ f = h$ and Tf is an epi, then g is a T-algebra homomorphism $B \longrightarrow C$.*

Proof. Let $\alpha : TA \longrightarrow A$, $\beta : TB \longrightarrow B$, and $\gamma : TC \longrightarrow C$ be the respective T-algebra structures on A, B, and C. By commutativity of



and Tf being an epi, we directly conclude that g is a T-algebra homomorphism $B \longrightarrow C$. □

Lemma 3.4.4. *Suppose T maps reflexive coequalisers to epimorphisms. If $i : B \longrightarrow UC$ is such that $U(i^\sharp)$ reflexively coequalises $q_1, q_2 : A \longrightarrow TB$ in $\mathbf{C}$, then $i^\sharp$ reflexively coequalises $q_1^\sharp, q_2^\sharp : GA \longrightarrow GB$.*

Proof. For $k \in \{1, 2\}$ we have

$$U(i^\sharp \circ q_k^\sharp) = U(i^\sharp) \circ U(q_k^\sharp) = U((U(i^\sharp) \circ q_k)^\sharp),$$

[5] It is well known that (regular epi, mono) is a factorisation system in any regular category. We note that (reflexive regular epi, mono) forms a factorisation system in the same way. However, the theory in this section does not actually need a factorisation system; the instantiation of $\mathcal{E}$ is only invoked to obtain the right notion of reachability.

so by $U(i^\sharp)$ coequalising q_1 and q_2 we have $i^\sharp \circ q_1^\sharp = i^\sharp \circ q_2^\sharp$. If a T-algebra homomorphism $f : TB \longrightarrow Z$ is such that $f \circ q_1^\sharp = f \circ q_2^\sharp$, then

$$Uf \circ q_1 = Uf \circ U(q_1^\sharp) \circ \eta_A = Uf \circ U(q_2^\sharp) \circ \eta_A = Uf \circ q_2,$$

which because $U(i^\sharp)$ coequalises q_1 and q_2 yields a unique function $u : UC \longrightarrow UZ$ such that $u \circ U(i^\sharp) = f$. It remains to show that u is a T-algebra homomorphism. Note that since $U(i^\sharp)$ is a reflexive coequaliser, $TU(i^\sharp)$ is an epi by assumption on T. Precomposing u with $U(i^\sharp)$ yields the T-algebra homomorphism f, so by $TU(i^\sharp)$ being an epi and Lemma 3.4.3 we conclude u is a T-algebra homomorphism $C \longrightarrow Z$. Reflexivity of the pair will follow from Lemma 3.4.13. □

Before defining an abstract Nerode equivalence, we recall the classical definition for languages of words. Given a language $L : A^\star \longrightarrow 2$, the equivalence $R \subseteq A^\star \times A^\star$ is defined as

$$R = \{(u, v) \in A^\star \times A^\star \mid \forall w \in A^\star . L(uw) = L(vw)\}.$$

In this setting, $I = 1$ and $O = 2$. A function $Q \times A \longrightarrow Q$ corresponds to an algebra for the monad $T = (-) \times A^\star$, whose unit and multiplication are defined using the unit and multiplication of the monoid $A^\star$. If $p_1, p_2 : R \longrightarrow A^\star \cong 1 \times A^\star$ are the projections, we note that R is defined to be the largest relation making the following diagram commute.

$$\begin{array}{ccccc}
R \times A^\star & & \xrightarrow{p_2 \times \mathsf{id}} & & 1 \times A^\star \times A^\star \\
\downarrow{\scriptstyle p_1 \times \mathsf{id}} & & & & \downarrow{\scriptstyle \mu} \\
 & & & & 1 \times A^\star \\
 & & & & \downarrow{\scriptstyle L} \\
1 \times A^\star \times A^\star & \xrightarrow{\mu} & 1 \times A^\star & \xrightarrow{L} & 2
\end{array}$$

This leads to an abstract definition, using a limit[6] to generalise what it means to be maximal.

Definition 3.4.5 (Nerode equivalence)**.** Given a language $L : TI \longrightarrow O$ and an object R with morphisms $p_1, p_2 : R \longrightarrow TI$, we say that (R, p_1, p_2) is the *Nerode equivalence* of L if the diagram below on the left commutes and for all objects S with a reflexive pair $q_1, q_2 : S \longrightarrow TI$ such that the diagram in the middle commutes there is a unique morphism $u : S \longrightarrow R$ making the diagram on the right commute. In this case we say that L has a Nerode equivalence.

$$\begin{array}{ccccc}
TR & & \xrightarrow{Tp_2} & & TTI \\
\downarrow{\scriptstyle Tp_1} & & & & \downarrow{\scriptstyle \mu} \\
 & & & & TI \\
 & & & & \downarrow{\scriptstyle L} \\
TTI & \xrightarrow{\mu} & TI & \xrightarrow{L} & O
\end{array}
\qquad
\begin{array}{ccccc}
TS & & \xrightarrow{Tq_2} & & TTI \\
\downarrow{\scriptstyle Tq_1} & & & & \downarrow{\scriptstyle \mu} \\
 & & & & TI \\
 & & & & \downarrow{\scriptstyle L} \\
TTI & \xrightarrow{\mu} & TI & \xrightarrow{L} & O
\end{array}
\qquad
\begin{array}{ccccc}
 & & S & & \\
 & {\scriptstyle q_1} \swarrow & \vdots{\scriptstyle u} & \searrow {\scriptstyle q_2} & \\
TI & \xleftarrow{p_1} & R & \xrightarrow{p_2} & TI
\end{array}$$

[6]Note that it is not exactly a limit, as the defining property works with cones under T.

To show the versatility of our definition, we briefly explain a different example where the language is a set of words. This example cannot be recovered from the original definition by Arbib and Manes [AM74].

Example 3.4.6 (Syntactic congruence)**.** Let T be the list monad $(-)^\star$, so that $\mathsf{EM}(T)$ is the category of monoids, $I = A$, and $O = 2$. Given a language $L : A^\star \longrightarrow 2$, the Nerode equivalence as defined above is then the largest relation $R \subseteq A^\star \times A^\star$ such that

$$\frac{n \in \mathbb{N} \qquad (u_1, v_1), \dots, (u_n, v_n) \in R}{L(u_1 \cdots u_n) = L(v_1 \cdots v_n)}.$$

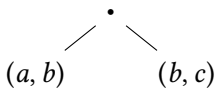
Equivalently, R is the largest relation such that

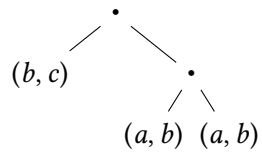
$$\frac{(u, v) \in R \qquad w, x \in A^\star}{L(wux) = L(wvx)},$$

which is precisely the *syntactic congruence* of the language.

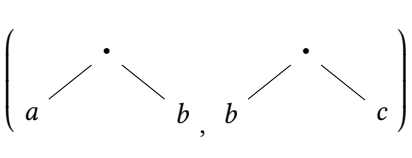
We now give a more intuitive example of what a Nerode equivalence is required to contain in the case of binary trees.

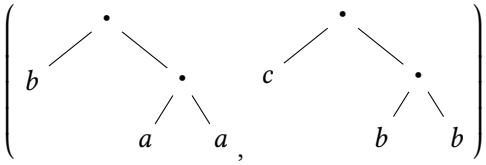
Example 3.4.7 (Binary trees)**.** Let T be the monad assigning to each set X the set of binary trees with leaves in X, and take $I = \{a, b, c\}$. Then the Nerode congruence $R \subseteq TI \times TI$ for a language $L : TI \longrightarrow O$ is such that if for instance $(a, b), (b, c) \in R$, where we consider a, b, and c as trees, then we can consider trees in TR such as





and deduce that the pairs of projected trees





must be such that for each pair (t_1, t_2) we have $L(t_1) = L(t_2)$.

We can show that the Nerode equivalence of a language in **Set** exists, as long as the monad is finitary. To define it concretely, we use the following piece of notation. For any set X and $x \in X$, denote by $\mathfrak{e}_x : 1 \longrightarrow X$ the constant x function, assuming no ambiguity of the set involved.

Proposition 3.4.8. *For* $\mathbf{C} = \mathbf{Set}$ *and* T *any finitary monad, every language* $L : TI \longrightarrow O$ *has a Nerode equivalence given by*

$$R = \{(u, v) \in TI \times TI \mid L \circ \mu \circ T[\mathsf{id}_{TI}, \mathfrak{e}_u] = L \circ \mu \circ T[\mathsf{id}_{TI}, \mathfrak{e}_v] : T(TI + 1) \longrightarrow O\}$$

with the corresponding projections $p_1, p_2 : R \longrightarrow TI$.

Proof. For each subset $X \subseteq R$, we define $p_X : R \longrightarrow TI$ by

$$p_X(r) = \begin{cases} p_1(r) & \text{if } r \notin X \\ p_2(r) & \text{if } r \in X. \end{cases}$$

We have $Tp_1 = Tp_\emptyset$ by definition. Consider any $t \in TR$ and let a finite $E \subseteq R$ with inclusion map $e : E \longrightarrow R$ and $t' \in TE$ be such that $T(e)(t') = t$. These exist because T is finitary. We will show by induction on E that

$$(L \circ \mu \circ Tp_\emptyset)(t) = (L \circ \mu \circ Tp_E)(t). \tag{3.1}$$

The case where $E = \emptyset$ is clear, so assume $E = E' \cup \{z\}$ with $z \notin E'$ and (3.1) holds when E' is substituted for E. We fix the singleton $1 = \{\Box\}$ and define $d : R \longrightarrow TI + 1$ by

$$d(r) = \begin{cases} (\kappa_1 \circ p_1)(r) & \text{if } r \notin E \\ (\kappa_1 \circ p_2)(r) & \text{if } r \in E' \\ \kappa_2(\Box) & \text{if } r = z, \end{cases}$$

where κ_1 and κ_2 are the coproduct injections. By this definition, we have $[\mathsf{id}_{TI}, \mathfrak{e}_{p_1(z)}] \circ d = p_{E'}$ and $[\mathsf{id}_{TI}, \mathfrak{e}_{p_2(z)}] \circ d = p_E$, so

$$\begin{aligned} (L \circ \mu \circ Tp_\emptyset)(t) &= (L \circ \mu \circ Tp_{E'})(t) && \text{(induction hypothesis)} \\ &= (L \circ \mu \circ T([\mathsf{id}_{TI}, \mathfrak{e}_{p_1(z)}] \circ d))(t) \\ &= (L \circ \mu \circ T([\mathsf{id}_{TI}, \mathfrak{e}_{p_2(z)}] \circ d))(t) && \text{(definition of } R) \\ &= (L \circ \mu \circ Tp_E)(t), \end{aligned}$$

thus concluding the proof of (3.1). Now $Tp_1 = Tp_\emptyset$ by definition and

$$Tp_E(t) = T(p_E \circ e)(t') = T(p_2 \circ e)(t') = Tp_2(t),$$

from which we find that $(L \circ \mu \circ Tp_1)(t) = (L \circ \mu \circ Tp_\emptyset)(t) = (L \circ \mu \circ Tp_E)(t) = (L \circ \mu \circ Tp_2)(t)$. As this argument works for any $t \in TR$, we have $L \circ \mu \circ Tp_1 = L \circ \mu \circ Tp_2$.

Now consider any set S with $q_1, q_2 : S \longrightarrow TI$ making

$$\begin{array}{ccccc} TS & & \xrightarrow{Tq_2} & & TTI \\ & & & & \downarrow{\scriptstyle \mu} \\ {\scriptstyle Tq_1}\downarrow & & & & TI \\ & & & & \downarrow{\scriptstyle L} \\ TTI & \xrightarrow{\mu} & TI & \xrightarrow{L} & O \end{array} \tag{3.2}$$

commute, and assume q_1 and q_2 have a common section $j : TI \longrightarrow S$. We define $u : S \longrightarrow R$ by $u(s) = (q_1(s), q_2(s))$. To see that this is indeed an element of R, note that for $k \in \{1, 2\}$,

$$\begin{aligned} L \circ \mu \circ T[\mathsf{id}_{TI}, \mathfrak{e}_{q_k(s)}] &= L \circ \mu \circ T[\mathsf{id}_{TI}, q_k \circ \mathfrak{e}_s] \\ &= L \circ \mu \circ T[q_k \circ j, q_k \circ \mathfrak{e}_s] && \text{(section)} \\ &= L \circ \mu \circ Tq_k \circ T[j, \mathfrak{e}_s], \end{aligned}$$

and therefore $L \circ \mu \circ T[\mathsf{id}_{TI}, \mathfrak{e}_{q_1(s)}] = L \circ \mu \circ T[\mathsf{id}_{TI}, \mathfrak{e}_{q_2(s)}]$ follows from (3.2). By definition, u is the unique map making the diagram below commute.

$$\begin{array}{ccccc} & & S & & \\ & {\scriptstyle q_1}\swarrow & \vdots{\scriptstyle u} & \searrow{\scriptstyle q_2} & \\ TI & \xleftarrow{p_1} & R & \xrightarrow{p_2} & TI \end{array}$$

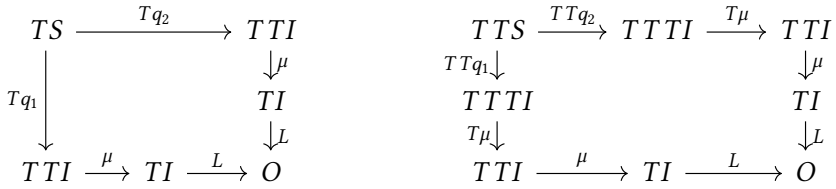
□

The definition of R above states that $u, v \in TI$ are related if and only if the elements of TI formed by putting either u or v in any *context* and then applying μ have the same value under L. A context is an element of $T(TI + 1)$, where $1 = \{\Box\}$ denotes a *hole* where either u or v can be plugged in. In the tree automata literature, such contexts, although restricted to contain a single instance of $\Box$, are used in algorithms for minimisation [HMM09] and learning [Sak90; DH07]. We will discuss such learning algorithms in the next chapter (Section 4.6).

A result related to Proposition 3.4.8 is Theorem 3.1 in [Boj15], which does not assume T is finitary and is given for any category of sorted sets. However, it does not construct a Nerode equivalence but shows the existence of a minimal T-automaton.

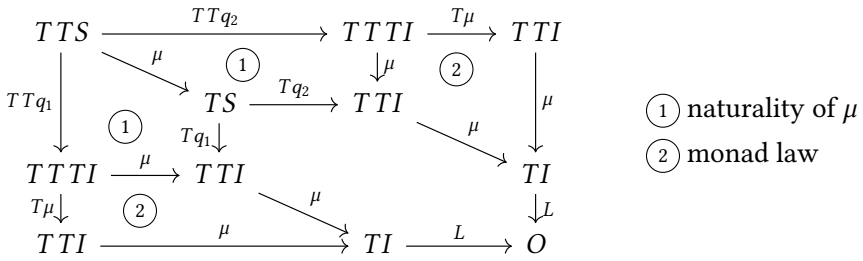
With Theorem 3.4.11 below we show one of our main results, that under a few mild assumptions the abstract equivalence is in fact a congruence: its quotient (coequaliser) is a T-automaton. Moreover, this T-automaton is minimal. Intuitively, given a language $L : TI \longrightarrow O$ that has a Nerode equivalence, we use the equivalence to quotient the T-automaton (GI, η, L). We first need two technical lemmas before we can prove the theorem.

Lemma 3.4.9. *Given a language L and $q_1, q_2 : S \longrightarrow TI$ making the diagram below on the left commute, the diagram on the right commutes.*



Furthermore, if (q_1, q_2) is a reflexive pair, then so is $(\mu_I \circ Tq_1, \mu_I \circ Tq_2)$.

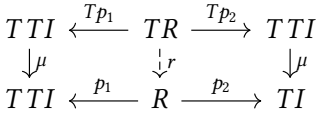
Proof. We extend the assumption to the following commutative diagram.



As for reflexivity, $(\mu_I \circ Tq_1, \mu_I \circ Tq_2)$ is the composition of the reflexive pairs (μ_I, μ_I) and (Tq_1, Tq_2). □

Lemma 3.4.10. *If $\mathbf{C}$ has coproducts, then for any Nerode equivalence (R, p_1, p_2) there exists a unique T-algebra structure $u : TR \longrightarrow R$ making p_1 and p_2 T-algebra homomorphisms $(R, u) \longrightarrow (TI, \mu)$ that have a common section.*

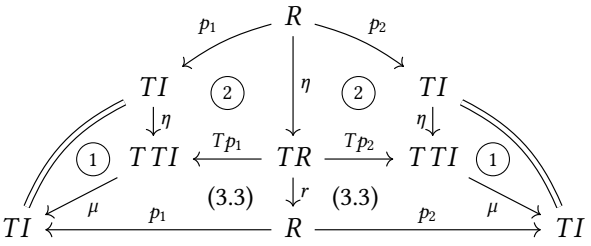
Proof. Let $L : TI \longrightarrow O$ be a language with Nerode equivalence (R, p_1, p_2). Then (p_1, p_2) is a reflexive pair by the Nerode equivalence property, since $(\mathsf{id}_{TI}, \mathsf{id}_{TI})$ is a reflexive pair trivially satisfying the Nerode equivalence condition. We apply Lemma 3.4.9 to obtain from the Nerode equivalence property a unique morphism $r : TR \longrightarrow R$ making the diagram below commute.

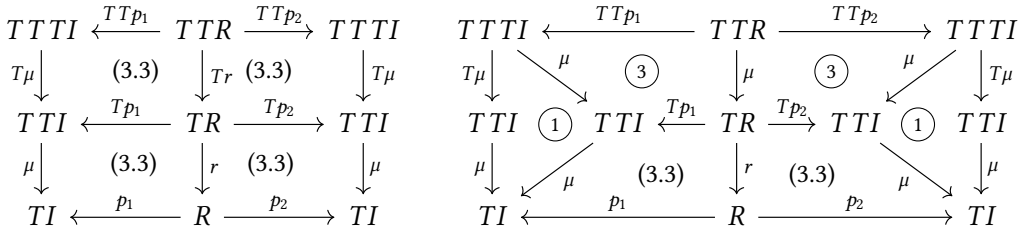


(3.3)

We need to show that (R, r) is a T-algebra. The first commutative diagram below shows that $r \circ \eta_R$ preserves p_1 and p_2, so since id_R also does this we must have $r \circ \eta_R = \mathsf{id}_R$ by the uniqueness

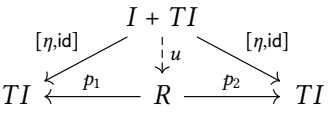
property of the Nerode equivalence.





As for the other two, we use a double application of Lemma 3.4.9 to see that the pair $(\mu \circ T\mu \circ TT p_1, \mu \circ T\mu \circ TT p_2)$ satisfies the Nerode equivalence conditions. Commutativity of the two diagrams then shows that both $r \circ Tr$ and $r \circ \mu$ are the unique map commuting with the pairs $(\mu \circ T\mu \circ TT p_1, \mu \circ T\mu \circ TT p_2)$ and (p_1, p_2), so they must be equal and (TR, r) is a T-algebra.

It remains to show that p_1 and p_2 have a common section in $\mathsf{EM}(T)$. To this end, note that $([\eta_I, \mathsf{id}_{TI}], [\eta_I, \mathsf{id}_{TI}])$ is a reflexive pair trivially satisfying the Nerode equivalence condition. Thus, we obtain by the Nerode equivalence property a unique morphism $u : I + TI \longrightarrow R$ making



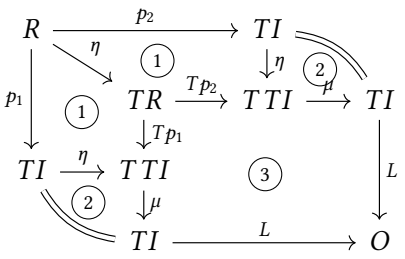
commute. Then for $k \in \{1, 2\}$,

$$p_k \circ (u \circ \kappa_1)^\sharp = (p_k \circ u \circ \kappa_1)^\sharp = (p_k \circ [\eta_I, \mathsf{id}_{TI}])^\sharp = \eta_I^\sharp = \mathsf{id}_{(TI,\mu)}. \qquad \square$$

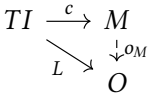
Theorem 3.4.11. *If* C *has coproducts and reflexive coequalisers and T preserves reflexive coequalisers, then for every language that has a Nerode equivalence there exists a minimal T-automaton accepting it.*

Proof. Let $L : TI \longrightarrow O$ be the language with Nerode equivalence (R, p_1, p_2) and $c : T \longrightarrow M$ the coequaliser of p_1 and p_2 in C. By Lemma 3.4.10 there exists a T-algebra structure on R making p_1 and p_2 T-algebra homomorphisms into (TI, μ) that have a common section. Since

T preserves reflexive coequalisers, they are lifted by U, and we have a morphism $m : TM \longrightarrow M$ making (M, m) a T-algebra such that c is a T-algebra homomorphism $(TI, \mu) \longrightarrow (M, m)$. Since the diagram below on the left commutes and c coequalises p_1 and p_2, there is a unique morphism o_M rendering the diagram on the right commutative.



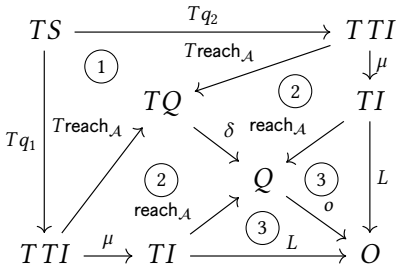
(1) naturality of η
(2) monad law
(3) Nerode equivalence



(3.4)

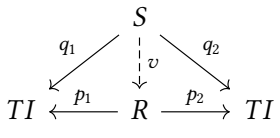
Choosing $i_M = c \circ \eta : I \longrightarrow M$, we obtain a T-automaton $\mathcal{M} = (M, m, i_M, o_M)$. Note that $U(i_M^\sharp) = c$, so c is the reachability map of $\mathcal{M}$. Hence, we find that $\mathcal{L}_\mathcal{M} = L$ by (3.4). The morphism c coequalises the reflexive pair (p_1, p_2) by definition, so $\mathcal{M}$ is reachable.

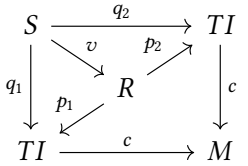
To see that $\mathcal{M}$ is minimal, consider any reachable T-automaton $\mathcal{A} = (Q, \delta, i, o)$ such that $\mathcal{L}_\mathcal{A} = L$. Reachability amounts to the reachability map $\mathsf{reach} : TI \longrightarrow UQ$ being the reflexive coequaliser of a pair of morphisms $q_1, q_2 : S \longrightarrow TI$. From commutativity of



(1) $\mathsf{reach}_\mathcal{A}$ coequalises q_1 and q_2
(2) $\mathsf{reach}_\mathcal{A}$ is a T-algebra homomorphism
(3) $\mathcal{L}_\mathcal{A} = L$

we obtain by the Nerode equivalence property a unique morphism $v : S \longrightarrow R$ making the diagram below on the left commute.

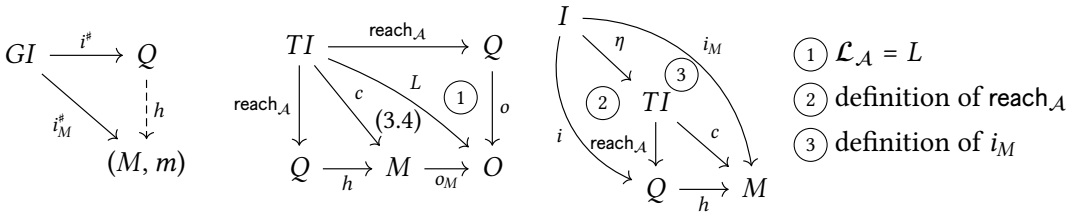




Extending this with c, the coequaliser of p_1 and p_2, gives the commutative diagram on the right. Recall that $U(i_M^\sharp) = c$. We now find

$$i_M^\sharp \circ q_1^\sharp = (U(i_M^\sharp) \circ q_1)^\sharp = (c \circ q_1)^\sharp = (c \circ q_2)^\sharp = (U(i_M^\sharp) \circ q_2)^\sharp = i_M^\sharp \circ q_2^\sharp.$$

Here the first and last equality apply a general naturality property of the adjunction. Since $\mathsf{reach}_{\mathcal{A}} = U(i^\sharp)$ is the reflexive coequaliser of q_1 and q_2, $i^\sharp$ is the reflexive coequaliser of $q_1^\sharp$ and $q_2^\sharp$ by Lemma 3.4.4. We then obtain a unique T-algebra homomorphism $h : (Q, \delta) \longrightarrow (M, m)$ making the diagram below on the left commute.



From commutativity of the other diagrams we find $o_M \circ h = o$ (using that $\mathsf{reach}_{\mathcal{A}}$ is epi) and $h \circ i = i_M$. Thus, h is a T-automaton homomorphism $\mathcal{A} \longrightarrow \mathcal{M}$. To see that it is unique, note that any T-automaton homomorphism $h' : \mathcal{A} \longrightarrow \mathcal{M}$ is a T-algebra homomorphism $(Q, \delta) \longrightarrow (M, m)$ such that $h' \circ i = i_M$. It is then not hard to see that $h' \circ i^\sharp = (h' \circ i)^\sharp = i_M^\sharp$. We conclude that $h' = h$ by the uniqueness property of h satisfying $h \circ i^\sharp = i_M^\sharp$. □

Remark 3.4.12. We briefly discuss the conditions of the above theorem in the specific case of $\mathbf{C} = \mathbf{Set}$ with T a finitary monad. This includes the setting of tree automata in **Set**, as a monad on **Set** is finitary if and only if $\mathsf{EM}(T)$ is equivalent to the category of algebras for a signature modulo equations. Proposition 3.4.8 shows that all Nerode equivalences exist here. Furthermore, Lack and Rosický [LR11] observe that an endofunctor on **Set** is finitary if and only if it preserves sifted colimits, of which reflexive coequalisers form an instance.

To conclude this section we show that the converse of the previous theorem also holds, using the existence of kernel pairs rather than coproducts. We need a technical lemma first.

Lemma 3.4.13. *If $q_1, q_2 : A \longrightarrow TB$ is a reflexive pair in $\mathbf{C}$, then so is $(q_1^\sharp, q_2^\sharp)$ in $\mathsf{EM}(T)$.*

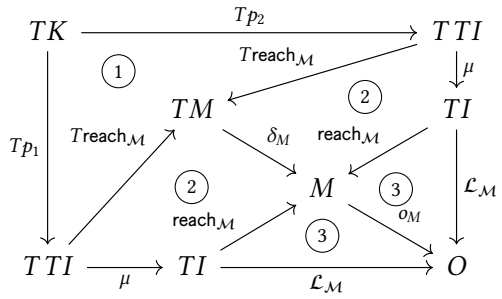
Proof. Assume $j : TB \longrightarrow A$ is the common section of q_1 and q_2. Then, for $k \in \{1, 2\}$,

$$q_k^\sharp \circ (\eta_A \circ j \circ \eta_B)^\sharp = U((U(q_k^\sharp) \circ \eta_A \circ j \circ \eta_B)^\sharp) = U((q_k \circ j \circ \eta_B)^\sharp) = U(\eta_B^\sharp) = \mathsf{id}_{TB}.$$ □

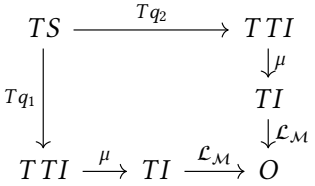
Theorem 3.4.14. *If $\mathbf{C}$ has kernel pairs and reflexive coequalisers and T preserves reflexive coequalisers, then every language that has a minimal T-automaton has a Nerode equivalence.*

Proof. Let $\mathcal{M} = (M, \delta_M, i_M, o_M)$ be a minimal T-automaton and $p_1, p_2 : K \longrightarrow TI$ the kernel pair of its reachability map $\mathsf{reach}_{\mathcal{M}} : GI \longrightarrow M$. We claim that K together with p_1 and p_2 forms

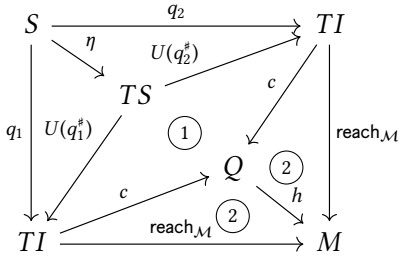
the Nerode equivalence of $\mathcal{L}_\mathcal{M}$. To see this, note that the diagram below on the left commutes.



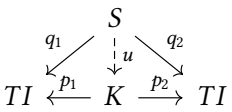
(1) kernel pair
(2) $\mathsf{reach}_\mathcal{M}$ is a T-algebra homomorphism
(3) definition of $\mathcal{L}_\mathcal{M}$



Now if S with $q_1, q_2 : S \longrightarrow TI$ is any reflexive pair making the diagram on the right commute, we let $c : TI \longrightarrow Q$ be the coequaliser of $U(q_1^\sharp)$ and $U(q_2^\sharp)$, noting that this is a reflexive pair by Lemma 3.4.13. Then since T preserves reflexive coequalisers, they are lifted by U, meaning that there exists a unique T-algebra structure $\delta : TQ \longrightarrow Q$ making $c : GI \longrightarrow (Q, \delta)$ a T-algebra homomorphism that is the coequaliser of $q_1^\sharp$ and $q_2^\sharp$. We also have $\mathcal{L}_\mathcal{M} \circ U(q_1^\sharp) = \mathcal{L}_\mathcal{M} \circ U(q_2^\sharp)$ by commutativity of the diagram on the right, so with c coequalising $U(q_1^\sharp)$ and $U(q_2^\sharp)$ there is a unique morphism $o : Q \longrightarrow O$ such that $o \circ c = \mathcal{L}_\mathcal{M}$. Setting $i = c \circ \eta_I$, we have a T-automaton (Q, δ, i, o) with reachability map $U(i^\sharp) = c$ that accepts the language $\mathcal{L}_\mathcal{M}$. By $\mathcal{M}$ being minimal there exists a unique T-automaton homomorphism $h : (Q, \delta, i, o) \longrightarrow \mathcal{M}$. Then the diagram below on the left commutes.



(1) c coequalises $U(q_1^\sharp)$ and $U(q_2^\sharp)$
(2) uniqueness of reachability maps



By p_1 and p_2 being the kernel pair of $\mathsf{reach}_\mathcal{M}$ there exists a unique morphism $u : S \longrightarrow K$ making the diagram on the right commute. □

3.5 Discussion

Although our minimisation construction in Section 3.3 suggests an algorithm, developing its details is left open. For classical tree automata there exist sophisticated variants of partition refinement [HMM09; AHK07], akin to Hopcroft's classical algorithm. A generalisation to the current algebraic setting is an interesting direction of research, for which a natural starting

point would be to try and integrate in our setting the efficient coalgebraic algorithm presented in [Dor+17].

Further, we characterised the minimal automaton as the greatest fixed point of a monotone function, recovering the notion of forward bisimulations as its post-fixed points (although it is perhaps more natural to think of these as congruences). This characterisation suggests an integration with up-to techniques [PS12; Bon+17; BP15], which have, to the best of our knowledge, not been applied to tree automata. In particular, we are interested in applying these algorithms to decide equivalence of series-parallel rational and series-rational expressions [LW00]. These may be in scope if were to generalise the minimisation procedure to the level of T-automata as defined in Section 3.4. This is because series-parallel algebras form a variety, which corresponds to a monad T.

Chapter 4

Categorical Automata Learning

Angluin's algorithm $L^\star$ (Section 2.3) has served as a basis for many automata learning algorithms that work for more expressive models than plain deterministic automata: I/O automata [AV10], weighted automata [BV96], register automata [IHS15; Aar+15], nominal automata [Moe+17], and families of DFAs (which describe ω-regular languages) [AF16]. Many of these extensions were developed independently and, though they bear close resemblance to the original algorithm, arguments of correctness and termination had to be repeated every time. This motivated Jacobs and Silva to provide a categorical understanding of $L^\star$ and capture essential data structures in an abstract way in the hope of developing a generic, modular, and parametric framework for automata learning based on (co)algebra [JS14]. Their work was taken further in Van Heerdt's master thesis [Hee16], with an explicit description of the categorical conditions under which their constructions work. Moreover, an abstract data structure central to the algorithm was identified, along with conditions for correctness. This work then formed the basis of a wider project on developing a *Categorical Automata Learning Framework*—CALF.[1]

In this chapter we introduce CALF. Most importantly, we will introduce an automata learning algorithm that generalises $L^\star$. We also instantiate our algorithm to derive a learning algorithm for generalised tree automata in **Set**. These tree automata are more general than the ones considered in previous literature [Sak90; DH03; BM07] because the transition functor is taken from a class that is strictly larger than the polynomial functors. That is, instead of automata where the transitions move from an ordered n-tuple of states to a next state, the automata we consider may have transitions originating from for instance an unordered (finite) set of states.

[1] `http://www.calf-project.org`

The work in the present chapter complements other recent work on abstract automata learning algorithms: Barlocco, Kupke, and Rot [BKR19] gave an algorithm for coalgebras of a functor, whereas Urbat and Schröder [US19] provided an algorithm for structures that can be represented as both algebras and coalgebras. Our focus instead is on algebras, such as tree automata, that cannot be covered by the aforementioned frameworks. A more detailed comparison will be given in Section 4.7.

We start by developing in Section 4.1 the abstract data structure that in CALF replaces the observation table, and accordingly we introduce suitable notions of closedness, consistency, and hypothesis. Our contributions are then as follows.

1. We devise an abstract iterative process to ensure closedness and consistency (Section 4.2).

2. We provide an abstract treatment of counterexamples, together with an analysis of progress made when processing a counterexample (Section 4.3).

3. We then put together a step-by-step generalisation of all components of $\mathrm{L}^\star$ for categorical automata (Section 4.4). This results in an algorithm that we prove correct.

4. We provide results that characterise conditions on the abstract datastructure under which the corresponding hypothesis is correct (Section 4.5). We then show how these can be used to devise an abstract minimisation algorithm from which reachability analysis, partition refinement, and even other learning algorithms can be recovered.

5. In Section 4.6 we finally instantiate our abstract $\mathrm{L}^\star$ algorithm to a concrete setting, providing the first learning algorithm for tree automata derived abstractly.

The chapter concludes with a discussion of related work in Section 4.7.

Our categorical setting involves the following assumptions, which we make throughout the chapter. We work in an arbitrary category $\mathbf{C}$ with finite coproducts, a final object, and a factorisation system $(\mathcal{E}, \mathcal{M})$ and fix a functor $F : \mathbf{C} \to \mathbf{C}$ and objects I and O in $\mathbf{C}$. We assume F preserves morphisms in $\mathcal{E}$ and admits an algebraically free monad $(F^\star, \eta, \mu)$.

4.1 Abstract Data Structures

In this section we introduce the basic notions underpinning CALF: generalisations of the observation table and the notions of closedness, consistency, and hypothesis. We call the gen-

eralised table a *wrapper* because it wraps a target object with two morphisms: one going in and one going out.

Definition 4.1.1 (Wrapper)**.** A *wrapper* for a *target* object X is a pair of morphisms $\mathcal{W} = \left(S \xrightarrow{\mathsf{sel}} X, X \xrightarrow{\mathsf{cla}} P\right)$. We define $e_{\mathcal{W}} = (\mathsf{cla} \circ \mathsf{sel})^{\triangleright}$ and $m_{\mathcal{W}} = (\mathsf{cla} \circ \mathsf{sel})^{\triangleleft}$ and call the object through which $\mathsf{cla} \circ \mathsf{sel}$ factorises $H_{\mathcal{W}}$, as indicated below.

$$\begin{array}{ccccc} S & \xrightarrow{\mathsf{sel}} & X & \xrightarrow{\mathsf{cla}} & P \\ & {\scriptstyle e_{\mathcal{W}}} \searrow & & \nearrow {\scriptstyle m_{\mathcal{W}}} & \\ & & H_{\mathcal{W}} & & \end{array}$$

In the above definition, $\mathsf{sel} : S \longrightarrow X$ can be seen as a *selector* of elements of X while $\mathsf{cla} : X \longrightarrow P$ can be seen as a *classifier*. This two-sided approximation induces the composition $\mathsf{cla} \circ \mathsf{sel} : S \longrightarrow P$ that both selects and classifies, with its image factorisation $H_{\mathcal{W}}$ intuitively approximating X. Note that $H_{\mathcal{W}}$ in general is neither a subobject nor a quotient of X, though using factorisation system properties one can show it is both a quotient of a subobject and a subobject of a quotient.

Example 4.1.2 (Observation table wrapper)**.** As in Example 2.2.11, let $\mathbf{C} = \mathbf{Set}$ with the factorisation system $(\mathcal{E}, \mathcal{M}) = (\text{surjective}, \text{injective})$, and define $F = (-) \times A$ for a finite set A, $I = 1 = \{*\}$, and $O = 2 = \{0, 1\}$. Recall that automata with these parameters are deterministic automata (DAs). Consider a DA $\mathcal{A} = (Q, \delta, i, o)$. For all $S, E \subseteq A^\star$, we define $\mathsf{sel}_{\mathcal{A},S} : S \longrightarrow Q$ and $\mathsf{cla}_{\mathcal{A},E} : Q \longrightarrow 2^E$ by

$$\mathsf{sel}_{\mathcal{A},S}(w) = \mathsf{reach}_{\mathcal{A}}(*, w) \qquad\qquad \mathsf{cla}_{\mathcal{A},E}(q)(e) = (o \circ \delta^\star)(q, e).$$

Let $\mathcal{W} = (\mathsf{sel}_{\mathcal{A},S}, \mathsf{cla}_{\mathcal{A},E})$. One can show the following connection between the composition of these maps and the language of $\mathcal{A}$ (see Proposition 4.1.3 below for a formal proof):

$$(\mathsf{cla}_{\mathcal{A},E} \circ \mathsf{sel}_{\mathcal{A},S})(s)(e) = \mathcal{L}_{\mathcal{A}}(*, se).$$

(See Proposition 4.1.3 below.) This composed function $\mathsf{cla}_{\mathcal{A},E} \circ \mathsf{sel}_{\mathcal{A},S} : S \longrightarrow 2^E$ is precisely the upper part of the observation table with rows S and columns E in Angluin's algorithm for regular languages (Section 2.3). The image of $\mathsf{cla}_{\mathcal{A},E} \circ \mathsf{sel}_{\mathcal{A},S}$ is precisely the set of rows that appear in the table, which are used as states in the hypothesis, and can be obtained as $H_{\mathcal{W}}$.

Before we define hypotheses in this abstract framework, we need generalised notions of closedness and consistency. So far we have only recovered the upper part of the table with our wrapper, but closedness and consistency rely on the lower part. Given a wrapper

$\mathcal{W} = \left(S - \mathsf{sel} \to Q, Q - \mathsf{cla} \to P\right)$ for[2] an automaton (Q, δ, i, o), we claim that the composition

$$FS \xrightarrow{F\mathsf{sel}} FQ \xrightarrow{\delta} Q \xrightarrow{\mathsf{cla}} P$$

is an approximation of the dynamics δ that instantiates to the lower part of the table in the setting detailed in Example 4.1.2. The result below proves this. Furthermore, the compositions $\mathsf{cla} \circ i : I \longrightarrow P$ and $o \circ \mathsf{sel} : S \longrightarrow O$ will be used to determine the initial state and output map of the hypothesis by incurring additional closedness and consistency requirements that are satisfied automatically throughout runs of the $\mathtt{L}^\star$ algorithm.

Proposition 4.1.3. *Let* $\mathbf{C} = \mathbf{Set}$ *with* $(\mathcal{E}, \mathcal{M}) = (\textit{surjective}, \textit{injective})$, $F = (-) \times A$ *for a finite set* A, $I = 1$, *and* $O = 2$. *Given* $S, E \subseteq A^\star$, *we have, using the maps defined in Example 4.1.2,*

$$\begin{aligned}
\mathsf{cla}_{\mathcal{A},E} \circ \mathsf{sel}_{\mathcal{A},S} &: S \longrightarrow 2^E & (\mathsf{cla}_{\mathcal{A},E} \circ \mathsf{sel}_{\mathcal{A},S})(s)(e) &= \mathcal{L}_{\mathcal{A}}(*, se) \\
\mathsf{cla}_{\mathcal{A},E} \circ \delta \circ (\mathsf{sel}_{\mathcal{A},S} \times \mathsf{id}_A) &: S \times A \longrightarrow 2^E & (\mathsf{cla}_{\mathcal{A},E} \circ \delta \circ (\mathsf{sel}_{\mathcal{A},S} \times \mathsf{id}_A))(s,a)(e) &= \mathcal{L}_{\mathcal{A}}(*, sae) \\
\mathsf{cla}_{\mathcal{A},E} \circ i &: 1 \longrightarrow 2^E & (\mathsf{cla}_{\mathcal{A},E} \circ i)(*)(e) &= \mathcal{L}_{\mathcal{A}}(*, e) \\
o \circ \mathsf{sel}_{\mathcal{A},S} &: S \longrightarrow 2 & (o \circ \mathsf{sel}_{\mathcal{A},S})(s) &= \mathcal{L}_{\mathcal{A}}(*, s).
\end{aligned}$$

Proof. For all words $u, v \in A^\star$, we have

$$\delta^\star(\mathsf{reach}_{\mathcal{A}}(*, u), v) = \mathsf{reach}_{\mathcal{A}}(*, uv), \tag{4.1}$$

since

$$\begin{aligned}
\delta^\star(\mathsf{reach}_{\mathcal{A}}(*, u), v) &= (\delta^\star \circ (\mathsf{reach}_{\mathcal{A}} \times \mathsf{id}_{A^\star}))((*, u), v) \\
&= (\mathsf{reach}_{\mathcal{A}} \circ \mu_1)((*, u), v) && (\mathsf{reach}_{\mathcal{A}} \text{ is an algebra homomorphism}) \\
&= \mathsf{reach}_{\mathcal{A}}(*, uv) && (\text{definition of } \mu).
\end{aligned}$$

We then have

$$\begin{aligned}
(\mathsf{cla}_{\mathcal{A},E} \circ \mathsf{sel}_{\mathcal{A},S})(s)(e) &= (\mathsf{cla}_{\mathcal{A},E} \circ \mathsf{reach}_{\mathcal{A}})(*, s)(e) && (\text{definition of } \mathsf{sel}_{\mathcal{A},S}) \\
&= (o \circ \delta^\star)(\mathsf{reach}_{\mathcal{A}}(*, s), e) && (\text{definition of } \mathsf{cla}_{\mathcal{A},E}) \\
&= (o \circ \mathsf{reach}_{\mathcal{A}})(*, se) && (4.1) \\
&= \mathcal{L}_{\mathcal{A}}(*, se) && (\text{definition of } \mathcal{L}_{\mathcal{A}}),
\end{aligned}$$

[2] We often declare a wrapper $\mathcal{W}$ for an automaton $\mathcal{A}$, with $\mathcal{W}$ referencing the state space of $\mathcal{A}$. This phrasing does not have any special meaning; "for" simply indicates quantification.

$$\begin{aligned}
(\mathsf{cla}_{\mathcal{A},E} \circ \delta \circ (\mathsf{sel}_{\mathcal{A},S} \times \mathsf{id}_A))(s,a)(e) &= (\mathsf{cla}_{\mathcal{A},E} \circ \delta)(\mathsf{reach}_{\mathcal{A}}(*,s),a)(e) && \text{(defintion of } \mathsf{sel}_{\mathcal{A},S}) \\
&= (\mathsf{cla}_{\mathcal{A},E} \circ \mathsf{reach}_{\mathcal{A}})(*,sa)(e) && \text{(definition of } \mathsf{reach}_{\mathcal{A}}) \\
&= (o \circ \delta^\star)(\mathsf{reach}_{\mathcal{A}}(*,sa),e) && \text{(definition of } \mathsf{cla}_{\mathcal{A},E}) \\
&= (o \circ \mathsf{reach}_{\mathcal{A}})(*,sae) && (4.1) \\
&= \mathcal{L}_{\mathcal{A}}(*,sae) && \text{(definition of } \mathcal{L}_{\mathcal{A}}),
\end{aligned}$$

$$\begin{aligned}
(\mathsf{cla}_{\mathcal{A},E} \circ i)(*)(e) &= (o \circ \delta^\star)(i(*),e) && \text{(definition of } \mathsf{cla}_{\mathcal{A},E}) \\
&= (o \circ \delta^\star \circ (i \times \mathsf{id}_{A^\star}))(*,e) \\
&= (o \circ \mathsf{reach}_{\mathcal{A}})(*,e) && \text{(definition of } \mathsf{reach}_{\mathcal{A}}) \\
&= \mathcal{L}_{\mathcal{A}}(*,e) && \text{(definition of } \mathcal{L}_{\mathcal{A}}),
\end{aligned}$$

and

$$\begin{aligned}
(o \circ \mathsf{sel}_{\mathcal{A},S})(s) &= (o \circ \mathsf{reach}_{\mathcal{A}})(*,s) && \text{(definition of } \mathsf{reach}_{\mathcal{A}}) \\
&= \mathcal{L}_{\mathcal{A}}(*,s) && \text{(definition of } \mathcal{L}_{\mathcal{A}}). \qquad \square
\end{aligned}$$

Crucially, the above result shows that the compositions are independent of the target DA (fixing its language) and can in fact be computed entirely using membership queries. We now proceed to give the full abstract closedness and consistency definitions. The definitions are relative to an automaton of which we want to approximate the structure.

Definition 4.1.4 (Closedness and consistency)**.** We say that a wrapper $\mathcal{W} = \left(S - \mathsf{sel} \to Q, Q - \mathsf{cla} \to P\right)$ is $\mathcal{A}$*-closed* for an automaton $\mathcal{A} = (Q, \delta, i, o)$ if there exist morphisms $FS \longrightarrow H_{\mathcal{W}}$ and $I \longrightarrow H_{\mathcal{W}}$ making the diagrams below commute.

$$\begin{array}{ccccc}
FS & \xrightarrow{F\mathsf{sel}} & FQ & \xrightarrow{\delta} & Q \\
\downarrow & & & & \downarrow{\scriptstyle \mathsf{cla}} \\
H_{\mathcal{W}} & & \xrightarrow{\quad m_{\mathcal{W}} \quad} & & P
\end{array}
\qquad\qquad
\begin{array}{ccc}
I & \xrightarrow{\quad i \quad} & Q \\
\downarrow & & \downarrow{\scriptstyle \mathsf{cla}} \\
H_{\mathcal{W}} & \xrightarrow{\quad m_{\mathcal{W}} \quad} & P
\end{array}$$

The wrapper $\mathcal{W}$ is $\mathcal{A}$*-consistent* if there exist morphisms $FH_{\mathcal{W}} \longrightarrow P$ and $H_{\mathcal{W}} \longrightarrow O$ making the diagrams below commute.

$$\begin{array}{ccccc}
FS & & \xrightarrow{\quad Fe_{\mathcal{W}} \quad} & & FH_{\mathcal{W}} \\
{\scriptstyle F\mathsf{sel}}\downarrow & & & & \downarrow \\
FQ & \xrightarrow{\delta} & Q & \xrightarrow{\mathsf{cla}} & P
\end{array}
\qquad\qquad
\begin{array}{ccc}
S & \xrightarrow{\quad e_{\mathcal{W}} \quad} & H_{\mathcal{W}} \\
{\scriptstyle \mathsf{sel}}\downarrow & & \downarrow \\
Q & \xrightarrow{\quad o \quad} & O
\end{array}$$

To make sense of these properties in **Set**, we have the following result.

Proposition 4.1.5. *Consider given morphisms f, g, i, j as below, with $g \in \mathcal{M}$ and $i \in \mathcal{E}$.*

A morphism $h : A \longrightarrow B$ making the triangle on the left commute exists if and only if for all $a \in A$ there exists $b \in B$ such that $g(b) = f(a)$. A morphism $k : Y \longrightarrow Z$ making the triangle on the right commute exists if and only if for all $x_1, x_2 \in X$ such that $i(x_1) = i(x_2)$ we have $j(x_1) = j(x_2)$.

Proof. If h exists, then for all $a \in A$ we have that $h(a) \in B$ satisfies $g(h(a)) = f(a)$. Conversely, assume for each $a \in A$ there exists $b_a \in B$ such that $g(b_a) = f(a)$. We define h by $h(a) = b_a$, which satisfies

$$g(h(a)) = g(b_a) = f(a)$$

as required.

If k exists, then for all $x_1, x_2 \in X$ with $i(x_1) = i(x_2)$ we have

$$j(x_1) = k(i(x_1)) = k(i(x_2)) = j(x_2).$$

Conversely, assume that for all $x_1, x_2 \in X$ such that $i(x_1) = i(x_2)$ we have $j(x_1) = j(x_2)$. Then k defined by $k(i(x)) = j(x)$ for all $x \in X$ is well defined. □

Using this result, we recover the original notions of closedness and consistency in the DA setting.

Example 4.1.6 (Closedness and consistency for DAs)**.** Recall the morphisms defined in Example 4.1.2 and consider a DA $\mathcal{A} = (Q, \delta, i, o)$. Given $S, E \subseteq A^\star$, the wrapper $\mathcal{W} = (\mathsf{sel}_{\mathcal{A},S}, \mathsf{cla}_{\mathcal{A},E})$ is $\mathcal{A}$-closed if and only if there exists $s \in S$ such that

$$(\mathsf{cla}_{\mathcal{A},E} \circ \mathsf{sel}_{\mathcal{A},S})(s) = (\mathsf{cla}_{\mathcal{A},E} \circ i)(*)$$

and for all $t \in S \times A$ there exists $s \in S$ such that

$$(\mathsf{cla}_{\mathcal{A},E} \circ \mathsf{sel}_{\mathcal{A},S})(s) = (\mathsf{cla}_{\mathcal{A},E} \circ \delta \circ (\mathsf{sel}_{\mathcal{A},S} \times \mathsf{id}_A))(t).$$

The first condition holds immediately if $\varepsilon \in S$ because $(\mathsf{cla}_{\mathcal{A},E} \circ \mathsf{sel}_{\mathcal{A},S})(\varepsilon) = \mathsf{cla}_{\mathcal{A},E} \circ i$ via Proposition 4.1.3; in the second condition, recall from Proposition 4.1.3 that

$$\mathsf{cla}_{\mathcal{A},E} \circ \delta \circ (\mathsf{sel}_{\mathcal{A},S} \times \mathsf{id}_A) : S \times A \longrightarrow 2^E$$

represents the lower part of the observation table associated with S and E. Thus, this is the same as the closedness condition in $\mathrm{L}^\star$ (Section 2.3).

The wrapper $\mathcal{W}$ is $\mathcal{A}$-consistent if and only if for all $s_1, s_2 \in S$ such that $(\mathsf{cla}_{\mathcal{A},E} \circ \mathsf{sel}_{\mathcal{A},S})(s_1) = (\mathsf{cla}_{\mathcal{A},E} \circ \mathsf{sel}_{\mathcal{A},S})(s_2)$ it holds that

$$(o \circ \mathsf{sel}_{\mathcal{A},S})(s_1) = (o \circ \mathsf{sel}_{\mathcal{A},S})(s_2)$$

and

$$(\mathsf{cla}_{\mathcal{A},E} \circ \delta \circ (\mathsf{sel}_{\mathcal{A},S} \times \mathsf{id}_A))(s_1) = (\mathsf{cla}_{\mathcal{A},E} \circ \delta \circ (\mathsf{sel}_{\mathcal{A},S} \times \mathsf{id}_A))(s_2).$$

The first condition holds immediately if $\varepsilon \in E$ because $(\mathsf{cla}_{\mathcal{A},E} \circ \mathsf{sel}_{\mathcal{A},S})(s)(\varepsilon) = (o \circ \mathsf{sel}_{\mathcal{A},S})(s)$ for all $s \in S$ via Proposition 4.1.3; the second condition says that observations with the same behaviour (i.e., the same row) should lead to rows with the same content in the lower part of the table.

Remark 4.1.7. In a more general setting, if $\mathcal{E}$ contains only regular epimorphisms, then for any wrapper $\mathcal{W} = \left(S - \mathsf{sel} \to Q, Q - \mathsf{cla} \to P\right)$ for an automaton $\mathcal{A} = (Q, \delta, i, o)$, we have that $e_{\mathcal{W}}$ is the coequaliser of morphisms $f, g : X \longrightarrow S$ and $Fe_{\mathcal{W}}$ is the coequaliser of morphisms $p, q : Y \longrightarrow FS$ for certain objects X and Y. The wrapper $\mathcal{W}$ is $\mathcal{A}$-consistent in this situation if and only if the equalities below hold.

$$o \circ \mathsf{sel} \circ f = o \circ \mathsf{sel} \circ g \qquad\qquad \mathsf{cla} \circ \delta \circ F\mathsf{sel} \circ p = \mathsf{cla} \circ \delta \circ F\mathsf{sel} \circ q.$$

A dual remark applies for closedness when $\mathcal{M}$ contains only regular monomorphisms.

The following main result of this section shows that closedness and consistency correspond precisely to a particular automaton structure on $H_{\mathcal{W}}$. This correspondence is used to construct the hypothesis. We omit the proof below because it will follow from the more general result of Proposition 4.5.2.

Theorem 4.1.8 (Hypothesis)**.** *A wrapper $\left(S - \mathsf{sel} \to Q, Q - \mathsf{cla} \to P\right)$ is $\mathcal{A}$-closed and $\mathcal{A}$-consistent for an automaton $\mathcal{A} = (Q, \delta, i, o)$ if and only if $H_{\mathcal{W}}$ extends to an automaton $\mathcal{H}_{\mathcal{W}} = (H_{\mathcal{W}}, \delta_{\mathcal{W}}, i_{\mathcal{W}}, o_{\mathcal{W}})$ making the diagrams below commute. We call the automaton $\mathcal{H}_{\mathcal{W}}$* the hypothesis*, assuming the wrapper and target automaton are clear from the context.*

$$\begin{array}{ccccc} FQ & \xleftarrow{F\mathsf{sel}} & FS & \xrightarrow{Fe_{\mathcal{W}}} & FH_{\mathcal{W}} \\ {\scriptstyle\delta}\downarrow & & & & \downarrow{\scriptstyle\delta_{\mathcal{W}}} \\ Q & \xrightarrow{\mathsf{cla}} & P & \xleftarrow{m_{\mathcal{W}}} & H_{\mathcal{W}} \end{array} \qquad \begin{array}{ccc} I & \xrightarrow{i} & Q \\ {\scriptstyle i_{\mathcal{W}}}\downarrow & & \downarrow{\scriptstyle\mathsf{cla}} \\ H_{\mathcal{W}} & \xrightarrow{m_{\mathcal{W}}} & P \end{array} \qquad \begin{array}{ccc} S & \xrightarrow{e_{\mathcal{W}}} & H_{\mathcal{W}} \\ {\scriptstyle\mathsf{sel}}\downarrow & & \downarrow{\scriptstyle o_{\mathcal{W}}} \\ Q & \xrightarrow{o} & O \end{array}$$

4.2 Abstract Iterations

A crucial point for the development of the abstract algorithm is defining what it means to resolve the "current" closedness and consistency defects. We refer to these as *local* defects, by which we mean the ones directly visible. For instance, in the DA example, the local closedness defects are the rows from the bottom part missing in the upper part, together with the empty word row if it is missing. The local consistency defects are the pairs of row labels that should be distinguished based on differing acceptance of those labels by the target, or differing rows when the labels are extended with a single symbol.

Definition 4.2.1 (Local closedness and consistency)**.** We say that a wrapper $\mathcal{W} = \left(S - \mathsf{sel} \to Q, Q - \mathsf{cla} \to P\right)$ is *locally $\mathcal{A}$-closed w.r.t.* $\mathsf{sel}' : S' \longrightarrow Q$ for an automaton $\mathcal{A} = (Q, \delta, i, o)$ if $\mathsf{sel}'^{\triangleleft} \leq \mathsf{sel}^{\triangleleft}$ and there exist morphisms $FS' \longrightarrow H_{\mathcal{W}}$ and $I \longrightarrow H_{\mathcal{W}}$ making the diagrams below commute.

$$\begin{array}{ccccc} FS' & \xrightarrow{F\mathsf{sel}'} & FQ & \xrightarrow{\delta} & Q \\ \downarrow & & & & \downarrow{\scriptstyle \mathsf{cla}} \\ H_{\mathcal{W}} & & \xrightarrow{m_{\mathcal{W}}} & & P \end{array} \qquad \begin{array}{ccc} I & \xrightarrow{i} & Q \\ \downarrow & & \downarrow{\scriptstyle \mathsf{cla}} \\ H_{\mathcal{W}} & \xrightarrow{m_{\mathcal{W}}} & P \end{array}$$

The wrapper $\mathcal{W}$ is *locally $\mathcal{A}$-consistent w.r.t.* $\mathsf{cla}' : Q \longrightarrow P'$ if $\mathsf{cla}^{\triangleright} \leq \mathsf{cla}'^{\triangleright}$ and there exist morphisms $FH_{\mathcal{W}} \longrightarrow P'$ and $H_{\mathcal{W}} \longrightarrow O$ making the diagrams below commute.

$$\begin{array}{ccccc} FS & & \xrightarrow{Fe_{\mathcal{W}}} & & FH_{\mathcal{W}} \\ {\scriptstyle F\mathsf{sel}}\downarrow & & & & \downarrow \\ FQ & \xrightarrow{\delta} & Q & \xrightarrow{\mathsf{cla}'} & P' \end{array} \qquad \begin{array}{ccc} S & \xrightarrow{e_{\mathcal{W}}} & H_{\mathcal{W}} \\ {\scriptstyle \mathsf{sel}}\downarrow & & \downarrow \\ Q & \xrightarrow{o} & O \end{array}$$

Note that a wrapper $(\mathsf{sel}, \mathsf{cla})$ as in the above definition is $\mathcal{A}$-closed if and only if it is locally $\mathcal{A}$-closed w.r.t. sel and $\mathcal{A}$-consistent if and only if it is locally $\mathcal{A}$-consistent w.r.t. cla.

Example 4.2.2 (Local closedness and consistency for DAs)**.** Recall the morphisms defined in Example 4.1.2. Given an automaton $\mathcal{A} = (Q, \delta, i, o)$ and $S, S', E \subseteq A^\star$, the wrapper $\mathcal{W} = (\mathsf{sel}_{\mathcal{A},S}, \mathsf{cla}_{\mathcal{A},E})$ is locally $\mathcal{A}$-closed w.r.t. $\mathsf{sel}_{\mathcal{A},S'}$ if $S' \subseteq S$, there exists $s \in S$ such that

$$(\mathsf{cla}_{\mathcal{A},E} \circ \mathsf{sel}_{\mathcal{A},S})(s) = (\mathsf{cla}_{\mathcal{A},E} \circ i)(*),$$

and for all $t \in S' \times A$ there exists $s \in S$ such that

$$(\mathsf{cla}_{\mathcal{A},E} \circ \mathsf{sel}_{\mathcal{A},S})(s) = (\mathsf{cla}_{\mathcal{A},E} \circ \delta \circ (\mathsf{sel}_{\mathcal{A},S'} \times \mathsf{id}_A))(t).$$

The last condition is equivalent to the property that any row in the bottom part of the table (S', E) can be found in the top part of the table (S, E).

Given $E' \subseteq A^\star$, the wrapper $\mathcal{W}$ is locally $\mathcal{A}$-consistent w.r.t. $\mathsf{cla}_{\mathcal{A},E'}$ if $E' \subseteq E$ and for all $s_1, s_2 \in S$ such that $(\mathsf{cla}_{\mathcal{A},E} \circ \mathsf{sel}_{\mathcal{A},S})(s_1) = (\mathsf{cla}_{\mathcal{A},E} \circ \mathsf{sel}_{\mathcal{A},S})(s_2)$ it holds that

$$(o \circ \mathsf{sel}_{\mathcal{A},S})(s_1) = (o \circ \mathsf{sel}_{\mathcal{A},S})(s_2)$$

and

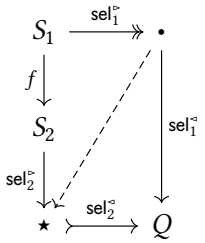
$$(\mathsf{cla}_{\mathcal{A},E'} \circ \delta \circ (\mathsf{sel}_{\mathcal{A},S} \times \mathsf{id}_A))(s_1) = (\mathsf{cla}_{\mathcal{A},E'} \circ \delta \circ (\mathsf{sel}_{\mathcal{A},S} \times \mathsf{id}_A))(s_2).$$

This last condition is equivalent to the property that for all $s_1, s_2 \in S$ mapping to the same row in the upper part of (S, E), the rows for $s_1 a$ and $s_2 a$ are the same in the lower part of (S, E') for all $a \in A$.

Proposition 4.2.4 below shows that for each wrapper $(S - \mathsf{sel} \to Q, Q - \mathsf{cla} \to P)$ for an automaton $\mathcal{A} = (Q, \delta, i, o)$ we can always find $\mathsf{sel}' : S' \longrightarrow Q$ such that $(\mathsf{sel}', \mathsf{cla})$ is locally $\mathcal{A}$-closed w.r.t. sel. This is done by adding the object obtained by going one level higher along the functor $F_I = I + F(-)$. That is, $S' = S + F_I S$. In the DA case this is equivalent to adding the empty word and all single letter successors of the current words. We first need the following technical result.

Lemma 4.2.3. *For all morphisms* $\mathsf{sel}_1 : S_1 \longrightarrow Q$, $\mathsf{sel}_2 : S_2 \longrightarrow Q$, *and* $f : S_1 \longrightarrow S_2$ *such that* $\mathsf{sel}_2 \circ f = \mathsf{sel}_1$ *we have* $\mathsf{sel}_1^\triangleleft \leq \mathsf{sel}_2^\triangleleft$.

Proof. This follows directly from the unique diagonal obtained in the commutative diagram below.

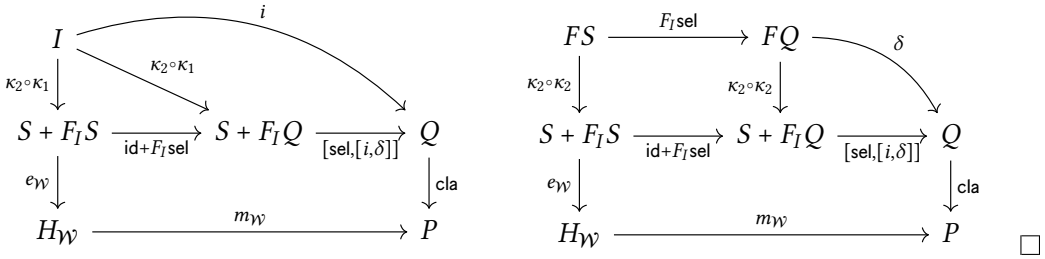


□

Proposition 4.2.4. *Given an automaton* $\mathcal{A} = (Q, \delta, i, o)$ *and any morphisms* $\mathsf{sel} : S \longrightarrow Q$ *and* $\mathsf{cla} : Q \longrightarrow P$, *the wrapper* $([\mathsf{sel}, [i, \delta] \circ F_I \mathsf{sel}], \mathsf{cla})$ *is locally* $\mathcal{A}$*-closed w.r.t.* sel.

Proof. Let $\mathcal{W} = ([\mathsf{sel}, [i, \delta] \circ F_I \mathsf{sel}], \mathsf{cla})$. Note that $\mathsf{sel}^\triangleleft \leq [\mathsf{sel}, [i, \delta] \circ F_I \mathsf{sel}]^\triangleleft$ by Lemma 4.2.3 (via $\kappa_1 : S \longrightarrow S + F_I S$). Below we see that there exist morphisms $I \longrightarrow H_{\mathcal{W}}$ and $FS \longrightarrow H_{\mathcal{W}}$

satisfying the required commutativity conditions from Definition 4.2.1.



□

4.3 Counterexamples

In the original L* algorithm counterexamples are used to refine the state space of the hypothesis. A crucial property for termination, which we prove at a high level of generality in this section, is that adding counterexamples to a closed and consistent table results in a table which fails to be either closed or consistent, and hence needs to be extended. In turn, this results in progress being made in the algorithm. We will show how we can use *recursive coalgebras* [Osi74; Tay99] as witnesses for discrepancies—i.e., as counterexamples—between a hypothesis and the target language in our abstract approach. Recursive coalgebras have been used to generalise prefix-closedness in an automata learning context in earlier work [Hee+18b], and in particular to generalise counterexamples [BKR19; US19]. Let us first recall the definition, in which we use the functor $F_I = I + F(-) : \mathbf{C} \longrightarrow \mathbf{C}$.

Definition 4.3.1 (Recursive coalgebras)**.** A coalgebra $\rho : S \longrightarrow F_I S$ is *recursive* if for every algebra $x : F_I X \longrightarrow X$ there is a unique morphism $x^\rho : S \longrightarrow X$ making the diagram below commute.

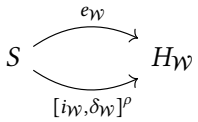
$$\begin{array}{ccc} F_I S & \xrightarrow{F_I x^\rho} & F_I X \\ {\scriptstyle\rho}\uparrow & & \downarrow{\scriptstyle x} \\ S & \xrightarrow{x^\rho} & X \end{array}$$

The uniqueness property makes these morphisms commute with algebra homomorphisms. That is, if $\rho : S \longrightarrow F_I S$ is recursive and (X, x) and (Y, y) are F_I-algebras, then for any F_I-algebra morphism $h : (X, x) \longrightarrow (Y, y)$ we have $h \circ x^\rho = y^\rho$. Given an automaton $\mathcal{A} = (Q, \delta, i, o)$ and a recursive $\rho : S \longrightarrow F_I S$, the map $[i, \delta]^\rho : S \longrightarrow Q$ can be seen as a generalised reachability map. Indeed, noting that $[\eta_I, \theta_I] : F_I F^\star(I) \longrightarrow F^\star(I)$ is the initial F_I-algebra and thus has an inverse, the reachability map of $\mathcal{A}$ can be recovered as $\mathsf{reach}_\mathcal{A} = [i, \delta]^{[\eta_I, \theta_I]^{-1}} : F^\star(I) \longrightarrow Q$. The following example shows that recursive coalgebras generalise prefix-closed sets.

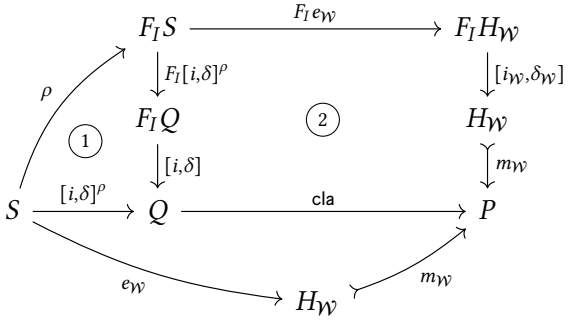
Example 4.3.2 (Prefix-closedness)**.** A *prefix-closed* subset $S \subseteq A^\star$ is easily equipped with a coalgebra structure $\rho : S \longrightarrow 1 + S \times A$ that detaches the last letter from each non-empty word and assigns $*$ to the empty one. Such a coalgebra is recursive, with the unique map into an algebra being defined as a restricted reachability map. In fact, Adámek et al. [AMM20] have shown that under certain conditions that are satisfied in the DA setting, recursivity of a coalgebra is equivalent to having a coalgebra homomorphism into $[\eta_1, \theta_1]^{-1}$. This means that every recursive coalgebra is isomorphic to one given by a prefix-closed multiset of words. If the unique morphism into $[\eta_1, \theta_1]$ is injective, the multiset becomes a set.

In $\mathsf{L}^\star$, using a prefix-closed set of words to label the rows of the table guarantees a reachable hypothesis. Analogously, using a recursive F_I-coalgebra as the selector in a wrapper targeting the state space of an automaton guarantees a reachable hypothesis, as we show next. The commutativity result below shows the stronger result that instantiates in $\mathsf{L}^\star$ to the fact that after reading a word from the row set the hypothesis ends up in the state given by the row indexed by that word.

Proposition 4.3.3. *Given an automaton $\mathcal{A} = (Q, \delta, i, o)$, a recursive $\rho : S \longrightarrow F_I S$, and $\mathsf{cla} : Q \longrightarrow P$, consider the wrapper $\mathcal{W} = \big(S - [i,\delta]^\rho \to Q, Q - \mathsf{cla} \to P\big)$. If $\mathcal{W}$ is $\mathcal{A}$-closed and $\mathcal{A}$-consistent, the diagram below commutes and $\mathcal{H}_\mathcal{W}$ is reachable.*



Proof. We will first show that $e_\mathcal{W} = [i_\mathcal{W}, \delta_\mathcal{W}]^\rho$. This follows from the commutative diagram below as a result of $m_\mathcal{W}$ being a mono, together with the uniqueness property of $[i_\mathcal{W}, \delta_\mathcal{W}]^\rho$.



(1) definition of $[i, \delta]^\rho$
(2) Theorem 4.1.8

Now $i^\sharp_\mathcal{W} \circ [\eta_I, \theta_I]^\rho = [i_\mathcal{W}, \delta_\mathcal{W}]^\rho = e_\mathcal{W} \in \mathcal{E}$, so $i^\sharp_\mathcal{W} \in \mathcal{E}$ by [AHS09, Proposition 14.11 via duality]. Thus, $H_\mathcal{W}$ is reachable. □

Since the unique morphisms induced by a recursive coalgebra are essentially generalised reachability maps (see also [CUV06, Corollary 3]), we can accordingly define a generalised version of the language of an automaton.

Definition 4.3.4 (Generalised language)**.** Given a recursive coalgebra $\rho : S \longrightarrow F_I S$, we define for any automaton $\mathcal{A} = (Q, \delta, i, o)$ its *ρ-language* as the composition

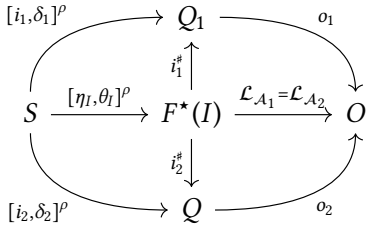
$$\mathcal{L}^{\rho}_{\mathcal{A}} = S \xrightarrow{[i,\delta]^{\rho}} Q \xrightarrow{o} O.$$

Next, we show that language equivalence of automata can be characterised via equivalence w.r.t. all generalised languages.

Proposition 4.3.5 (Language equivalence via recursion)**.** *Given automata $\mathcal{A}_1 = (Q_1, \delta_1, i_1, o_1)$ and $\mathcal{A}_2 = (Q_2, \delta_2, i_2, o_2)$, we have $\mathcal{L}_{\mathcal{A}_1} = \mathcal{L}_{\mathcal{A}_2}$ if and only if for all recursive $\rho : S \longrightarrow F_I S$ we have $\mathcal{L}^{\rho}_{\mathcal{A}_1} = \mathcal{L}^{\rho}_{\mathcal{A}_2}$.*

Proof. First assume that $\mathcal{L}^{\rho}_{\mathcal{A}_1} = \mathcal{L}^{\rho}_{\mathcal{A}_2}$ for all recursive $\rho : S \longrightarrow F_I S$. Note that $F^{\star}(I)$ is the initial algebra of F_I; thus $[\eta_I, \theta_I] : F_I F^{\star}(I) \longrightarrow F^{\star}(I)$ has an inverse. One easily sees that this inverse is recursive, with the corresponding unique maps into algebras being reachability maps. Thus, $\mathcal{L}_{\mathcal{A}_1} = o_1 \circ i_1^{\sharp} = o_2 \circ i_2^{\sharp} = \mathcal{L}_{\mathcal{A}_2}$.

Conversely, assume $\mathcal{L}_{\mathcal{A}_1} = \mathcal{L}_{\mathcal{A}_2}$. Given a recursive coalgebra $\rho : S \longrightarrow F_I S$, we have that $[i_1, \delta_1]^{\rho} = i_1^{\sharp} \circ [\eta_I, \theta_I]^{\rho}$ and $[i_2, \delta_2]^{\rho} = i_2^{\sharp} \circ [\eta_I, \theta_I]^{\rho}$ by uniqueness. Thus, the diagram below commutes.



□

The above result suggests an abstract notion of a witness of the case when two automata do not accept the same language: there is a recursive coalgebra witnessing the difference via the corresponding generalised languages. This will be particularly useful in describing counterexamples for hypotheses, which we thus formalise as follows.

Definition 4.3.6 (Counterexample)**.** Given an automaton $\mathcal{A} = (Q, \delta, i, o)$, a wrapper $\big(S - \mathsf{sel} \to Q, Q - \mathsf{cla} \to P\big)$ that is $\mathcal{A}$-closed and $\mathcal{A}$-consistent is said to be *$\mathcal{A}$-correct up to* a recursive $\rho : S' \longrightarrow F_I S'$ if $\mathcal{L}^{\rho}_{\mathcal{H}_{\mathcal{W}}} = \mathcal{L}^{\rho}_{\mathcal{A}}$. An *$\mathcal{A}$-counterexample* for $\mathcal{W}$ (or $\mathcal{H}_{\mathcal{W}}$) is a recursive $\rho : S' \longrightarrow F_I S'$ such that $\mathcal{W}$ is not $\mathcal{A}$-correct up to ρ.

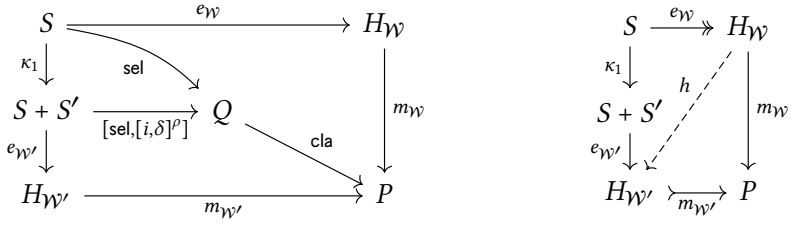
Proposition 4.3.7 (Counterexample existence)**.** *Given an automaton $\mathcal{A} = (Q, \delta, i, o)$ and a wrapper $\left(S - \mathsf{sel} \to Q, Q - \mathsf{cla} \to P\right)$ that is $\mathcal{A}$-closed and $\mathcal{A}$-consistent, we have $\mathcal{L}_{H_\mathcal{W}} \neq \mathcal{L}_\mathcal{A}$ if and only if there exists an $\mathcal{A}$-counterexample for $\mathcal{W}$.*

Proof. Follows directly from Proposition 4.3.5. □

Given a counterexample, the algorithm should adjust its wrapper to accommodate the new information. Below we show that if after adding a recursive coalgebra to the wrapper, the resulting wrapper is closed and consistent, then the coalgebra was not a counterexample.

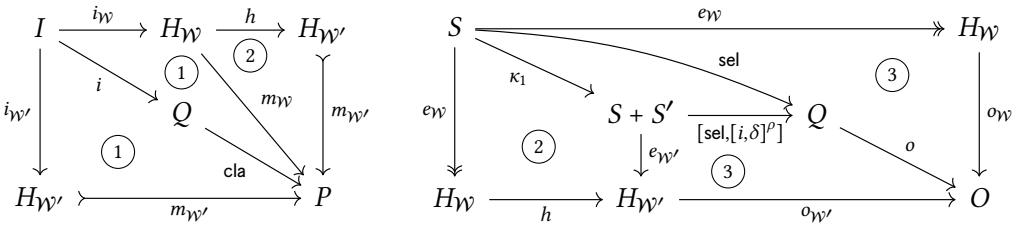
Theorem 4.3.8 (Correctness up to via closedness and consistency)**.** *Consider an automaton $\mathcal{A} = (Q, \delta, i, o)$ and a wrapper $\mathcal{W} = \left(S - \mathsf{sel} \to Q, Q - \mathsf{cla} \to P\right)$ that is $\mathcal{A}$-closed and $\mathcal{A}$-consistent. For any recursive coalgebra $\rho : S' \longrightarrow F_I S'$ such that the wrapper $([\mathsf{sel}, [i, \delta]^\rho], \mathsf{cla})$ is $\mathcal{A}$-closed and $\mathcal{A}$-consistent, we have that $\mathcal{W}$ is $\mathcal{A}$-correct up to ρ.*

Proof. Let $\mathcal{W}' = ([\mathsf{sel}, [i, \delta]^\rho], \mathsf{cla})$. Since the diagram below on the left commutes, we obtain a unique diagonal $h : H_\mathcal{W} \longrightarrow H_{\mathcal{W}'}$ on the right.

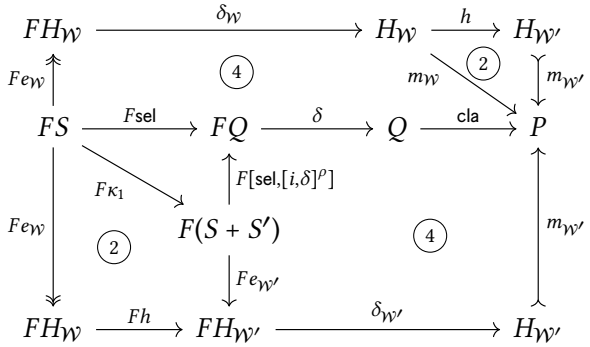


We will show that h is an automaton homomorphism, namely that it commutes with the initial states ($h \circ i_\mathcal{W} = i_{\mathcal{W}'}$), outputs ($o_{\mathcal{W}'} \circ h = o_\mathcal{W}$), and dynamics ($h \circ \delta_\mathcal{W} = \delta_{\mathcal{W}'} \circ Fh$). Noting that

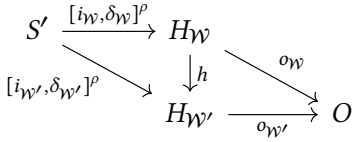
$m_{\mathcal{W}'}$ is a mono and $e_{\mathcal{W}}$ is an epi, this follows from commutativity of the diagrams below.



(1) $\mathcal{A}$-closedness (2) definition of h (3) $\mathcal{A}$-consistency (4) Theorem 4.1.8

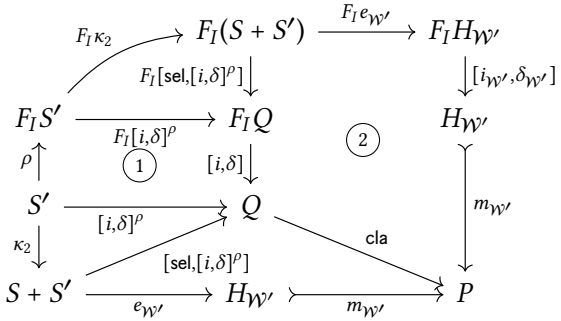


The fact that h is an automaton homomorphism $\mathcal{H}_{\mathcal{W}} \longrightarrow \mathcal{H}_{\mathcal{W}'}$ implies in particular that $h \circ [i_{\mathcal{W}}, \delta_{\mathcal{W}}]^\rho = [i_{\mathcal{W}'}, \delta_{\mathcal{W}'}]^\rho$. It follows that the diagram below commutes.



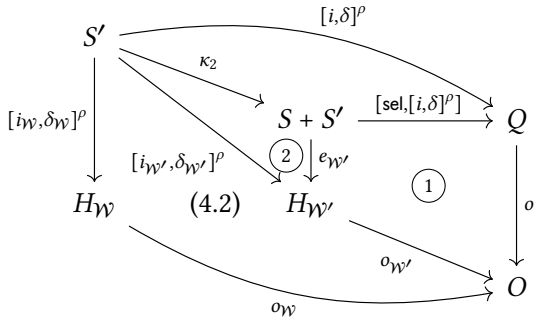
(4.2)

We now show that $e_{\mathcal{W}'} \circ \kappa_2 = [i_{\mathcal{W}'}, \delta_{\mathcal{W}'}]^\rho$. This follows by the uniqueness property of $[i_{\mathcal{W}'}, \delta_{\mathcal{W}'}]^\rho$ from commutativity of the diagram below, using that $m_{\mathcal{W}'}$ is monic.



(1) definition of $[i, \delta]^\rho$
(2) Theorem 4.1.8

The commutative diagram below completes the proof.



(1) consistency
(2) previous observation

□

Equivalently, the above shows that adding the counterexample information will lead to either a closedness or a consistency defect.

Corollary 4.3.9 (Counterexample progress)**.** *Given an automaton $\mathcal{A} = (Q, \delta, i, o)$, a wrapper $\left(S - \mathsf{sel} \to Q, Q - \mathsf{cla} \to P\right)$ that is $\mathcal{A}$-closed and $\mathcal{A}$-consistent, and a recursive $\rho : S' \longrightarrow F_I S'$ such that ρ is an $\mathcal{A}$-counterexample for $\mathcal{W}$, we have that $([\mathsf{sel}, [i, \delta]^\rho], \mathsf{cla})$ is either not $\mathcal{A}$-closed or not $\mathcal{A}$-consistent.*

4.4 An Abstract Automata Learning Algorithm

We can now describe our general algorithm, for which we fix a *target* automaton $\mathcal{A}_\mathsf{t} = (Q_\mathsf{t}, \delta_\mathsf{t}, i_\mathsf{t}, o_\mathsf{t})$ throughout this section. Similarly to $\mathtt{L}^\star$ (Section 2.3), the algorithm is organised into two procedures: Algorithm 4.1, which contains the abstract procedure for making a wrapper closed and consistent, and Algorithm 4.2, containing the learning iterations. These generalise the analogous procedures in $\mathtt{L}^\star$, Algorithm 2.1 and Algorithm 2.2, respectively.

The procedure in Algorithm 4.1 assumes that for each wrapper $\mathcal{W} = \left(S - \mathsf{sel} \to Q_\mathsf{t}, Q_\mathsf{t} - \mathsf{cla} \to P\right)$ there exists $\mathsf{cla}' : Q_\mathsf{t} \longrightarrow P'$ such that $(\mathsf{sel}, \mathsf{cla}')$ is locally $\mathcal{A}$-consistent w.r.t. cla; for local closedness we do not need this assumption, thanks to Proposition 4.2.4. We use local closedness and local consistency in the steps in lines 4 and 6 to fix closedness and consistency defects. We will see later in Section 4.6 that for a large class of functors in **Set**, existence of these maps ensuring local consistency can be proved.

In Algorithm 4.2, the wrapper is initialised with trivial maps and extended to be closed and consistent using the subroutine Fix (line 1). The main loop constructs the corresponding hypothesis and poses and equivalence query for it, denoted by EQ (line 2). If this equivalence query returns a counterexample in the form of a recursive coalgebra, it is used to update the wrapper (line 3). Note that instead of concretely updating sel to become $[\mathsf{sel}, [i_\mathsf{t}, \delta_\mathsf{t}]^\rho]$ we only

Algorithm 4.1. Make wrapper $\mathcal{A}_t$-closed and $\mathcal{A}_t$-consistent

1: **function** Fix(sel, cla)
2: **while** (sel, cla) is not $\mathcal{A}_t$-closed or not $\mathcal{A}_t$-consistent **do**
3: **if** (sel, cla) is not $\mathcal{A}_t$-closed **then**
4: sel $\longleftarrow$ sel$'$ such that (sel$'$, cla) is locally $\mathcal{A}_t$-closed w.r.t. sel
5: **else if** (sel, cla) is not $\mathcal{A}_t$-consistent **then**
6: cla $\longleftarrow$ cla$'$ such that (sel, cla$'$) is locally $\mathcal{A}_t$-consistent w.r.t. cla
7: **return** sel, cla

Algorithm 4.2. Abstract automata learning algorithm

1: $\mathsf{sel}, \mathsf{cla} \longleftarrow \textsc{Fix}(! : 0 \longrightarrow Q_t, ! : Q_t \longrightarrow 1)$
2: **while** $\mathsf{EQ}(\mathcal{H}_{(\mathsf{sel},\mathsf{cla})}) = \rho : S \longrightarrow F_I S$ **do**
3: $\mathsf{sel} \longleftarrow \mathsf{sel}'$ such that $\mathsf{sel}'^{\triangleleft} = [\mathsf{sel}, [i_t, \delta_t]^{\rho}]^{\triangleleft}$
4: $\mathsf{sel}, \mathsf{cla} \longleftarrow \textsc{Fix}(\mathsf{sel}, \mathsf{cla})$
5: **return** $\mathcal{H}_{(\mathsf{sel},\mathsf{cla})}$

Figure 4.1: Generalised Learning Algorithm.

require the $\mathcal{M}$ parts of their factorisations to be the same, which leaves room for optimisations that avoid adding the exact same information multiple times. The updated wrapper is then again passed on to the subroutine Fix (line 4) to be made closed and consistent. If the equivalence query instead is successful, we return the hypothesis (line 5).

We note that the algorithm references the automaton $\mathcal{A}_t$, which is not known before termination. However, note that we only need this automaton to describe the definitions of closedness and consistency. We will see that in concrete instances of the algorithm these definitions actually depend only on the language accepted by $\mathcal{A}_t$.

We now define a run of the algorithm as a stream of wrappers, each of which corresponds to the wrapper held by the algorithm at a given point in time during a possible execution. This will give us a formal object to use for reasoning about termination.

Definition 4.4.1 (Run of the algorithm)**.** A *run* of the algorithm is a family of wrappers

$$\{\mathcal{W}_n = (S_n \xrightarrow{-\mathsf{sel}_n \to} Q_t, Q_t \xrightarrow{-\mathsf{cla}_n \to} P_n)\}_{n \in \mathbb{N}}$$

satisfying the following conditions:

1. $\mathsf{sel}_0 : 0 \longrightarrow Q_\mathsf{t}$ and $\mathsf{cla}_0 : Q_\mathsf{t} \longrightarrow 1$ are the unique morphisms;

2. if $\mathcal{W}_n$ is not $\mathcal{A}_\mathsf{t}$-closed, then $\mathsf{cla}_{n+1} = \mathsf{cla}_n$ and sel_{n+1} is such that $(\mathsf{sel}_{n+1}, \mathsf{cla}_n)$ is locally $\mathcal{A}_\mathsf{t}$-closed w.r.t. sel_n;

3. if $\mathcal{W}_n$ is $\mathcal{A}_\mathsf{t}$-closed but not $\mathcal{A}_\mathsf{t}$-consistent, then $\mathsf{sel}_{n+1} = \mathsf{sel}_n$ and cla_{n+1} is s.t. $(\mathsf{sel}_n, \mathsf{cla}_{n+1})$ is locally $\mathcal{A}_\mathsf{t}$-consistent w.r.t. cla_n;

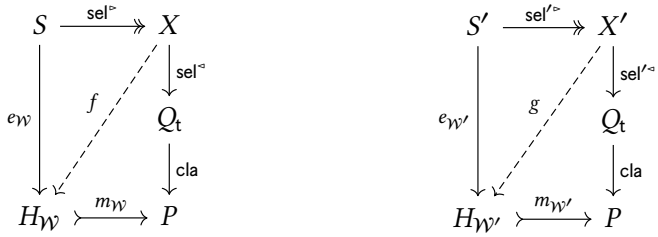
4. if $\mathcal{W}_n$ is $\mathcal{A}_\mathsf{t}$-closed and $\mathcal{A}_\mathsf{t}$-consistent and we obtain through an equivalence query for $\mathcal{H}_{\mathcal{W}_n}$ a counterexample $\rho : S \longrightarrow F_I S$ for $\mathcal{W}_n$, then $\mathsf{sel}^\triangleleft_{n+1} = [\mathsf{sel}_n, [i_\mathsf{t}, \delta_\mathsf{t}]^\rho]^\triangleleft$ and $\mathsf{cla}_{n+1} = \mathsf{cla}_n$; and

5. if $\mathcal{W}_n$ is $\mathcal{A}_\mathsf{t}$-closed and $\mathcal{A}_\mathsf{t}$-consistent and $\mathcal{A}_\mathsf{t}$-correct up to all recursive F_I-coalgebras, then $\mathcal{W}_{n+1} = \mathcal{W}_n$.

One easily sees that the definition above follows exactly the way the algorithm is defined. To prove that termination corresponds to convergence of runs of the algorithm, however, we need a technical result first.

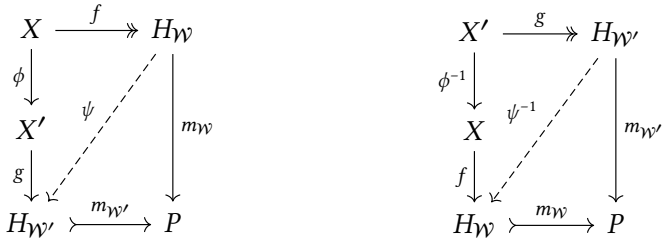
Lemma 4.4.2. *Let* $\mathsf{sel} : S \longrightarrow Q_\mathsf{t}$, $\mathsf{sel}' : S' \longrightarrow Q_\mathsf{t}$, *and* $\mathsf{cla} : Q_\mathsf{t} \longrightarrow P$ *be such that* $\mathsf{sel}^\triangleleft$ *and* $\mathsf{sel}'^\triangleleft$ *are isomorphic subobjects. If* $(\mathsf{sel}, \mathsf{cla})$ *is* $\mathcal{A}_\mathsf{t}$*-closed and* $\mathcal{A}_\mathsf{t}$*-consistent, then so is* $(\mathsf{sel}', \mathsf{cla})$.

Proof. Write $\mathcal{W} = (\mathsf{sel}, \mathsf{cla})$ and $\mathcal{W}' = (\mathsf{sel}', \mathsf{cla})$. Let X and X' be the respective objects through which sel and sel' factorise, and denote by $\phi : X \longrightarrow X'$ the subobject isomorphism ($\mathsf{sel}'^\triangleleft \circ \phi = \mathsf{sel}^\triangleleft$). We define $f : X \longrightarrow H_\mathcal{W}$ and $g : X' \longrightarrow H_{\mathcal{W}'}$ as the unique diagonals in the diagrams below.



Note that $\mathsf{sel}^\triangleright$ and $e_\mathcal{W}$ are in $\mathcal{E}$, and therefore so is f; similarly, since $\mathsf{sel}'^\triangleright$ and $e_{\mathcal{W}'}$ are in $\mathcal{E}$, so is g [AHS09, Proposition 14.9 via duality]. We now define $\psi : H_\mathcal{W} \longrightarrow H_{\mathcal{W}'}$ and $\psi^{-1} : H_\mathcal{W} \longrightarrow$

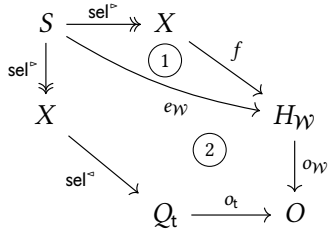
$H_{\mathcal{W}'}$ as the unique diagonals in the diagrams below.



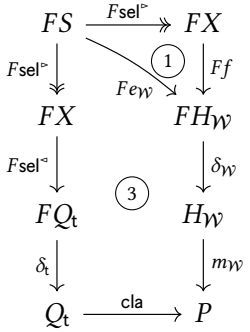
It is a standard result that ψ and ψ^{-1} are inverse to each other [AHS09, Proposition 14.7], as suggested by their names. To show closedness and consistency, we will need the following two equations.

$$o_{\mathcal{W}} \circ f = o_{\mathsf{t}} \circ \mathsf{sel}^{\triangleleft} \qquad\qquad m_{\mathcal{W}} \circ \delta_{\mathcal{W}} \circ Ff = \mathsf{cla} \circ \delta_{\mathsf{t}} \circ F\mathsf{sel}^{\triangleleft}. \tag{4.3}$$

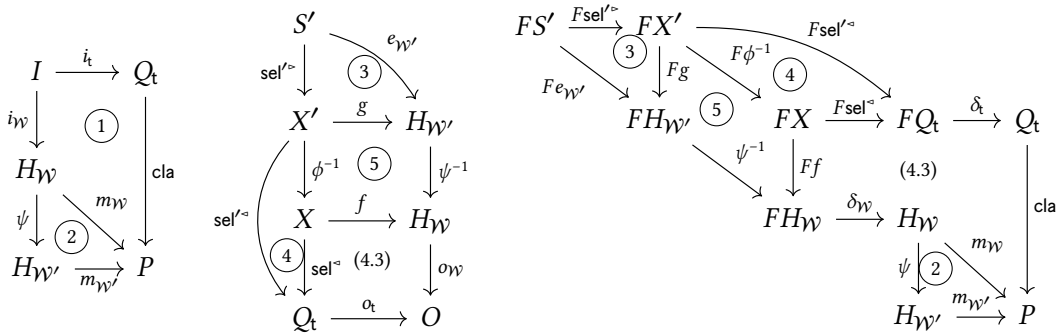
Note that both $\mathsf{sel}^{\triangleright}$ and $F\mathsf{sel}^{\triangleright}$ are in $\mathcal{E}$ because F preserves morphisms in $\mathcal{E}$, and that they are therefore both epis. We use this to prove (4.3) with the commutative diagrams below.



① definition of f
② consistency
③ Theorem 4.1.8



Now the diagrams below commute.



① closedness
② definition of ψ
③ definition of g
④ subobject morphism
⑤ definition of ψ^{-1}

It follows from Theorem 4.1.8 that $\mathcal{W}'$ is $\mathcal{A}_{\mathsf{t}}$-closed and $\mathcal{A}_{\mathsf{t}}$-consistent. □

We can now prove the following correspondence.

Proposition 4.4.3. *Algorithm 4.2 terminates if and only if for all runs* $\{\mathcal{W}_n\}_{n\in\mathbb{N}}$ *there exists* $n \in \mathbb{N}$ *such that* $\mathcal{W}_{n+1} = \mathcal{W}_n$.

Proof. Note that a possible execution of the algorithm corresponds precisely to a run. Thus, if Algorithm 4.2 terminates, then clearly $\mathcal{W}_{n+1} = \mathcal{W}_n$ for all runs $\{\mathcal{W}_n\}_{n\in\mathbb{N}}$, as provided by clause 5 in the definition of a run.

Conversely, we need to show that clause 5 is the only way from which $\mathcal{W}_{n+1} = \mathcal{W}_n$ can be achieved. Thus, if for some $n \in \mathbb{N}$ the wrapper is updated due to a closedness or consistency issue or due to a counterexample, we need to show that $\mathcal{W}_{n+1} \neq \mathcal{W}_n$. Suppose towards a contradiction that $\mathcal{W}_{n+1} = \mathcal{W}_n$. We will derive a contradiction for each of the three cases.

Suppose $\mathcal{W}_n$ is not $\mathcal{A}_\mathrm{t}$-closed. From the definition of a run we obtain that $\mathcal{W}_{n+1}$ is locally $\mathcal{A}_\mathrm{t}$-closed w.r.t. sel_n. Thus, $\mathcal{W}_n$ is locally $\mathcal{A}_\mathrm{t}$-closed w.r.t. sel_n, which equivalently provides the contradiction that $\mathcal{W}_n$ is $\mathcal{A}_\mathrm{t}$-closed.

Suppose $\mathcal{W}_n$ is $\mathcal{A}_\mathrm{t}$-closed but not $\mathcal{A}_\mathrm{t}$-consistent. Again from the definition of a run we obtain that $\mathcal{W}_{n+1}$ is locally $\mathcal{A}_\mathrm{t}$-consistent w.r.t. cla_n. Thus, $\mathcal{W}_n$ is locally $\mathcal{A}_\mathrm{t}$-consistent w.r.t. cla_n, which equivalently provides the contradiction that $\mathcal{W}_n$ is $\mathcal{A}_\mathrm{t}$-consistent.

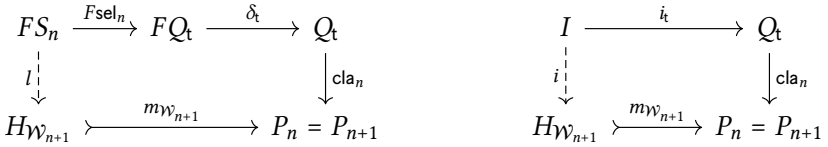
Suppose $\mathcal{W}_n$ is $\mathcal{A}_\mathrm{t}$-closed and $\mathcal{A}_\mathrm{t}$-consistent and we obtain a counterexample $\rho : S \longrightarrow F_I S$ for $\mathcal{W}_n$. By the definition of a run we have $\mathsf{sel}_{n+1}^{\triangleleft} = [\mathsf{sel}_n, [i_\mathrm{t}, \delta_\mathrm{t}]^\rho]^{\triangleleft}$, so $\mathsf{sel}_n^{\triangleleft} = [\mathsf{sel}_n, [i_\mathrm{t}, \delta_\mathrm{t}]^\rho]^{\triangleleft}$. By Lemma 4.4.2 this implies that $([\mathsf{sel}_n, [i_\mathrm{t}, \delta_\mathrm{t}]^\rho], \mathsf{cla}_n)$ is also $\mathcal{A}_\mathrm{t}$-closed and $\mathcal{A}_\mathrm{t}$-consistent, which by Corollary 4.3.9 contradicts the fact that ρ is a counterexample for $\mathcal{W}_n$. □

To prove termination we will need an invariant on the way successive wrapper selectors and classifiers are ordered (see Section 2.2.1). This invariant will be provided in Lemma 4.4.6, which first requires two additional lemmas.

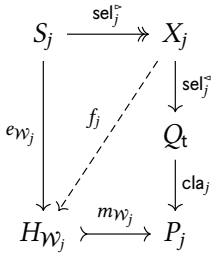
Lemma 4.4.4. *For any run* $\{\mathcal{W}_n = (\mathsf{sel}_n, \mathsf{cla}_n)\}_{n\in\mathbb{N}}$ *and* $n \in \mathbb{N}$, *if* $\mathsf{sel}_{n+1}^{\triangleleft} \leq \mathsf{sel}_n^{\triangleleft}$, *then* $\mathcal{W}_n$ *is* $\mathcal{A}_\mathrm{t}$*-closed.*

Proof. For each $j \in \mathbb{N}$, denote by S_j the domain of sel_j and by P_j the codomain of cla_j, and let X_j be the object through which sel_j factorises. Assume towards a contradiction that $\mathcal{W}_n$ is not $\mathcal{A}_\mathrm{t}$-closed. By the definition of a run we then have that $\mathsf{cla}_{n+1} = \mathsf{cla}_n$ and $\mathcal{W}_{n+1}$ is locally $\mathcal{A}_\mathrm{t}$-closed w.r.t. sel_n. This means that there exist morphisms $l : FS_n \longrightarrow H_{n+1}$ and $i : I \longrightarrow H_{\mathcal{W}_{n+1}}$

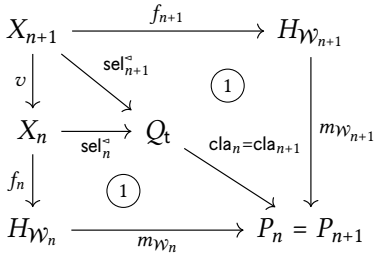
making the diagrams below commute.



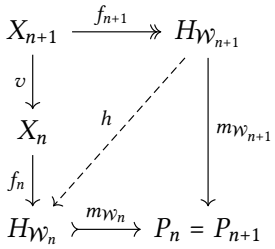
We define for every $j \in \mathbb{N}$ the morphism $f_j : X_n \longrightarrow H_n$ as the unique diagonal in the commutative square below.



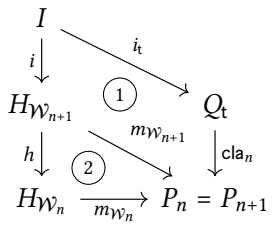
Note that $f_j \in \mathcal{E}$ because $e_{\mathcal{W}_j} \in \mathcal{E}$. Thus, we can define $h : H_{\mathcal{W}_{n+1}} \longrightarrow H_{\mathcal{W}_n}$ as the unique diagonal in the commutative diagram below, where we write $v : X_{n+1} \longrightarrow X_n$ for the witness of $\mathsf{sel}_{n+1}^{\triangleleft} \leq \mathsf{sel}_n^{\triangleleft}$.



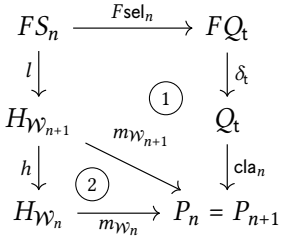
(1) definition of f_n or f_{n+1}



Now the diagrams below commute, leading to the desired contradiction that $\mathcal{W}_n$ is $\mathcal{A}_\mathsf{t}$-closed.



(1) local closedness
(2) definition of h

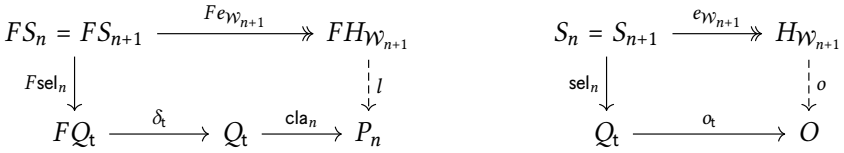


□

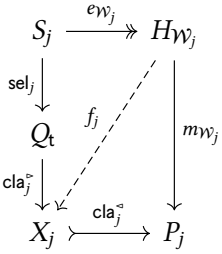
Lemma 4.4.5. *For any run* $\{\mathcal{W}_n = (\mathsf{sel}_n, \mathsf{cla}_n)\}_{n\in\mathbb{N}}$ *and* $n \in \mathbb{N}$*, if* $\mathcal{W}_n$ *is* $\mathcal{A}_\mathsf{t}$*-closed and* $\mathsf{cla}_n^{\triangleright} \leq \mathsf{cla}_{n+1}^{\triangleright}$*, then* $\mathcal{W}_n$ *is* $\mathcal{A}_\mathsf{t}$*-consistent.*

Proof. For each $j \in \mathbb{N}$, denote by S_j the domain of sel_j and by P_j the codomain of cla_j, and let X_j be the object through which cla_j factorises. Assume towards a contradiction that $\mathcal{W}_n$ is not $\mathcal{A}_\mathsf{t}$-consistent. By the definition of a run of the algorithm we then have that $\mathsf{sel}_{n+1} = \mathsf{sel}_n$ and $\mathcal{W}_{n+1}$

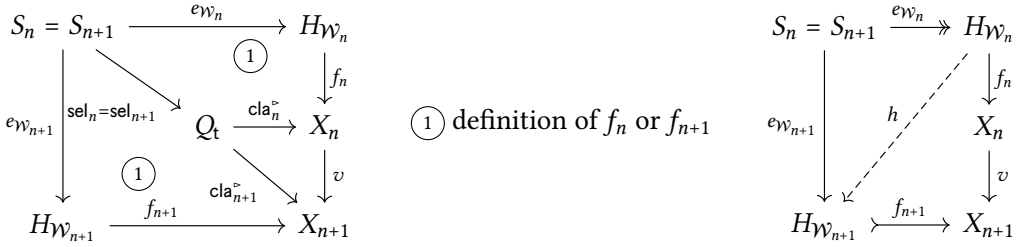
is locally $\mathcal{A}_\mathsf{t}$-consistent w.r.t. cla_n. This means that there exist morphisms $l : FH\mathcal{W}_n \longrightarrow P_{n+1}$ and $o : H\mathcal{W}_{n+1} \longrightarrow O$ making the diagrams below commute.



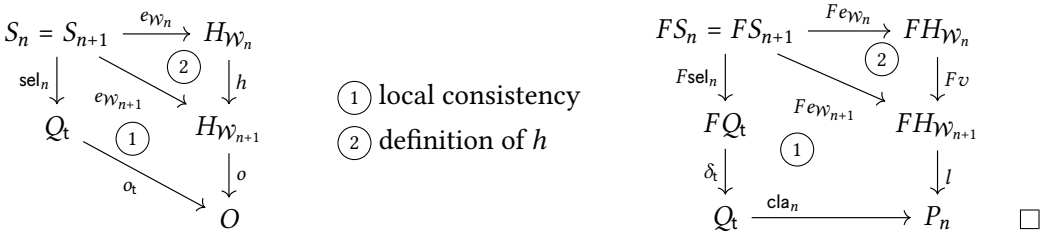
We define for every $j \in \mathbb{N}$ the morphism $f_j : H\mathcal{W}_j \longrightarrow X_j$ as the unique diagonal in the commutative square below.



Note that $f_j \in \mathcal{M}$ because $m\mathcal{W}_j \in \mathcal{M}$. Thus, we can define $h : H\mathcal{W}_n \longrightarrow H\mathcal{W}_{n+1}$ as the unique diagonal in the commutative diagram below, where we write $v : X_n \longrightarrow X_{n+1}$ for the witness of $\mathsf{cla}_n^{\triangleright} \leq \mathsf{cla}_{n+1}^{\triangleright}$.



Now the diagrams below commute, leading to the desired contradiction that $\mathcal{W}_n$ is $\mathcal{A}_\mathsf{t}$-consistent.



We now introduce the invariant concerning the relations between successive wrappers in a run.

Lemma 4.4.6. *Consider a run* $\{\mathcal{W}_n = (\mathsf{sel}_n, \mathsf{cla}_n)\}_{n\in\mathbb{N}}$ *and* $n \in \mathbb{N}$. *We have* $\mathsf{sel}_n^{\triangleleft} \leq \mathsf{sel}_{n+1}^{\triangleleft}$ *and* $\mathsf{cla}_{n+1}^{\triangleright} \leq \mathsf{cla}_n^{\triangleright}$ *for all* $n \in \mathbb{N}$. *Moreover, if* $\mathsf{sel}_{n+1}^{\triangleleft} \leq \mathsf{sel}_n^{\triangleleft}$, *then* $\mathsf{sel}_{n+1} = \mathsf{sel}_n$; *if* $\mathsf{cla}_n^{\triangleright} \leq \mathsf{cla}_{n+1}^{\triangleright}$, *then* $\mathsf{cla}_{n+1} = \mathsf{cla}_n$.

Proof. We consider each of the cases listed in the definition of a run of the algorithm. If $\mathcal{W}_n$ is not $\mathcal{A}_\mathsf{t}$-closed, then $\mathsf{cla}_{n+1} = \mathsf{cla}_n$ and $\mathsf{sel}_n^{\triangleleft} \leq \mathsf{sel}_{n+1}^{\triangleleft}$ by the definition of local closedness. Supposing $\mathsf{sel}_{n+1}^{\triangleleft} \leq \mathsf{sel}_n^{\triangleleft}$ leads by Lemma 4.4.4 to the contradiction that $\mathcal{W}_n$ is $\mathcal{A}_\mathsf{t}$-closed.

If $\mathcal{W}_n$ is $\mathcal{A}_\mathsf{t}$-closed but not $\mathcal{A}_\mathsf{t}$-consistent, then $\mathsf{sel}_{n+1} = \mathsf{sel}_n$ and we have $\mathsf{cla}_{n+1}^{\triangleright} \leq \mathsf{cla}_n^{\triangleright}$ by the definition of local consistency. Supposing $\mathsf{cla}_n^{\triangleright} \leq \mathsf{cla}_{n+1}^{\triangleright}$ leads by Lemma 4.4.5 to the contradiction that $\mathcal{W}_n$ is $\mathcal{A}_\mathsf{t}$-consistent.

If $\mathcal{W}_n$ is $\mathcal{A}_\mathsf{t}$-closed and $\mathcal{A}_\mathsf{t}$-consistent and we obtain a counterexample $\rho : S \longrightarrow F_I S$ for $\mathcal{W}_n$, then $\mathsf{cla}_{n+1} = \mathsf{cla}_n$ and $\mathsf{sel}_{n+1}^{\triangleleft} = [\mathsf{sel}_n, [i_\mathsf{t}, \delta_\mathsf{t}]^\rho]^{\triangleleft}$. We have $\mathsf{sel}_n^{\triangleleft} \leq [\mathsf{sel}_n, [i_\mathsf{t}, \delta_\mathsf{t}]^\rho]^{\triangleleft}$ using Lemma 4.2.3. Suppose $\mathsf{sel}_{n+1}^{\triangleleft} \leq \mathsf{sel}_n^{\triangleleft}$. Then $[\mathsf{sel}_n, [i_\mathsf{t}, \delta_\mathsf{t}]^\rho]^{\triangleleft} = \mathsf{sel}_{n+1}^{\triangleleft} \leq \mathsf{sel}_n^{\triangleleft}$, so $\mathsf{sel}_n^{\triangleleft}$ and $[\mathsf{sel}_n, [i_\mathsf{t}, \delta_\mathsf{t}]^\rho]^{\triangleleft}$ are isomorphic subobjects. By Lemma 4.4.2 this implies that $([\mathsf{sel}_n, [i_\mathsf{t}, \delta_\mathsf{t}]^\rho], \mathsf{cla}_n)$ is also $\mathcal{A}_\mathsf{t}$-closed and $\mathcal{A}_\mathsf{t}$-consistent, which by Corollary 4.3.9 contradicts the fact that ρ is a counterexample for $\mathcal{W}_n$.

If $\mathcal{W}_n$ is $\mathcal{A}_\mathsf{t}$-closed and $\mathcal{A}_\mathsf{t}$-consistent and correct up to all recursive F_I-coalgebras, then we immediately have $\mathsf{sel}_{n+1} = \mathsf{sel}_n$ and $\mathsf{cla}_{n+1} = \mathsf{cla}_n$. □

Putting the above results together, we obtain the following theorem showing that the algorithm terminates. Moreover, it necessarily terminates with a correct automaton, for which we give conditions that guarantee minimality.

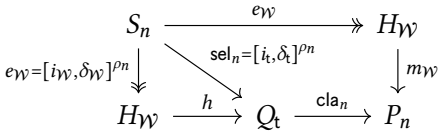
Theorem 4.4.7 (Termination)**.** *If* Q_t *has finitely many subobject and quotient isomorphism classes, then for all runs* $\{\mathcal{W}_n = (\mathsf{sel}_n, \mathsf{cla}_n)\}_{n\in\mathbb{N}}$ *there exists* $n \in \mathbb{N}$ *such that* $\mathcal{W}_n$ *is closed and consistent and the corresponding hypothesis is correct. If* $\mathcal{A}_\mathsf{t}$ *is minimal and for all* $k \in \mathbb{N}$ *there exists a recursive* $\rho_k : S_k \longrightarrow F_I S_k$ *such that* $\mathsf{sel}_k = [i_\mathsf{t}, \delta_\mathsf{t}]^{\rho_k}$, *then the final hypothesis is minimal.*

Proof. We will show that $\{\mathcal{W}_n\}_{n\in\mathbb{N}}$ converges, for which it suffices to show that both $\{\mathsf{sel}_n\}_{n\in\mathbb{N}}$ and $\{\mathsf{cla}_n\}_{n\in\mathbb{N}}$ converge. Suppose $\{\mathsf{sel}_n\}_{n\in\mathbb{N}}$ does not converge. By Lemma 4.4.6 there exist $i_n \in \mathbb{N}$ for all $n \in \mathbb{N}$ such that $i_{n+1} > i_n$, $\mathsf{sel}_{i_n}^{\triangleleft} \leq \mathsf{sel}_{i_{n+1}}^{\triangleleft}$, and $\mathsf{sel}_{i_n+1}^{\triangleleft} \not\leq \mathsf{sel}_{i_n}^{\triangleleft}$ for all $n \in \mathbb{N}$. Note that isomorphic subobjects are ordered in both directions. Using transitivity of the order on subobjects we know that for all $m, n \in \mathbb{N}$ with $m \neq n$ we have that $\mathsf{sel}_{i_m}^{\triangleleft}$ and $\mathsf{sel}_{i_n}^{\triangleleft}$ are not isomorphic subobjects. This contradicts the fact that Q_t has finitely many subobject isomorphism classes. Thus, $\{\mathsf{sel}_n\}_{n\in\mathbb{N}}$ must converge.

Now suppose $\{\mathsf{cla}_n\}_{n\in\mathbb{N}}$ does not converge. By Lemma 4.4.6 there exist $i_n \in \mathbb{N}$ for all $n \in \mathbb{N}$ such that $i_{n+1} > i_n$, $\mathsf{cla}_{i_n+1}^{\triangleright} \leq \mathsf{cla}_{i_n}^{\triangleright}$, and $\mathsf{cla}_{i_n}^{\triangleright} \not\leq \mathsf{cla}_{i_n+1}^{\triangleright}$ for all $n \in \mathbb{N}$. Note that iso-

morphic quotients are ordered in both directions. Using transitivity of the order on quotients we know that for all $m, n \in \mathbb{N}$ with $m \neq n$ the quotients $\mathsf{cla}_{i_m}^{\triangleleft}$ and $\mathsf{cla}_{i_n}^{\triangleleft}$ are not isomorphic. This contradicts the fact that Q_t has finitely many quotient isomorphism classes. Thus, $\{\mathsf{cla}_n\}_{n\in\mathbb{N}}$ must converge. We conclude that $\{\mathcal{W}_n\}_{n\in\mathbb{N}}$ converges, and by Proposition 4.4.3 the algorithm terminates. By definition, it does so with a correct hypothesis.

Now assume that $\mathcal{A}_\mathrm{t}$ is minimal and that for all $k \in \mathbb{N}$ there exists a recursive $\rho_k : S_k \longrightarrow F_I S_k$ such that $\mathsf{sel}_k = [i_\mathrm{t}, \delta_\mathrm{t}]^{\rho_k}$. Let $n \in \mathbb{N}$ be such that $\mathcal{W}_{n+1} = \mathcal{W}_n$, which by the above we know exists, and define $\mathcal{W} = \mathcal{W}_n$. We know from Proposition 4.3.3 that $\mathcal{H}_\mathcal{W}$ is reachable. Together with correctness of $\mathcal{H}_\mathcal{W}$ and minimality of $\mathcal{A}_\mathrm{t}$ there exists a unique automaton homomorphism $h : \mathcal{H}_\mathcal{W} \longrightarrow \mathcal{A}_\mathrm{t}$. We show that $\mathsf{cla}_n \circ h = m_\mathcal{W}$ with the commutative diagram below, where we precompose with the epi $e_\mathcal{W}$ and use that automaton homomorphisms commute with generalised reachability maps.

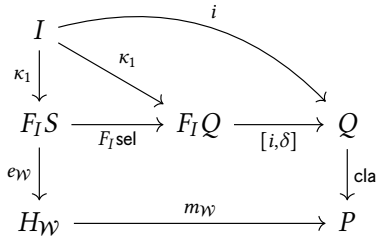


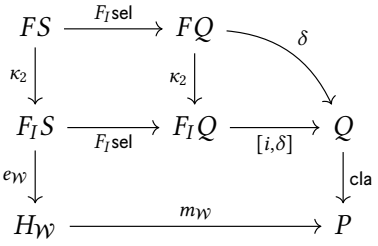
It follows that $h \in \mathcal{M}$ [AHS09, Proposition 14.11]. Being an automaton homomorphism, h commutes with the reachability maps: $h \circ i^\sharp_\mathcal{W} = i^\sharp_\mathrm{t}$. Because $i^\sharp_\mathcal{W} \in \mathcal{E}$ and $i^\sharp_\mathrm{t} \in \mathcal{E}$, we have $h \in \mathcal{E}$ [AHS09, Proposition 14.9 via duality]. Since $\mathcal{E} \cap \mathcal{M}$ contains only isomorphisms [AHS09, Proposition 14.6], it follows that h is an isomorphism of automata and therefore that the hypothesis is minimal. □

Let us discuss how the minimality condition in the above theorem can be satisfied. We need to make sure that for a run $\{\mathcal{W}_n = (\mathsf{sel}_n, \mathsf{cla}_n)\}_{n\in\mathbb{N}}$ of the algorithm we have for any $n \in \mathbb{N}$ that sel_n is *induced by a recursive coalgebra* (that is, there exists a recursive $\rho_n : S_n \longrightarrow F_I S_n$ such that $\mathsf{sel}_n = [i_\mathrm{t}, \delta_\mathrm{t}]^{\rho_n}$). To do so, we will give more concrete instructions to replace line 4 in Algorithm 4.1 and line 3 in Algorithm 4.2. Together with the fact that initially $\mathsf{sel}_0 : 0 \longrightarrow Q_\mathrm{t}$ is trivially induced by a (unique) recursive coalgebra, this will make it an invariant that the selector is induced by a recursive coalgebra. Regarding line 4 in Algorithm 4.1, suppose $\mathsf{sel} : S \longrightarrow Q_\mathrm{t}$ is such that there exists a recursive $\rho : S \longrightarrow F_I S$ satisfying $\mathsf{sel} = [i_\mathrm{t}, \delta_\mathrm{t}]^\rho$. We propose to choose $\mathsf{sel}' = [i_\mathrm{t}, \delta_\mathrm{t}] \circ F_I[i_\mathrm{t}, \delta_\mathrm{t}]^\rho$, which the following general result shows is also induced by a recursive coalgebra.

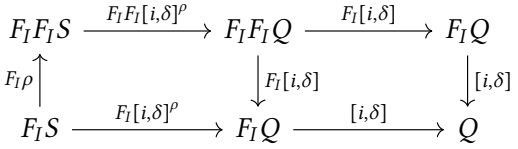
Proposition 4.4.8. *Given an automaton $\mathcal{A} = (Q, \delta, i, o)$, a recursive $\rho : S \longrightarrow F_I S$, and $\mathsf{cla} : Q \longrightarrow P$, the wrapper $([i, \delta] \circ F_I[i, \delta]^\rho, \mathsf{cla})$ is locally $\mathcal{A}$-closed w.r.t. $[i, \delta]^\rho$. Furthermore, $F_I\rho$ is also recursive and $[i, \delta] \circ F_I[i, \delta]^\rho = [i, \delta]^{F_I\rho}$.*

Proof. Let $\mathsf{sel} = [i, \delta]^\rho$ and $\mathcal{W} = ([i, \delta] \circ F_I \mathsf{sel}, \mathsf{cla})$. Note that $\mathsf{sel}^\blacktriangleleft \leq ([i, \delta] \circ F_I \mathsf{sel})^\blacktriangleleft$ by Lemma 4.2.3 (via ρ). Below we see that there exist morphisms $I \longrightarrow H_\mathcal{W}$ and $FS \longrightarrow H_\mathcal{W}$ satisfying the required commutativity conditions from Definition 4.2.1.





We know from [CUV06, Proposition 6] that $F_I\rho$ is recursive, so we have $[i, \delta] \circ F_I \mathsf{sel} = [i, \delta]^{F_I\rho}$ by uniqueness from commutativity of the diagram below.



□

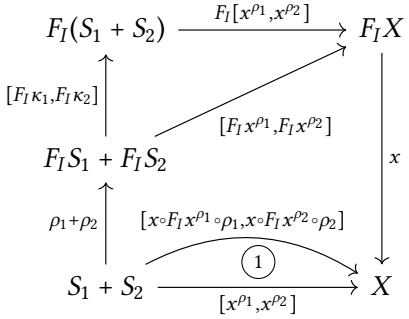
Regarding the counterexample handling in line 3 of Algorithm 4.2, we simply note that if $\mathsf{sel}_1 : S_1 \longrightarrow Q_\mathsf{t}$ and $\mathsf{sel}_2 : S_2 \longrightarrow Q_\mathsf{t}$ are induced by recursive coalgebras, then so is $[\mathsf{sel}_1, \mathsf{sel}_2]$ by the result below. Thus, given $\mathsf{sel} : S \longrightarrow Q_\mathsf{t}$ and the counterexample $\rho : S' \longrightarrow F_I S'$ we can choose $\mathsf{sel}' = [\mathsf{sel}, [i_\mathsf{t}, \delta_\mathsf{t}]^\rho] : S + S' \longrightarrow Q_\mathsf{t}$.

Proposition 4.4.9. *Given two recursive coalgebras $\rho_1 : S_1 \longrightarrow F_I S_1$ and $\rho_2 : S_2 \longrightarrow F_I S_2$, the composition*

$$\rho' = S_1 + S_2 \xrightarrow{\rho_1 + \rho_2} F_I S_1 + F_I S_2 \xrightarrow{[F_I\kappa_1, F_I\kappa_2]} F(S_1 + S_2)$$

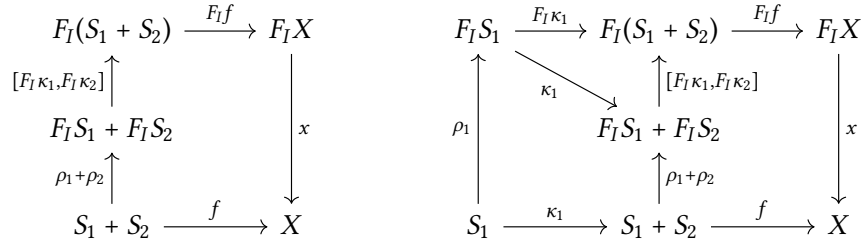
is also a recursive coalgebra, and for any algebra $x : F_I X \longrightarrow X$ we have $x^{\rho'} = [x^{\rho_1}, x^{\rho_2}]$.

Proof. Consider any algebra $x : F_I X \longrightarrow X$, and note that the diagram below commutes.



(1) definitions of x^{ρ_1} and x^{ρ_2}

Suppose $f : S_1 + S_2 \longrightarrow X$ is any morphism making the diagram below on the left commute.



Commutativity of the diagram on the right shows that $f \circ \kappa_1 = x^{\rho_1}$ by the uniqueness property of x^{ρ_1}. Analogously, we have $f \circ \kappa_2 = x^{\rho_2}$. Thus, $f = [x^{\rho_1}, x^{\rho_2}]$. □

In Section 4.6 we extensively detail an instantiation of our abstract algorithm for a large class of automata in **Set**. Furthermore, Chapter 6 will be based on a dualisation of the algorithm. First, however, we develop more general results that characterise which wrappers induce a correct hypothesis. As we will, these see apply to many more algorithms than just $\mathtt{L}^\star$.

4.5 Other Learning Algorithms and Minimisation

We have seen in this chapter that wrappers generalise the data structure used in the $\mathtt{L}^\star$ algorithm. In this section we will show that we can also use these structures to develop an abstract minimisation algorithm (Algorithm 4.3), which we prove correct in Theorem 4.5.6. We show that instances of this algorithm in the DFA setting are reachability analysis and merging equivalent states, and we derive variants of those that can in fact be seen as learning algorithms, one of which is known from the literature [Ang81]. In order to simplify the proofs in this section, we first generalise closedness and consistency to be relative to arbitrary algebras.

Definition 4.5.1 (Closedness and consistency relative to an algebra)**.** Given a wrapper $\mathcal{W} = \big(S - \mathsf{sel} \to X, X - \mathsf{cla} \to P\big)$, a functor $G : \mathbf{C} \longrightarrow \mathbf{C}$, and an algebra $x : GX \longrightarrow X$, we say that $\mathcal{W}$ is *x-closed* if there exists morphism $GS \longrightarrow H_\mathcal{W}$ making the diagram below on the left commute; it is *x-consistent* if there exists a morphism $GH_\mathcal{W} \longrightarrow P$ making the diagram on the right commute.

We leave the functor implicit, as it will be clear from the context.

We see that a wrapper $\mathcal{W} = \left(S - \mathsf{sel} \to Q, Q - \mathsf{cla} \to P\right)$ is $\mathcal{A}$-closed for an automaton $\mathcal{A} = (Q, \delta, i, o)$ if and only if it is i-closed and δ-closed; it is $\mathcal{A}$-consistent if and only if it is δ-consistent and $(\mathsf{cla}, \mathsf{sel})$ is o-closed as a wrapper in $\mathbf{C}^{\mathbf{op}}$. This correspondence allows us to recover the hypothesis result from Theorem 4.1.8 via multiple applications of the following core property.

Proposition 4.5.2. *Given a wrapper $\mathcal{W} = \left(S - \mathsf{sel} \to X, X - \mathsf{cla} \to P\right)$, a functor $G : \mathbf{C} \to \mathbf{C}$ that preserves morphisms in $\mathcal{E}$, and an algebra $x : GX \to X$, we have that $\mathcal{W}$ is x-closed and x-consistent if and only if there exists an algebra $a : GH_{\mathcal{W}} \to H_{\mathcal{W}}$ making the diagram below commute.*

$$\begin{array}{ccccc} GX & \xleftarrow{G\mathsf{sel}} & GS & \xrightarrow{Ge_{\mathcal{W}}} & GH_{\mathcal{W}} \\ {\scriptstyle x}\downarrow & & & & \downarrow{\scriptstyle a} \\ X & \xrightarrow{\mathsf{cla}} & P & \xleftarrow{m_{\mathcal{W}}} & H_{\mathcal{W}} \end{array}$$

Proof. First assume that $\mathcal{W}$ is x-closed and x-consistent. This means that there are morphisms $f : GS \to H_{\mathcal{W}}$ and $g : GH_{\mathcal{W}} \to P$ making the diagrams below commute.

$$\begin{array}{ccccc} GS & \xrightarrow{G\mathsf{sel}} & GX & \xrightarrow{x} & X \\ {\scriptstyle f}\downarrow & & & & \downarrow{\scriptstyle \mathsf{cla}} \\ H_{\mathcal{W}} & & \xrightarrow{m_{\mathcal{W}}} & & P \end{array} \qquad \begin{array}{ccccc} GS & & \xrightarrow{Ge_{\mathcal{W}}} & & GH_{\mathcal{W}} \\ {\scriptstyle G\mathsf{sel}}\downarrow & & & & \downarrow{\scriptstyle g} \\ GX & \xrightarrow{x} & X & \xrightarrow{\mathsf{cla}} & P \end{array}$$

We define $a : GH_{\mathcal{W}} \to H_{\mathcal{W}}$ as the unique diagonal in the commutative diagram below.

$$\begin{array}{ccc} GS & \xrightarrow{Ge_{\mathcal{W}}} & GH_{\mathcal{W}} \\ {\scriptstyle f}\downarrow & \overset{a}{\swarrow} & \downarrow{\scriptstyle g} \\ H_{\mathcal{W}} & \xrightarrow{m_{\mathcal{W}}} & P \end{array}$$

Conversely, assume there exists $a : GH_{\mathcal{W}} \to H_{\mathcal{W}}$ making the diagram in the statement commute. We define

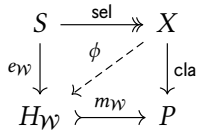
$$f = GS \xrightarrow{Ge_{\mathcal{W}}} GH_{\mathcal{W}} \xrightarrow{a} H_{\mathcal{W}} \qquad\qquad g = GH_{\mathcal{W}} \xrightarrow{a} H_{\mathcal{W}} \xrightarrow{m_{\mathcal{W}}} P$$

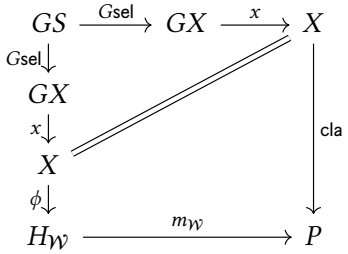
The required properties are satisfied precisely by the definition of a. □

Recall that a wrapper $\mathcal{W} = (S - \mathsf{sel} \to X, X - \mathsf{cla} \to P)$ intuitively can be seen as a two-sided approximation of X: sel selects a part of X while cla provides a classification of X. With this intuition we develop two properties that together imply the approximation is completely accurate: intuitively, we have selected everything if $\mathsf{sel} \in \mathcal{E}$, and the classification is faithful if $\mathsf{cla} \in \mathcal{M}$. We first show that the former implies closedness while the latter implies consistency.

Lemma 4.5.3. *Given a wrapper $\mathcal{W} = (S - \mathsf{sel} \to X, X - \mathsf{cla} \to P)$, a functor $G : \mathbf{C} \longrightarrow \mathbf{C}$, and an algebra $x : GX \longrightarrow X$, if $\mathsf{sel} \in \mathcal{E}$, then $\mathcal{W}$ is x-closed; if $\mathsf{cla} \in \mathcal{M}$, then $\mathcal{W}$ is x-consistent.*

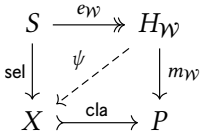
Proof. If $\mathsf{sel} \in \mathcal{E}$, we obtain a diagonal $\phi : X \longrightarrow H_{\mathcal{W}}$ in the commutative square below on the left.

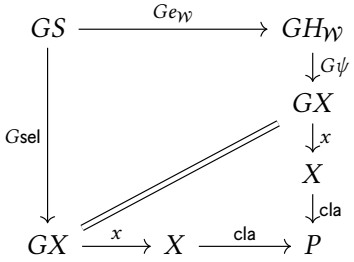




Then the diagram on the right commutes, which means $\mathcal{W}$ is x-closed.

If $\mathsf{cla} \in \mathcal{M}$, we obtain a diagonal $\psi : H_{\mathcal{W}} \longrightarrow X$ in the commutative square below on the left.

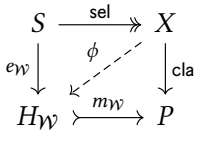


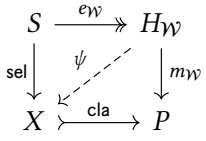


Now the diagram on the right commutes, which means $\mathcal{W}$ is x-consistent. □

Going a little further, if we select everything and satisfy consistency, or if we classify faithfully and satisfy closedness, then there exists a homorphism preserving the relevant structure between the hypothesis and the target.

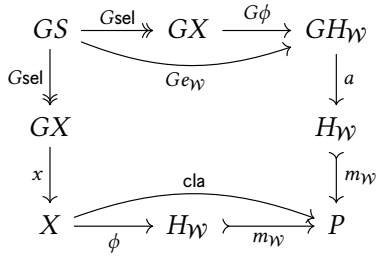
Proposition 4.5.4. *Given a wrapper $\mathcal{W} = (S - \mathsf{sel} \to X, X - \mathsf{cla} \to P)$, a functor $G : \mathbf{C} \longrightarrow \mathbf{C}$ that maps morphisms in $\mathcal{E}$ to epis, and an algebra $x : GX \longrightarrow X$, if $\mathsf{sel} \in \mathcal{E}$ and $\mathcal{W}$ is x-consistent, then $\mathcal{W}$ is also x-closed and the unique diagonal $\phi : X \longrightarrow H_{\mathcal{W}}$ in the diagram below on the left is an algebra homomorphism $(X, x) \longrightarrow (H_{\mathcal{W}}, a)$; if $\mathsf{cla} \in \mathcal{M}$ and $\mathcal{W}$ is x-closed, then $\mathcal{W}$ is also x-consistent and the unique diagonal $\psi : H_{\mathcal{W}} \longrightarrow X$ in the diagram below on the right is an algebra homomorphism $(H_{\mathcal{W}}, a) \longrightarrow (X, x)$. In both cases $a : GH_{\mathcal{W}} \longrightarrow H_{\mathcal{W}}$ is the algebra obtained via Proposition 4.5.2.*





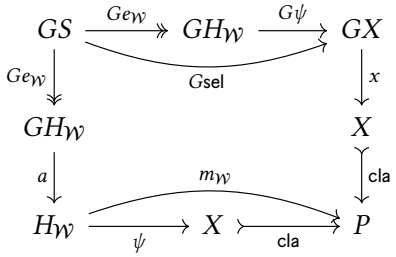
(4.4)

Proof. First suppose $\mathsf{sel} \in \mathcal{E}$ and $\mathcal{W}$ is x-consistent. We know from Lemma 4.5.3 that $\mathcal{W}$ is also x-closed. Using that $G\mathsf{sel}$ is an epi and $m_{\mathcal{W}}$ a mono, the commutative diagram below shows that ϕ is an algebra homomorphism $(X, x) \longrightarrow (H_{\mathcal{W}}, a)$.



Commutativity follows from the definition of ϕ and from Proposition 4.5.2.

Now suppose $\mathsf{cla} \in \mathcal{M}$ and $\mathcal{W}$ is x-closed. We know from Lemma 4.5.3 that $\mathcal{W}$ is also x-consistent. Using that $Ge_{\mathcal{W}}$ is an epi and cla a mono, the commutative diagram below shows that ψ is an algebra homomorphism $(H_{\mathcal{W}}, a) \longrightarrow (X, x)$.



Commutativity follows from the definition of ψ and from Proposition 4.5.2. □

This implies in particular that both selecting everything and classifying faithfully allow us to recover the target structure up to isomorphism.

Corollary 4.5.5. *Given a wrapper $\mathcal{W} = (S - \mathsf{sel} \to X, X - \mathsf{cla} \to P)$, a functor $G : \mathbf{C} \longrightarrow \mathbf{C}$, and an algebra $x : GX \longrightarrow X$, if $\mathsf{sel} \in \mathcal{E}$ and $\mathsf{cla} \in \mathcal{M}$, then $\mathcal{W}$ is x-closed and x-consistent and the unique diagonals as in* (4.4) *form an algebra isomorphism between (X, x) and $(H_{\mathcal{W}}, a)$, where $a : GH_{\mathcal{W}} \longrightarrow H_{\mathcal{W}}$ is the algebra obtained via Proposition 4.5.2.*

We now specialise Proposition 4.5.4 to automata in order to obtain conditions under which the target automaton is recovered up to isomorphism as the hypothesis. To arrive at an isomorphism rather than a homomorphism, an additional assumption is required of the automaton being reachable. For this reason the result below does not have the symmetry of Proposition 4.5.4.

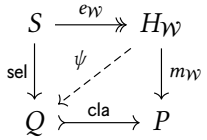
Algorithm 4.3. Abstract minimisation algorithm

Require: a morphism $\mathsf{cla} : Q_t \longrightarrow P \in \mathcal{M}$, assume $\mathcal{A}_t$ is reachable

1: $\mathsf{sel} \longleftarrow ! : 0 \longrightarrow Q_t$
2: **while** $(\mathsf{sel}, \mathsf{cla})$ is not $\mathcal{A}_t$-closed **do**
3: $\quad \mathsf{sel} \longleftarrow \mathsf{sel}'$ such that $(\mathsf{sel}', \mathsf{cla})$ is locally $\mathcal{A}_t$-closed w.r.t. sel
4: **return** $\mathcal{H}_{(\mathsf{sel},\mathsf{cla})}$

Theorem 4.5.6. *Consider a reachable automaton $\mathcal{A} = (Q, \delta, i, o)$ and let $\mathcal{W} = (S - \mathsf{sel} \to Q, Q - \mathsf{cla} \to P)$ be a wrapper. If $\mathcal{W}$ is $\mathcal{A}$-closed and $\mathsf{cla} \in \mathcal{M}$, then $\mathcal{W}$ is $\mathcal{A}$-consistent and $\mathcal{H}_{\mathcal{W}}$ and $\mathcal{A}$ are isomorphic.*

Proof. We define $\psi : H_{\mathcal{W}} \longrightarrow Q$ as the unique diagonal in the commutative square below.

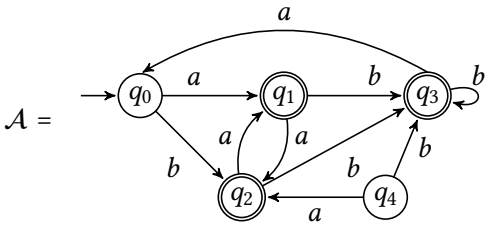


Recall that the initial state map i of the automaton is an algebra for the constant functor I while the output map o is a coalgebra for the constant functor O. Hence, we can apply Proposition 4.5.4 (or its dual for the coalgebra) to them and to δ to find that $\psi : H_{\mathcal{W}} \longrightarrow Q$ is an automaton homomorphism $\mathcal{H}_{\mathcal{W}} \longrightarrow \mathcal{A}$. Then $\mathsf{reach}_{\mathcal{A}} = \psi \circ \mathsf{reach}_{\mathcal{H}_{\mathcal{W}}}$. Since $\mathsf{reach}_{\mathcal{A}}$ is in $\mathcal{E}$, this means that $\psi \in \mathcal{E}$ [AHS09, Proposition 14.11 via duality]. Because $\mathsf{sel} = \psi \circ e_{\mathcal{W}}$ by the definition of ψ, $\mathsf{sel} \in \mathcal{E}$. Therefore, we can apply Corollary 4.5.5, again three times, and obtain an automaton isomorphism between $\mathcal{A}$ and $\mathcal{H}_{\mathcal{W}}$. □

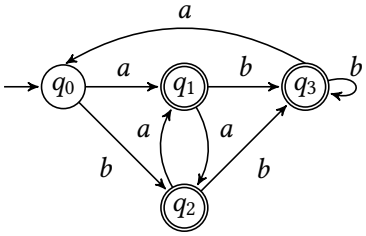
The above result provides the foundation for Algorithm 4.3: assume a wrapper that faithfully classifies a reachable target, make it closed, and obtain the target up to isomorphism. Here we fix, as in Section 4.4, a target automaton $\mathcal{A}_t = (Q_t, \delta_t, i_t, o_t)$, which the algorithm assumes to be reachable. It further requires a given classifier $\mathsf{cla} : Q_t \longrightarrow P \in \mathcal{M}$. Although the target automaton is not initially known to the algorithm, we assume we are able to decide $\mathcal{A}_t$-closedness for the wrappers under consideration and advance them through local closedness.

To see how this could be used in practice, suppose $\mathcal{A}$ is any given automaton, known to the algorithm, and let $\mathcal{A}_t$ be its reachable part. Letting $\mathsf{cla} : \mathcal{A}_t \longrightarrow \mathcal{A} \in \mathcal{M}$ be the unique embedding, we can apply Algorithm 4.3 to compute the reachable part of $\mathcal{A}$. To make this more concrete, we give an example for DFAs below.

Example 4.5.7 (DFA reachability analysis)**.** Consider the following DFA $\mathcal{A} = (Q, \delta, i, o)$ over the alphabet $A = \{a, b\}$.



Let $\mathcal{A}_\mathsf{r} = (Q_\mathsf{r}, \delta_\mathsf{r}, i_\mathsf{r}, o_\mathsf{r})$ be the reachable part of $\mathcal{A}$. We define $\mathsf{cla} : Q_\mathsf{r} \longrightarrow Q$ to be the automaton embedding. Given a set $S \subseteq Q_\mathsf{r}$, we define $\mathsf{sel}_S : S \longrightarrow Q_\mathsf{r}$ be the inclusion, which gives us a wrapper $(\mathsf{sel}_S, \mathsf{cla})$ for Q_r. Starting from $S = \emptyset$, note that this wrapper is not $\mathcal{A}$-closed. We will repeatedly find the smallest superset S' of S such that $(\mathsf{sel}_{S'}, \mathsf{cla})$ is locally $\mathcal{A}_\mathsf{r}$-closed w.r.t. sel_S and update S to S'. We have that $(\mathsf{sel}_{\{q_0\}}, \mathsf{cla})$ is locally $\mathcal{A}_\mathsf{r}$-closed w.r.t. sel_S, so we update $S = \{q_0\}$. Now $(\mathsf{sel}_{\{q_0,q_1,q_2\}}, \mathsf{cla})$ is locally $\mathcal{A}_\mathsf{r}$-closed w.r.t. sel_S, so we update $S = \{q_0, q_1, q_2\}$. Finally, $(\mathsf{sel}_{\{q_0,q_1,q_2,q_3\}}, \mathsf{cla})$ is locally $\mathcal{A}_\mathsf{r}$-closed w.r.t. sel_S, so we update $S = \{q_0, q_1, q_2, q_3\}$. We then have that $(\mathsf{sel}_S, \mathsf{cla})$ is $\mathcal{A}_\mathsf{r}$-closed, so by Theorem 4.5.6 it is $\mathcal{A}_\mathsf{r}$-consistent and its hypothesis is isomorphic to $\mathcal{A}_\mathsf{r}$. One can show that the hypothesis construction simply consists in taking restricting the structure of Q to the state space S. Thus, the final hypothesis is the one below.



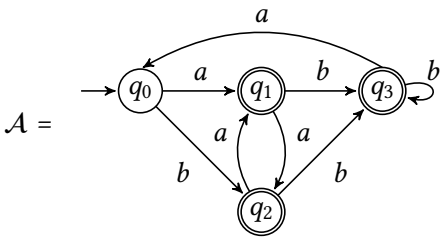
A different approach is in a learning setting where we know that the target DFA has at most n states and each of its states accepts a different language than the other states. In this case the restricted language map $\mathsf{cla} : Q_\mathsf{t} \longrightarrow 2^{A^{<n}}$, where $A^{<n}$ contains all words in $A^\star$ of length less than n, satisfies $\mathsf{cla} \in \mathcal{M}$. One can use an observation table and make it closed in the usual way to obtain the reachable version of $\mathcal{A}_\mathsf{t}$ as the hypothesis.

We know that DAs can be described with the setting $\mathbf{C} = \mathbf{Set}$, $F = (-) \times A$, $I = 1$, and $O = 2$, but they can equivalently be recovered in $\mathbf{C} = \mathbf{Set}^{\mathbf{op}}$ with $F = (-)^A$, $I = 2$, and $O = 1$. To see this, note that a structure as below on the left in $\mathbf{Set}^{\mathbf{op}}$ is a structure as below on the right in

Set.

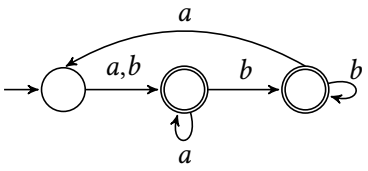
This is precisely a DA with its transition function curried. The algebraically free monad $((-)^A)^\star$ in $\mathbf{Set}^{\mathbf{op}}$ assigns to each set X the set $X^{A^\star}$, its unit has components $\eta_X : X^{A^\star} \longrightarrow X$ (in **Set**) given by $\eta_X(l) = l(\varepsilon)$, and its multiplication has components $\mu_X : X^{A^\star} \longrightarrow (X^{A^\star})^{A^\star}$ given by $\mu_X(f)(u)(v) = f(uv)$. Given a DA $\mathcal{A} = (Q, \delta, o, i)$ in $\mathbf{Set}^{\mathbf{op}}$, this leads to a reachability map in the form of the function $\mathsf{obs}_\mathcal{A} : Q \longrightarrow 2^{A^\star}$ that assigns to each state the language it accepts. It is referred to as the *observability map* of $\mathcal{A}$, and the property of reachability is in this dual setting referred to as *observability*: distinct states accept distinct languages. By applying Algorithm 4.3 we recover an algorithm to merge equivalent states.

Example 4.5.8 (DFA observability analysis)**.** Consider the DFA $\mathcal{A} = (Q, \delta, o, i)$ in $\mathbf{Set}^{\mathbf{op}}$ given below.



By factorising the observability map of $\mathcal{A}$ we know there is an observable DFA $\mathcal{A}_\mathsf{o} = (Q_\mathsf{o}, \delta_\mathsf{o}, o_\mathsf{o}, i_\mathsf{o})$ with an automaton quotient $\mathsf{sel} : Q \longrightarrow Q_\mathsf{o}$ that merges language equivalent states. We start from a trivial classifier $\mathsf{cla} : Q_\mathsf{o} \longrightarrow 1$. Note that the wrapper this induces is $(\mathsf{cla}, \mathsf{sel})$ in $\mathbf{Set}^{\mathbf{op}}$. It is not $\mathcal{A}_\mathsf{o}$-closed, since q_0 is identified with the other states while it is the only rejecting state. Adjusting to $\mathsf{cla} : Q_\mathsf{o} \longrightarrow 2$ in such a way that $(\mathsf{cla} \circ \mathsf{sel})(q_0) = 0$ and $(\mathsf{cla} \circ \mathsf{sel})(q) = 1$ for all $q \in Q \setminus \{q_0\}$ results in a wrapper that is locally $\mathcal{A}_\mathsf{o}$-closed w.r.t. the trivial classifier. Note that this classifier is well-defined because Q_o will not identify states with differing outputs. Similarly, in the next step we update the classifier to $\mathsf{cla} : Q_\mathsf{o} \longrightarrow \{0, 1, 2\}$ in such a way that $(\mathsf{cla} \circ \mathsf{sel})(q_0) = 0$, and $(\mathsf{cla} \circ \mathsf{sel})(q_1) = (\mathsf{cla} \circ \mathsf{sel})(q_2) = 1$, and $(\mathsf{cla} \circ \mathsf{sel})(q_3) = 2$. This classifier is $\mathcal{A}_\mathsf{o}$-closed, and by Theorem 4.5.6 it is consistent and its hypothesis is isomorphic to $\mathcal{A}_\mathsf{o}$. We obtain this hypothesis by quotienting $\mathcal{A}$ according to $\mathsf{cla} \circ \mathsf{sel}$. This gives the DFA

below.



Again, we can carry this approach to a learning setting where we know that the target DFA is reachable and has at most n states. In this case the restricted reachability map $\mathsf{sel} : A^{<n} \longrightarrow Q_{\mathrm{t}}$, where $A^{<n}$ contains all words in $A^\star$ of length less than n, satisfies $\mathsf{sel} \in \mathcal{E}$. One can use an observation table and make it consistent in the usual way to obtain the minimisation of $\mathcal{A}_{\mathrm{t}}$ as the hypothesis. In the more general case where $\mathsf{sel} : S \longrightarrow Q_{\mathrm{t}}$ is any restricted reachability map for a provided finite set $S \subseteq A^\star$ satisfying $\mathsf{sel} \in \mathcal{E}$, this automata learning algorithm is called `ID` and was introduced by Angluin [Ang81].

4.6 Learning Generalised Tree Automata

In this section we instantiate the development of Section 4.4 to a wide class of **Set** endofunctors. This yields an abstract algorithm for *generalised* tree automata—i.e., automata accepting sets of trees, possibly subject to equations—which include bottom-up tree automata and unordered tree automata. These are examples that were not in scope of any of the existing abstract learning frameworks in the literature.

We first introduce the running examples for this section.

Example 4.6.1 (Tree automata)**.** Let Γ be a ranked alphabet, i.e., a finite set where $\gamma \in \Gamma$ comes with $\mathsf{arity}(\gamma) \in \mathbb{N}$. The set of Γ-trees over a finite set of leaf symbols I is the smallest set $T_\Gamma(I)$ such that $I \subseteq T_\Gamma(I)$, and for all $\gamma \in \Gamma$ we have that $t_1, \dots, t_{\mathsf{arity}(\gamma)} \in T_\Gamma(I)$ implies $(\gamma, t_1, \dots, t_{\mathsf{arity}(\gamma)}) \in T_\Gamma(I)$. The alphabet Γ gives rise to the polynomial functor $FX = \coprod_{\gamma \in \Gamma} X^{\mathsf{arity}(\gamma)}$. The corresponding free F-algebra monad $F^\star$ is precisely T_Γ, where the unit turns elements into leaves, and the multiplication flattens a nested tree into a tree. A bottom-up deterministic tree automaton is then an automaton over F with a finite input set I and output set $O = 2$.

Example 4.6.2 (Unordered tree automata)**.** Consider the finite powerset functor $\mathcal{P}_{\mathrm{fin}} : \mathbf{Set} \longrightarrow \mathbf{Set}$, mapping a set to its finite subsets. The corresponding free $\mathcal{P}_{\mathrm{fin}}$-monad maps a set X to the set of finitely-branching unordered trees with nodes in X. Automata over $\mathcal{P}_{\mathrm{fin}}$, with output set $O = 2$ and finite I, accept sets of such trees. Note that unordered trees can be seen as trees

over a ranked alphabet $\Gamma = \{\mathfrak{s}_i \mid i \in \mathbb{N}\}$, where $\mathsf{arity}(\mathfrak{s}_i) = i$, satisfying equations that collapse duplicate branches and identify lists of branches up to permutations.

Automata in these examples are algebras for endofunctors with the following properties: they are *strongly finitary* [AMV03]—i.e., they preserve both filtered colimits and finite sets; and they preserve weak pullbacks. In this section we will show that the abstract algorithm from Section 4.4 can be concretely instantiated for any finite automaton over a strongly finitary, weak-pullback-preserving functor $F : \mathbf{Set} \longrightarrow \mathbf{Set}$ and a finite input set I. For the rest of this section we assume these properties for F and I, and we fix a target automaton $\mathcal{A}_\mathsf{t} = (Q_\mathsf{t}, \delta_\mathsf{t}, i_\mathsf{t}, o_\mathsf{t})$. The strongly finitary condition will be needed to ensure all abstract ingredients of the algorithm can be instantiated with terminating procedures; the weak pullback preservation allows us to solve local consistency. If the target automaton has a finite state space, the algorithm terminates by Theorem 4.4.7.

In Section 4.6.1 we instantiate wrappers to ones with a specific format, which make use of *contexts* to generalise string concatenation to trees, and we show how these wrappers and their hypotheses can be computed, whereupon we develop procedures for local closedness (Section 4.6.2) and local consistency (Section 4.6.3). We conclude by identifying a set of suitable (finite) counterexamples in Section 4.6.4. Altogether, this makes the ingredients of our abstract algorithm concrete for the present setting.

4.6.1 Contextual Wrappers

Denote by 1 the set $\{\Box\}$. Given $x \in X$ for any set X, we write $\mathfrak{e}_x : 1 \longrightarrow X$ for the function that assigns x to $\Box$. Note that for all functions $f : X \longrightarrow Y$ we have

$$\mathfrak{e}_{f(x)} = f \circ \mathfrak{e}_x. \tag{4.5}$$

We use the set 1 to define the set of *contexts* $F^\star(I+1)$, where the *holes* $\Box$ occurring in a context $c \in F^\star(I+1)$ can be used to plug in further data such as another context or a tree, e.g., in the case of Example 4.6.1. In fact, it is well known that $F^\star(I + (-))$ forms a monad [LG02], of which we denote the unit by $\hat{\eta}_X = F^\star\kappa_2 \circ \eta_X : X \longrightarrow F^\star(I + X)$ and the multiplication by $\hat{\mu}_X : F^\star(I + F^\star(I + X)) \longrightarrow F^\star(I + X)$. We now introduce a class of wrappers where, intuitively, contexts are used to distinguish inequivalent states. These wrappers will induce observation tables with trees as their row labels and contexts as their column labels.

Definition 4.6.3 (Contextual wrappers)**.** Given $S \subseteq F^\star I$ and $E \subseteq F^\star(I+1)$, we define

- $\mathsf{sel}_S : S \longrightarrow Q_\mathsf{t}$ as the restriction of the reachability map of $\mathcal{A}_\mathsf{t}$ to S; and

- $\mathsf{cla}_E : Q_\mathfrak{t} \longrightarrow O^E$, as the function given by

$$\mathsf{cla}_E(q)(e) = (o_\mathfrak{t} \circ [i_\mathfrak{t}, \mathfrak{e}_q]^\sharp)(e).$$

A *contextual wrapper* is a wrapper of the form $(S \xrightarrow{\mathsf{sel}_S} Q_\mathfrak{t}, Q \xrightarrow{\mathsf{cla}_E} O^E)$ for some S and E.

The map $[i_\mathfrak{t}, \mathfrak{e}_q]^\sharp$ in the definition of cla_E above plugs the state $q \in Q$ into the holes of the context $e \in E \subseteq F^\star(I + 1)$ to produce an element of $F^\star Q$. In the following example we show how contextual wrappers for the DA setting ($F = (-) \times A$, $I = 1$, $O = 2$) are the wrappers corresponding to observation table wrappers.

Example 4.6.4. In the case of DAs, contextual wrappers are essentially those of Example 4.1.2. In fact, sel_S is the restriction of $i^\sharp : A^\star \cong 1 \times A^\star \longrightarrow Q_\mathfrak{t}$ to $S \subseteq A^\star \cong 1 \times A^\star$. For cla_E, a bit of care is required due to the generality of contexts. Note that $((-) \times A)^\star(I + 1) = \{*, \square\} \times A^\star$. We have $E \subseteq \{*, \square\} \times A^\star$, and

$$\mathsf{cla}_E(q)(x) = \begin{cases} (o_\mathfrak{t} \circ \delta_\mathfrak{t}^\star)(q, e) & \text{if } x = (\square, e) \\ (o_\mathfrak{t} \circ \delta_\mathfrak{t}^\star)(i_\mathfrak{t}(*), e) & \text{if } x = (*, e). \end{cases}$$

Note that the second case is not useful as a context distinguishing two states, because it does not depend on q. Moreover, choosing $q = i_\mathfrak{t}(*)$ unifies the two cases. Thus, we need not consider any contexts not containing a hole, and we can choose columns $E \subseteq \{\square\} \times A^\star \cong A^\star$.

Let us consider contextual wrappers for tree automata (Example 4.6.1). We have that $S \subseteq T_\Gamma(I)$ is a set of Γ-trees over I, and $E \subseteq T_\Gamma(I + 1)$ is formed by contexts, i.e., Γ-trees where a special leaf $\square$ may occur. The function $\mathsf{sel}_S(t)$ is the state reached after reading the tree t, and $\mathsf{cla}_E(q)(t)$ can be seen as a generalisation of the DA case: it is the output of the state reached via $\delta^\star$ after replacing every occurrence of $\square$ with q and $x \in I$ with $i(x)$ in t. Therefore $(\mathsf{sel}_S \circ \mathsf{cla}_E) : S \longrightarrow O^E$ is the upper part of an observation table where rows are labelled by trees, columns by contexts, and rows are computed by plugging labels into each column context and querying the language. When E contains only contexts with exactly one instance of $\square$, this corresponds precisely to the observation tables of [DH03; BM07].

We show in Proposition 4.6.6 how to compute the morphisms induced by a wrapper that are used in the definitions of closedness, consistency, and the hypothesis. In particular, we show that they can be computed by querying the language $\mathcal{L}_\mathcal{A}$. This in practice amounts to asking membership queries to the teacher. The following lemma shows how an arbitrary row for a given row label $u \in F^\star I$ can be computed: plug u into a context taken from the column label set and evaluate the resulting tree in the language.

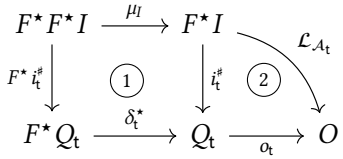
Lemma 4.6.5. *Given $E \subseteq F^\star(I+1)$ with inclusion $k : E \longrightarrow F^\star(I+1)$, we have for all $u \in F^\star I$,*

$$(\mathsf{cla}_E \circ i_\mathfrak{t}^\#)(u) = E \xrightarrow{k} F^\star(I+1) \xrightarrow{F^\star[\eta_I, \mathfrak{e}_u]} F^\star F^\star I \xrightarrow{\mu_I} F^\star I \xrightarrow{\mathcal{L}_{\mathcal{A}_\mathfrak{t}}} O.$$

Proof. First note that (omitting the inclusion k)

$$\begin{aligned}
(\mathsf{cla}_E \circ i_\mathfrak{t}^\#)(u) &= o_\mathfrak{t} \circ [i_\mathfrak{t}, \mathfrak{e}_{i_\mathfrak{t}^\#(u)}]^\# && \text{(definition of } \mathsf{cla}_E\text{)} \\
&= o_\mathfrak{t} \circ \delta_\mathfrak{t}^\star \circ F^\star[i_\mathfrak{t}, \mathfrak{e}_{i_\mathfrak{t}^\#(u)}] && \text{(definition of } (-)^\#\text{)} \\
&= o_\mathfrak{t} \circ \delta_\mathfrak{t}^\star \circ F^\star[i_\mathfrak{t}, i_\mathfrak{t}^\# \circ \mathfrak{e}_u] && (4.5) \\
&= o_\mathfrak{t} \circ \delta_\mathfrak{t}^\star \circ F^\star[i_\mathfrak{t}^\# \circ \eta_I, i_\mathfrak{t}^\# \circ \mathfrak{e}_u] && \text{(property of } (-)^\#\text{)} \\
&= o_\mathfrak{t} \circ \delta_\mathfrak{t}^\star \circ F^\star i_\mathfrak{t}^\# \circ F^\star[\eta_I, \mathfrak{e}_u].
\end{aligned}$$

It remains to show that $o_\mathfrak{t} \circ \delta_\mathfrak{t}^\star \circ F^\star i_\mathfrak{t}^\# = \mathcal{L}_{\mathcal{A}_\mathfrak{t}} \circ \mu_I$, which follows by commutativity of the diagram below.



(1) $i_\mathfrak{t}^\#$ is an $F^\star$-algebra homomorphism (2) definition of $\mathcal{L}_{\mathcal{A}_\mathfrak{t}}$ □

Proposition 4.6.6 (Computing wrapper morphisms)**.** *Given $S \subseteq F^\star I$ with inclusion $j : S \longrightarrow F^\star I$ and $E \subseteq F^\star(I+1)$ with inclusion $k : E \longrightarrow F^\star(I+1)$, we have*

$$\begin{aligned}
\mathsf{cla}_E \circ \mathsf{sel}_S &: S \longrightarrow O^E && s \mapsto E \xrightarrow{k} F^\star(I+1) \xrightarrow{F^\star[\eta_I, \mathfrak{e}_{j(s)}]} F^\star F^\star I \xrightarrow{\mu_I} F^\star I \xrightarrow{\mathcal{L}_{\mathcal{A}_\mathfrak{t}}} O \\
\mathsf{cla}_E \circ \delta_\mathfrak{t} \circ F\mathsf{sel}_S &: FS \longrightarrow O^E && f \mapsto E \xrightarrow{k} F^\star(I+1) \xrightarrow{F^\star[\eta_I, \theta_I \circ Fj \circ \mathfrak{e}_f]} F^\star F^\star I \xrightarrow{\mu_I} F^\star I \xrightarrow{\mathcal{L}_{\mathcal{A}_\mathfrak{t}}} O \\
\mathsf{cla}_E \circ i_\mathfrak{t} &: I \longrightarrow O^E && x \mapsto E \xrightarrow{k} F^\star(I+1) \xrightarrow{F^\star[\mathsf{id}_I, \mathfrak{e}_x]} F^\star I \xrightarrow{\mathcal{L}_{\mathcal{A}_\mathfrak{t}}} O \\
o_\mathfrak{t} \circ \mathsf{sel}_S &: S \longrightarrow O && s \mapsto \mathcal{L}_{\mathcal{A}_\mathfrak{t}}(s).
\end{aligned}$$

Proof. For the first equation, we derive

$$\begin{aligned}
(\mathsf{cla}_E \circ \mathsf{sel}_S)(s) &= (\mathsf{cla}_E \circ i_\mathfrak{t}^\#)(s) && \text{(definition of } \mathsf{sel}_S\text{)} \\
&= \mathcal{L}_{\mathcal{A}_\mathfrak{t}} \circ \mu_I \circ F^\star[\eta_I, \mathfrak{e}_{j(s)}] \circ k && \text{Lemma 4.6.5.}
\end{aligned}$$

For the second equation, we derive

$$\begin{aligned}
(\mathsf{cla}_E \circ \delta_{\mathsf{t}} \circ F\mathsf{sel}_S)(f) &= (\mathsf{cla}_E \circ \delta_{\mathsf{t}} \circ F i_{\mathsf{t}}^{\#} \circ Fj)(f) && \text{(definition of } \mathsf{sel}_S\text{)} \\
&= (\mathsf{cla}_E \circ i_{\mathsf{t}}^{\#} \circ \theta_I \circ Fj)(f) && (i_{\mathsf{t}}^{\#} \text{ is an } F\text{-algebra homomorphism}) \\
&= \mathcal{L}_{\mathcal{A}_{\mathsf{t}}} \circ \mu_I \circ F^{\star}[\eta_I, \mathfrak{e}_{(\theta_I \circ Fj)(f)}] \circ k && \text{Lemma 4.6.5} \\
&= \mathcal{L}_{\mathcal{A}_{\mathsf{t}}} \circ \mu_I \circ F^{\star}[\eta_I, \theta_I \circ Fj \circ \mathfrak{e}_f] \circ k && \text{(4.5).}
\end{aligned}$$

For the third equation, we derive

$$\begin{aligned}
(\mathsf{cla}_E \circ i_{\mathsf{t}})(s) &= (\mathsf{cla}_E \circ i_{\mathsf{t}}^{\#} \circ \eta_I \circ j)(s) && \text{(property of } i_{\mathsf{t}}^{\#}\text{)} \\
&= \mathcal{L}_{\mathcal{A}_{\mathsf{t}}} \circ \mu_I \circ F^{\star}[\eta_I, \mathfrak{e}_{(\eta_I \circ j)(s)}] \circ k && \text{Lemma 4.6.5} \\
&= \mathcal{L}_{\mathcal{A}_{\mathsf{t}}} \circ \mu_I \circ F^{\star}[\eta_I, \eta_I \circ \mathfrak{e}_{j(s)}] \circ k && \text{(4.5)} \\
&= \mathcal{L}_{\mathcal{A}_{\mathsf{t}}} \circ \mu_I \circ F^{\star} \eta_I \circ F^{\star}[\mathsf{id}_I, \mathfrak{e}_{j(s)}] \circ k \\
&= \mathcal{L}_{\mathcal{A}_{\mathsf{t}}} \circ F^{\star}[\mathsf{id}_I, \mathfrak{e}_{j(s)}] \circ k && \text{(monad law).}
\end{aligned}$$

Finally, for the fourth equation, we derive:

$$\begin{aligned}
(o_{\mathsf{t}} \circ \mathsf{sel}_S)(s) &= (o_{\mathsf{t}} \circ i_{\mathsf{t}}^{\#})(s) && \text{(definition of } \mathsf{sel}_S\text{)} \\
&= \mathcal{L}_{\mathcal{A}_{\mathsf{t}}}(s) && \text{(definition of } \mathcal{L}_{\mathcal{A}_{\mathsf{t}}}\text{).}
\end{aligned}$$

□

Example 4.6.7. As in the DA case, the maps of Proposition 4.6.6 correspond to the observation table. The proposition tells us how they can be computed by querying the language.

For bottom-up tree automata:

- $\mathsf{cla}_E \circ \mathsf{sel}_s$ is the upper part of the observation table, as explained in Example 4.6.4;
- $\mathsf{cla}_E \circ \delta_{\mathsf{t}} \circ F\mathsf{sel}_S$ is the bottom part of the table. In fact, in this case the successor rows for S are labelled by $FS = \coprod_{\gamma \in \Gamma} S^{\mathsf{arity}(\gamma)}$, i.e., by trees obtained by adding a new root symbol to those from S. Successor rows are then computed by plugging these trees into the contexts E, and querying the language. Note that this requires using the map θ to convert the additional root and its arguments into a tree before they are plugged into a context.
- $\mathsf{cla}_E \circ i_{\mathsf{t}}$ returns the leaf rows, i.e., those labelled by the leaf symbols I;
- $o_{\mathsf{t}} \circ \mathsf{sel}_S$ queries the language for each row label.

For unordered tree automata the maps are similar. The key difference is that now rows are labelled by trees and contexts up to equations. As a consequence, there is just one successor

row for each set of trees in S, whereas in the previous case we have one successor row for each symbol $\gamma \in \Gamma$ and $\mathsf{arity}(\gamma)$-list of trees from S.

To see that hypotheses can be represented, consider finite $S \subseteq F^\star I$ and $E \subseteq F^\star(I+1)$ and consider the wrapper $\mathcal{W} = (\mathsf{sel}_S, \mathsf{cla}_E)$. Assuming closedness and consistency, the state space of the associated hypothesis is given by the image of $\mathsf{cla}_E \circ \mathsf{sel}_S : S \longrightarrow O^E$. Furthermore, the automaton structure of the hypothesis is defined by

$$i_{\mathcal{W}}(x) = (\mathsf{cla}_E \circ i_{\mathsf{t}})(x) \qquad o_{\mathcal{W}}(e_{\mathcal{W}}(s)) = (o_{\mathsf{t}} \circ \mathsf{sel}_S)(s) \qquad \delta_{\mathcal{W}}(F(e_{\mathcal{W}})(x)) = (\mathsf{cla}_E \circ \delta_{\mathsf{t}} \circ F\mathsf{sel}_S)(x).$$

We know from Proposition 4.6.6 how to compute those functions via membership queries.

The hypothesis automaton for bottom-up and unordered tree automata, as in the DA case (see Example 4.1.2), is obtained by taking distinct rows as states. See Example 4.6.7 for the description of the hypothesis input, output, and transition maps for those automata types.

4.6.2 Witnessing Local Closedness

We now show how the general notion of local closedness can be concretely instantiated for **Set** automata.

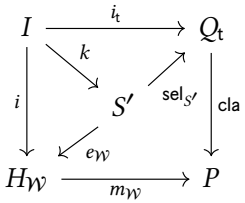
Lemma 4.6.8 (Local closedness for **Set** automata)**.** *Given $S \subseteq S' \subseteq F^\star I$ and $E \subseteq F^\star(I+1)$, the wrapper $(\mathsf{sel}_{S'}, \mathsf{cla}_E)$ is locally $\mathcal{A}_{\mathsf{t}}$-closed w.r.t. sel_S if there exist $k : I \longrightarrow S'$ and $\ell : FS \longrightarrow S'$ such that*

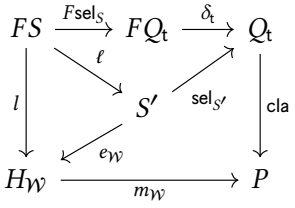
$$\mathsf{sel}_{S'} \circ k = i_{\mathsf{t}} \qquad\qquad \mathsf{sel}_{S'} \circ \ell = \delta_{\mathsf{t}} \circ F\mathsf{sel}_S.$$

Proof. Let $\mathcal{W} = (\mathsf{sel}_{S'}, \mathsf{cla}_E)$, and choose

$$i = e_{\mathcal{W}} \circ k \qquad\qquad l = e_{\mathcal{W}} \circ \ell.$$

The necessary diagrams now commute:





Note that $\mathsf{sel}_S \leq \mathsf{sel}_{S'}$ because $S \subseteq S'$. Thus, $\mathcal{W}$ is locally $\mathcal{A}_{\mathsf{t}}$-closed w.r.t. sel_S. □

Example 4.6.9. For bottom-up tree automata, local closedness holds if the table (S', E) already contains each leaf row (left equation), and it contains every successor row for S, namely $FS = \coprod_{\gamma \in \Gamma} S^{\mathsf{arity}(\gamma)}$ (right equation).

For unordered tree automata the condition is similar, and now involves successor trees in $\mathcal{P}_{\mathrm{fin}}(S)$.

The following proposition guarantees that we can always extend S to make the wrapper locally closed. Moreover, this can be done in such a way that if sel_S is a generalised reachability map and S' is the extension of S, then the resulting sel'_S is again a generalised reachability map.

Proposition 4.6.10. *Given finite $S \subseteq F^\star I$ and $E \subseteq F^\star(I+1)$, there exists a finite $S' \subseteq F^\star I$ such that $(\mathsf{sel}_{S'}, \mathsf{cla}_E)$ is locally $\mathcal{A}_\mathsf{t}$-closed w.r.t. sel_S. Furthermore, if there exists a recursive $\rho : S \longrightarrow F_I S$ such that $[\eta_I, \theta_I]^\rho : S \longrightarrow F^\star I$ is the inclusion map, then there exists a recursive $\rho' : S' \longrightarrow F_I S'$ such that $[\eta_I, \theta_I]^{\rho'} : S' \longrightarrow F^\star I$ is the inclusion map.*

Proof. Let $j : S \longrightarrow F^\star I$ be the inclusion map and define

$$S' = S \cup \{([\eta_I, \theta_I] \circ F_I j)(x) \mid x \in F_I S\} \subseteq F^\star I.$$

Since S and I are finite and F preserves finite sets, S' is also finite. We choose $k : I \longrightarrow S'$ and $\ell : FS \longrightarrow S'$ by setting

$$k(x) = \eta_I(x) \qquad\qquad \ell(x) = (\theta_I \circ Fj)(x).$$

Note that k and ℓ are well-defined by construction of S'. Using the definitions of $\mathsf{sel}_{S'}$ and k, we can then derive that

$$(\mathsf{sel}_{S'} \circ k)(x) = i_\mathsf{t}^\sharp(k(x)) = i_\mathsf{t}^\sharp(\eta_I(x)) = i_\mathsf{t}(x)$$

Furthermore, we find that

$$\begin{aligned}
(\mathsf{sel}_{S'} \circ \ell)(x) &= i_\mathsf{t}^\sharp(\ell(x)) && \text{(definition of } \mathsf{sel}_{S'}\text{)} \\
&= i_\mathsf{t}^\sharp(\theta_I(Fj(x))) && \text{(definition of } \ell\text{)} \\
&= \delta_\mathsf{t}(F(i_\mathsf{t}^\sharp)(Fj(x))) && (i_\mathsf{t}^\sharp \text{ is an } F\text{-algebra homomorphism}) \\
&= \delta_\mathsf{t}(F(i_\mathsf{t}^\sharp \circ j)(x)) \\
&= (\delta_\mathsf{t} \circ F\mathsf{sel}_S)(x) && \text{(definition of } \mathsf{sel}_S\text{)}
\end{aligned}$$

Hence $(\mathsf{sel}_{S'}, \mathsf{cla}_E)$ is locally $\mathcal{A}_\mathsf{t}$-closed w.r.t. sel_S, by Lemma 4.6.8.

Given a recursive $\rho : S \longrightarrow F_I S$ such that $[\eta_I, \theta_I]^\rho : S \longrightarrow F^\star I$ is the inclusion map, define $\rho' : S' \longrightarrow F_I S'$ for all $s \in S$ and $x \in F_I S$ by

$$\rho'(s) = (F_I j \circ \rho)(s) \qquad\qquad \rho'(([\eta_I, \theta_I] \circ F_I j)(x)) = F_I j(x).$$

To see that this is well-defined, note that $[\eta_I, \theta_I] \circ F_I j$ is injective. Moreover, if $s \in S$ and $x \in F_I S$ are such that

$$s = ([\eta_I, \theta_I] \circ F_I j)(x),$$

we have

$$([\eta_I, \theta_I]^{-1} \circ j)(s) = F_I j(x).$$

Because the inclusion map $j = [\eta_I, \theta_I]^\rho$ by assumption, it follows that

$$(F_I j \circ \rho)(s) = ([\eta_I, \theta_I]^{-1} \circ j)(s) = F_I j(x).$$

This completes the proof of ρ' being well-defined.

Note that ρ is a subcoalgebra of $[\eta_I, \theta_I]^{-1}$ via the inclusion map j by assumption. The definition of ρ' makes ρ' also a subcoalgebra of $[\eta_I, \theta_I]^{-1}$ via the inclusion map $S' \longrightarrow F^\star I$. This implies that it is recursive [ALM07, Theorem 3.17],[3] and thus its inclusion map must be $[\eta_I, \theta_I]^{\rho'} : S' \longrightarrow F^\star I$ by uniqueness. □

Example 4.6.11. To better understand this proposition, it is worth describing what the relevant recursive coalgebras are for the automata of Example 4.6.1 and Example 4.6.2. For bottom-up tree automata, prefix-closed subsets of $T_\Gamma(I)$ are sets of trees closed under taking subtrees. Every prefix-closed S can be made into a recursive coalgebra $S \longrightarrow \coprod_{\gamma \in \Gamma} S^{\mathsf{arity}(\gamma)} + I$ that returns the root symbol and its arguments, if applied to a tree of non-zero depth; and a leaf otherwise. For unordered tree automata, a recursive coalgebra $S \longrightarrow \mathcal{P}_{\mathrm{fin}} S + I$ on a subtree-closed set S may just return the set of subtrees or a leaf.

Note that using the construction from the proof of Proposition 4.6.10 to find a locally closed wrapper in the algorithm is rather inefficient, as it involves adding all successor rows to the table. For instance, in the case of Example 4.6.4, it adds rows obtained by adding a new root symbol to existing row labels in all possible ways, for each symbol in the alphabet. One may optimise the algorithm by adding instead only missing rows, and one instance of each.

[3] As stated in the introduction of [ALM07], preservation of inverse images is implied by preservation of weak pullbacks.

4.6.3 Witnessing Local Consistency

Analogously to the previous section, we now show how local consistency can be concretely instantiated for **Set** automata.

Lemma 4.6.12 (Local consistency for **Set** automata)**.** *Let $S \subseteq F^\star I$ and $E \subseteq E' \subseteq F^\star(I+1)$, with S finite. Furthermore, suppose that for $s, s' \in S$ with $(\mathsf{cla}_{E'} \circ \mathsf{sel}_S)(s) = (\mathsf{cla}_{E'} \circ \mathsf{sel}_S)(s')$ we have*

$$(o_{\mathfrak{t}} \circ \mathsf{sel}_S)(s) = (o_{\mathfrak{t}} \circ \mathsf{sel}_S)(s') \qquad \mathsf{cla}_E \circ \delta_{\mathfrak{t}} \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_s]) = \mathsf{cla}_E \circ \delta_{\mathfrak{t}} \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_{s'}])$$

Then $\mathcal{W} = (\mathsf{sel}_S, \mathsf{cla}_{E'})$ is locally $\mathcal{A}_{\mathfrak{t}}$-consistent w.r.t. cla_E.

Proof. Since $e_{\mathcal{W}}$ is surjective, so is $Fe_{\mathcal{W}}$. We define the function $l : FH_{\mathcal{W}} \longrightarrow O^E$ for all $p \in FS$ by

$$l(Fe_{\mathcal{W}}(p)) = (\mathsf{cla}_E \circ \delta_{\mathfrak{t}} \circ F\mathsf{sel}_S)(p).$$

By definition this satisfies the local consistency condition. It remains to show that the function is well-defined. Consider $y, z \in FS$ such that $Fe_{\mathcal{W}}(y) = Fe_{\mathcal{W}}(z)$. Equivalently, $Fe_{\mathcal{W}} \circ \mathfrak{e}_y = Fe_{\mathcal{W}} \circ \mathfrak{e}_z$. Denote by K the kernel

$$\{(s, s') \mid s, s' \in S, e_{\mathcal{W}}(s) = e_{\mathcal{W}}(s')\}$$

and let $j : K \longrightarrow S \times S$ be the inclusion. Because F preserves weak pullbacks, the pullback square below on the left is taken by F to the weak pullback square below on the right.

$$\begin{array}{ccc} K & \xrightarrow{\pi_1 \circ j} & S \\ {\scriptstyle \pi_2 \circ j}\downarrow & & \downarrow{\scriptstyle \mathcal{W}^\triangleright} \\ S & \xrightarrow{\mathcal{W}^\triangleright} & H_{\mathcal{W}} \end{array} \qquad\qquad \begin{array}{ccc} FK & \xrightarrow{F(\pi_1 \circ j)} & FS \\ {\scriptstyle F(\pi_2 \circ j)}\downarrow & & \downarrow{\scriptstyle F\mathcal{W}^\triangleright} \\ FS & \xrightarrow{F\mathcal{W}^\triangleright} & FH_{\mathcal{W}} \end{array}$$

By the weak pullback property we obtain $x \in FK$ such that $F(\pi_1 \circ j) \circ \mathfrak{e}_x = \mathfrak{e}_y$ and $F(\pi_2 \circ j) \circ \mathfrak{e}_x = \mathfrak{e}_z$. That is, $F(\pi_1 \circ j)(x) = y$ and $F(\pi_2 \circ j)(x) = z$.

We will define a sequence of contexts (or rather elements of $F(S+1)$) in order to transform y into z by replacing one element of S at a time. Using that S is finite, write $K = \{(s_1, s_1'), \ldots, (s_n, s_n')\}$. For all $1 \leq m \leq n$ we define $f_m : K \longrightarrow S + 1$ by

$$f_m(s_k, s_k') = \begin{cases} \kappa_1(s_k') & \text{if } k < m \\ \kappa_2(\square) & \text{if } k = m \\ \kappa_1(s_k) & \text{if } k > m. \end{cases}$$

Furthermore, let $c_m = F(f_m)(x) \in F(S+1)$. We will prove that

$$(\mathsf{cla}_E \circ \delta_{\mathfrak{t}} \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_{s_1}]))(c_1) = (\mathsf{cla}_E \circ \delta_{\mathfrak{t}} \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_{s_n}]))(c_n), \tag{4.6}$$

for which it suffices by induction to prove for all $1 \leq m < n$ that

$$(\mathsf{cla}_E \circ \delta_\mathsf{t} \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_{s_m}]))(c_m) = (\mathsf{cla}_E \circ \delta_\mathsf{t} \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_{s_{m+1}}]))(c_{m+1}).$$

Note that

$$[\mathsf{id}_S, \mathfrak{e}_{s'_m}] \circ f_m = [\mathsf{id}_S, \mathfrak{e}_{s_{m+1}}] \circ f_{m+1}$$

by the definitions of f_m and f_{m+1}, so

$$\begin{aligned}
&(\mathsf{cla}_E \circ \delta_\mathsf{t} \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_{s_m}]))(c_m) \\
&= (\mathsf{cla}_E \circ \delta_\mathsf{t} \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_{s'_m}]))(c_m) && \text{(assumption)} \\
&= (\mathsf{cla}_E \circ \delta_\mathsf{t} \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_{s'_m}] \circ f_m))(x) && \text{(definition of } c_m\text{)} \\
&= (\mathsf{cla}_E \circ \delta_\mathsf{t} \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_{s_{m+1}}] \circ f_{m+1}))(x) \\
&= (\mathsf{cla}_E \circ \delta_\mathsf{t} \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_{s_{m+1}}]))(c_{m+1}) && \text{(definition of } c_{m+1}\text{).}
\end{aligned}$$

This concludes the proof of (4.6). Then

$$\begin{aligned}
(\mathsf{cla}_E \circ \delta_\mathsf{t} \circ F\mathsf{sel}_S)(y) &= (\mathsf{cla}_E \circ \delta_\mathsf{t} \circ F(\mathsf{sel}_S \circ \pi_1 \circ j))(x) && \text{(definition of } x\text{)} \\
&= (\mathsf{cla}_E \circ \delta_\mathsf{t} \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_{s_1}] \circ f_1))(x) && \text{(definition of } f_1\text{)} \\
&= (\mathsf{cla}_E \circ \delta_\mathsf{t} \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_{s_1}]))(c_1) && \text{(definition of } c_1\text{)} \\
&= (\mathsf{cla}_E \circ \delta_\mathsf{t} \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_{s_n}]))(c_n) && \text{(4.6)} \\
&= (\mathsf{cla}_E \circ \delta_\mathsf{t} \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_{s'_n}]))(c_n) && \text{(assumption)} \\
&= (\mathsf{cla}_E \circ \delta_\mathsf{t} \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_{s'_n}]) \circ f_n)(x) && \text{(definition of } c_n\text{)} \\
&= (\mathsf{cla}_E \circ \delta_\mathsf{t} \circ F(\mathsf{sel}_S \circ \pi_2 \circ j))(x) && \text{(definition of } f_n\text{)} \\
&= (\mathsf{cla}_E \circ \delta_\mathsf{t} \circ F\mathsf{sel}_S)(z) && \text{(definition of } x\text{).}
\end{aligned}$$

We conclude that l is well-defined.

We define $o : H_\mathcal{W} \longrightarrow O$ by

$$o(e_\mathcal{W}(s)) = (o_\mathsf{t} \circ \mathsf{sel}_S)(s).$$

Again the local consistency condition is satisfied by definition, but we need to show that the function is well-defined. Consider $s_1, s_2 \in S$ such that $e_\mathcal{W}(s_1) = e_\mathcal{W}(s_2)$. Then

$$(\mathsf{cla}_{E'} \circ \mathsf{sel}_S)(s_1) = (m_\mathcal{W} \circ e_\mathcal{W})(s_1) = (m_\mathcal{W} \circ e_\mathcal{W})(s_2) = (\mathsf{cla}_{E'} \circ \mathsf{sel}_S)(s_2),$$

so $(o_\mathsf{t} \circ \mathsf{sel}_S)(s_1) = (o_\mathsf{t} \circ \mathsf{sel}_S)(s_2)$. Note that $\mathsf{cla}_{E'} \leq \mathsf{cla}_E$ because $E \subseteq E'$. Thus, $\mathcal{W}$ is locally $\mathcal{A}_\mathsf{t}$-consistent w.r.t. cla_E. □

Example 4.6.13. For bottom-up tree automata, local consistency amounts to requiring the following for the table for (S, E'). For every pair of trees $s, s' \in S$ such that the corresponding rows are equal we must have:

- both s and s' are either accepted or rejected;
- successor rows obtained by extending s and s' in the same way are equal. Formally, comparable extensions of s and s' are obtained by plugging them into the same "one-level" context from $F(S+1) = \coprod_{\gamma\in\Gamma}(S + \{\Box\})^{\mathsf{arity}(\gamma)}$.

For unordered-tree automata, we need to compare s and s' only when they are equationally inequivalent. Note that one-level contexts are also up to equations, which means that the position of the hole in the context is irrelevant for computing extensions of s and s'.

Proposition 4.6.15 below ensures that we can always make the wrapper locally consistent by finding a suitable finite E'. We need the following lemma to prove the result.

Lemma 4.6.14. *For all $F^\star$-algebras (X, x), $p : I \longrightarrow X$, and $c \in F^\star(I + X)$, the diagram below commutes.*

$$\begin{array}{ccc} F^\star(I + F^\star(I+X)) & \xrightarrow{\hat{\mu}_X} & F^\star(I+X) \\ {\scriptstyle F^\star(\mathsf{id}_I + \mathfrak{e}_c)}\uparrow & & \downarrow{\scriptstyle [p,\mathsf{id}_X]^\sharp} \\ F^\star(I+1) & \xrightarrow{[p,\mathfrak{e}_{[p,\mathsf{id}_X]^\sharp(c)}]^\sharp} & X \end{array}$$

Proof. Given any set Y and an $F^\star(I+(-))$-algebra (Z, z), the extension of a morphism $f : Y \longrightarrow Z$ to the $F^\star(I+(-))$-algebra homomorphism $f^\natural : F^\star(I+Y) \longrightarrow Z$ is given by $f^\natural = z \circ F^\star(\mathsf{id}_I + f)$. We can supply X with the $F^\star(I+(-))$-algebra structure $[p, \mathsf{id}_X]^\sharp : F^\star(I+X) \longrightarrow X$. Thus,

$$\begin{aligned} [p, \mathfrak{e}_{[p,\mathsf{id}_X]^\sharp(c)}]^\sharp &= [p, \mathfrak{e}_{\mathsf{id}_X^\natural(c)}]^\sharp \\ &= [p, \mathsf{id}_X]^\sharp \circ F^\star(\mathsf{id}_I + \mathfrak{e}_{\mathsf{id}_X^\natural(c)}) \\ &= \mathfrak{e}^\natural_{\mathsf{id}_X^\natural(c)} \\ &= (\mathsf{id}_X^\natural \circ \mathfrak{e}_c)^\natural \qquad (4.5) \\ &= \mathsf{id}_X^\natural \circ \mathfrak{e}_c^\natural \\ &= [p, \mathsf{id}_X]^\sharp \circ \mathfrak{e}_c^\natural \\ &= [p, \mathsf{id}_X]^\sharp \circ \hat{\mu}_X \circ F^\star(\mathsf{id}_I + \mathfrak{e}_c). \end{aligned}$$

□

Proposition 4.6.15. *Given finite $S \subseteq F^\star I$ and $E \subseteq F^\star(I+1)$, the set $E' \subseteq F^\star(I+1)$ is defined as*

$$\begin{aligned} E' = E &\cup \{(\eta_{I+1} \circ \kappa_2)(\Box)\} \\ &\cup \{(\hat{\mu}_1 \circ F^\star(\mathsf{id}_I + c_x))(e) \mid e \in E, x \in F(S+1)\}, \end{aligned}$$

where $c_x : 1 \longrightarrow F^\star(I+1)$, *with* $c_x = \theta_{I+1} \circ F[F^\star \kappa_1 \circ j, \hat{\eta}_1] \circ \mathfrak{e}_x$, *where* $j : S \longrightarrow F^\star I$ *is set inclusion. It holds that* E' *is finite and* $(\mathsf{sel}_S, \mathsf{cla}_{E'})$ *is locally* $\mathcal{A}_\mathsf{t}$*-consistent w.r.t.* cla_E.

Proof. Note that since S is finite and F preserves finite sets we have that $F(S+1)$ is also finite. Together with the fact that E is finite it follows that E' is finite. Suppose $s_1, s_2 \in S$ are such that $(\mathsf{cla}_{E'} \circ \mathsf{sel}_S)(s_1) = (\mathsf{cla}_{E'} \circ \mathsf{sel}_S)(s_2)$. For all $s \in S$ we have

$$\begin{aligned}
(o_\mathsf{t} \circ \mathsf{sel}_S)(s) &= (o_\mathsf{t} \circ \mathfrak{e}_{\mathsf{sel}_S(s)})(\square) \\
&= (o_\mathsf{t} \circ [i_\mathsf{t}, \mathfrak{e}_{\mathsf{sel}_S(s)}] \circ \kappa_2)(\square) \\
&= (o_\mathsf{t} \circ [i_\mathsf{t}, \mathfrak{e}_{\mathsf{sel}_S(s)}]^\sharp \circ \eta_{I+1} \circ \kappa_2)(\square) \\
&= (\mathsf{cla}_{E'} \circ \mathsf{sel}_S)(s)((\eta_{I+1} \circ \kappa_2)(\square)) \qquad \text{(definition of } \mathsf{cla}_{E'}\text{)},
\end{aligned}$$

so

$$\begin{aligned}
(o_\mathsf{t} \circ \mathsf{sel}_S)(s_1) &= (\mathsf{cla}_{E'} \circ \mathsf{sel}_S)(s_1)((\eta_{I+1} \circ \kappa_2)(\square)) \\
&= (\mathsf{cla}_{E'} \circ \mathsf{sel}_S)(s_2)((\eta_{I+1} \circ \kappa_2)(\square)) \\
&= (o_\mathsf{t} \circ \mathsf{sel}_S)(s_2).
\end{aligned}$$

Furthermore, for all $s \in S$ we have

$$\begin{aligned}
& F[F^\star i_\mathsf{t} \circ j, F^\star \mathfrak{e}_{\mathsf{sel}_S(s)} \circ \eta_1] \\
&= F[F^\star [i_\mathsf{t}, \mathfrak{e}_{\mathsf{sel}_S(s)}] \circ F^\star \kappa_1 \circ j, F^\star [i_\mathsf{t}, \mathfrak{e}_{\mathsf{sel}_S(s)}] \circ F^\star \kappa_2 \circ \eta_1] \\
&= FF^\star [i_\mathsf{t}, \mathfrak{e}_{\mathsf{sel}_S(s)}] \circ F[F^\star \kappa_1 \circ j, F^\star \kappa_2 \circ \eta_1] \\
&= FF^\star [i_\mathsf{t}, \mathfrak{e}_{\mathsf{sel}_S(s)}] \circ F[F^\star \kappa_1 \circ j, \hat{\eta}_1] \qquad \text{(definition of } \hat{\eta}_1\text{)},
\end{aligned}$$

so for all $s \in S$ and $x \in F(S+1)$ we have

$$\begin{aligned}
& (\delta_\mathsf{t} \circ F[\mathsf{sel}_S, \mathfrak{e}_{\mathsf{sel}_S(s)}])(x) \\
&= (\delta_\mathsf{t} \circ F[\delta_\mathsf{t}^\star \circ F^\star i_\mathsf{t} \circ j, \delta_\mathsf{t}^\star \circ F^\star \mathfrak{e}_{\mathsf{sel}_S(s)} \circ \eta_1])(x) && \text{(definition of } \mathsf{sel}_S\text{)} \\
&= (\delta_\mathsf{t} \circ F\delta_\mathsf{t}^\star \circ F[F^\star i_\mathsf{t} \circ j, F^\star \mathfrak{e}_{\mathsf{sel}_S(s)} \circ \eta_1])(x) \\
&= (\delta_\mathsf{t}^\star \circ \theta_{Q_\mathsf{t}} \circ F[F^\star i_\mathsf{t} \circ j, F^\star \mathfrak{e}_{\mathsf{sel}_S(s)} \circ \eta_1])(x) && (\delta_\mathsf{t}^\star \text{ is an } F\text{-algebra homomorphism}) \\
&= (\delta_\mathsf{t}^\star \circ \theta_{Q_\mathsf{t}} \circ FF^\star [i_\mathsf{t}, \mathfrak{e}_{\mathsf{sel}_S(s)}] \circ F[F^\star \kappa_1 \circ j, \hat{\eta}_1])(x) && \text{(shown above)} \\
&= (\delta_\mathsf{t}^\star \circ F^\star [i_\mathsf{t}, \mathfrak{e}_{\mathsf{sel}_S(s)}] \circ \theta_{I+1} \circ F[F^\star \kappa_1 \circ j, \hat{\eta}_1])(x) \\
&= (\delta_\mathsf{t}^\star \circ F^\star [i_\mathsf{t}, \mathfrak{e}_{\mathsf{sel}_S(s)}] \circ \theta_{I+1} \circ F[F^\star \kappa_1 \circ j, \hat{\eta}_1] \circ \mathfrak{e}_x)(\square) \\
&= (\delta_\mathsf{t}^\star \circ F^\star [i_\mathsf{t}, \mathfrak{e}_{\mathsf{sel}_S(s)}] \circ c_x)(\square) && \text{(definition of } c_x\text{)} \\
&= ([i_\mathsf{t}, \mathfrak{e}_{\mathsf{sel}_S(s)}]^\sharp \circ c_x)(\square) \\
&= [i_\mathsf{t}, \mathsf{id}_{Q_\mathsf{t}}]^\sharp ((F^\star(\mathsf{id}_I + \mathfrak{e}_{\mathsf{sel}_S(s)}) \circ c_x)(\square)).
\end{aligned}$$

Note that for all $s \in S$, $x \in F(S+1)$ we have

$$\begin{aligned}
&\hat{\mu}_{Q_t} \circ F^\star(\mathsf{id}_I + \mathfrak{e}_{(F^\star(\mathsf{id}_I + \mathfrak{e}_{\mathsf{sel}_S(s)}) \circ c_x)(\square)}) \\
&= \hat{\mu}_{Q_t} \circ F^\star(\mathsf{id}_I + (F^\star(\mathsf{id}_I + \mathfrak{e}_{\mathsf{sel}_S(s)}) \circ \mathfrak{e}_{c_x(\square)})) && (4.5) \\
&= \hat{\mu}_{Q_t} \circ F^\star(\mathsf{id}_I + (F^\star(\mathsf{id}_I + \mathfrak{e}_{\mathsf{sel}_S(s)}) \circ c_x)) && (\text{definition of } \mathfrak{e}_{c_x(\square)}) \\
&= \hat{\mu}_{Q_t} \circ F^\star(\mathsf{id}_I + (F^\star(\mathsf{id}_I + \mathfrak{e}_{\mathsf{sel}_S(s)}))) \circ F^\star(\mathsf{id}_I + c_x) \\
&= F^\star(\mathsf{id}_I + \mathfrak{e}_{\mathsf{sel}_S(s)}) \circ \hat{\mu}_1 \circ F^\star(\mathsf{id}_I + c_x),
\end{aligned}$$

so for all $s \in S$, $x \in F(S+1)$, and $e \in E$,

$$\begin{aligned}
&(\mathsf{cla}_E \circ \delta_t \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_s]))(x)(e) \\
&= (\mathsf{cla}_E \circ \delta_t \circ F[\mathsf{sel}_S, \mathfrak{e}_{\mathsf{sel}_S(s)}])(x)(e) && (4.5) \\
&= (o_t \circ [i_t, \mathfrak{e}_{(\delta_t \circ F[\mathsf{sel}_S, \mathfrak{e}_{\mathsf{sel}_S(s)}])(x)}]^\sharp)(e) && (\text{definition of } \mathsf{cla}_E) \\
&= (o_t \circ [i_t, \mathfrak{e}_{[i_t, \mathsf{id}_{Q_t}]^\sharp((F^\star(\mathsf{id}_I + \mathfrak{e}_{\mathsf{sel}_S(s)}) \circ c_x)(\square))}]^\sharp)(e) && (\text{shown earlier}) \\
&= (o_t \circ [i_t, \mathsf{id}_{Q_t}]^\sharp \circ \hat{\mu}_{Q_t} \circ F^\star(\mathsf{id}_I + \mathfrak{e}_{(F^\star(\mathsf{id}_I + \mathfrak{e}_{\mathsf{sel}_S(s)}) \circ c_x)(\square)}))(e) && (\text{Lemma 4.6.14}) \\
&= (o_t \circ [i_t, \mathsf{id}_{Q_t}]^\sharp \circ F^\star(\mathsf{id}_I + \mathfrak{e}_{\mathsf{sel}_S(s)}) \circ \hat{\mu}_1 \circ F^\star(\mathsf{id}_I + c_x))(e) && (\text{shown above}) \\
&= (o_t \circ [i_t, \mathfrak{e}_{\mathsf{sel}_S(s)}]^\sharp \circ \hat{\mu}_1 \circ F^\star(\mathsf{id}_I + c_x))(e) \\
&= (\mathsf{cla}_{E'} \circ \mathsf{sel}_S)(s)((\hat{\mu}_1 \circ F^\star(\mathsf{id}_I + c_x))(e)) && (\text{definition of } \mathsf{cla}_{E'}),
\end{aligned}$$

and therefore for all $x \in F(S+1)$ and $e \in E$,

$$\begin{aligned}
&(\mathsf{cla}_E \circ \delta_t \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_{s_1}]))(x)(e) \\
&= (\mathsf{cla}_{E'} \circ \mathsf{sel}_S)(s_1)((\hat{\mu}_1 \circ F^\star(\mathsf{id}_I + c_x))(e)) \\
&= (\mathsf{cla}_{E'} \circ \mathsf{sel}_S)(s_2)((\hat{\mu}_1 \circ F^\star(\mathsf{id}_I + c_x))(e)) && (\text{assumption}) \\
&= (\mathsf{cla}_E \circ \delta_t \circ F(\mathsf{sel}_S \circ [\mathsf{id}_S, \mathfrak{e}_{s_2}]))(x)(e).
\end{aligned}$$

Thus, it follows from Lemma 4.6.12 that $(\mathsf{sel}_S, \mathsf{cla}_{E'})$ is locally $\mathcal{A}_t$-consistent w.r.t. cla_E. □

We remark that the above definition of E' results in a highly inefficient procedure that involves plugging all possible one level contexts with subtrees in S to each element of E and collecting those in E'. One can optimise it by incrementally adding elements of the proposed E' to E that distinguish rows not distinguished by the current elements of E and that need to be added in order to satisfy the conditions of Lemma 4.6.12.

4.6.4 Finite Counterexamples

Finally, we refine Proposition 4.3.5 to show that the teacher can always pick a finite counterexample.

Proposition 4.6.16 (Language equivalence via finite recursion)**.** *Given an automaton $\mathcal{A} = (Q, \delta, i, o)$, we have $\mathcal{L}_{\mathcal{A}_t} = \mathcal{L}_{\mathcal{A}}$ if and only if $\mathcal{L}^{\rho}_{\mathcal{A}_t} = \mathcal{L}^{\rho}_{\mathcal{A}}$ for all recursive $\rho : S \to F_I S$ such that S is finite.*

Proof. Suppose that for all recursive coalgebras $\rho : S \to F_I S$ such that S is finite we have $o_t \circ [i_t, \delta]^{\rho} = o \circ [i, \delta]^{\rho}$. Given $t \in F^\star I$, note that $(F^\star I, [\eta_I, \theta_I])$ is the initial algebra of functor F_I, which by being finitary is also the colimit of the initial sequence of F_I [Adá74] and hence isomorphic to $(\bigcup_{n\in\mathbb{N}} F_I^n\emptyset, a)$ for an initial algebra structure $a : F_I\left(\bigcup_{n\in\mathbb{N}} F_I^n\emptyset\right) \to \bigcup_{n\in\mathbb{N}} F_I^n\emptyset$. Let $\phi : (F^\star I, [\eta_I, \theta_I]) \to (\bigcup_{n\in\mathbb{N}} F_I^n\emptyset, a)$ be the isomorphism. There exists $n \in \mathbb{N}$ such that $\phi(t) \in F_I^n\emptyset$. The set $F_I^n\emptyset$ is finite by F_I preserving finite sets and the carrier of a recursive coalgebra $\rho : F_I^n\emptyset \to F_I^{n+1}\emptyset$ by [CUV06, Proposition 6], with $a^{\rho} : F_I^n\emptyset \to \bigcup_{n\in\mathbb{N}} F_I^n\emptyset$ being the inclusion. Then $S = \{\phi^{-1}(x) \mid x \in F_I^n\emptyset\}$ is also finite and the carrier of a recursive coalgebra $\rho' : S \to F_I S$, with $[\eta_I, \theta_I]^{\rho'} : S \to F^\star I$ being the inclusion. Moreover, $t \in S$. Thus,

$$
\begin{aligned}
\mathcal{L}_{\mathcal{A}_t}(t) &= (\mathcal{L}_{\mathcal{A}_t} \circ [\eta_I, \theta_I]^{\rho'})(t) \\
&= (o_t \circ i_t^{\#} \circ [\eta_I, \theta_I]^{\rho'})(t) && \text{(definition of } \mathcal{L}_{\mathcal{A}_t}) \\
&= (o_t \circ [i_t, \delta_t]^{\rho'})(t) && (i_t^{\#} \text{ is an } F_I\text{-algebra homomorphism}) \\
&= \mathcal{L}^{\rho'}_{\mathcal{A}_t}(t) && \text{(definition of } \mathcal{L}^{\rho'}_{\mathcal{A}}) \\
&= \mathcal{L}^{\rho'}_{\mathcal{A}}(t) && \text{(assumption)} \\
&= (o \circ [i, \delta]^{\rho'})(t) && \text{(definition of } \mathcal{L}^{\rho'}_{\mathcal{A}}) \\
&= (o \circ i^{\#} \circ [\eta_I, \theta_I]^{\rho'})(t) && (i^{\#} \text{ is an } F_I\text{-algebra homomorphism}) \\
&= (\mathcal{L}_{\mathcal{A}} \circ [\eta_I, \theta_I]^{\rho'})(t) && \text{(definition of } \mathcal{L}_{\mathcal{A}}) \\
&= \mathcal{L}_{\mathcal{A}}(t).
\end{aligned}
$$

The converse follows from Proposition 4.3.5. □

Corollary 4.6.17 (Finite counterexample existence)**.** *Given a closed and consistent wrapper $\mathcal{W}$ for Q_t, we have $\mathcal{L}_{\mathcal{H}_{\mathcal{W}}} \neq \mathcal{L}_{\mathcal{A}_t}$ if and only if there exists a counterexample $\rho : S \to F_I S$ for $\mathcal{W}$ such that S is finite.*

Example 4.6.18. Recall from Example 4.6.11 that finite recursive coalgebras for bottom-up (resp. unordered) tree automata are coalgebras $\rho : S \to \coprod_{\gamma\in\Gamma} S^{\mathsf{arity}(\gamma)} + I$ (resp. $\rho : S \to \mathcal{P}_{\mathrm{fin}} S +$

I). Therefore, finite counterexamples are recursive coalgebras of this form such that S is finite or, more concretely, a finite subtree-closed set of trees.

4.7 Related work

Barlocco et al. [BKR19] proposed an abstract algorithm for learning coalgebras. It stipulates the tests to be formed by an abstract version of coalgebraic modal logic. On the one hand, the notion of wrapper and closedness from CALF essentially instantiate to that setting; on the other hand, the combination of logic and coalgebra is precisely what enables to define an actual learning algorithm in [BKR19]. The current work focuses on algebras rather than coalgebras, and is orthogonal. In particular, it covers (bottom-up) tree automata, which is outside the scope of [BKR19].

Urbat and Schröder have recently proposed another categorical approach to automata learning [US19], which—similarly to the work of Barlocco et al.—makes stronger assumptions than in CALF in order to define a learning algorithm. Their work focuses primarly on automata, assuming that the systems of interest can be viewed both as algebras and coalgebras, and the generality comes from allowing to instantiate these in various categories. Moreover, it allows covering algebraic recognisers in certain cases, through a reduction to automata over a carefully constructed alphabet; this (orthogonal) extension allows covering, e.g., ω-languages as well as tree languages. However, the reduction to automata makes this process quite different than the approach to tree learning in the present chapter: it makes use of an automaton over all (flat) contexts, yielding an infinite alphabet, and therefore the algorithmic aspect is not clear. The extension to an actual algorithm for learning tree automata is mentioned as future work in [US19]. In the present chapter, this is achieved by learning algebras directly.

Concrete algorithms for learning tree automata and languages have appeared in the literature [Sak90; DH03; BM07]. The inference of regular tree languages using membership and equivalence queries appeared in [DH03], who extended earlier work of Sakakibara [Sak90]. Later, [BM07] provided a learning algorithm for regular tree automata using only membership queries. The instantiated algorithm in our work has elements (such as the use of contexts) close to the concrete algorithms. However, our focus is on presenting an algebraic framework that can effectively be instantiated to recover such concrete algorithms in a modular and canonical fashion, with proofs of correctness and termination stemming from the general framework.

Chapter 5

Learning Weighted Automata over Principal Ideal Domains

Weighted finite automata (WFAs) are an important model made popular due to their applicability in image processing and speech recognition tasks [CK93; MPR05]. The model is also prevalent in other areas, including bioinformatics [AMT08] and formal verification [AKL11]. Passive learning algorithms and associated complexity results have appeared in the literature (see e.g. [BM12] for an overview), whereas active learning has been less studied [BM15; BV96]. Furthermore, existing learning algorithms, both passive and active, have been developed assuming the weights in the automaton are drawn from a field, such as the real numbers.[1] To the best of our knowledge, no learning algorithms, whether passive or active, have been developed for WFAs in which the weights are drawn from a general semiring.

In this chapter, we explore *active learning* for WFAs over a general semiring. The main contributions are as follows:

1. We introduce a weighted variant of $\mathtt{L}^\star$ parametric on an arbitrary semiring, together with sufficient conditions for termination (Section 5.2).

2. We show that for general semirings our algorithm might not terminate. In particular, if the semiring is the natural numbers, one of the steps of the algorithm does not always converge (Section 5.3).

3. We prove that the algorithm terminates if the semiring is a *principal ideal domain*, covering the known case of fields, but also the integers. This yields the first active learning

[1] Balle and Mohri [BM15] define WFAs generically over a semiring but then restrict to fields from Section 3 onwards as they present an overview of existing learning algorithms.

algorithm for WFAs over the integers (Section 5.4).

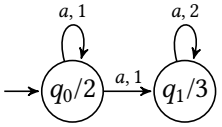
We start in Section 5.1 by explaining the learning algorithm for WFAs over the reals and pointing out the challenges in extending it to arbitrary semirings. We do not attempt to fit the developments of this chapter in the framework of Chapter 4, and no explicit category theory will be used. However, in Chapter 6 we will generalise the main algorithm from the present chapter (Algorithm 5.1) and relate it to Algorithm 4.2.

5.1 Original Algorithm for Fields

In this section we give an overview of the work developed in the chapter through examples. We start by informally explaining the general algorithm for learning weighted automata that we introduce in Section 5.2, for the case where the semiring is a field. More specifically, for simplicity we consider the field of real numbers throughout this section. Later in the section, we illustrate why this algorithm does not work for an arbitrary semiring.

Angluin's $\mathtt{L}^\star$ algorithm, as introduced in Section 2.3, provides a procedure to learn the minimal DFA accepting a certain (unknown) regular language. In the weighted variant we will introduce in Section 5.2, for the specific case of the field of real numbers, the algorithm produces the minimal WFA accepting a weighted rational language (or formal power series) $\mathcal{L} : A^\star \longrightarrow \mathbb{R}$.

A WFA over $\mathbb{R}$ consists of a set of states, a linear combination of initial states, a transition function that for each state and input symbol produces a linear combination of successor states, and an output value in $\mathbb{R}$ for each state (Definition 5.2.1). As an example, consider the WFA over $A = \{a\}$ below.



Here q_0 is the only initial state, with weight 1, as indicated by the arrow into it that has no origin. When reading a, q_0 transitions with weight 1 to itself and also with weight 1 to q_1; q_1 transitions with weight 2 just to itself. The output of q_0 is 2 and the output of q_1 is 3.

The language of a WFA is determined by letting it read a given word and determining the final output according to the weights and outputs assigned to individual states. More precisely, suppose we want to read the word aaa in the example WFA above. Initially, q_0 is assigned weight 1 and q_1 weight 0. Processing the first a then leads to q_0 retaining weight 1, as it has a self-loop with weight 1, and q_1 obtaining weight 1 as well. With the next a, the weight of q_0 still remains 1, but the weight of q_1 doubles due to its self-loop of weight 1 and

is added to the weight 1 coming from q_0, leading to a total of 3. Similarly, after the last a the weights are 1 for q_0 and 7 for q_1. Since q_0 has output 2 and q_1 output 3, the final result is $2 \cdot 1 + 3 \cdot 7 = 23$.

The learning algorithm assumes access to a *teacher*, who answers two types of queries:

- *membership queries*, consisting of a single word $w \in A^\star$, to which the teacher replies with a weight $\mathcal{L}(w) \in \mathbb{R}$; and
- *equivalence queries*, consisting of a hypothesis WFA $\mathcal{A}$, to which the teacher replies **yes** if its language $\mathcal{L}_\mathcal{A}$ equals the target language $\mathcal{L}$. If not, the teacher returns a counterexample, i.e., a word $w \in A^\star$ such that $\mathcal{L}(w) \neq \mathcal{L}_\mathcal{A}(w)$.

As in Section 2.3, the learning algorithm incrementally builds an *observation table*, which at each stage contains partial information about the language $\mathcal{L}$ determined by two finite sets $S, E \subseteq A^\star$. The algorithm fills the table through membership queries. As an example, and to set notation, consider the following table (over $A = \{a\}$).

		E: ε	a	aa
S	ε	0	1	3
	a	1	3	7
$S \cdot A$	aa	3	7	15

$$\mathsf{row} : S \longrightarrow \mathbb{R}^E$$
$$\mathsf{row}(u)(v) = \mathcal{L}(uv)$$
$$\mathsf{srow} : S \cdot A \longrightarrow \mathbb{R}^E$$
$$\mathsf{srow}(ua)(v) = \mathcal{L}(uav)$$

This table indicates that $\mathcal{L}$ assigns 0 to ε, 1 to a, 3 to aa, 7 to aaa, and 15 to $aaaa$. For instance, we see that $\mathsf{row}(a)(aa) = \mathsf{srow}(aa)(a) = 7$. Since row and srow are fully determined by the language $\mathcal{L}$, we will refer to an observation table as a pair (S, E), leaving the language $\mathcal{L}$ implicit.

If the observation table (S, E) satisfies certain properties described below, then it represents a WFA (S, δ, i, o), called *the hypothesis*, as follows:

- $\delta : S \longrightarrow (\mathbb{R}^S)^A$ is a linear map defined by choosing for $\delta(s)(a)$ a linear combination over S of which the rows evaluate to $\mathsf{srow}(sa)$;
- $i : S \longrightarrow \mathbb{R}$ is the initial weight map defined as $i(\varepsilon) = 1$ and $i(s) = 0$ for $s \neq \varepsilon$;
- $o : S \longrightarrow \mathbb{R}$ is the output weight map defined as $o(s) = \mathsf{row}(s)(\varepsilon)$.

For this to be well-defined, we need to have $\varepsilon \in S$ (for the initial weights) and $\varepsilon \in E$ (for the output weights), and for the transition function there is a crucial property of the table that needs to hold: closedness. In the weighted setting, a table is closed if for all $t \in S \cdot A$, there exist $r_s \in \mathbb{R}$ for all $s \in S$ such that

$$\mathsf{srow}(t) = \sum_{s \in S} r_s \cdot \mathsf{row}(s).$$

If this is not the case for a given $t \in S \cdot A$, the algorithm adds t to S. The table is repeatedly extended in this manner until it is closed. The algorithm then constructs a hypothesis, using the closedness witnesses to determine transitions, and poses an equivalence query to the teacher. It terminates when the answer is **yes**; otherwise it extends the table with the counterexample provided by adding all its suffixes to E, and the procedure continues by closing again the resulting table. In the next subsection we describe the algorithm through an example.

Remark 5.1.1. The original $\mathrm{L}^\star$ algorithm (Section 2.3) requires a second property to construct a hypothesis, called *consistency*. Consistency is difficult to check in extended settings, so the present chapter is based on a variant of the algorithm inspired by Maler and Pnueli [MP95] where only closedness is checked and counterexamples are handled differently. See also Figure 2.3 in Section 2.3. In Chapter 6 we will give an overview of consistency in different settings.

5.1.1 Example: Learning a Weighted Language over the Reals

Throughout this section we consider the following weighted language:

$$\mathcal{L} : \{a\}^\star \to \mathbb{R} \qquad\qquad \mathcal{L}(a^j) = 2^j - 1.$$

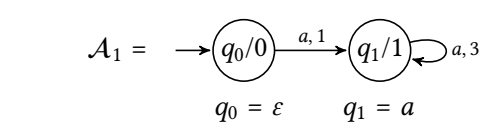
The minimal WFA recognising it has 2 states. We will illustrate how the weighted variant of Angluin's algorithm recovers this WFA.

We start from $S = E = \{\varepsilon\}$, and fill the entries of the table on the left below by asking membership queries for ε and a. The table is not closed and hence we build the table on its right, adding the membership result for aa. The resulting table is closed, as $\mathsf{srow}(aa) = 3 \cdot \mathsf{row}(a)$, so we construct the hypothesis $\mathcal{A}_1$.

	ε
ε	0
a	1

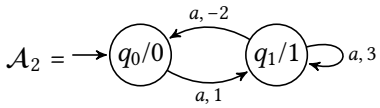
	ε
ε	0
a	1
aa	3

$\mathcal{A}_1 =$



The teacher replies **no** and gives the counterexample aaa, which is assigned 9 by the hypothesis automaton $\mathcal{A}_1$ but 7 in the language. Therefore, we extend $E \leftarrow E \cup \{a, aa, aaa\}$. The table becomes the one below. It is closed, as $\mathsf{srow}(aa) = 3 \cdot \mathsf{row}(a) - 2 \cdot \mathsf{row}(\varepsilon)$, so we construct a new hypothesis $\mathcal{A}_2$.

	ε	a	aa	aaa
ε	0	1	3	7
a	1	3	7	15
aa	3	7	15	31



The teacher replies **yes** because $\mathcal{A}_2$ accepts the intended language assigning $2^j - 1 \in \mathbb{R}$ to the word a^j, and the algorithm terminates with the correct automaton.

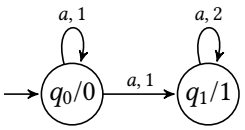
5.1.2 Learning Weighted Languages over Arbitrary Semirings

Consider now the same language as above, but represented as a map over the semiring of natural numbers $\mathcal{L} : \{a\}^\star \longrightarrow \mathbb{N}$ instead of a map $\mathcal{L} : \{a\}^\star \longrightarrow \mathbb{R}$ over the reals. Accordingly, we consider a variant of the learning algorithm over the semiring $\mathbb{N}$ rather than the algorithm over $\mathbb{R}$ described above. For the first part, the run of the algorithm for $\mathbb{N}$ is the same as above, but after receiving the counterexample we can no longer observe that $\mathsf{srow}(aa) = 3 \cdot \mathsf{row}(a) - 2 \cdot \mathsf{row}(\varepsilon)$, since $-2 \notin \mathbb{N}$. In fact, there are no $m, n \in \mathbb{N}$ such that $\mathsf{srow}(aa) = m \cdot \mathsf{row}(\varepsilon) + n \cdot \mathsf{row}(a)$. To see this, consider the first two columns in the table and note that $\frac{3}{7}$ is bigger than $\frac{0}{1} = 0$ and $\frac{1}{3}$, so it cannot be obtained as a linear combination of the latter two using natural numbers. We thus have a closedness defect and update $S \leftarrow S \cup \{aa\}$, leading to the table below.

	ε	a	aa	aaa
ε	0	1	3	7
a	1	3	7	15
aa	3	7	15	31
aaa	7	15	31	63

Again, the table is not closed, since $\frac{7}{15} > \frac{3}{7}$. In fact, these closedness defects continue appearing indefinitely, witnessing non-termination of the algorithm. This is shown formally in Section 5.3.

Note, however, that there does exist a WFA over $\mathbb{N}$ accepting this language:



(5.1)

The reason that the algorithm cannot find the correct automaton is closely related to the algebraic structure induced by the semiring. In the case of the reals, the algebras are vector spaces and the closedness checks induce increases in the dimension of the hypothesis WFA, which in turn cannot exceed the dimension of the minimal one for the language. In the case of commutative monoids, the algebras for the natural numbers, the notion of dimension does not exist. In Section 5.4 we show that one can get around this problem for a class of semirings that includes the integers.

The rest of this chapter is organised as follows. First, we introduce in Section 5.2 our general algorithm with its (parameterised) termination proof of Theorem 5.2.10. We then proceed to prove non-termination of the example discussed above over the natural numbers in Section 5.3 before instantiating our algorithm to PIDs in Section 5.4 and showing that it terminates (Theorem 5.4.10). We conclude with a discussion of related and future work in Section 5.5.

5.2 Generalised WFA Learning Algorithm

In this section we define the general algorithm for WFAs over a semiring $\mathbb{S}$, as described informally in Section 5.1. Our algorithm assumes the existence of a *closedness strategy* (Definition 5.2.4), which allows one to check whether a table is closed, and in case it is, provide relevant witnesses. We then introduce sufficient conditions on $\mathbb{S}$ and on the language $\mathcal{L}$ to be learned under which the algorithm terminates. First, we fix a semiring $\mathbb{S}$ and a finite alphabet A, which will be used throughout this chapter, and recall the definition of WFAs and their languages.

Definition 5.2.1 (WFA)**.** A *weighted finite automaton (WFA)* over $\mathbb{S}$ is a tuple (Q, δ, i, o), where Q is a finite set, $\delta : Q \to (\mathbb{S}^Q)^A$, and $i, o : Q \to \mathbb{S}$.

A *weighted language* (or just *language*) over $\mathbb{S}$ is a function $A^\star \to \mathbb{S}$. To define the language accepted by a WFA $\mathcal{A} = (Q, \delta, i, o)$, we first introduce the notions of *reachability map* $\mathsf{reach}_\mathcal{A} : V(A^\star) \to V(Q)$ and *observability map* $\mathsf{obs}_\mathcal{A} : V(Q) \to \mathbb{S}^{A^\star}$ as the semimodule homomorphisms given by

$$\mathsf{reach}^\dagger_\mathcal{A}(\varepsilon) = i \qquad \mathsf{obs}_\mathcal{A}(m)(\varepsilon) = o^\sharp(m)$$
$$\mathsf{reach}^\dagger_\mathcal{A}(ua) = \delta^\sharp(\mathsf{reach}^\dagger_\mathcal{A}(u))(a) \qquad \mathsf{obs}_\mathcal{A}(m)(au) = \mathsf{obs}_\mathcal{A}(\delta^\sharp(m)(a))(u).$$

The *language accepted by a WFA* $\mathcal{A} = (Q, \delta, i, o)$ is the function $\mathcal{L}_\mathcal{A} : A^\star \to \mathbb{S}$ given by $\mathcal{L}_\mathcal{A} = \mathsf{obs}_\mathcal{A}(i)$. Equivalently, one can define this as $\mathcal{L}_\mathcal{A} = o^\sharp \circ \mathsf{reach}^\dagger_\mathcal{A}$.

Moving on to the learning algorithm setup, an observation table in the WFA setting is very similar to the one in $\mathrm{L}^\star$ (Section 2.3). We define it formally as follows.

Definition 5.2.2 (Observation table)**.** An *observation table* (or just *table*) (S, E) consists of two sets $S, E \subseteq A^\star$. We write $\mathsf{Table}_{\mathsf{fin}} = \mathcal{P}_{\mathrm{fin}}(A^\star) \times \mathcal{P}_{\mathrm{fin}}(A^\star)$ for the set of finite tables (where $\mathcal{P}_{\mathrm{fin}}(\mathrm{X})$ denotes the collection of finite subsets of a set X). Given a language $\mathcal{L} : A^\star \longrightarrow \mathbb{S}$, an observation table (S, E) determines the *row function* $\mathsf{row}_{(S,E,\mathcal{L})} : S \longrightarrow \mathbb{S}^E$ and the *successor row* function $\mathsf{srow}_{(S,E,\mathcal{L})} : S \cdot A \longrightarrow \mathbb{S}^E$ as follows:

$$\mathsf{row}_{(S,E,\mathcal{L})}(w)(v) = \mathcal{L}(wv) \qquad\qquad \mathsf{srow}_{(S,E,\mathcal{L})}(wa)(v) = \mathcal{L}(wav).$$

We often write $\mathsf{row}_{\mathcal{L}}$ and $\mathsf{srow}_{\mathcal{L}}$, or simply row and srow, when the parameters are clear from the context.

A table is *closed* if the successor rows are linear combinations of the existing rows in S. To make this precise, we use the linearisation $\mathsf{row}^\sharp$ (Definition 2.1.4), which extends row to linear combinations of words in S.

Definition 5.2.3 (Closedness)**.** Given a language $\mathcal{L}$, a table (S, E) is *closed* if for all $w \in S$ and $a \in A$ there exists $\alpha \in V(S)$ such that $\mathsf{srow}(wa) = \mathsf{row}^\sharp(\alpha)$.

This definition corresponds to the notion of closedness described in Section 5.1.

A further important ingredient of the algorithm is a method for checking whether a table is closed. This is captured by the notion of closedness strategy.

Definition 5.2.4 (Closedness strategy)**.** Given a language $\mathcal{L}$, a *closedness strategy* for $\mathcal{L}$ is a family of computable functions

$$\mathsf{cs} : S \cdot A \longrightarrow \{\bot\} \cup V(S),$$

one for each table $(S, E) \in \mathsf{Table}_{\mathsf{fin}}$, satisfying the following two properties:

- if $\mathsf{cs}(t) = \bot$, then there is no $\alpha \in V(S)$ such that $\mathsf{row}^\sharp(\alpha) = \mathsf{srow}(t)$; and
- if $\mathsf{cs}(t) \neq \bot$, then $\mathsf{row}^\sharp(\mathsf{cs}(t)) = \mathsf{srow}(t)$.

The relevant table will be clear from the context.

Thus, given a closedness strategy as above, a table (S, E) is closed if and only if $\mathsf{cs}(t) \neq \bot$ for all $t \in S \cdot A$. More specifically, for each $t \in S \cdot A$ we have that $\mathsf{cs}(t) \neq \bot$ if and only if the (successor) row corresponding to t already forms a linear combination of rows labelled by S.

In that case, such a linear combination is given by $\mathsf{cs}(t)$. This is used to close tables in our learning algorithm, introduced below.

Examples of semirings and (classes of) languages that admit a closedness strategy are described at the end of this section. Important for our algorithm will be that closedness strategies are computable. This problem is equivalent to solving systems of equations $A\underline{x} = \underline{b}$, where A is the matrix whose columns are $\mathsf{row}(s)$ for $s \in S$, $\underline{x}$ is a vector of length $|S|$, and $\underline{b}$ is the vector consisting of the row entries in $\mathsf{srow}(t)$ for some $t \in S \cdot A$. These observations motivate the following definition.

Definition 5.2.5 (Solvability)**.** A semiring $\mathbb{S}$ is *solvable* if a solution to any finite system of linear equations of the form $A\underline{x} = \underline{b}$ is computable.

We have the following correspondence.

Proposition 5.2.6. *For any language accepted by a WFA over any semiring there exists a closedness strategy if and only if the semiring is solvable.*

Proof. If the semiring is solvable, we obtain a closedness strategy by the remarks prior to Definition 5.2.5. Conversely, we can construct a language that is non-zero on finitely many words and encode in a table (S, E) a given linear equation. To be able to freely choose the value in each table cell, we can consider a sufficiently large alphabet to make sure S and E contain only single-letter words. This avoids dependencies within the table. □

We now have all the ingredients to formulate the algorithm to learn weighted languages over a general semiring. It requires a fixed closedness strategy cs for the target language $\mathcal{L}$. The pseudocode is displayed in Algorithm 5.1.

The algorithm keeps a table (S, E), and starts by initialising both S and E to contain just the empty word. The inner while loop (lines 3–4) uses the closedness strategy to repeatedly check whether the current table is closed, and adds new rows in case it is not. Once the table is closed, a hypothesis is constructed, again using the closedness strategy (lines 5–8). This hypothesis is then given to the teacher for an equivalence check. The equivalence check is modelled by EQ (line 9) as follows: if the hypothesis is incorrect, the teacher non-deterministically returns a counterexample $w \in A^\star$, the condition evaluates to `true`, and the suffixes of w are added to E; otherwise, if the hypothesis is correct, the condition on line 9 evaluates to `false`, and the algorithm returns the correct hypothesis on line 12.

Algorithm 5.1. Abstract learning algorithm for a WFA over $\mathbb{S}$

$S, E \longleftarrow \{\varepsilon\}$
while `true` **do**
  while $\mathsf{cs}(t) = \bot$ for some $t \in S \cdot A$ **do**
    $S \longleftarrow S \cup \{t\}$
  for $s \in S$ **do**
    $o(s) \longleftarrow \mathsf{row}(s)(\varepsilon)$
    for $a \in A$ **do**
      $\delta(s)(a) \longleftarrow \mathsf{cs}(sa)$
  if $\mathsf{EQ}(S, \delta, \varepsilon, o) = w \in A^\star$ **then**
    $E \longleftarrow E \cup \mathsf{suffixes}(w)$
  else
    return $(S, \delta, \varepsilon, o)$

5.2.1 Termination of the General Algorithm

The main question remaining is: under which conditions does this algorithm terminate and hence learns the unknown weighted language? We proceed to give abstract conditions under which it terminates. There are two main requirements:

1. A way of measuring progress the algorithm makes with the observation table when it distinguishes linear combinations of rows that were previously equal, together with a bound on this progress (Definition 5.2.7).

2. An assumption on the *Hankel matrix* of the target language (Definition 5.2.8), which makes sure we encounter finitely many closedness defects throughout any run of the algorithm. More specifically, we assume that the Hankel matrix satisfies a finite approximation property (Definition 5.2.9).

The first assumption is captured by the definition of *progress measure*.

Definition 5.2.7 (Progress measure)**.** A *progress measure* for a language $\mathcal{L}$ is a function $\mathsf{size}_{\mathcal{L}} : \mathsf{Table}_{\mathsf{fin}} \longrightarrow \mathbb{N}$ such that

(a) there exists $n \in \mathbb{N}$ such for all $(S, E) \in \mathsf{Table}_{\mathsf{fin}}$ we have $\mathsf{size}_{\mathcal{L}}(S, E) \leq n$;

(b) given $(S, E), (S, E') \in \mathsf{Table}_{\mathsf{fin}}$ and $s_1, s_2 \in V(S)$ such that $E \subseteq E'$ and $\mathsf{row}^{\#}_{(S,E,\mathcal{L})}(s_1) = \mathsf{row}^{\#}_{(S,E,\mathcal{L})}(s_2)$ but $\mathsf{row}^{\#}_{(S,E',\mathcal{L})}(s_1) \neq \mathsf{row}^{\#}_{(S,E',\mathcal{L})}(s_2)$, we have $\mathsf{size}_{\mathcal{L}}(S, E') > \mathsf{size}_{\mathcal{L}}(S, E)$.

A progress measure assigns a 'size' to each table, in such a way that (a) there is a global bound on the size of tables, and (b) if we extend a table with some proper tests in E, i.e., such that some combinations of rows in $\mathsf{row}^\sharp$ that were equal before get distinguished by a newly added test, then the size of the extended table is properly above the size of the original table. This is used to ensure that, when adding certain counterexamples supplied by the teacher, the size of the table, measured according to the above $\mathsf{size}_{\mathcal{L}}$ function, properly increases.

The second assumption that we use for termination is phrased in terms of the *Hankel matrix* associated with the target language $\mathcal{L}$, which represents $\mathcal{L}$ as (the semimodule generated by) the infinite table where both the rows and columns contain all words. The Hankel matrix is defined as follows.

Definition 5.2.8 (Hankel matrix)**.** Given a language $\mathcal{L} : A^\star \longrightarrow \mathbb{S}$, the *semimodule generated by a table* (S, E) is given by the image of $\mathsf{row}^\sharp$. We refer to the semimodule generated by the table $(A^\star, A^\star)$ as the *Hankel matrix* of $\mathcal{L}$.

The Hankel matrix is approximated by the tables that occur during the execution of the algorithm. For termination, we will therefore assume that this matrix satisfies the following finite approximation condition.

Definition 5.2.9 (Ascending chain condition)**.** We say that a semimodule M satisfies the *ascending chain condition* if for all inclusion chains of subsemimodules of M,

$$S_1 \subseteq S_2 \subseteq S_3 \subseteq \cdots,$$

there exists $n \in \mathbb{N}$ such that for all $m \geq n$ we have $S_m = S_n$.

Given the notions of progress measure, Hankel matrix and ascending chain condition, we can formulate the general theorem for termination of Algorithm 5.1.

Theorem 5.2.10 (Termination of the abstract learning algorithm)**.** *In the presence of a progress measure, Algorithm 5.1 terminates whenever the Hankel matrix of the target language $\mathcal{L}$ satisfies the ascending chain condition (Definition 5.2.9).*

Proof. Suppose the algorithm does not terminate. Then there is a sequence $\{(S_n, E_n)\}_{n\in\mathbb{N}}$ of tables where (S_0, E_0) is the initial table and (S_{n+1}, E_{n+1}) is formed from (S_n, E_n) after resolving a closedness defect or adding columns due to a counterexample.

We write H_n for the semimodule generated by the table $(S_n, A^\star)$. We have $S_n \subseteq S_{n+1}$ and thus $H_n \subseteq H_{n+1}$. Note that a closedness defect for (S_n, E_n) is also a closedness defect for $(S_n, A^\star)$, so if we resolve the defect in the next step, the inclusion $H_n \subseteq H_{n+1}$ is strict. Since these are

all included in the Hankel matrix, which satisfies the ascending chain condition, there must be an n such that for all $k \geq n$ we have that (S_k, E_k) is closed.

In Section 6.5 we will show that in a general table used for learning automata with side-effects given by a monad there exists a suffix of each counterexample for the corresponding hypothesis that when added as a column label leads to either a closedness defect or to distinguishing two combinations of rows in the table (see Proposition 6.5.3 and the discussion preceding it). Since WFAs are automata with side-effects given by the free semimodule monad[2] and we add all suffixes of the counterexample to the set of column labels, this also happens in our algorithm. Thus, for all $k \geq n$ where we process a counterexample, there must be two linear combinations of rows distinguished, as closedness is already guaranteed. Then the semimodule generated by (S_k, E_k) is a strict quotient of the semimodule generated by (S_{k+1}, E_{k+1}). By the progress measure we then find $\mathsf{size}_{\mathcal{L}}(S_k, E_k) < \mathsf{size}_{\mathcal{L}}(S_{k+1}, E_{k+1})$, which cannot happen infinitely often. We conclude that the algorithm must terminate. □

To illustrate the ingredients required for Algorithm 5.1 and its termination (Theorem 5.2.10), we consider two classes of semirings for which learning algorithms are already known in the literature [BV96].

Example 5.2.11 (Weighted languages over fields)**.** Consider any field for which the basic operations are computable. Solvability is then satisfied via a procedure such as Gaussian elimination, so by Proposition 5.2.6 there exists a closedness strategy. Hence, we can instantiate Algorithm 5.1 with $ being such a field.

For termination, we show that the hypotheses of Theorem 5.2.10 are satisfied whenever the target language is accepted by a WFA. First, a progress measure is given by the dimension of the vector space generated by the table. To see this, note that if we distinguish two linear combinations of rows, we can assume without loss of generality that one of these linear combinations in the extended table uses only basis elements. This in turn can be rewritten to distinguishing a single row from a linear combination of rows using field operations, with the property that the extended version of the single row is a basis element. Hence, the row was not a basis element in the original table, and therefore the dimension of the vector space generated by the table has increased. Adding rows and columns cannot decrease this dimension, so it is bounded by the dimension of the Hankel matrix. Since the language we want to learn is accepted by a WFA, the associated Hankel matrix has a finite dimension [CP71; Fli74] (see also, e.g., [BM12]), providing a bound for our progress measure.

[2]We note that Chapter 6 assumes the monad to preserve finite sets. However, the relevant arguments do not depend on this.

Finally, for any ascending chain of subspaces of the Hankel matrix, these subspaces are of finite dimension bounded by the dimension of the Hankel matrix. The dimension increases along a strict subspace relation, so the chain converges.

Example 5.2.12 (Weighted languages over finite semirings)**.** Consider any finite semiring. Finiteness allows us to apply a brute force approach to solving systems of equations. This means the semiring is solvable, and hence a closedness strategy exists by Proposition 5.2.6.

For termination, we can define a progress measure by assigning to each table the size of the image of $\mathsf{row}^\sharp$. Distinguishing two linear combinations of rows increases this measure. If the language we want to learn is accepted by a WFA, then the Hankel matrix contains a subset of the linear combinations of the languages of its states. Since there are only finitely many such linear combinations, the Hankel matrix is finite, which bounds our measure. A finite semimodule such as the Hankel matrix in this case does not admit infinite chains of subspaces, which means the ascending chain condition is satisfied. We conclude by Theorem 5.2.10 that Algorithm 5.1 terminates for the instance that the semiring $\mathbb{S}$ is finite if the target language is accepted by a WFA over $\mathbb{S}$.

For the Boolean semiring, an instance of the above finite semiring example, WFAs are nondeterministic finite automata. The algorithm we recover by instantiating Algorithm 5.1 to this case is close to the algorithm first described by Bollig et al. [Bol+09]. The main differences are that in their case the hypothesis has a state space given by a minimally generating subset of the distinct rows in the table rather than all elements of S, and they do apply a notion of consistency.

In Section 5.4 we will show that Algorithm 5.1 can learn WFAs over principal ideal domains—notably including the integers—thus providing a strict generalisation of existing techniques.

5.3 Issues with Arbitrary Semirings

We concluded the previous section with examples of semirings for which Algorithm 5.1 terminates if the target language is accepted by a WFA. In this section, we prove a negative result for the algorithm over the semiring $\mathbb{N}$: we show that it does not terminate on a certain language over $\mathbb{N}$ accepted by a WFA over $\mathbb{N}$, as anticipated in Section 5.1.2. This means that Algorithm 5.1 does not work well for arbitrary semirings. The problem is that the Hankel matrix of a language recognised by WFA does not necessarily satisfy the ascending chain

condition that is used to prove Theorem 5.2.10. In the example given in the proof below, the Hankel matrix is not even finitely generated.

Theorem 5.3.1. *There exists a WFA $\mathcal{A}_{\mathbb{N}}$ over $\mathbb{N}$ such that Algorithm 5.1 does not terminate when given $\mathcal{L}_{\mathcal{A}_{\mathbb{N}}}$ as target, regardless of the closedness strategy used.*

Proof. Let $\mathcal{A}_{\mathbb{N}}$ be the automaton over the alphabet $\{a\}$ given in (5.1) in Section 5.1.2. Formally, $\mathcal{A}_{\mathbb{N}} = (Q, \delta, i, o)$, where

$$Q = \{q_0, q_1\} \qquad i = q_0 \qquad o(q_0) = 0$$
$$\delta(q_0)(a) = q_0 + q_1 \qquad \delta(q_1)(a) = 2q_1 \qquad o(q_1) = 1.$$

As mentioned in Section 5.1.2, the language $\mathcal{L} : \{a\}^\star \to \mathbb{N}$ accepted by $\mathcal{A}_{\mathbb{N}}$ is given by $\mathcal{L}(a^j) = 2^j - 1$. This can be shown more precisely as follows. First one shows by induction on j that $\mathsf{obs}_{\mathcal{A}_{\mathbb{N}}}(q_1)(a^j) = 2^j$ for all $j \in \mathbb{N}$—we leave the straightforward argument to the reader. Second, we show, again by induction on j, that $\mathsf{obs}_{\mathcal{A}_{\mathbb{N}}}(q_0)(a^j) = 2^j - 1$. This implies the claim, as $\mathcal{L} = \mathsf{obs}_{\mathcal{A}_{\mathbb{N}}}(q_0)$. For $j = 0$ we have $\mathsf{obs}_{\mathcal{A}_{\mathbb{N}}}(q_0)(a^j) = o(q_0) = 0 = 2^0 - 1$ as required. For the inductive step, let $j = k + 1$ and assume $\mathsf{obs}_{\mathcal{A}_{\mathbb{N}}}(q_0)(a^k) = 2^k - 1$. We calculate

$$\begin{aligned}
\mathsf{obs}_{\mathcal{A}_{\mathbb{N}}}(q_0)(a^{k+1}) &= \mathsf{obs}_{\mathcal{A}_{\mathbb{N}}}(q_0 + q_1)(a^k) \\
&= \mathsf{obs}_{\mathcal{A}_{\mathbb{N}}}(q_0)(a^k) + \mathsf{obs}_{\mathcal{A}_{\mathbb{N}}}(q_1)(a^k) \\
&= (2^k - 1) + 2^k \\
&= 2^{k+1} - 1.
\end{aligned}$$

Note that in particular the language $\mathcal{L}$ is injective.

Towards a contradiction, suppose the algorithm does terminate with table (S, E). Let $J = \{j \in \mathbb{N} \mid a^j \in S\}$ and define $n = \max(J)$. Since the algorithm terminates with table (S, E), the latter must be closed. In particular, there exist $k_j \in \mathbb{N}$ for all $j \in J$ such that $\sum_{j \in J} k_j \cdot \mathsf{row}_{\mathcal{L}}(a^j) = \mathsf{srow}_{\mathcal{L}}(a^n a)$. We consider two cases. First assume $E = \{\varepsilon\}$ and let $\mathcal{A} = (Q', \delta', i', o')$ be the hypothesis. For all $l \in \mathbb{N}$ we have $\mathsf{row}^\sharp_{\mathcal{L}}(\mathsf{reach}^\dagger_{\mathcal{A}}(a^l))(\varepsilon) = 2^l - 1$ because $\mathcal{A}$ must be correct. Thus, if $a^l \in S \cdot A$, then $\mathsf{row}^\sharp_{\mathcal{L}}(\mathsf{reach}^\dagger_{\mathcal{A}}(a^l)) = \mathsf{srow}_{\mathcal{L}}(a^l)$. In particular,

$$\mathsf{row}^\sharp_{\mathcal{L}}(\mathsf{reach}^\dagger_{\mathcal{A}}(a^n a)) = \mathsf{srow}_{\mathcal{L}}(a^n a) = \sum_{j \in J} k_j \cdot \mathsf{row}_{\mathcal{L}}(a^j).$$

Note that we can choose the k_j such that $\mathsf{reach}^{\dagger}_{\mathcal{A}}(a^n a) = \sum_{j \in J} k_j \cdot a^j$. Since

$$\begin{aligned}\mathsf{row}^{\#}_{\mathcal{L}}\left(\delta'^{\#}\left(\sum_{j\in J} k_j \cdot a^j\right)(a)\right) &= \mathsf{row}^{\#}_{\mathcal{L}}\left(\sum_{j\in J} k_j \cdot \delta'(a^j)(a)\right)\\ &= \sum_{j\in J} k_j \cdot \mathsf{row}_{\mathcal{L}}(\delta'(a^j)(a))\\ &= \sum_{j\in J} k_j \cdot \mathsf{srow}_{\mathcal{L}}(a^j a),\end{aligned}$$

we have

$$\mathsf{row}^{\#}_{\mathcal{L}}(\mathsf{reach}^{\dagger}_{\mathcal{A}}(a^n aa)) = \sum_{j\in J} k_j \cdot \mathsf{srow}_{\mathcal{L}}(a^j a)$$

and therefore

$$\sum_{j\in J} k_j \cdot \mathsf{srow}_{\mathcal{L}}(a^j a)(\varepsilon) = \mathsf{row}^{\#}_{\mathcal{L}}(\mathsf{reach}^{\dagger}_{\mathcal{A}}(a^n aa))(\varepsilon) = 2^{n+2} - 1.$$

Then

$$\begin{aligned}2^{n+2} - 1 = \sum_{j\in J} k_j \cdot \mathsf{srow}_{\mathcal{L}}(a^j a)(\varepsilon) &= \sum_{j\in J} k_j(2^{j+1} - 1)\\ &= 2\left(\sum_{j\in J} k_j(2^j - 1)\right) + \sum_{j\in J} k_j = 2(2^{n+1} - 1) + \sum_{j\in J} k_j,\end{aligned}$$

so $\sum_{j\in J} k_j = 1$. This is only possible if there is $j_1 \in J$ such that $k_{j_1} = 1$ and $k_j = 0$ for all $j \in J \setminus \{j_1\}$. However, this implies that $\mathsf{row}_{\mathcal{L}}(a^{j_1}) = \mathsf{srow}_{\mathcal{L}}(a^n a)$, which contradicts injectivity of $\mathcal{L}$ as $n \geq j_1$. Thus, the algorithm did not terminate.

For the other case, assume there is $a^m \in E$ such that $m \geq 1$. We have

$$2^{n+1} - 1 = \mathsf{srow}_{\mathcal{L}}(a^n a)(\varepsilon) = \sum_{j\in J} k_j \cdot \mathsf{row}_{\mathcal{L}}(a^j)(\varepsilon) = \sum_{j\in J} k_j(2^j - 1),$$

so we can calculate

$$\begin{aligned}
\sum_{j\in J} k_j(2^{j+m}-1) &= \sum_{j\in J} k_j \cdot \mathsf{row}_{\mathcal{L}}(a^j)(a^m) \\
&= \mathsf{srow}_{\mathcal{L}}(a^n a)(a^m) \\
&= 2^{n+m+1}-1 \\
&= 2^m(2^{n+1}-1)+2^m-1 \\
&= 2^m\left(\sum_{j\in J} k_j(2^j-1)\right)+2^m-1 \\
&= \left(\sum_{j\in J} k_j(2^{j+m}-2^m)\right)+2^m-1 \\
&= \left(\sum_{j\in J} k_j(2^{j+m}-1)\right)+\left(\sum_{j\in J} k_j(1-2^m)\right)+2^m-1
\end{aligned}$$

and conclude that

$$\left(\sum_{j\in J} k_j(1-2^m)\right)+2^m-1=0.$$

Since $m \geq 1$ this is only possible if there is $j_1 \in J$ such that $k_{j_1} = 1$ and $k_j = 0$ for all $j \in J \setminus \{j_1\}$. However, this implies $\mathsf{row}_{\mathcal{L}}(a^{j_1}) = \mathsf{srow}_{\mathcal{L}}(a^n a)$, which again contradicts injectivity of $\mathcal{L}$ as $n \geq j_1$. Thus, the algorithm did not terminate. □

Remark 5.3.2. Our proof shows non-termination for a bigger class of algorithms than Algorithm 5.1; it uses only the definition of the hypothesis, that closedness is satisfied before constructing the hypothesis, that S and E contain the empty word, and that termination implies correctness. For instance, adding the prefixes of a counterexample to S instead of its suffixes to E will not fix the issue.

We have thus shown that our algorithm does not instantiate to a terminating one for an arbitrary semiring. To contrast this negative result, in the next section we identify a class of semirings not previously explored in the learning literature where we can guarantee a terminating instantiation.

5.4 Learning WFAs over PIDs

We show that for a subclass of semirings, namely *principal ideal domains (PIDs)*, the abstract learning algorithm of Section 5.2 terminates. This subclass includes the integers, Gaussian integers, and rings of polynomials in one variable with coefficients in a field. We will prove that

the Hankel matrix of a language over a PID accepted by a WFA has analogous properties to those of vector spaces—finite rank, a notion of progress measure, and the ascending chain condition. We also give a sufficient condition for PIDs to be solvable, which by Proposition 5.2.6 guarantees the existence of a closedness strategy for the learning algorithm.

To define PIDs, we first need to introduce ideals. Given a ring $\mathbb{S}$, a *(left) ideal* I of $\mathbb{S}$ is an additive subgroup of $\mathbb{S}$ such that for all $s \in \mathbb{S}$ and $i \in I$ we have $si \in I$. The ideal I is *(left) principal* if it is of the form $I = \mathbb{S}s$ for some $s \in \mathbb{S}$.

Definition 5.4.1 (PID)**.** A *principal ideal domain* $\mathbb{P}$ is a non-zero commutative ring in which every ideal is principal and where for all $p_1, p_2 \in \mathbb{P}$ such that $p_1 p_2 = 0$ we have $p_1 = 0$ or $p_2 = 0$.

A module M over a PID $\mathbb{P}$ is called *torsion free* if for all $p \in \mathbb{P}$ and any $m \in M$ such that $p \cdot m = 0$ we have $p = 0$ or $m = 0$. It is a standard result that a module over a PID is torsion free if and only if it is free [Jac12, Theorem 3.10].

The next definition of *rank* is analogous to that of the dimension of a vector space and will form the basis for the progress measure.

Definition 5.4.2 (Rank)**.** We define the *rank* of a finitely generated free module $V(X)$ over a PID as $\mathsf{rank}(V(X)) = |X|$.

This definition extends to any finitely generated free module over a PID, as $V(X) \cong V(Y)$ for finite sets X and Y implies $|X| = |Y|$ [Jac12, Theorem 3.4].

Now that we have a candidate for a progress measure function, we need to prove it has the required properties. The following lemmas will help with this.

Lemma 5.4.3. *Given finitely generated free modules M, N over a PID such that $\mathsf{rank}(M) \geq \mathsf{rank}(N)$, any surjective module homomorphism $f : N \longrightarrow M$ is injective.*

Proof. Since $\mathsf{rank}(M) \geq \mathsf{rank}(N)$, there exists a surjective module homomorphism $g : M \longrightarrow N$. Therefore $g \circ f : N \longrightarrow N$ is surjective and by [Orz71] an iso. In particular, f is injective. □

Lemma 5.4.4. *If M and N are finitely generated free modules over a PID such that there exists a surjective module homomorphism $f : N \longrightarrow M$, then $\mathsf{rank}(M) \leq \mathsf{rank}(N)$. If f is not injective, then $\mathsf{rank}(M) < \mathsf{rank}(N)$.*

Proof. Let $f : N \longrightarrow M$ be a surjective module homomorphism. Suppose towards a contradiction that $\mathsf{rank}(M) > \mathsf{rank}(N)$. By Lemma 5.4.3 f is injective, so M is isomorphic to a submodule of N and $\mathsf{rank}(M) \leq \mathsf{rank}(N)$ [Jac12]; contradiction.

For the second part, suppose f is not injective and assume towards a contradiction that $\mathsf{rank}(M) \geq \mathsf{rank}(N)$. Again by Lemma 5.4.3 f is injective, which is a contradiction with our assumption. Thus, in this case $\mathsf{rank}(M) < \mathsf{rank}(N)$. □

The lemma below states that the Hankel matrix of a weighted language over a PID has finite rank which bounds the rank of any module generated by an observation table. This will be used to define a progress measure, used to prove termination of the learning algorithm for weighted languages over PIDs.

Lemma 5.4.5 (Hankel matrix rank for PIDs)**.** *When targeting a language accepted by a WFA over a PID, any module generated by an observation table is free. Moreover, the Hankel matrix has finite rank that bounds the rank of any module generated by an observation table.*

Proof. Given a WFA $\mathcal{A} = (Q, \delta, i, o)$, let M be the free module generated by Q. Note that the Hankel matrix is the image of the composition $\mathsf{obs}_{\mathcal{A}} \circ \mathsf{reach}_{\mathcal{A}}$. Consider the image of the module homomorphism $\mathsf{reach}_{\mathcal{A}} : V(A^\star) \longrightarrow M$, which we write as R. Since R is a submodule of M, we know from [Jac12] that R is free and finitely generated with $\mathsf{rank}(R) \leq \mathsf{rank}(M)$. The Hankel matrix can now be obtained as the image of the restriction of $\mathsf{obs}_{\mathcal{A}} : M \longrightarrow \mathbb{S}^{A^\star}$ to the domain R. Let H be this image, which we know is finitely generated because R is. Since H is a submodule of the torsion free module $\mathbb{S}^{A^\star}$, it is also torsion free and therefore free. We also have a surjective module homomorphism $s : R \longrightarrow H$, so by Lemma 5.4.4 we find $\mathsf{rank}(H) \leq \mathsf{rank}(R)$.

Let N be the module generated by an observation table (S, E). We have that N is a quotient of the module generated by $(S, A^\star)$, which in turn is a submodule of H. Using again [Jac12] and Lemma 5.4.4 we conclude that N is free and finitely generated with $\mathsf{rank}(N) \leq \mathsf{rank}(H)$. □

The fact that the Hankel matrix has finite rank in the statement of Lemma 5.4.5 above would follow from a PID variant of Fliess's theorem [Fli74], which states that the size of the minimal WFA over a field is the dimension of the Hankel matrix of its language. We are not aware of such a generalisation and leave this for future work.

Proposition 5.4.6 (Progress measure for PIDs)**.** *There exists a progress measure for any language $\mathcal{L}$ accepted by a WFA over a PID.*

Proof. Define $\mathsf{size}_{\mathcal{L}}(S, E) = \mathsf{rank}(M)$, where M is the module generated by the table (S, E). By Lemma 5.4.5 this is bounded by the rank of the Hankel matrix. If M and N are modules generated by two tables such that N is a strict quotient of M, then by Lemma 5.4.4 we have $\mathsf{rank}(M) > \mathsf{rank}(N)$. □

Recall that, for termination of the algorithm, Theorem 5.2.10 requires a progress measure, which we defined above, and it requires the Hankel matrix of the language to satisfy the ascending chain condition (Definition 5.2.9). Proposition 5.4.7 shows that the latter is always the case for languages over PIDs.

Proposition 5.4.7 (Ascending chain condition PIDs)**.** *The Hankel matrix of a language accepted by a WFA over a PID satisfies the ascending chain condition.*

Proof. Let H be the Hankel matrix, which has finite rank by Lemma 5.4.5. If

$$M_1 \subseteq M_2 \subseteq M_3 \subseteq \cdots$$

is any chain of submodules of H, then $M = \bigcup_{i \in \mathbb{N}} M_i$ is a submodule of H and therefore also of finite rank [Jac12]. Let B be a finite basis of M. There exists $n \in \mathbb{N}$ such that $B \subseteq M_n$, so $M_n = M$. □

The last ingredient for the abstract algorithm is solvability of the semiring: the following fact provides a sufficient condition for a PID to be solvable.

Proposition 5.4.8 (PID solvability)**.** *A PID* $\mathbb{P}$ *is solvable if all of its ring operations are computable and if each element of* $\mathbb{P}$ *can be* effectively *factorised into irreducible elements.*

Proof. It is well-known that a system of equations of the form $A\underline{x} = \underline{b}$ with integer coefficients can be efficiently solved via computing the Smith normal form [Smi61] of A. The algorithm generalises to principal ideal domains, if we assume that the factorisation of any given element of the principal ideal domain[3] into irreducible elements is computable, cf. the algorithm in [Jac53, p. 79-84]. To see that all steps in this algorithm can be computed, one has to keep in mind that the factorisation can be used to determine the greatest common divisor of any two elements of the principal ideal domain. □

Remark 5.4.9. In the case that we are dealing with an Euclidean domain $\mathbb{P}$, a sufficient condition for $\mathbb{P}$ to be solvable is that Euclidean division is computable (again this can be deduced from inspecting the algorithm in [Jac53, p. 79-84]). Such a PID behaves essentially like the ring of integers.

Putting everything together, we obtain the main result of this section.

[3]Note that factorisations exist as each principal ideal domain is also a unique factorisation domain, cf. e.g. [Jac12, Thm. 2.23].

Theorem 5.4.10 (Termination for PIDs)**.** *Algorithm 5.1 can be instantiated and terminates for any language accepted by a WFA over a PID of which all ring operations are computable and of which each element can be effectively factorised into irreducible elements.*

Proof. To instantiate the algorithm, we need a closedness strategy. According to Proposition 5.2.6 it is sufficient for the PID to be solvable, which is shown by Proposition 5.4.8. Proposition 5.4.6 provides a progress measure, and we know from Proposition 5.4.7 that the Hankel matrix satisfies the ascending chain condition, so by Theorem 5.2.10 the algorithm terminates. □

The example run given in Section 5.1.1 is the same when performed over the integers. We note that if the teacher holds an automaton model of the correct language, equivalence queries are decidable by lifting the embedding of the PID into its *quotient field* to the level of WFAs and checking equivalence there.

5.5 Discussion

We have introduced a general algorithm for learning WFAs over arbitrary semirings, together with sufficient conditions for termination. We have shown an inherent termination issue over the natural numbers and proved termination for a subclass of semirings—principal ideal domains (PIDs). Our work extends the results by Bergadano and Varricchio [BV96], who showed that WFAs over fields could be learned from a teacher. Although a PID can be embedded into its corresponding field of fractions, we note that the WFAs produced when learning over this field potentially have weights outside the PID.

On the technical level, a variation on WFAs is given by probabilistic automata, where transitions point to convex rather than linear combinations of states. One easily adapts the example from Section 5.3 to show that learning probabilistic automata has a similar termination issue. On the positive side, Tappler et al. [Tap+19] have shown that deterministic MDPs can be learned using an $L^\star$ based algorithm. The deterministic MDPs in *loc.cit.* are very different from the automata in our paper, as their states generate observable output that allows to identify the current state based on the generated input-output sequence.

Algorithmic issues with WFAs over arbitrary semirings have been identified before. For instance, Krob [Kro94] showed that language equivalence is undecidable for WFAs over the tropical semiring. We conjecture that for each semiring over which language equivalence of WFAs is undecidable our algorithm will not in general terminate. This conjecture is based on the connection between learning and testing [Ber+05; HSS17a]: one should be able to

transform the learning algorithm into an equivalence testing method. We do not believe the converse implication, that non-termination of our algorithm implies language equivalence is undecidable, to be true. Evidence for this can be seen from the similar setting of probabilistic automata mentioned above. Although the adapted learning algorithm does not terminate, equivalence is easily decided by considering the automata as WFAs over the field of real numbers and testing equivalence on that level.

One drawback of the ascending chain condition on the Hankel matrix is that this does not give any indication of the number of steps the algorithm requires. Indeed, the submodule chains traversed, although converging, may be arbitrarily long. We would like to measure and bound the progress made when fixing closedness defects, but this turns out to be challenging for PIDs: the rank of the module generated by the table may not increase. We leave an investigation of alternative measures to future work.

Our counterexample over the natural numbers justifies the ascending chain condition imposed on the Hankel matrix, as the condition breaks in that case. We have not identified an example where the lack of a progress measure leads to the algorithm failing to terminate. Future research will have to determine whether the progress measure requirement is essential.

We would like to adapt the algorithm so that for PIDs it always produces minimal automata. At the moment this is already the case for fields,[4] since adding a row due to a closedness defect preserves linear independence of the image of row. For PIDs things are more complicated—adding rows towards closedness may break linear independence and thus a basis needs to be found in $\mathsf{row}^\sharp$. This complicates the construction of the hypothesis.

Our results show that, on the one hand, WFAs can be learned over finite semirings and arbitrary PIDs (assuming computability of the relevant operations) and, on the other hand, that there exists an infinite commutative semiring for which they cannot be learned. However, there are many classes of semirings in between commutative semirings and PIDs such as integral domains, GCD domains, and unique factorisation domains. For such classes we would like to conclusively know whether their WFAs can be learned by our general algorithm.

[4] There is one exception: the language that assigns 0 to every word, which is accepted by a WFA with no states. The algorithm initialises the set of row labels, which constitute the state space of the hypothesis, with the empty word.

Chapter 6

Learning Automata with Side-Effects

A limitation in automata learning is that the state spaces of models of real systems can become too large to be handled by tools. This demands compositional methods and techniques that enable compact representation of behaviors. In this chapter, we show how monads can be used to add optimisations to learning algorithms in order to obtain compact representations. Monads allow us to take an abstract approach, in which category theory is used to devise an optimised learning algorithm and a generic correctness proof for a broad class of compact models. We direct our attention to monads on **Set** and accordingly provide concrete algorithms. This chapter can be seen as an extended case study of a large class of instances of the general algorithm presented in Chapter 4.

The inspiration for this work is quite concrete: it is a well-known fact that non-deterministic finite automata (NFAs) can be much smaller than deterministic ones for a regular language. The subtle point is that given a regular language, there is a canonical deterministic automaton accepting it—the minimal one—but there might be many "minimal" non-deterministic automata accepting the same language. This raises a challenge for learning algorithms: which non-deterministic automaton should the algorithm learn? In one answer to this, Bollig et al. [Bol+09] developed a version of Angluin's $\mathtt{L}^\star$ algorithm, $\mathtt{NL}^\star$, in which they use a particular class of NFAs, *Residual Finite State Automata* (RFSAs). These do admit minimal canonical representatives. Although $\mathtt{NL}^\star$ indeed is a first step in incorporating a more compact representation of regular languages, there are several questions that remain to be addressed.

DFAs and NFAs are formally connected by the subset construction. Underpinning this construction is the rich algebraic structure of the set of languages and of the state space of the DFA obtained by determinising an NFA. The state space of a determinised DFA—consisting of

subsets of the state space of the original NFA—has a join-semilattice structure. Moreover, this structure is preserved in language acceptance: given subsets of states U and V, the language of $U \cup V$ is the union of the languages of the first two. Formally, the function that assigns to each state its language is a join-semilattice map into the join-semilattice of languages structured with the subset relation. The set of languages is even richer: it has the structure of a complete atomic Boolean algebra. This leads to several questions: Can we exploit this structure and have even more compact representations? What if we slightly change the setting and look at weighted languages over a semiring, which have the structure of a semimodule (or vector space, if the semiring is a field)?

The latter question is strongly motivated by the widespread use of weighted languages and corresponding *weighted finite automata* (WFAs) in verification, from the formal verification of quantitative properties [CDH08; DG05; Kup14], to probabilistic model-checking [BGC09], to the verification of on-line algorithms [AKL10]. These are the automata we generalised a learning algorithm for in Chapter 5.

Our key insight is that the algebraic structures mentioned above are in fact algebras for a monad T. In the case of join-semilattices this is the powerset monad, and in the case of vector spaces it is the free vector space monad. These monads can be used to define a notion of T-automaton, with states having the structure of an algebra for the monad T, which generalises non-determinism as a side-effect. From T-automata we can derive a compact, equivalent version by taking as states a set of *generators* and transferring the algebraic structure of the original state space to the transition structure of the automaton.

This perspective enables us to generalise $\mathrm{L}^\star$ to a new algorithm $\mathrm{L}^\star_T$, which learns compact automata featuring non-determinism and other side-effects captured by a monad. Moreover, $\mathrm{L}^\star_T$ incorporates further optimisations arising from the monadic representation, which lead to more scalable algorithms.

Besides the theoretical aspects, we devote large part of this chapter to implementation and experimental evaluation. Monads are key for us to faithfully translate theory into practice. We provide a library that implements all aspects of our general framework, making use of Haskell monads. For any monad, the library allows the programmer to obtain a basic, correct-by-construction instance of the algorithm and of its optimised versions for free. This enables the programmer to experiment with different optimisations with minimal effort. Our library also allows the programmer to re-define some basic operations, if a more efficient version is available, in order to make the algorithm more amenable to real-world usage. For instance, generators can be computed efficiently in the vector space case via Gaussian elimination. The library already provides efficient algorithms for weighted and non-deterministic automata.

We start by giving an overview of our approach in Section 6.1, which is followed by preliminary notions regarding automata with side-effects in Section 6.2. We then present the main contributions:

1. In Section 6.3, we develop a general algorithm $\mathrm{L}^\star_T$, which generalises the NFA one we will discuss in Section 6.1 to an arbitrary monad T capturing side-effects, and we provide a general correctness proof for our algorithm.

2. In Section 6.4, we describe the first optimisation, which replaces the hypothesis by a succinct one, and prove its correctness.

3. In Section 6.5 we describe the second optimisation that replaces the counterexample handling method. We also show how it can be combined with the optimisation of Section 6.4, and how it can lead to a further small optimisation, where the consistency check on the table is dropped. We show that the correctness proof remains valid for each of these modifications of the algorithm.

4. In Section 6.6 we show how $\mathrm{L}^\star_T$ can be applied to several automata models, highlighting further case-specific optimisations when available.

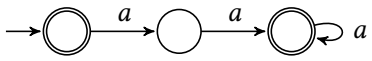
5. In Section 6.7 we describe our library and explain in detail how it can be instantiated to NFAs and WFAs. The implementation of monads for these two cases is non-trivial, due to specific Haskell requirements. We also give efficient versions of both instances. To the best of our knowledge, we are the first ones to implement an Angluin-style learning algorithm for WFAs, and to provide optimisations for it.

6. Finally, in Section 6.8 we describe experimental results for the non-deterministic and weighted cases, comparing all the optimisations enabled by our library. In particular, for NFAs we show that the counterexample handling optimisation, not available to Bollig et al. [Bol+09], leads to an improvement in the number of membership queries, as happens in the DFA case.

6.1 Overview of the Approach

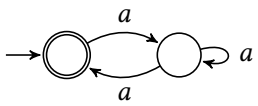
In this section, we discuss the challenges in adapting the $\mathrm{L}^\star$ algorithm to learn automata with side-effects, illustrating them through a concrete example—NFAs. We then highlight our main contributions.

Learning non-deterministic automata. As is well known, NFAs can be smaller than the minimal DFA for a given language. For example, the language $\mathcal{L} = \{w \in \{a\}^\star \mid |w| \neq 1\}$ accepted by the minimal DFA



(6.1)

is also accepted by the smaller NFA



(6.2)

Though in this example, which we chose for simplicity, the state reduction is not massive, it is known that in general NFAs can be exponentially smaller than the minimal DFA [Koz12].

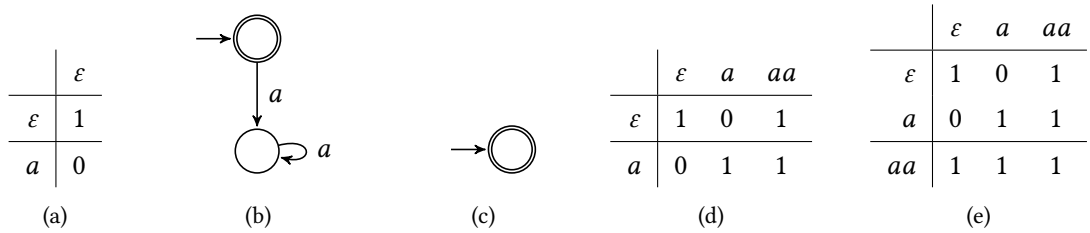
Learning NFAs can lead to a substantial gain in space complexity, but it is challenging. The main difficulty is that NFAs do not have a canonical minimal representative: there may be several non-isomorphic state-minimal NFAs accepting the same language, which poses problems for the development of the learning algorithm. In one answer to this, Bollig et al. [Bol+09] proposed to use a particular class of NFAs, namely RFSAs, which do admit minimal canonical representatives. However, their solution for NFAs does not extend to other automata, such as weighted or alternating. In this chapter we present a solution that works for any side-effect, specified as a monad.

The crucial observation underlying our approach is that the language semantics of an NFA is defined in terms of its determinisation, i.e., the DFA obtained by taking sets of states of the NFA as state space. In other words, this DFA is defined over an algebraic structure induced by the powerset, namely a (complete) *join semilattice* (JSL) whose join operation is set union. This automaton model does admit minimal representatives, which leads to the key idea for our algorithm: learning NFAs as automata over JSLs. In order to do so, we use an extended table where rows have a JSL structure, defined as follows. The join of two rows is given by an element-wise or, and the bottom element is the row containing only zeroes. More precisely, the new table consists of the two functions

$$\mathsf{row}^\sharp : \mathcal{P}(S) \longrightarrow 2^E \qquad\qquad \mathsf{srow}^\sharp : \mathcal{P}(S) \longrightarrow (2^E)^A$$

given by $\mathsf{row}^\sharp(U) = \bigvee\{\mathsf{row}(s) \mid s \in U\}$ and $\mathsf{srow}^\sharp(U)(a) = \bigvee\{\mathsf{srow}(s)(a) \mid s \in U\}$. Formally, these functions are JSL homomorphisms, and they induce the following general definitions:

- The table is *closed* if for all $U \subseteq S$, $a \in A$ there is $U' \subseteq S$ such that $\mathsf{row}^\sharp(U') = \mathsf{srow}^\sharp(U)(a)$.



	ε
ε	1
a	0

(a)

(b)

(c)

	ε	a	aa
ε	1	0	1
a	0	1	1

(d)

	ε	a	aa
ε	1	0	1
a	0	1	1
aa	1	1	1

(e)

Figure 6.1: Example run of the $\mathrm{L}^\star$ adaptation for NFAs on $\mathcal{L} = \{w \in \{a\}^\star \mid |w| \neq 1\}$.

- The table is *consistent* if for all $U_1, U_2 \subseteq S$ such that $\mathsf{row}^\#(U_1) = \mathsf{row}^\#(U_2)$ we have $\mathsf{srow}^\#(U_1) = \mathsf{srow}^\#(U_2)$.

We remark that our algorithm does not actually store the whole extended table, which can be quite large. It only needs to store the original table over S because all other rows in $\mathcal{P}(S)$ are freely generated and can be computed as needed, with no additional membership queries. Thus, we remain close to the original $\mathrm{L}^\star$ algorithm (Section 2.3). The only lines in Figure 2.1 that need to be adjusted are lines 4 and 7 of Algorithm 2.1, where closedness and consistency are replaced with the new notions given above. Moreover, the hypothesis is now built from the extended table.

Optimisations. We also present two optimisations to our algorithm. For the first one, note that the state space of the hypothesis constructed by the algorithm can be very large since it encodes the entire algebraic structure. We show that we can extract a *minimal set of generators* from the table and compute a *succinct hypothesis* in the form of an automaton with side-effects, without any algebraic structure. For JSLs, this consists in only taking rows that are not the join of other rows, i.e., the *join-irreducibles*. By applying this optimisation to this specific case, we essentially recover the learning algorithm of Bollig et al. [Bol+09]. The second optimisation is a generalisation of the optimised counterexample handling method of Rivest and Schapire [RS93], originally intended for $\mathrm{L}^\star$ and DFAs. Recall that the Maler and Pnueli [MP95] variation on $\mathrm{L}^\star$ adds all suffixes of a given counterexample to the column set E (Figure 2.3 in Section 2.3). The optimisation due to Rivest and Schapire [RS93] consists in processing counterexamples by adding a *single* suffix of the counterexample to E. This can avoid the algorithm posing a large number of membership queries.

Example run. We now run the new algorithm on the language $\mathcal{L} = \{w \in \{a\}^\star \mid |w| \neq 1\}$ considered earlier. Starting from $S = E = \{\varepsilon\}$, the observation table (Figure 6.1a) is im-

mediately closed and consistent. (It is closed because we have $\mathsf{row}^\sharp(\{a\}) = \mathsf{row}^\sharp(\emptyset)$.) This gives the JSL hypothesis shown in Figure 6.1b, which leads to an NFA hypothesis having a single state that is initial, accepting, and has no transitions (Figure 6.1c). The hypothesis is incorrect, and the teacher may supply us with counterexample aa. Adding the suffixes a and aa to E leads to the table in Figure 6.1d. The table is not closed: $\mathsf{srow}(a)(a) \neq \mathsf{row}^\sharp(\emptyset)$ and $\mathsf{srow}(a)(a) \neq \mathsf{row}^\sharp(\{\varepsilon\})$. Thus, we add a to S. The resulting table (Figure 6.1e) is closed and consistent ($\mathsf{srow}(a)(a) = \mathsf{row}^\sharp(\{\varepsilon, a\})$). We note that row aa is the union of other rows: $\mathsf{row}^\sharp(\{aa\}) = \mathsf{row}^\sharp(\{\varepsilon, a\})$ (i.e., it is not a join-irreducible), and therefore can be ignored when building the succinct hypothesis. This hypothesis has two states, ε and a, and indeed it is the correct one (6.2).

In the next section we formally introduce automata with side-effects given by a monad, which generalise automata such as NFAs and WFAs.

6.2 Automata with Side-Effects

We fix a monad (T, η, μ) with T preserving finite sets, as well as a finite alphabet A and a T-algebra O that models outputs of automata. This setting allows us to define a general notion of automaton with algebraic structure in the form of an algebra for a monad that is preserved by the automaton operations. The assumption that T preserves finite sets guarantees that the algebraic structure on a finite automaton can be represented.

Definition 6.2.1 (T-automaton)**.** A *T-automaton* is a quadruple (Q, δ, i, o), where Q is a T-algebra, the *transition map* $\delta : Q \to Q^A$ and *output map* $o : Q \to O$ are T-algebra homomorphisms, and $i \in Q$ is the *initial state*.[1]

One can recover T-automata as automata according to Definition 2.2.10 by choosing the base category $\mathsf{EM}(T)^{\mathbf{op}}$, the initial state object O, the output object $T(1)$, and the transition functor $(-)^A$.

Example 6.2.2 (DFAs as T-automata)**.** DFAs are Id-automata when $O = 2 = \{0, 1\}$ is used to distinguish accepting from rejecting states. For the more general case of O being an arbitrary set, DFAs generalise into automata called *Moore automata*.

Example 6.2.3 (NFAs as T-automata)**.** Recall that $\mathcal{P}$-algebras are JSLs, and their homomorphisms are join-preserving functions. In a $\mathcal{P}$-automaton, Q is equipped with a join operation,

[1] Our notion of T-automaton generalises the T-automata in for instance [Jac06], where they are restricted to have a free state space and represented as in Proposition 6.2.4 below. (However, the T-automata in [Jac06] do allow different dynamics functors.)

and Q^A is a join-semilattice with pointwise join: $(f \vee g)(a) = f(a) \vee g(a)$ for $a \in A$. Since the automaton maps preserve joins, we have, in particular, $\delta(q_1 \vee q_2)(a) = \delta(q_1)(a) \vee \delta(q_2)(a)$. One can represent an NFA over a set of states S as a $\mathcal{P}$-automaton by taking $Q = (\mathcal{P}(S), \bigcup)$ and $O = 2$, the Boolean join-semilattice with the *or* operation as its join. Let $i \subseteq S$ be the set of initial states and $o : \mathcal{P}(Q) \longrightarrow 2$ and $\delta : \mathcal{P}(S) \longrightarrow \mathcal{P}(S)^A$ the respective extensions of the NFA's output ($S \longrightarrow O$) and transition functions ($S \longrightarrow \mathcal{P}(S)^A$). The resulting $\mathcal{P}$-automaton is precisely the determinised version of the NFA.

More generally, an automaton with side-effects given by a monad T always represents a T-automaton with a free state space.

Proposition 6.2.4 (Succinct automata)**.** *A T-automaton of the form $((TX, \mu_X), \delta, i, o)$, for any set X, is completely defined by the set X with the element $i \in TX$ and functions*

$$\delta^\dagger : X \longrightarrow (TX)^A \qquad\qquad o^\dagger : X \longrightarrow O.$$

Proof. Since $(-)^\sharp$ has an inverse $(-)^\dagger$, the T-algebra homomorphisms δ and o can be represented by the functions $\delta^\dagger$ and $o^\dagger$. □

We call such a T-automaton a *succinct* automaton, which we sometimes identify with the representation $(X, \delta^\dagger, i, o^\dagger)$. These automata are closely related to the ones studied in [GMS14].

A *language* is a function $\mathcal{L} : A^\star \longrightarrow O$. For every T-automaton we have an *observability* and a *reachability* map, telling respectively which state is reached by reading a given word and which language each state recognises.

Definition 6.2.5 (Reachability/observability maps)**.** The *reachability map* of a T-automaton $\mathcal{A}$ is a function $\mathsf{reach}_\mathcal{A} : A^\star \longrightarrow Q$ inductively defined as: $\mathsf{reach}_\mathcal{A}(\varepsilon) = i$ and $\mathsf{reach}_\mathcal{A}(ua) = \delta(\mathsf{reach}_\mathcal{A}(u))(a)$. The *observability map* of $\mathcal{A}$ is a function $\mathsf{obs}_\mathcal{A} : Q \longrightarrow O^{A^\star}$ given by: $\mathsf{obs}_\mathcal{A}(q)(\varepsilon) = o(q)$ and $\mathsf{obs}_\mathcal{A}(q)(av) = \mathsf{obs}_\mathcal{A}(\delta(q)(a))(v)$.

The *language accepted by* $\mathcal{A}$ is the map $\mathcal{L}_\mathcal{A} = \mathsf{obs}_\mathcal{A}(i) = o_\mathcal{A} \circ \mathsf{reach}_\mathcal{A} : A^\star \longrightarrow O$.

Example 6.2.6. For an NFA $\mathcal{A}$ represented as a $\mathcal{P}$-automaton, as seen in Example 6.2.3, $\mathsf{obs}_\mathcal{A}(q)$ is the language of q in the traditional sense. Note that q, in general, is a set of states: $\mathsf{obs}_\mathcal{A}(q)$ takes the union of languages of singleton states. The set $\mathcal{L}_\mathcal{A}$ is the language accepted by the initial states, i.e., the language of the NFA. The reachability map $\mathsf{reach}_\mathcal{A}(u)$ returns the set of states reached via all paths reading u.

Given a language $\mathcal{L} : A^\star \longrightarrow O$, there exists a (unique) *minimal T-automaton* $\mathsf{M}_\mathcal{L}$ accepting $\mathcal{L}$. Its existence follows from the general fact that the factorisation system (surjective functions, injective functions) lifts to the category of T-automata (see for example [Hee16]). Below we give a concrete definition that follows from this abstract perspective.

Definition 6.2.7 (Minimal T-automaton for $\mathcal{L}$)**.** Let $t_\mathcal{L} : A^\star \longrightarrow O^{A^\star}$ be the function giving the *residual languages* of $\mathcal{L}$, namely $t_\mathcal{L}(u) = \lambda v.\mathcal{L}(uv)$. The minimal T-automaton $\mathsf{M}_\mathcal{L}$ accepting $\mathcal{L}$ has state space $M = \mathsf{img}(t^\sharp_\mathcal{L})$, initial state $i = t_\mathcal{L}(\varepsilon)$, and T-algebra homomorphisms $o : M \longrightarrow O$ and $\delta : M \longrightarrow M^A$ given by $o(t^\sharp_\mathcal{L}(U)) = \mathcal{L}(U)$ and $\delta(t^\sharp_\mathcal{L}(U))(a)(v) = t^\sharp_\mathcal{L}(U)(av)$.

In the following, we will also make use of the *minimal Moore automaton* accepting $\mathcal{L}$: the minimal deterministic automaton without algebraic structure. Although this always exists—by instantiating Definition 6.2.7 with $T = \mathsf{Id}$—it need not be finite. The following property says that finiteness of Moore automata and of T-automata accepting the same language are related.

Proposition 6.2.8. *The minimal Moore automaton accepting $\mathcal{L}$ is finite if and only if the minimal T-automaton accepting $\mathcal{L}$ is finite.*

Proof. The left to right implication is proved by freely generating a T-automaton from the Moore one via the monad unit, and by recalling that T preserves finite sets. The resulting T-automaton accepts $\mathcal{L}$ and is finite, therefore any of its quotients, including the minimal T-automaton accepting $\mathcal{L}$, is finite. The right to left implication follows by forgetting the algebraic structure of the T-automaton: this yields a finite Moore automaton accepting $\mathcal{L}$. □

6.3 A General Algorithm

In this section we introduce our extension of $\mathtt{L}^\star$ to learn automata with side-effects. The algorithm is parametric in the notion of side-effect, represented as the monad T, and is therefore called $\mathtt{L}^\star_T$. We fix a language $\mathcal{L} : A^\star \longrightarrow O$ that is to be learned, and we assume that there is a finite T-automaton accepting $\mathcal{L}$. This assumption generalises the requirement of $\mathtt{L}^\star$ that $\mathcal{L}$ is regular (i.e., accepted by a specific class of T-automata, see Example 6.2.2).

An observation table is identified by a pair (S, E) of finite sets $S, E \subseteq A^\star$ such that $\varepsilon \in S \cap E$. The actual table is a representation of the following functions associated with the sets S and E:

$$\mathsf{row} : S \longrightarrow O^E \qquad\qquad \mathsf{srow} : S \longrightarrow (O^E)^A$$

They are given by $\mathsf{row}(s)(e) = \mathcal{L}(se)$ and $\mathsf{srow}(s)(a)(e) = \mathcal{L}(sae)$. For $O = 2$, we recover exactly the $\mathrm{L}^\star$ observation table. The key idea for $\mathrm{L}^\star_T$ is defining closedness and consistency over the free T-extensions of those functions.

Definition 6.3.1 (Closedness and Consistency)**.** The table is *closed* if for all $U \in T(S)$ and $a \in A$ there exists a $U' \in T(S)$ such that $\mathsf{row}^\sharp(U') = \mathsf{srow}^\sharp(U)(a)$. The table is *consistent* if for all $U_1, U_2 \in T(S)$ such that $\mathsf{row}^\sharp(U_1) = \mathsf{row}^\sharp(U_2)$ we have $\mathsf{srow}^\sharp(U_1) = \mathsf{srow}^\sharp(U_2)$.

For closedness, we do not need to check the entire image of $\mathsf{srow}^\sharp$ against the image of $\mathsf{row}^\sharp$, but only the image of srow, thanks to the following result.

Lemma 6.3.2. *If for all $s \in S$ and $a \in A$ there is $U \in T(S)$ such that $\mathsf{row}^\sharp(U) = \mathsf{srow}(s)(a)$, then the table is closed.*

Proof. Let $m : \mathsf{img}(\mathsf{row}^\sharp) \hookrightarrow O^E$ be the embedding of the image of $\mathsf{row}^\sharp$ into its codomain. The definition of closedness given in Definition 6.3.1 amounts to requiring the existence of a function $c : T(S) \longrightarrow \mathsf{img}(\mathsf{row}^\sharp)^A$ making the following diagram commute:

$$\begin{array}{ccc} T(S) & & \\ {\scriptstyle c}\downarrow & \searrow^{\mathsf{srow}^\sharp} & \\ \mathsf{img}(\mathsf{row}^\sharp)^A & \xrightarrow[m^A]{} & (O^E)^A \end{array} \tag{6.3}$$

It is easy to see that the condition of this lemma corresponds to requiring the existence of a function $c_0 : S \longrightarrow \mathsf{img}(\mathsf{row}^\sharp)^A$ making the diagram below on the left in **Set** commute.

$$\begin{array}{ccc} S & & \\ {\scriptstyle c_0}\downarrow & \searrow^{\mathsf{srow}} & \\ \mathsf{img}(\mathsf{row}^\sharp)^A & \xrightarrow[m^A]{} & (O^E)^A \end{array} \qquad \begin{array}{ccc} T(S) & & \\ {\scriptstyle T(c_0)}\downarrow & \searrow^{T(\mathsf{srow})} & \\ T(\mathsf{img}(\mathsf{row}^\sharp)^A) & \xrightarrow[T(m^A)]{} & T((O^E)^A) \\ \downarrow & & \downarrow \\ \mathsf{img}(\mathsf{row}^\sharp)^A & \xrightarrow[m^A]{} & (O^E)^A \end{array}$$

This diagram can be made into a diagram of T-algebra homomorphisms as on the right, where the compositions of the left and right legs give respectively $c_0^\sharp$ and $\mathsf{srow}^\sharp$. This diagram commutes because the top triangle commutes by functoriality of T, and the bottom square commutes by m^A being a T-algebra homomorphism. Therefore we have that (6.3) commutes for $c = c_0^\sharp$. □

Example 6.3.3. For NFAs represented as $\mathcal{P}$-automata, the properties are as presented in Section 6.1. Recall that for $T = \mathcal{P}$ and $O = 2$, the Boolean join-semilattice, $\mathsf{row}^\sharp$ and

$\mathsf{srow}^\sharp$ describe a table where rows are labelled by subsets of S. Then we have, for instance, $\mathsf{row}^\sharp(\{s_1, s_2\})(e) = \mathsf{row}(s_1)(e) \vee \mathsf{row}(s_2)(e)$, i.e., $\mathsf{row}^\sharp(\{s_1, s_2\})(e) = 1$ if and only if $\mathcal{L}(s_1 e) = 1$ or $\mathcal{L}(s_2 e) = 1$. Closedness amounts to check whether each row in the bottom part of the table is the join of a set of rows in the top part. Consistency amounts to check whether, for all sets of rows $U_1, U_2 \subseteq S$ in the top part of the table whose joins are equal, the joins of rows $U_1 \cdot \{a\}$ and $U_2 \cdot \{a\}$ in the bottom part are also equal, for all $a \in A$.

As in the original $\mathrm{L}^\star$ algorithm, closedness and consistency allow us to define a hypothesis.

Definition 6.3.4 (Hypothesis T-automaton)**.** Given a closed and consistent table (S, E), we can define the *hypothesis* T-automaton $\mathcal{H}$, with state space $H = \mathsf{img}(\mathsf{row}^\sharp)$, $i = \mathsf{row}(\varepsilon)$, and output and transitions

$$o : H \longrightarrow O \qquad o(\mathsf{row}^\sharp(U)) = \mathsf{row}^\sharp(U)(\varepsilon)$$
$$\delta : H \longrightarrow H^A \qquad \delta(\mathsf{row}^\sharp(U)) = \mathsf{srow}^\sharp(U).$$

Well-definedness of the above T-automaton follows from the abstract treatment of Chapter 4, instantiated to the category of T-algebras and their homomorphisms.

We can now give the algorithm $\mathrm{L}^\star_T$. Similarly to the example in Section 6.1, we only have to adjust lines 4 and 7 of Algorithm 2.1 in Figure 2.1. The resulting algorithm is shown in Figure 6.2. As in Figure 2.1, the hypothesis for a table (S, E) is denoted by $\mathcal{H}_{(S,E)}$. In this case it is constructed via Definition 6.3.4.

The pseudocode in Figure 6.2 hides a few computability assumptions. In lines 3 and 4 of Algorithm 6.1 we need to be able to decide whether closedness holds, and if not find a witness that this is not the case. Given that A is finite and T preserves finite sets this can be done by enumerating all bottom rows and checking for each of them whether there is an element of TS for which the corresponding row is equal to that bottom row. This requires enumerability of the (finite) set TS and decidability of $\mathsf{row}^\sharp$. Similarly, lines 6 and 7 require decidability of consistency, with a witness being obtained if it does not hold. This could be achieved by enumerating all elements of $TS \times TS$, checking which ones lead to the same row, and for those ones checking for each element of A whether the corresponding bottom rows match. Again, this requires enumerability of TS and decidability of $\mathsf{row}^\sharp$, as well as decidability of $\mathsf{srow}^\sharp$. These same requirements also make sure the hypothesis of Definition 6.3.4 is computable.

6.3.1 Correctness

Correctness of $\mathrm{L}^\star_T$ amounts to proving that, for any target language $\mathcal{L}$, the algorithm terminates returning the minimal T-automaton $\mathsf{M}_\mathcal{L}$ accepting $\mathcal{L}$. As in the original $\mathrm{L}^\star$ algorithm,

Algorithm 6.1. Make table closed and consistent

1: **function** Fix(S, E)
2: **while** (S, E) is not closed or not consistent **do**
3: **if** (S, E) is not closed **then**
4: find $s \in S$,$a \in A$ such that $\forall U \in T(S)$. $\mathsf{srow}(s)(a) \neq \mathsf{row}(U)$
5: $S \longleftarrow S \cup \{sa\}$
6: **else if** (S, E) is not consistent **then**
7: find $U_1, U_2 \in T(S)$, $a \in A$ and $e \in E$ such that
$\mathsf{row}^\sharp(U_1) = \mathsf{row}^\sharp(U_2)$ and $\mathsf{srow}^\sharp(U_1)(a)(e) \neq \mathsf{srow}^\sharp(U_2)(a)(e)$
8: $E \longleftarrow E \cup \{ae\}$
9: **return** S, E

Algorithm 6.2. $\mathrm{L}^\star_T$ algorithm

1: $S, E \longleftarrow$ Fix$(\{\varepsilon\}, \{\varepsilon\})$
2: **while** $\mathsf{EQ}(\mathcal{H}_{(S,E)}) = c \in A^\star$ **do**
3: $E \longleftarrow E \cup \mathsf{suffixes}(c)$
4: $S, E \longleftarrow$ Fix(S, E)
5: **return** $\mathcal{H}_{(S,E)}$

Figure 6.2: Adaptation of $\mathrm{L}^\star$ for T-automata.

when the algorithm terminates the final hypothesis by definition accepts $\mathcal{L}$—the algorithm only terminates once an equivalence query yields a positive outcome.

Our proof of termination and minimality is based on a bound on the size of the state space H of the hypothesis and showing that H increases with each operation performed in the algorithm. For the processing of counterexamples, we need the following lemma showing that adding the suffixes of a counterexample to E (line 3) will either distinguish two rows or cause a closedness defect. We defer the proof of Lemma 6.3.5 below to the stronger Proposition 6.5.3 in Section 6.5.

Lemma 6.3.5. *If $z \in A^\star$ is such that $\mathcal{L}_\mathcal{H}(z) \neq \mathcal{L}(z)$, then after adding* $\mathsf{suffixes}(z)$ *to E we have that either two rows become distinguished (there exist $U_1, U_2 \in T(S)$ such that* $\mathsf{row}^\sharp(U_1) \neq \mathsf{row}^\sharp(U_2)$*, whereas they were previously equal) or the updated table is not closed.*

We now give the full proof.

Theorem 6.3.6 (Correctness of $\mathtt{L}^\star_T$)**.** *The $L^\star_T$ algorithm (Figure 6.2) terminates returning the minimal T-automaton accepting $\mathcal{L}$.*

Proof. We argue that the state space H of the hypothesis increases while the algorithm loops, and that H cannot be larger than M, the state space of $\mathsf{M}_{\mathcal{L}}$. When a closedness defect is resolved (line 5), a row that was not previously found in the image of $\mathsf{row}^\sharp : T(S) \longrightarrow O^E$ is added, so the set H grows larger. When a consistency defect is resolved (line 8), two previously equal rows become distinguished, which also increases the size of H. As for counterexamples, Lemma 6.3.5 shows that adding their suffixes to E (line 3) will either distinguish two rows or cause a closedness defect, which will be fixed during the next iteration, causing H to increase.

Now, note that by increasing S or E, the hypothesis state space H never decreases in size. Moreover, for $S = A^\star$ and $E = A^\star$, $\mathsf{row}^\sharp = t^\sharp_{\mathcal{L}}$. Therefore, since H and M are defined as the images of $\mathsf{row}^\sharp$ and $t^\sharp_{\mathcal{L}}$, respectively, the size of H is bounded by that of M. As H increases while the algorithm loops, the algorithm must terminate and thus correctly finds the minimal automaton accepting $\mathcal{L}$. □

Note that the learning algorithm of Bollig et al. does not terminate using this counterexample processing method [Bol+08, Appendix F]. This is due to their notion of consistency being weaker than ours: we have shown that progress is guaranteed because a consistency defect, in our sense, is created using this method.

Query complexity. The complexity of automata learning algorithms is usually measured in terms of the number of both membership and equivalence queries asked, as it is common to assume that computations within the algorithm are insignificant compared to evaluating the system under analysis in applications. The cost of answering queries themselves is not considered, as it depends on the implementation of the teacher, which the algorithm abstracts from.

The table is a T-algebra homomorphism, so membership queries for rows labelled in S are enough to determine all other rows. We measure the query complexities in terms of the number of states n of the minimal Moore automaton, the number of states t of the minimal T-automaton, the size k of the alphabet, and the length m of the longest counterexample. Note that t cannot be smaller than n.

Remark 6.3.7. The number t can be much bigger than n. For example, when $T = \mathcal{P}$, t may be in $\mathcal{O}(2^n)$. Take the language $\{a^p\}$, for some $p \in \mathbb{N}$ and a singleton alphabet $\{a\}$. Its residual

languages are $\emptyset$ and $\{a^i\}$ for all $0 \leq i \leq p$. Thus, the minimal DFA accepting the language has $p + 2$ states. However, the residual languages w.r.t. sets of words are all the subsets of $\{\varepsilon, a, aa, \dots, a^p\}$—hence, the minimal T-automaton has 2^{p+1} states.

The maximum number of closedness defects fixed by the algorithm is n, as a closedness defect for the setting with algebraic structure is also a closedness defect for the setting without that structure. The maximum number of consistency defects fixed by the algorithm is t, as fixing a consistency defect distinguishes two rows that were previously identified. Since counterexamples lead to consistency defects, this also means that the algorithm will not pose more than t equivalence queries. A word is added to S when fixing a closedness defect, and $\mathcal{O}(m)$ words are added to S when processing a counterexample. The number of rows that we need to fill using queries is therefore in $\mathcal{O}(tmk)$. The number of columns added to the table is given by the number of times a consistency defect is fixed and thus in $\mathcal{O}(t)$. Altogether, the number of membership queries is in $\mathcal{O}(t^2mk)$.

6.4 Succinct Hypotheses

We now describe the first of two optimisations, which is enabled by the use of monads. Our algorithm produces hypotheses that can be quite large, as their state space is the image of $\mathsf{row}^\sharp$, which has the whole set $T(S)$ as its domain. For instance, when $T = \mathcal{P}$, $T(S)$ is exponentially larger than S. We will show how we can represent *succinct* hypotheses, whose state space is given by a subset of S, and how we can compute a suitable subset of S that has a minimality property and still induces an equivalent succinct hypothesis. We start by defining sets of *generators for the table*.

Definition 6.4.1. A set $S' \subseteq S$ is a *set of generators for the table* whenever for all $s \in S$ there is $U \in T(S')$ such that $\mathsf{row}(s) = \mathsf{row}^\sharp(U)$.[2]

Intuitively, given $s \in S$ and $U \in T(S')$ as in the above definition, U is the decomposition of s into a "combination" of generators. When $T = \mathcal{P}$, S' generates the table whenever each row can be obtained as the join of a set of rows labelled by S'. Explicitly: for all $s \in S$ there is $\{s_1, \dots, s_n\} \subseteq S'$ such that $\mathsf{row}(s) = \mathsf{row}^\sharp(\{s_1, \dots, s_n\}) = \mathsf{row}(s_1) \vee \cdots \vee \mathsf{row}(s_n)$.

Recall that $\mathcal{H}$, with state space H, is the hypothesis automaton for the table. The existence of generators S' allows us to compute a T-automaton with state space $T(S')$ equivalent to $\mathcal{H}$. We call this the *succinct hypothesis*, although $T(S')$ may be larger than H. Proposition 6.2.4

[2]Here and hereafter we assume that $T(S') \subseteq T(S)$, and more generally that T preserves inclusion maps. To eliminate this assumption, one could take the inclusion map $f : S' \hookrightarrow S$ and write $\mathsf{row}^\sharp(T(f)(U))$ instead of $\mathsf{row}^\sharp(U)$.

tells us that the succinct hypothesis can be represented as an automaton with side-effects in T that has S' as its state space. This results in a lower space complexity when storing the hypothesis.

We now show how the succinct hypothesis is computed. Observe that, if generators S' exist, $\mathsf{row}^\sharp$ factors through the restriction of itself to $T(S')$. Denote this latter function $\widehat{\mathsf{row}}^\sharp$. Since we have $T(S') \subseteq T(S)$, the image of $\widehat{\mathsf{row}}^\sharp$ coincides with $\mathsf{img}(\mathsf{row}^\sharp) = H$, and therefore the surjection restricting $\widehat{\mathsf{row}}^\sharp$ to its image has the form $e : T(S') \longrightarrow H$. Any right inverse $\mathsf{dec} : H \longrightarrow T(S')$ of the function e (that is, $e \circ \mathsf{dec} = \mathsf{id}_H$, but whereas e is a T-algebra homomorphism, dec need not be one) yields a succinct hypothesis as follows. We refer to such a function dec as a *decomposition function*.

Definition 6.4.2 (Succinct Hypothesis)**.** Given a table (S, E) and a set of generators $S' \subseteq S$ with decomposition function $\mathsf{dec} : H \longrightarrow T(S')$, the *succinct hypothesis* is the T-automaton $\mathcal{S} = (T(S'), \delta, i, o)$ given by $i = \mathsf{dec}(\mathsf{row}(\varepsilon))$ and

$$\begin{aligned} o^\dagger &: S' \longrightarrow O & o^\dagger(s) &= \mathsf{row}(s)(\varepsilon) \\ \delta^\dagger &: S' \longrightarrow T(S')^A & \delta^\dagger(s)(a) &= \mathsf{dec}(\mathsf{srow}(s)(a)). \end{aligned}$$

This definition is inspired by that of a *scoop*, due to Arbib and Manes [AM75b]. Below we prove that any succinct hypothesis accepts the same language as the actual hypothesis. This ensures that we can replace the hypothesis constructed in line 2 of Algorithm 6.2 with a succinct one without invalidating correctness of the algorithm.

Proposition 6.4.3. *Any succinct hypothesis of $\mathcal{H}$ accepts the language of $\mathcal{H}$.*

Proof. Assume a right inverse $\mathsf{dec} : H \longrightarrow T(S')$ of $e : T(S') \longrightarrow H$. We first prove $\mathsf{obs}_\mathcal{H} \circ e^\dagger = \mathsf{obs}^\dagger_\mathcal{S}$, by induction on the length of words. For all $s \in S'$, we have

$$\begin{aligned} \mathsf{obs}_\mathcal{H}(e^\dagger(s))(\varepsilon) &= o_\mathcal{H}(e^\dagger(s)) && \text{(definition of } \mathsf{obs}_\mathcal{H}) \\ &= o_\mathcal{H}(\mathsf{row}(s)) && \text{(definition of } e) \\ &= \mathsf{row}(s)(\varepsilon) && \text{(definition of } o_\mathcal{H}) \\ &= o^\dagger_\mathcal{S}(s) && \text{(definition of } o_\mathcal{S}) \\ &= \mathsf{obs}^\dagger_\mathcal{S}(s)(\varepsilon) && \text{(definition of } \mathsf{obs}_\mathcal{S}). \end{aligned}$$

Now assume that for a given $v \in A^\star$ and all $s \in S'$ we have $\mathsf{obs}_\mathcal{H}(e^\dagger(s))(v) = \mathsf{obs}^\dagger_\mathcal{S}(s)(v)$. Then,

for all $s \in S'$ and $a \in A$,

$$\begin{aligned}
\mathsf{obs}_{\mathcal{H}}(e^{\dagger}(s))(av) &= \mathsf{obs}_{\mathcal{H}}(\delta_{\mathcal{H}}(e^{\dagger}(s))(a))(v) && \text{(definition of } \mathsf{obs}_{\mathcal{H}}) \\
&= \mathsf{obs}_{\mathcal{H}}(\delta_{\mathcal{H}}(\mathsf{row}(s))(a))(v) && \text{(definition of } e) \\
&= \mathsf{obs}_{\mathcal{H}}(\mathsf{srow}(s)(a))(v) && \text{(definition of } \delta_{\mathcal{H}}) \\
&= (\mathsf{obs}_{\mathcal{H}} \circ e \circ \mathsf{dec})(\mathsf{srow}(s)(a))(v) && (e \circ \mathsf{dec} = \mathsf{id}_H) \\
&= (\mathsf{obs}_{\mathcal{S}} \circ \mathsf{dec})(\mathsf{srow}(s)(a))(v) && \text{(induction hypothesis)} \\
&= \mathsf{obs}_{\mathcal{S}}(\delta_{\mathcal{S}}^{\dagger}(s)(a))(v) && \text{(definition of } \delta_{\mathcal{S}}) \\
&= \mathsf{obs}_{\mathcal{S}}^{\dagger}(s)(av) && \text{(definition of } \mathsf{obs}_{\mathcal{S}}).
\end{aligned}$$

This concludes the proof of $\mathsf{obs}_{\mathcal{H}} \circ e^{\dagger} = \mathsf{obs}_{\mathcal{S}}^{\dagger}$. Then

$$\mathsf{obs}_{\mathcal{H}} \circ e = (\mathsf{obs}_{\mathcal{H}} \circ e)^{\dagger^{\#}} = (\mathsf{obs}_{\mathcal{H}} \circ e^{\dagger})^{\#} = \mathsf{obs}_{\mathcal{S}}^{\dagger^{\#}} = \mathsf{obs}_{\mathcal{S}},$$

so

$$\begin{aligned}
\mathsf{obs}_{\mathcal{S}}(i_{\mathcal{S}}) &= (\mathsf{obs}_{\mathcal{S}} \circ \mathsf{dec})(\mathsf{row}(\varepsilon)) && \text{(definition of } i_{\mathcal{S}}) \\
&= (\mathsf{obs}_{\mathcal{H}} \circ e \circ \mathsf{dec})(\mathsf{row}(\varepsilon)) && (\mathsf{obs}_{\mathcal{H}} \circ e = \mathsf{obs}_{\mathcal{S}}) \\
&= \mathsf{obs}_{\mathcal{H}}(\mathsf{row}(\varepsilon)) && (e \circ \mathsf{dec} = \mathsf{id}_H) \\
&= \mathsf{obs}_{\mathcal{H}}(i_{\mathcal{H}}) && \text{(definition of } i_{\mathcal{H}}). \qquad \square
\end{aligned}$$

We now give a simple procedure to compute a *minimal* set of generators, that is, a set S' such that no proper subset is a set of generators. This generalises a procedure defined by Angluin et al. [AEF15] for non-deterministic, universal, and alternating automata.

Proposition 6.4.4. *The following algorithm returns a minimal set of generators for the table:*

$S' \leftarrow S$
while *there are* $s \in S'$ *and* $U \in T(S' \setminus \{s\})$ *s.t.* $\mathsf{row}^{\#}(U) = \mathsf{row}(s)$ ***do***
  $S' \leftarrow S' \setminus \{s\}$
return S'

Proof. Minimality is obvious, as S' not being minimal would keep the loop guard true.

We prove that S' is a set of generators throughout a run of the algorithm. For clarity, we denote by $d_{S'} : S \longrightarrow T(S')$ the function associated with a set of generators S'. The main idea is incrementally building $d_{S'}$ while building S'. In the first line, S is a set of generators, with $d_S = \eta_S : S \longrightarrow T(S)$. For the loop, suppose S' is a set of generators. If the loop guard is false,

the algorithm returns the set of generators S'. Otherwise, suppose there are there are $s \in S'$ and $U \in T(S' \setminus \{s\})$ such that $\mathsf{row}^\sharp(U) = \mathsf{row}(s)$. Then there is a function

$$f : S' \longrightarrow T(S' \setminus \{s\}) \qquad\qquad f(s') = \begin{cases} U & \text{if } s' = s \\ \eta(s') & \text{if } s' \neq s \end{cases}$$

that satisfies $\mathsf{row}(s') = \mathsf{row}^\sharp(f(s'))$ for all $s' \in S'$, from which it follows that $\mathsf{row}^\sharp(U') = \mathsf{row}^\sharp(f^\sharp(U'))$ for all $U' \in T(S')$. Then we can set $d_{S'\setminus\{s\}}$ to $f^\sharp \circ d_{S'} : S \longrightarrow T(S' \setminus \{s\})$ because $\mathsf{row}(s') = \mathsf{row}^\sharp(d_{S'\setminus\{s\}}(s'))$ for all $s' \in S$. Therefore, $S' \setminus \{s\}$ is a set of generators. □

To determine whether U as in the above algorithm exists, one can always naively enumerate all possibilities, using that T preserves finite sets. This is what we call the basic algorithm. For specific algebraic structures, one may find more efficient methods, as we show in the following example.

Example 6.4.5 (RFSAs)**.** Consider the powerset monad $T = \mathcal{P}$. We now exemplify two ways of computing succinct hypotheses, which are inspired by canonical RFSAs [DLT02]. The basic idea is to start from a deterministic automaton and to remove states that are equivalent to a set of other states. The algorithm given in Proposition 6.4.4 computes a minimal S' that only contains labels of rows that are not the join of other rows. (In case two rows are equal, only one of their labels is kept.) In other words, as mentioned in Section 6.1, S' contains labels of join-irreducible rows. To concretize the algorithm efficiently, we use a method introduced by Bollig et al. [Bol+09], which essentially exploits the natural order $a \leq b \iff a \vee b = b$ on the JSL of table rows. In contrast to the basic exponential algorithm, this results in a polynomial one.[3] Bollig et al. determine whether a row is a join of other rows by comparing the row just to the join of rows below it. Like them, we make use of this also to compute right inverses of e, for which we will formalise the order.

The function $e : \mathcal{P}(S') \longrightarrow H$ tells us which sets of rows are equivalent to a single state in H. We show two right inverses $H \longrightarrow \mathcal{P}(S')$ for it. The first one,

$$\mathsf{dec}_1(h) = \{s \in S' \mid \mathsf{row}(s) \leq h\},$$

stems from the construction of the *canonical RFSA* of a language [DLT02]. The resulting natural construction of a succinct hypothesis was first used by Bollig et al. [Bol+09]. This succinct hypothesis has a "maximal" transition function, meaning that no more transitions can be added without changing the language of the automaton.

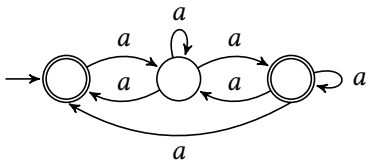
[3] When we refer to computational complexities, as opposed to query complexities, they are in terms of the sizes of S, E, and A.

The second inverse is, for $h \in H$,

$$\mathsf{dec}_2(h) = \{s \in S' \mid \mathsf{row}(s) \leq h \wedge \forall s' \in S'.\, \mathsf{row}(s) \leq \mathsf{row}(s') \leq h \implies \mathsf{row}(s) = \mathsf{row}(s')\},$$

resulting in a more economical transition function, where some redundancies are removed. This corresponds to the *simplified canonical RFSA* [DLT02].

Example 6.4.6. Consider $T = \mathcal{P}$, and recall the table in Figure 6.1e. When $S' = S$, the right inverse given by dec_1 yields the succinct hypothesis shown below.



Note that $\mathsf{dec}_1(\mathsf{row}(aa)) = \{\varepsilon, a, aa\}$. Taking dec_2 instead, the succinct hypothesis is just the DFA (6.1) because $\mathsf{dec}_2(\mathsf{row}(aa)) = \{aa\}$. Rather than constructing a succinct hypothesis directly, our algorithm first reduces the set S'. In this case, we have $\mathsf{row}(aa) = \mathsf{row}^\sharp(\{\varepsilon, a\})$, so we remove aa from S'. Now dec_1 and dec_2 coincide and produce the NFA (6.2). Minimising the set S' in this setting essentially comes down to determining what Bollig et al. [Bol+09] call the *prime* rows of the table.

Remark 6.4.7. The algorithm in Proposition 6.4.4 implicitly assumes an order in which elements of S are checked. Although the algorithm is correct for any such order, different orders may give results that differ in size. Consider for instance the free semimodule monad V for the semiring $\mathbb{Z}_6$ of integers modulo 6, the free output object $\mathbb{Z}_6 \cong V(1)$, and the language $\mathcal{L} : \{a\}^\star \to \mathbb{Z}_6$ given by $\mathcal{L}(a^n) = (n+1) \bmod 6$. The observation table for $S = \{\varepsilon, a, aa\}$ and $E = \{\varepsilon\}$ is given below.

	ε
ε	1
a	2
aa	3
aaa	4

We initialise $S' = S$. Note that $\mathsf{row}(\varepsilon) = \mathsf{row}^\sharp(2 \times a + aa)$, so we can remove ε from S'. After doing so, however, neither a nor aa can be removed from S'. If, instead, from $S' = S$ we observe that $\mathsf{row}(a) = \mathsf{row}^\sharp(2 \times \varepsilon)$ and $\mathsf{row}(aa) = \mathsf{row}^\sharp(3 \times \varepsilon)$, we may end up with the singleton $S' = \{\varepsilon\}$.

6.5 Optimised Counterexample Handling

The second optimisation we give generalises the counterexample processing method due to Rivest and Schapire [RS93], which improves the worst case complexity of the number of membership queries needed in $\mathtt{L}^\star$. Maler and Pnueli [MP95] proposed to add all suffixes of the counterexample to the set E instead of adding all prefixes to the set S. This eliminates the need for consistency checks in the deterministic setting. The method by Rivest and Schapire finds a *single* suffix of the counterexample and adds it to E. This suffix is chosen in such a way that it either distinguishes two existing rows or creates a closedness defect, both of which imply that the hypothesis automaton will grow.

The main idea is finding the distinguishing suffix via the hypothesis automaton $\mathcal{H}$. Given $u \in A^\star$, let q_u be the state in $\mathcal{H}$ reached by reading u, i.e., $q_u = \mathsf{reach}_{\mathcal{H}}(u)$. For each $q \in H$, we pick any $U_q \in T(S)$ that yields q according to the table, i.e., such that $\mathsf{row}^\sharp(U_q) = q$. Then for a counterexample z we have that the residual language w.r.t. U_{q_z} does not "agree" with the residual language w.r.t. z.

The above intuition can be formalised as follows. Let $\mathcal{R} : A^\star \longrightarrow O^{A^\star}$ be given for all $u \in A^\star$ by $\mathcal{R}(u) = t^\sharp_{\mathcal{L}}(U_{q_u})$. Given $u \in A^\star$, $\mathcal{R}(u)$ computes the residual language of $U_{q_u} \in T(S)$. If the hypothesis is correct, the residual languages $\mathcal{R}(u)$ and $t_{\mathcal{L}}(u)$ should be equal for all words $u \in A^\star$. We will show that $\mathcal{R}$ can be used to analyse a counterexample in order to extract a distinguishing suffix from it. Towards this result we have the following technical lemma, saying that a counterexample z distinguishes the residual languages $t_{\mathcal{L}}(z)$ and $\mathcal{R}(z)$.

Lemma 6.5.1. *If $z \in A^\star$ is such that $\mathcal{L}_{\mathcal{H}}(z) \neq \mathcal{L}(z)$, then $t_{\mathcal{L}}(z)(\varepsilon) \neq \mathcal{R}(z)(\varepsilon)$.*

Proof. We have

$$
\begin{aligned}
t_{\mathcal{L}}(z)(\varepsilon) &= \mathcal{L}(z) && \text{(definition of } t_{\mathcal{L}}\text{)}\\
&\neq \mathcal{L}_{\mathcal{H}}(z) && \text{(assumption)}\\
&= (o_{\mathcal{H}} \circ \mathsf{reach}_{\mathcal{H}})(z) && \text{(definition of } \mathcal{L}_{\mathcal{H}}\text{)}\\
&= \mathsf{reach}_{\mathcal{H}}(z)(\varepsilon) && \text{(definition of } o_{\mathcal{H}}\text{)}\\
&= q_z(\varepsilon) && \text{(definition of } q_z\text{)}\\
&= \mathsf{row}^\sharp(U_{q_z})(\varepsilon) && \text{(definition of } U_{q_z}\text{)}\\
&= t^\sharp_{\mathcal{L}}(U_{q_z})(\varepsilon) && \text{(definitions of row and } t_{\mathcal{L}}\text{)}\\
&= \mathcal{R}(z)(\varepsilon) && \text{(definition of } \mathcal{R}\text{).}
\end{aligned}
$$

□

We assume that $U_{q_\varepsilon} = \eta(\varepsilon)$. For a counterexample z, we then have $\mathcal{R}(\varepsilon)(z) = t_{\mathcal{L}}(\varepsilon)(z) \neq \mathcal{R}(z)(\varepsilon)$. While reading z, the hypothesis automaton passes a sequence of states q_{u_0},

$q_{u_1}, q_{u_2}, \ldots, q_{u_n}$, where $u_0 = \epsilon$, $u_n = z$, and $u_{i+1} = u_i a$ for some $a \in A$ is a prefix of z. If z were correctly classified by $\mathcal{H}$, all residuals $\mathcal{R}(u_i)$ would classify the remaining suffix v of z, i.e., such that $z = u_i v$, in the same way. However, the previous lemma tells us that, for a counterexample z, this is not case, meaning that for some suffix v we have $\mathcal{R}(ua)(v) \neq \mathcal{R}(u)(av)$. In short, this inequality is discovered along a transition in the path to z.

Corollary 6.5.2. *If $z \in A^\star$ is such that $\mathcal{L}_\mathcal{H}(z) \neq \mathcal{L}(z)$, then there are $u, v \in A^\star$ and $a \in A$ such that $uav = z$ and $\mathcal{R}(ua)(v) \neq \mathcal{R}(u)(av)$.*

To find such a decomposition efficiently, Rivest and Schapire use a binary search algorithm. We conclude with the following result that turns the above property into the elimination of a closedness witness. That is, given a counterexample z and the resulting decomposition uav from the above corollary, we show that, while currently $\mathsf{row}^\#(U_{q_{ua}}) = \mathsf{srow}^\#(U_{q_u})(a)$, after adding v to E we have $\mathsf{row}^\#(U_{q_{ua}})(v) \neq \mathsf{srow}^\#(U_{q_u})(a)(v)$. (To see that the latter follows from the proposition below, note that for all $U \in T(S)$ and $e \in E$, $\mathsf{row}^\#(U)(e) = t^\#_\mathcal{L}(U)(e)$ and for each $a' \in A$, $\mathsf{srow}^\#(U)(a')(e) = t^\#_\mathcal{L}(U)(a'e)$.) The inequality means that either we have a closedness defect, or there still exists some $U \in T(S)$ such that $\mathsf{row}^\#(U) = \mathsf{srow}^\#(U_{q_u})(a)$. In this case, the rows $\mathsf{row}^\#(U)$ and $\mathsf{row}^\#(U_{q_{ua}})$ have become distinguished by adding v, which means that the size of H has increased. A closedness defect also leads to an increase in the size of H, so in any case we make progress.

Proposition 6.5.3. *If $z \in A^\star$ is such that $\mathcal{L}_\mathcal{H}(z) \neq \mathcal{L}(z)$, then there are $u, v \in A^\star$ and $a \in A$ such that $\mathsf{row}^\#(U_{q_{ua}}) = \mathsf{srow}^\#(U_{q_u})(a)$ and $t^\#_\mathcal{L}(U_{q_{ua}})(v) \neq t^\#_\mathcal{L}(U_{q_u})(av)$.*

Proof. By Corollary 6.5.2 we have $u, v \in A^\star$ and $a \in A$ such that $\mathcal{R}(ua)(v) \neq \mathcal{R}(u)(av)$. This directly yields the inequality by the definition of $\mathcal{R}$. Furthermore,

$$
\begin{aligned}
\mathsf{row}(U_{q_{ua}}) &= q_{ua} && \text{(definition of } U_{q_{ua}}\text{)} \\
&= \mathsf{reach}_\mathcal{H}(ua) && \text{(definition of } q_{ua}\text{)} \\
&= \delta_\mathcal{H}(\mathsf{reach}_\mathcal{H}(u))(a) && \text{(definition of } \mathsf{reach}_\mathcal{H}\text{)} \\
&= \delta_\mathcal{H}(q_u)(a) && \text{(definition of } q_u\text{)} \\
&= \delta_\mathcal{H}(\mathsf{row}^\#(U_{q_u}))(a) && \text{(definition of } U_{q_u}\text{)} \\
&= \mathsf{srow}^\#(U_{q_u})(a) && \text{(definition of } \delta_\mathcal{H}\text{).}
\end{aligned}
$$

□

Thus, the optimised algorithm is as follows. Compared to Algorithm 6.2, the only line that changes is line 3. Here instead of adding all suffixes of the counterexample to the set E, we find according to Proposition 6.5.3 above words $u, v \in A^\star$ and $a \in A$ such that $\mathsf{row}^\#(U_{q_{ua}}) =$

$\mathsf{srow}^\sharp(U_{q_u})(a)$ and $t^\sharp_{\mathcal{L}}(U_{q_{ua}})(v) \neq t^\sharp_{\mathcal{L}}(U_{q_u})(av)$. We then add v to the set E, which as discussed above guarantees that either causes a closedness defect or distinguishes two previously equal rows in $\mathsf{srow}^\sharp$, thus making progress like the original $\mathtt{L}^\star_T$. This means that the correctness proof Theorem 6.3.6 applies also to the present variation on $\mathtt{L}^\star_T$.

6.5.1 Using the Succinct Hypothesis

We now show how to combine the optimised counterexample processing method with the succinct hypothesis optimisation from Section 6.4. Recall that the succinct hypothesis $\mathcal{S}$ is based on a right inverse $\mathsf{dec} : H \longrightarrow T(S')$ of $e : T(S') \longrightarrow H$. Choosing such a dec is equivalent to choosing U_q for each $q \in H$. We then redefine $\mathcal{R}$ using the reachability map of the succinct hypothesis. Specifically, $\mathcal{R}(u) = t^\sharp_{\mathcal{L}}(\mathsf{reach}_{\mathcal{S}}(u))$ for all $u \in A^\star$.

Unfortunately, there is one complication. We assumed earlier that $U_{q_\varepsilon} = \eta(\varepsilon)$, or more specifically $\mathcal{R}(\varepsilon)(z) = \mathcal{L}(z)$. This now may be impossible because we do not necessarily have $\varepsilon \in S'$. We show next that if this equality does not hold, then there are two rows that we can distinguish by adding z to E. Thus, after testing whether $\mathcal{R}(\varepsilon)(z) = \mathcal{L}(z)$, we either add z to E (if the test fails) or proceed with the original method.

Proposition 6.5.4. *If $z \in A^\star$ is such that $\mathcal{R}(\varepsilon)(z) \neq \mathcal{L}(z)$, then $\mathsf{row}^\sharp(i_{\mathcal{S}}) = \mathsf{row}(\varepsilon)$ and $t^\sharp_{\mathcal{L}}(i_{\mathcal{S}})(z) \neq t_{\mathcal{L}}(\varepsilon)(z)$.*

Proof. We have $\mathsf{row}^\sharp(i_{\mathcal{S}}) = \mathsf{row}^\sharp(\mathsf{dec}(\mathsf{row}(\varepsilon))) = \mathsf{row}(\varepsilon)$ by the definitions of $i_{\mathcal{S}}$ and dec, and

$$\begin{aligned}
t^\sharp_{\mathcal{L}}(\mathsf{dec}(\mathsf{row}(\varepsilon)))(z) &= t^\sharp_{\mathcal{L}}(i_{\mathcal{S}})(z) && \text{(definition of } i_{\mathcal{S}}\text{)}\\
&= t^\sharp_{\mathcal{L}}(\mathsf{reach}_{\mathcal{S}}(\varepsilon))(z) && \text{(definition of } \mathsf{reach}_{\mathcal{S}}\text{)}\\
&= \mathcal{R}(\varepsilon)(z) && \text{(definition of } \mathcal{R}\text{)}\\
&\neq \mathcal{L}(z) && \text{(assumption)}\\
&= t_{\mathcal{L}}(\varepsilon)(z) && \text{(definition of } t_{\mathcal{L}}\text{).}
\end{aligned}$$

□

To see that the original method still works, we prove the analogue of Proposition 6.5.3 for the new definition of $\mathcal{R}$ with Proposition 6.5.7. This first requires an additional lemma.

Lemma 6.5.5. *If $z \in A^\star$ is such that $\mathcal{L}_{\mathcal{S}}(z) \neq \mathcal{L}(z)$ and $\mathcal{R}(\varepsilon)(z) = \mathcal{L}(z)$, then $\mathcal{R}(\varepsilon)(z) \neq \mathcal{R}(z)(\varepsilon)$.*

Proof. We have

$$\begin{aligned}
\mathcal{R}(\varepsilon)(z) &= \mathcal{L}(z) && \text{(assumption)}\\
&\neq \mathcal{L}_{\mathcal{S}}(z) && \text{(counterexample)}\\
&= (o_{\mathcal{S}} \circ \mathsf{reach}^{\dagger}_{\mathcal{S}})(z) && \text{(definition of } \mathcal{L}_{\mathcal{S}})\\
&= (\mathsf{row}^{\#} \circ \mathsf{reach}^{\dagger}_{\mathcal{S}})(z)(\varepsilon) && \text{(definition of } o_{\mathcal{S}})\\
&= t^{\#}_{\mathcal{L}}(\mathsf{reach}^{\dagger}_{\mathcal{S}}(z))(\varepsilon) && \text{(definition of } \mathsf{row}^{\#})\\
&= \mathcal{R}(z)(\varepsilon) && \text{(definition of } \mathcal{R}).
\end{aligned}$$

□

Corollary 6.5.6. *If $z \in A^\star$ is such that $\mathcal{L}_{\mathcal{S}}(z) \neq \mathcal{L}(z)$ and $\mathcal{R}(\varepsilon)(z) = \mathcal{L}(z)$, then there are $u, v \in A^\star$ and $a \in A$ such that $uav = z$ and $\mathcal{R}(ua)(v) \neq \mathcal{R}(u)(av)$.*

We can now prove the analogue of Proposition 6.5.3. Recall that this shows that a suffix of the counterexample can be added to E in order to either cause a closedness defect, or to distinguish two rows in the table. More specifically, the result below finds $u, v \in A^\star$ and $a \in A$ such that adding v to E will distinguish the previously equal $\mathsf{row}^{\#}(\mathsf{reach}^{\dagger}_{\mathcal{S}}(ua))$ and $\mathsf{srow}^{\#}(\mathsf{reach}^{\dagger}_{\mathcal{S}}(u))(a)$.

Proposition 6.5.7. *If $z \in A^\star$ is such that $\mathcal{L}_{\mathcal{S}}(z) \neq \mathcal{L}(z)$ and $\mathcal{R}(\varepsilon)(z) = \mathcal{L}(z)$, then there are $u, v \in A^\star$ and $a \in A$ such that $\mathsf{row}^{\#}(\mathsf{reach}^{\dagger}_{\mathcal{S}}(ua)) = \mathsf{srow}^{\#}(\mathsf{reach}^{\dagger}_{\mathcal{S}}(u))(a)$ and $t^{\#}_{\mathcal{L}}(\mathsf{reach}^{\dagger}_{\mathcal{S}}(ua))(v) \neq t^{\#}_{\mathcal{L}}(\mathsf{reach}^{\dagger}_{\mathcal{S}}(u))(av)$.*

Proof. Let u, a, and v be as in Corollary 6.5.6. Thus,

$$t^{\#}_{\mathcal{L}}(\mathsf{reach}^{\dagger}_{\mathcal{S}}(ua))(v) = \mathcal{R}(ua)(v) \neq \mathcal{R}(u)(av) = t^{\#}_{\mathcal{L}}(\mathsf{reach}^{\dagger}_{\mathcal{S}}(u))(av).$$

Furthermore, since for all $s \in S$ and $b \in A$ we have

$$\begin{aligned}
((\mathsf{row}^{\#})^{A} \circ \delta^{\dagger}_{\mathcal{S}})(s)(b) &= \mathsf{row}^{\#}(\delta^{\dagger}_{\mathcal{S}}(s)(b))\\
&= (\mathsf{row}^{\#} \circ \mathsf{dec})(\mathsf{srow}(s)(b)) && \text{(definition of } \delta^{\dagger}_{\mathcal{S}})\\
&= \mathsf{srow}(s)(b) && \text{(definition of dec),}
\end{aligned}$$

it follows that $(\mathsf{row}^{\#})^{A} \circ \delta_{\mathcal{S}} = \mathsf{srow}^{\#}$. Therefore,

$$\begin{aligned}
\mathsf{row}^{\#}(\mathsf{reach}^{\dagger}_{\mathcal{S}}(ua)) &= \mathsf{row}^{\#}(\delta_{\mathcal{S}}(\mathsf{reach}^{\dagger}_{\mathcal{S}}(u))(a)) && \text{(definition of } \mathsf{reach}^{\dagger}_{\mathcal{S}})\\
&= ((\mathsf{row}^{\#})^{A} \circ \delta_{\mathcal{S}})(\mathsf{reach}^{\dagger}_{\mathcal{S}}(u))(a)\\
&= \mathsf{srow}^{\#}(\mathsf{reach}^{\dagger}_{\mathcal{S}}(u))(a).
\end{aligned}$$

□

Again, we summarise the full algorithm based on $\mathrm{L}^\star_T$ from Figure 6.2. Now there are two differences: instead of the hypothesis T-automaton we construct a succinct hypothesis, and instead of adding all suffixes of a given counterexample to the set E we only add a single suffix to the set E. Thus, we only have to replace lines 2 and 3 in Algorithm 6.2. This is done as follows.

Instead of implicitly constructing the hypothesis from Definition 6.3.4, we take the current set of rows $S \subseteq A^\star$ and apply Proposition 6.4.4 to find a minimal set of generators $S' \subseteq S$ for the table. We then construct the corresponding succinct hypothesis (Definition 6.4.2) and submit it in an equivalence query. If this equivalence query returns a counterexample $z \in A^\star$, we first test whether $\mathcal{R}(\varepsilon)(z) = \mathcal{L}(z)$. If the test fails, we simply add z to E. If the test succeeds, we apply Proposition 6.5.7 to find $u, v \in A^\star$ and $a \in A$ such that $\mathsf{row}^\#(\mathsf{reach}^\dagger_S(ua)) = \mathsf{srow}^\#(\mathsf{reach}^\dagger_S(u))(a)$ and $t^\#_{\mathcal{L}}(\mathsf{reach}^\dagger_S(ua))(v) \neq t^\#_{\mathcal{L}}(\mathsf{reach}^\dagger_S(u))(av)$. We then add v to E. As before, we have shown with the results and discussion above that processing a counterexample in this way leads to either a new closedness defect or to two previously equal rows in $\mathsf{srow}^\#$ being distinguished, thus making progress like the original $\mathrm{L}^\star_T$. This means that the correctness proof Theorem 6.3.6 applies also to this variation on $\mathrm{L}^\star_T$.

Example 6.5.8. Recall the succinct hypothesis $\mathcal{S}$ from Figure 6.1c for the table in Figure 6.1a. Note that $S' = S$ cannot be further reduced. The hypothesis is based on the right inverse $\mathsf{dec} : H \longrightarrow \mathcal{P}(S)$ of $e : \mathcal{P}(S) \longrightarrow H$ given by $\mathsf{dec}(\mathsf{row}(\varepsilon)) = \{\varepsilon\}$ and $\mathsf{dec}(\mathsf{row}^\#(\emptyset)) = \emptyset$. This is the only possible right inverse because e is bijective. For the prefixes of the counterexample aa we have $\mathsf{reach}_S(\varepsilon) = \{\varepsilon\}$ and $\mathsf{reach}_S(a) = \mathsf{reach}_S(aa) = \emptyset$. Note that $t^\#_{\mathcal{L}}(\{\varepsilon\})(aa) = 1$ while $t_{\mathcal{L}}(\emptyset)(a) = t_{\mathcal{L}}(\emptyset)(\varepsilon) = 0$. Thus, $\mathcal{R}(\varepsilon)(aa) \neq \mathcal{R}(a)(a)$. Adding a to E would indeed create a closedness defect.

Query complexity. Again, we measure the membership and equivalence query complexities in terms of the number of states n of the minimal Moore automaton, the number of states t of the minimal T-automaton, the size k of the alphabet, and the length m of the longest counterexample.

A counterexample now gives an additional column instead of a set of rows, and we have seen that this leads to either a closedness defect or to two rows being distinguished. Thus, the number of equivalence queries is still at most t, and the number of columns is still in $\mathcal{O}(t)$. However, the number of rows that we need to fill using membership queries is now in $\mathcal{O}(nk)$. This means that a total of $\mathcal{O}(tnk)$ membership queries is needed to fill the table.

Apart from filling the table, we also need queries to analyze counterexamples. The binary search algorithm mentioned after Corollary 6.5.2 requires for each counterexample $\mathcal{O}(\log m)$

computations of $\mathcal{R}(x)(y)$ for varying words x and y. Let r be the maximum number of queries required for a single such computation. Note that for $u, v \in A^\star$, and letting $\alpha : TO \longrightarrow O$ be the algebra structure on O, we have $\mathcal{R}(u)(v) = \alpha(T(\mathsf{ev}_v \circ t_{\mathcal{L}})(U_{q_u}))$ for the original definition of $\mathcal{R}$ and

$$\mathcal{R}(u)(v) = \alpha(T(\mathsf{ev}_v \circ t_{\mathcal{L}})(\mathsf{reach}^\dagger_{\mathcal{S}}(u)))$$

in the succinct hypothesis case. Since the restricted map $T(\mathsf{ev}_v \circ t_{\mathcal{L}}) : TS \longrightarrow TO$ is completely determined by $\mathsf{ev}_v \circ t_{\mathcal{L}} : S \longrightarrow O$, r is at most $|S|$, which is bounded by n in this optimised algorithm. For some examples (see for instance the writer automata in Section 6.6), we even have $r = 1$. The overall membership query complexity is $\mathcal{O}(tnk + tr \log m)$.

Dropping consistency. We described the counterexample processing method based around Proposition 6.5.3 in terms of the succinct hypothesis $\mathcal{S}$ rather than the actual hypothesis $\mathcal{H}$ by showing that $\mathcal{R}$ can be defined using $\mathcal{S}$. Since the definition of the succinct hypothesis does not rely on the property of consistency to be well-defined, this means we could drop the consistency check from the algorithm altogether. We can still measure progress in terms of the size of the set H, but it will not be the state space of an actual hypothesis during intermediate stages. This observation also explains why Bollig et al. [Bol+09] are able to use a weaker notion of consistency in their algorithm. Interestingly, they exploit the canonicity of their choice of succinct hypotheses to arrive at a polynomial membership query complexity that does not involve the factor t.

6.6 Examples

In this section we list several examples that can be seen as T-automata and hence learned via an instance of $\mathtt{L}^\star_T$. We remark that, since our algorithm operates on finite structures (recall that T preserves finite sets), for each automaton type one can obtain a basic, correct-by-construction instance of $\mathtt{L}^\star_T$ for free, by plugging the concrete definition of the monad into the abstract algorithm. However, we note that this is not how $\mathtt{L}^\star_T$ is intended to be used in a real-world context; it should be seen as an abstract specification of the operations each concrete implementation needs to perform, or, in other words, as a template for real implementations. This view is the one we will take for our implementation in the next section.

For each instance below, we discuss whether certain operations admit a more efficient implementation than the basic one, based on the specific algebraic structure induced by the monad. Due to our general treatment, the optimisations of Section 6.4 and Section 6.5 apply to all of these instances.

Non-deterministic automata. As discussed before, non-deterministic automata are $\mathcal{P}$-automata with a free state space, provided that $O = 2$ is equipped with the "or" operation as its $\mathcal{P}$-algebra structure:

$$\bot = 0 \vee 0 = 0 \qquad\qquad 0 \vee 1 = 1 \vee 0 = 1 \vee 1 = 1.$$

We also mentioned that, as Bollig et al. [Bol+09] showed, there is a polynomial time algorithm to check whether a given row is the join of other rows. This gives an efficient method for handling closedness straight away. Moreover, as shown in Example 6.4.5, it allows for an efficient construction of the succinct hypothesis. Unfortunately, checking for consistency defects seems to require a number of computations exponential in the number of rows. However, as explained at the end of Section 6.5, we can in fact drop consistency altogether.

Universal automata. Just like non-deterministic automata, universal automata can be seen as $\mathcal{P}$-automata with a free state space. The difference is that the $\mathcal{P}$-algebra structure on $O = 2$ is dual: it is given by the "and" rather than the "or" operation. Thus, universal automata accept a word when all paths reading that word lead to accepting states. One can dualise the optimised specific algorithms for the case of non-deterministic automata, which is precisely what Angluin et al. [AEF15] have done.

Partial automata. Consider the *maybe monad* $\mathtt{Maybe}(X) = 1 + X$, with natural transformations having components $\eta_X : X \longrightarrow 1 + X$ and $\mu_X : 1 + 1 + X \longrightarrow 1 + X$ defined in the standard way. Partial automata with states X can be represented as $\mathtt{Maybe}$-automata with state space $\mathtt{Maybe}(X) = 1 + X$, where there is an additional *sink state*, and output algebra $O = \mathtt{Maybe}(1) = 1 + 1$. Here the left value is for rejecting states, including the sink one. The transition map $\delta : 1 + X \longrightarrow (1 + X)^A$ represents an undefined transition as one going to the sink state. The algorithm $\mathtt{L}^\star_{\mathtt{Maybe}}$ is mostly like $\mathtt{L}^\star$, except that implicitly the table has an additional row with zeroes in every column. Since the monad only adds a single element to each set, there is no need to optimise the basic algorithm for this specific case.

Weighted automata. Recall from Chapter 2 the *free semimodule monad* V, sending a set X to the free semimodule over a finite semiring $\mathbb{S}$. Weighted automata over a set of states X can be represented as V-automata whose state space is the semimodule $V(X)$, the output function $o : V(X) \longrightarrow \mathbb{S}$ assigns a weight to each state, and the transition map $\delta : V(X) \longrightarrow V(X)^A$ sends each state and each input symbol to a linear combination of states. The obvious semimodule structure on $\mathbb{S}$ extends to a pointwise structure on the potential rows of the table. The basic

algorithm loops over all linear combinations of rows to check closedness and over all pairs of combinations of rows to check consistency, making them extremely expensive operations. If $\mathbb{S}$ is a field, a row can be decomposed into a linear combination of other rows in polynomial time using standard techniques from linear algebra. As a result, there are efficient procedures for checking closedness and constructing succinct hypotheses. Below we show that consistency in this setting is equivalent to closedness of the *transpose* of the table. This trick is due to Bergadano and Varricchio [BV96], who first studied learning of weighted automata. Define the *transpose* of a table for the language $\mathcal{L}$ given by $S, E \subseteq A^\star$ as the table with row labels $\mathsf{rev}(E)$ and column labels $\mathsf{rev}(S)$ for the language $\mathsf{rev}(\mathcal{L})$, where rev reverses all words in a language.

Proposition 6.6.1. *If $T = V$ over a field $\mathbb{F}$ and the transpose of a given table is closed, then that table is consistent.*

Proof. Suppose there is an $l \in V(S)$ such that $\mathsf{row}^\#(l)(e) = 0$ for all $e \in E$. For any $a \in A$ and $e \in E$ there are by closedness of the transposed table a finite set J, $\{v_j\}_{j\in J} \subseteq \mathbb{F}$, and $\{e_j\}_{j\in J} \subseteq E$ such that for every $s \in S$,

$$\mathsf{srow}(s)(a)(e) = \sum_{j\in J} v_j \times \mathsf{row}(s)(e_j). \tag{6.4}$$

Let K be a finite set and $\{v'_k\}_{k\in K} \subseteq \mathbb{F}$ and $\{s_k\}_{k\in K} \subseteq S$ such that $l = \sum_{k\in K} v'_k \times s_k$. Then

$$\begin{aligned}
\mathsf{srow}^\#(l)(a)(e) &= \mathsf{srow}^\#\left(\sum_{k\in K} v'_k \times s_k\right)(a)(e) && \text{(definition of } l\text{)}\\
&= \left(\sum_{k\in K} v'_k \times \mathsf{srow}(s_k)\right)(a)(e) && \text{(definition of } (-)^\#\text{)}\\
&= \sum_{k\in K} v'_k \times \mathsf{srow}(s_k)(a)(e) && \text{(pointwise vector space structure)}\\
&= \sum_{k\in K} v'_k \times \sum_{j\in J} v_j \times \mathsf{row}(s_k)(e_j) && (6.4)\\
&= \sum_{j\in J} v_j \times \sum_{k\in K} v'_k \times \mathsf{row}(s_k)(e_j)\\
&= \sum_{j\in J} v_j \times \left(\sum_{k\in K} v'_k \times \mathsf{row}(s_k)\right)(e_j) && \text{(pointwise vector space structure)}\\
&= \sum_{j\in J} v_j \times \mathsf{row}^\#\left(\sum_{k\in K} v'_k \times s_k\right)(e_j) && \text{(definition of } (-)^\#\text{)}\\
&= \sum_{j\in J} v_j \times \mathsf{row}^\#(l)(e_j) && \text{(definition of } l\text{)}\\
&= \sum_{j\in J} v_j \times 0 = 0. && \qquad \square
\end{aligned}$$

In the case that $\mathbb{S}$ is a principal ideal domain, the weighted automata are the ones studied in Chapter 5. Indeed, one can see Algorithm 5.1 as an instance of Algorithm 6.2 with a succinct hypothesis representation and without performing the consistency check. Note that in Chapter 5 we did not have the finiteness constraint that is assumed in the present chapter: we assume that V preserves finite sets, which is the case if and only if $\mathbb{S}$ is finite.

Alternating automata. We use the characterisation of alternating automata due to Bertrand and Rot [BR18]. Recall that, given a partially ordered set $(P, \leq)$, an *upset* is a subset U of P such that, if $x \in U$ and $x \leq y$, then $y \in U$. Given $Q \subseteq P$, we write $\uparrow Q$ for the *upward closure* of Q, that is the smallest upset of P containing Q. We consider the monad $\mathtt{A}$ that maps a set X to the set of all upsets of $\mathcal{P}(X)$. Its unit is given by $\eta_X(x) = \uparrow\{\{x\}\}$ and its multiplication by

$$\mu_X(U) = \{V \subseteq X \mid \exists_{W \in U}\, \forall_{Y \in W}\, \exists_{Z \in Y}\, Z \subseteq V\}.$$

Algebras for the monad $\mathtt{A}$ are *completely distributive lattices* [Mar79]. The sets of sets in $\mathtt{A}(X)$ can be seen as DNF formulae over elements of X, where the outer powerset is disjunctive and the inner one is conjunctive. Accordingly, we define an algebra structure $\beta : \mathtt{A}(2) \longrightarrow 2$ on the output set 2 by letting $\beta(U) = 1$ if $\{1\} \in U$, 0 otherwise. Alternating automata with states X can be represented as $\mathtt{A}$-automata with state space $\mathtt{A}(X)$, output map $o : \mathtt{A}(X) \longrightarrow 2$, and transition map $\delta : \mathtt{A}(X) \longrightarrow \mathtt{A}(X)^A$, sending each state to a DNF formula over X. The only difference with the usual definition of alternating automata is that $\mathtt{A}(X)$ is not the full set $\mathcal{PP}(X)$, which is not a monad [KS18]. However, for each formula in $\mathcal{PP}(X)$ there is an equivalent one in $\mathtt{A}(X)$.

An adaptation of $\mathtt{L}^\star$ for alternating automata was introduced by Angluin et al. [AEF15] and further investigated by Berndt et al. [Ber+17]. The former found that given a row $r \in 2^E$ and a set of rows $X \subseteq 2^E$, r is equal to a DNF combination of rows from X (where logical operators are applied component-wise) if and only if it is equal to the combination defined by $Y = \{\{x \in X \mid x(e) = 1\} \mid e \in E \wedge r(e) = 1\}$. We can reuse this idea to efficiently find closedness defects and to construct the hypothesis. Even though the monad $\mathtt{A}$ formally requires the use of DNF formulae representing upsets, in the actual implementation we can use smaller formulae, e.g., Y above instead of its upward closure. In fact, it is easy to check that DNF combinations of rows are invariant under upward closure. Similar as before, we do not know of an efficient way to ensure consistency, but we could drop it.

Writer automata. The examples considered so far involve existing classes of automata. To further demonstrate the generality of our approach, we introduce a new (as far as we know)

type of automaton, which we call *writer automaton*.

The *writer monad* $\mathtt{Writer}(X) = \mathbb{M} \times X$ for a finite monoid $\mathbb{M}$ has a unit $\eta_X : X \longrightarrow \mathbb{M} \times X$ given by adding the unit e of the monoid, $\eta_X(x) = (e, x)$, and a multiplication $\mu_X : \mathbb{M}\times\mathbb{M}\times X \longrightarrow \mathbb{M} \times X$ given by performing the monoid multiplication, $\mu_X(m_1, m_2, x) = (m_1 m_2, x)$. In Haskell, the writer monad is used for such tasks as collecting successive log messages, where the monoid is given by the set of sets or lists of possible messages and the multiplication adds a message.

The algebras for this monad are sets Q equipped with an $\mathbb{M}$-action. One may take the output object to be the set $\mathbb{M}$ with the monoid multiplication as its action. `Writer`-automata with a free state space can be represented as deterministic automata that have an element of $\mathbb{M}$ associated with each transition. The semantics is as expected: $\mathbb{M}$-elements multiply along paths and finally multiply with the output of the last state to produce the actual output.

The basic learning algorithm has polynomial time complexity. To determine whether a given row is a combination of rows in the table, i.e., whether it is given by a monoid value applied to one of the rows in the table, one simply tries all of these values. This allows us to check for closedness, to minimise the generators, and to construct the succinct hypothesis, in polynominal time. Consistency involves comparing all ways of applying monoid values to rows and, for each comparison, at most $|A|$ further comparisons between one-letter extensions. The total number of comparisons is clearly polynomial in $|\mathbb{M}|$, $|S|$ and $|A|$.

6.7 Implementation

We have implemented the general $\mathrm{L}^\star_T$ algorithm in Haskell[4], taking full advantage of the monads provided by its standard library. Apart from the high-level implementation, our library provides

- a basic implementation for weighted automata over a finite semiring, with a polynomial time variation for the case where the semiring is a field[5];
- an implementation for non-deterministic automata that has polynomial time implementations for ensuring closedness and constructing the hypothesis, but not for ensuring consistency;

[4] `http://www.calf-project.org/files/LStarT_hs.tar.gz`

[5] Despite the assumption throughout this chapter that the monad preserves finite set, our implementation can learn weighted automata over infinite fields and thus implements the algorithm introduced by Bergadano and Varricchio [BV96], which was first studied in a categorical context in [JS14].

- a variation on the previous algorithm that uses the notion of consistency defined by Bollig et al. [Bol+09];
- instantiations of the basic algorithm to the monad being $(-) + E$, for E a finite set of exceptions, and `Writer`, both of which result in polynomial time algorithms;

In this section we describe the main structure and ingredients of our library. After recalling monads in Haskell in Section 6.7.1, we start with the formalisation of automata in Section 6.7.2. We then introduce teachers in Section 6.7.3 before exploring the actual learning algorithm in Section 6.7.4. We give details for the non-deterministic and weighted case, whose monads deserve a closer analysis.

6.7.1 Monads

We note that a monad in Haskell is specified as a *Kleisli triple* $(T, \eta, (-)^\sharp)$, where T assigns to every set X a set TX, η consists of a component $\eta_X : X \longrightarrow TX$ for each set X, and $(-)^\sharp$ provides for each function $f : X \longrightarrow TY$ an extension $f^\sharp : TX \longrightarrow TY$. These need to satisfy

$$f^\sharp \circ \eta = f \qquad \eta^\sharp = \mathsf{id} \qquad (g^\sharp \circ f)^\sharp = g^\sharp \circ f^\sharp.$$

Kleisli triples are in a one-to-one correspondence with monads. On both sides of this correspondence we have the same T and η, which for a Kleisli triple are turned into a functor with a natural transformation by setting $Tf = (\eta \circ f)^\sharp$. Furthermore, $(-)^\sharp$ and μ are obtained from each other by $f^\sharp = \mu \circ Tf$ and $\mu = \mathsf{id}^\sharp$. Indeed, under this correspondence the $(-)^\sharp$ operation is a specific instance of the extension operation defined for a monad, with the T-algebra codomain restricted to free T-algebras. In Haskell, the η of the Kleisli triple is written `return`, and, given $f : X \longrightarrow TY$ and $x \in TX$, $f^\sharp(x)$ is written `x >>= f` and referred to as the *bind* operation. Furthermore, for any $f : X \longrightarrow Y$, Tf is given by `fmap f`.

Some basic **Set** monads cannot directly be written down in Haskell because their definition can only be given on types equipped with an equality check, or, for reasons of efficiency, a total order. For example, the `Set` type provided by `Data.Set` comes with a `union` function that has the following signature:

```
union :: Ord a => Set a -> Set a -> Set a
```

One will have to use unions in one way or another in defining the bind of the powerset monad. However, since this bind needs to be of type

```
(>>=) :: Set a -> (a -> Set b) -> Set b
```

and does not assume an `Ord` instance on `b`, the powerset monad cannot be defined in this way.

One solution is to delay the monadic computations in a wrapper type whose constructors are used to define a monad instance: the free monad. Specifically, we endow the *freer monad* of Kiselyov and Ishii [KI15] with a constraint parameter:

```
data CFree c m a where
    Return :: a -> CFree c m a
    Bind :: (c b) => m b -> (b -> CFree c m a) -> CFree c m a
```

Such a constrained free monad was first defined by George Giorgidze, but only for the specific case where `m` is `Set` and `c` is `Ord`.[6] On the constrained free monad we can define a complete `Monad` instance:

```
instance Monad (CFree c m) where
    return = Return
    f >>= g = case f of
        Return a -> g a
        Bind s h -> s `Bind` (h >=> g)
```

This is the same code as used by [KI15], but we note that on the last line, since `s` is the first argument of `Bind` in `f`, we know that the appropriate constrained needed to invoke `Bind` on the right-hand side, with again `s` as its first argument, is satisfied.

Finally, if there is a monad that is defined only on types satisfying a certain constraint, then we can convert from our free monad type with that constraint back to the actual "monad":

```
class ConstrainedMonad c m | m -> c where
    constrainedReturn :: (c a) => a -> m a
    constrainedBind :: (c a, c b) => m b -> (b -> m a) -> m a

unCFree :: (ConstrainedMonad c m, c a) => CFree c m a -> m a
unCFree f = case f of
    Return a -> constrainedReturn a
    Bind s g -> s `constrainedBind` (unCFree . g)
```

Note that operations such as equality checks for `CFree c m` use `unCFree` to delegate the operation to whatever is defined for `m`. This means that in code that abstracts from the monad we seem to be working with `m` as a monad.

As an example, the `Set` "monad" becomes

[6] `https://hackage.haskell.org/package/set-monad-0.2.0.0`

```
instance ConstrainedMonad Ord Set where
    constrainedReturn = Set.singleton
    s `constrainedBind` f = Set.unions [f a | a <- Set.toList s]
```

We may then use `CFree Ord Set` as the monad.

To implement the free semimodule monad in Haskell, we use the `Map` type from `Data.Map`. Note that the monad will be defined in the first argument for that type, so we need to create an auxiliary type to swap the arguments.

```
newtype Linear s k = Linear {fromLinear :: Map k s}
```

Defining the monad again requires `Ord` constraints.

```
instance (Semiring s, Eq s) => ConstrainedMonad Ord (Linear s) where
    constrainedReturn a = Linear $ Map.singleton a mempty
    l `constrainedBind` f = Linear .
        foldl' (\m (k, s) -> ladd m . lscale s . fromLinear $ f k) Map.empty .
        Map.toList . lminimize $ fromLinear l
```

The function `lscale` scales a map by an element from the semiring; `ladd` adds two maps together. Both operations are pointwise. The monad we can use is `CFree Ord (Linear s)`.

6.7.2 Automata

We model an automaton as a simple deterministic automaton.

```
data Aut a o q = Aut {
    initial :: q,
    delta :: q -> a -> q,
    out :: q -> o }
```

For such automata, we can easily implement reachability and language functions, as well as bisimulation. Bisimulation is used to realise exact equivalence queries for the teachers that hold an automaton accepting the language to be learned. To optimise for the monad in the same way the learning algorithm is optimised, we use *bisimulation up to context* [San98; Bon+17].

```
bisimT :: (Eq o) => ((t q, t r) -> [(t q, t r)] -> Bool) ->
    [a] -> Aut a o (t q) -> Aut a o (t r) -> Maybe [a]
```

Here `t` represents the monad that we optimise for. Up to context means that, when considering a pair `p :: (t q, t r)` of next states and the current relation `b :: [(t q, t r)]`, the pair `p` does not need to be added to the relation if it can be obtained as a combination of the elements of `b`, using the free algebra structures of `t q` and `t r`. The first argument of `bisimT` is a function that should determine this. Because of this abstraction, we do not actually need to constrain `t` to be a monad here. For the `Identity` monad, one can simply use `elem` as the first argument. The second argument is the alphabet.

Succinct automata optimised by a monad `t` enjoy a more concrete representation involving maps.

```
data SAut t a o q = SAut {
    sinitial :: t q,
    sdelta :: Map q (Map a (t q)),
    sout :: Map q o }
```

This is the type of the automata that the $L^\star_T$ implementation learns. The concrete representation allows the automaton to be displayed and exported. Of course, one can determinise a succinct automaton using `t`-algebras for `a -> t q` and `o`.

```
det :: (Monad t, Ord q, Ord a) =>
    Alg t (a -> t q) -> Alg t o -> SAut t a o q -> Aut a o (t q)
```

The type `Alg t x` is defined to be `t x -> x`. We allow an arbitrary algebra on `a -> t q` rather than assuming the component `t (a -> t q) -> a -> t (t q)` of the distributive law used in earlier sections because this allows us to run the delayed monadic computations discussed earlier, which would otherwise pile up and cause serious performance issues.

6.7.3 Teaching

A teacher in our implementation is an object that comprises membership and equivalence functions. It also records the alphabet.

```
data Teacher s a o q = Teacher {
    membership :: [a] -> s o,
    equivalence :: Aut a o q -> s (Maybe [a]),
    alphabet :: [a] }
```

`Teacher` objects are parameterized by a monad `s` that serves a different purpose than optimising the learning algorithm: it is the monad of side-effects allowed by the implementation of

queries. Whereas the `Identity` monad suffices for a predefined automaton, one may have to use the `IO` monad to interact with an actual black-box system. By allowing an arbitrary monad rather than assuming the `IO` monad, we are able to build features such as query counters and a cache on top of any teacher through the use of *monad transformers*. A monad transformer provides for any monad a new monad into which the original one can be embedded. For example, the `StateT x s` monad adds a state with values in `x` to an existing monad `s`. This is the transformer that enables the addition of query counters and a cache to a teacher:

```
countTeacher :: (Monad s) =>
    Teacher s a o q -> Teacher (StateT (Int, Int) s) a o q
cacheTeacher :: (Monad s, Ord a) =>
    Teacher s a o q -> Teacher (StateT (Map [a] o) s) a o q
```

The most basic teacher holds an automaton that it uses to determine membership and equivalence, the latter of which is implemented through bisimulation.

```
autTeacherT :: (Monad s, Eq o) => ((t q, t r) -> [(t q, t r)] -> Bool) ->
    [a] -> Aut a o (t q) -> Teacher s a o (t r)
```

It implements a `Teacher` for any monad `s` because it does not have any side-effects.

We also provide a teacher that implements equivalence queries through random testing.

```
randomTeacher :: (Monad s, Eq o) => Int -> State StdGen [a] ->
    [a] -> ([a] -> s o) -> Teacher (StateT StdGen s) a o q
```

Its first argument is the number of tests per equivalence query, while the second argument samples test words: `StdGen` is a random number generator. Once more we use the `StateT` monad transformer, in this case to add a random number generator state to the monad `s` that the membership query function, which is the last argument, may use. This query function is used both for membership queries and for generated test queries. Note that this particular teacher does not give any guarantees on the validity of positive responses to equivalence queries. We do also provide the random sampling teacher suggested by Angluin [Ang87], which guarantees that on a positive answer the hypothesis is *probably approximately correct*, a notion introduced by Valiant [Val84].

```
pacTeacher :: (Monad s, Eq o) => Double -> Double -> State StdGen [a] ->
    [a] -> ([a] -> s o) -> Teacher (StateT (Int, StdGen) s) a o q
```

Here the first argument is the accuracy ϵ, while the second one is the confidence ∂. Both should be values between 0 and 1. If $d : A^\star \longrightarrow [0,1]$ is the distribution represented by the third argument (converting between Haskell types and sets for convenience) and $l_1, l_2 : A^\star \longrightarrow O$ are the languages of the hypothesis and the target, the guarantee is that, with probability at least $1 - \partial$, $\sum_{u \in A^\star, l_1(u) \neq l_2(u)} d(u) \leq \epsilon$. Compared to `randomTeacher`, an `Int` has been added to the state because the number of tests depends on the number of equivalence queries that have already been asked.

6.7.4 Learning

We define a `Learner` type that allows us to switch between variations on $\mathrm{L}^\star_T$ and to optimise certain specific procedures.

```
data Learner t a o = Learner {
    decomposeRow :: ObservationTable a o -> [[a]] -> [o] -> Maybe (t [a]),
    consistencyDefect :: Maybe (ObservationTable a o -> Maybe [a]),
    ceh :: CEHandler }
```

The function `decomposeRow` takes an observation table, a list of labels `l`, and a row `r`, and determines whether `r` can be obtained as a combination of the rows with labels in `l`. If this is the case, it returns the combination, which has type `t [a]`. This function is used to check closedness, to minimise the labels used as states for the hypothesis, and to construct the hypothesis. If `consistencyDefect` is set to `Nothing`, it indicates that consistency should be solved by solving closedness for what we call the *transpose* of the table (swapping S and E and reversing their words while considering the reverse of the target language as the target language); otherwise, it contains a function that given an observation table produces a new column to fix one of its consistency defects, unless the table is already consistent. Solving closedness for the transpose of the table always ensures consistency, but in general it may add more columns than necessary. Lastly, `CEHandler` is a type that enumerates our adaptations of the three counterexample handling methods: the original one by Angluin [Ang87], the one by Maler and Pnueli [MP95], and the one by Rivest and Schapire [RS93].

To enable basic implementations of `decomposeRow` and `consistencyDefect` that work for any monad T (preserving finite sets), we need to be able to loop over the values of TS. In order to facilitate this, there is a class `Concrete f` whose only member function turns a list of values of any type into a list of values with type `f` applied to that type. It is intended to be the concrete application of a functor to a set (represented as a list). We provide the func-

tions `lazyDecomposeRow` and `lazyConsistencyDefect`, both conditioned with a `Concrete t` constraint, which directly enable a basic version of the learning algorithm.

To optimise the algorithm in a specific setting, a programmer only has to adjust these two functions. We provide such optimised functions for the cases of non-deterministic and weighted automata (over a field). Regarding the former case, we provide `crfsaDecompose` and `scrfsaDecompose`, which are essentially the right inverses corresponding to the canonical and simplified canonical RFSA, respectively, as explained in Example 6.4.5. Our optimised weighted algorithm uses Gaussian elimination in a function called `gaussianDecomposeRow` and solves consistency by solving closedness for the transpose of the table, a method readily available regardless of the monad.

Enabling our adaptation of the counterexample handling method due to Rivest and Schapire requires an additional condition. Recall that this method requires us to pose membership queries for combinations of words, which can be done by extending the membership query function (the language) of type `[a] -> o` to one of type `t [a] -> o` using the algebra structure defined on `o`. However, our membership query function actually has type `[a] -> s o`, and there is no reason to assume any interaction between `s` and `t`. As a workaround, we will assume an instance of `Supported` for the monad `t`, where `Supported` is a class defined as follows:

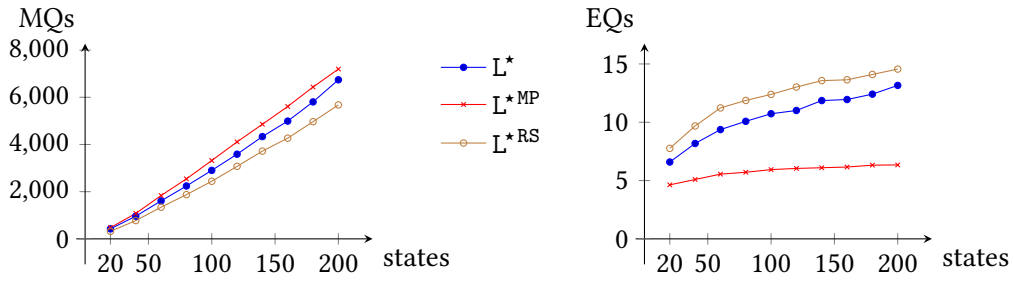
```
class Supported f where
    supp :: (Ord a) => f a -> [a]
```

Given any `u :: f a` and `g :: a -> b`, we require `supp u` to be such that the computation of `fmap g u` only evaluates `g` on the elements of `supp u`. Naturally, we want `supp u` to be as small as possible: it should contain exactly those elements of type `a` that are present in `u`. As an example, recall that the free semimodule monad with values in a semiring `s` can be defined on a type `a` as `Map a s`, where we identify a missing value for an element with that element being assigned zero. Given `u :: Map a s`, `supp u` is given by the keys of the map `u` that are assigned a non-zero value.

Using the instance for a monad `t`, the membership query function can be extended by querying the words in the support of a given element of `t [a]` sequentially, constructing a partial membership query function defined only on that support, and evaluating the extension of that function. This method works because we assume that the side-effects exhibited by `s` do not influence future membership queries.

Finally, our general $\mathtt{L}^\star_T$ implementation has the following signature:

```
lStarT :: (Monad s, Monad t, Supported t, Ord a, Eq o) =>
```



(a) Membership queries

(b) Equivalence queries

Figure 6.3: L* variations on random DFAs.

```
Alg t (a -> t [a]) -> Alg t o ->
Teacher s a o (t [a]) -> Learner t a o -> s (SAut t a o [a])
```

6.8 Experiments

In this section we analyze the performance, in terms of number of queries, of several variations of our algorithm by running them on randomly generated WFAs, NFAs, and plain Moore automata. Our aim is to show the effect of exploiting the right monad and of using our adapted optimised counterexample handling method. We note that different algorithms considered may produce different types of automata. However, when we compare two algorithms they will produce automata accepting the same language. We will compare them on how many membership and equivalence queries they need.

The experiments are run using the implementation discussed in Section 6.7. In all cases we use an alphabet of size 3. Random Moore automata are generated by choosing for each state an output and further for each input symbol a next state using uniform distributions. The WFAs are over the field of size 5. Here the outputs are chosen in the same way, and for each pair of states and each input symbol, we create a transition symbol from the first to the second state with a random weight chosen uniformly. We take the average of 100 iterations for each of the sizes for which we generate automata. Membership query results in tables will be rounded to whole numbers. We use bisimulation to find counterexamples in all experiments, exploiting the fact that the target automaton will be known. We cache membership queries so that the counts exclude duplicates.

For reference, Figure 6.3 compares L* and the two counterexample handling variations by Maler and Pnueli (denoted MP) and by Rivest and Schapire (denoted RS), on randomly

generated DFAs of size 20 through 200 with increments of 20. Compared to $\mathtt{L}^\star$, both $\mathtt{L}^{\star\,\mathtt{MP}}$ and $\mathtt{L}^{\star\,\mathtt{RS}}$ remove the need for consistency checks. Interestingly, whereas $\mathtt{L}^{\star\,\mathtt{RS}}$ compared to $\mathtt{L}^\star$ improves in membership queries and worsens in equivalence queries, the situation is reversed for $\mathtt{L}^{\star\,\mathtt{MP}}$.

6.8.1 Comparing $\mathtt{L}^\star_V$ to $\mathtt{L}^\star$

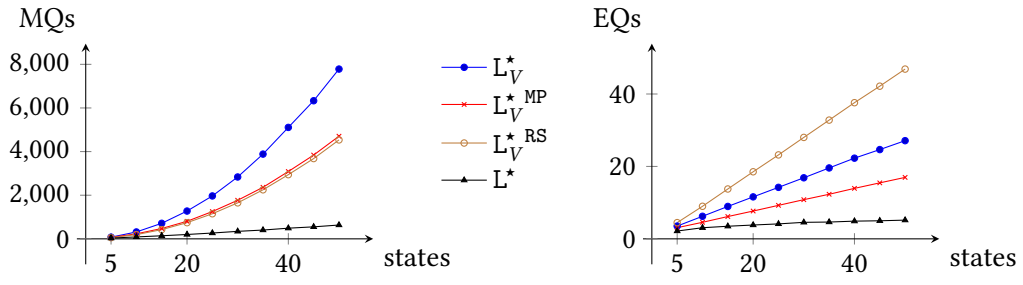
In Table 6.1 we compare the performance of $\mathtt{L}^\star$ with that of $\mathtt{L}^\star_V$. (Recall that V is the free vector space monad.)

Here $\mathtt{L}^\star$ is the obvious generalisation of the original $\mathtt{L}^\star$ algorithm to learn Moore automata—DFAs with outputs in an arbitrary set, which here is the field with five elements. Thus, as opposed to $\mathtt{L}^\star_V$, $\mathtt{L}^\star$ ignores the vector space structure on the output set. In both cases we consider the three different counterexample handling methods. The algorithms are run on randomly generated WFAs of sizes 1 through 4. As expected, each $\mathtt{L}^\star_V$ variation provides a massive gain over the corresponding $\mathtt{L}^\star$ variation in terms of membership queries, and a more modest one in terms of equivalence queries. Comparing the results of the $\mathtt{L}^\star$ variations, we see that the membership query results of $\mathtt{L}^\star$ and $\mathtt{L}^{\star\,\mathtt{RS}}$ are extremely close together. Other than that, the ordering of the counterexample handling methods is the same as with the DFA experiments. The $\mathtt{L}^\star_V$ variations will be compared in more detail later.

Now we run $\mathtt{L}^\star$ and variations of $\mathtt{L}^\star_V$ on randomly generated Moore automata of sizes 5 through 50 with increments of 5. We chose to compare the $\mathtt{L}^\star_V$ variations only to $\mathtt{L}^\star$ because of its average performance in between $\mathtt{L}^{\star\,\mathtt{MP}}$ and $\mathtt{L}^{\star\,\mathtt{RS}}$ as seen in Figure 6.3. The results are shown in Figure 6.4. We see that, in terms of membership queries, both `RS` and `MP` counterexample handling methods improve over the one by Angluin in this setting, and `MP` performs best in terms of either query type. In these experiments, $\mathtt{L}^\star$ performs much better than the algorithms that attempt to take advantage of the non-existent vector space structure. Together with the

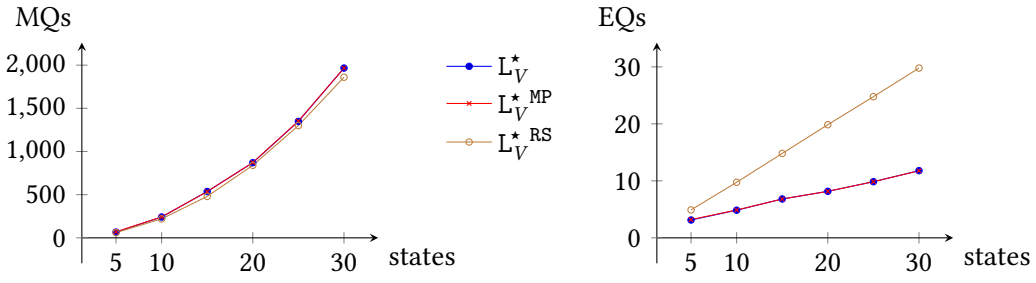
	MQs						EQs					
Size	$\mathtt{L}^\star$	$\mathtt{L}^\star_V$	$\mathtt{L}^{\star\,\mathtt{MP}}$	$\mathtt{L}^\star_V{}^{\mathtt{MP}}$	$\mathtt{L}^{\star\,\mathtt{RS}}$	$\mathtt{L}^\star_V{}^{\mathtt{RS}}$	$\mathtt{L}^\star$	$\mathtt{L}^\star_V$	$\mathtt{L}^{\star\,\mathtt{MP}}$	$\mathtt{L}^\star_V{}^{\mathtt{MP}}$	$\mathtt{L}^{\star\,\mathtt{RS}}$	$\mathtt{L}^\star_V{}^{\mathtt{RS}}$
1	10	4	10	4	10	4	1.00	1.00	1.00	1.00	1.00	1.00
2	105	15	154	15	104	11	1.86	1.73	1.86	1.73	1.86	1.73
3	845	27	1003	27	844	24	2.84	2.10	2.16	2.14	3.00	2.81
4	5570	50	7904	50	5567	40	3.71	2.88	2.90	2.83	3.97	3.78

Table 6.1: $\mathtt{L}^\star$ variations and $\mathtt{L}^\star_V$ variations on random WFAs.



(a) Membership queries (b) Equivalence queries

Figure 6.4: $L^\star_V$ variations and $L^\star$ on random Moore automata.



(a) Membership queries (b) Equivalence queries

Figure 6.5: $L^\star_V$ variations on random WFAs.

results in Table 6.1, this is consistent with the findings of Angluin et al. [AEF15]: they found that for DFAs and non-deterministic, universal, and alternating automata, the adaptation of $L^\star$ that takes advantage of the exact type of structure of the randomly generated target automata performs the best.

Figure 6.5 illustrates the performance of $L^\star_V$ variations on randomly generated WFAs. Here we generated WFAs of sizes 5 through 30 with increments of 5. We emphasize that in Table 6.1 we could not go beyond size 4, because of performance issues with $L^\star$. There is hardly any difference between the use of Angluin's counterexample handling method and MP, neither in terms of membership queries, nor in terms of equivalence queries. Interestingly, while the RS method performs worse than the other methods in terms of equivalence queries, as usual, it provides no significant gain in terms of membership queries. We ran these experiments also with the variations on the MP and RS algorithms where we drop the consistency checks. In both cases the differences were negligible.

6.8.2 Comparing $\mathtt{NL}^\star$ to $\mathtt{L}^\star$

We now consider learning algorithms for NFAs. To generate random NFAs, we use the strategy introduced by Tabakov and Vardi [TV05] with a transition density of 1.25, meaning that for each input symbol there are on average 1.25 transitions originating from each state. According to Tabakov and Vardi, this density results in the largest equivalent minimal DFAs. Like Tabakov and Vardi, we let half of the states be accepting. We ran several variations of $\mathtt{L}^\star$ and $\mathtt{NL}^\star$ on randomly generated NFAs of sizes 4 through 16 with increments of 4. The results are shown in Table 6.2. Here $\mathtt{NL}^{\star\,\mathtt{MP}}$ refers to the original algorithm by Bollig et al. [Bol+09], with their notion of consistency; $\mathtt{NL}^{\star\,\mathtt{RS}}$ is the same algorithm, but using the counterexample handling method that we adapted from Rivest and Schapire's. The variations $\mathtt{NL}^{\star\,\mathtt{MP-}}$ and $\mathtt{NL}^{\star\,\mathtt{RS-}}$ drop the consistency checks altogether. Unfortunately, doing the full consistency check was not computationally feasible. As expected, the $\mathtt{NL}^\star$ algorithms yield a great improvement over $\mathtt{L}^\star$ in terms of membership queries, and in most cases they also improve in terms of equivalence queries. This was already observed by Bollig et al. The exception is $\mathtt{NL}^{\star\,\mathtt{RS-}}$, which, despite having the best membership query results, requires by far the most equivalence queries. As happened to $\mathtt{L}^\star$ on DFAs, switching within $\mathtt{NL}^\star$ from the MP to the RS counterexample handling method improves the performance in terms of membership queries and worsens it in terms of equivalence queries. Dropping consistency altogether turns out to increase both query numbers.

6.9 Discussion

We have presented $\mathtt{L}^\star_T$, a general adaptation of $\mathtt{L}^\star$ that uses monads to learn an automaton with algebraic structure, as well as a method for finding a succinct equivalent based on its generators. Furthermore, we adapted the optimised counterexample handling method of Rivest and

	MQs					EQs				
Size	$\mathtt{L}^\star$	$\mathtt{NL}^{\star\,\mathtt{MP}}$	$\mathtt{NL}^{\star\,\mathtt{MP-}}$	$\mathtt{NL}^{\star\,\mathtt{RS}}$	$\mathtt{NL}^{\star\,\mathtt{RS-}}$	$\mathtt{L}^\star$	$\mathtt{NL}^{\star\,\mathtt{MP}}$	$\mathtt{NL}^{\star\,\mathtt{MP-}}$	$\mathtt{NL}^{\star\,\mathtt{RS}}$	$\mathtt{NL}^{\star\,\mathtt{RS-}}$
4	138	79	82	55	54	3.75	3.01	3.64	3.59	4.64
8	1792	666	729	389	381	10.38	6.52	8.83	9.37	14.10
12	11130	2467	2701	1331	1286	18.93	11.08	14.92	17.88	27.61
16	38256	5699	6240	3036	2999	28.81	15.75	22.59	27.29	45.53

Table 6.2: $\mathtt{L}^\star$ and $\mathtt{NL}^\star$ variations on random NFAs.

Schapire [RS93] to this setting and discussed instantiations to non-deterministic, universal, partial, weighted, alternating, and writer automata. We have provided a prototype implementation in Haskell, using which we obtained experimental results confirming that exploiting the algebraic structure reduces the number of queries posed. The results also reveal that the best counterexample handling method depends on the type of automata considered and the algebraic structure exploited by the algorithm. We found that there is a significant gain in membership queries compared to the $\mathtt{NL}^\star$ algorithm by Bollig et al. [Bol+09] when using our adapted optimised counterexample handling method.

Related Work. An adaptation of $\mathtt{L}^\star$ that produces NFAs was first developed by Bollig et al. [Bol+09]. Their algorithm learns a special subclass of NFAs consisting of RFSAs, which were introduced by Denis et al. [DLT02]. Angluin et al. [AEF15] unified algorithms for NFAs, universal automata, and alternating automata, the latter of which was further improved by Berndt et al. [Ber+17]. We are able to provide a more general framework, which encompasses and goes beyond those classes of automata. Moreover, we study optimised counterexample handling, which [AEF15; Bol+09; Ber+17] do not consider.

The algorithm for weighted automata over an arbitrary field was studied in a category theoretical context by Jacobs and Silva [JS14]. The algorithm itself was introduced by Bergadano and Varricchio [BV96]. The theory of succinct automata used for our hypotheses is based on the work of Arbib and Manes [AM75b], revamped to more recent category theory.

Our library is currently a prototype, which is not intended to compete with a state-of-the-art tool such as LearnLib [IHS15] or other automata learning libraries like libalf [Bol+10]. Our Haskell implementation does not provide the computational efficiency achieved by LearnLib, which furthermore includes the TTT-algorithm with its optimised data structure that in particular replaces the observation table by a tree [IHS14]. Such an optimisation is ad-hoc for DFAs, and an extension to other classes of automata is not trivial. First steps in this direction have been set by [HSS17a], who have studied the tree data structure in a more general setting. We intend to further pursue investigation in this direction, in order to allow for optimised data structures in a future version of our library. We note that, although libalf supports NFAs, none of the existing tools and libraries offers the flexibility of our library, in terms of available optimisations and classes of models that can be learned.

Future Work. Whereas our general algorithm effortlessly instantiates to monads that preserve finite sets, a major challenge lies in investigating monads that do not enjoy this property. The algorithm for weighted automata generalises to an infinite field [BV96; JS14] and, as we

saw in Chapter 5, even a principal ideal domain. However, we also showed in Chapter 5 that for an infinite semiring in general we cannot guarantee termination, which is because a finitely generated semimodule may have an infinite chain of strict submodules. Intuitively, this means that while fixing closedness defects increases the size of the hypothesis state space semimodule, an infinite number of steps may be needed to resolve all closedness defects. In future work we would like to characterize more precisely for which semirings we can learn, and ideally formulate this characterisation on the monad level.

As a result of the correspondence between learning and conformance testing [Ber+05; HSS17a], it should be possible to include in our framework the W-method [Cho78], which is often used in case studies deploying $\mathtt{L}^\star$ (e.g. [Cha+14; RP15]). We defer a thorough investigation of conformance testing to future work.

Chapter 7

Further Directions

In this thesis we have developed CALF: a categorical automata learning framework built around an abstract version of the $\mathtt{L}^\star$ algorithm. Using this framework we recovered and extended an algorithm to learn tree automata, we generalised a learning algorithm for WFAs to include WFAs over PIDs, and we provided a general algorithm to learn automata with side-effects, parametric on a monad. In this chapter we explore the perspective CALF offers on various avenues that could be pursued for future work within the field of automata learning.

A first direction worth exploring is optimising classes of learning algorithms, such as the one from Chapter 6. We have already shown that the counterexample handling optimisation due to Rivest and Schapire [RS93] can be transferred from the DFA setting to our algorithm for a wide range of automata with side-effects (Section 6.5). This transfer was enabled by our categorical view on these automata, interpreting them in the category of algebras for a monad. Apart from the counterexample handling method, potential ingredients of the algorithm to optimise include the way closedness and consistency are ensured, as well as the data structure that represents the wrappers.

One alternative datastructure to replace the observation tables of $\mathtt{L}^\star$ was proposed in [KV94], where so-called *classification trees* are used instead. In an observation table, prefixes are distinguished when they are assigned different rows, which are obtained by appending suffixes from a fixed set and querying membership on the resulting concatenations. A classification tree generalises this by making the suffixes form the nodes of a binary tree. Given a prefix, one starts at the root node. Taking the suffix corresponding to the current node, one queries membership on the concatenation of the prefix and the suffix, the result of which determines which of the two subtrees is considered next. The set of leaves of such a tree forms a set of labels that serve to distinguish prefixes. When two previously equivalent prefixes are

discovered to be different, for instance due to a consistency defect, one simply replaces their leaf with a node containing the distinguishing experiment. This gains much efficiency compared to the analogous process for an observation table, which requires a membership query for every row when a column is added.

Both observation tables and classification trees can be modelled using the notion of wrapper that is central to CALF [HSS17a]. The categorical view tells us more about redesigning classification trees to be applied in learning algorithms for different types of automata: for instance, in the WFA setting the operation that assigns to a prefix its label needs to factor through a linear map from the target (minimal) linear weighted automaton to the set of labels. This constraint generalises to automata with side-effects (Chapter 6), where the corresponding algebraic structure needs to be preserved. It turns out that this is a challenging constraint to maintain in general, and transferring the optimisation by adapting it to suitable alternatives is left to future research.

Another avenue to explore further is to study existing automata learning algorithms by comparing them an algorithm derived from CALF, and to derive algorithms for other types of automata when a new application is identified. Here, the categorical view directly drives the design of the algorithm: the majority of the effort lies in modelling the automaton using the categorical notion of automaton (Definition 2.2.10). Using this model, major parts of developing an algorithm come for free by instantiating Algorithm 4.2.

Fitting an automaton type in the categorical model can be a challenging task. For instance, so far no suitable categorical treatment of register automata has been identified. Although nominal automata [BKL14], which recognise the same class of languages, can be modelled categorically and have been learned with an adaptation of $L^\star$ [Moe+17], they are not as succinct as traditional register automata can be. This is because the "registers" of a state of a nominal automaton must adhere to fixed equality constraints. Thus, if two values can but do not have to be the same, different states are needed. This explodes as the number of registers lacking such clear relations increases. Progress on this drawback of nominal automata was made by Moerman and Rot [MR20], who introduced *separated nominal automata* based on the theory of *nominal renaming sets* [GH08]. These automata achieve the same succinctness as register automata, but unfortunately they can only be used to accept languages of words in which data values do not occur twice. Future research may lead to a hybrid automaton model that combines the expressivity of nominal automata with the succinctness of separated nominal automata.

One of the open problems identified in Chapter 5 is the search for a more precise characterisation of the semirings for which our algorithm terminates. In Chapter 6 we generalised

this problem: we wondered if we could characterise the monads for which our general algorithm to learn automata with side-effects, currently presented with a limitation to monads preserving finite sets, terminates. As a first step, it would seem that the termination proof Theorem 5.2.10 generalises without much effort to the level of monads, along with the definitions of closedness strategy and progress measure that it uses. Once this generalisation of the theory in Chapter 6 is made, one additional example worth exploring is that of *subsequential transducers*, which accumulate words of output symbols along their transitions. Here the side-effect is a monad pairing the set of output words with a given set, which is an operation that does not preserve finite sets and is therefore not covered by Chapter 6 in its current form.

A broader goal for future work is to exhibit an extensive portfolio of automata learning algorithms covered by CALF. Apart from subsequential transducers, we expect to be able to cover for instance the families of DFAs of Angluin and Fisman [AF16] in the form of the dependent coalgebras of Ciancia and Venema [CV12]. These are also an instance of the categorical algorithm in [US19], but as with tree automata in that work a reduction to an automaton accepting words is used. We expect to recover an algorithm similar to the one by Angluin and Fisman.

Another class of automata for which automata learning techniques have not yet been developed is given by pomset automata, which are used in concurrency theory [Kap+19]. We expect that the flexibility of CALF will allow the development of such algorithms and their applications in verification.

Ultimately, CALF and the automata learning algorithm at its core provide a powerful toolbox for future studies of automata learning. The framework offers a deep understanding that can be used to compare automata learning algorithms in general, and to transfer optimisations between them; at the same time, new algorithms that are correct by construction can be derived from the abstract template. Being formulated on an abstract level, the framework provides a strong guideline for designing algorithms but does not obstruct choices of implemenation details, leaving room in particular for ad hoc optimisations.

Bibliography

[AAM76] Brian D.O. Anderson, Michael A. Arbib and Ernest G. Manes. *Foundations of system theory: finitary and infinitary conditions*. Vol. 115. Lecture Notes in Econ. and Math. Syst. Springer, 1976. DOI: 10.1007/978-3-642-45479-0.

[Aar+15] Fides Aarts, Paul Fiterău-Broștean, Harco Kuppens and Frits W. Vaandrager. "Learning Register Automata with Fresh Value Generation". In: *ICTAC*. Vol. 9399. LNCS. Springer, 2015, pp. 165–183. DOI: 10.1007/978-3-319-25150-9_11.

[Adá+12] Jiří Adámek, Filippo Bonchi, Mathias Hülsbusch, Barbara König, Stefan Milius and Alexandra Silva. "A Coalgebraic Perspective on Minimization and Determinization". In: *FoSSaCS*. Vol. 7213. LNCS. Springer, 2012, pp. 58–73. DOI: 10.1007/978-3-642-28729-9_4.

[Adá74] Jiří Adámek. "Free algebras and automata realizations in the language of categories". In: *Commentationes Mathematicae Universitatis Carolinae* 15 (1974), pp. 589–602. URL: http://dml.cz/dmlcz/105583.

[Adá77] Jiří Adámek. "Realization theory for automata in categories". In: *J. Pure and Appl. Algebra* 9 (1977), pp. 281–296. DOI: 10.1016/0022-4049(77)90071-8.

[AEF15] Dana Angluin, Sarah Eisenstat and Dana Fisman. "Learning regular languages via alternating automata". In: *IJCAI*. 2015, pp. 3308–3314.

[AF16] Dana Angluin and Dana Fisman. "Learning regular omega languages". In: *Theor. Comput. Sci.* 650 (2016), pp. 57–72. DOI: 10.1016/j.tcs.2016.07.031.

[AHK07] Parosh Aziz Abdulla, Johanna Högberg and Lisa Kaati. "Bisimulation Minimization of Tree Automata". In: *Int. J. Found. Comput. Sci.* 18 (2007), pp. 699–713. DOI: 10.1142/S0129054107004929.

[AHS09] Jiří Adámek, Horst Herrlich and George E. Strecker. *Abstract and Concrete Categories - The Joy of Cats*. Dover Publications, 2009. URL: http://store.doverpublications.com/0486469344.html.

[AKL10] Benjamin Aminof, Orna Kupferman and Robby Lampert. "Reasoning about online algorithms with weighted automata". In: *ACM Trans. Algorithms* 6 (2010), 28:1–28:36. DOI: 10.1145/1721837.1721844.

[AKL11] Benjamin Aminof, Orna Kupferman and Robby Lampert. "Formal Analysis of Online Algorithms". In: *ATVA*. 2011, pp. 213–227. DOI: 10.1007/978-3-642-24372-1_16.

[ALM07] Jiří Adámek, Dominik Lücke and Stefan Milius. "Recursive coalgebras of finitary functors". In: *RAIRO Theor. Informatics Appl.* 41 (2007), pp. 447–462. DOI: 10.1051/ita:2007028.

[AM74] Michael A. Arbib and Ernest G. Manes. "Machines in a category: An expository introduction". In: *SIAM review* 16 (1974), pp. 163–192. URL: https://www.jstor.org/stable/2028458.

[AM75a] Michael A. Arbib and Ernest G. Manes. "Adjoint machines, state-behavior machines, and duality". In: *J. Pure and Appl. Algebra* 6 (1975), pp. 313–344. DOI: 10.1016/0022-4049(75)90028-6.

[AM75b] Michael A. Arbib and Ernest G. Manes. "Fuzzy machines in a category". In: *Bull. Austral. Math. Soc.* 13 (1975), pp. 169–210. DOI: 10.1017/S0004972700024412.

[AMM20] Jirí Adámek, Stefan Milius and Lawrence S. Moss. "On Well-Founded and Recursive Coalgebras". In: *FoSSaCS*. Vol. 12077. LNCS. Springer, 2020, pp. 17–36. DOI: 10.1007/978-3-030-45231-5_2.

[AMT08] Cyril Allauzen, Mehryar Mohri and Ameet Talwalkar. "Sequence kernels for predicting protein essentiality". In: *ICML*. 2008, pp. 9–16. DOI: 10.1145/1390156.1390158.

[AMV03] Jiří Adámek, Stefan Milius and Jiří Velebil. "Free iterative theories: a coalgebraic view". In: *Math. Struct. Comput. Sci.* 13 (2003), pp. 259–320. DOI: 10.1017/S0960129502003924.

[Ang81] Dana Angluin. "A note on the number of queries needed to identify regular languages". In: *Inf. Control.* 51 (1981), pp. 76–87. DOI: 10.1016/S0019-9958(81)90090-5.

[Ang87] Dana Angluin. "Learning Regular Sets from Queries and Counterexamples". In: *Inf. Comput.* 75 (1987), pp. 87–106. DOI: 10.1016/0890-5401(87)90052-6.

[Ang88] Dana Angluin. "Queries and concept learning". In: *Mach. Learn.* 2 (1988), pp. 319–342. DOI: 10.1007/BF00116828.

[AT89] Jiří Adámek and Vera Trnková. *Automata and algebras in categories*. Vol. 37. Mathematics and Its Applications. Kluwer, 1989.

[AV10] Fides Aarts and Frits W. Vaandrager. "Learning I/O Automata". In: *CONCUR*. Vol. 6269. LNCS. Springer, 2010, pp. 71–85. DOI: 10.1007/978-3-642-15375-4_6.

[Awo10] Steve Awodey. *Category theory*. Oxford University Press, 2010.

[AZ69] Michael A. Arbib and H. Paul Zeiger. "On the relevance of abstract algebra to control theory". In: *Autom.* 5 (1969), pp. 589–606. DOI: 10.1016/0005-1098(69)90026-0.

[Bal+97] José L. Balcázar, Josep Díaz, Ricard Gavaldà and Osamu Watanabe. "Algorithms for Learning Finite Automata from Queries: A Unified View". In: *Advances in Algorithms, Languages, and Complexity*. Springer, 1997, pp. 53–72. DOI: 10.1007/978-1-4613-3394-4_2.

[Bar70] Michael Barr. "Coequalizers and free triples". In: *Math. Z.* 116 (1970), pp. 307–322. DOI: 10.1007/BF01111838.

[Ber+05] Therese Berg, Olga Grinchtein, Bengt Jonsson, Martin Leucker, Harald Raffelt and Bernhard Steffen. "On the correspondence between conformance testing and regular inference". In: *FASE*. Vol. 3442. LNCS. Springer, 2005, pp. 175–189. DOI: 10.1007/978-3-540-31984-9_14.

[Ber+17] Sebastian Berndt, Maciej Liśkiewicz, Matthias Lutter and Rüdiger Reischuk. "Learning Residual Alternating Automata". In: *AAAI*. AAAI Press, 2017, pp. 1749–1755. URL: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14748.

[BGC09] Christel Baier, Marcus Größer and Frank Ciesinski. "Model checking linear-time properties of probabilistic systems". In: *Handbook of Weighted automata*. 2009. DOI: 10.1007/978-3-642-01492-5_13.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008. URL: https://mitpress.mit.edu/books/principles-model-checking.

[BKL14] Mikołaj Bojańczyk, Bartek Klin and Sławomir Lasota. "Automata theory in nominal sets". In: *LMCS* 10 (2014). DOI: 10.2168/LMCS-10(3:4)2014.

[BKR19] Simone Barlocco, Clemens Kupke and Jurriaan Rot. "Coalgebra learning via duality". In: *FoSSaCS*. Vol. 11425. LNCS. Springer, 2019, pp. 62–79. DOI: 10.1007/978-3-030-17127-8_4.

[Blo12] Alwin Blok. "Interaction, observation and denotation". MA thesis. ILLC Amsterdam, 2012. URL: https://eprints.illc.uva.nl/872/1/MoL-2012-06.text.pdf.

[BM07] Jérôme Besombes and Jean-Yves Marion. "Learning tree languages from positive examples and membership queries". In: *Theor. Comput. Sci.* 382 (2007), pp. 183–197. DOI: 10.1016/j.tcs.2007.03.038.

[BM12] Borja Balle and Mehryar Mohri. "Spectral Learning of General Weighted Automata via Constrained Matrix Completion". In: *NIPS*. 2012, pp. 2168–2176. URL: http://papers.nips.cc/paper/4697-spectral-learning-of-general-weighted-automata-via-constrained-matrix-completion.

[BM15] Borja Balle and Mehryar Mohri. "Learning Weighted Automata". In: *CAI*. 2015, pp. 1–21. DOI: 10.1007/978-3-319-23021-4_1.

[Boj15] Mikołaj Bojańczyk. "Recognisable languages over monads". In: *DLT*. Springer. 2015, pp. 1–13.

[Bol+08] Benedikt Bollig, Peter Habermehl, Carsten Kern and Martin Leucker. *Angluin-Style Learning of NFA (Research Report LSV-08-28)*. Tech. rep. ENS Cachan, 2008. URL: http://www.lsv.fr/Publis/RAPPORTS_LSV/PDF/rr-lsv-2008-28.pdf.

[Bol+09] Benedikt Bollig, Peter Habermehl, Carsten Kern and Martin Leucker. "Angluin-Style Learning of NFA". In: *IJCAI*. 2009, pp. 1004–1009. URL: http://ijcai.org/Proceedings/09/Papers/170.pdf.

[Bol+10] Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker, Daniel Neider and David R Piegdon. "libalf: The automata learning framework". In: *CAV*. Vol. 6174. LNCS. Springer, 2010, pp. 360–364. DOI: 10.1007/978-3-642-14295-6_32.

[Bol+13] Benedikt Bollig, Peter Habermehl, Martin Leucker and Benjamin Monmege. "A fresh approach to learning register automata". In: *DLT*. Vol. 7907. LNCS. Springer, 2013, pp. 118–130. DOI: 10.1007/978-3-642-38771-5_12.

[Bon+17] Filippo Bonchi, Daniela Petrisan, Damien Pous and Jurriaan Rot. "A general account of coinduction up-to". In: *Acta Inf.* 54 (2017), pp. 127–190. DOI: 10.1007/s00236-016-0271-4.

[BP15] Filippo Bonchi and Damien Pous. "Hacking nondeterminism with induction and coinduction". In: *Commun. ACM* 58 (2015), pp. 87–95. DOI: 10.1145/2713167.

[BR18] Meven Bertrand and Jurriaan Rot. "Coalgebraic determinization of alternating automata". In: *CoRR* (2018). arXiv: 1804.02546. URL: https://arxiv.org/abs/1804.02546.

[BV96] Francesco Bergadano and Stefano Varricchio. "Learning Behaviors of Automata from Multiplicity and Equivalence Queries". In: *SIAM J. Comput.* 25 (1996), pp. 1268–1280. DOI: 10.1137/S009753979326091X.

[Cas+16] Sofia Cassel, Falk Howar, Bengt Jonsson and Bernhard Steffen. "Active learning for extended finite state machines". In: *Formal Asp. Comput.* 28 (2016), pp. 233–263. DOI: 10.1007/s00165-016-0355-5.

[CDH08] Krishnendu Chatterjee, Laurent Doyen and Thomas A. Henzinger. "Quantitative Languages". In: *CSL*. 2008, pp. 385–400. DOI: 10.1007/978-3-540-87531-4_28.

[CGP99] Edmund M. Clarke Jr., Orna Grumberg and Doron A. Peled. *Model Checking*. MIT press, 1999.

[Cha+14] Georg Chalupar, Stefan Peherstorfer, Erik Poll and Joeri de Ruiter. "Automated Reverse Engineering using Lego®". In: *WOOT*. USENIX Association, 2014. URL: https://www.usenix.org/conference/woot14/workshop-program/presentation/chalupar.

[Cha+15] Martin Chapman, Hana Chockler, Pascal Kesseli, Daniel Kroening, Ofer Strichman and Michael Tautschnig. "Learning the Language of Error". In: *ATVA*. Vol. 9364. LNCS. Springer, 2015, pp. 114–130. DOI: 10.1007/978-3-319-24953-7_9.

[Cho+10] Chia Yuan Cho, Domagoj Babić, Eui Chul Richard Shin and Dawn Song. "Inference and Analysis of Formal Models of Botnet Command and Control Protocols". In: *CCS*. ACM, 2010, pp. 426–439. DOI: 10.1145/1866307.1866355.

[Cho78] Tsun S. Chow. "Testing Software Design Modeled by Finite-State Machines". In: *IEEE Trans. Software Eng.* 4 (1978), pp. 178–187. DOI: 10.1109/TSE.1978.231496.

[CK93] Karel Culik II and Jarkko Kari. “Image compression using weighted finite automata”. In: *Comput. Graph.* 17 (1993), pp. 305–313. DOI: 10.1016/0097-8493(93)90079-O.

[CNP93] Hugues Calbrix, Maurice Nivat and Andreas Podelski. “Ultimately periodic words of rational ω-languages”. In: *MFPS*. Vol. 802. LNCS. Springer, 1993, pp. 554–566. DOI: 10.1007/3-540-58027-1_27.

[CP71] Jack W. Carlyle and Azaria Paz. “Realizations by Stochastic Finite Automata”. In: *J. Comput. Syst. Sci.* 5 (1971), pp. 26–40. DOI: 10.1016/S0022-0000(71)80005-3.

[CUV06] Venanzio Capretta, Tarmo Uustalu and Varmo Vene. “Recursive coalgebras from comonads”. In: *Inf. Comput.* 204 (2006), pp. 437–468. DOI: 10.1016/j.ic.2005.08.005.

[CV12] Vincenzo Ciancia and Yde Venema. “Stream automata are coalgebras”. In: *CMCS*. Vol. 7399. LNCS. Springer, 2012, pp. 90–108. DOI: 10.1007/978-3-642-32784-1_6.

[DG05] Manfred Droste and Paul Gastin. “Weighted Automata and Weighted Logics”. In: *ICALP*. 2005, pp. 513–525. DOI: 10.1007/11523468_42.

[DH03] Frank Drewes and Johanna Högberg. “Learning a Regular Tree Language from a Teacher”. In: *DLT*. Vol. 2710. LNCS. 2003, pp. 279–291. DOI: 10.1007/3-540-45007-6_22.

[DH07] Frank Drewes and Johanna Högberg. “Query learning of regular tree languages: How to avoid dead states”. In: *Theory of Comput. Syst.* 40 (2007), pp. 163–185. DOI: 10.1007/s00224-005-1233-3.

[DLT02] François Denis, Aurélien Lemay and Alain Terlutte. “Residual finite state automata”. In: *Fundam. Inform.* 51 (2002), pp. 339–368. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi51-4-02.

[Dor+17] Ulrich Dorsch, Stefan Milius, Lutz Schröder and Thorsten Wißmann. “Efficient Coalgebraic Partition Refinement”. In: *CONCUR*. Vol. 85. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 32:1–32:16. DOI: 10.4230/LIPIcs.CONCUR.2017.32.

[FJV16] Paul Fiterău-Broştean, Ramon Janssen and Frits Vaandrager. "Combining model learning and model checking to analyze TCP implementations". In: *CAV*. Vol. 9780. LNCS. Springer. 2016, pp. 454–471. DOI: 10.1007/978-3-319-41540-6_25.

[Fli74] Michel Fliess. "Matrices de Hankel". In: *J. Math. Pures Appl* 53 (1974), pp. 197–222.

[Gab01] Murdoch J. Gabbay. "A theory of inductive definitions with α-equivalence: semantics, implementation, programming language". PhD thesis. University of Cambridge, 2001.

[GH08] Murdoch James Gabbay and Martin Hofmann. "Nominal renaming sets". In: *LPAR*. Vol. 5330. LNCS. Springer, 2008, pp. 158–173. DOI: 10.1007/978-3-540-89439-1_11.

[GMS14] Sergey Goncharov, Stefan Milius and Alexandra Silva. "Towards a coalgebraic Chomsky hierarchy". In: *TCS*. Springer. 2014, pp. 265–280. DOI: 10.1007/978-3-662-44602-7_21.

[Gog72a] Joseph A. Goguen. "Minimal realization of machines in closed categories". In: *Bull. Amer. Math. Soc.* 78 (1972), pp. 777–783. DOI: 10.1090/S0002-9904-1972-13032-5.

[Gog72b] Joseph A. Goguen. "Realization is universal". In: *Math. Syst. Theory* 6 (1972), pp. 359–374. DOI: 10.1007/BF01843493.

[Gol67] E Mark Gold. "Language identification in the limit". In: *Inf. Control.* 10 (1967), pp. 447–474. DOI: 10.1016/S0019-9958(67)91165-5.

[Gol72] E. Mark Gold. "System identification via state characterization". In: *Autom.* 8 (1972), pp. 621–636. DOI: 10.1016/0005-1098(72)90033-7.

[Hee+18a] Gerco van Heerdt, Justin Hsu, Joël Ouaknine and Alexandra Silva. "Convex language semantics for nondeterministic probabilistic automata". In: *ICTAC*. Vol. 11187. LNCS. Springer. 2018, pp. 472–492. DOI: 10.1007/978-3-030-02508-3_25.

[Hee+18b] Gerco van Heerdt, Bart Jacobs, Tobias Kappé and Alexandra Silva. "Learning to Coordinate". In: *It's All About Coordination*. LNCS. Springer, 2018, pp. 139–159. DOI: 10.1007/978-3-319-90089-6_10.

[Hee+19a] Gerco van Heerdt, Joshua Moerman, Matteo Sammartino and Alexandra Silva. "A (co)algebraic theory of succinct automata". In: *JLAMP* 105 (2019), pp. 112–125. DOI: 10.1016/j.jlamp.2019.02.008.

[Hee+19b] Gerco van Heerdt, Tobias Kappé, Jurriaan Rot, Matteo Sammartino and Alexandra Silva. "Tree automata as algebras: Minimisation and determinisation". In: *CALCO*. Vol. 139. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 6:1–6:22. DOI: 10.4230/LIPIcs.CALCO.2019.6.

[Hee+20a] Gerco van Heerdt, Tobias Kappé, Jurriaan Rot, Matteo Sammartino and Alexandra Silva. "A Categorical Framework for Learning Generalised Tree Automata". In: *CoRR* (2020). arXiv: 2001.05786. URL: https://arxiv.org/abs/2001.05786.

[Hee+20b] Gerco van Heerdt, Clemens Kupke, Jurriaan Rot and Alexandra Silva. "Learning Weighted Automata over Principal Ideal Domains". In: *FoSSaCS*. LNCS. Springer. 2020, pp. 602–621. DOI: 10.1007/978-3-030-45231-5_31.

[Hee16] Gerco van Heerdt. "An Abstract Automata Learning Framework". MA thesis. Radboud University Nijmegen, 2016. URL: https://www.ru.nl/publish/pages/769526/gerco_van_heerdt.pdf.

[HMM09] Johanna Högberg, Andreas Maletti and Jonathan May. "Backward and forward bisimulation minimization of tree automata". In: *Theor. Comput. Sci.* 410 (2009), pp. 3539–3552. DOI: 10.1016/j.tcs.2009.03.022.

[Hol82] William M. Holcombe. *Algebraic Automata Theory*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1982. DOI: 10.1017/CBO9780511525889.

[How+12] Falk Howar, Bernhard Steffen, Bengt Jonsson and Sofia Cassel. "Inferring canonical register automata". In: *VMCAI*. Vol. 7148. LNCS. Springer, 2012, pp. 251–266. DOI: 10.1007/978-3-642-27940-9_17.

[HSS17a] Gerco van Heerdt, Matteo Sammartino and Alexandra Silva. "CALF: Categorical Automata Learning Framework". In: *CSL*. Vol. 82. LIPIcs. 2017, 29:1–29:24. DOI: 10.4230/LIPIcs.CSL.2017.29.

[HSS17b] Gerco van Heerdt, Matteo Sammartino and Alexandra Silva. "Optimizing automata learning via monads". In: *CoRR* (2017). arXiv: 1704.08055. URL: http://arxiv.org/abs/1704.08055.

[HSS20] Gerco van Heerdt, Matteo Sammartino and Alexandra Silva. "Learning Automata with Side-Effects". In: *CMCS*. 2020, to appear.

[IHS14] Malte Isberner, Falk Howar and Bernhard Steffen. "The TTT algorithm: A redundancy-free approach to active automata learning". In: *RV*. Vol. 8734. LNCS. Springer, 2014, pp. 307–322. DOI: 10.1007/978-3-319-11164-3_26.

[IHS15] Malte Isberner, Falk Howar and Bernhard Steffen. "The Open-Source LearnLib: A Framework for Active Automata Learning". In: *CAV*. Vol. 9206. LNCS. Springer, 2015, pp. 487–495. DOI: 10.1007/978-3-319-21690-4_32.

[Isb15] Malte Isberner. "Foundations of active automata learning: an algorithmic perspective". PhD thesis. Technical University of Dortmund, 2015. URL: http://hdl.handle.net/2003/34282.

[Jac06] Bart Jacobs. "A bialgebraic review of deterministic automata, regular expressions and languages". In: *Algebra, Meaning, and Computation*. Vol. 4060. LNCS. Springer, 2006, pp. 375–404. DOI: 10.1007/11780274_20.

[Jac12] Nathan Jacobson. *Basic algebra I*. Courier Corporation, 2012.

[Jac17] Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*. Vol. 59. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2017. DOI: 10.1017/CBO9781316823187.

[Jac53] Nathan Jacobson. *Lectures in Abstract Algebra*. Vol. 31. GTM. Springer, 1953.

[JS14] Bart Jacobs and Alexandra Silva. "Automata Learning: A Categorical Perspective". In: *Horizons of the Mind. A Tribute to Prakash Panangaden*. Vol. 8464. LNCS. 2014, pp. 384–406. DOI: 10.1007/978-3-319-06880-0_20.

[Kap+19] Tobias Kappé, Paul Brunet, Bas Luttik, Alexandra Silva and Fabio Zanasi. "On series-parallel pomset languages: Rationality, context-freeness and automata". In: *JLAMP* 103 (2019), pp. 130–153. DOI: 10.1016/j.jlamp.2018.12.001.

[Kel80] G.M. Kelly. "A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves, and so on". In: *Bull. Austral. Math. Soc.* 22 (1980), pp. 1–83. DOI: 10.1017/S0004972700006353.

[KI15] Oleg Kiselyov and Hiromi Ishii. "Freer monads, more extensible effects". In: *Haskell*. Vol. 50. ACM, 2015, pp. 94–105. DOI: 10.1145/2804302.2804319.

[KK14] Barbara König and Sebastian Küpper. "Generic Partition Refinement Algorithms for Coalgebras and an Instantiation to Weighted Automata". In: *TCS*. Vol. 8705. Lecture Notes in Computer Science. Springer, 2014, pp. 311–325. DOI: 10.1007/978-3-662-44602-7_24.

[Koz12] Dexter C. Kozen. *Automata and computability*. Springer Science & Business Media, 2012. DOI: 10.1007/978-1-4612-1844-9.

[KR16] Bartek Klin and Jurriaan Rot. "Coalgebraic trace semantics via forgetful logics". In: *LMCS* 12 (2016). DOI: 10.2168/LMCS-12(4:10)2016.

[Kro94] Daniel Krob. "The equality problem for rational series with multiplicities in the tropical semiring is undecidable". In: *IJAC* 4 (1994), pp. 405–425. DOI: 10.1142/S0218196794000063.

[KS18] Bartek Klin and Julian Salamanca. "Iterated Covariant Powerset is not a Monad". In: *MFPS*. Vol. 341. Electron. Notes Theor. Comput. Sci. Elsevier, 2018, pp. 261–276. DOI: 10.1016/j.entcs.2018.11.013.

[Kup14] Denis Kuperberg. "Linear Temporal Logic for Regular Cost Functions". In: *LMCS* 10 (2014). DOI: 10.2168/LMCS-10(1:4)2014.

[KV94] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT press, 1994. URL: https://mitpress.mit.edu/books/introduction-computational-learning-theory.

[Lan13] Saunders Mac Lane. *Categories for the working mathematician*. Vol. 5. Springer, 2013. DOI: 10.1007/978-1-4757-4721-8.

[LG02] Christoph Lüth and Neil Ghani. "Composing monads using coproducts". In: *ICFP*. ACM, 2002, pp. 133–144. DOI: 10.1145/581478.581492.

[LR11] Stephen Lack and Jiří Rosický. "Notions of Lawvere theory". In: *Appl. Categ. Struct.* 19 (2011), pp. 363–391. DOI: 10.1007/s10485-009-9215-2.

[LW00] Kamal Lodaya and Pascal Weil. "Series-parallel languages and the bounded-width property". In: *Theor. Comput. Sci.* 237 (2000), pp. 347–380. DOI: 10.1016/S0304-3975(00)00031-1.

[Man76] Ernest G Manes. *Algebraic theories*. Vol. 26. Graduate Texts in Mathematics. Springer, 1976. DOI: 10.1007/978-1-4612-9860-1.

[Mar79] George Markowsky. "Free completely distributive lattices". In: *Proc. Amer. Math. Soc.* 74 (1979), pp. 227–228. DOI: 10.1090/S0002-9939-1979-0524290-9.

[Moe+17] Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin and Michał Szynwelski. "Learning nominal automata". In: *POPL*. ACM, 2017, pp. 613–625. DOI: 10.1145/3009837.3009879.

[Moo56] Edward F. Moore. "Gedanken-experiments on sequential machines". In: *J. Symb. Log.* Annals of Mathematical Studies 23 (1956), pp. 129–153. DOI: 10.2307/2964500.

[MP95] Oded Maler and Amir Pnueli. "On the Learnability of Infinitary Regular Sets". In: *Inf. Comput.* 118 (1995), pp. 316–326. DOI: 10.1006/inco.1995.1070.

[MPR05] Mehryar Mohri, Fernando Pereira and Michael Riley. "Weighted Automata in Text and Speech Processing". In: *CoRR* (2005). arXiv: cs/0503077. URL: https://arxiv.org/abs/cs/0503077.

[MR20] Joshua Moerman and Jurriaan Rot. "Separation and Renaming in Nominal Sets". In: *CSL*. Vol. 152. LIPIcs. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020, 31:1–31:17. DOI: 10.4230/LIPIcs.CSL.2020.31.

[Orz71] Morris Orzech. "Onto endomorphisms are isomorphisms". In: *Amer. Math. Monthly* 78 (1971), pp. 357–362. DOI: 10.2307/2316897.

[Osi74] Gerhard Osius. "Categorical set theory: a characterization of the category of sets". In: *J. Pure and Appl. Algebra* 4 (1974), pp. 79–119. DOI: 10.1016/0022-4049(74)90032-2.

[Pit13] Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Vol. 57. Cambridge University Press, 2013. URL: http://www.gabbay.org.uk/papers/thesis.pdf.

[PO98] Jorge M. Pena and Arlindo L. Oliveira. "A New Algorithm For The Reduction Of Incompletely Specified Finite State Machines". In: *ICCAD*. ACM / IEEE Computer Society, 1998, pp. 482–489. DOI: 10.1145/288548.289075.

[PS12] Damien Pous and Davide Sangiorgi. "Enhancements of the bisimulation proof method". In: *Advanced Topics in Bisimulation and Coinduction*. Vol. 52. Cambridge tracts in theoretical computer science. Cambridge University Press, 2012, pp. 233–289.

[Rot15] Jurriaan Rot. "Enhanced coinduction". PhD thesis. Leiden University, 2015. URL: https://openaccess.leidenuniv.nl/handle/1887/35814.

[RP15] Joeri de Ruiter and Erik Poll. "Protocol state fuzzing of TLS implementations". In: *USENIX Security*. USENIX Association, 2015, pp. 193–206. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/de-ruiter.

[RS93] Ronald L. Rivest and Robert E. Schapire. "Inference of Finite Automata Using Homing Sequences". In: *Inf. Comput.* 103 (1993), pp. 299–347. DOI: 10.1006/inco.1993.1021.

[Rut00] Jan J. M. M. Rutten. "Universal coalgebra: a theory of systems". In: *Theor. Comput. Sci.* 249 (2000), pp. 3–80. DOI: 10.1016/S0304-3975(00)00056-6.

[Rut19] Jan Rutten. "The Method of Coalgebra: exercises in coinduction". In: (2019). URL: https://ir.cwi.nl/pub/28550/.

[Rut98] Jan J. M. M. Rutten. "Automata and Coinduction (An Exercise in Coalgebra)". In: *CONCUR*. Vol. 1466. LNCS. Springer, 1998, pp. 194–218. DOI: 10.1007/BFb0055624.

[Sak90] Yasubumi Sakakibara. "Learning context-free grammars from structural data in polynomial time". In: *Theor. Comput. Sci.* 76 (1990), pp. 223–242. DOI: 10.1016/0304-3975(90)90017-C.

[San98] Davide Sangiorgi. "On the bisimulation proof method". In: *Mathematical Structures in Computer Science* 8 (1998), pp. 447–479. DOI: 10.1017/S0960129598002527.

[SHV16] Mathijs Schuts, Jozef Hooman and Frits Vaandrager. "Refactoring of legacy software using model learning and equivalence checking: an industrial experience report". In: *IFM*. Vol. 9681. LNCS. 2016, pp. 311–325. DOI: 10.1007/978-3-319-33693-0_20.

[Sil+13] Alexandra Silva, Filippo Bonchi, Marcello M. Bonsangue and Jan J. M. M. Rutten. "Generalizing determinization from automata to coalgebras". In: *LMCS* 9 (2013). DOI: 10.2168/LMCS-9(1:9)2013.

[Smi61] Henry J. Stephen Smith. "On Systems of Linear Indeterminate Equations and Congruences". In: *Philosophical Transactions of the Royal Society of London* 151 (1861), pp. 293–326. URL: http://www.jstor.org/stable/108738.

[Tap+19] Martin Tappler, Bernhard K. Aichernig, Giovanni Bacci, Maria Eichlseder and Kim G. Larsen. "L^*-Based Learning of Markov Decision Processes". In: *FM*. 2019, pp. 651–669. DOI: 10.1007/978-3-030-30942-8_38.

[Tay99] Paul Taylor. *Practical foundations of mathematics*. Vol. 59. Cambridge studies in advanced mathematics. Cambridge University Press, 1999.

[TV05] Deian Tabakov and Moshe Y Vardi. "Experimental evaluation of classical automata constructions". In: *LPAR*. Vol. 5. LNCS. Springer, 2005, pp. 396–411. DOI: 10.1007/11591191_28.

[US19] Henning Urbat and Lutz Schröder. "Automata Learning: An Algebraic Approach". In: *CoRR* (2019). arXiv: 1911.00874. URL: https://arxiv.org/abs/1911.00874.

[Vaa17] Frits W. Vaandrager. "Model learning". In: *Commun. ACM* 60 (2017), pp. 86–95. DOI: 10.1145/2967606.

[Val84] Leslie G. Valiant. "A theory of the learnable". In: *Commun. ACM* 27 (1984), pp. 1134–1142. DOI: 10.1145/1968.1972.

[Vas73] M.P. Vasilevskii. "Failure diagnosis of automata". In: *Cybern. Syst. Anal.* 9 (1973), pp. 653–665. DOI: 10.1007/BF01068590.

[Vil96] Juan Miguel Vilar. "Query Learning of Subsequential Transducers". In: *ICGI*. Vol. 1147. LNCS. Springer, 1996, pp. 72–83. DOI: 10.1007/BFb0033343.

[Wiß+19] Thorsten Wißmann, Stefan Milius, Shin-ya Katsumata and Jérémy Dubut. "A Coalgebraic View on Reachability". In: *CoRR* (2019). arXiv: 1901.10717. URL: https://arxiv.org/abs/1901.10717.

Acknowledgements

First of all I would like to thank my supervisors, Alexandra, Jurriaan, and Matteo, for their guidance and inspiration that led to the present thesis. Apart from providing support throughout my PhD you encouraged me to participate in many schools, conferences, and workshops, as well as to enjoy life in London. You have shown that a great supervisor is also a great friend. Thank you very much.

Thanks to the reviewers of my submissions for all their valuable comments, and to the examiners Bartek and Nikos for taking the time to study my work and for providing insightful remarks and useful corrections that helped to finalise it. My coauthors Bart, Clemens, Joël, Joshua, Justin, and Tobias deserve much credit for all the fruitful discussions and hard work. A big thank you to Clemens for inviting me to the awesome city of Glasgow for a research visit. Hopefully we will be able to repeat this and, most importantly, visit Balloch in the future!

Over the past four years I had the pleasure of working in a friendly and stimulating research environment at UCL. I am grateful for having been able to share the PhD journey with Louis and Tobias, who started and finished around the same time as I did. Thanks to Bas, Benjamin, Cristoph, Diana, Fabio, Fred, Maria, the two Pauls, Robin, Simon, Sonia, Stefan, Thomas, and really everyone else in and around PPLV for all the interesting discussions and fun moments in pubs and restaurants and on other adventures. Many thanks also to our visitors, including Jana, Joshua, Justin, Steffen, and Tiago, for the great times spent together.

I am deeply grateful to the organisers of the conferences and schools I participated in: ProbProgSchool 2017, CSL 2017, the Jerusalem Winter School in Computer Science and Engineering, the FoPPS Summer School 2018, ICTAC 2018, CALCO 2019, and SPLV 2019. This list excludes many smaller events that I was fortunate to be able to attend. I am particularly happy to have been at the Bellairs workshop on Learning and Verification in 2019. Many thanks also to all participants for making these events a success and creating great social environments.

My move to London was also the first time I moved away from my family. I was lucky to

spend my first two years at Chester House, which provided a very friendly and welcoming environment with many conveniences that eased the transition. Many thanks to all the friends I made there. I enjoyed your company at dinner, our regular film and boardgame nights, and the occasional night out. When I moved to a flat I was lucky to have three great flatmates: Kate, Rorie, and Tobias. Thank you for often being around for a chat, for the outings to the pub and other events, for sorting out the bills, and most of all for teaching me how to cook.

On my adventures in London I met many wonderful people, whom I would like to thank for sharing their time with me. A special mention goes to Sarah, who introduced me to numerous cool places in the city. I also want to thank Matthias, who has stayed in touch with me since high school and even came to visit me.

Thanks to my parents Frans and Marieke and my brother Marcel for supporting me throughout the years and for hosting me during holidays and in particular during the months leading up to the thesis submission deadline. Thanks to Cynthia and Hugh for hosting me during the period leading up to my viva, and for helping Tsiu-Kim organise a wonderful viva party.

Finally, my many thanks go out to Tsiu-Kim. Thank you for your love and support, for keeping me entertained during lockdown, and for all the fun things we do together.